Why Max and Average Poolings are Optimal in Convolutional Neural Networks[⋆]

Ahnaf Farhan, Olga Kosheleva[0000−0002−1244−1650], and Vladik Kreinovich[0000−0002−1244−1650]

University of Texas at El Paso, El Paso, TX 79968, USA
afarhan@miners.utep.edu, olgak@utep.edu, vladik@utep.edu

Abstract. In many practical situations, we do not know the exact relation between different quantities; this relation needs to be determined based on the empirical data. This determination is not easy – especially in the presence of different types of uncertainty. When the data comes in the form of time series and images, many efficient techniques for such determination use algorithms for training convolutional neural network. As part of this training, such networks "pool" several values corresponding to nearby temporal or spatial points into a single value. Empirically, the most efficient pooling algorithm consists of taking the maximum of the pooled values; the next optimal is taking the arithmetic mean. In this paper, we provide a theoretical explanation for this empirical optimality.

Keywords: Data processing under uncertainty · Convolutional neural networks · Pooling · Max pooling · Optimization

1 Formulation of the Problem

Need for data processing. The main objectives of science and engineering are to describe the world, to predict the future behavior of the world's systems, and to find the best way to improve this behavior.

The current state of the world is described by numerical values of different physical quantities. Some of these values can be directly measured – e.g., we can measure the distance to a nearby city, the temperature, humidity, and wind speed at different Earth locations. Other quantities are difficult (or even impossible) to measure directly. For example, it is difficult to directly measure the distance to a nearby star, the temperature on the surface of the Sun, etc. Since we cannot measure these quantities y directly, we have to determine them indirectly: namely, we measure the values of easier-to-measure quantities $x_1, \ldots, x_n$ which are related to the desired quantity y, and then use the measurement results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ to compute an estimate $\widetilde{y}$ for the desired quantity y. The corresponding computations form an important case of *data processing*.

⋆ This work was partially supported by the US National Science Foundation via grant HRD-1242122 (Cyber-ShARE Center of Excellence).

Similar computations are needed to estimate the future values of the quantities of interest and the values of necessary control based on the known current value of related quantities.

Need for machine learning. In some cases, we know the exact relation $y = f(x_1, \ldots, x_n)$ between the desired quantity y and the measured quantities x_1, $\ldots$, x_n. For example, we know the exact equations of celestial mechanics, so we can predict the future locations of planets, the time of solar and lunar eclipses, etc.

In many other cases, however, we do not know this relation. In such cases, we need to determine the corresponding relation from the available data. Namely, in several situations $k = 1, \ldots, K$, we measure (or otherwise estimate) the values $x_1^{(k)}$, $\ldots$, $x_n^{(k)}$, $y^{(k)}$ of all the quantities, and then use this data to find a dependence $f(x_1, \ldots, x_n)$ for which $y^{(k)} \approx f\left(x_1^{(k)}, \ldots, x_n^{(k)}\right)$ for all k. Algorithms for reconstructing the dependence from empirical data are known as *machine learning.*

At present, the most efficient machine learning algorithms are the algorithms of *deep neural networks*; see, e.g., [1–3].

It is important to take uncertainty into account. In the ideal situation, when all the values are known exactly, it is often relative easy to find the desired dependence. For example, if it turns out that all the values corresponding to the dependence $y = f(x_1)$ fit a straight line, we conclude that the dependence is linear.

In reality, measurements are never absolutely accurate. There is always measurement uncertainty, because of which the measurement results are, in general, somewhat different from the actual values of the corresponding quantities. This uncertainty makes machine learning much more complicated. For example, even if the actual dependence is linear, we corresponding pairs $\left(\widetilde{x}_1^{(k)}, \widetilde{y}^{(k)}\right)$ do not exactly lie on the same straight line.

Need for convolutional neural networks. In many practical situations, the available data comes in terms of *time series* – when we have values measured at equally spaced time moments – on in terms of an *image* – when we have data corresponding to a grid of spatial locations. Neural networks for processing such data are known as *convolutional neural networks.*

Need for pooling. It is possible to decrease the distortions caused by measurement errors if we take into account that usually, the actual values at nearby points in time or space are close to each other. As a result, instead of using the measurement-distorted value at each point, we can take into account that values at nearby points are also results of measuring practically the same quantity – and combine ("pool together") these values into a single more accurate estimate.

Which pooling techniques work better: empirical results. In principle, we can have many different pooling algorithms. It turns out that empirically, in general, the most efficient pooling algorithm is *max-pooling*, when we take the maximum $\max(a_1, \ldots, a_m)$ of m neighboring values $a_1, \ldots, a_m$ [3]. The next efficient is *average pooling* [3], when we take the arithmetic average

$$a = \frac{a_1 + \ldots + a_m}{m}.$$

What we do in this paper. In this paper, we provide a theoretical explanation for this empirical observation: namely, we prove that max and average poolings are indeed optimal.

2 Analysis of the Problem

What is pooling: towards a precise definition. We start with m values $a_1, \ldots, a_m$, and we want to generate a single value a that represents all these values.

In the case of arithmetic average, pooling means that we select a value a for which $a_1 + \ldots + a_m = a + \ldots + a$ (m times). In general, pooling means that we select some combination operation $*$ and we then select the value a for which $a_1 * \ldots * a_m = a * \ldots * a$ (m times). For example, if, as a combination operation, we select $\max(a, b)$, then the corresponding condition

$$\max(a_1, \ldots, a_n) = \max(a, \ldots, a) = a$$

describes the max-pooling.

From this viewpoint, selecting pooling means selecting an appropriate combination operation. Thus, selecting the optimal pooling means selecting the optimal combination operation.

Natural properties of a combination operation. The combination operation transforms two non-negative values – such as intensity of an image at a given location – into a single non-negative value. The result of applying this operation should not depend on the order in which we combine the values. Thus, we should have $a * b = b * a$ (commutativity) and $a * (b * c) = (a * b) * c$ (associativity).

What does it mean to have an optimal pooling? Optimality means that on the set of all possible combination operations, we have a preference relation $\preceq$, so that $A \preceq B$ means that the operation B is better than (or of the same quality as) the operation A.

This relation should be transitive: if C is better than B and B is better than A, then C should be better than A.

An operation A is optimal if it is better than (or of the same quality as) any other operation B: $B \preceq A$.

For some preference relations, we may have several different optimal combination operations. We can then use this non-uniqueness to optimize something else. For example, if there are several different combination operations with the best average-case accuracy of machine learning results, we can select, among them, the one for which the average computation time is the smallest possible. If after this, we still get several optimal operations, we can use the remaining non-uniqueness to optimize yet another criterion – until we get a *final* criterion, for which there is only one optimal combination operation.

Scale-invariance. Numerical values of a physical quantity depend on the choice of a measuring unit – and on the choice of a starting point. For example, if we replace meters with centimeters, for the same physical quantity, the numerical quantity is multiplied by 100. In general, if we replace the original unit with a unit which is λ times smaller, then all numerical values get multiplied by 100.

It is reasonable to require that the preference relation should not change if we simply change the measuring unit. Let us describe this requirement in precise terms. If, in the original units, we had the operation $a*b$, then, in the new units, the operation will take the following form:

– first, we transform the value a and b into the new units, so we get $a' = \lambda \cdot a$ and $b' = \lambda \cdot b$;
– then, we combine the new numerical values, getting $(\lambda \cdot a) * (\lambda \cdot b)$;
– finally, we re-scale the result to the original units, getting $R_\lambda(*)$ defined as

$$aR_\lambda(*)b \stackrel{\text{def}}{=} \lambda^{-1} \cdot ((\lambda \cdot a) * (\lambda \cdot b).$$

It therefore makes sense to require that is $* \preceq *'$, then for every $\lambda > 0$, we get $R_\lambda(*) \preceq R_\lambda(*')$.

Shift-invariance. The numerical values also change if we change the starting point for measurements. For example, when measuring intensity, we can measure the actual intensity of an image, or we can take into account that there is always some noise $a_0 > 0$, and use the noise-only level a_0 as the new starting point. In this case, instead of each original value a, we get a new numerical value $a' = a - a_0$ for describing the same physical quantity.

If we apply the combination operation in the new units, then in the old units, we get a slightly different result; namely,

– first, we transform the value a and b into the new units, so we get $a' = a - a_0$ and $b' = b - a_0$;
– then, we combine the new numerical values, getting $(a - a_0) * (b - a_0)$;
– finally, we re-scale the result to the original units, getting $S_{a_0}(*)$ defined as

$$aS_{a_0}(*)b \stackrel{\text{def}}{=} (a - a_0) * (b - a_0) + a_0.$$

It makes sense to require that the preference relation not change if we simply change the starting point: so if $* \preceq *'$, then for every a_0, we get $S_{a_0}(*) \preceq S_{a_0}(*')$.

Weak version of shift-invariance. Alternatively, we can have a weaker version of this "shift-invariance" if we require that shifts in a and b imply a possibly different shift in $a * b$, i.e., if we shift both a and b by a_0, then the value $a * b$ is shifted by some value $f(a_0)$ which is, in general, different from a_0.

Now, we are ready to formulation our results.

3 Definitions and Results

Definition 1. *By a* combination operation*, we mean a commutative, associative operation $a*b$ that transforms two non-negative real numbers a and b into a non-negative real number $a*b$.*

Definition 2. *By an* optimality criterion*, we need a transitive reflexive relation $\preceq$ on the set of all possible combination operations.*

- *We say that a combination operation $*_{\rm opt}$ is* optimal *with respect to the optimality criterion $\preceq$ if $* \preceq *_{\rm opt}$ for all combination operations $*$.*
- *We say that the optimality criterion is* final *if there exists exactly one combination operation which is optimal with respect to this criterion.*

Definition 3.

- *We say that an optimality criterion is* scale-invariant *if for all $\lambda > 0$, $* \preceq *'$ implies $R_\lambda(*) \preceq R_\lambda(*')$, where $aR_\lambda(*)b \stackrel{\rm def}{=} \lambda^{-1}\cdot((\lambda\cdot a)*(\lambda\cdot b))$.*
- *We say that an optimality criterion is* shift-invariant *if for all a_0, $* \preceq *'$ implies $S_{a_0}(*) \preceq S_{a_0}(*')$, where $aS_{a_0}(*)b \stackrel{\rm def}{=} ((a-a_0)*(b-a_0))+a_0$.*
- *We say that an optimality criterion is* weakly shift-invariant *if for every a_0, there exists a value $f(s_0)$ such that $* \preceq *'$ implies $W_{a_0}(*) \preceq W_{a_0}(*')$, where*

$$aW_{a_0}(*)b \stackrel{\rm def}{=} ((a-a_0)*(b-a_0))+f(a_0).$$

Proposition 1. *For every final, scale- and shift-invariant optimality criterion, the optimal combination operation has one of the following two forms: $a*b = \min(a,b)$ or $a*b=\max(a,b)$.*

Since the max combination operation corresponds to max-pooling, this result explains why max-pooling is empirically the best combination operation.

Proposition 2. *For every final, scale-invariant and weakly shift-invariant optimality criterion, the optimal combination operation has one of the following four forms: $a*b=0$, $a*b=\min(a,b)$, $a*b=\max(a,b)$, or $a*b=a+b$.*

Since the addition combination operation corresponds to average-based pooling, this result explains why max-pooling and average-pooling are empirically the best combination operations.

4 Proofs

General part of the two proofs. Let us first prove that in Proposition 1, the optimal combination operation $*_{\rm opt}$ is itself scale- and shift-invariant, i.e., $R_\lambda(*_{\rm opt}) = *_{\rm opt}$ for all $\lambda > 0$ and $S_{a_0}(*_{\rm opt}) = *_{\rm opt}$ for all a_0.

Let us prove this for scale-invariance; for shift-invariance, the proof is similar. The fact that $*_{\rm opt}$ is optimal means that $* \preceq *_{\rm opt}$ for all $*$. In particular, $R_{\lambda^{-1}}(*) \preceq *_{\rm opt}$ for all $*$. Due to scale-invariance of the optimality criterion, this implies that $* \preceq R_\lambda(*_{\rm opt})$ for all $*$. Thus, the combination operation $R_\lambda(*_{\rm opt})$ is also optimal. But since the optimality criterion is final, there is only one optimal combination operation, hence $R_\lambda(*_{\rm opt}) = *_{\rm opt}$. Scale-invariance is proven. Shift-invariance is proven similarly.

For Proposition 2, we can similarly prove that the optimal combination operation is scale-invariant and weakly shift-invariant, i.e., that $R_\lambda(*_{\rm opt}) = *_{\rm opt}$ for all $\lambda > 0$ and $W_{a_0}(*_{\rm opt}) = *_{\rm opt}$ for all a_0.

Proof of Proposition 1.

1°. Let $a * b$ be the optimal combination operation. We have shown that this operation is scale-invariant and shift-invariant. Let us prove that it has one of the above two forms.

For every pair (a, b), we can have three different cases: $a = b$, $a < b$, and $a > b$. Let us consider them one by one.

2°. Let us first consider the case when $a = b$.

Let us denote $v \stackrel{\text{def}}{=} 1 * 1$. From scale-invariance with $\lambda = 2$, from $1 * 1 = v$, we get $2 * 2 = 2v$. From shift-invariance with $s = 1$, from $1 * 1 = v$, we get $2 * 2 = v + 1$. Thus, $2v = v + 1$, hence $v = 1$, and $1 * 1 = 1$.

For $a > 0$, by applying scale-invariance with $\lambda = a$ to the formula $1 * 1 = 1$, we get $a * a = a$.

For $a = 0$, if we denote $c \stackrel{\text{def}}{=} 0 * 0$, then, by applying shift-invariance with $s = 1$ to $0 * 0 = c$, we get $1 * 1 = c + 1$. Since we already know that $1 * 1 = 1$, this means that $c + 1 = 1$ and thus, that $c = 0$, i.e., that $0 * 0 = 0$.

So, for all $a \ge 0$, we have $a * a = a$. In this case, $\min(a, a) = \max(a, a) = a$, so we have $a * a = \min(a, a)$ and $a * a = \max(a, a)$.

3°. Let us first consider the case when $a < b$. In this case, $b - a > 0$.

Let us denote $t \stackrel{\text{def}}{=} 0 * 1$. By applying scale-invariance with $\lambda = b - a > 0$ to the formula $0 * 1 = t$, we conclude that

$$0 * (b - a) = (b - a) \cdot t. \tag{1}$$

Now, by applying shift-invariance with $s = a$ to the formula (1), we conclude that

$$a * b = (b - a) \cdot t + a. \tag{2}$$

To find possible values of t, let us take into account that the combination operation should be associative. This means, in particular, that for all possible triples a, b, and c for which we have $a < b < c$, we must have

$$a * (b * c) = (a * b) * c. \tag{3}$$

Since $b < c$, by the formula (2), we have

$$b * c = (c - b) * t + b.$$

Since $t \geq 0$, we have $b * c \geq b$ and thus, $a < b * c$. So, to compute $a * (b * c)$, we can also use the formula (2), and get

$$a * (b * c) = (b * c - a) \cdot t + a = ((c - b) \cdot t + b) \cdot t + a = c \cdot t^2 + b \cdot (t - t^2) + a. \quad (4)$$

Let us restrict ourselves to the case when $a * b < c$. In this case, the general formula (2) implies that

$$(a * b) * c = (c - a * b) \cdot t + a * b = (c - ((b - a) \cdot t + a)) \cdot t + (b - a) \cdot t + a,$$

so

$$(a * b) * c = c \cdot t + b \cdot (t - t^2) + a \cdot (1 - t)^2. \quad (5)$$

Due to associativity, formulas (4) and (5) must coincide for all a, b, and c for which $a < b < c$ and $c > a * b$. Since these two linear expressions must be equal for all sufficiently large values of c, the coefficients at c must be equal, i.e., we must have $t = t^2$. From $t = t^2$, we conclude that $t - t^2 = t \cdot (1 - t) = 0$, so either $t = 0$ or $1 - t = 0$ (in which case $t = 1$).

If $t = 0$, then the formula (2) has the form $a * b = a$, i.e., since $a < b$, the form $a * b = \min(a, b)$.

If $t = 1$, then the formula (2) has the form $a * b = (b - a) + a = b$, i.e., since $a < b$, the form $a * b = \max(a, b)$.

4°. If $a > b$, then, by commutativity, we have $a * b = b * a$, where now $b < a$. So:

– if $t = 0$, then, due to Part 3 of this proof, we have $b * a = \min(b, a)$; since $a * b = b * a$ and since clearly $\min(a, b) = \min(b, a)$, we can conclude that $a * b = \min(a, b)$ for $a > b$ as well;
– if $t = 1$, then, due to Part 3 of this proof, we have $b * a = \max(b, a)$; since $a * b = b * a$ and since clearly $\max(a, b) = \max(b, a)$, we can conclude that $a * b = \max(a, b)$ for $a > b$ as well.

So, either we have $a*b = \min(a, b)$ for all a and b, or we have $a*b = \max(a, b)$ for all a and b. The proposition is proven.

Proof of Proposition 2.

1°. Let $a * b$ be the optimal combination operation. We have proven that this operation is scale-invariant and weakly shift-invariant – which means that $a*b = c$ implies $(a + s) * (b + s) = c + f(s)$. Let us prove that the optimal operation $*$ has one of the above four forms.

1°. Let us first prove that $0 * 0 = 0$.

Indeed, let s denote $0 * 0$. Due to scale-invariance, $0 * 0 = s$ implies that $(2 \cdot 0) * (2 \cdot 0) = 2s$, i.e., that $0 * 0 = 2s$. So, we have $s = 2s$, hence $s = 0$ and $0 * 0 = 0$.

2°. Similarly, if we denote $v \overset{\text{def}}{=} 1 * 1$, then, due to scale-invariance with $\lambda = a$, $1 * 1 = v$ implies that $a * a = v \cdot a$ for all a.

On the other hand, due to weak shift-invariance with $a_0 = a$, $0*0 = 0$ implies that $a * a = f(a)$. Thus, we conclude that $f(a) = v \cdot a$.

2°. Let us now consider the case when $a < b$ and, thus, $b - a > 0$.

Let us denote $t \overset{\text{def}}{=} 0 * 1$. From scale-invariance with $\lambda = b - a$, from $0 * 1 = t \geq 0$, we get $0 * (b - a) = t \cdot (b - a)$. From weak shift-invariance with $a_0 = a$, we get $a * b = t \cdot (b - a) + v \cdot a$, i.e.,

$$a * b = t \cdot b + (v - t) \cdot a. \tag{6}$$

Similarly to the proof of Proposition 1, to find possible values of t, let us take into account that the combination operation should be associative. This means, in particular, that for all possible triples a, b, and c for which we have $a < b < c$, we must have

$$a * (b * c) = (a * b) * c.$$

Since $b < c$, by the formula (6), we have

$$b * c = t \cdot c + (v - t) \cdot b.$$

3°. We know that $t \geq 0$. This means that we have either $t > 0$ and $t = 0$.

4°. Let us first consider the case when $t > 0$. In this case, for sufficiently large c, we have $b * c > a$. So, by applying the formula (6) to a and $b * c$, we conclude that

$$a * (b * c) = t \cdot (b * c) + (v - t) \cdot a = t^2 \cdot c + t \cdot (v - t) \cdot b + (v - t) \cdot a. \tag{7}$$

For sufficient large c, we also have $a * b < c$. In this case, the general formula (6) implies that

$$(a * b) * c = (t \cdot b + (v - t) \cdot a) * c = t \cdot c + t \cdot (v - t) \cdot b + (v - t)^2 \cdot a. \tag{8}$$

Due to associativity, formulas (7) and (8) must coincide for all a, b, and c for which $a < b < c$, $c > a * b$, and $b * c > a$. Since these two linear expressions must be equal for all sufficiently large values of c, the coefficients at c must be equal, i.e., we must have $t = t^2$.

From $t = t^2$, we conclude that $t - t^2 = t \cdot (1 - t) = 0$. Since we assumed that $t > 0$, we must have $t - 1 = 0$, i.e., $t = 1$.

The coefficients at a must also coincide, so we must have $v - t = (v - t)^2$, hence either $v - t = 0$ or $v - t = 1$. In the first case, the formula (6) becomes $a * b = b$, i.e., $a * b = \max(a, b)$ for all $a \leq b$. Since the operation $*$ is commutative, this equality is also true for $b \leq a$ and is, thus, true for all a and b.

In the second case, the formula (6) becomes $a * b = a + b$ for all $a \leq b$. Due to commutativity, this formula holds for all a and b.

5°. Let us now consider the case when $t = 0$. In this case, the formula (6) takes the form $a * b = (v - t) \cdot a$.

Here, $a * b \geq 0$, thus $v - t \geq 0$. If $v - t = 0$, this implies that $a * b = 0$ for all $a \leq b$ and thus, due to commutativity, for all a and b.

Let us now consider the remaining case when $v - t > 0$. In this case, if $a < b < c$, then for sufficiently large c, we have $a * b < c$, hence

$$(a * b) * c = (v - t) \cdot (a * b) = (v - t) \cdot ((v - t) \cdot a) = (v - t)^2 \cdot a.$$

On the other hand, here $b * c = (v - t) \cdot b$. So, for sufficiently large b, we have $(v - t) \cdot b > a$, thus

$$a * (b * c) = (v - t) \cdot a.$$

Due to associativity, we have $(v - t)^2 \cdot a = (v - t) \cdot a$, hence $(v - t)^2 = v - t$ and, since $v - t > 0$, we have $v - t = 1$. In this case, the formula (6) takes the form $a * b = a = \min(a, b)$ for all $a \le b$. Thus, due to commutativity, we have $a * b = \min(a, b)$ for all a and b.

We have thus shown that the combination operation indeed has one of the four forms. Proposition 2 is therefore proven.

References

1. C. Baral, O. Fuentes, and V. Kreinovich, “Why Deep Neural Networks: A Possible Theoretical Explanation”, In: M. Ceberio and V. Kreinovich (eds.), *Constraint Programming and Decision Making: Theory and Applications*, Springer Verlag, Berlin, Heidelberg, 2018, pp. 1–6.
2. A. Gholamy, J. Parra, V. Kreinovich, O. Fuentes, and E. Anthony, “How to Best Apply Deep Neural Networks in Geosciences: Towards Optimal ‘Averaging’ in Dropout Training”, In: J. Watada, S. C. Tan, P. Vasant, E. Padmanabhan, and L. C. Jain (eds.), *Smart Unconventional Modelling, Simulation and Optimization for Geosciences and Petroleum Engineering*, Springer Verlag, to appear.
3. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Leaning*, MIT Press, Cambridge, Massachusetts, 2016.