Brandenburgische Technische Universität Cottbus - Senftenberg

Faculty of Mathematics, Computer Science, Physics, Electrical Engineering and Information Technology

Institute of Computer Science

COMPUTER SCIENCE REPORTS

Report 01/21

June 2021

Generating CQQL Conditions from Classifying CNNs

Ingo Schmitt

Computer Science Reports
Brandenburg University of Technology Cottbus - Senftenberg
ISSN: 1437-7969
https://doi.org/10.26127/BTUOpen-5550

Send requests to: BTU Cottbus - Senftenberg
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Ingo Schmitt
ingo.schmitt@b-tu.de
http://www.b-tu.de/fg-dbis

Generating CQQL Conditions from Classifying CNNs

Computer Science Reports
01/21
June 2021

Brandenburg University of Technology Cottbus - Senftenberg

Faculty of Mathematics, Computer Science, Physics, Electrical Engineering and Information Technology

Institute of Computer Science

Computer Science Reports
Brandenburg University of Technology Cottbus - Senftenberg
Institute of Computer Science

BTU Cottbus - Senftenberg
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Research Groups:	Headed by:
Computer Engineering	Prof. Dr. M. Hübner
Computer Network and Communication Systems	Prof. Dr. O. Hohlfeld
Data Structures and Software Dependability	Prof. Dr. M. Heiner
Database and Information Systems	Prof. Dr. I. Schmitt
IT-Security	Prof. Dr. A. Panchenko
Programming Languages and Compiler Construction	Prof. Dr. P. Hofstedt
Software and Systems Engineering	N.N.
Theoretical Computer Science	Prof. Dr. K. Meer
Graphics Systems	Prof. Dr. D. Cunningham
Wireless Systems	Prof. Dr. P. Langendörfer
Distributed Systems and Operating Systems	Prof. Dr. J. Nolte
Internet-Technology	Prof. Dr. G. Wagner

CR Subject Classification (1998): G.3

Printing and Binding: BTU Cottbus - Senftenberg

ISSN: 1437-7969
https://doi.org/10.26127/BTUOpen-5550

Generating CQQL Conditions from Classifying CNNs

Ingo Schmitt

Brandenburgische Technische Universität Cottbus-Senftenberg
Institut für Informatik
Konrad-Wachsmann-Allee 5, 03046 Cottbus
schmitt@b-tu.de

July 5, 2021

Abstract

Convolutional neural networks are often successfully used for classification problems. Usually, a huge number of weights need to be learnt by use of training data. However, the learnt weights give no insight how the cnn really works. Thus, a cnn can be seen as a black box solution. In our approach we develop a method to generate a commuting quantum query language (cqql) condition from a sample derived from a given cnn or from training input. The query language cqql is inspired by quantum logic and its conditions obey the rules of a Boolean algebra. The evaluation of a cqql condition provides values from the unit interval $[0, 1]$ and establishes therefore an elegant bridge between logic and a cnn. The underlying assumption is that a condition (a logic expression) gives much more understanding than pure cnn weights. Furthermore, the rich theory of Boolean algebra can be used for manipulating logic expressions. After extracting a cqql condition from a cnn or its training data we can use logic as a way to predict classes alternatively to a cnn.

1 Introduction

The general classification problem is given by following components:

- Let $D = \{o\}$ be a set of objects. Every objects is characterized by its values for n given attributes: $o = (o_1, \ldots, o_n)$.

- Let $Cl = \{class_1, \ldots, class_k\}$ be a given set of k classes.
- Let m be a mapping from D to Cl, that is $m : D \to Cl$. The mapping is typically not explicitly known on all objects and is our learning target.
- Let $O \subset D$ be a subset of D where for every object the class membership is known. That is, we have $M = \{(o, m(o)|o \in O\}$.
- Let $TR \subset M$ be a set of training objects and $TE = M \setminus TR$ be the test set.

The problem is now to find a mapping function $cl : D \to Cl$, called a classifier, from given training objects TR. The mapping should make a good prediction on TE, that is, the target is to fulfill $\forall(o, m(o)) \in TE : cl(o) = m(o)$. The accuracy of a classifier cl can be measured by the fraction of correct classified objects:

$$Accuracy = \frac{|\{(o, m(o)) \in TE | m(o) = cl(o)\}|}{|TE|}$$

In the following, we reformulate the introduced k-class problem to k one-class problems for $i = 1, \ldots, k$:

$$\begin{array}{rcl} m_i : D & \to & \{0, 1\} \\ m_i(o) & \mapsto & \begin{cases} 1 & \text{if } m(o) = class_i \\ 0 & \text{otherwise.} \end{cases} \end{array}$$

We try to learn the corresponding classifiers $cl_i : D \to \{0, 1\}$ with high accuracy. Next, we will focus on $i = 1$ and will write abbreviatedly m and cl instead of m_1 and cl_1.

One way to construct a classifier cl is to use deep learning in combination with a convolutional neural network [1]. A cnn can be seen as a function approximator converging to m. In this work, we are interested in the classification results of a given cnn. We assume a cnn, which is already trained by use of TR. Typically, a cnn computes the classification result by performing calculations on the input values influenced by a huge number of learnt weights. Unlike other classifiers, e.g. decision trees, a cnn does not give any insight into the performed mapping. As result, for a cnn, we cannot completely rely on its output on $D \setminus O$. In critical applications, e.g. medicin, control of air planes, however, reliability and explainability are essential. Furthermore, we cannot check fairness of the classification and cannot perform a comparison with possibly available background information.

In order to overcome these problems, we extract a Boolean condition from a cnn because a logical formula gives us the demanded insights. The starting point is a set $T = \{(x, y)\}$ with $T \subseteq M$ or a sample from a cnn with $y = cl(x) = cnn(x)$ for some x.

As preparation step we map all x from T to $x' \in [0,1]^n$ by use of a monotonic mapping function. As result we can interprete x' as a tuple of n logical cqql values. The mapping function can be linear $x'_i = (x_i - min_i)/(\max_i - \min_i)$ or be based on an exponential function $x'_i = e^{-d_i}$ where $d_i = distance(x_i, v_i)$ is the distance to a well selected value v_i. In the following, we assume the x values from T to be an element of the hyper cube $[0,1]^n$.

2 Commuting Quantum Query Language and Operand Weighting

The language cqql was introduced in [2, 3, 4]. Syntactically and semantically, a cqql expression is very near to a Boolean expression. As starting point we assume an object o consisting of values for n attributes: $o = (o_1, \ldots, o_n)$. We define a Boolean atomic predicate as a function on an attribute which returns a value from $\{0,1\}$ for a given attribute value. In contrast, we define a similarity atomic predicate as a function on an attribute which returns a value from $[0,1]$ for a given attribute value. The value 1 can be interpreted as true and the value 0 as false. Let *Atoms* be a set of atomic conditions with the restriction, that for each attribute no more than one similarity predicate is defined. As discussed in [2] this restriction refers to the term *commuting* projectors in quantum mechanics and is important to obtain a Boolean lattice as sublattice of quantum logic. A cqql expression is recursively defined as:

- Every element $e \in Atoms$ is a cqql expression.
- Negation: If e is a cqql expression then $(\neg e)$ is a cqql expression.
- Conjunction and disjunction: If e_1 and e_2 are cqql expressions then $(e_1 \wedge e_2)$ and $(e_1 \vee e_2)$ are cqql expressions.
- Exclusive disjunction: Let e_1 and e_2 be two cqql expressions. If it can be proven by applying rules of the Boolean algebra that $e_1 \wedge e_2$ refers to *false* for all objects o then $(e_1 \dot{\vee} e_2)$ is a cqql expression.

As mentioned above, the semantics of cqql obeys the rules of the Boolean algebra whereas the evaluation of a cqql expression yields a value from $[0,1]$ instead of *true* and *false*. Only a cqql expression in a special syntactical form (cqql normal from, see [2, 3, 4]) can be arithmetically evaluated. Let $attrib(e)$ be the set of attributes involved by a possibly nested expression e. The normal form requires that for each conjunction $e_1 \wedge e_2$ and for each disjunction $e_1 \vee e_2$ (but not

for exclusive disjunctions) the attribute sets are disjoint: $attrib(e_1) \cap attrib(e_2) = \emptyset$. If a cqql expression is not in cqql normal form then it can be syntactically transformed into that normal form by applying laws from the Boolean algebra.

A cqql expression e in the required normal form is evaluated against an object o by recursively defining $eval : cqql \times D \to [0,1]$:

- Atomic predicate: If e is an atomic predicate then $eval(e,o) \in [0,1]$ returns the result from applying the corresponding function on o.
- Negation: $eval(\neg e, o) = 1 - eval(e,o)$.
- Conjunction: $eval(e_1 \wedge e_2, o) = eval(e_1,o) * eval(e_2,o)$.
- Disjunction: $eval(e_1 \vee e_2, 0) = eval(e_1,0) + eval(e_2,o) - eval(e_1,o) * eval(e_2,o)$.
- Exclusive disjunction: $eval(e_1 \dot{\vee} e_2, o) = eval(e_1,o) + eval(e_2,o)$.

In the following, If the object o is uniquely given from the context then we will drop it and simply write $eval(e)$ instead of $eval(e,o)$.

We now extend the expressive power of a normalized cqql expression by introducing *weighted conjunction* $(e_1 \wedge_{\theta_1,\theta_2} e_2)$ and *weighted disjunction* $(e_1 \vee_{\theta_1,\theta_2} e_2)$. Values of weights θ_1, θ_2 are values out of $[0,1]$. A weight 0 means that the corresponding operand has no influence and a weight 1 equals the unweighted case (full influence). We regard every θ_i as an atomic similarity predicate returning as evaluation a constant value out of $[0,1]$. For evaluation, we can easily map weighted conjunction and disjunction to their unweighted cases:

$$\begin{aligned} (e_1 \wedge_{\theta_1,\theta_2} e_2) &\to ((e_1 \vee \neg\theta_1) \wedge (e_2 \vee \neg\theta_2)) \\ (e_1 \vee_{\theta_1,\theta_2} e_2) &\to ((e_1 \wedge \theta_1) \vee (e_2 \wedge \theta_2)) \end{aligned}$$

If no weight predicate occurs multiply then one can easily show that the weight mapping yields an expression in cqql normal form that can be directly evaluated as described above.

So far, we introduced binary conjunction and disjunction. Since the evaluation of a cqql expression obeys Boolean laws (commutativity, associativity), we can write n-ary disjunction and n-ary conjunction as short form for a nested binary operations.

For classification we need a threshold $th \in [0,1]$ for a cqql expression e in order to obtain a classifier:

$$cl_e(o) = \begin{cases} 1 & \text{if } eval(e,o) \geq th \\ 0 & \text{otherwise.} \end{cases}$$

Thus, for a certain classification problem, we need to find a good cqql expression (condition) together with a well chosen threshold value.

From the laws of the Boolean algebra we know that every expression e can be expressed in the complete disjunctive normal form, that is, every expression is equivalent to a subset of 2^n minterms. We assume for every of the n object attributes exactly one atomic predicate c_j. The minterm subset relation can be expressed by use of minterm weights $\theta_i \in \{0, 1\}$:

$$e = \bigvee_{i=1}^{2^n} minterm_{i,\theta_i} = \bigvee_{i=1}^{2^n} minterm_i \wedge \theta_i \tag{1}$$

$$minterm_i = \bigwedge_{j=1}^{n} c_{i,j} \tag{2}$$

$$c_{ij} = \begin{cases} c_j & \text{if } (i-1) \;\&\; 2^{j-1} > 0 \\ \neg c_j & \text{otherwise.} \end{cases} \tag{3}$$

Symbol '&' stands for bitwise `and`.

Notice that the disjunction of two different complete minterms is always exclusive. Thus, e is in cqql normal form and its evaluation against object o yields

$$eval(e, o) = \sum_{i=1}^{2^n} \theta_i \prod_{j=1}^{n} c_{i,j}^o \tag{4}$$

$$c_{ij}^o = \begin{cases} eval(c_j, o) & \text{if } (i-1) \;\&\; 2^{j-1} > 0 \\ 1 - eval(c_j, o) & \text{otherwise.} \end{cases} \tag{5}$$

3 Extracting a CQQL Condition

From the given set $T = \{(x, y)\}$ we extract a good classifier cl_e based on a cqql condition e. A good classifier is a classifier with high accuracy. Therefore, we try to maximize the accuracy for expression (1) depending on the minterm weights θ_i.

At first, we have to adapt the formula for accuracy to our cqql scenario. That is, we regard $y, eval(e, x) \in [0, 1]$ as evaluation results of logic expressions where a value near or equal to 1 corresponds to *true* and a value near or equal to 0 to *false*. For a given (x, y)-pair and the evaluation of a cqql condition e we can distinguish four cases:

	y	$\neg y$
$eval(e, x)$	$y \wedge e$	$\neg y \wedge e$
$eval(\neg e, x)$	$y \wedge \neg e$	$\neg y \wedge \neg e$

The green cases on the diagonal (correct alarm, correct reject) refer to the correct results. Accuracy can now be measured as sum over the two correct cases over all pairs $(x, y) \in T$:

$$
\begin{aligned}
accuracy(e) &= \sum_{(x,y)\in T} y * eval(e,x) + \sum_{(x,y)\in T} (1-y)(1-eval(e,x)) \\
&= \sum_{(x,y)\in T} (eval(e,x)\cdot(2y-1)+1-y) \\
&= \sum_{(x,y)\in T} eval(e,x)\cdot(2y-1) + \sum_{(x,y)\in T} (1-y) \\
&= \sum_{(x,y)\in T} \left(\sum_{i=1}^{2^n} \theta_i \cdot \prod_{j=1}^{n} c_{ij}^x\right)\cdot(2y-1) + \sum_{(x,y)\in T} (1-y) \\
&= \sum_{i=1}^{2^n} \theta_i \sum_{(x,y)\in T} \left((2y-1)\cdot \prod_{j=1}^{n} c_{ij}^x\right) + \sum_{(x,y)\in T} (1-y)
\end{aligned}
$$

We see, that accuracy is linearly dependent on the minterm weights θ_i for fixed T-pairs. The first derivative provides the constant gradient on θ_i:

$$
\frac{\partial accuracy(e)}{\partial \theta_i} = \sum_{(x,y)\in T} (2y-1)\cdot \prod_{j=1}^{n} c_{ij}^x.
$$

Because of $y \in \{0,1\}$ we can reformulate:

$$
\frac{\partial accuracy(e)}{\partial \theta_i} = \sum_{(x,1)\in T} \prod_{j=1}^{n} c_{ij}^x - \sum_{(x,0)\in T} \prod_{j=1}^{n} c_{ij}^x.
$$

For maximizing accuracy a minterm weight θ_i should get the value 1 if $\frac{\partial accuracy(e)}{\partial \theta_i} > 0$ and 0 otherwise:

$$
\theta_i = \begin{cases} 1 & \text{if } \sum_{(x,1)\in T} \prod_{j=1}^{n} c_{ij}^x > \sum_{(x,0)\in T} \prod_{j=1}^{n} c_{ij}^x \\ 0 & otherwise. \end{cases} \tag{6}
$$

In other words, for the decision whether a minterm should be active or not it is sufficient to compare the impact of positive training data $(x,1) \in T$ against the impact of the negative training data $(x,0) \in T$ on that minterm. Be aware, that the decision depends on the number of positive training objects. Therefore,

$$
\frac{|\{(x,1)\in T\}|}{|T|}
$$

should reflect the probability of class occurrence.

After applying our decision rule in (6) we obtain the formula:

$$e = \bigvee_{\theta_i=1} \bigwedge_{j=1}^{n} c_{i,j}.$$

Next, we have to find the threshold value th. The value should lie in interval $[\min_1, \max_0]$ where $\min_1 = \min_{(x,1)\in T} eval(e,x)$ and $\max_0 = \max_{(x,0)\in T} eval(e,x)$. In order to find a threshold which maximizes discrete accuracy acc we use the training data:

$$best\_th = \underset{th=eval(e,x),(x,\_)\in T}{\arg\max} acc(e,th).$$

The term $acc(e,th)$ refers to the classical discrete accuracy based on the set T and the discrete classifier cl_e determined by cqql condition e and threshold th. Of course, $acc(e,th)$ is typically smaller than 1. The value reflects how strong the connection between input and output can be described by a logic expression and that depends on the given scenario.

4 Processing of CQQL Conditions

The extracted cqql condition e can now be presented to the user and gives an understanding of the logical connection between input and class decision. Using laws of Boolean algebra, the user can manipulate condition e and check if a property p holds in e. It is enough to test $p \wedge \neg e = false$ to see if property p holds.

A convenient way of presenting e is to transform it into the reduced conjunctive normal form. Reduced means that the maxterms do not necessarily contain the atomic predicates for all n attributes:

$$e = \bigwedge_i maxterm_i.$$

Every maxterm is a disjunction which can be easily transformed to a logical clause (implication with head and tail), e.g.:

$$a \vee b \vee \neg c \vee \neg d = a \vee b \vee \neg(c \wedge d) = c \wedge d \rightarrow a \vee b.$$

For detecting a class on an input all clauses must be fullfilled. Thus, as Boolean expression, every clause can be presented and analyzed independently from the others. However, please notice, that condition e in reduced conjunctive normal form is not necessarily in cqql normal form. As consequence, e cannot be directly evaluated. Even worse, in cqql, a clause cannot be interpreted independently from the others.

The solution of that problem is straightforward: we introduce a local threshold for every clause and obtain as evaluation result either *true* (value 1) or *false* (value 0). After that, every clause can be interpreted alone and they altogether can be evaluated as Boolean logic expression where no cqql normal form is required anymore.

The new problem now is how to find the local thresholds th_i for every maxterm (disjunction). We designed a greedy algorithm based on maxterm scores (maxterm evaluation results) on training data T. Instead of maximizing globally with immense costs we perform small local optimization steps in order to find the th_i values for maximizing discrete accuracy. The algorithms consists of three steps:

1. *Initialization*: We construct a sorted score list from T for each maxterm and initialize local thresholds and further variables.
2. *Loop and best maxterm list*: For every local optimization step the best maxterm list needs to be selected. The corresponding local threshold is set to its next score value and the variables are updated accordingly. All local threshold updates are collected in a list.
3. *Best threshold values*: Find the local thresholds by scanning the threshold list.

The initialization step is given in detail by following substeps:

1.1 Sort for each $maxterm_i$ evaluation scores $score_{ij}$ from training objects tr_j, if $score_{ij} \in [min_1^i, max_0^i]$, into $list_i$ (maxterm lists)

1.2 Set $th_i = min_1^i$ for all maxterms

1.3 Let x_{00} be the number of training objects tr_j where $\exists i : score_{ij} < min_1^i$

1.4 Remove all x_{00}-objects from all maxterm lists

1.5 Let x_{11} be the number of non-x_{00} training objects tr_j where $\exists i : score_{ij} > max_0^i$

1.6 For all remaining training objects tr_j (neither x_{00} nor x_{11}): if $y_j = 1$ then increment x_{11}, otherwise increment x_{10}

1.7 Let $Acc = (x_{00} + x_{11})/anz * 2$ where $anz = |TR|$

The second step of iteratively finding the best maxterm list for threshold updates is given by following substeps:

2.1 For each minterm list i run through its elements until next score $\neq th_i$ is found

2.1.1 Let $gain_i$ be the number of seen object where $y = 1$

2.1.2 Let $lost_i$ be the number of seen object where $y = 0$

2.2 Let $listi$ be the maxterm list where $gain_i - lost_i$ is at highest

2.3 Set th_i to the next higher score in $listi$ and put it into the threshold list

2.4 Remove seen objects tr_j with $score_{ij} < th_i$ from all maxterm lists

2.5 Update: $x_{00} = x_{00} + gain_i, x_{10} = x_{10} - gain_i, x_{01} = x_{01} + lost_i$, $x_{11} = x_{11} - lost_i$ and Acc

2.6 If $x_{10} > 0$ loop to step 2.1 for finding next threshold update

As third step take from all seen threshold updates from the threshold list the thresholds with highest accuracy value.

5 Efficient Extraction of a CQQL Condition

Following our approach described above for n attributes we have to consider an exponential number of minterms (2^n). Thus, the condition extraction is feasible only for a small number of attributes. Instead of considering minterms n atomic predicates we suggest to examine minterms with less ($k \leq n$) atomic predicates. The number of sets of k attributes is given by the binomial coefficient $\binom{n}{k}$. For every of the $\binom{n}{k}$ attribute subsets there are 2^k variants (negation of an atomic predicate or not). Therefore, we obtain $\binom{n}{k} * 2^k$ minterms which can be higher than 2^n. For example, let $\{A, B, C\}$ be the attribute set and $k = 2$. Then we obtain 12 minterms:

$$\binom{3}{2} = 3 \underbrace{\left\{ \begin{array}{cccc} A \wedge B & A \wedge \neg B & \neg A \wedge B & \neg A \wedge \neg B \\ A \wedge C & A \wedge \neg C & \neg A \wedge C & \neg A \wedge \neg C \\ B \wedge C & B \wedge \neg C & \neg B \wedge C & \neg B \wedge \neg C \end{array} \right.}_{2^2=4}$$

The idea of a fast approximation algorithm is to randomly generate k-minterms and to check if they are valid against the training data T. In that way, the condition becomes stepwisely better and better in terms of accuracy. A k-minterm is valid if the difference of the positive and negative impact is higher than a given threshold th.

Algorithm for extracting reduced $|minterms^r| = k$ with $k \leq n$:

1. Let $th > 0$ be a threshold for deciding the relevance of a reduced minterm and $clist$ an empty list
2. Initialize $\varphi_+ = false, \varphi_- = false$, we will collect all relevant formulas (positive/negative) incrementally in φ_+, φ_-

3. Generate randomly a reduced minterm c out of the $\binom{n}{k} * 2^k$ ones
4. If $c \notin clist$ and $\varphi_+ \not\models c$ and $\varphi_- \not\models c$ then:
   (a) Perform $test_c = \sum_1 c - \sum_0 c$ on training data
   (b) Append c to $clist$
   (c) If $test_c > th$ then $\varphi_+ = \varphi_+ \vee c$
   (d) If $test_c < -th$ then $\varphi_- = \varphi_- \vee c$
5. If not abort then go to step 3
6. Return result $\varphi_+ \wedge \neg\varphi_-$

For the abortion different variants are possible:

- after a given number of iterations
- after certain time is consumed
- after accuracy reaches a certain threshold
- after a certain number of consecutive failed hits

6 Conclusion

In our work we developed a way to extract a cqql condition from a pre-trained convolutional neural network which maximizes accuracy. In that way, a certain part of the mapping semantics can be described in logic. A logic expression can be much better understood than network weights. We transformed the extracted condition into the conjunctive normal form in order to isolate single clauses. Furthermore, we discussed the computational complexity of our approach and developed an approximation algorithm which allows us to extract a condition as trade-off between extraction efficiency and effectiveness in terms of accuracy.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[2] Ingo Schmitt. *Quantum query processing: unifying database querying and information retrieval*. Citeseer, 2006.

[3] Ingo Schmitt. "Qql: A db&ir query language". In: *The VLDB journal* 17.1 (2008), pp. 39–56.

[4] Ingo Schmitt and Daniel Baier. “Logic based conjoint analysis using the commuting quantum query language”. In: *Algorithms from and for Nature and Life*. Springer, 2013, pp. 481–489.