UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Neural Network Supervision: Notes on Loss Functions, Labels and Confidence Estimation

Gil Keren

Dissertation eingereicht an der Fakultät für Informatik und Mathematik der Universität Passau zur Erlangung des Grades eines Doktors der Naturwissenschaften

A dissertation submitted to the faculty of computer science and mathematics in partial fulfillment of the requirements for the degree of doctor of natural sciences

Advisor: Prof. Dr. habil. Björn Schuller

Passau, May 2019

Acknowledgements

For supporting my research process and providing valuable advice, I would like to thank my doctoral advisor Prof. Dr. habil. Björn Schuller, as well as my other colleagues and collaborators who took part in the work leading to this dissertation: Sivan Sabato, Thomas Kehrenberg, Maximilian Schmitt and Nicholas Cummins.

Abstract

We consider a number of enhancements to the standard neural network training paradigm. First, we show that carefully designed parameter update rules may replace the need for a loss function and its gradient. We introduce a parameter update rule that generalises the standard cross-entropy gradient, and allows directly controlling the relative effect of easy and hard examples on the training process. We show that the proposed update rule cannot be derived by using a loss function and yields better classification accuracy compared to training with the standard cross-entropy loss.

In addition, we study the effect of the loss function choice on the learnt representations. We introduce the Single Logit Classification (SLC) task: classifying whether a given class is the correct class for a given example, in a computationally efficient manner, based on the appropriate class logit alone. A natural principle is proposed, the Principle of Logit Separation (PoLS), as a guideline for choosing and designing loss functions suitable for the SLC task. We mathematically analyse the alignment of eleven existing and novel loss functions with this principle. Experiment results show that using loss functions that are aligned with this principle results in a representation in the logits layer in which each logit is more informative of its class correctness, leading to a considerably better SLC accuracy.

Further, we attempt to alleviate the dependency of standard neural network models on large amounts of quality labels. The task of weakly supervised one-shot detection is considered, in which at training time the model is trained without any localisation labels, and at test time it needs to identify and localise instances of unseen classes. We propose the attention similarity networks (ASN) for this task. ASN use a Siamese neural network to compute a similarity score between an exemplar and different locations in a target example. Then, an attention mechanism performs localisation by learning to attend to the correct locations. The ASN model outperforms the relevant baselines for weakly supervised one-shot detection tasks in the audio and computer vision domains.

Finally, we consider the problem of quantifying prediction confidence in the regression setting. We propose two novel algorithms for emitting calibrated prediction intervals for neural network regressors, at any given confidence level. The two algorithms require binning of the output space and training the neural network regressor as a classifier. Then, the calibration algorithms choose the intervals in the output space, making sure they contain the amount of posterior probability mass that results in the desired confidence level.

Table of contents

Chapter 1

Introduction

In recent years, neural network models have demonstrated large empirical success across multiple domains such as object recognition [42, 65], object detection and segmentation [41, 95, 96], image captioning [128], machine translation [5, 117] and speech recognition [13]. This cross-domain empirical success has established neural networks as the model of choice for many large-scale industrial applications.

Success in the above listed applications is obtained using methods that have much in common. In all cases, a model is designed to emit the appropriate predictions for the task at hand. These predictions are then compared to ground truth labels using a loss function. Learning is done by minimising the value of the loss function on a large labelled training sample. This standard supervised learning method is the dominant approach for training neural network models, and indeed yields high task accuracy across a variety of domains. However, the standard supervised learning paradigm has a number of potential limitations. In this work we discuss four of those potential limitations, and propose remedies for some cases, as discussed in detail below.

1.1 Training without a loss function

A loss function $L(p,y)$ generally compares the network predictions p with the labels y, to create a scalar value that indicates some measure of distance between them. During learning, learnable model parameters θ are changed such that the value of L is minimised over a labelled training sample. To minimise the value of L, the most common learning algorithms for neural networks are gradient-based learning algorithms, such as Stochastic Gradient Descent (SGD), Momentum Polyak [93], RMSProp Tieleman and Hinton [114], AdaGrad Duchi et al. [27], AdaDelta Zeiler [132] and Adam Kingma and Ba [62].

Gradient-based learning algorithms use a simple optimisation principle: changing the parameters by a small step in direction of the negative gradient $-\nabla_\theta L(p,y)$ will lead to a reduction in the value of the loss function L. In SGD, the gradient is scaled and directly used as the parameter update, while in other algorithms, such as Momentum, RMSProp, AdaGrad, AdaDelta and Adam, the gradient is processed through some transformation, normally a form of averaging and scaling, to produce the parameter updates. In all cases, the parameter updates in gradient-based algorithms originate in the gradient of some loss function. While this approach can only guarantee the convergence of the loss function to a local minimum under some conditions, with no guarantees regarding a global minimum, these algorithms generally work well in practice and are used for obtaining state-of-the-art results in all of the application domains mentioned above.

While this approach of guiding the learning process using the gradient of a loss function is natural and sensible, it limits the set of possible parameter update rules to those that originate in the gradient of some loss function. Alternatively, one could replace the gradient in the standard training procedure by any function $g(p,y,\theta)$, that denotes the parameter updates given the model predictions, labels and the current values of the learnable parameters. Using $g(p,y,\theta) = -\nabla_\theta L(p,y)$ is a special case that amounts to standard gradient-based learning. The function g could be referred to as a *pseudo-gradient*. All the above mentioned gradient-based learning algorithms could be used with the pseudo-gradient in place of the gradient, without any additional changes necessary. Any such pseudo-gradient g will result in a different update rule for the parameters. There could be many sensible and useful update rules, each originating from a different choice of the function g. As many functions cannot be written as a gradient of some function, a priori it is natural to assume that there may exist some sensible and useful update rules that do not originate from the gradient of any loss function. Therefore, the first research question of this work is formulated as follows:

> **Question 1: Are there cases in which update rules that do not originate from a gradient of any loss function are preferable over standard gradient-based update rules?**

We address this question in Chapter 2, covering the work published in Keren et al. [56]. In the setting considered in this chapter, we attempt to tune the relative importance of easy and hard training examples on the training procedure. We present an update rule for the parameters that is a generalisation of the cross-entropy gradient, and directly controls the relative effect of easy and hard examples on the parameter updates. The new update rule allows tuning for the optimal sensitivity for hard examples. A proof is given, that the proposed update rule does not originate from the gradient of any loss function. Experiments with both a toy example and common benchmark datasets show the effectiveness of our proposed

update rule, obtaining classification accuracy gains by tuning for the optimal sensitivity for hard training examples. In addition, we find that the level of optimal sensitivity depends on the depth of the neural network used.

1.2 Differences between loss functions

The previous section and the research question it poses consider the option of learning neural network parameters without using a loss function and its gradient. In the chapter that considers this research question, we show that this is a viable alternative to the standard training paradigm in the setting we identify and potentially many other settings. However, the use of loss functions and their gradient is still the most common approach, and may be preferable in many learning settings.

As discussed above, a loss function normally compares the model predictions with the known labels, to create some measure of dissimilarity between them. For a given task, there is usually a variety of possible loss functions, all computing sensible dissimilarity measures between the model predictions and the labels. In the presence of an abundance of loss functions, all designed to suit the task at hand, it may be unclear to a machine learning researcher or practitioner what the subtle differences between the possible loss functions are, and which loss function is the most suitable for their needs.

Consider the task of multiclass classification. An abundance of loss functions were proposed for this task, including the most common cross-entropy loss [44], the Mean Squared Error (MSE) loss, the binary cross-entropy loss and the max-margin loss [18]. Glorot and Bengio [32] study the loss surfaces of the cross-entropy and the MSE losses, and find that the plateaus in the loss surface are less present with the cross-entropy loss, making it more suitable for gradient-based optimisation. Other works have found that minimising the cross-entropy loss results in faster convergence, compared to the max-margin loss and the MSE losses [47, 106]. In addition, Janocha and Czarnecki [47] found that in the context of image classification, the cross-entropy loss is more robust to training with noisy examples, compared to the max-margin and the MSE losses.

While most existing works have focused on the difference between classification losses in terms of noise robustness and optimisation properties, such as convergence speed and the loss surface topology, other differences between classification loss functions may exist. In this work, we are interested in the differences in the learnt representations that result from using each loss function. A better understanding of the representations resulting from each loss function may be useful, as those representations are often used in downstream tasks. For example, in transfer learning it is a common practice to use a pretrained network on

a set of source classes to extract a representation of the input in some intermediate layer, and use this representation to classify the example to a set of target classes [25, 130]. Other works use the representation at the topmost layer of the neural network as inputs to a Support Vector Machine (SVM) classifier [94, 112]. The above discussion leads to the next research question that is considered in this work:

> **Question 2: How does the loss function used affect the learnt representations in a neural network, and could those differences assist with the choice of a loss function to use?**

In chapter 3 of this work we attempt to investigate the above research question. The chapter covers the work published in Keren et al. [57] and Keren et al. [58] (the latter was announced accepted, subject to a minor revision, in May 2019). We study the learnt representations in the top layer of a neural network classifier, that are called the class logits, that result from using a variety of loss functions. We do this in the context of the *Single Logit Classification* (SLC) task, that is the binary classification of a single logit to determine whether it corresponds to the correct class for the given example. We first show that applying SLC instead of computing all class logits may be useful in settings of classification over a large number of classes, as in these cases computing all classes' logits results in high computational burden at test time. We hypothesise that the loss function used shapes the representation in the logits layer in a manner that affects performance in the SLC task. We propose a simple principle named *the Principle of Logit Separation*, as a possible guideline for choosing the right loss function for success in SLC. We analyse the alignment of eleven different classification loss functions with our proposed principle. In experiments, we show that the Principle of Logit Separation is indeed a main ingredient for high SLC accuracy, as loss functions that are aligned with this principle produce representations in the logits layer that are much more informative for class correctness, and result in a considerable increase in SLC accuracy.

1.3 Low-quality supervision

The above discussion points out the considerations need to be made for choosing an appropriate loss function, and an alternative training mechanism that does not use a loss function. In both cases, the underlying assumption is that there exists a large enough labelled training set, allowing some training mechanisms to compare the model predictions with the labels, in order to emit the parameter updates. However, this assumption does not hold in many applications, in which available labels are not sufficient, or their quality does not allow directly comparing them to model predictions.

For example, existing labels could be noisy, i.e., for some portion of the examples, the available labels are wrong and do not represent the correct target values. In some other situations, for a classification task, labels may exists for some set of classes, but for other classes that appear at test time, there are no labels available. Another example is when the detail level of the existing labels is not sufficient, and they contain only partial information, which is less detailed than the prediction the model needs to emit. We refer to such settings as *low-quality supervision* settings. In those settings, existing labels lack some of their required properties, therefore are partial and of a low quality in some sense. We distinguish this setting from the unsupervised or semi-supervised learning settings, that normally refer to settings in which labels do not exists, or exist with a sufficient quality but in a low quantity.

Low-quality supervision settings naturally appear, as the process of collection high-quality labels is often time-consuming and expensive. Indeed, automatic or semi-automatic label collection procedures [2] may be fast and cheap, but the resulting labels may contain a certain level of noise. In other settings, as label collection procedures that involve human effort are in general expensive, collected datasets may include labels for a limited set of classes, or with a limited detail level. For example, the Open Images V4 dataset [67] contains about 79M machine generated training labels for approximately 8K classes for image-level object classification, but less than 15M human created bounding box training labels for 600 classes for an object detection task. While 15M is still considered a large number of available labels, it is more than five times smaller than the number of available image-level labels. Similarly, while 600 classes is considered a large number of classes for an object detection task, the number of classes for the lower-quality labels is much larger.

We consider the detection task. In the computer vision domain this is normally the task of object detection [75, 96], in which the model needs to emit the class that appears in a given image and the location in which it appears. In the speech domain, the query-by-example spoken term detection [14, 40, 89, 126, 127] is a detection task, in which the model has to determine whether a given word appears in a given sentence recording, and to localise this appearance in time. In this task, the different words are treated as the different classes. For a detection task, a high-quality labelled dataset would need to contain both instance-level labels that indicate which class appears in training instances, and bounding box localisation labels that indicate the location in the instance of the appearing class. As discussed above, existing datasets that contain only instance-level labels may be considerably larger than datasets that contain both instance-level and localisation labels, both in number of classes and existing labels count. This motivates the attempt to perform the detection task in the low-quality supervision setting, by only utilising the larger datasets that do not contain any localisation labels. This leads to the next research question we consider:

Question 3: Can neural network models utilise large datasets of instance-level labels, to perform a detection task for a large number of classes?

We attempt to answer the above question by proposing an appropriate candidate model in Chapter 4. The chapter covers the work appearing in the preprint Keren et al. [59]. The proposed model is designed for the weakly supervised one-shot detection, for both the computer vision and the audio domains. In this task, a model emits a detection output by utilising only instance-level labels at training time. In addition, at test time the instances for detection belong to classes that were not present at training time. The model is designed as follows. As inputs, both an exemplar of some class and a larger example are given. The larger example may or may not contain an instance of the same class as the exemplar. The model embeds the exemplar and every region of the larger examples, and computes the similarity score between the exemplar and the regions' representations. A novel attention mechanism is used to weight all the regions' similarity scores and combine them into a single instance-level prediction, whether the larger example contains an instance of the same class as the exemplar. For emitting localisation information, the attention weights are used.

This model performs the detection task in the context of low-quality supervision, utilising large datasets that contain partial information. The challenge of not having localisation information at training time is avoided by using an attention mechanism. Moreover, we go beyond the classes appearing at training time and perform one-shot detection. by designing the model to not be conditioned on class identities, the model can indeed successfully operate on instances of unseen classes. Experiments with data from both the computer vision and the audio domains show that the proposed model manages to simultaneously identify and localise instances of classes unseen at training time, and outperform the baseline models for this task.

1.4 Calibrated prediction intervals

Most focus in the research community and specifically in this work is given to improving model accuracy for a variety of tasks. However, another important success measure for machine learning models is their ability to produce reliable prediction confidence estimates. Consider for example medical applications, in which the meaning of a wrong decision can be disastrous. In such applications, one would want to distinguish between confident and unconfident positive decisions made by a machine learning model. Moreover, reliable prediction confidence estimates may facilitate human trust in machine learning models, aiding in deploying them as decision support systems.

One of the main challenges in emitting reliable confidence estimates for machine learning models is that this feature cannot simply be learnt using the same training set that was used for training the model to emit accurate predictions. The problem is that neural networks are in general powerful function approximators [19], therefore in many cases they manage to predict the correct label on the entire training set. In that case, when using the same training set to learn to emit prediction confidence, the model will learn to always predict perfect confidence. To illustrate this difficulty, assume a binary classification model that emits a prediction $0 \leq p \leq 1$ and a prediction confidence estimate c. The classification labels are denoted as $y \in \{0,1\}$, and $f(p,y)$ is the *correctness label*, i.e., whether the model predicted correctly: $f(p,y) = 1$ if $y = 1$ and $p \geq 0.5$ or if $y = 0$ and $p < 0.5$, and $f(p,y) = 0$ otherwise. To train such model to emit correct predictions and correct prediction confidence estimates, a loss function of the following form may be used: $L(p,c,y) = L_1(p,y) + L_2(c, f(p,y))$, where L_1 increases classification accuracy by minimising difference between predictions and labels, and L_2 increases prediction confidence accuracy by minimising the difference between the prediction confidence estimate and the correctness labels. As neural networks are powerful function approximators, it is often the case that after training the neural network emits practically perfect predictions on the training set. In that case, L_1 is minimised, meaning that for all training examples $L_1(p,y) \sim 0$ and therefore $f(p,y) = 1$. To minimise L_2, all the network has to do is to always predict $c = 1$. In conclusion, minimising L to 0 implies that the neural network will ignore the input example and will emit a perfect prediction confidence $c \sim 1$, making its prediction confidence estimates obsolete.

One way of circumventing the above mentioned issue is to use a large enough training set or a small enough neural network, such that the network does not manage to yield perfect predictions on the training set. However, very large training sets are not always available, and limiting the network size normally reduces task accuracy. Another possible way of avoiding this issue is by first using the training set to learn to produce good accuracy in the task at hand. Then, use the trained model to emit predictions on a validation set, and learn an additional model to emit prediction confidence estimates, by training it on the validation set task predictions and their correctness labels. The problem with this approach is that it requires an additional amount of data, in order for the validation set to be sufficiently large for training the second model.

The approach used in practice for classification models is to train a softmax classifier normally for high classification accuracy, then observe the posterior probabilities emitted by the model on unseen examples [36]. For a vector of class probabilities $p = (p_1, ..., p_k)$ emitted by the model, the value $c = \max_{1 \leq i \leq k} p_i$ has a reasonably high correlation with classification correctness, and is used as the prediction confidence estimate. However, as

pointed by [36], c is normally an uncalibrated confidence estimate: c is given as a probability, but is not equal to the real probability of its corresponding prediction being correct. For example, a value of $c = 0.8$ for a given example does not necessarily correspond to a 0.8 probability of this example being classified correctly. It is possible, that examples with $c \sim 0.8$ are classified correctly 90% of the times, or perhaps in another model only 50% of the times. Guo et al. [36] propose several methods for calibrating the prediction confidence estimate using the validation set, to match between values of c and actual probabilities of the examples being correctly classified.

For a regression task, prediction confidence estimates may be given in the form of prediction intervals. A prediction interval with a confidence level α is an interval, in which the label is expected to fall with a probability of α. Standard neural network regressors are designed to emit a point prediction [42, 115, 129], by having a single unit in their top layer that contains the prediction of the label. Therefore, it is unclear how to design neural network regressors that will be able to produce prediction intervals for a given confidence level. Aiming at the goal of producing reliable prediction confidence estimates for neural network regressors, our next research question is stated as follows:

Question 4: How can we design neural network regressors that are able to produce prediction intervals for any given confidence level?

This research question is addressed in Chapter 5, covering the work published in Keren et al. [53]. The problem is addressed in two stages. First, neural network regressors are designed as classifiers. This is done by binning the output space into a finite set of bins, and learning the regression task using a multiclass neural network softmax classifier. Since the softmax classifier emits a probability distribution over classes, this approach yields a probability distribution of the label over the output space. In this stage, we produce prediction intervals with a confidence level α by including a sufficient amount of bins in the output space, such that their total assigned probability is close to α. Since we are using a neural network softmax classifier, following the above discussion, the assigned probability to a given bin may not be calibrated with the actual probability of the label falling in this bin. Therefore, the resulting prediction intervals in this stage are referred to as uncalibrated prediction intervals. At the second stage, we propose two calibration algorithms to transform uncalibrated prediction intervals into calibrated ones. The proposed algorithms use the validation set to adjust the boundaries of the prediction intervals, such that a prediction interval with a confidence level of α will indeed correspond to a probability of α that the label lies within the boundaries of this interval.

Chapter 2

Tunable Sensitivity for Large Errors

2.1 Motivation

In recent years, neural network models trained using supervised learning algorithms demonstrated state-of-the-art results across a variety of domains such as computer vision [65, 110], speech recognition [13, 43], natural language processing [109] and computational paralinguistics [54, 60]. Standard implementations of those models include the application of a stochastic gradient-based algorithm, normally minimising the cross-entropy loss [44].

However, the cross-entropy loss is just a surrogate for the true task performance measure, such as classification accuracy in the case of a classification task, word-error-rates for speech recognition or the BLEU score for machine translation. When using the cross-entropy classification loss, it is shown below that the effect of a given training example on the gradient is linear in its *prediction bias*, which is the difference between the network-predicted class probabilities and the target class probabilities. In particular, a wrong confident prediction results in a larger effect on the gradient, compared to another wrong prediction that is less confident.

In contrast, humans sometimes employ a different strategy for learning. When learning a new concept, humans may ignore at first those examples that are too hard to learn, and focus on examples that are easier to understand. On the other hand, when improving proficiency regarding a familiar concept, humans may focus on those examples that are still harder to comprehend, as the latter convey more information for the advanced learner. Motivated by the above, Keren et al. [56] propose a learning algorithm that allows tuning its sensitivity to easy and hard training examples. Intuition from human cognition was previously used to inspire the design guidelines of artificial learning systems [6, 16, 69].

Keren et al. [56] show that it is possible to tune the relative effect of easy and hard examples on the training procedure by directly controlling the parameter updates during

learning. The parameter updates are computed directly from the network predictions and the labels, not mediated by the usage of a loss function and its gradient, relating to the research question posed in Section 1.1. Specifically, the proposed learning algorithm generalises the cross-entropy gradient, where the new *pseudo-gradient* can be used instead of the standard gradient in every gradient-based learning algorithm such as Momentum [93], RMSProp [114], and Adam [62]. The pseudo-gradient is parameterised by a hyperparameter $k > 0$ that controls the sensitivity of the training process to easy and hard examples, replacing the fixed linear dependence on the prediction bias of the cross-entropy loss. Using $k = 1$ results in an update rule that is identical to using the cross-entropy loss gradient.

Intuitively, the optimal sensitivity level to hard examples should be correlated with network depth. When the network is relatively shallow, its modeling capacity is limited. Therefore, it may be beneficial to ignore those hard examples that the network fails to fit, as adjusting the model based on those examples may lead to a decrease in overall prediction accuracy. On the other hand, deeper networks have relatively high modeling capacity. In this case, it may be beneficial to allow the training process higher sensitivity to hard examples, thereby possibly improving the accuracy of the final learnt model.

This chapter describes the derivation of the training algorithm for neural networks that is described in Keren et al. [56]. In addition, experiment results on several benchmark datasets from Keren et al. [56] are reported. Aligned with the motivation described above, it was found that using high sensitivity for harder examples is beneficial for relatively deep networks, and in almost all cases resulted in better classification accuracy. Similarly, using low sensitivity was found to improve classification accuracy for shallower networks. In conclusion, it is shown that tuning for the optimal sensitivity for hard examples leads in many cases to better overall performance for the classification task at hand. Furthermore, a proof given in Keren et al. [56] is included, that shows that the proposed update rule is not equivalent to using the gradient of any loss function. This points out that at least in some situations, directly controlling the parameter updates without using a loss function is beneficial, a step towards answering the first guiding research question of this work.

2.2 Related work

Choosing the best optimisation objective for neural classifiers is not a new challenge. In the past, the MSE loss was typically used with gradient-based learning in neural networks [98]. However, a line of studies found this loss function both empirically and theoretically inferior to the cross-entropy loss. These works include demonstrating that the cross-entropy loss has better learning speed [73], better performance [33] and a more suitable shape of the error

surface [32]. Other cost functions were considered as well, such as the one proposed in Silva et al. [102], but it is not clearly advantageous to to cross-entropy.

The method described in this chapter allows controlling the sensitivity of the training procedure to easy and hard examples. When using low sensitivity, this can be seen as a form of outlier detection or noise reduction. Some previous works have attempted to remove training examples that are identified as outliers or noise. Smith and Martinez [104] preprocessed data to detect label noise induced from overlapping classes, and in another work [48] the authors use an auxiliary neural network to detect noisy examples. In contrast, the approach described below requires no preprocessing and only a minimal modification to existing gradient-based algorithms, and allows emphasising examples with a large prediction bias instead of ignoring them.

The emphasis on "easy" and "hard" examples during neural network training has been addressed in the framework of Curriculum Learning [6]. In this framework, it is proposed to first present easy examples to the network, then to gradually add harder and harder examples as training continuous. In Kumar et al. [66], easy and hard examples are defined based on their fit to current model parameters. A curriculum learning algorithm is proposed, where a hyperparameter controls the ratio of easy and hard examples that are presented to the learner at any given time. The method proposed in this chapter is simpler than the curriculum learning approaches, as the examples can be presented to the network in random order.

2.3 Linear dependence on classification error

Consider a standard neural network softmax classifier. This neural network processes an input x and outputs a discrete posterior probability distribution $(p_1, ..., p_n)$ over the n possible classes. For obtaining the posterior probability distribution, the network feeds the input x through a series of layers (such as, but not limited to, fully-connected layers, convolutional layers or recurrent layers) to emit a vector of *logits* $(z_1, ..., z_n)$ that are the unnormalised class scores. Then, a softmax normalisation function is applied to normalise the logits into posterior probabilities:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}. \tag{2.1}$$

At training time, an additional class label $y \in \{1, ..., n\}$ is given to the model, and the *cross-entropy* loss function is computed:

$$\ell(x, y) = -\log(p_y). \tag{2.2}$$

When standard learning with SGD is performed, the network parameter updates are proportional to the gradient of the loss function with respect to the network parameters θ:

$$\frac{\partial \ell}{\partial \theta} = \sum_{j=1}^{n} \frac{\partial \ell}{\partial z_j} \frac{\partial z_j}{\partial \theta}, \tag{2.3}$$

As ℓ is the cross-entropy loss, we have that $\frac{\partial \ell}{\partial z_j} = \frac{\partial \ell}{\partial p_y} \frac{\partial p_y}{\partial z_j}$. Plugging in the derivatives of the cross-entropy loss and the softmax function:

$$\frac{\partial \ell}{\partial p_y} = -\frac{1}{p_y},$$
$$\frac{\partial p_y}{\partial z_j} = \begin{cases} p_y(1-p_y) & j = y, \\ -p_y p_j & j \neq y, \end{cases}$$

we get that

$$\frac{\partial \ell}{\partial z_j} = \begin{cases} p_j - 1 & y = j \\ p_j & \text{otherwise.} \end{cases} \tag{2.4}$$

We define the *prediction bias* ε_j for class j and example (x, y) as the (signed) difference between p_j, the probability assigned by the model to class j, and the target probability for this class, i.e., 1 if $j = y$ or 0 otherwise. We get that $\varepsilon_j = p_j - 1$ if $j = y$ or $\varepsilon_j = p_j$ otherwise. Plugging this back in 2.3 we get

$$\frac{\partial \ell}{\partial \theta} = \sum_{j=1}^{n} \frac{\partial z_j}{\partial \theta} \varepsilon_j. \tag{2.5}$$

As stated in Keren et al. [56] and derived from Eq. 2.5, the first conclusion for this chapter is that when using the cross entropy loss, the effect of any single training example on the gradient is linear in the prediction bias of the current network for this example. As discussed in Section 2.1, the model's classification performance may improve if a non-linear dependence of the prediction bias on the gradient / parameter updates is considered.

2.4 Generalising the gradient

Keren et al. [56] propose a generalisation of the gradient, that allows non-linear dependence of the gradient on ε. Consider a function $f : [-1, 1] \to \mathbb{R}^n$ and let $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_n)$, and define

$$g(\theta) := \sum_{j=1}^{n} \frac{\partial z_j}{\partial \theta} f_j(\varepsilon), \tag{2.6}$$

where f_j is the j'th component of f. When $f_j = \varepsilon_j$, we get $g(\theta) = \frac{\partial \ell}{\partial \theta}$, with the standard linear dependence of the parameter updates on the prediction error. However, we are now at liberty to study other assignments for f. Keren et al. [56] denote g as the *pseudo-gradient*, that can be used in place of the standard gradient in any gradient-based algorithm. Note that, as we discuss in Section 2.7 below, the pseudo-gradient g is not a gradient of any loss function.

2.5 Constraints on the pseudo-gradient

The authors of Keren et al. [56] consider what types of functions f are suitable for replacing the gradient and facilitate learning. First, f is expected be monotonic non-decreasing, to reflect a larger parameter update for larger classification errors. Further, f is expected to be positive when $j \neq y$ and negative otherwise. In addition to these natural properties, Keren et al. [56] introduce an additional non-trivial constraint on f. The motivation for this constraint in the following example. Consider a standard softmax classifier MLP with a single-hidden layer (see Figure 2.1). The inputs to the softmax layer are $z_j = \langle w_j, h \rangle + b_j$ and the outputs of the hidden layer are $h(i) = \langle w'_i, x \rangle + b'_i$, where x is the input vector, and b'_i, w'_i are the scalar bias and weight vector between the input layer and the hidden layer.

Suppose that at some point during training, hidden unit i is connected to all units j in the softmax layer with the same positive weight a, i.e., for all $1 \leq j \leq n$, $w_j(i) = a$. Now, assume training example (x, y) is presented to the model and let l be some coordinate in the input layer.

How should this training example change the weight $w'_i(l)$? If $x(l) = 0$, it does not need to make any change in the weight, therefore we consider the case where $x(l) \neq 0$. Unit i in the hidden layer is the only one affected by the change of $w'_i(l)$. From the definition of p_j, it is easy to see that the model output probabilities are fully determined by the ratios $e^{z_j}/e^{z_{j'}}$, or equivalently, by the differences $z_j - z_{j'}$, for all j, j'. Now, $z_j - z_{j'} = \langle w_j, h \rangle + b_j - \langle w_{j'}, h \rangle + b_{j'}$. Therefore, $\frac{\partial (z_j - z_{j'})}{\partial h(i)} = w_j(i) - w_{j'}(i) = a - a = 0$, and therefore

$$\frac{\partial (z_j - z_{j'})}{\partial w'_i(l)} = \frac{\partial (z_j - z_{j'})}{\partial h(i)} \frac{\partial h(i)}{\partial w'_i(l)} = 0. \tag{2.7}$$

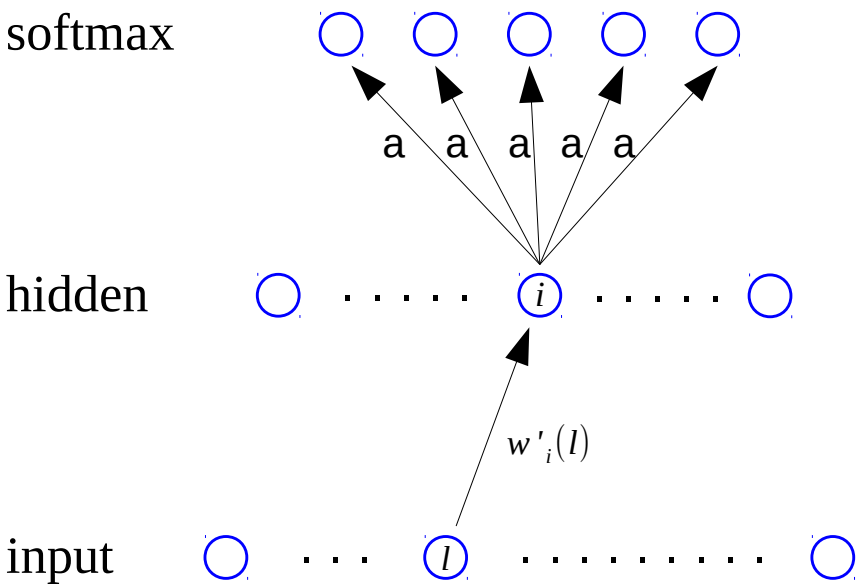


Fig. 2.1 Illustrating the motivating example for constraints on the pseudo-gradient. Reprinted with permission from Keren et al. [56].

This shows that when all weights from unit i to the output units are equal, the weight $w'_i(l)$ should not be changed for any l. In addition, Keren et al. [56] report that in their preliminary experiments, it was beneficial to indeed keep those weights fixed in those situations, as otherwise explosion of the weights may cause numerical instability.

We would like this behaviour to persist also for pseudo-gradients, i.e., to make sure that in this case $g(w'_i(l)) = 0$. Keren et al. [56] derive that

$$\begin{aligned} 0 = g(w'_i(l)) &= \sum_{j=1}^{n} \frac{\partial z_j}{\partial w'_i(l)} f(\varepsilon_j) \\ &= \sum_{j=1}^{n} \frac{\partial z_j}{\partial h(i)} \frac{\partial h(i)}{\partial w'_i(l)} f(\varepsilon_j) = \sum_{j=1}^{n} a \cdot x(l) \cdot f(\varepsilon_j). \end{aligned}$$

And after dividing by $a \cdot x(l)$, they derive the desired property for the function f, for any vector ε of prediction biases:

$$f_y(\varepsilon) = -\sum_{j \neq y} f_j(\varepsilon). \tag{2.8}$$

Note that, since $\sum_{j=1}^{n} \varepsilon_j = 0$, this property holds for the cross-entropy loss, where $f_j(\varepsilon) = \varepsilon_j$.

2.6 Polynomial dependence on the classification error

In the case of the cross-entropy loss, f was the identity function that led to a linear dependence of the gradient on the prediction error. Keren et al. [56] consider the natural generalisation of the linear dependence to a polynomial dependence. Combined with the constraint from Eq. 2.8, they propose the following assignment of the function f:

$$f_j(\varepsilon) = \begin{cases} -|\varepsilon_y|^k & j = y, \\ \frac{|\varepsilon_y|^k}{\sum_{i \neq y} \varepsilon_i^k} \cdot \varepsilon_j^k & \text{otherwise.} \end{cases} \tag{2.9}$$

where $k > 0$ is a hyperparameter. To make sure the constraint from Eq. 2.8 is satisfied, the normalisation term $\frac{|\varepsilon_y|^k}{\sum_{i \neq y} \varepsilon_i^k}$ is used. The case of $k = 1$ is equivalent to the standard gradient, and each choice of k leads to a different pseudo-gradient.

To illustrate the relationship between the value of k and the effect of prediction biases with different sizes on the pseudo-gradient, we plot $f_y(\varepsilon)$ as a function of ε_y for several values of k (see Figure 2.2). It is important to note that absolute values of the pseudo-gradient are not important, since the pseudo-gradient is normally multiplied by the learning rate and the choice of the latter can effectively determine the pseudo-gradient's magnitude.

As shown in the figure, when k is large, the pseudo-gradient (and the weight updates) is more strongly affected by examples with a large prediction bias. In other words, the learning process is much more affected by those examples that the network has a large error on, compared to examples the network classifies relatively well. This follows since $\frac{|\varepsilon|^k}{|\varepsilon'|^k}$ is monotonic increasing in k for $\varepsilon > \varepsilon'$. On the other hand, when using a small positive k we get that $\frac{|\varepsilon|^k}{|\varepsilon'|^k}$ approaches to 1, and in this case the pseudo-gradient tends to be affected by all examples in an equal matter, regardless of each example's prediction bias. In conclusion, the choice of f, parameterised by k, allows tuning the sensitivity of the training process to large errors.

2.7 Non-existence of a loss function

The above sections demonstrate how to design a pseudo-gradient in order to directly control the relative effect of hard and easy examples on the parameter updates. A natural question

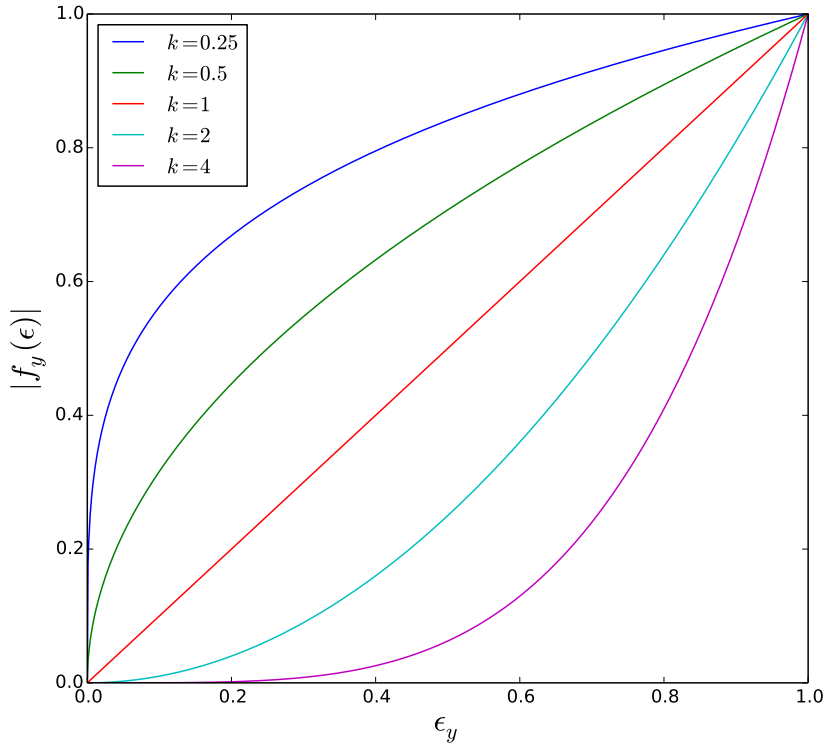


Fig. 2.2 Size of $f_y(\varepsilon)$ for different choices of k. Lines are in the same order as in the legend. Reprinted with permission from Keren et al. [56].

that arises is whether this pseudo-gradient is the gradient of some loss function. Keren et al. [56] show that the answer is in general negative.

Lemma 1. *Assume f as in Eq. 2.9 with $k \neq 1$, and $g(\theta)$ the resulting pseudo-gradient. There exists a neural network with parameters Θ for which $\{g(\theta)\}_{\theta\in\Theta}$ is not a gradient of any cost function.*

We bring the proof of this lemma as it appears in Keren et al. [56].

Proof. Consider a neural network with three units in the output layer, and at least one hidden layer. Let (x,y) be a labelled example, and suppose that there exists some cost function $\bar{\ell}((x,y);\Theta)$, differentiable in Θ, such that for g as defined in Eq. 2.6 and f defined in Eq. 2.9 for some $k > 0$, we have $g(\theta) = \frac{\partial\bar{\ell}}{\partial\theta}$ for each parameter θ in Θ. We now show that this is only possible if $k = 1$.

Under the assumption on $\bar{\ell}$, for any two parameters θ_1, θ_2,

$$\frac{\partial}{\partial \theta_2}\left(\frac{\partial \bar{\ell}}{\partial \theta_1}\right) = \frac{\partial^2 \bar{\ell}}{\partial \theta_1 \theta_2} = \frac{\partial}{\partial \theta_1}\left(\frac{\partial \bar{\ell}}{\partial \theta_2}\right),$$

hence

$$\frac{\partial g(\theta_1)}{\partial \theta_2} = \frac{\partial g(\theta_2)}{\partial \theta_1}. \tag{2.10}$$

Recall our notations: $h(i)$ is the output of unit i in the last hidden layer before the softmax layer, $w_j(i)$ is the weight between the hidden unit i in the last hidden layer, and unit j in the softmax layer, z_j is the input to unit j in the softmax layer, and b_j is the bias of unit j in the softmax layer.

Let (x,y) such that $y = 1$. From Eq. 2.10 and Eq. 2.6, we have

$$\frac{\partial}{\partial w_2(1)} \sum_{j=1}^{n} \frac{\partial z_j}{\partial w_1(1)} f_j(\varepsilon) = \frac{\partial}{\partial w_1(1)} \sum_{j=1}^{n} \frac{\partial z_j}{\partial w_2(1)} f_j(\varepsilon).$$

Plugging in f as defined in Eq. 2.9, and using the fact that $\frac{\partial z_j}{\partial w_i(1)} = 0$ for $i \neq j$, we get:

$$-\frac{\partial}{\partial w_2(1)}\left(\frac{\partial z_1}{\partial w_1(1)} \cdot |\varepsilon_1|^k\right) = \frac{\partial}{\partial w_1(1)}\left(\frac{\partial z_2}{\partial w_2(1)} \cdot \frac{\varepsilon_2^k}{\varepsilon_2^k + \varepsilon_3^k} \cdot |\varepsilon_1|^k\right).$$

Since $y = 1$, we have $\varepsilon = (p_1 - 1, p_2, p_3)$. In addition, $\frac{\partial z_j}{\partial w_j(1)} = h(1)$ and $\frac{\partial h(1)}{\partial w_j(1)} = 0$ for $j \in [2]$. Therefore

$$-\frac{\partial}{\partial w_2(1)}\left((1-p_1)^k\right) = \frac{\partial}{\partial w_1(1)}\left(\frac{p_2^k}{p_2^k + p_3^k}(1-p_1)^k\right). \tag{2.11}$$

Next, we evaluate each side of the equation separately, using the following:

$$\frac{\partial p_j}{\partial w_j(1)} = \frac{\partial p_j}{\partial z_j}\frac{\partial z_j}{\partial w_j(1)} = h(1)p_j(1-p_j),$$
$$\forall j \neq i, \quad \frac{\partial p_j}{\partial w_i(1)} = \frac{\partial p_j}{\partial z_i}\frac{\partial z_i}{\partial w_i(1)} = -h(1)p_i p_j.$$

For the LHS of Eq. 2.11, we have

$$-\frac{\partial}{\partial w_2(1)}(1-p_1)^k = -k(1-p_1)^{k-1}h(1)p_1p_2.$$

For the RHS,

$$\frac{\partial}{\partial w_1(1)}\frac{p_2^k}{p_2^k+p_3^k}(1-p_1)^k = -\frac{kh(1)p_1p_2^k(1-p_1)^k}{p_2^k+p_3^k}.$$

Hence Eq. 2.11 holds if and only if:

$$1 = \frac{p_2^{k-1}(1-p_1)}{p_2^k+p_3^k}.$$

For $k=1$, this equality holds since $p_1+p_2+p_3=1$. However, for any $k\neq 1$, there are values of p_1,p_2,p_3 such that this does not hold. We conclude that our choice of f does not lead to a pseudo-gradient g which is the gradient of any cost function. □

2.8 A toy example

To further motivate the choice of f given in Eq. 2.9, Keren et al. [56] illustrate the benefits of this choice using a toy example with simple data distribution and neural network. They consider a neural network with no hidden layers, and a single input unit that is connected to an output layer with two units and softmax activation. The input to the network is denoted as x, and the output unit i before softmax activation is given by $z_i = xw_i + b_i$, where w_i and b_i are the network's learnable weights and biases respectively for $i \in \{0,1\}$. The network's *prediction function* maps a given input to a class label and is denoted as $\hat{y}(x) \mapsto \{-1,1\}$.

It is not hard to see that the set of prediction functions that this simple network can represent is exactly the set of threshold functions, namely functions of the form $\hat{y}(x) = \text{sign}x - t$ or $\hat{y}(x) = -\text{sign}x - t$ for $t \in \mathbb{R}$, where t is referred to as the *decision threshold*. Let $\alpha \in (\frac{1}{2},1)$, and suppose that labelled examples are drawn independently at random from the following distribution $\mathscr{D}$ over $\mathbb{R}\times\{-1,+1\}$: uniformly sampling x from $[-1,1]$, and the label is set to 1 for $x \in [0,\alpha]$ and -1 otherwise. For this distribution, the prediction function that yields the best accuracy is $\hat{y}(x) = \text{sign}(x)$, i.e., the decision threshold is 0. Keren et al. [56] show that when using the cross-entropy loss for this learning problem, the resulting prediction function is not optimal. Specifically, they formulate and prove the following lemma:

Lemma 2. *Assume the cross-entropy loss is used for learning the classification problem with network and data structure described above. Given a large enough training set, the value of the cross-entropy loss is not minimal when the decision threshold is* 0*.*

We give the proof of this lemma as appeared in Keren et al. [56].

Proof. Suppose that there is an assignment of network parameters that minimises the cross-entropy which induces a threshold at 0. The output of the softmax layer is determined uniquely by $\frac{e^{z_0}}{e^{z_1}}$, or equivalently by $z_0 - z_1 = x(w_0 - w_1) + b_0 - b_1$. Therefore, we can assume without loss of generality that $w_1 = b_1 = 0$. Denote $w := w_0, b := b_0$. If $w = 0$ in the minimising assignment, then all examples are classified as members of the same class and in particular, the classification threshold is not zero. Therefore we may assume $w \neq 0$. In this case, the classification threshold is $\frac{-b}{w}$. Since we assume a minimal solution at zero, the minimising assignment must have $b = 0$.

When the training set size approaches infinity, the cross-entropy on the sample approaches the expected cross-entropy on $\mathscr{D}$. Let $\mathrm{CE}(w,b)$ be the expected cross-entropy on $\mathscr{D}$ for network parameter values w, b. Then

$$\mathrm{CE}(w,b) = -\frac{1}{2}\Big(\int_{-1}^{0} \log(p_0(x))dx + \int_{0}^{\alpha} \log(p_1(x))dx + \int_{\alpha}^{1} \log(p_0(x))dx\Big).$$

And we have:

$$\log(p_0(x)) = \log\frac{e^{wx+b}}{e^{wx+b}+1} = wx + b - \log(e^{wx+b}+1),$$
$$\log(p_1(x)) = \log\frac{1}{e^{wx+b}+1} = -\log(e^{wx+b}+1).$$

Therefore

$$\frac{\partial \mathrm{CE}(w,b)}{\partial b} = -\frac{1}{2}\frac{\partial}{\partial b}\Big(\int_{-1}^{0}(wx+b)dx - \int_{-1}^{0}\log(e^{wx+b}+1)dx - \int_{0}^{\alpha}\log(e^{wx+b}+1)dx + \int_{\alpha}^{1}(wx+b)dx - \int_{\alpha}^{1}\log(e^{wx+b}+1)dx\Big)$$

$$= -\frac{1}{2}(1 - \frac{\partial}{\partial b}\left(\int_{-1}^{1} \log(e^{wx+b}+1)dx\right) + 1 - \alpha).$$

Differentiating under the integral sign, we get

$$\frac{\partial \mathrm{CE}(w,b)}{\partial b} = -\frac{1}{2}\left(2 - \alpha - \int_{-1}^{1} \frac{e^{wx+b}}{e^{wx+b}+1}\right)$$

Since we assume the cross-entropy has a minimal solution with $b = 0$, we have

$$\begin{aligned} 0 &= -2\frac{\partial \mathrm{CE}(w, b=0)}{\partial b} \\ &= 2 - \alpha - \frac{1}{w}(\log(e^{w}+1) - \log(e^{-w}+1)). \end{aligned}$$

Therefore

$$w(2-\alpha) = \log\frac{e^{w}+1}{e^{-w}+1} = \log(e^{w}) = w.$$

Since $\alpha \neq 1$, it must be that $w = 0$. This contradicts our assumption, hence the cross-entropy does not have a minimal solution with a threshold at 0. □

The intuition of the above proof can be traced to the fact that inputs to the network in the range $(\alpha, 1]$ cannot be classified correctly when the decision threshold is close to 0, therefore they result in large loss and shift the optimal decision threshold to be larger than 0. Thus, for this simple setting, there is motivation to move away from the cross-entropy loss to an alternative training mechanism, that is less affected by the large errors. This can be achieved using the update rule proposed in Keren et al. [56], with $k < 1$. Using $k < 1$, the pseudo-gradient is less sensitive to large errors, and this should allow shifting the decision threshold closer to the optimal threshold of 0. On the other hand, using $k > 1$ will cause the opposite and undesirable effect, and the decision threshold would move further away from 0 towards a larger value, to better accommodate the inputs in $(\alpha, 1]$. Thus, it is expected that using the above described update rule will yield a monotonically increasing classification error in k, i.e., the smaller the values of k will be, the better the classification accuracy should be.

Keren et al. [56] conduct experiments using their proposed update rule and the network and data structure described above, with $\alpha = 0.95$, to validate this hypothesis. They generated 30,000 examples for each of the training, validation and test sets. Weights were initialised by uniformly sampling from the interval $(-0.1, 0.1)$ and biases were initialised to 0. Batch gradient descent with a learning rate of 0.01 was used, i.e., the pseudo-gradient is calculated using all training examples for every update of the four network parameters. The pseudo-gradient was calculated according to Eq. 2.6 with f defined in Eq. 2.9. A variety of k values

Table 2.1 Toy example experiment results as presented in Keren et al. [56]

k	Test error	Threshold	CE Loss
4	8.36%	0.116	0.489
2	6.73%	0.085	0.361
1	4.90%	0.049	0.288
0.5	4.27%	0.037	0.299
0.25	4.04%	0.030	0.405
0.125	3.94%	0.028	0.625
0.0625	3.61%	0.022	1.190

were used, ranging from 0.0625 to 4. After each training iteration the misclassification rate on the validation set was computed, and training was stopped after 3000 iterations with no improvement larger than 0.001%. At the end of training the misclassification rate on the test set was computed.

Table 2.1 reports the results of the experiments performed by Keren et al. [56]. For each value of k, the table reports the misclassification rate on the test set, the resulting decision threshold and the cross-entropy loss on the test set, all averaged across 10 runs for each value of k. The results confirm the hypothesis regarding the behaviour of this network with the different values of k, and further motivate the benefits of using $k \neq 1$. The misclassification rate on the test set is monotonic in k, demonstrating that indeed the smallest value of k was optimal for this problem. In particular, the optimal value of k yielded better performance compared to $k = 1$, which is equivalent to training with the cross-entropy loss. Moreover, the value of the cross-entropy loss (that was computed, even though this loss was not used for training) is not aligned with the misclassification rate, further supporting the claim that training using the cross-entropy loss may be suboptimal in some cases, and the update rule proposed in Keren et al. [56] is preferable.

2.9 Experiments

[56] conduct experiments using four benchmark image classification datasets: the MNIST dataset [72], consisting of grayscale 28x28 pixel images of handwritten digits, with 10 classes, 60,000 training examples and 10,000 test examples, the Street View House Numbers dataset (SVHN) [85], consisting of RGB 32x32 pixel images of digits cropped from house numbers, with 10 classes 73,257 training examples and 26,032 test examples and the CIFAR-10 and CIFAR-100 datasets [64], consisting of RGB 32x32 pixel images of 10/100 object

classes, with 50,000 training examples and 10,000 test examples. All datasets were linearly transformed such that all features are in the interval $[-1,1]$.

The neural networks that were used for these experiments contain either one, three or five hidden layers with different number of hidden units. SGD with momentum [108] is used for optimisation with batch size of 128 examples and different values for the learning rate and momentum hyperparameters. For each value of k, the pseudo-gradient is used according to Eq. 2.6 with the choice of f from Eq. 2.9. Gradient-Clipping [91] with a threshold of 100 is used for the network with more than one hidden layer. Weights of the hidden layers were initialised using the initialisation scheme from Glorot and Bengio [32], weights of the output layer and all biases were initialised with 0.

In each experiment, cross-validation was used to select the best value of k. For networks with one hidden layer, k was selected out of the values $\{4,2,1,0.5,0.25,0.125,0.0625\}$. For networks with 3 or 5 hidden layers, k was selected out of the values $\{4,2,1,0.5,0.25\}$, where the smaller values were removed due to their performance in preliminary experiments with deeper networks. The learning rate was optimised using cross-validation for each value of k separately, since the size of the pseudo-gradient vector can differ by orders of magnitude for different values of k, as is evident from Eq. 2.9.

For each experiment configuration, defined by a dataset, network architecture and momentum, Keren et al. [56] selected an initial learning rate η, based on preliminary experiments on the training set. The following procedure was then carried out for $\eta/2,\eta,2\eta$, for every tested value of k:

1. Randomly split the training set into 5 equal parts, $S_1,\ldots,S_5$.
2. Run the iterative training procedure on $S_1 \cup S_2 \cup S_3$, until there is no improvement in test prediction error for 15 epochs on the early stopping set, S_4.
3. Select the network model that did the best on S_4.
4. Calculate the validation error of the selected model on the validation set, S_5.
5. Repeat the process for t times after permuting the roles of $S_1,\ldots,S_5$. We set $t = 10$ for MNIST, and $t = 7$ for CIFAR-10/100 and SVHN.
6. Let $\mathrm{err}_{k,\eta}$ be the average of the t validation errors.

Finally, $\mathrm{argmin}_\eta \mathrm{err}_{k,\eta}$ is computed. If the minimum was found with the minimal or the maximal η that was tried, the above process was also performed using half the η or double the η, respectively. This continued iteratively until there was no need to add learning rates. At the end of this process $(k^*,\eta^*) = \mathrm{argmin}_{k,\eta} \mathrm{err}_{k,\eta}$ is selected, and the network is retrained with

parameters k^*, η^* on the training sample, using one fifth of the sample as an early stopping set. Test error of the resulting model is compared to the test error of a model retrained in the same way, except that we set $k = 1$ (leading to standard cross-entropy training), and the learning rate to $\eta_1^* = \text{argmin}_\eta \, \text{err}_{1,\eta}$. The final learning rates in the selected models were in the range $[10^{-1}, 10]$ for MNIST, and $[10^{-4}, 1]$ for the other datasets.

The results from Keren et al. [56] are reported in Tables 2.2, 2.3, 2.4 for experiments using networks with one, three and five layers respectively. The best performing value of k is reported, in addition to the test misclassification rate and cross-entropy loss on the test set using $k = 1$ and the selected k. The first observation from the results in the tables is that $k < 1$ was performing best for shallow networks, $k > 1$ for deeper networks, and k values of around 1 for networks with intermediate depth. This finding is aligned with the hypothesis from Section 2.1, and indeed points out that the optimal sensitively level to hard examples is correlated with network capacity.

Further, for the shallow networks the test cross-entropy loss is almost always larger with the selected k than with $k = 1$. This finding shows that the optimisation process proposed in Keren et al. [56] does not necessarily minimises the cross-entropy loss, but yet optimises for the true task performance measure, that is the misclassification rate. On the contrary, in the experiments with three and five layers, the cross-entropy is improved when using the best performing k value. This may be explained by the fact that examples with a large prediction bias have a high cross-entropy loss, and so focusing training on these examples reduces the empirical cross-entropy loss, and therefore also the true cross-entropy loss. Overall, the results demonstrate that, as expected, in many cases the optimal value of k differ from the default $k = 1$, and is correlated with network capacity, and that the learning process can benefit from using a learning algorithm that is free of loss functions.

Table 2.2 Experiment results for single-layer networks, as appear in Keren et al. [56]. Mom' stands for the momentum value.

				Test Error [%]		Test CE Loss	
Dataset	Units	Mom'	Selected k	$k=1$	Selected k	$k=1$	Selected k
MNIST	400	0	0.5	1.71	**1.70**	**0.0757**	0.148
MNIST	800	0	0.5	**1.66**	1.67	**0.070**	0.137
MNIST	1100	0	0.5	1.64	**1.62**	**0.068**	0.131
MNIST	400	0.5	0.5	1.76	**1.74**	**0.078**	0.167
MNIST	800	0.5	0.5	1.67	**1.65**	**0.072**	0.150
MNIST	1100	0.5	0.5	1.67	**1.65**	**0.071**	0.145
MNIST	400	0.9	0.5	1.75	1.75	**0.073**	0.140
MNIST	800	0.9	2	1.71	**1.63**	0.070	**0.054**
MNIST	1100	0.9	0.5	1.74	**1.69**	**0.069**	0.127
SVHN	400	0	0.25	16.84	**16.09**	**0.658**	1.575
SVHN	800	0	0.25	16.19	**15.71**	**0.641**	1.534
SVHN	1100	0	0.25	15.97	**15.68**	**0.636**	1.493
SVHN	400	0.5	0.25	16.88	**16.16**	**0.661**	1.576
SVHN	800	0.5	0.125	16.09	**15.64**	**0.648**	3.108
SVHN	1100	0.5	0.25	16.04	**15.53**	**0.626**	1.525
SVHN	400	0.9	0.125	16.65	**16.30**	**0.679**	2.861
SVHN	800	0.9	0.25	16.15	**15.68**	**0.675**	1.632
SVHN	1100	0.9	0.25	15.85	**15.47**	**0.640**	1.657
CIFAR-10	400	0	0.125	48.15	**46.91**	**1.435**	5.609
CIFAR-10	800	0	0.125	46.92	**46.14**	**1.390**	5.390
CIFAR-10	1100	0	0.125	46.63	**46.00**	**1.356**	5.290
CIFAR-10	400	0.5	0.25	48.32	**47.06**	**1.430**	3.034
CIFAR-10	800	0.5	0.125	46.91	**46.01**	**1.388**	5.645
CIFAR-10	1100	0.5	0.25	46.43	**45.84**	**1.410**	2.820
CIFAR-10	400	0.9	0.0625	48.19	**46.71**	**1.518**	11.049
CIFAR-10	800	0.9	0.125	47.09	**46.16**	**1.616**	5.294
CIFAR-10	1100	0.9	0.125	46.71	**45.77**	**1.850**	5.904
CIFAR-100	400	0.5	0.25	75.18	**74.41**	**3.302**	6.931
CIFAR-100	800	0.5	0.25	74.04	**73.78**	**3.260**	7.449
CIFAR-100	1100	0.5	0.125	73.69	**73.11**	**3.239**	13.557
CIFAR-100	400	0.9	0.25	74.96	**74.28**	**3.306**	7.348
CIFAR-100	800	0.9	0.125	74.12	**73.47**	**3.327**	13.267
CIFAR-100	1100	0.9	0.25	73.47	**73.19**	**3.235**	7.489

Table 2.3 Experiment results for 3-layer networks, as appear in Keren et al. [56]. Mom' stands for the momentum value.

				Test Error [%]		Test CE Loss	
Dataset	Units	Mom'	Selected k	$k=1$	Selected k	$k=1$	Selected k
MNIST	400	0	1	—	—	—	—
MNIST	800	0	1	—	—	—	—
MNIST	400	0.5	1	—	—	—	—
MNIST	800	0.5	1	—	—	—	—
MNIST	400	0.9	1	—	—	—	—
MNIST	800	0.9	0.5	1.60	**1.53**	**0.091**	0.189
SVHN	400	0.5	2	16.52	16.52	1.604	**0.968**
SVHN	800	0.5	1	—	—	—	—
SVHN	400	0.9	1	—	—	—	—
SVHN	800	0.9	2	16.14	**15.96**	1.651	**1.062**
CIFAR-10	400	0.5	2	46.81	**46.63**	3.023	**2.121**
CIFAR-10	800	0.5	1	—	—	—	—
CIFAR-10	400	0.9	2	47.52	**46.92**	2.226	**2.010**
CIFAR-10	800	0.9	2	45.27	**44.26**	2.855	**2.341**
CIFAR-100	400	0.5	0.5	75.20	**74.95**	**3.378**	4.511
CIFAR-100	800	0.5	1	—	—	—	—
CIFAR-100	400	0.9	0.25	74.97	**74.52**	**3.356**	8.520
CIFAR-100	800	0.9	0.5	74.48	**73.17**	**4.133**	8.642

Table 2.4 Experiment results for 5-layer networks, as appear in Keren et al. [56]. Mom' stands for the momentum value.

				TEST ERROR [%]		TEST CE LOSS	
DATASET	UNITS	MOM'	SELECTED k	$k=1$	SELECTED k	$k=1$	SELECTED k
MNIST	400	0.5	0.5	1.71	**1.69**	**0.113**	0.224
MNIST	800	0.5	0.25	1.61	**1.60**	**0.118**	0.390
MNIST	400	0.9	1	—	—	—	—
MNIST	800	0.9	4	**1.58**	1.60	0.098	**0.060**
SVHN	400	0.5	4	17.41	**16.49**	1.436	**0.708**
SVHN	800	0.5	0.5	17.07	**16.61**	**1.343**	2.604
SVHN	400	0.9	4	17.89	**16.54**	1.284	**0.718**
SVHN	800	0.9	2	16.24	**15.73**	1.647	**0.998**
CIFAR-10	400	0.5	2	48.05	**47.85**	2.017	**1.962**
CIFAR-10	800	0.5	4	**44.21**	44.24	4.610	**1.677**
CIFAR-10	400	0.9	4	47.91	**47.57**	2.202	**1.648**
CIFAR-10	800	0.9	2	45.69	**44.11**	3.316	**2.171**
CIFAR-100	400	0.5	2	75.69	**75.48**	3.611	**3.228**
CIFAR-100	800	0.5	2	74.10	**73.57**	4.650	**4.439**
CIFAR-100	400	0.9	1	—	—	—	—
CIFAR-100	800	0.9	4	**74.32**	74.62	3.872	**3.432**

Chapter 3

The Principle of Logit Separation

3.1 Motivation

With the increase in available computational resources and data collection rate, machine learning models are often required to perform fine-grained classification over a very large number of classes. These classification problems naturally appear in computer vision, language modeling and machine translation, and were studied in several works [37, 124]. Datasets with up to hundreds of thousands of classes are already used in industry [22, 90].

For neural network classifiers, the large number of classes can imply a high computational burden at test time. Indeed, when using a standard neural network classifier with a softmax layer, computing the class logits, that are the unnormalised class scores, is linear in the number of classes [35]. This can be prohibitively slow for high-load systems, such as search engines or real-time machine translation systems.

However, in many applications the task at test time is not a full classification of a given example. Instead, every time the classifier is used, the task is to detect whether a given example belongs to a small subset of classes, possibly containing a single class. The small subset of possible classes to test for can change every time the classifier is used. Consider the case of real-time image search [78, 95] from a live of feed of multiple cameras. When the user queries for images that contain an instance of class A, the classifier needs to process a large number of images and decide whether each image contains an instance of class A. Using the system for the seconds time, the user now queries for images containing instances of class B. New images are now processed through the classifier, to determine which of them contain an instance of class B. This setting can apply to cameras installed on autonomous vehicles, security cameras for detecting objects of interest, or live face recognition, where the person to be identified changes based on a user query.

In the setting we consider, the classifier will have to support a single class (or a small set of classes) for every use, but as this class may change each time the classifier is used, in total this classifier will need to support a large number of possible classes across all the different times it is used. As the number of classes is large, it is reasonable to train a single classifier that can handle queries of all possible classes, instead of training a separate classifier for each class.

For this type of applications, one would ideally want a test time computation that does not depend on the total number of possible classes. A natural approach is to compute a single logit that corresponds to the class of interest, and decide whether the input instance belongs to this class based on this logit alone. However, the value of this logits alone may not be enough to indicate whether its corresponding class is correct, as it may be meaningful only in comparison to the values of other classes' logits. In such case, the value of this single logit alone cannot be used to determine class correctness. Keren et al. [57] name this binary classification task of inferring class correctness from its corresponding logit alone *Single Logit Classification* (SLC). Figure 3.1 demonstrates the speedups yielded by SLC, i.e., by computing only a single logit instead of logits of all possible classes, as the number of classes increase. For instance, computing only a single logit yields a 10x speedup at test time, when the number of possible classes is approximately $400,000$. The speedup increases with the number of possible classes. See Section 3.7.3 for more details about the obtained speedups.

Motivated by the research question posed in Section 1.2, Keren et al. [57] find that the loss function used affects the nature of the representation in the logits layer in a manner that is strongly connected to accuracy in the SLC task. As a first result, Keren et al. [57] show that when using the cross-entropy loss for training a neural network classifier, a single logit at test time is not informative enough to classify whether it belongs to the correct class of the example. This means, that the cross-entropy loss yields poor performance in the SLC task. Further, Keren et al. [57] identify a simple principle they name *the Principle of Logit Separation* (PoLS), that is an essential characteristic a loss function must have in order to yield good performance in SLC. The principle states that a loss function should optimise for the following property regarding the representation in the logits layer:

> *The value of any logit that belongs to the correct class of any training example should be larger than the value of any logit that belongs to a wrong class of any (same or other) training example.*

A formal definition of the Principle of Logit Separation is given in Section 3.3. See Figure 3.2 for an illustration. Keren et al. [57] and Keren et al. [58] study the alignment of nine existing loss functions with the PoLS, and find that the PoLS is satisfied by the self-normalisation [24] and Noise-Contrastive Estimation [81] training objectives, proposed

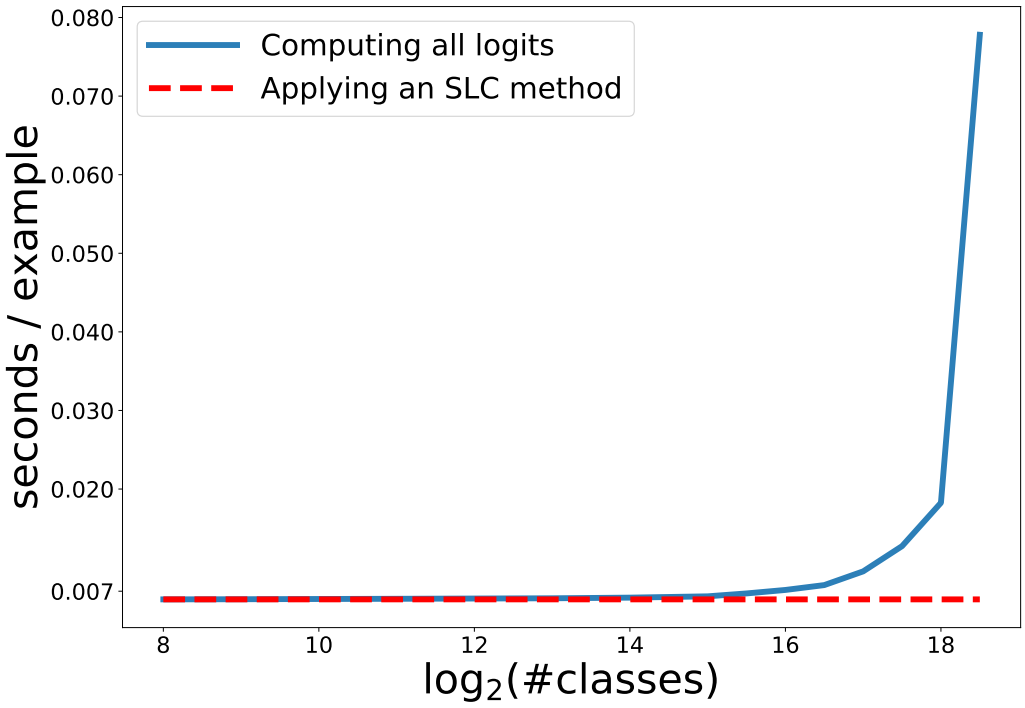


Fig. 3.1 Computation time of the logits from network inputs, using an inception-V3 image classification architecture [111], where the topmost layer is replaced with the appropriate number of classes. When applying SLC, computation cost is fixed regardless of the number of classes, which can lead to considerable speedups when the number of classes is large. Reprinted with permission from Keren et al. [57].

for calculating posterior distributions in the context of natural language processing, as well as by the binary cross-entropy loss used in multilabel settings [45, 121] and the sigmoid Tanimoto loss [47] that is inspired by a distance measure for binary vectors. In contrast, the principle is not satisfied by the standard cross-entropy loss, the max-margin loss, the softmax Tanimoto loss [47], the sigmoid Cauchy-Schwartz divergence and the softmax Cauchy-Schwarz divergence [20, 47]. In addition, two novel loss functions are proposed by Keren et al. [57], namely the batch cross-entropy and the batch max-margin losses, that are extensions of the cross-entropy and max-margin losses, that are designed to operate on batches of examples and to satisfy the PoLS. In total, eleven different loss functions are studied. `Tensorflow` code for optimising the new batch losses is publicly available at https://github.com/EIHW/Logit_Separation.

Experiments performed in Keren et al. [57] show that SLC accuracy of loss functions that satisfy the PoLS is considerably higher compared to loss functions that are not aligned

with the PoLS. This verifies that the Principle of Logit Separation is indeed a crucial ingredient in shaping the network representations in the logits layer in a manner that yields good SLC accuracy, thus partially answering the research question posed in Section 1.2. Specifically, objectives that satisfy the Principle of Logit Separation produce logits that are more informative as a standalone value, and in almost all cases yielded a 20% relative accuracy improvement over loss functions that are not aligned with the PoLS, such as the standard cross-entropy loss, and in many cases a much larger improvement. In another set of experiments, Keren et al. [57] show that when using loss functions that are aligned with the PoLS, SLC does not cause any decrease in binary classification accuracy for a given class, compared to the case where all logits are computed, while keeping the computation cost independent of the total number of classes. Finally, Keren et al. [57] compute the speedup factor one may gain by applying SLC instead of computing all classes' logits, and show considerable speedup gains for a large number of classes.

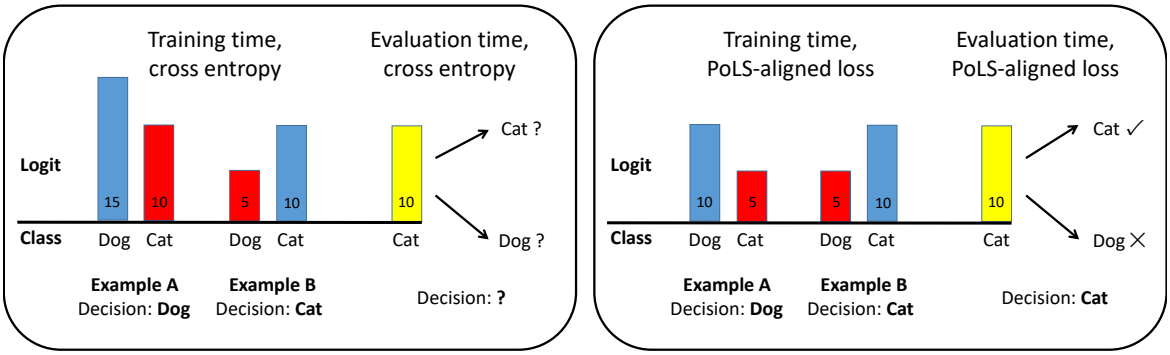


Fig. 3.2 The Principle of Logit Separation. Left: When training with the cross-entropy loss, there is no constraint on the relationship between true and false logits in different examples. Therefore, the logit for the class 'Cat' can have the same value in two training examples, once as a true logit (blue) and once as a false logit (red). At test time, there is no way to decide whether this value (yellow) indicate that the logit is true or false. Right: When training with a loss function that is aligned with the PoLS, true logits are larger than false logits across the training set. Therefore, logit values at test time better indicate whether this is a true or false logit. Reprinted with permission from Keren et al. [57].

3.2 Related work

This section reviews existing methods that are relevant for faster test-time classification. The hierarchical softmax layer [82] replaces the flat softmax layer with a binary tree with classes

as leaves, making the computational complexity of calculating the posterior probability of each class logarithmic in the number of classes. A drawback of this method is the additional construction of the binary tree of classes, which requires expert knowledge or data-driven methods. Inspired by the hierarchical softmax approach, Grave et al. [35] exploit unbalanced word distributions to form clusters that explicitly minimise the average time for computing the posterior probabilities over the classes. The authors report an impressive speedup factor of between 2 and 10 for posterior probability computation, but their computation time still depends on the total number of classes. Differentiated softmax was introduced in Chen et al. [15] as a less computationally expensive alternative to the standard softmax mechanism, in the context of neural language models. With differentiated softmax, each class (word) is represented in the last hidden layer using a different dimensionality, with higher dimensions for more frequent classes. This allows a faster computation for less frequent classes. However, this method is applicable only for highly unbalanced class distributions. Several sampling-based approaches were developed in the context of language modeling, with the goal of approximating the softmax function at training time. Notable examples are importance sampling [7], negative sampling [80], and Noise Contrastive Estimation (NCE) [38, 81]. These methods do not necessarily improve the test time computational burden, however it is shown below that the NCE loss can be used for the SLC task.

3.3 The Principle of Logit Separation

When attempting to perform the SLC task, an example is represented only by the single logit that corresponds to the class of interest. Since this is a binary classification task, a natural approach is to set a classification threshold, such that if the logit is above the threshold then the example is classified as belonging to the class of interest. We refer to logits that should be classified as positive, i.e., logits that belong to the true class of their examples, as *true logits*, and refer to other logits as *false logits*. In order for the threshold approach to work well, we would want all true logits to have larger values than all false logits, across the entire training sample (in fact, it is sufficient to require this for every class separately, but for simplicity we stick to the stronger requirement). The Principle of Logit Separation (PoLS) captures this desired property, as illustrated in Figure 3.2.

Keren et al. [57] formally define the Principle of Logit Separation: Let $[k] := \{1, \ldots, k\}$ be the possible class labels. Assume that the training sample is $S = ((x_1, y_1), \ldots, (x_n, y_n))$, where $x_i \in \mathbb{R}^d$ are the training examples, and $y_i \in [k]$ are the labels of these examples. For a neural network model parameterised by θ, denote by $z_y^\theta(x)$ the value of the logit assigned by

the model to example x for class y. The Principle of Logit Separation (PoLS) can be formally stated as follows:

Definition 1 (The Principle of Logit Separation)**.** *The* Principle of Logit Separation *holds for a labelled set S and a model θ, if for any $(x,y),(x',y') \in S$ (including the case $x = x', y = y'$) and any $y'' \neq y'$, we have $z^{\theta}_{y}(x) > z^{\theta}_{y''}(x')$.*

The above definition assures that every true logit $z^{\theta}_{y}(x)$ is larger than any false logit $z^{\theta}_{y''}(x')$, across the entire labelled set S. When this property holds across the entire train and test examples, it guarantees a perfect accuracy in the SLC task, since true logits can be separated from false logits using a simple classification threshold. Therefore, a natural approach is to use a training objective that optimises for this property on the training set. For a loss ℓ, denote $\ell(S,\theta)$ as the value of this loss function on a sample S with a model θ. A loss ℓ is said to be aligned with the PoLS if a small enough value of $\ell(S,\theta)$ assures that the requirement from Definition 1 holds for the model θ. The following sections study the alignment with the PoLS of several loss functions.

3.4 Existing objectives that do not satisfy the PoLS

In this section we discuss a number of loss functions previously proposed in the literature, and show, as was shown in Keren et al. [57] and Keren et al. [58], that these classification loss functions are not aligned with the PoLS.

3.4.1 The cross-entropy loss

As discussed in the introduction to this work, the cross-entropy loss is the standard loss function for neural network classifiers. On a single example, it is defined as follows:

$$\ell(z,y) = -\log(p_y), \tag{3.1}$$

where

$$p_y := \frac{e^{z_y}}{\sum_{j=1}^{k} e^{z_j}} = \Big(\sum_{j=1}^{k} e^{z_j - z_y}\Big)^{-1}.$$

Note that p_y is the probability assigned by the softmax layer. It is not hard to see that the cross-entropy loss is not aligned with the PoLS. As the loss depends only on the difference between the different logits, minimising it would lead to a large difference between true and false logits for every example separately. However, it does not guarantee that true logits in one training example are larger than false logits in another training example. Formally, Keren et al. [57]

provide the following counter-example that shows that this loss function is not aligned with the PoLS. Let $S = ((x_1, 1), (x_2, 2))$ be the training sample, and let θ_α, for $\alpha > 0$, be a model such that $z^{\theta_\alpha}(x_1) = (2\alpha, \alpha)$, and $z^{\theta_\alpha}(x_2) = (-2\alpha, -\alpha)$. Then, $\ell(S_{\theta_\alpha}) = 2\log(1 + e^{-\alpha})$. Therefore, for any $\varepsilon > 0$, there is some $\alpha > 0$ such that $\ell(S_{\theta_\alpha}) \leq \varepsilon$, but $z_2^{\theta_\alpha}(x_1) > z_2^{\theta_\alpha}(x_2)$, contradicting an alignment with PoLS.

3.4.2 The max-margin loss

The max-margin training objectives are mostly known for their role in Support Vector Machines. These training objectives were also used for neural network models [47, 105]. In this chapter we consider the multiclass classification version proposed in Crammer and Singer [18], that is defined for a single example as follows:

$$\ell(z, y) = \max(0, \gamma - z_y + \max_{j \neq y} z_j), \tag{3.2}$$

where $\gamma > 0$ is a hyperparameter, denoting the minimal desired separation margin between true and false logits of a given example. One can see that this loss function is not aligned with the PoLS. Similarly to the cross-entropy loss, the max-margin loss optimises for a γ difference between the true and false logits of a given examples. However, this does not guarantee that true logits in one example are larger than false logits in another example. Keren et al. [57] give a precise counter example proving that this loss function is not aligned with the PoLS: consider the same training sample S as defined in the counter-example for the cross-entropy loss above, and the model θ_α defined there. Setting $\alpha = \gamma$, we have $\ell(S_{\theta_\gamma}) = 0$. Thus for any $\varepsilon > 0$, $\ell(S_{\theta_\gamma}) < \varepsilon$, but $z_2^{\theta_\gamma}(x_1) > z_2^{\theta_\gamma}(x_2)$, contradicting an alignment with PoLS.

3.4.3 Softmax Cauchy-Schwarz divergence

The Cauchy-Schwarz divergence is based on the Cauchy-Schwarz inequality for inner products, and was defined in Kampa et al. [51] for two non-negative real vectors:

$$D_{CS}(f, g) = -\log \frac{f \cdot g}{||f|| \, ||g||}, \tag{3.3}$$

where $\cdot$ denotes an inner product and $||\cdot||$ is an L2 norm. The Cauchy-Schwarz divergence can be viewed as first computing the cosine similarity between the vectors f and g, which results in a value $0 \leq \cos\alpha \leq 1$ where α is the angle between f and g. For two parallel vectors $\cos\alpha = 1$, while $\cos\alpha = 0$ for two perpendicular vectors. Then, a negative logarithm is applied, similarly to the cross-entropy loss, to transform the result to range from 0 for

parallel vectors to ∞ for perpendicular vectors, which makes it a suitable training objective for matching the vectors f and g.

The Cauchy-Schwarz divergence was used as a classification loss function in Czarnecki et al. [20], Janocha and Czarnecki [47]. For neural network multiclass classifiers, Janocha and Czarnecki [47] view f and g as vectors of class probabilities, where f is the target probability vector that assigns a probability of 1 to the correct class and 0 to other classes, and g is the model output. As mentioned in Janocha and Czarnecki [47], the model output class probability vector g can be obtained by applying the sigmoid function $\sigma(z) = (1+e^{-z})^{-1}$ to every logit, or by applying the softmax normalisation function on the logits vector.

This gives rise to two separate loss functions, the softmax Cauchy-Schwarz divergence and the sigmoid Cauchy-Schwarz divergence. We analyse the alignment with the PoLS for each loss separately. The softmax Cauchy-Schwarz divergence is given by

$$\ell(z,y) = -\log \frac{p \cdot t}{||p||\,||t||},$$

where $t_y = 1$ and $t_j = 0$ for $j \neq y$, and p is the model output distribution given by the softmax function

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = \big(\sum_{j=1}^{k} e^{z_j - z_i}\big)^{-1}.$$

Since t is centred in the coordinate y, the loss term can be simplified to

$$\ell(z,y) = -\log \frac{p_y}{||p||}, \tag{3.4}$$

that is defined for every output probability distribution p, since the softmax function never assigns a probability of 0 to any class.

The softmax Cauchy-Schwarz divergence is not aligned with the PoLS. This can be seen using a counter-example similar to the one used for the cross-entropy loss. Let $S = ((x_1,1),(x_2,2))$ be the training sample, and let θ_α, for $\alpha > 0$, be a model such that $z^{\theta_\alpha}(x_1) = (2\alpha, \alpha)$, and $z^{\theta_\alpha}(x_2) = (-2\alpha, -\alpha)$. Further, from the definition of the softmax function, $p^{\theta_\alpha}(x_1) = p^{\theta_\alpha}(x_2)$ therefore also $\ell(z^{\theta_\alpha}(x_1),1) = \ell(z^{\theta_\alpha}(x_2),2)$. Since $\ell(S_{\theta_\alpha})$ converges to 0 when α converges to ∞, for every $\varepsilon > 0$ there exists $\alpha > 0$ such that $\ell(S_{\theta_\alpha}) \leq \varepsilon$, but $z_2^{\theta_\alpha}(x_1) > z_2^{\theta_\alpha}(x_2)$, contradicting an alignment with PoLS.

3.4.4 Sigmoid Cauchy-Schwarz divergence

When the model output class probability vector g is obtained by applying the sigmoid function $\sigma(z) = (1+e^{-z})^{-1}$ to the output logits, it gives rise to the sigmoid Cauchy-Schwarz divergence loss function.

The sigmoid Cauchy-Schwarz divergence is given by

$$\ell(z,y) = -\log \frac{\sigma(z) \cdot t}{||\sigma(z)||\,||t||},$$

where $\cdot$ denotes an inner product, t is the same as for the softmax Cauchy-Schwarz divergence, and σ is applied coordinate-wise. Since t is centred in the coordinate y, the loss term can be simplified to

$$\ell(z,y) = -\log \frac{\sigma(z_y)}{||\sigma(z)||}, \tag{3.5}$$

that is defined for every output logit vector z, single the sigmoid function does not assign a probability of 0 to any class.

The sigmoid Cauchy-Schwarz divergence is not aligned with the PoLS, since it depends only on the ratio between $\sigma(z_y)$ and σ of other coordinates in the logit vector. Formally, the following counter-example shows that this loss is not aligned with the PoLS. Let $S = ((x_1,1),(x_2,1))$ be the training sample, and let θ_α, for $\alpha > 0$, be a model such that $\sigma(z^{\theta_\alpha}(x_1)) = (0.5, 0.5/\alpha)$, and $z^{\theta_\alpha}(x_2) = (0.5/\alpha, 0.5/\alpha^2)$. It is easy to derive that $\ell(z^{\theta_\alpha}(x_1),1) = \ell(z^{\theta_\alpha}(x_2),1) = -\log \frac{\alpha}{\sqrt{\alpha^2+1}}$. As the last expression converges to 0 when α converges to ∞, for every $\varepsilon > 0$ there exists $\alpha > 0$ such that $\ell(S_{\theta_\alpha}) \leq \varepsilon$, but $z_2^{\theta_\alpha}(x_1) \not< z_1^{\theta_\alpha}(x_2)$, contradicting an alignment with PoLS.

3.4.5 Softmax Tanimoto loss

The Tanimoto loss was proposed in Janocha and Czarnecki [47], based on the Tanimoto distance that is a distance measure for binary vectors [34]. The Tanimoto loss is defined as

$$D_{Tan}(f,g) = -\frac{f \cdot g}{||f|| + ||g|| - f \cdot g}, \tag{3.6}$$

where $\cdot$ is an inner product and $||\cdot||$ is an L2 norm. Similarly to the Cauchy-Schwarz divergence, f and g are viewed as vectors of class probabilities, where f is the target probability vector that assigns a probability of 1 to the correct class and 0 to other classes, and g is the model output. Also for this loss, Janocha and Czarnecki [47] mention that the

model output class probability vector g can be obtained by using the softmax or the sigmoid functions, that gives rise to two different loss functions.

When using the softmax function to obtain class probabilities, since $f_y = 1$ and $f_j = 0$ otherwise, the Tanimoto loss amounts to

$$\ell(z,y) = -\frac{p_y}{1+||p||-p_y} = -\frac{1}{\frac{1+||p||}{p_y}-1}, \tag{3.7}$$

where p and p_y are as defined in Section 3.4.3. Since $||p|| \geq p_y$, it follows that $\frac{1+||p||}{p_y} \geq 2$ and $\frac{1+||p||}{p_y} = 2$ if and only if $p_y = 1$. Therefore, the minimal value of ℓ is -1, which is non-standard for loss functions that are normally defined as non-negative function. However, for gradient computation purposes it does not matter, and this loss can be used without any changes for training neural network classifiers.

The softmax Tanimoto loss is not aligned with the PoLS, again because the softmax function depends only on different between logits. Formally, consider a counter-example similar to the one used for the cross-entropy loss. Let $S = ((x_1,1),(x_2,2))$ be the training sample, and let θ_α, for $\alpha > 0$, be a model such that $z^{\theta_\alpha}(x_1) = (2\alpha,\alpha)$, and $z^{\theta_\alpha}(x_2) = (-2\alpha,-\alpha)$. Further, from the definition of the softmax function, $p^{\theta_\alpha}(x_1) = p^{\theta_\alpha}(x_2)$ therefore also $\ell(z^{\theta_\alpha}(x_1),1) = \ell(z^{\theta_\alpha}(x_2),2)$. Since $\ell(S_{\theta_\alpha})$ converges to -1 when α converges to ∞, for every $\varepsilon > -1$ there exists $\alpha > 0$ such that $\ell(S_{\theta_\alpha}) \leq \varepsilon$, but $z_2^{\theta_\alpha}(x_1) > z_1^{\theta_\alpha}(x_2)$, contradicting an alignment with PoLS. We find that the sigmoid Tanimoto loss is aligned with the PoLS, and we discuss this loss function in Section 3.5.4

3.5 Existing objectives that satisfy the PoLS

A few loss functions that were previously proposed in the literature are aligned with the PoLS. These loss functions are often used to solve problems that are somewhat related to the PoLS. This section explores these loss functions, contains proves that they are indeed aligned with the PoLS.

3.5.1 Self-normalisation

The self-normalisation loss [24] was introduced in the context of machine translation models. In many variations of these models a posterior probability over the entire vocabulary is calculated. As the vocabulary often contains hundreds of thousands of words, the self-normalisation loss was introduced in order to avoid this costly computation. The self-normalisation loss is composed of the standard cross-entropy loss and an additional term.

Let $\alpha > 0$ be a hyperparameter, and p_y as defined in Eq. 3.1. The self-normalisation loss is defined by

$$\ell(z,y) = -\log(p_y) + \alpha \cdot \log^2(\sum_{j=1}^{k} e^{z_j}). \tag{3.8}$$

The motivation for this loss is as follows: when the loss is minimised, the term $(\sum_{j=1}^{k} e^{z_j}$ equals 1, therefore the exponentiated logit e^{z_j} can be interpreted as class probabilities for class j. Devlin et al. [24] use this property to compute the posterior probability for only a subset of all possible classes, and report a speedup by a factor of 15.

Intuitively, this loss should be useful for the SLC task as well: if the softmax normalisation term is always close to 1 then there is no need to compute it, therefore it is enough to compute the (exponentiated) logit for the class of interest alone to determine the class's posterior probability, and infer whether this is the correct class. Indeed, Keren et al. [57] show that this loss is aligned with the PoLS. Minimising the first term in the loss requires the true logit to be as far as possible from false logits for a given example. Minimising the second term requires all exponentiated logits to sum to one. Therefore, when both terms are minimised, the true logit converges 0, while all false logits converge negative infinity. Doing this for all training examples results in all true logits being larger than all false logits in the entire training set.

Keren et al. [57] provide a formal proof that the self-normalisation loss is aligned with the PoLS. Let a training sample S and a neural network model θ, and consider an example $(x,y) \in S$. We consider the two terms of the loss in order. First, consider $-\log(p_y)$. From the definition of p_y (Eq. 3.1) we have that

$$-\log(p_y) = \log(\sum_{j=1}^{k} e^{z_j - z_y}) = \log(1 + \sum_{j \neq y} e^{z_j - z_y}).$$

Set $\varepsilon_0 := \log(1+e^{-2})$. Then, if $-\log(p_y) < \varepsilon_0$, we have $\sum_{j \neq y} e^{z_j - z_y} \leq e^{-2}$, which implies that (a) $\forall j \neq y, z_j \leq z_y - 2$ and (b) $e^{z_y} \geq \sum_{j=1}^{k} e^{z_j}/(1+e^{-2}) \geq \frac{1}{2}\sum_{j=1}^{k} e^{z_j}$. Second, consider the second term. There is an $\varepsilon_1 > 0$ such that if $\log^2(\sum_{j=1}^{k} e^{z_j}) < \varepsilon_1$ then (c) $2e^{-1} < \sum_{j=1}^{k} e^{z_j} < e$, which implies $e^{z_y} < e$ and hence (d) $z_y < 1$. Now, let θ such that $\ell(S_\theta) \leq \varepsilon := \min(\varepsilon_0, \varepsilon_1)$. Then $\forall (x,y) \in S$, $\ell(z^\theta(x), y) \leq \varepsilon$. From (b) and (c), $e^{-1} < \frac{1}{2}\sum_{j=1}^{k} e^{z_j} < e^{z^y}$, hence $z_y > -1$. Combining with (d), we get $-1 < z_y < 1$. Combined with (a), we get that for $j \neq y$, $z_j < -1$. To summarise, $\forall (x,y), (x',y') \in S$ and $\forall y'' \neq y'$, we have that $z_y^\theta(x) > -1 > z_{y''}^\theta(x')$, implying PoLS-alignment.

3.5.2 Noise Contrastive Estimation

Noise Contrastive Estimation (NCE) [38, 81] was proposed, similarly to self-normalisation, in the context of natural language processing. This mechanism was proposed in order to speed up training of neural language models over large vocabularies. In NCE, the multiclass problem is viewed as a collection of binary classification problems, each for a single class. Each binary classification problem classifies whether a given word is from the data distribution or a noise distribution. At training time, only t words from the noise distribution are used, instead of the entire vocabulary, which leads to a significant speedup. Similarly to self-normalisation, NCE, is known to produce a self-normalised logit vector [3]. This property makes the NCE loss a good fit for the SLC task, as a single logit is may be meaningful as a standalone value and not only in comparison to other logits from the same example.

In the version given by Mnih and Teh [81], the NCE loss is defined based on a distribution over the possible classes, denoted by $q = (q(1), \ldots, q(k))$, where $\sum_{i=1}^{k} q(i) = 1$. The NCE loss, in an equivalent notation given by Keren et al. [57], is

$$\ell(z, y) = -\log g_y - t \cdot \mathbb{E}_{j \sim q} \left[\log(1 - g_j) \right], \tag{3.9}$$

where

$$g_j := (1 + t \cdot q(j) \cdot e^{-z_j})^{-1}.$$

During training, the second term in the loss is usually approximated by a Monte-Carlo approximation, using t random samples of $j \sim q$, to speed up training time [81].

Keren et al. [57] observe that the NCE loss is aligned with the PoLS. They first observe that g_j is of a similar form to $\sigma(z_j)$ where $\sigma(z) = (1 + e^{-z})^{-1}$ is the sigmoid function. Therefore, when the above term is minimised for a single example, the true logit converges to infinity while all false logits converge to negative infinity. When this term is minimised for the entire training set, all true logits are larger than all false logits across the training set. The formal proof from Keren et al. [57] is given below.

We prove that the NCE loss satisfies the PoLS: g_j is monotonic increasing in z_j. Hence, if the loss is small, g_y is large and g_j for $j \neq y$, is small. Formally, fix t, and let a training sample S. There is an $\varepsilon_0 > 0$ such that if $-\log g_j \leq \varepsilon_0$, then $z_j > 0$. Also, there is an $\varepsilon_1 > 0$ (which depends on q) such that if $-\mathbb{E}_{j \sim q} \left[\log(1 - g_j) \right] \leq \varepsilon_1$ then $\forall j \neq y$, $\log(1 - g_j)$ must be small enough so that $z_j < 0$. Now, consider θ such that $\ell(S_\theta) \leq \varepsilon := \min(\varepsilon_0, \varepsilon_1)$. Then for every $(x, y) \in S$, $\ell(z^\theta(x), y) \leq \varepsilon$. This implies that for every $(x, y), (x', y') \in S$ and $y'' \neq y'$, we have that $z^\theta_y(x) > 0 > z^\theta_{y''}(x')$, thus this loss is aligned with the PoLS.

3.5.3 Binary cross-entropy

The binary cross-entropy loss is often used in multilabel classification setting. In this setting, each example can belong to multiple classes, and the goal of learning is to classify a given example into all classes it belongs to. The common approach for this task [45, 121] is solving a set of binary classification problems, each determines whether the given example belongs to a specific class, using a single neural network model. This is done by minimising the sum of cross-entropy losses that correspond to the binary problems. In this setting, the label of each example is a binary vector $(r_1,\dots,r_k)$, where $r_j = 1$ if x belongs to class j and 0 otherwise. The loss for a single training example with logits z and label-vector r is

$$\ell(z,(r_1,\dots,r_k)) = -\sum_{j=1}^{n} r_j \log(\sigma(z_j)) + (1-r_j)\log(1-\sigma(z_j)),$$

where $\sigma(z) = (1+e^{-z})^{-1}$ is the sigmoid function. This loss can also be used for our setting of multiclass problems, by defining $r_j := \mathbf{1}_{j=y}$ for an example (x,y). This gives the multiclass loss

$$\ell(z,y) = -\log(\sigma(z_y)) + \sum_{j\neq y} \log(1-\sigma(z_j)). \tag{3.10}$$

As pointed by Keren et al. [57], the binary cross-entropy, when applied to the multiclass setting, is also aligned with the PoLS. Indeed, similarly to the NCE loss, when the binary cross-entropy is minimised for a single example, the true logit z_y converges to infinity, while all false logits converge to negative infinity. Minimising this loss across the entire trainings set results in all true logits having larger values than all false logits across the training set.

Keren et al. [57] also provide a formal proof for this: for g_j as in Eq. 3.9, $g_j = \sigma(z_j - \ln(t\cdot q(j)))$. Since $\sigma(z_j)$ is monotonic, the proof method for NCE carries over and thus the binary cross-entropy loss satisfies the PoLS as well.

3.5.4 Sigmoid Tanimoto loss

When using the sigmoid function to obtain class probabilities, the Tanimoto loss from Eq. 3.6 amounts to

$$\ell(z,y) = -\frac{\sigma(z_y)}{1+||\sigma(z)|| - \sigma(z_y)} = -\frac{1}{\frac{1+||\sigma(z)||}{\sigma(z_y)} - 1}, \tag{3.11}$$

since $f_y = 1$ and $f_j = 0$ for $j \neq y$. Since $||\sigma(z)|| \geq \sigma(z_y)$, it follows that $\frac{1+||\sigma(z)||}{\sigma(z_y)} \geq 2$ and $\frac{1+||\sigma(z)||}{\sigma(z_y)} = 2$ if and only if $\sigma(z_y) = 1$ and $\sigma(z_j) = 0$ for $j \neq y$. Therefore, similarly to the

softmax Tanimoto loss, the minimal value of ℓ is -1, which is non-standard but has no practical implications for neural network training.

This loss function is aligned with the PoLS. From the above discussion, if ℓ converges to its minimal value of -1 then $\sigma(z_y)$ converges to 1 and $\sigma(z_j)$ converges to 0 for $j \neq y$. This implies that when ℓ converges to its minimum, z_y converges to ∞ while z_j converges to $-\infty$ for $j \neq y$. Now consider a training sample S and a model θ. It follows that there exists $\varepsilon > 0$ such that if $\ell(S_\theta) < -1 + \varepsilon$ then for every $(x,y),(x',y') \in S$ and $y'' \neq y'$, we have that $z^\theta_y(x) > 0 > z^\theta_{y''}(x')$, thus this loss is aligned with the PoLS.

3.6 Novel objectives that satisfy the PoLS

In this section we describe the two novel loss functions proposed by Keren et al. [57]. The two novel training objectives are aligned with the PoLS, therefore are suitable for the SLC task. These loss functions are extensions of the cross-entropy and the max-margin losses that were studied in Section 3.4, adapted to operate on a *batch* of training examples, instead of a single example. In addition, the new loss functions do not add any new hyperparameter to be tuned, as the batch size is already a hyperparameter of gradient-based optimisation algorithms such as SGD.

When the cross-entropy and the max-margin losses are minimised, they guarantee a large difference between the true and false logits for every example separately. The generalisations proposed in Keren et al. [57] transform this property to apply on a batch of examples, which guarantees the difference between true and false logits across a batch. Finally, shuffling the training batches during the training process enforces a difference between true and false logits across the entire training set, which is equivalent to alignment with the PoLS.

3.6.1 Batch cross-entropy

The first novel loss proposed in Keren et al. [57] generalises the cross-entropy loss. The cross-entropy loss can be written as the Kullback-Leibler (KL) divergence between a target and a model output distributions, as described below. The KL divergence between two discrete probability distributions P and Q over $[k]$ is defined as

$$\mathrm{KL}(P||Q) := \sum_{i=j}^{k} P(j)\log(P(j)/Q(j)).$$

For an example (x,y), let $P_{(x,y)}$ be the distribution over $[k]$ which deterministically outputs y, and let Q_x be the distribution defined by the softmax normalised logits, $Q_x(j) = e^{z_j}/\sum_{i=1}^{k} e^{z_i}$.

Then, it is easy to see that for p_y as defined in Eq. 3.1, $\mathrm{KL}(P_{(x,y)}||Q_x) = -\log p_y$, is exactly the cross-entropy loss as given in Eq. 3.1.

The batch version of the cross-entropy loss was defined in Keren et al. [57] using the KL divergence between distributions over batches of examples as follows. Denote the i-th example in a batch by (x_i, y_i), and define P_B to be the target distribution over $[m] \times [k]$:

$$P_B(i,j) := \begin{cases} \frac{1}{m} & j = y_i, \\ 0 & \text{otherwise.} \end{cases}$$

Let Q_B be the distribution defined by the softmax normalised logits over the entire batch B. Formally, denote $Z(B) := \sum_{i=1}^{m} \sum_{j=1}^{k} e^{z_j(x_i)}$. Then $Q_B(i,j) := e^{z_j(x_i)}/Z(B)$. We then define the batch cross-entropy loss as follows.

Definition 2 (The batch cross-entropy loss)**.** *Let $m > 1$ be an integer, and let B be a uniformly random batch of size m from S. The* batch cross-entropy loss *of a training sample S is*

$$\ell(S) := \mathbb{E}_B[L_c(B)], \quad \textit{where} \quad L_c(B) := \mathrm{KL}(P_B||Q_B).$$

The batch cross-entropy loss is aligned with the PoLS. When the loss is minimised, each exponentiated true logit converges to $\frac{1}{m}$, therefore each true logit converges to $\log \frac{1}{m}$. On the other hand, each exponentiated false logit converges to 0, therefore each false logit converges to negative infinity. When this loss is minimised over the training set, all true logits are larger than all false logits in the training set, therefore the loss is aligned with the PoLS. Keren et al. [57] give a formal proof for this, using an additional lemma, as given below.

If true logits are greater than false logits in every batch separately when using, then the PoLS is satisfied on the whole sample, since every pair of examples appears together in some batch. The following lemma formalises this:

Lemma 3. *If L is aligned with the PoLS, and ℓ is defined by $\ell(S_\theta) := \mathbb{E}_B[L(B_\theta)]$, then ℓ is also aligned with the PoLS.*

Proof. Assume a training sample S and a neural network model θ. Since L is aligned with the PoLS, there is some $\varepsilon' > 0$ such if $L(B_\theta) < \varepsilon'$, then for each $(x,y),(x',y') \in B$ and $y'' \neq y'$ we have that $z^\theta_y(x) > z^\theta_{y''}(x')$. Let $\varepsilon = \varepsilon' / \binom{n}{m}$, and assume $\ell(S_\theta) < \varepsilon$. Since there are $\binom{n}{m}$ batches of size m in S, this implies that for every batch B of size m, $L(B_\theta) \leq \varepsilon'$. For any $(x,y),(x',y') \in S$, there is a batch B that includes both examples. Thus, for $y'' \neq y'$, $z^\theta_y(x) > z^\theta_{y''}(x')$. Since this holds for any two examples in S, ℓ is also PoLS-aligned. □

To show that the batch cross-entropy satisfies the PoLS, it is needed to show that L_c does, which by Lemma 3 implies this for ℓ. By the continuity of KL, and since for discrete distributions, $\mathrm{KL}(P||Q) = 0 \iff P \equiv Q$, there is an $\varepsilon > 0$ such that if $L(B_\theta) \equiv \mathrm{KL}(P_B||Q_B^\theta)] < \varepsilon$, then for all i, j, $|P_B(i,j) - Q_B^\theta(i,j)| \leq \frac{1}{2m}$. Therefore, for each example $(x,y) \in B$,

$$\frac{e^{z_y^\theta(x)}}{Z(B)} > \frac{1}{2m}, \qquad \text{and} \qquad \forall j \neq y, \quad \frac{e^{z_j^\theta(x)}}{Z(B)} < \frac{1}{2m}.$$

It follows that for any two examples $(x,y),(x',y') \in B$, if $y \neq y'$, then $z_y^\theta(x) > \frac{1}{2m} > z_{y'}^\theta(x')$. Therefore L satisfies the PoLS, which completes the proof.

3.6.2 Batch max-margin

A second novel loss function that was proposed in Keren et al. [57] is the batch version of the max-margin loss that was defined in Eq. 3.2. For a batch B, the minimal true logit and maximal false logit in B are defined:

$$z_+^B := \min_{(x,y)\in B} z_y(x), \quad \text{and} \quad z_-^B := \max_{(x,y)\in B, j\neq y} z_j(x).$$

The batch max-margin loss was defined in Keren et al. [57] as follows:

Definition 3 (The batch max-margin loss)**.** *Let $m > 1$ be an integer, and let B be a uniformly random batch of size m from S. Let ℓ be the single-example max-margin loss defined in Eq. 3.2, let $\gamma > 0$ be the max-margin hyper-parameter. The* batch max-margin *is defined by*

$$\ell(S) := \mathbb{E}_B[L_m(B)],$$

where

$$L_m(B) := \frac{1}{m}\max(0, \gamma - z_+^B + z_-^B) + \frac{1}{m}\sum_{(x,y)\in B} \ell(z(x),y).$$

The batch max-margin loss is aligned with the PoLS. Minimising the first term in $L_m(B)$ makes sure that the smallest true logit in a given batch is greater than the largest false logit in this batch. Shuffling the batches during training makes this property hold for the entire training set, which conforms with the definition of the PoLS. The second term of $L_m(B)$ is not necessary for alignment with the PoLS, but was rather added to increase convergence speed. Indeed, without this term at every training iteration only two logits are affected: the smallest true logit and the largest false logit, which has the potential to slow down training by requiring many training iterations. This intuition was confirmed in initial experiments as reported by Keren et al. [57].

We include the formal proof of the alignment of this loss function with the PoLS, as was given in Keren et al. [57]: to show that the batch max-margin loss satisfies the PoLS, we show this for L_m and invoke Lemma 3. Set $\varepsilon = \gamma/m$. If $L(B_\theta) < \varepsilon$, then $\gamma - z_+^B + z_-^B < \gamma$, implying $z_+^B > z_-^B$. Hence, any $(x,y),(x',y') \in B$ such that $y \neq y'$ satisfy $z_y^\theta(x) \geq z_+^B > z_-^B \geq z_y^\theta(x')$. Thus L is aligned with the PoLS, implying the same for ℓ.

3.7 Experiments

The first results in this section show that PoLS plays a dominant role in obtaining high accuracy in the SLC task. Specifically, it is shown that all loss functions that are aligned with the PoLS yield better accuracy in the SLC task than all losses that are not aligned with the PoLS. Then, SLC accuracy is compared to the computationally expensive case, in which all logits are computed and used for determining a given class's correctness. Keren et al. [57] find that when using a PoLS-aligned loss, a single logit used in the SLC task is predictive of class correctness at a level comparable to this of a normalised logit that is obtained using a softmax normalisation that requires computing all logits. Lastly, the speedups obtained by computing a single logit only are reported, for different neural network architectures. For instance, a speedup of x8-x21 is obtained when the number of classes is as high as $400,000$.

3.7.1 PoLS and SLC accuracy

Keren et al. [57] experimented with the SLC task with neural networks that were trained using seven of the considered training objectives. Since the Cauchy-Schwarz divergence and the Tanimoto losses are practically never used in practice for training neural network classifiers, we do not include those objectives in our experiments and only include the analysis of their PoLS alignment. To evaluate the success of a learnt model in the SLC task, Keren et al. [57] measured, for each class j and each threshold T, the precision and recall in identifying examples from class j using the test $z_j > T$, and calculated the Area Under the Precision-Recall curve (AUPRC) defined by the entire range of possible thresholds. In addition, Keren et al. [57] also measured the precision at fixed recall values (with dictate the threshold T to use) 0.9 (Precision@0.9) and 0.99 (Precision@0.99). The averages of those values across all classes in a given dataset are reported.

Five computer vision datasets are used for those evaluations, using their built-in train/test splits: MNIST [72], SVHN [85] CIFAR-10 and CIFAR-100 [64]. The fifth dataset is the Imagenet dataset [99], which has 1000 classes, demonstrating the scalability of the PoLS approach to many classes. The Imagenet dataset is highly computationally intensive due

Table 3.1 Results on Single Logit classification (SLC), using the different loss functions, as was reported in Keren et al. [57]. 'Prec'@x' is the precision at a given recall value. 'Improvement' denotes the relative improvement of the mean score for PoLS methods, over the mean score for other methods. Lower values are better. In almost all cases, loss functions that are aligned with the PoLS (under the dashed line) yield a mean relative improvement of at least 20% in SLC accuracy measures, and sometimes considerably more.

Dataset	Method	1-AUPRC	1-Prec'@0.9	1-Prec'@0.99
MNIST	CE	0.008	0.005	0.203
	max-margin	0.012	0.018	0.262
	self-norm	0.002	0.001	**0.021**
	NCE	0.002	0.002	**0.021**
	binary CE	0.002	**0.000**	0.037
	batch CE	**0.001**	0.001	0.022
	batch max-margin	0.002	0.001	0.034
	Improvement	82.0%	91.3%	88.4%
SVHN	CE	0.023	0.028	0.545
	max-margin	0.021	0.025	0.532
	self-norm	**0.015**	0.014	0.298
	NCE	0.021	0.017	0.320
	binary CE	**0.015**	0.016	0.312
	batch CE	**0.015**	**0.013**	**0.280**
	batch max-margin	0.018	0.020	0.384
	Improvement	23.4%	39.6%	40.8%
CIFAR-10	CE	0.109	0.326	0.703
	max-margin	0.094	0.285	0.705
	self-norm	0.073	0.204	0.599
	NCE	0.081	0.214	**0.594**
	binary CE	**0.070**	0.210	0.607
	batch CE	0.072	**0.202**	0.602
	batch max-margin	0.075	0.226	0.636
	Improvement	26.9%	30.9%	13.6%
CIFAR-100	CE	0.484	0.866	0.974
	max-margin	0.490	0.893	0.977
	self-norm	0.378	0.807	0.970
	NCE	0.383	**0.795**	0.964
	binary CE	0.426	0.870	0.978
	batch CE	**0.371**	**0.795**	**0.961**
	batch max-margin	0.468	0.903	0.983
	Improvement	16.8%	5.2%	0.5%
Imagenet (1000 classes) ($6 \cdot 10^6$ iterations)	CE	0.366	0.739	0.932
	batch CE	**0.245**	**0.563**	**0.865**
	Improvement	33.1%	23.8%	7.2%

to its large size, therefore Keren et al. [57] experiment with only two representative loss functions with this dataset, one that is aligned with the PoLS and one that is not aligned with it. For every dataset, a fixed network architecture is used for experiments with all loss functions.

Standard network architectures were used, that were fixed before conducting the experiments. For the MNIST dataset, an MLP network is used, comprised of two layers with 500 units each, and an output layer with 10 units, one for each class, whose outputs are the logits. For the SVHN, CIFAR-10 and CIFAR-100 datasets, a convolutional neural network [71] is used, with 6 convolutional layers and one dense layer with 1024 units. The first, third and fifth convolutional layers used a 5×5 kernel, where other convolutional layers used a 1×1 kernel. The first two convolutional layers were comprised of 128 feature maps, where convolutional layers three and four had 256 feature maps, and convolutional layers five and six had 512 feature maps. Max-pooling layers with 3×3 kernel size and a 2×2 stride were applied after the second, fourth and sixth convolutional layers. In all networks, batch normalisation [46] was applied to the output of every fully-connected or convolutional layer, followed by a rectified-linear non-linearity. For every combination of a training objective and a dataset (with its fixed network architecture), the best learning rate among $1, 0.1, 0.01, 0.001$ was optimised for by using the classification accuracy on the validation set. For the Imagenet experiments, an Inception-V3 network [111] was used, as it appears in the `tensorflow` [1] repository, using all default hyperparameters from this implementation, only changing the loss function used. The Inception model was trained for 10^6 iterations for each of the loss functions used.

The experiment results from Keren et al. [57] are reported in Table 3.1. As many of the reported metrics are close to their maximal value of 1, the values of one minus each metric are reported to better visualise the relative improvement, making lower values better. The results above the dashed line for each dataset correspond to loss functions that are not aligned with the PoLS, where the results below the dashed line correspond to loss function that are aligned with PoLS. Best results for each combination of metric and dataset are indicated in boldface. The bottom row for each dataset indicate the mean relative improvement, i.e., the relative improvement of the mean of each metric across all PoLS-aligned loss functions, over the mean of each metric across all loss functions that are not aligned with the PoLS.

The results in Table 3.1 indicate that for all evaluation metrics, the mean relative improvement for PoLS-aligned losses is usually at least 20%, and in many cases considerably more. This concludes that, aligned with the hypothesis of Keren et al. [57], alignment with the PoLS is a crucial ingredient for success in SLC task.

3.7.2 SLC vs computing all logits

This section investigates whether SLC causes performance degradation in binary classification. Specifically, accuracy in SLC is compared to the computationally expensive case in which a model is trained using the cross-entropy loss, then all logits are computed and used for softmax normalisation. The normalised logit is used for binary classification. Experiment setting and reported metrics are identical to those used in Section 3.7.1.

Table 3.2 Comparing binary classification with a single logit (SLC) vs. all logits as was reported in Keren et al. [57]. Lower values are better. 'Prec'@x' is the precision at a given recall value. PoLS-aligned SLC methods are above the dashed line. Results are comparable, thus SLC does not cause any degradation in binary classification accuracy, compared to the case where all logits are computed.

Dataset	Method	1-AUPRC	1-Prec'@0.9	1-Prec'@0.99
MNIST	Mean PoLS methods	0.002	0.001	0.027
	batch CE	0.001	0.001	0.022
	CE with all logits	0.001	0.000	0.020
SVHN	Mean PoLS methods	0.017	0.016	0.319
	batch CE	0.015	0.013	0.280
	CE with all logits	0.015	0.016	0.313
CIFAR-10	Mean PoLS methods	0.074	0.211	0.608
	batch CE	0.072	0.202	0.602
	CE with all logits	0.074	0.214	0.648
CIFAR-100	Mean PoLS methods	0.405	0.834	0.971
	batch CE	0.371	0.795	0.961
	CE with all logits	0.380	0.801	0.973
Imagenet	batch CE	0.245	0.563	0.865
	CE with all logits	0.223	0.566	0.872

Results from Keren et al. [57] are presented in Table 3.2. The two rows above the dashed line contain the results for the SLC task, for the mean of all losses that are aligned with the PoLS, and for the batch CE loss, that performed best of the experiments in Section 3.7.1. The row below the dashed line contains the results for the computationally expensive case, in which all cross-entropy logits are computed and used for softmax normalisation. The results

show that the cross-entropy with all logits case are comparable to those using a single logit alone, and specifically to the case where the batch CE loss is used. This shows that even though computing all logits and using them for normalisation is computationally expensive, it still does not yield any considerable benefit in determining whether a given class is the correct one for a given example. Therefore, we conclude that in our setting of binary classification with multiple classes at test time, SLC, and specifically batch cross-entropy, is an attractive alternative to standard cross-entropy with all logits.

3.7.3 SLC speedups

This section evaluates the speedups gained by performing SLC instead of computing all class logits. Keren et al. [57] used five prominent image classification architectures (Alexnet [65], VGG-16 [103], Inception-V3 [111], Resnet-50 and Resnet-101 [42]). As the goal is comparing test time computation speed, only the forward-pass time is measured for each given network. To measure SLC computation time, the top layer is replaced by a layer with single unit, and the time to compute the single logit given an input to the network is measured. To measure the computation time when computing the logits of k classes, the top layer is replaced with a layer containing k units, and the time it takes to compute all logits is again measured, given an input to the network.

The computation time of a given network generally does not depend on the network's input or task accuracy. Specifically, the measured forward-pass time is approximately identical for inputs that originates from natural images datasets and random inputs of the same dimensionality. Therefore, in the experiments presented below Keren et al. [57] use randomly initialised networks without training and random noise as input to those networks. Computation is done using `Tensorflow` and a single NVIDIA Maxwell Titan-X GPU, and forward-pass computation time per example is averaged across 100 minibatches of 32 examples. The public implementation of all architectures is used, as appears in the `tensorflow` repository.

The timing results appear in Table 3.3. For each network architecture, the first row reports the forward-pass time for computing a single logit, while the other rows report the forward-pass time for computing all logits for different number of classes. The raw timings are reported, as well as the speedup factor that is obtained by computing a single logit alone, that is the ratio between the raw timing results. Naturally, the speedup increases with the number of classes.

For networks with up to $2^{14} = 16384$ classes, the speedup is relatively small, since computation of the network layers dominates the overall forward-pass computation time. In contrast, when there are many classes, the computation of logits dominates the forward-pass

Table 3.3 Speedup experiment results as appear in Keren et al. [57]. When the number of examples is large, SLC results in a considerable speedup.

Architecture	Classes	Inference Time [s]	SLC Speedup
Alexnet	1 (SLC)	$3.6 \cdot 10^{-3}$	—
	2^{10}	$3.7 \cdot 10^{-3}$	x1.04
	2^{14}	$4.0 \cdot 10^{-3}$	x1.14
	2^{16}	$5.7 \cdot 10^{-3}$	x1.59
	2^{18}	$20.2 \cdot 10^{-3}$	x5.68
	$2^{18.5}$	$76.0 \cdot 10^{-3}$	x21.38
VGG-16	1 (SLC)	$9.4 \cdot 10^{-3}$	—
	2^{10}	$9.6 \cdot 10^{-3}$	x1.02
	2^{14}	$9.9 \cdot 10^{-3}$	x1.05
	2^{16}	$11.4 \cdot 10^{-3}$	x1.20
	2^{18}	$26.4 \cdot 10^{-3}$	x2.79
	$2^{18.5}$	$80.5 \cdot 10^{-3}$	x8.52
Inception-V3	1 (SLC)	$6.0 \cdot 10^{-3}$	—
	2^{10}	$6.2 \cdot 10^{-3}$	x1.03
	2^{14}	$6.5 \cdot 10^{-3}$	x1.09
	2^{16}	$7.3 \cdot 10^{-3}$	x1.22
	2^{18}	$18.7 \cdot 10^{-3}$	x3.11
	$2^{18.5}$	$76.6 \cdot 10^{-3}$	x12.75
Resnet-50	1 (SLC)	$6.1 \cdot 10^{-3}$	—
	2^{10}	$6.4 \cdot 10^{-3}$	x1.04
	2^{14}	$6.6 \cdot 10^{-3}$	x1.08
	2^{16}	$7.4 \cdot 10^{-3}$	x1.20
	2^{18}	$19.1 \cdot 10^{-3}$	x3.11
	$2^{18.5}$	$78.0 \cdot 10^{-3}$	x12.69
Resnet-101	1 (SLC)	$8.0 \cdot 10^{-3}$	—
	2^{10}	$8.2 \cdot 10^{-3}$	x1.03
	2^{14}	$8.3 \cdot 10^{-3}$	x1.04
	2^{16}	$9.4 \cdot 10^{-3}$	x1.19
	2^{18}	$23.5 \cdot 10^{-3}$	x2.95
	$2^{18.5}$	$80.4 \cdot 10^{-3}$	x10.10

computation time. Hence, SLC obtains a x2.8-x.5.7 speedup for $2^{18} = 262144$ classes, and x8.5-x21.3 speedup for $2^{18.5} = 370727$ classes.

The experiments in Sections 3.7.1 and 3.7.2, show that the findings scale well from 10 to 100 and 1000 classes, and are expected to scale further to models with a larger number of classes. Ideally, one would have directly tested datasets with hundreds of thousands of classes, to show that those results scale well to datasets with many more classes, where the speedup gains are large. However, since such datasets are very large (for instance, the Imagenet-21K dataset has 21,000 classes and more than 14 million examples), these experiments were infeasible with our computational resources. In comparison, a single Inception-V3 model for the significantly smaller Imagenet dataset ($\sim$ 1 million examples), took Keren et al. [57] approximately three weeks to train on a single GPU.

Chapter 4

Weakly Supervised One-Shot Detection

4.1 Motivation

State-of-the-art performance of object detection models has rapidly improved over the past few years. Among other factors, recent improvements are due to the rapidly growing sizes of available datasets for those tasks, normally containing a large number of labelled examples from a limited number of classes [22, 75]. In contrast to the limited number of classes normally available in existing detection datasets, humans are required in daily life to correctly identify and localise a much larger number of classes.

In attempt to overcome the dependency on large amount of data for a large number of classes, one-shot learning models attempt to generalise from a single labelled example to other members of this example's class. Recently, some works have demonstrated empirical success in one-shot classification, using models that are not conditioned on class identities [63, 119]. Such a model is given at every training iteration a target example and a small number of exemplars from N random classes. These models are then trained to identify which of the N exemplars is of the same class as the target example. Class identities are not used in this process, but rather only labels indicating which of the N exemplars correspond to the same class of the target example. At evaluation time, both the exemplars and target example belong to classes unseen at training time. Since these models learn to identify class similarity between the exemplars and the target example, classification output can be emitted for unseen classes without any additional training with those classes.

The above described training process facilitates generalisation in class space, from classes seen at training time to unseen classes. Furthermore, a design that is independent of class identities may also improve performance in the traditional classification setting, as transfer between classes may be facilitated. This chapter further explores and establishes the

paradigm of models that are not conditioned on class identities, with the aim of simultaneously classifying and localising classes that were not seen at training time.

Normally, for neural network models, generalisation to unseen examples of a given class may require a large number of labelled examples from this class to appear at training time. Training a model with no conditioning on class identities may induce generalisation in class space to unseen classes, but equivalently, may require a large number of classes to be present at training time. Indeed, previous works reporting empirical success with this paradigm also used a large number of classes at training time [55, 63, 119]. Hence, applying this approach for a one-shot localisation task may require a large dataset with localisation labels for a very large number of classes.

In most cases, corpora that contain a bounding box information are smaller than ones containing only instance-level labels, as the former are in general harder or more expensive to collect. Even in domains where corpora with bounding box information may be considered large, such as computer vision, equivalent corpora containing only instance-level labels are larger, and one may aspire to use those larger corpora to improve task performance. Therefore, instead of relying on a large enough corpus containing bounding box information, this chapter considers the detection problem in the low-quality supervision setting, that is discussed in Section 1.3. In this setting, the problem becomes a weakly supervised one-shot detection problem, where one-shot detection is learnt using corpora containing only weaker, instance-level labels. For the same reasons, previous work attempted to perform weakly supervised object detection (non-one-shot) using image-level labels [9, 113].

To overcome the lack of localisation labels, one may need to introduce additional structure to the model. For this purpose, context-based attention models have previously demonstrated the ability to learn to focus on relevant parts of a given input. This ability was learnt in a weakly supervised manner without using any labels indicating the part of the input the model should focus on [5, 17, 128].

This chapter presents the work of Keren et al. [59] on the weakly supervised one-shot detection task, as motivated by the research question posed in Section 1.3, by further exploring and establishing models with no conditioning on class identities. The weakly supervised one-shot detection task was considerably less explored compared to its fully-supervised counterparts. The model presented by Keren et al. [59] takes a single example of a given class, which is named the *exemplar*, and a larger *target example* which may or may not contain an instance of the same class as the exemplar. Keren et al. [59] use the framework of Siamese neural networks [10, 63] and apply it in a convolutional manner to identify regions in the target example that are similar to the exemplar in a higher-level representation space. Motivated by the above, the use of localisation labels, e.g., bounding boxes was replaced

by a novel attention mechanism, and only binary labels are used, that indicate whether the target example contains an instance of the same class as the exemplar. To the best of our knowledge, the work in Keren et al. [59] is the first time a Siamese network is used in tandem with an attention mechanism to learn similarity between different object parts in a weakly supervised manner.

In experiments, the model proposed by Keren et al. [59] was used for detection tasks in the audio and computer vision domains. In the audio domain, the detection task is the query-by-example spoken term detection task [14, 40, 89], where the model needs to identify whether a given spoken word appears in a given utterance, and localise this appearance in time. In the computer vision domain, Keren et al. [59] experiment with a character detection task using the Omniglot dataset [68]. In both domains, experiment result show that the proposed model was able to identify and localise instances of classes unseen at training time, and to outperform the appropriate baseline models.

4.2 Related work

Related work that is not otherwise mentioned is discussed below. Other approaches for one-shot learning appear in the literature, including the work presented in Hariharan and Girshick [39]. In this work, a few-shot recognition task is performed by a learner that tunes its feature representation on a set of base classes that have many training instances, to better perform on classes with a small number of examples. For the one-shot object detection task, a model for (fully-supervised) few-shot object detection was presented in Dong et al. [26]. This model is comprised of a pipeline that utilises a large unlabelled dataset for finding additional training examples.

Previous work also considers models for weakly supervised object detection. In Wang et al. [120], a pretrained convolutional neural network (CNN) was used to describe image regions and then learn object categories as corresponding visual topics. In Teh et al. [113], a model was introduced that computes an attention score for every location in an image. Then, one feature vector describing an image is constructed by combining the location scores, and is used for classifying the image. Localisation is done using the attention weights. Similarly, Bilen and Vedaldi [9] start from a pretrained CNN for image classification on a large dataset, then computes scores for each class at each location. These scores are combined into a single image level score, and the network is optimised for the classification task. Detection is again performed according to the class location scores. However, all of the above approaches depend on a predefined set of of classes, and are not suited for one-shot detection. In the audio domain, a related model for query-by-example spoken term detection was proposed

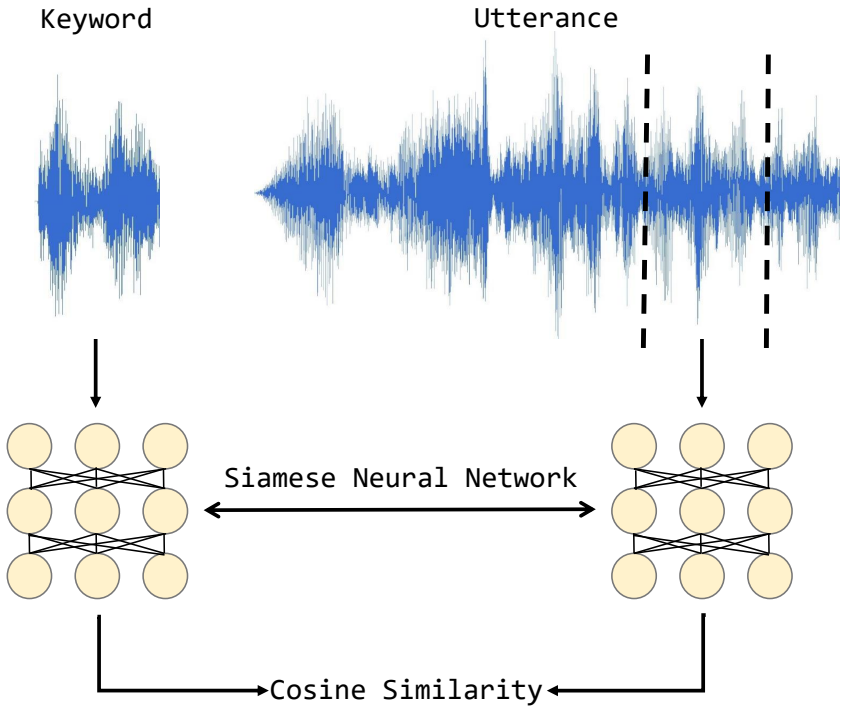


Fig. 4.1 The computation of a similarity score between an exemplar and a target location, for the case of detecting audio keywords in longer utterances. Both the exemplar (keyword) and the target location (part of the utterance) are mapped into a representation space using a Siamese neural network, and a cosine similarity between the representations is calculated. Reprinted with permission from Keren et al. [59].

[4]. This model uses an long short-term memory (LSTM) network for scoring the existence of a keyword in different utterance locations. These scores are combined for a single score for the utterance, that is used for training on a binary classification task.

4.3 Method

This section describes the weakly supervised one-shot detection model presented in Keren et al. [59], that is named Attention Similarity Networks (ASN). The model takes an *exemplar* x, that is a single instance of a class of interest, and a *target example* B, which may or may not contain an instance of the same class as the exemplar. Normally, the target example is spatially larger than the exemplar, and contains different locations, each could contain an instance of the same class as the exemplar. The different locations in the target example are called *target locations* by Keren et al. [59]. The model outputs a *similarity map* $s(x,B)$, such

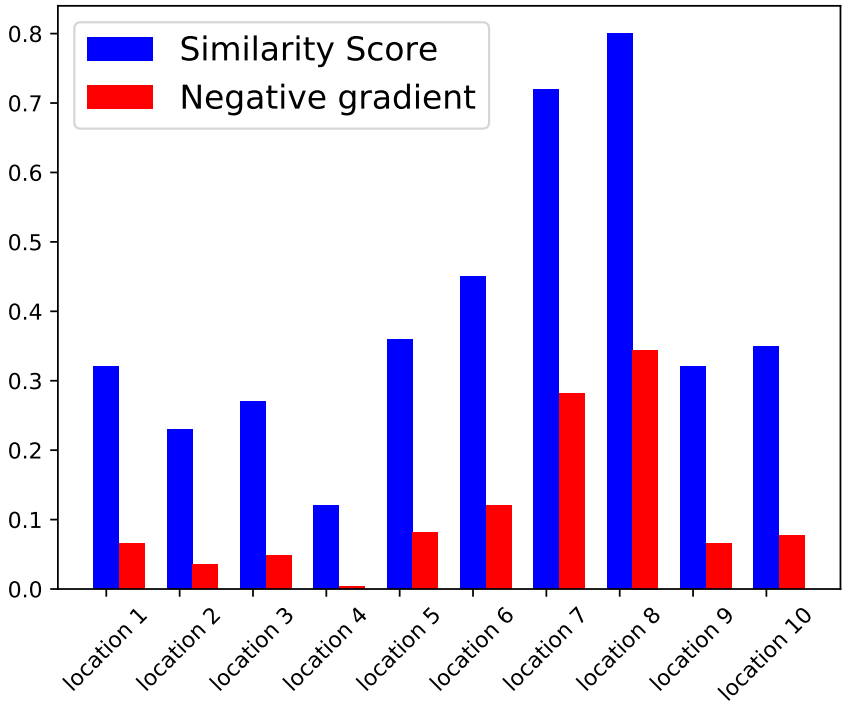


Fig. 4.2 The reason for the self-reinforcing loop: example relation between similarity scores s_l and the negative gradients $-\frac{\partial \ell}{\partial s_l}$, in a positive exemplar-target pair with ten different locations, computed according to Eq. 4.4 for an example vector of similarity scores. The negative gradient is increasing with the similarity scores. As a result, locations with high similarity scores will get even higher similarity scores during training, compared to other locations. Reprinted with permission from Keren et al. [59].

that $s_l(x,B)$ is a quantitative measure of the similarity between the exemplar and location l in the target example. More specifically, $s_l(x,B)$ is an estimation of the degree to which location l contains an instance of the same class as the exemplar. For simplicity of notation, s_l is written instead of $s_l(x,B)$ when the context is clear.

4.3.1 Similarity scores

As mentioned above, the similarity score s_l should contain the degree to which location l in the target example contains an instance of the same class as the exemplar. As class similarity is a higher-level measure of similarity, Keren et al. [59] do not use distance in raw pixel space to compute s_l, but rather the distance between higher-level representations of both the exemplar and the location l in the target example. To this end, both the exemplar and

location l are mapped using a neural network to a latent representation space, and similarity or dissimilarity in this space are optimised for.

s_l is modelled using the cosine similarity between outputs of a Siamese neural network [10, 63]:

$$s_l = \frac{f_\theta(x) \cdot f_\theta(B_l)}{\|f_\theta(x)\| \|f_\theta(B_l)\|}, \tag{4.1}$$

where B_l is location l in the target example B, and f_θ is a neural network parameterised by θ, that is embedding both the exemplar and target locations into the latent embedded space. From the properties of the cosine similarity, s_l is in the unit interval. Figure 4.1 contains a visualisation of the above computation. Since s_l is computed for all locations l, the above computation was implemented by Keren et al. [59] as a convolutional application of a Siamese neural network [8] across all possible locations l in the target example, followed by an application of the cosine similarity to the resulting map. When f_θ is chosen to be a CNN, computation may be considerably reduced by applying f_θ to the entire target example, and then extracting the parts of the resulting map that correspond to the appropriate locations [31].

4.3.2 Weakly supervised detection

A *Siamese pair* (x, B_l) is considered positive if the target location B_l contains an instance of the same class as the exemplar x. Otherwise, the Siamese pair is considered negative. Similarly, an *exemplar-target pair* (x, B) is considered positive if the target example B contains an instance of the same class as the exemplar x, or negative, otherwise. During training, the goal is to increase the similarity score of positive Siamese pairs, while decreasing similarity scores for negative ones. Recall, that at training time the model has no access to labels indicating whether a given Siamese pair is positive or negative. Instead, the model is only using binary labels $y_{x,B} \in \{0, 1\}$, indicating whether an exemplar-target pair is a positive or a negative one.

For negative exemplar-target pairs (x, B), the model should assign a similarity score $s_l = 0$ to all Siamese pairs, as all Siamese pairs are negative. For positive exemplar-target pairs, in the absence of labels that contain localisation information, an additional credit assignment arises, namely, which Siamese pairs should be assigned a high similarity score, and which ones should assigned a low similarity score. In other words, the model needs to identify which Siamese pairs are positive and which are negative.

Keren et al. [59] attempt to overcome this issue by introducing a novel attention mechanism, that makes the different location compete for a high similarity score. Related models for one-shot learning were introduced in Bilen and Vedaldi [9], Teh et al. [113], but these are

not suited for the one-shot learning setting. The attention mechanism from Keren et al. [59] operates as follows. Attention weights are computed by applying a softmax normalisation across the similarity map

$$w_l = \frac{\exp(s_l/T)}{\sum_{l'} \exp(s_{l'}/T)}, \tag{4.2}$$

where T is the softmax temperature. Then, a single similarity score is computed for the exemplar-target pair (x, B) using the locations' attention weights

$$\hat{y}_{x,B} = \sum_l w_l s_l, \tag{4.3}$$

which is again in the unit interval. This instance-level prediction can now be compared with the exisitng instance-level labels. The loss for the exemplar-target pair is a simple MSE loss

$$\ell(x, B) = (\hat{y}_{x,B} - y_{x,B})^2.$$

Using the gradient of the softmax function, we can compute

$$\begin{aligned}\frac{\partial \hat{y}}{\partial s_l} &= w_l + \frac{s_l w_l (1 - w_l)}{T} - \sum_{l' \neq l} \frac{s_{l'} w_l w_{l'}}{T} \\ &= w_l (1 + \frac{s_l}{T} - \frac{1}{T} \sum_{l'} w_l s_l) = w_l (1 + \frac{s_l - \hat{y}_{x,B}}{T}),\end{aligned}$$

therefore the gradient of the above loss function with respect to similarity scores is

$$\frac{\partial \ell}{\partial s_l}(x, B) = 2(\hat{y}_{x,B} - y_{x,B}) w_l (1 + \frac{s_l - \hat{y}_{x,B}}{T}). \tag{4.4}$$

The gradient in Eq. 4.4 is monotonic decreasing in s_l and w_l for positive exemplar-target pairs ($y_{x,B} = 1$). Therefore, all similarity scores for all Siamese pairs in positive exemplar-target pairs will increase when propagating this gradient to the network, but score of Siamese pairs with an already large similarity score will be increased more, compared to scores of Siamese pairs with a low similarity score. This is a self-reinforcing loop: in positive exemplar-target pairs, Siamese pairs with high similarity scores will get even higher similarity scores during training, therefore also higher attention weights and again a more negative gradient that will increase similarity scores for these locations even more. Similarly, similarity scores of Siamese pairs with a low similarity score will be increased less and will remain low. Figure 4.2 contains a visualisation of the connection between the similarity scores and the gradient propagated to them.

The conclusion from above discussion about the gradient is that it should be enough to make positive Siamese pairs have a slightly larger similarity scores compared to negative Siamese pairs, and the self-reinforcing loop will increase this difference. This slight advantage for positive Siamese pairs should naturally occur, as explained below. Positive Siamese pairs have in general more in common with each other than negative pairs have in common with each other. Therefore, gradient updates based on negative Siamese pairs can be seen as noise and are likely to cancel one another, while gradient updates based on positive Siamese pairs are likely to statistically be in the same direction and aggregate. The resulting difference in similarity scores between positive and negative Siamese pairs should be enough for the self-reinforcing loop to begin and eventually assign considerably larger similarity scores to positive Siamese pairs compared to negative ones.

4.3.3 One-shot learning

The model presented by Keren et al. [59] is using an exemplar, a target example, and a binary label indicating the existence of the exemplar's class in the target example. Not conditioning the model on class identities may allow the model to better generalise to unseen classes, as exemplars and target examples from unseen classes may share some characteristics with those from classes seen at training time. This can be seen as the model implicitly learning to represent a class by its examples, then being able to generalise to similar classes in this class representation space.

Keren et al. [59] define T as the distribution of available classes at training and evaluation time. X_L is denoted to be the distribution of exemplar-target pairs (x, B) such that x belongs to class L, and B contains an instance of class L with a probability of 0.5. At training time, classes are sampled from a uniform distribution T', with a support set that is a subset of the support set of T. During training, the loss over exemplar-target pairs sampled from X_L that correspond to classes from the support set of T'. Formally, the goal of training is then choosing model parameters Θ such that

$$\Theta = \arg\min_{\theta} \mathbb{E}_{L \sim T'}[\mathbb{E}_{(x,B) \sim X_L} \ell(x, B)]]. \tag{4.5}$$

Training the model according to Eq. 4.5 is hypothesised by Keren et al. [59] to yield a model that generalises to good performance with classes $L \sim T$ that were not seen at training time, without any need for additional training with those classes.

4.3.4 Detection

In the detection setting, given a target example, the model needs to predict which of the N possible classes appears in the target example, and localise those appearances. To do so, the model is run N times, each time using an exemplar from a different class. This is done by Keren et al. [59] in a computationally efficient manner, as $f_\theta(B_l)$ from Eq. 4.1 is only computed once for each target location.

For emitting detections, a candidate detection is emitted for an exemplar-target pair (x, B) at location l such that $s_l = \max_{l'} s_{l'}$. Note that a simplifying assumption is made here, that the target example may contain no more than one instance of the same class of the exemplar. One may omit this assumption by emitting multiple candidate detections for an exemplar target pair, but for simplicity we follow this assumption in this chapter.

4.4 Experiments

4.5 Audio data

For evaluating the weakly supervised one-shot detection model in the audio domain, Keren et al. [59] tackle the query-by-example spoken term detection task [14, 40, 89, 126, 127]. In this task, the different classes are the different words. A recording of a word uttered by a speaker is an exemplar of some class, referred to below as a keyword. A recording of a longer audio utterance is a target example. The goal in this task is to determine whether the same word as the keyword appears in the utterance, and emit a bounding box location in time for the appearance.

Keren et al. [59] construct a large-scale dataset for the query-by-example spoken term detection task, using two separate existing corpora: a speech recognition corpus and an audio keywords corpus. The audio keywords were downloaded from the Shtooka project website (http://www.shtooka.net). All keywords are in English, and are less than one second long. Every audio keyword appears in exactly one recording, meaning that every class has only a single exemplar. Each keyword was allocated to either the training, validation or test set. This split strategy was done to ensure that model evaluation is done on classes that were not seen at training time, resulting in a one-shot learning setting.

The textual form of words can appear as a part of other, longer words (for example, ‘the’ is a part of ‘their’ and ‘further’), which results in an undefined desired behaviour for the model. For this reason, Keren et al. [59] did not use short words, that are more prone to this issue. Therefore, only words that consist of four letters or more were used. In total, the

training set for this task contains 5442 keywords (classes), and the validation and the test set contain 908 keywords each.

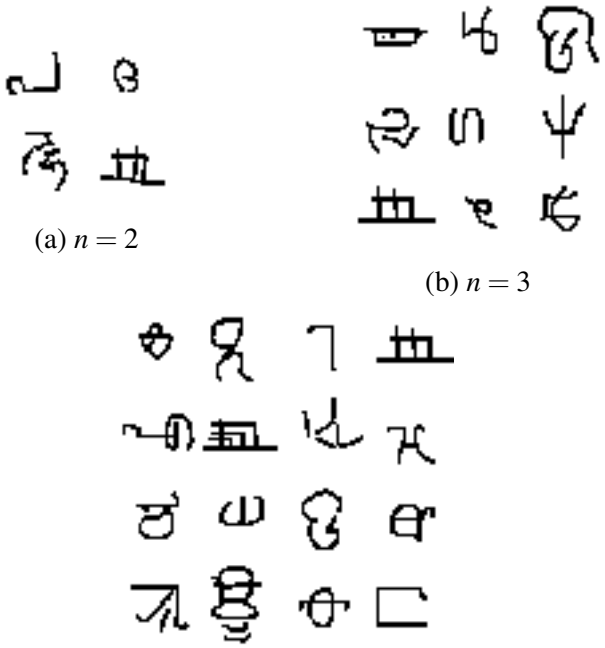
All audio utterances that were used are from the Librispeech corpus [88]. The Librispeech corpus contains 1000 hours of annotated English speech, from 2484 speakers. Each utterance in the Librispeech corpus was allocated to one of the training, validation or test sets, according to the official split of this corpus, which is gender balanced. Each utterance was cut to be exactly five seconds long.

The training set is comprised of keyword-utterance pairs, which are the exemplar-target pairs, and the corresponding binary labels indicating the existence of the keyword in the utterance. The keyword-utterance pairs are referred to as either positive or negative pairs, as explained in Section 4.3.2. It is important to note that the keyword and the utterance are always from different recordings, made by different speakers. All positive keyword-utterance pairs where both the keyword and utterance were allocated to the training set are contained in the training set. In order to balance the positive and negative classes in the training set, Keren et al. [59] randomly sample a number of training utterances that do not contain the keyword and add the resulting negative pairs to the training set. In total, the training set contains 330,018 keyword-utterance pairs.

For constructing the validation and test sets, Keren et al. [59] first add all positive keyword-utterance pairs that belong to the appropriate evaluation set (validation or test). For evaluating the model on detection over a number of unseen possible classes (different unseen classes every time though), for every positive keyword-utterance pair in the evaluation set another N negative keyword-utterance pairs were added, that comprise of the same utterance as in the positive pair. Note that the more negative keyword-utterance pairs that are added to the model, the more false positives the model is likely to emit, which will impair the overall performance of the detection model. Keren et al. [59] create the validation and test set as described above with $N \in \{10, 20, 50\}$, and name the resulting test sets $test10$, $test20$ and $test50$ respectively. See Section 4.5.3 for details about the detection task evaluation method.

Labels regarding the temporal location of a keyword in an utterance were extracted using forced alignment. Specifically, the Montreal Forced Aligner [79] was used with default parameters. These labels were used for creating ground truth bounding boxes, that are used when evaluating the detection model on the validation and test sets. However, these labels were not used at training time, as this work considers a weakly supervised model.

As some words are more common than others, every keyword has a different number of keyword-utterance pairs that it is a part of, which can result in a small number of keywords dominate the training or evaluation procedure. To counter this effect, in each of the training, validation and test sets, Keren et al. [59] count the number of keyword-utterance pairs



(a) $n = 2$

(b) $n = 3$

(c) $n = 4$

Fig. 4.3 Sample tiled Omniglot images, used for the weakly supervised one-shot detection task. Reprinted with permission from Keren et al. [59].

that each keyword appears in, and use only keywords that are below the 85th percentile in this count. Overall, the training, validation and test sets contain 3592, 259 and 285 audio keywords respectively.

4.5.1 Computer vision data

The Omniglot dataset [68] contains images of 1623 different characters from 50 different alphabets. As discussed above, using models that are not conditioned on class identities may facilitate generalisation to classes unseen at training time, but this may require a large number of classes to appear at training time. In this setting, the different classes are the different characters the dataset contains. The Omniglot dataset contains a large number of characters, therefore it may be suitable for the one-shot learning task, and was used in previous works for similar tasks [63, 119].

The dataset contains images of single characters. To adjust this dataset for a detection task, Keren et al. [59] tile n^2 single character images in a square grid and create a large square image with $n \times n$ images of different single characters. Those images are non-overlapping and are arranged in a random order. Figure 4.3 contains examples of such images. All images of single characters are downsampled to 32×32 pixels to reduce the computational

requirements of the model. Given an image of a single character (an exemplar), the detection task is to determine whether a given large image (the target example) contains an instance of the same character as the exemplar, and determine the location of this instance in the large image. It is important to note, that the target image may contain the same character as in the exemplar, but a different image thereof.

The different alphabets of the Omniglot dataset were allocated into non-overlapping training, validation and test sets. For the test set, the alphabets from the official dataset split were used. Training, validation and test sets contain exemplar-target pairs where all characters in both the exemplar and the target example are from the appropriate training, validation or test sets. For the evaluation sets (validation and test) for a detection task over N classes, for every image of a character, we create a single target example, than one positive exemplar-target pair and $N-1$ negative ones.

4.5.2 Network specifications

The model used in Keren et al. [59] represents the exemplar and the locations in the target example using a neural network f_θ, as appears in Eq. 4.1. In experiments with both the audio and computer vision domains, f_θ is modelled as CNN, with one-dimensional convolutions for the audio data and two-dimensional convolutions for the image data. A similar network was used in Keren et al. [56] and in Keren et al. [57]. The network specifications were described in Keren et al. [59] as follows. The network is comprised of eight convolutional layers, each using a kernel size of five (5×5 for the image data), with a stride of one (1×1 for the image data). The first four convolutional layers are comprised of 256 feature maps, while the last four convolutional layers are comprised of 512 feature maps. max-pooling with a kernel size and stride of 2 is applied after every second convolutional layer, to reduce representation size. Every convolutional layer is followed by a batch normalisation operation [46] and a rectified-linear activation function. Both the exemplars and target example were processed through this network, to obtain their representations.

For computing the cosine similarity from Eq. 4.1 for the different locations, the exemplar representation was convolved with the representation of the target example, after normalising the L^2 norm of the exemplar representation and each location in the target example's representation. A softmax temperature of $T = \frac{1}{3}$ was used for computing the attention weights according to Eq. 4.2. Training is done with SGD according to Eq. 4.5, using a minibatch size of 64 exemplar-target pairs and a learning rate of 0.1. Hyperparameters were tuned using the validation set. In the computer vision experiments, the model was first trained on the training set for determining the optimal number of training iterations, then retrained using the training and validation sets for that number of iterations.

4.5.3 Evaluation

The proposed ASN model for weakly supervised one-shot detection is evaluated in experiments with both the audio and computer vision domains in Keren et al. [59]. As discussed above, bounding box information is not used for training, bur rather only for model evaluation on the test set.

For the audio experiments, dynamic time warping (DTW) was used as a baseline. When using DTW, sequences of different lengths are matched with each other [125]. The duration of articulation of a word normally differs between different utterances and speakers, therefore DTW is a suitable approach for query-by-example spoken term detection [50]. DTW does not require any training phase, therefore suitable for the one-shot query-by-example paradigm. Mel-frequency cepstral coefficients (MFCCs) were used as acoustic features, as was done in Joder et al. [50]. MFCC coefficients $1-12$ were used, extracted from frames of 20 ms shifted by 10 ms using the OPENSMILE toolkit [29].

The DTW algorithm finds the shortest path between the MFCC representation over time of each keyword and a given segment from an utterance. Then, a cost value C is returned, describing the sum of the Euclidean distances of the shortest path between the sequences. Keren et al. [59] convert this cost value to a similarity measure between the keyword and a location in the utterance:

$$s_l = \exp \frac{-C}{\sigma}, \tag{4.6}$$

where the hyperparameter $\sigma = 50$ yielded best result on the validation set. For emitting detections, for an exemplar-target pair (x,B) Keren et al. [59] define $\hat{y}_{x,B} = \max_l s_l(x,B)$, and consider this a possible detection at location l where the maximum was acquired with confidence $\hat{y}_{x,B}$.

As a baseline for the computer vision experiments, Exemplar Support Vector Machine (Exemplar-SVM) based on Histogram of Oriented Gradients (HOG)-features was used. The Exemplar-SVM approach for the purpose of object recognition in images has been proposed in Malisiewicz et al. [76]. In Malisiewicz et al. [76], the authors use the HOG-representation of objects in images and trained several SVM classifiers, with only a single positive instance (object is present in a bounding box) and a large number of negative ones (object is not present) for each SVM, and combined the outputs of the models within an ensemble. HOG features were also used for the recognition of handwritten characters in Surinta et al. [107].

As opposed to the task from Malisiewicz et al. [76], the authors of Keren et al. [59] use only a single positive exemplar. Complexity hyperparameters for the exemplar and the negative sample were tuned on the validation set. For each image of a single character from the test set characters, Keren et al. [59] train an SVM classifier to separate this image from

all character images in the training set. HOG features were used for training the SVM, as those were found to yield better results compared to raw pixels. For every exemplar-target pair (x, B), the trained SVM for a single character image x is convolved with the larger image B, to get the similarity scores $s_l(x, B)$ for the different locations. For emitting detections, for an exemplar-target pair (x, B) Keren et al. [59] define $\hat{y}_{x,B} = \max_l s_l(x, B)$, and consider a possible detection at location l where the maximum was acquired with confidence $\hat{y}_{x,B}$.

All the considered methods process an exemplar-target pair (x, B) and output a possible detection with confidence $\hat{y}_{x,B}$ at a location l. A detection is emitted if the confidence is larger than a threshold t, that is chosen using the validation set for each method separately. In the audio experiments, start and end points of emitted detections are shifted by constants a and b respectively, that are again chosen using the validation set for each method separately. This shift was not considered for the image experiments, as in those experiments there was only a small number of possible locations for the detection.

For each method, an average precision is computed from the emitted detections and the ground truth bounding boxes, using an intersection over union (IoU) value of 0.5 [28]. In contrast to Everingham et al. [28], average precision in Keren et al. [59] is computed for all classes together, instead of computing the mean of the average precision of the different classes. The reason for this deviation from the well-established evaluation protocol is the small number of ground truth boxes per class in the one-shot learning setting.

Every method is originally trained for binary classification, to determine whether an instance of the same class as the exemplar appears in the target example. Therefore, Keren et al. [59] also evaluate every method's accuracy in this binary classification task. This evaluation is done by computing the precision at given recall levels of 0.5, 0.9 and 0.99, using the confidence score $\hat{y}_{x,B}$ for each keyword-utterance pair.

For the audio experiments, three different test sets were created, for 10-way, 20-way and 50-way weakly supervised one-shot detection, as described in Section 4.5. For the computer vision experiments, test sets for 5-way, 10-way and 20-way weakly supervised one-shot detection were created for $n \in \{2, 3, 4\}$, as described in Section 4.5.1. Results for experiments performed by Keren et al. [59] in both domains are found in Table 4.1. The results in the table show that the proposed attention similarity networks considerably outperformed the baseline methods for both the audio and computer vision domains, for all test sets. Specifically, in the audio domain the proposed model yielded AP scores of 42.6%, 38.3% and 23.6% for the detection task over 10, 20 and 50 unseen classes, compared to AP scores of 8.9%, 6.0% and 3.7% with DTW. In the computer vision domain, using $n = 3$ for example, the proposed method yielded AP scores of 58.0%, 49.3% and 37.2% for the detection task over

Table 4.1 Results for experiments in the audio and computer vision domains, as appear in Keren et al. [59]. Average precision (AP[%]) and precision at given recall levels (Pr'@x[%]) are reported on the different N-way test sets for the weakly supervised one-shot detection task, for the proposed attention similarity networks (ASN) and the dynamic time warping and Exemplar-SVM baselines. Higher scores are better. For both domains, the proposed attention similarity networks outperforms the baseline in all performance measures.

MODEL	SET	AP	PR'@0.5	PR'@0.9	PR'@0.99
AUDIO - ASN	10-WAY	42.6	73.1	25.9	12.7
	20-WAY	38.3	50.9	16.0	6.5
	50-WAY	23.6	29.9	5.8	2.6
AUDIO - DYNAMIC TIME WARPING	10-WAY	8.9	14.6	11.1	10.4
	20-WAY	6.0	7.8	5.8	5.5
	50-WAY	3.7	3.3	2.4	2.3
IMAGE - ASN ($n = 2$)	5-WAY	75.4	89.3	52.3	27.0
	10-WAY	67.1	79.2	36.3	14.9
	20-WAY	54.4	61.8	18.1	7.4
IMAGE - EXEMPLAR-SVM	5-WAY	42.6	50.8	25.0	20.7
	10-WAY	33.8	31.2	12.9	10.4
	20-WAY	26.6	17.7	6.6	5.2
IMAGE - ASN ($n = 3$)	5-WAY	58.0	73.1	39.4	24.5
	10-WAY	49.3	63.1	22.0	12.0
	20-WAY	37.2	41.5	11.9	6.2
IMAGE - EXEMPLAR-SVM	5-WAY	31.1	40.3	23.4	20.5
	10-WAY	24.8	23.0	12.0	10.3
	20-WAY	19.6	12.4	6.1	5.2
IMAGE - ASN ($n = 4$)	5-WAY	43.2	56.7	27.6	22.3
	10-WAY	32.1	34.4	13.7	10.9
	20-WAY	23.8	19.8	7.2	5.2
IMAGE - EXEMPLAR-SVM	5-WAY	26.3	34.8	22.6	20.4
	10-WAY	20.8	19.1	11.5	10.2
	20-WAY	16.4	10.0	5.8	5.1

5, 10 and 20 unseen classes, compared to AP scores of 31.1%, 24.8% and 19.6% with the Exemplar-SVM.

The attention similarity networks proposed by Keren et al. [59] managed to simultaneously identify and localise instances of classes unseen at training time. Experiments results show that it is indeed feasible to generalise to unseen classes when using a model that is not

conditioned on class identities, and that localisation information can be learnt in a weakly supervised manner, when pairing a Siamese similarity network with an attention mechanism.

In further experiments conducted by Keren et al. [59], the trained attention similarity networks were used to compute AP using different IoU values. Results are depicted in Figure 4.4. The results show that further reducing the IoU requirement considerably improves the AP results. For example, for the audio data and 10-way one-shot detection, reducing the IoU threshold from 0.5 to 0.4 improved the AP from 42.6 to 47.3. Further reducing the IoU to 0.3 and 0.2 resulted in AP values of 50.5 and 54.7 respectively. These finding indicate that in many cases the correct class was detected, but the emitted detection location was inaccurate, thus further motivating future work for refining those detection locations.

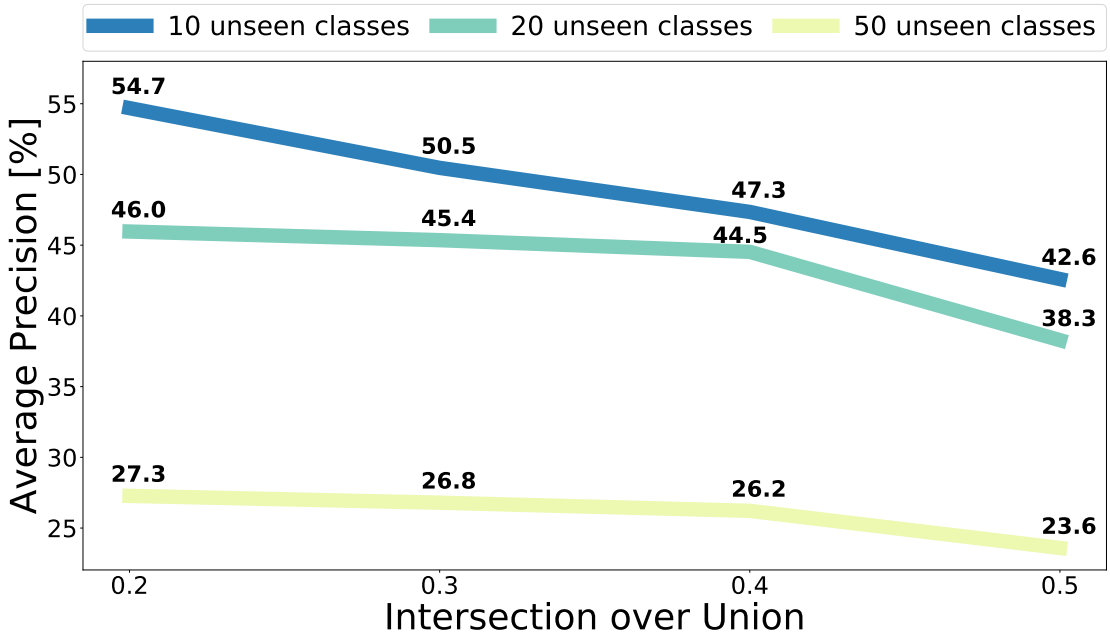


Fig. 4.4 Average precision (AP) of the attention similarity networks for the audio keywords detection task for the 10-way, 20-way and 50-way weakly supervised one-shot detection with different intersection over union (IoU) thresholds. Reprinted with permission from Keren et al. [59].

Chapter 5

Calibrated Prediction Intervals

5.1 Motivation

Over the past few years, deep learning systems have improved overall task accuracy for a wide range of machine learning applications. In addition, there is an increasing research interest in the estimation of prediction uncertainty, i.e., estimating the chance that a model prediction may be wrong and the deviation of the prediction from the ground truth value. In a wide range of application, this estimation of prediction uncertainty may be crucial. For example, in the healthcare industry, the implications of a wrong machine learning prediction may be life-threatening. Another example is autonomous driving, where a wrong confident decision regarding the existence of a pedestrian on the road may have critical implications.

For a regression problem, the estimation of prediction uncertainty can be given in the form of a *prediction interval*, an interval in which the ground truth label is expected to fall with a prescribed probability. Neural network regressors are often designed with a single unit in their top layer that contains the predicted label [42, 115, 129]. Using these models, producing prediction intervals cannot be done in a straight forward manner. Other neural network regressors are designed to function as softmax classifiers, but binning the real numbers into a finite number of bins and learning the task as a standard classification task [87, 116]. The latter models allow emitting a posterior probability distribution over the output space. Using this method, one can produce prediction intervals for any given confidence level α in the following manner: compute the expected value of the posterior probability distribution over the output space, and take a number of bins around this expected value that together contain α of the probability mass. This procedure is illustrated in Figure 5.1.

However, producing prediction intervals with a confidence level α in the manner described above does not guarantee that these intervals contain their respective labels with a probability

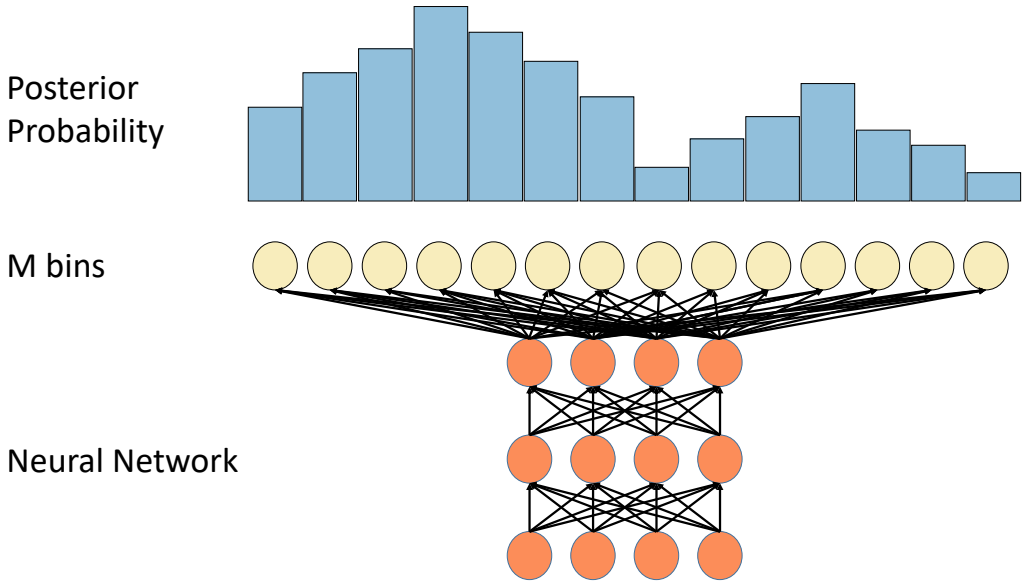


Fig. 5.1 A neural network regressor designed as a softmax classifier. By binning the output space into M bins, one can design a neural network regressor as a softmax classifier over M classes, and derive a posterior distribution over the output space instead of a single point estimate. This allows emitting prediction intervals that contain a prescribed amount of the posterior probability mass. However, we show that the resulting prediction intervals will normally be miscalibrated, i.e., will not correspond to the desired confidence level. Reprinted with permission from Keren et al. [53].

of α. For example, a neural network producing overconfident predictions may allocate almost all posterior probability mass in one output bin, but in reality the label may be in that selected bin only in a small percent of the cases. In that case, we say that the prediction intervals are miscalibrated. Recent work has shown that modern neural network classifiers indeed tend to produce overconfident predictions and therefore their output is miscalibrated, in the sense that a posterior probability mass assigned to a given class does not reflect the actual probability of the true label being that class [36]. Therefore, when designing neural network regressors as classifiers, we expect the resulting prediction intervals to be miscalibrated as well.

Neural network were not always considered to produce miscalibrated confidence predictions. For example, Guo et al. [36], Niculescu-Mizil and Caruana [86] show that neural network models that were commonly used a decade and more ago produce well calibrated probability estimations for a classification task. It was additionally shown that modern neural networks tend to produce miscalibrated probability estimations as a result of changes to the

models and the training procedure [36], such as the increase in model capacity, the use of batch normalisation [46] and not using weight decay.

Motivated by the above, Keren et al. [53] present two novel methods for producing calibrated prediction intervals for neural network regressors, for any given confidence level, thus answering the research question posed in Section 1.4. Both methods operate as post-processing methods, using the predictions of a pretrained model. In addition, both methods are based on designing the neural network regressor as a classifier, do not require retraining the model and are very fast to compute. The first proposed method, *empirical calibration*, maps the amount of posterior probability mass allocated to the different bins to the probability of labels falling in those bins. By doing so, this method finds the amount of posterior probability mass that yields prediction intervals in the desired level of confidence. The second proposed method, *temperature scaling*, is an adaptation of a related method proposed in Guo et al. [36] for the classification setting, to the setting of emitting prediction intervals for regression. By adjusting the softmax temperature, one could smooth or sharpen the posterior probability distribution. Thus, the temperature scaling method finds the softmax temperature that aligns the posterior probability mass of the different bins with the probability of the label falling in those bins, allowing to produce calibrated prediction intervals.

Keren et al. [53] conduct experiments with their two proposed calibration methods, using four benchmark datasets from the audio and the computer vision domains. They find that as expected, using a neural network classifier for the regression task and producing prediction intervals using the posterior probability mass results in prediction intervals that are poorly calibrated. On the contrary, Keren et al. [53] find that applying their proposed calibration methods results in prediction intervals that are considerably better calibrated, a finding that was consistent across all datasets, neural network architectures and confidence levels.

In addition, Keren et al. [53] find that splitting the output space into a larger number of bins results in tighter prediction intervals, i.e., those intervals are of a shorter length for the same level of confidence, therefore better indicating the location of the label. Finally, Keren et al. [53] show that using neural network classifiers for a regression task does not cause any degradation in regression accuracy, as measured by the mean squared error (MSE). The source code for using the two proposed calibration method is made publicly available in https://github.com/cruvadom/Prediction_Intervals.

5.2 Related work

Despite network calibration being a more recent problem for neural networks, calibration and confidence estimation themselves are not new problems, e.g., Brummer and Leeuwen

[11], Dawid [21], Deng and Schuller [23], Jiang [49], Platt et al. [92], Weintraub et al. [122], Wessel et al. [123], Yu et al. [131]. More recently, a plethora of calibration and uncertainty quantification approaches have been proposed and developed for contemporary neural networks in the wider machine learning community. Bayesian neural networks produce a probabilistic relationship between the network input and output [52, 84], but often suffer from tractability issues. Ensemble based approaches, bootstrapping, and Monte Carlo based approaches have also been proposed, for example Gal and Ghahramani [30], Khosravi et al. [61], Lakshminarayanan et al. [70], Naumov et al. [83]. While such approaches can produce calibrated prediction intervals, they often require training and testing a multitude of different individual networks which considerably increases the associated time and computational costs [74]. Closely related to the current work, a range of post-processing calibration methods for neural network classifiers were evaluated with a range of different networks topologies [36]. The authors found some of the evaluated methods to successfully calibrate the outputs of the classification models, but counterpart methods for producing calibrated prediction intervals for neural network regressors are still absent.

5.3 Posterior prediction intervals

Consider neural network regressors that process an input $x \in \mathbb{R}$ with an associated label at training time $y \in \mathbb{R}$. The standard neural network regressor is normally designed to have a single unit in its top layer, containing the network's prediction [42, 115, 129]. At training time, the network's prediction is then compared to the labels using the MSE loss, which is the training objective. Using this design, the network outputs a single point estimate, and does not allow emitting prediction interval in a straight forward manner. In contrast, a natural approach for producing prediction intervals is to construct a model that emits a probability distribution p_x over the real numbers, and construct prediction intervals that contain a prescribed amount of probability mass. Denote by $\hat{Y}_x$ the random variable that is distributed according to p_x. Keren et al. [53] define

Definition 4. *The interval (u_x, v_x) is called a posterior α-prediction interval if*

$$\mathbb{P}[u_x < \hat{Y}_x < v_x] = \alpha \quad \textit{and} \quad u_x < \mathbb{E}[\hat{Y}_x] < v_x.$$

The posterior α prediction interval is simply an interval around the expected prediction of the network $\mathbb{E}[\hat{Y}_x]$, that contains α of the posterior probability distribution p_x. α is referred to as the *confidence level* of the posterior α-prediction interval.

For emitting a probability distribution over the real numbers, neural network regressors can be designed to operate as neural network classifiers, as was done in Oord et al. [87], van den Oord et al. [116]. The real numbers are split into a M bins with edges

$$-\infty = a_0 < a_1 < ... < a_M = \infty,$$

and for training purposes the continuous label y is replaced with a class label $t \in \{0, ..., M-1\}$ such that

$$a_t \leq y < a_{t+1}.$$

The top layer of the neural network is replaced and contains M units to accommodate M classes. Softmax normalisation is applied on the outputs of this top layer to produce a vector of class probabilities

$$(pr_0, ..., pr_{M-1}).$$

At training time, the task is learnt as a classification task using the cross-entropy loss. The process above results in a categorical probability distribution, that covers the real numbers by splitting them into M bins. Though not required in the setting of this work, one may transform this distribution into a continuous distribution over the real numbers by using a uniform distribution over values inside each bin, and a suitable alternative for the bins with edges in $\pm\infty$.

Designing a neural network regressor as a classifier allows replacing the single point prediction with a distribution over the real numbers, which in turn allows computing posterior α-prediction intervals.

5.4 Calibrated prediction intervals

Section 5.3 describes how neural network regressors can emit posterior α-prediction intervals, instead of a single point estimate. However, a confidence level of α is not guaranteed to correspond to a probability of α that the label falls within the interval boundaries. Indeed, consider for example the case of neural networks that produce overly confident predictions. These networks will tend to allocate almost all posterior probability mass in small regions of the output space, resulting is very narrow posterior α-prediction intervals. Despite the confident predictions, the resulting posterior α-prediction intervals may contain their respective labels with probability α_0, with $\alpha_0 < \alpha$. Similarly, neural networks that produce predictions that are not confident enough, may lead to posterior α-prediction intervals that are

too wide, again not reflecting the actual probability of the label falling within the boundaries of those intervals.

Ideally, prediction intervals with a confidence level of α should correspond to a probability of *alpha* of the label falling withing their boundaries. Keren et al. [53] define the notion of calibrated α-prediction intervals:

Definition 5. *A set of intervals $\{(u_x, v_x)\}_{x \in X}$ is said to be a set of calibrated α-prediction intervals if*

$$\mathbb{P}_{x,y \sim X,Y}[u_x < y < v_x] = \alpha,$$

where X, Y corresponds to the joint distribution of inputs and labels of the given regression task.

Keren et al. [53] refer to α as the confidence level of the calibrated α-prediction intervals. In regression analysis, a calibrated α-prediction interval is an estimate of an interval in which the label will lie, with a certain probability α. Calibrated α-prediction intervals capture information about the uncertainty of the predicted value across the output space, and convey information that is absent from a single point estimate of the label, that might be critical for a wide range of practical applications.

Recent work shows that modern neural network classifiers often produce non-calibrated outputs, i.e., the posterior probability assigned to a given class is not equal to the probability of this class being the correct class for the relevant example [36]. Therefore, using neural network regressors as modern neural network classifiers, and constructing posterior α-prediction intervals using the posterior probability distribution of the classifiers, is most likely to produce posterior α-prediction intervals that are not calibrated α-prediction intervals.

In the light of the above, Keren et al. [53] present two novel methods for emitting calibrated α-prediction intervals for neural network regressors, that are presented below. Recall the setting from Section 5.3, in which neural network regressors are designed as classifiers over M bins, and produce a categorical probability distribution over the different bins: $(pr_0, ..., pr_{M-1})$.

The network's real-valued prediction (point estimate) is computed as the expectation of the posterior probability distribution

$$\hat{y} = \sum_{i=0}^{M-1} pr_i * c_i, \tag{5.1}$$

where c_i is the mean of real-valued training labels that correspond to class label i. The class (bin) that contains $\hat{y}$ is denoted as $\hat{t}$:

$$\hat{t} = r \quad \text{s.t.} \quad a_r \leq \hat{y} \leq a_{r+1}. \tag{5.2}$$

As a first stage for applying each of the proposed calibration methods by Keren et al. [53], posterior α-prediction intervals are computed. These are set to be the smallest symmetric interval around $\hat{t}$ that contains α of the neural network's posterior probability mass. Formally, the posterior α-prediction intervals are computed by Keren et al. [53] as

$$(u_x^{\alpha}, v_x^{\alpha}) = (a_{\hat{t}-i}, a_{\hat{t}+1+i}), \tag{5.3}$$

such that i is the minimal non-negative integer (possibly zero) for which

$$pr_{\hat{t}-i} + ... + pr_{\hat{t}+i} \geq \alpha. \tag{5.4}$$

It is important to note that the edges of the posterior α-prediction intervals are restricted to be the edges of the bins, therefore the condition from Definition 4 is only met approximately.

Both calibration methods proposed by Keren et al. [53] apply post-processing to the outputs of a pretrained model processing unseen examples, and therefore do not require any retraining of the neural network model. Hyperparameters for the methods are chosen using a validation set, and the chosen values are used when applying the methods on the test set predictions.

5.4.1 Empirical calibration

Observe that a set of posterior α_0-prediction intervals $(u_x^{\alpha_0}, v_x^{\alpha_0})$ as defined according to Equation 5.3 is always also a set of calibrated α_1-prediction intervals for

$$\alpha_1 = \mathbb{P}_{x,y\sim X,Y}[u_x^{\alpha_0} < y < v_x^{\alpha_0}]. \tag{5.5}$$

The last statement is true, since for any given set of posterior α_0-prediction intervals, there exist some probability α_1 of the labels falling withing the boundaries of those intervals. For this probability, this set of intervals is also a set of calibrated α_1-prediction intervals by definition.

When applying the empirical calibration method, the goal is to find α_0 such that the posterior α_0-prediction intervals are calibrated α-prediction intervals, for a desired confidence level α. Note that $\mathbb{P}_{x,y\sim X,Y}[u_x^{\alpha_0} < y < v_x^{\alpha_0}]$ is increasing in α_0 with fixed points in 0 and 1,

since the size of an interval is increasing with the amount of posterior probability mass it contains, therefore the label is more likely to fall within this interval's boundaries.

Thus, the empirical calibration method proposed by Keren et al. [53] is comprised of a binary search along different values of $\alpha_0 \in [0,1]$ to find α_0 such that $|\mathbb{P}_{x,y\sim X,Y}[u_x^{\alpha_0} < y < v_x^{\alpha_0}] - \alpha| < \varepsilon$ on the validation set, for a given error tolerance ε. A value of $\varepsilon = 0.001$ is used for the experiments performed by Keren et al. [53]. Having a small level of error tolerance is mandatory, since it may be impossible to find calibrated prediction interval with a confidence level of *exactly* α on a finite validation set. The value of α_0 that is found in the binary search is the one that should be used for performing empirical calibration on the test set.

5.4.2 Temperature scaling

Neural network classifiers emit a vector of class probabilities $(pr_0, ..., pr_{M-1})$ that is computed from the logits (the outputs of the top layer) $(z_0, ..., z_{M-1})$ using a softmax function:

$$pr_i = \frac{\exp(z_i/T)}{\sum_{j=0}^{M-1} \exp(z_j/T)}, \tag{5.6}$$

where T is called the softmax temperature. A default temperature of $T = 1$ is used during training. Equation 5.6 can be written as

$$pr_i = \frac{1}{\sum_{j=0}^{M-1} \exp((z_j - z_i)/T)}, \tag{5.7}$$

showing that the softmax output depends only on the differences between the logits' values and the temperature used. As a result, scaling the logits by a scalar value before applying the softmax normalisation will result in changing the smoothness of the posterior probability distribution. Specifically, a low temperature $0 < T < 1$ will result in a "sharper" distribution, i.e., probability mass will concentrate more in classes with large logit values. On the other hand, using a temperature $1 < T < \infty$ results in "smoothing" of the posterior probability distribution, i.e., probability mass is distributed more equally between all classes.

Guided by the property discussed above, the temperature scaling method proposed in Keren et al. [53] tunes the softmax temperature at evaluation time for computing class probabilities. Temperature scaling was used in Guo et al. [36] for calibrating the output probabilities of neural network *classifiers*, and Keren et al. [53] extend this method to the regression and prediction intervals setting. A network that produces overly confident predictions will result in posterior α-prediction intervals that are too narrow, i.e., $\mathbb{P}_{x,y\sim X,Y}[u_x^{\alpha} < y < v_x^{\alpha}] < \alpha$.

For this case, a softmax temperature $T > 1$ may be used at evaluation time to reduce the network's confidence by smoothing the posterior probability distribution, thus increasing the size of posterior α-prediction intervals. Equivalently, a low temperature $0 < T < 1$ should be used to increase the network's confidence and decrease the width of posterior prediction intervals.

Keren et al. [53] define

$$F_\alpha(T) = \mathbb{P}_{x,y\sim X,Y}[u_x^\alpha < y < v_x^\alpha] \tag{5.8}$$

where u_x^α and v_x^α are the posterior α-prediction intervals that now depend also on T. As an increase in T increases the width of the posterior prediction intervals, the function $F_\alpha(T)$ is continuous and monotonic increasing in T, with $\lim_{T\to 0} F_\alpha(T) = 0$ and $\lim_{T\to\infty} F_\alpha(T) = 1$. As a result, there exist a temperature T^* such that $F_\alpha(T^*) = \alpha$.

Motivated by the above, the temperature scaling method performs a binary search along different values of T to find a temperature value T^* such that

$$|F_\alpha(T^*) - \alpha| < \varepsilon \tag{5.9}$$

on the validation set, for the desired confidence level α and a given error tolerance of ε. An error tolerance of $\varepsilon = 0.001$ is used in the experiments performed by Keren et al. [53]. The small error tolerance is mandatory, as the validation set is finite, therefore it may be impossible to find a set of calibrated prediction intervals with a confidence level of *exactly* α.

5.5 Experiments

Keren et al. [53] experimented with their two proposed calibration methods on four benchmark datasets from the audio and computer vision domains. The description of the tasks and the neural network used is given below.

5.5.1 Age prediction (audio)

The first task that was considered is the prediction of speakers' age based on a recording of their speech, using the aGender corpus [12, 100]. The aGender corpus contains audio recordings of predefined utterances and natural speech, annotated for the speakers' age and gender. Keren et al. [53] split into speaker independent training, validation and test sets, according to the split used in Keren and Schuller [60]. In total, the three sets contain more than 38 hours of audio, in more than 53,000 utterances. The total number of speakers is 611,

such that 331 speakers were assigned to the training set, 140 to the validation set, and 299 to the test set. Mel-Frequency Cepstrum Coefficients (MFCC) features were extracted from each recordings, using frames of 25 ms shifted by 10 ms. From every frame 13 features were extracted. Mean and standard deviation normalisation were applied across features and time, for every recording separately.

5.5.2 SNR prediction

The second regression task from the audio domain that was used in the experiments is prediction of Signal-to-Noise Ratio (SNR) of speech audio utterances with background noise. For constructing this task's corpus, Keren et al. [53] used clean speech utterances from the degree of nativeness corpus from the INTERSPEECH 2016 computational paralinguistics challenge [54, 101] and background noise recordings from the CHiME-4 challenge [118]. The native language corpus contains more than 64 hours of clean speech utterances from 5,132 speakers of 11 different native languages, split into speaker independent training, validation, and test sets. The background noises are recordings of four different environments: bus, café, pedestrian area, street junction, and are 14 hours in total. For creating the training set, training speech utterances were mixed with random segments of the background noises according to a random SNR in the range $[0, 25]$. The SNR was then used as the real-valued label for the regression task. The validation and test sets were created in a similar manner, using the corresponding clean utterances from the native language corpus and dedicated portions of the noise recordings. A short-time Fourier transform (STFT) is applied on every recording to extract 201 magnitude spectrogram features from every 25 ms frame, where frames are shifted by 10 ms. The magnitude spectrogram features were then normalised across features and time, for every utterance separately, to have a mean of 0 and a standard deviation of 1.

5.5.3 Age prediction (images)

The first dataset Keren et al. [53] experimented with in the computer vision domain is the Wikipedia faces dataset [97]. The dataset contains 62,359 images of people (one image per person) crawled from Wikipedia, labelled with the age of each person at the time the picture was taken. Since the dataset has no official training, validation, and test splits, 60% of the examples were randomly allocated to the training set, 20% to the validation set and 20% to the test set. As the dimensions of the different images vary, every image was resized to 224×224 pixels before feeding it to the neural network. In addition, pixel values were normalised for every image separately, to have a mean of 0 and a standard deviation of 1.

5.5.4 ISO speed prediction

The second images dataset Keren et al. [53] experimented with is the MIRFLICKR-25000 dataset. The MIRFLICKR-25000 dataset consists of 25,000 images downloaded from the social photography site Flickr through its public API [77]. In addition to images, the dataset contains additional metadata on every image, such as the ISO speed, that measures the sensitivity of the camera's film or sensor to light. The ISO speed affects the brightness of photos, therefore a regression task for predicting the ISO speed of given images is sensible. Keren et al. [53] split the dataset and extracted features in the same way as described in Section 5.5.3.

5.5.5 Neural networks

As described in Section 5.3, Keren et al. [53] learn the regression tasks using a classification neural network, where the real numbers are split into M bins. For the audio experiments, the network used is comprised of two long short-term memory (LSTM) layers, each with 512 units. The output of the last time step in the top layer is fed into the fully-connected output layer, with the number of units equal to the number of bins used. Softmax normalisation is applied to the output layer's units.

For the computer vision experiments, a convolutional neural net (CNN) is used, that is comprised of 8 residual blocks [42]. Each residual block first applies a convolutional layer on the input, followed by batch normalisation [46] and a rectified linear activation function. A second convolutional layer is then applied on the output of the rectified linear activation, and the output is added to the block's input. Batch normalisation and another rectified linear activation are then applied, to emit the output of the residual block. Before applying the residual blocks, a convolutional layer with a 7×7 kernel is applied on the network's input, with a 2×2 stride and 64 feature maps. The output of this convolutional layer is fed to a sequence of 8 residual blocks, all using convolutional kernel size of 3×3 and 64,64,128,128,256,256,512,512 feature maps (one value for each residual block). A 2×2 stride is applied for residual blocks number 3, 5 and 7. A global average pooling is applied on the output of the last residual block, to average each of the 512 feature maps across all spatial locations. Similarly to the audio experiments, a fully-connected layer is then applied to project the 512 dimensional vector to the relevant number of bins, and a softmax normalisation is applied.

In all experiments, the training objective is the standard cross-entropy, and model parameters are learnt using the Adam optimiser [62] with default β_1, β_2 values and a learning rate of 0.001. Keren et al. [53] experimented with binning the real numbers into $M = 10, 30, 60$

bins to demonstrate that our method can operate successfully regardless of the number of bins, and to study the differences between the resulting prediction intervals with different number of bins. For a given number of classes M, class boundaries were set to $a_0, ..., a_M$ to be equally spaced between the minimum and maximum real-valued label values in the training set, and then $a_0 = -\infty$ and $a_M = \infty$ were set.

5.5.6 Calibration results

Each of the two calibration methods from Section 5.4 were evaluated on the four regression tasks, with different number of bins. For each task, three neural networks are trained using 10, 30 and 60 bins respectively. The two calibration methods are then applied to the outputs of the trained neural networks, to emit prediction intervals with confidence level of 66%, 80% and 90%. Hyperparameters for each calibration method were chosen on the validation set, and were then used when applying the relevant calibration method on the test set. All results are reported on the test set.

Each calibration method aims at producing calibrated α-prediction intervals. For assessing whether this goal was achieved, Keren et al. [53] measure the *calibration error*, which is the absolute difference between the desired confidence level α and the probability of the emitted prediction intervals to contain their respective labels. Keren et al. [53] define the calibration error as follows:

$$|\mathbb{P}_{x,y\sim X,Y}[u_x < y < v_x] - \alpha|, \tag{5.10}$$

where (u_x, v_x) is the prediction interval emitted by the calibration method for example x, and X, Y are distributed uniformly over the test set examples.

The calibration error for the posterior α-prediction intervals and the prediction intervals emitted by each of the calibration methods are reported in Table 5.1. The first observation from the table is that the posterior α-prediction intervals result in a large calibration error, and can be considered indeed as miscalibrated in general. This finding is consistent with findings from Guo et al. [36] regarding the miscalibration of modern neural network classifiers. Second, both the empirical calibration and the temperature scaling methods manage to considerably reduce the calibration error in all cases, normally down to small level of 0%-2%. These results show that both calibration methods proposed by Keren et al. [53] are suitable for emitting calibrated prediction intervals for neural network regressors. These findings are consistent across all datasets, confidence levels and number of bins used for training the networks.

Table 5.1 A comparison of test set calibration error ([%]) before ('Posterior' column) and after applying each of the the two proposed calibration methods for the different regression tasks, as appears in Keren et al. [53]. 'Empirical', 'Temp' and 'Confidence' columns represent empirical calibration, temperature scaling and the prediction intervals' confidence level respectively. In all cases, both of the proposed methods manage to considerably reduce the calibration error of prediction intervals, compared to prediction intervals based on the networks' posterior distribution (smaller numbers on the right side of the dashed line). Both of the proposed methods yield comparable performance. This result holds when training the network with either 10, 30 or 60 bins, with no clear advantage for a specific number of bins.

Dataset	Confidence	Bins	Posterior	Empirical	Temp'
Age (Audio)	66%	10	7.63	0.60	0.09
		30	12.40	2.25	1.80
		60	11.95	0.82	0.35
	80%	10	10.78	0.64	1.16
		30	15.37	2.63	3.13
		60	13.74	1.45	2.69
	90%	10	9.64	1.81	2.44
		30	11.83	2.53	3.13
		60	10.97	1.95	2.16
SNR	66%	10	22.11	6.44	7.22
		30	19.67	1.78	1.78
		60	12.22	0.11	0.78
	80%	10	14.56	1.67	0.56
		30	12.67	2.89	1.78
		60	9.22	1.44	1.78
	90%	10	7.56	0.89	0.44
		30	6.11	2.78	2.78
		60	4.78	0.00	0.11
Age (Images)	66%	10	0.29	0.01	0.14
		30	4.50	0.14	0.26
		60	13.43	0.22	0.29
	80%	10	3.90	0.15	0.05
		30	2.71	0.26	0.14
		60	14.98	0.22	0.63
	90%	10	4.46	0.11	0.08
		30	0.73	0.08	0.05
		60	14.34	0.25	0.02
Iso Speed	66%	10	17.39	0.76	0.82
		30	6.06	0.06	0.70
		60	7.21	0.21	0.58
	80%	10	6.58	0.15	0.67
		30	6.58	0.15	0.67
		60	4.45	0.03	0.06
	90%	10	3.55	0.21	0.24
		30	4.06	0.73	0.27
		60	3.70	0.24	0.91

It is important to note, that the calibration error does not vanish completely, even when using the two calibration methods. This is because the hyperparameters for the methods are chosen to perform best on the validation set, and do not perfectly generalise to the test set. Nevertheless, for most practical applications a calibration error of 1%-2% is sufficiently low (e.g., a confidence level of 81% instead of a desired 80% will not make a large difference in most applications). Calibration error for both methods is comparable. Keren et al. [53] report that both methods are fast to execute, typically taking 1-3 seconds for a test set of 10000 examples, depending on the number of bins used.

Furthermore, Keren et al. [53] compare the width of the resulting prediction intervals by using the empirical calibration and temperature scaling. Table 5.2 contains the average width of the prediction intervals for the test sets of the different regression tasks. As posterior α-prediction intervals were shown to yield a large calibration error, their width is not comparable with the width of calibrated α-prediction intervals and they are left out from this evaluation. First, the results in the table show that, naturally, the width of the resulting prediction intervals increases with the confidence level used. Second, the interesting observation from the table is that training the neural networks using a larger number of bins generally results in prediction intervals that are smaller, therefore better. Specifically, for all tasks except age prediction from audio signal, the width of the resulting calibrated prediction intervals is generally smaller when using 30 or 60 bins, compared to 10 bins. This finding can be attributed to the fact that binning the output space into a larger number of bins allows the network a more precise allocation of posterior probability mass.

Both calibration methods result in prediction intervals with comparable width, therefore the two methods can be used interchangeably for emitting calibrated prediction intervals for neural network regressors. Another important note is that the width of the prediction intervals that result from a certain network, closely depends on the regression task accuracy of this network. Better regression networks will allocate more of the posterior probability mass around true label, that will result in tighter prediction intervals.

5.5.7 Regression results

Lastly, Keren et al. [53] conduct an additional experiment to determine whether using a neural network classifier for the regression task reduces the accuracy in the regression task itself, as measured by the root MSE. To this end, a standard neural neural network regressor is trained. For each task the standard neural network regressor is trained with an identical architecture to the corresponding neural network classifier for this task, except the topmost layer that contains only a single unit, as described in Section 5.3. The regressor is trained with the same optimiser as the classifiers to minimise the mean squared error (MSE) between

Table 5.2 A comparison of the test set average width of prediction intervals using the two proposed calibration methods, empirical calibration and temperature scaling, as reported in Keren et al. [53]. 'Empirical', 'Temperature' and 'Confidence' columns represent empirical calibration, temperature scaling and the prediction intervals' confidence level respectively. For all datasets except 'Age (Audio)', training the network with more bins generally results in tighter prediction intervals, since the network can learn a more precise distribution of posterior probability (numbers in the 30 and 60 bins rows are generally smaller than in the 10 bins rows). The width of the intervals is comparable between the two calibration methods and naturally grows with the confidence level. Finally, the width of the intervals naturally depends on the performance of the neural network in the regression task.

Dataset	Confidence	Bins	Empirical	Temperature
Age (Audio)	66%	10	34.84	35.70
		30	34.05	36.35
		60	36.16	38.59
	80%	10	44.55	44.79
		30	43.84	44.13
		60	45.23	45.41
	90%	10	52.77	52.68
		30	52.69	52.30
		60	53.84	54.32
SNR	66%	10	2.60	2.60
		30	1.97	1.91
		60	1.64	1.58
	80%	10	3.49	3.31
		30	2.50	2.47
		60	2.22	2.15
	90%	10	4.74	4.63
		30	3.11	3.04
		60	2.96	2.94
Age (Images)	66%	10	20.99	20.92
		30	19.66	19.11
		60	18.71	19.80
	80%	10	27.48	27.65
		30	28.83	25.53
		60	25.71	25.81
	90%	10	35.60	35.43
		30	33.51	33.58
		60	34.63	33.72
Iso Speed	66%	10	2.23	2.20
		30	1.94	1.80
		60	2.21	2.08
	80%	10	3.22	3.23
		30	2.94	3.02
		60	2.90	2.91
	90%	10	4.35	4.44
		30	4.09	4.05
		60	3.83	3.79

the network's predictions and the labels. For the classification models, MSE is computed using the prediction $\hat{y}$ defined in Eq. 5.1.

Table 5.3 contains the root MSE for the standard neural network regressors, and the neural network regressors designed as classifiers with different number of bins. The table shows that in all tasks, regression accuracy for all networks is comparable. This concludes that designing neural networks regressors as classifiers allows producing calibrated prediction intervals, and does not cause any degradation in regression accuracy.

Table 5.3 Performance in the different regression tasks as measured by the root MSE, for a standard neural network regressor and a neural network classifier with different number of classes, as reported by Keren et al. [53]. The performance of the standard regressors is comparable to the performance of the models performing regression using a classification models. This indicates that using neural network classifiers to perform a regression task allows emitting calibrated prediction intervals, and does not cause any degradation in the regression performance.

Dataset	Standard	10 classes	30 classes	60 classes
Age (Audio)	20.07	19.73	19.95	20.04
SNR	1.32	1.21	1.41	1.30
Age (Images)	11.48	11.55	11.36	11.47
ISO Speed	1.29	1.28	1.28	1.29

Chapter 6

Conclusion

This work covers a number of possible limitations in the standard supervised learning paradigm for neural networks and proposes remedies for some cases. We summarise the major findings of this work, their limitations, and give an outlook on future research opportunities to extend this work and broaden its scope.

6.1 Training without a loss function

Chapter 2 considers the case of parameter update rules that do not originate from the gradient of any loss function. Following the first research question we pose in Section 1.1, in Chapter 2 we were interested in answering whether there are settings in which such update rules are preferable over the standard gradient-based update rules. The chapter describes the work done by Keren et al. [56], where a non-gradient-based training method is proposed, allowing tuning the network's sensitivity to easy and hard training examples. In that work, it is shown that in order to directly control the relative effect of easy and hard examples on the training process, one needs to use a generalisation of the cross-entropy gradient. This generalisation is derived, and incorporates a tunable parameter that controls the sensitivity of the training process to hard examples. In experiments with both a toy example and four benchmark datasets, it is shown that using the proposed parameter update rule that is not gradient-based, and selecting the value of the sensitivity parameter using cross validation, leads to improved classification accuracy compared to using the standard cross-entropy loss, thus giving a positive answer to the research question from Section 1.1. Moreover, the experiments show that the optimal level of sensitivity to hard examples is positively correlated with the depth of the network.

We discuss some limitations to the work presented in Chapter 2 and directions for future work on this topic. First, as the presented work is an initial exploratory work on this subject,

it experiments with the proposed update rule in the context of neural networks that are composed of fully-connected layers only. In most state-of-the-art industrial and academic applications, more complex neural networks are used, that often incorporate building blocks such as convolutional or recurrent layers, residual blocks and attention mechanisms. In future work, the proposed method should be further explored in and adapted to these settings.

Further, training with a loss function has some benefits, that one may forfeit when using a training mechanism that is free of a loss function. Loss functions are in many cases easier to design, as one only needs to come up with the error term to minimise, and automatic differentiation tools such as `tensorflow` may take care of gradient computation and parameter updates. In addition, the value of the loss function is often used as a marker for training progression. Future research may explore alternative markers for training progression for the case where no loss function is used, such as the size of the gradients. In all cases however, it is recommended for a neural network practitioner or researcher to consider the characteristics of the resulting gradient of the loss function used, as those may reveal effects such as vanishing or exploding gradients, that may largely affect the training process.

As explained in Section 1.1, there is a large variety of potential update rules that do not originate from any loss function. This work shows that at least in one setting, achieving a specific manner of control over the training procedure is possible by using an update rule that does not originate from a loss function, and it is unclear whether a similar effect could be achieved using a loss function. Future research should further identify and explore additional settings in which updates rules that do not originate from a loss function may offer additional control over specific attributes of the training procedure.

6.2 Differences between loss functions

We attempted to study the effect of the choice of a loss function on the learnt representations, and the possibility of using these differences to assist with the choice of loss function to use, as posed in the research question in Section 1.2. Chapter 3 aims at answering this research question in the context of the Single Logit Classification task (SLC), covering the work done in Keren et al. [57] and Keren et al. [58]. In the SLC task, the model classifies whether a certain class is the correct class for a given example, based on the logit value for this class alone. Chapter 3 motivates this setting by showing that for a classification task over a very large number of classes, the computation of all logits may be prohibitively slow at test time. In applications where a full classification of the example is not needed at test time, but rather only probing for the correctness of a small number of classes, SLC may be a viable alternative to standard classification.

Keren et al. [57] find that the loss function used is a critical ingredient in shaping the neural network representation in the logits layer in a manner that facilitates good accuracy in the SLC task. For example, it is shown that the standard cross-entropy loss results in logits that are only informative of class correctness in comparison with other logits. In other words, the cross-entropy loss results in poor performance in the SLC task. More generally, the Principle of Logit Separation (PoLS) is formulated, stating a property of the representation in the logits layer, that a given loss function should optimise for in order to obtain good accuracy in the SLC task. Eleven different loss functions are mathematically analysed with respect to the PoLS. In experiments, it is shown that PoLS-aligned loss functions yield considerably better accuracy in the SLC task. Further, it is demonstrated that training with a PoLS-aligned loss function and applying SLC leads to considerable speedups when there are many classes, with no degradation in accuracy. This largely answers the posed research question for the SLC setting.

There are a few possible limitations to the work presented in Chapter 3. First, the chapter considers a setting in which a model is trained using a multiclass classification dataset, but at test time the task changes: every time the model is used only a small number of classes are probed for. A large dataset for multiclass classification may be used at training time as these datasets are often large and publicly available [22], making this setting realistic in applications such as autonomous driving or security cameras. However, one may note that this setting is rather specific, and does not include the standard classification problems.

The considered work yields large speedups at test time, when using SLC instead of full classification with a very large number of classes. Specifically, results in Table 3.3 show that for the common image classification networks, speedups become meaningful when using at least $2^{16} - 2^{18}$ classes. The experiments presented in Chapter 3 show that the superiority of PoLS aligned loss functions in the SLC task scale with the number of classes, but experiments were done with datasets that only contain up to 1000 classes. The reason for not experimenting with a number of classes that results in the considerable speedups is that available datasets containing this number of classes are very large, and were beyond the computational resources limitations of the authors of Keren et al. [57]. Therefore, while Chapter 3 asserts that it is plausible that the results scale further to a very large number of classes, further work should establish this empirically with the appropriate experiments.

Finally, future work may extend the scope the PoLS by applying it to other training mechanisms that do not involve loss functions, such as the training mechanism described in Chapter 2.

6.3 Low-quality supervision

Another aim of this work, as posed in the research question in Section 1.3, is to utilise the abundance of low-quality labels in neural network training. Specifically, we were interested in using large datasets of instance-level labels, to perform a detection task over a large number of classes. In chapter 4, the work that appears in the preprint Keren et al. [59] is described, addressing the above mentioned research goal. To avoid the dependency of the model on localisation labels such as bounding boxes, the detection task is considered in a weakly supervised setting, using instance-level labels alone. Further, to avoid the requirement of having a large amount of labels from a very large number of classes, the detection task is considered in the one-shot learning setting. Thus, the task of weakly supervised one-shot detection is considered, in both the audio and the computer vision domains.

The novel model presented for the weakly supervised one-shot detection task is conditioned on an exemplar of a given class and a larger target example, that may or may not contain an instance of the same class as the exemplar. The model consists of a Siamese neural network, extracting a representation from the exemplar and every location in the target image. A similarity score between the representations of the exemplar and the different locations is computed. Then, an attention mechanism attends to locations of interest and combines the locations' similarity scores into one instance-level score. Attention weights are used at test time to emit the localisation predictions, thereby circumventing the dependency on the existence of localisation labels. Further, the model only performs binary classification to determine whether a given target example contains an instance of the same class as a given exemplar. By not conditioning the model on class identities, the model is able to generalise across class boundaries and operate in a one-shot learning setting on classes unseen at training time. Experiments with character detection in images and query-by-example spoken term detection show that the proposed model manages to identify and localise instances of unseen classes and to outperform the baseline models.

However, the model has some limitations that should be addressed in future work on this topic. First, the image data that was used by Keren et al. [59] in the detection task consisted of square grids of character images. In this task, the exemplar was always of the same size as each location in the grid, and the number of possible locations for detection was limited. In the audio setting, even though the number of possible detection locations was not limited, the implicit assumption was that the exemplars (keywords) are of the same duration as their appearances in the target examples (utterances). Future work should address the weakly supervised one-shot detection task using additional datasets, where the exemplar and its appearances in the target example may be of different sizes, and the number of possible detection locations is not limited, such as object detection datasets of natural images. The

issue of detection across varying sizes may be addressed by expanding the set of possible locations with locations of different sizes, while still extracting a fixed length representation from all locations, as was done in Ren et al. [96].

Further, in the current work detection locations are emitted using the attention weights. Specifically, the detection location is set to be the location with the highest score of similarity to the exemplar. While this is a viable method for the task used in this work, this approach is not suitable for detection tasks where the target image may contain multiple instances of a given class. Further work may avoid this issue by allowing the model to emit a detection in all non-overlapping locations that have a similarity score that is above a certain threshold. Finally, experiments in Chapter 4 show that in many cases the proposed model correctly identifies whether an instance of the same class as the exemplar appears in the target image, but emits a detection location that is inaccurate. Future work should strive to further tune the detection locations, thus largely improving the total accuracy of the model.

6.4 Calibrated prediction intervals

The final research question of this work, presented in Section 1.4, is concerned with quantifying the prediction confidence in the regression setting, and specifically with emitting calibrated prediction intervals for neural network regressors. Chapter 5 presents the work of Keren et al. [53] following the above research question. The chapter describes how one may obtain calibrated prediction intervals in two stages. First, it is suggested to design the neural network regressor as a classifier over the binned output space. By doing so, we obtain a posterior probability distribution over the output space, instead of a single point prediction. Given the distribution over the output space, one may emit intervals in the output space that contain a given amount of probability mass. However, it is shown that the amount of probability mass that an interval contains does not correspond to the probability that this interval contains the true label. At the second stage, two novel calibration algorithms are presented. Both algorithms operate as post-processing of the model outputs, and allow emitting prediction intervals that correspond to any prescribed confidence level.

It is important to note, that while the algorithms presented in Chapter 5 allow emitting calibrated prediction intervals for a given confidence level, the chapter does not describe how an important complementary operation may be performed. In many applications, one may want to emit prediction intervals with a fixed length, and have an algorithm that emits the varying confidence level to these fixed prediction intervals. Future work may present algorithms for this complementary operation, based on the algorithms presented in Chapter 5, as we explain below. The empirical calibration algorithm presented in Chapter 5 finds the

amount of posterior probability mass that is equivalent to the prescribed confidence level. One may extend this to a complete mapping between the different amounts of probability mass to the confidence level of their resulting prediction intervals. One may use this mapping to emit the confidence level for any given interval.

In addition, future work should further develop and improve upon the proposed calibration algorithms. For example, the proposed algorithms always use symmetric prediction intervals around the expected real-valued prediction of the model. When the posterior probability distribution over the output space is not symmetric around this value, one may emit prediction intervals that contain the same amount of probability mass, but are a of a shorter length, therefore better. Moreover, different strategies for binning the output space should be considered. In this work, the output space was always binned into bins of equal length. When the labels distribution is not uniform, future work may want to explore binning the output space according to the percentiles of the label distribution.

Finally, the proposed algorithms rely on post-processing of the model outputs, when the model was only trained for best classification accuracy. In contrast, one may strive to have a model that learns to quantify prediction confidence, and is trained to emit the correct confidence level. In future work, one may attempt to overcome the challenges of this approach, that were described in Section 1.4.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Amiriparian, S., Pugachevskiy, S., Cummins, N., Hantke, S., Pohjalainen, J., Keren, G., and Schuller, B. W. (2017). CAST a database: Rapid targeted large-scale big data acquisition via small-world modelling of social media platforms. In *Proceedings of the International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 340–345, San Antonio, TX.

[3] Andreas, J. and Klein, D. (2015). When and why are log-linear models self-normalizing? In *Proceddings of NAACL HLT*, pages 244–249, Denver, CO.

[4] Ao, C. and Lee, H. (2018). Query-by-example spoken term detection using attention-based multi-hop networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6264–6268, Calgary, Canada.

[5] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, San Diago, CA.

[6] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of ICML*, pages 41–48, Montreal, Canada.

[7] Bengio, Y. and Senecal, J. (2008). Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19(4):713–722.

[8] Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. S. (2016). Fully-convolutional Siamese networks for object tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 850–865, Amsterdam, Netherlands.

[9] Bilen, H. and Vedaldi, A. (2016). Weakly supervised deep detection networks. In *Proceedings of CVPR*, pages 2846–2854, Las Vegas, NV.

[10] Bromley, J., Bentz, J. W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Säckinger, E., and Shah, R. (1993). Signature verification using a "Siamese" time delay neural network. *IJPRAI*, 7(4):669–688.

[11] Brummer, N. and Leeuwen, D. A. V. (2006). On calibration of language recognition scores. In *Proceedings of IEEE Odyssey - the Speaker and Language Recognition Workshop*, pages 1–8, San Juan, Puerto Rico.

[12] Burkhardt, F., Eckert, M., Johannsen, W., and Stegmann, J. (2010). A database of age and gender annotated telephone speech. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC*, Valletta, Malta.

[13] Chan, W., Jaitly, N., Le, Q. V., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964, Shanghai, China.

[14] Chen, G., Parada, C., and Sainath, T. N. (2015). Query-by-example keyword spotting using long short-term memory networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5236–5240, South Brisbane, Australia.

[15] Chen, W., Grangier, D., and Auli, M. (2016). Strategies for training large vocabulary neural language models. In *Proceedings of the 54nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1975–1985, Berlin, Germany.

[16] Cho, K., Courville, A., and Bengio, Y. (2015). Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, 17(11):1875–1886.

[17] Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Proceedings of NIPS*, pages 577–585, Montreal, Canada.

[18] Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(Dec):265–292.

[19] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *MCSS*, 2(4):303–314.

[20] Czarnecki, W. M., Jozefowicz, R., and Tabor, J. (2015). Maximum entropy linear manifold for learning discriminative low-dimensional representation. In *Proceedings of ECML PKDD*, pages 52–67.

[21] Dawid, A. P. (1982). The well-calibrated bayesian. *Journal of the American Statistical Association*, 77(379):605–610.

[22] Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Li, F. (2009). Imagenet: A large-scale hierarchical image database. In *Proceedings of CVPR*, pages 248–255, Miami, FL.

[23] Deng, J. and Schuller, B. (2012). Confidence Measures in Speech Emotion Recognition Based on Semi-supervised Learning. In *Proceedings of INTERSPEECH*, pages 2226–2229, Portland, OR.

[24] Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R. M., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1370–1380, Baltimore, MD.

[25] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). DeCAF: A deep convolutional activation feature for generic visual recognition. In *Proceedings of ICML*, pages 647–655, Beijing, China.

[26] Dong, X., Zheng, L., Ma, F., Yang, Y., and Meng, D. (2017). Few-shot object detection. *arXiv preprint arXiv:1706.08249*.

[27] Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

[28] Everingham, M., Gool, L. J. V., Williams, C. K. I., Winn, J. M., and Zisserman, A. (2010). The Pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338.

[29] Eyben, F., Weninger, F., Gross, F., and Schuller, B. (2013). Recent developments in opensmile, the munich open-source multimedia feature extractor. In *Proceedings of the 21st ACM International Conference on Multimedia*, pages 835–838, Barcelona, Spain.

[30] Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of ICML*, pages 1050–1059, New York, NY.

[31] Girshick, R. B. (2015). Fast R-CNN. In *Proceedings of ICCV*, pages 1440–1448, Santiago, Chile.

[32] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, Sardinia, Italy.

[33] Golik, P., Doetsch, P., and Ney, H. (2013). Cross-entropy vs. squared error training: A theoretical and experimental comparison. In *Proceedings of INTERSPEECH*, pages 1756–1760, Lyon, France.

[34] Gower, J. C. (1985). Measures of similarity, dissimilarity and distance. *Encyclopedia of Statistical Sciences*, 5:397–405.

[35] Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. (2017). Efficient softmax approximation for GPUs. In *Proceedings of ICML*, pages 1302–1310, Sydney, Australia.

[36] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *Proceedings of ICML*, pages 1321–1330, Sydney, Australia.

[37] Gupta, M. R., Bengio, S., and Weston, J. (2014). Training highly multiclass classifiers. *Journal of Machine Learning Research*, 15(1):1461–1492.

[38] Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 297–304, Chia Laguna Resort, Sardinia, Italy.

[39] Hariharan, B. and Girshick, R. B. (2017). Low-shot visual recognition by shrinking and hallucinating features. In *Proceedings of ICCV*, pages 3037–3046, Venice, Italy.

[40] Hazen, T. J., Shen, W., and White, C. M. (2009). Query-by-example spoken term detection using phonetic posteriorgram templates. In *Proceedings of the Workshop on Automatic Speech Recognition & Understanding (ASRU)*, pages 421–426, Merano/Meran, Italy.

[41] He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. (2017). Mask R-CNN. In *Proceedings of ICCV*, pages 2980–2988, Venice, Italy.

[42] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of CVPR*, pages 770–778, Las Vegas, NV.

[43] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

[44] Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40(1):185–234.

[45] Huang, Y., Wang, W., Wang, L., and Tan, T. (2013). Multi-task deep neural network for multi-label learning. In *Proceedings of the International Conference on Image Processing, ICIP*, pages 2897–2900, Melbourne, Australia.

[46] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of ICML*, pages 448–456, Lille, France.

[47] Janocha, K. and Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*.

[48] Jeatrakul, P., Wong, K. W., and Fung, C. C. (2010). Data cleaning for classification using misclassification analysis. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 14(3):297–302.

[49] Jiang, H. (2005). Confidence measures for speech recognition: a survey. *Speech Communication*, 45(4):455 – 470.

[50] Joder, C., Weninger, F., Wöllmer, M., and Schuller, B. (2012). The TUM cumulative DTW approach for the MediaEval 2012 spoken web search task. In *Proceedings of the MediaEval 2012 Workshop*.

[51] Kampa, K., Hasanbelliu, E., and Príncipe, J. C. (2011). Closed-form Cauchy-Schwarz PDF divergence for mixture of gaussians. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 2578–2585, San Jose, CA.

[52] Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? In *Proceedings of NIPS*, pages 5574–5584, Long Beach, CA.

[53] Keren, G., Cummins, N., and Schuller, B. W. (2018a). Calibrated prediction intervals for neural network regressors. *IEEE Access*, 6:54033–54041.

[54] Keren, G., Deng, J., Pohjalainen, J., and Schuller, B. (2016). Convolutional neural networks with data augmentation for classifying speakers' native language. In *Proceedings of INTERSPEECH*, pages 2393–2397, San Francisco, CA.

[55] Keren, G., Han, J., and Schuller, B. (2018b). Scaling speech enhancement in unseen environments with noise embeddings. In *In Proceedings of the CHiME Workshop on Speech Processing in Everyday Environments*, pages 25–29, Hyderabad, India.

[56] Keren, G., Sabato, S., and Schuller, B. (2017). Tunable sensitivity to large errors in neural network training. In *Proceedings of AAAI*, pages 2087–2093, San Francisco, CA.

[57] Keren, G., Sabato, S., and Schuller, B. (2018c). Fast single-class classification and the principle of logit separation. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 227–236, Singapore.

[58] Keren, G., Sabato, S., and Schuller, B. (2019). Analysis of loss functions for fast single-class classification. *Knowledge and Information Systems (KAIS)*. to appear.

[59] Keren, G., Schmitt, M., Kehrenberg, T., and Schuller, B. (2018d). Weakly supervised one-shot detection with attention Siamese networks. *arXiv preprint arXiv:1801.03329*.

[60] Keren, G. and Schuller, B. (2016). Convolutional RNN: An enhanced model for extracting features from sequential data. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 3412–3419, Vancouver, Canada.

[61] Khosravi, A., Nahavandi, S., Srinivasan, D., and Khosravi, R. (2015). Constructing optimal prediction intervals by using neural networks and bootstrap method. *IEEE Transactions on Neural Networks and Learning Systems*, 26(8):1810–1815.

[62] Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of ICLR*, San Diago, CA.

[63] Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *Proceedings of the ICML Deep Learning Workshop*, Lille, France.

[64] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

[65] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of NIPS*, pages 1097–1105, Lake Tahoe, NV.

[66] Kumar, M. P., Packer, B., and Koller, D. (2010). Self-paced learning for latent variable models. In *Proceedings of NIPS*, pages 1189–1197, Vancouver, Canada.

[67] Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J. R. R., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Duerig, T., and Ferrari, V. (2018). The open images dataset V4: unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982*.

[68] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

[69] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2016). Building machines that learn and think like people. *arXiv preprint arXiv:1604.00289*.

[70] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Proceedings of NIPS*, pages 6402–6413, Long Beach, CA.

[71] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

[72] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[73] Levin, E. and Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, 2:625–640.

[74] Li, H., Wang, X., and Ding, S. (2018). Research and development of neural network ensembles: a survey. *Artificial Intelligence Review*, 49(4):455–479.

[75] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755, Zurich, Switzerland.

[76] Malisiewicz, T., Gupta, A., and Efros, A. A. (2011). Ensemble of exemplar-SVMs for object detection and beyond. In *Proceedings of ICCV*, pages 89–96, Barcelona, Spain.

[77] Mark J. Huiskes, B. T. and Lew, M. S. (2010). New trends and ideas in visual concept detection: The MIR flickr retrieval evaluation initiative. In *Proceedings of the ACM International Conference on Multimedia Information Retrieval (MIR)*, pages 527–536, New York, NY.

[78] Maturana, D. and Scherer, S. (2015). VoxNet: A 3D convolutional neural network for real-time object recognition. In *Proceedings of IROS*, pages 922–928, Hamburg, Germany.

[79] McAuliffe, M., Socolof, M., Mihuc, S., Wagner, M., and Sonderegger, M. (2017). Montreal forced aligner: trainable text-speech alignment using Kaldi. In *Proceedings of INTERSPEECH*, pages 498–502, Stockholm, Sweden.

[80] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*, pages 3111–3119, Lake Tahoe, NV.

[81] Mnih, A. and Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. In *Proceedings ICML*, pages 1751–1758, Edinburgh, UK.

[82] Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, Bridgetown, Barbados.

[83] Naumov, A., Spokoiny, V., and Ulyanov, V. (2017). Bootstrap confidence sets for spectral projectors of sample covariance. *arXiv preprint arXiv:1703.00871*.

[84] Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

[85] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, Granada, Spain.

[86] Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. In *Proceedings of ICML*, pages 625–632, Bonn, Germany.

[87] Oord, A. V., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In *Proceedings of ICML*, pages 1747–1756, New York, NY.

[88] Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: An ASR corpus based on public domain audio books. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, South Brisbane, Australia.

[89] Parada, C., Sethy, A., and Ramabhadran, B. (2009). Query-by-example spoken term detection for OOV terms. In *Proceedings of the Workshop on Automatic Speech Recognition & Understanding (ASRU)*, pages 404–409, Merano/Meran, Italy.

[90] Partalas, I., Kosmopoulos, A., Baskiotis, N., Artières, T., Paliouras, G., Gaussier, É., Androutsopoulos, I., Amini, M., and Gallinari, P. (2015). LSHTC: a benchmark for large-scale text classification. *arXiv preprint arXiv:1503.08581*.

[91] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of ICML*, pages 1310–1318, Atlanta, GA.

[92] Platt, J. et al. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 10(3):61–74.

[93] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[94] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. In *Proceedings of CVPR*, pages 512–519, Columbus, OH.

[95] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of CVPR*, pages 779–788, Las Vegas, NV.

[96] Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. In *Proceedings of NIPS*, pages 91–99, Montreal, Canada.

[97] Rothe, R., Timofte, R., and Gool, L. V. (2015). DEX: Deep EXpectation of apparent age from a single image. In *Proceedings of the International Conference on Computer Vision Workshops (ICCVW)*, Santiago, Chile.

[98] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, 5:3.

[99] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

[100] Schuller, B., Steidl, S., Batliner, A., Burkhardt, F., Devillers, L., Müller, C. A., and Narayanan, S. S. (2010). The INTERSPEECH 2010 paralinguistic challenge. In *Proceedings of INTERSPEECH*, pages 2794–2797, Makuhari, Japan.

[101] Schuller, B., Steidl, S., Batliner, A., Hirschberg, J., Burgoon, J. K., Baird, A., Elkins, A., Zhang, Y., Coutinho, E., and Evanini, K. (2016). The INTERSPEECH 2016 Computational Paralinguistics Challenge: Deception & Sincerity. In *Proceedings of INTERSPEECH*, pages 2001–2005, San Francsico, CA.

[102] Silva, L. M., De Sa, J. M., Alexandre, L., et al. (2006). New developments of the Z-EDM algorithm. In *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 1067–1072, Jinan, China.

[103] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *Proceedings of ICLR*, San Diago, CA.

[104] Smith, M. R. and Martinez, T. (2011). Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 2690–2697, San Jose, CA.

[105] Socher, R., Lin, C. C., Ng, A. Y., and Manning, C. D. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of ICML*, pages 129–136, Bellevue, WA.

[106] Solla, S. A., Levin, E., and Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, 2(6).

[107] Surinta, O., Karaaba, M. F., Mishra, T. K., Schomaker, L., and Wiering, M. A. (2015). Recognizing handwritten characters with local descriptors and bags of visual words. In *Proceedings of the International Conference on Engineering Applications of Neural Networks (EANN)*, pages 255–264.

[108] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of ICML*, pages 1139–1147, Atlanta, GA.

[109] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of NIPS*, pages 3104–3112, Montreal, Canada.

[110] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of CVPR*, pages 1–9, Boston, MA.

[111] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceddings of CVPR*, pages 2818–2826, Las Vegas, NV.

[112] Tang, Y. (2013). Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*.

[113] Teh, E. W., Rochan, M., and Wang, Y. (2016). Attention networks for weakly supervised object localization. In *Proceedings of the British Machine Vision Conference 2016 (BMVC)*, York, UK.

[114] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*.

[115] Toshev, A. and Szegedy, C. (2014). DeepPose: Human pose estimation via deep neural networks. In *Proceedings of CVPR*, pages 1653–1660, Columbus, OH.

[116] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. In *Proceedings of the 9th ISCA Speech Synthesis Workshop*, page 125, Sunnyvale, CA.

[117] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of NIPS*, pages 6000–6010, Long Beach, CA.

[118] Vincent, E., Watanabe, S., Nugraha, A. A., Barker, J., and Marxer, R. (2017). An analysis of environment, microphone and data simulation mismatches in robust speech recognition. *Computer Speech & Language*, 46:535–557.

[119] Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. In *Proceedings of NIPS*, pages 3630–3638, Barcelona, Spain.

[120] Wang, C., Ren, W., Huang, K., and Tan, T. (2014). Weakly supervised object localization with latent category learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 431–445, Zurich, Switzerland.

[121] Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., and Xu, W. (2016). CNN-RNN: a unified framework for multi-label image classification. In *Proceedings of CVPR*, pages 2285–2294, Las Vegas, NV.

[122] Weintraub, M., Beaufays, F., Rivlin, Z., Konig, Y., and Stolcke, A. (1997). Neural-network based measures of confidence for word recognition. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 887–890, Munich, Germany.

[123] Wessel, F., Schluter, R., Macherey, K., and Ney, H. (2001). Confidence measures for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 9(3):288–298.

[124] Weston, J., Makadia, A., and Yee, H. (2013). Label partitioning for sublinear ranking. In *Proceedings of ICML*, pages 181–189, Atlanta, GA.

[125] Wöllmer, M., Al-Hames, M., Eyben, F., Schuller, B., and Rigoll, G. (2009a). A multidimensional dynamic time warping algorithm for efficient multimodal fusion of asynchronous data streams. *Neurocomputing*, 73(1):366–380.

[126] Wöllmer, M., Eyben, F., Keshet, J., Graves, A., Schuller, B., and Rigoll, G. (2009b). Robust Discriminative Keyword Spotting for Emotionally Colored Spontaneous Speech Using Bidirectional LSTM Networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3949–3952, Taipei, Taiwan.

[127] Wöllmer, M., Eyben, F., Schuller, B., and Rigoll, G. (2010). Spoken Term Detection with Connectionist Temporal Classification: A Novel Hybrid CTC-DBN Decoder. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5274–5277, Dallas, TX.

[128] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of ICML*, pages 2048–2057, Lille, France.

[129] Yao, S., Zhao, Y., Shao, H., Zhang, A., Zhang, C., Li, S., and Abdelzaher, T. (2018). RDeepSense: Reliable deep mobile computing models with uncertainty estimations. *ACM Interactive Mobile Wearable Ubiquitous Technology*, 1(4):173:1–173:26.

[130] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Proceedings of NIPS*, pages 3320–3328, Montreal, Canada.

[131] Yu, D., Li, J., and Deng, L. (2011). Calibration of confidence measures in speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(8):2461–2473.

[132] Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.