Deep Learning (Partly) Demystified

Vladik Kreinovich

Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79968, USA
vladik@utep.edu
http://www.cs.utep.edu/vladik

1. Overview

- Successes of deep learning are partly due to appropriate selection of activation function, pooling functions, etc.
- Most of these choices have been made based on empirical comparison and heuristic ideas.
- In this talk, we show that:
  - many of these choices – and the surprising success of deep learning in the first place
  - can be explained by reasonably simple and natural mathematics.

2. Traditional Neural Networks: A Brief Reminder

- To explain deep neural networks, let us first briefly recall the motivations behind traditional ones.
- In the old days, computers were much slower.
- This was a big limitation that prevented us from solving many important practical problems.
- As a result, researchers started looking for ways to speed up computations.
- If a person has a task which takes too long for one person, a natural idea is to ask for help.
- Several people can work on this task in parallel – and thus, get the result faster; similarly:
  - if a computation task takes too long,
  - a natural idea is to have several processing units working in parallel.

3. Traditional Neural Networks (cont-d)

- In this case:
  - the overall computation time is just
  - the time that is needed for each of the processing unit to finish its sub-task.
- To minimize the overall time, it is therefore necessary to make these sub-tasks as simple as possible.
- In data processing, the simplest possible functions to compute are linear functions.
- However, if we only have processing units that compute linear functions, we will only compute linear functions.
- Indeed, a composition of linear functions is always linear.
- Thus, we need to supplement these units with some nonlinear units.

4. Traditional Neural Networks (cont-d)

- In general, the more inputs, the more complex (and thus longer) the resulting computations.
- So, the fastest possible nonlinear units are the ones that compute functions of one variable.
- So, our ideal computational device should consist of:
  - linear (L) units and
  - nonlinear units (NL) that compute functions of one variable.
- These units should work in parallel:
  - first, all the units from one layer will work,
  - then all units from another layer, etc.
- The fewer layers, the faster the resulting computations.
- One can prove that 1- and 2-layer schemes do not have a universal approximation property.

5. Traditional Neural Networks (cont-d)

- One can also prove that 3-layer neurons already have this property.
- There are two possible 3-layer schemes: L-NL-L and NL-L-NL.
- The first one is faster, since it uses slower nonlinear units only once.
- In this scheme, first, each unit from the first layer applies a linear transformation to the inputs $x_1, \ldots, x_n$:

$$z_k = \sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}.$$

- The values w_{ki} are known as *weights*.
- In the next NL layer, these values are transformed into $y_k = s_k(y_k)$, for some nonlinear functions $s_k(z)$.

6. Traditional Neural Networks (cont-d)

- Finally, in the last (L) layer, the values y_k are linearly combined into the final result

$$y=\sum_{k=1}^{K} W_k\cdot y_k-W_0=\sum_{k=1}^{K} W_k\cdot s_k\left(\sum_{i=1}^{n} w_{ki}\cdot x_i-w_{k0}\right)-W_0.$$

- This is exactly the formula that describes the traditional neural network.

- In the traditional neural network, usually, all the NL neurons compute the same function – sigmoid:

$$s_k(z)=\frac{1}{1+\exp(-z)}.$$

7. Why Go Beyond Traditional Neural Networks

- Traditional neural networks were invented when computers were reasonably slow.
- This prevented computers from solving important practical problems.
- For these computers, computation speed was the main objective.
- As we have just shown, this need led to what we know as traditional neural networks.
- Nowadays, computers are much faster.
- In most practical applications, speed is no longer the main problem.
- But the traditional neural networks:
  - while fast,
  - have limited accuracy of their predictions.

8. The More Models We Have, the More Accurately We Can Approximate

- As a result of training a neural network, we get the values of some parameters for which
  - the corresponding models
  - provides the best approximation to the actual data.
- Let a denote the number of parameters.
- Let b the number of bits representing each parameter.
- Then, to represent all parameters, we need $N = a \cdot b$ bits.
- Different models obtained from training can be described by different N-bit sequences.
- In general, for N bits, there are 2^N possible N-bit sequences.
- Thus, we can have 2^N possible models.

9. The More Models We Have (cont-d)

- In these terms, training simply means selecting one of these 2^N possible models.
- If we have only one model to represent the actual dependence, this model will be a very lousy description.
- If we can have two models, we can have more accurate approximations.
- In general, the more models we have, the more accurate representation we can have.
- We can illustrate this idea on the example of approximating real numbers from the interval $[0, 1]$.
- If we have only one model – e.g., the value $x = 0.5$, then we approximate every other number with accuracy 0.5.

10. The More Models We Have (cont-d)

- If we can have 10 models, then we can take 10 values $0.05, 0.15, \ldots, 0.95$.
- The first value approximates all the numbers from the interval $[0, 0.1]$ with accuracy 0.05.
- The second value approximates all the numbers from the interval $[0, 1, 0.2]$ with the same accuracy, etc.
- By selecting one of these values, we can approximate any number from $[0, 1]$ with accuracy 0.05.

11. How Many Models Can We Represent with a Traditional Neural Network

- Let us consider a traditional neural network with K neurons.
- Each neuron k is characterized by several weights W_k and w_{ki}.
- Let b denote the number of bits needed to describe all the weights corresponding to a single neuron.
- Then, overall, to describe all possible bit sequences resulted from training, we need $N = K \cdot b$ bits.
- As we mentioned, we can have 2^N different binary sequences of length N.
- So, at first glance, one may think that we can thus represent 2^N different models.
- However, the actual number of models is much smaller.

12. How Many Models (cont-d)

- If we swap two neurons, the resulting functions will not change:

$$f(x_1, \ldots, x_n) = \sum_{k=1}^{K} W_k \cdot s\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) - W_0.$$

- Indeed, the sum does not change if we swap two of added numbers.
- Similarly, if instead of swapping two neurons, we apply any permutation, we get the exact same model.
- For K neurons, there are $K!$ possible permutations.
- Thus, $K!$ different binary sequences represent the same model.
- So, by using N bits, instead of 2^N possible models, we can only have $\frac{2^N}{K!}$ possible models.

13. How Can We Achieve Better Accuracy: The Main Idea Behind Deep Learning

- The more models we can represent, the more accurate will be the resulting approximation; so:
  - when the overall number of bits is fixed – e.g., by the ability of our computers,
  - the only way to increase the number of models is to decrease $K!$, i.e., to decrease K.
- In the traditional neural networks, all the neurons are, in effect, in one layer – known as the hidden layer.
- The only way to decrease K is to make the number of neurons in each layer much smaller.
- This means that instead of placing the neurons into a single layer, we place then in many layers.
- We now have several layers – the construction is *deep*.

14. Which Activation Function Should We Use for Deep Learning?

- To answer this question, we need to recall that usually, we process the values of physical quantities.
- The numerical values of physical quantities depend on:
  - what measuring unit we use, and
  - for some quantities like temperature or time – what starting point we select for the measurement.
- If we change a measuring unit to a one which is λ times smaller, then all numerical values get multiplied by λ.
- So, instead of the original numerical value x, we get a new numerical value $x' = \lambda \cdot x$.
- For example, 2.5 feet becomes $12 \cdot 2.5 = 30$ inches.

15. Selecting an Activation Function (cont-d)

- Similarly:
  - if we replace the original starting point with the new point which is x_0 units before,
  - then each numerical value x is replaced by a new numerical value $x' = x + x_0$.
- We want to select an activation function $s(x)$ that would not depend on the choice of a measuring unit.
- In other words, we want to make sure that:
  - if $y = s(x)$ and we select a new measuring unit, i.e., switch to new numerical values

$$x' = \lambda \cdot x \text{ and } y' = \lambda \cdot y,$$

  - then for these new values x' and y', we will have the exact same dependence: $y' = s(x')$.

16. Selecting an Activation Function (cont-d)

- Substituting the expressions $x' = \lambda \cdot x$ and $y' = \lambda \cdot y$ into this formula, we conclude that $\lambda \cdot y = s(\lambda \cdot x)$.
- Here, $y = s(x)$, so we conclude that

$$s(\lambda \cdot x) = \lambda \cdot s(x)$$

  for all possible x and $\lambda > 0$.

- For $x = 1$, we conclude that $s(\lambda) = \lambda \cdot s(1)$.
- Let us denote $s(1)$ by c_+, and rename λ into z.
- Then, we conclude that for all $z > 0$, we get

$$s(z) = c_+ \cdot z.$$

- For $x = -1$, we conclude that $s(-\lambda) = \lambda \cdot s(-1)$.
- Let us denote $-s(-1)$ by c_- (so that $s(-1) = -c_-$) and $-\lambda$ by z (so that $\lambda = -z$).

17. Selecting an Activation Function (cont-d)

- Then, for all negative values z, we have

$$s(z) = (-c_-) \cdot (-z) = c_- \cdot z.$$

- Thus, we conclude that the activation function $s(z)$ should have the following *piecewise linear* form:
  - for $z > 0$, we have $s(z) = c_+ \cdot z$;
  - for $z < 0$, we have $s(z) = c_- \cdot z$.
- *Comment.* We must have $c_+ \neq c_-$; indeed:
  - otherwise, the function $s(z)$ would be linear, and
  - we know that with linear functions, we can only describe linear dependencies.

18. What Activation Function Is Actually Used in Deep Learning? Why

- To uniquely determine a piecewise linear function, we need to select two real numbers: c_+ and c_-.
- The simplest possible real numbers is 0 and 1.
- Thus, the simplest possible piecewise linear function has the form:
  - for $z > 0$, we have $s(z) = z$;
  - for $z < 0$, we have $s(z) = 0$.
- In other words, $s(z) = \max(z, 0)$.
- This function is known as *rectified linear function*, it is actually used in deep learning.

19. It Does Not Matter Which Piecewise Linear Activation Function to Use

- Indeed, the output of each neuron is linearly combined with other signals anyway.
- And any piecewise linear function can be represented as a linear combination of $\max(z, 0)$ and z:

$$s(z) = c_- \cdot z + (c_+ - c_-) \cdot \max(z, 0).$$

- Indeed:
  - for $z > 0$, the right-hand side is equal to

$$c_- \cdot z + c_+ \cdot 0 = c_- \cdot z,$$

  - for $z < 0$, the right-hand side is equal to

$$c_- \cdot z + (c_+ - c_-) \cdot z = (c_- + (c_+ - c_-)) \cdot z = c_+ \cdot z.$$

20. Why Cannot We Require Shift-Invariance Instead of Scale-Invariance?

- We mentioned that the numerical value of a physical quantity changes:
  - when we change the measuring unit *and*
  - when we change the starting point.
- However, we only considered invariance with respect to changing the unit (*scale-invariance*).
- What if we consider invariance with respect to changing the starting point (*shift-invariance*)?
- We want to make sure that:
  - when $y = s(x)$
  - then for $x' = x + x_0$ and $y' = y + x_0$, we will have

$$y' = s(x').$$

21. Shift-Invariance (cont-d)

- Substituting the expressions for x' and y' into the formula $y' = s(x')$, we get $y + x_0 = s(x + x_0)$.
- Here, $s(x) = y$, so we have $s(x + x_0) = s(x) + x_0$ for all possible values x and x_0.
- In particular, for $x = 0$, we get $s(x_0) = s(0) + x_0$.
- Renaming $s(0)$ as a and x_0 as z, we conclude that

$$s(z) = z + a.$$

- This is a linear function – thus, such neurons cannot describe any non-linear process.

22. Need for Pooling

- Often, we have a lot of data points to process.
- For example, even for a not very good 1000 by 1000 picture, we have 1,000,000 pixels.
- So to process such an image, we need to process 1,000,000 numbers.
- In a traditional neural network, we could use as many neurons as needed.
- However, in a deep neural network, there are only a few neurons in the first layer.
- Thus, before we start processing, we need to combine several input values into one.
- A similar procedure can also be applied at a later stage.
- This operation of combining several values into one is known as *pooling*.

23. Which Pooling Operation Shall We Use?

- Let us consider the case when we pool two values a and b into a single value c.
- Let us denote the resulting value c by $p(a, b)$.
- Of course, the pooling should not depend on the order, i.e., we should have $p(a, b) = p(b, a)$.
- In other words, the pooling operation should be *commutative*.
- It is reasonable to require that the result of pooling will not change if we:
  - change the measuring unit or
  - change the starting point for measurement.
- If $c = p(a, b)$, then $c' = p(a', b')$, where $a' = \lambda \cdot a$, $b' = \lambda \cdot b$, and $c' = \lambda \cdot c$.

24. Which Pooling Operation to Use (cont-d)

- If $c = p(a,b)$, then $c' = p(a',b')$, where $a' = a + a_0$, $b' = b + a_0$, and $c' = c + a_0$.
- From the first requirement:
  - substituting the expressions $a' = \lambda \cdot a$, $b' = \lambda \cdot b$, and $c' = \lambda \cdot c$ into the formula $c' = p(a',b')$,
  - we conclude that $\lambda \cdot c = p(\lambda \cdot a, \lambda \cdot b)$.
- Here, $c = p(a,b)$, so $p(\lambda \cdot a, \lambda \cdot b) = \lambda \cdot p(a,b)$.
- From the second requirement:
  - substituting the expressions $a' = a + a_0$, $b' = b + a_0$, and $c' = c + a_0$ into the formula $c' = p(a',b')$,
  - we conclude that $c + a_0 = p(a + a_0, b + a_0)$.
- Here, $c = p(a,b)$, so $p(a + a_0, b + a_0) = p(a,b) + a_0$.
- Let us use the resulting formulas to find the value $p(x,y)$ for all possible pairs (x,y).

25. Which Pooling Operation to Use (cont-d)

- Without losing generality, we can assume that $x < y$.
- Then, substituting $a = 0$, $a_0 = x$, and $b = y - x$ into the formula $p(a + a_0, b + a_0) = p(a, b) + a_0$, we get:

$$p(x, y) = p(0, y - x) + x.$$

- Substituting $\lambda = y - x$, $a = 0$, and $b = 1$ into the formula $p(\lambda \cdot a, \lambda \cdot b) = \lambda \cdot p(a, b)$, we get:

$$p(0, y - x) = (y - x) \cdot p(0, 1).$$

- Substituting this expression into the formula $p(x, y) = p(0, y - x) + x$ and denoting $p(0, 1)$ by α, we get:

$$p(x, y) = x + \alpha \cdot (y - x) = \alpha \cdot y + (1 - \alpha) \cdot x =$$

$$\alpha \cdot \max(x, y) + (1 - \alpha) \cdot \min(x, y).$$

26. Pooling Four Values

- Once we learn how to pool two values, we can pool four values easily:
  - divide the four values into two pairs,
  - pool results within each pair, and then
  - pool the two pooling results into a single value.
- It is reasonable to require that the result not depend on how we divide 4 values into pairs.
- Let us consider the values 0, 1, 1, and 2.
- First, we combine 0 with 1 and 1 with 2.
- Pooling 0 and 1 results in
$$\alpha \cdot 1 + (1 - \alpha) \cdot 0 = \alpha.$$
- Pooling 1 and 2 results in
$$\alpha \cdot 2 + (1 - \alpha) \cdot 1 = 2\alpha + 1 - \alpha = 1 + \alpha.$$

27. Pooling Four Values (cont-d)

- Here always $1+\alpha$ is larger than α.
- So combining the results α and $1+\alpha$ leads to

$$\alpha \cdot (1+\alpha) + (1-\alpha) \cdot \alpha = \alpha + \alpha^2 + \alpha - \alpha^2 = 2\alpha.$$

- What if we instead combine 1 with 1 and 0 with 2?
- Combining 1 with 1 results in

$$\alpha \cdot 1 + (1-\alpha) \cdot 1 = 1.$$

- Pooling 0 with 2 results in

$$\alpha \cdot 2 + (1-\alpha) \cdot 0 = 2\alpha.$$

- The resulting of pooling the resulting too values 1 and 2α depends on which of the two values is larger.
- If $2\alpha \geq 1$, i.e., if $\alpha \geq 0.5$, then we get

$$\alpha \cdot (2\alpha) + (1-\alpha) \cdot 1 = 2\alpha^2 + 1 - \alpha.$$

28. Pooling Four Values (cont-d)

- In this case, the desired equality is $2\alpha^2 + 1 - \alpha = 2\alpha$, i.e., $2\alpha^2 - 3\alpha + 1 = 0$.
- One can easily check that this quadratic equation has two solutions: $\alpha = 0.5$ and $\alpha = 1$.
- If $2\alpha \le 1$, i.e., if $\alpha \le 0.5$, then we get

$$\alpha \cdot 1 + (1 - \alpha) \cdot 2\alpha = \alpha + 2\alpha - 2\alpha^2 = 3\alpha - 2\alpha^2.$$

- In this case, the desired equality is $3\alpha - 2\alpha^2 = 2\alpha$, i.e.,

$$2\alpha^2 - \alpha = 0.$$

- One can easily check that this quadratic equation has two solutions: $\alpha = 0$ and $\alpha = 0.5$.
- So, we have three options: $\alpha = 0$, $\alpha = 0.5$, and $\alpha = 1$.

29. Pooling Four Values (cont-d)

- If $\alpha = 0$, then the pooling formula takes the form
$$p(x, y) = \min(x, y).$$
- If $\alpha = 0.5$, then the pooling formula takes the form $p(x, y) = 0.5 \cdot y + 0.5 \cdot x$, i.e., of the arithmetic average
$$p(x, y) = \frac{x + y}{2}.$$
- If $\alpha = 1$, then the pooling formula takes the form
$$p(x, y) = \max(x, y).$$
- We get three operations: minimum, maximum, and arithmetic average.
- These are indeed the ones which work most successfully in deep learning.

30. Sensitivity of Deep Learning: Phenomenon

- A problem with deep learning is that its results are often too sensitive to minor changes in the inputs.
- For example, changing a few pixels in a picture of a cat may result in this picture being misclassified as a dog.
- In practice, signals often come with noise.
- It is not good that a small noise can ruin the results.

31. Sensitivity of Deep Leaning: An Explanation

- Each neuron is affected by the noise.
- It can take the original noise level δ and amplify it to a higher level $c \cdot \alpha$ for some $c > 1$.
- In deep learning, we have several (L) layers.
- In the first layer, each neuron amplifies the noise level δ to $c \cdot \delta$.
- Neurons in the second layer amplify it even more, to

$$c \cdot (c \cdot \delta) = c^2 \cdot \delta.$$

- After the third layer, we get $c^3 \cdot \delta$, etc.
- After all L layers, we get $c^L \cdot \delta$.
- The exponential function c^L grows very fast with L.
- So, not surprisingly, we get a much higher noise level than for the traditional neural networks.

32. How to Deal with Sensitivity of Deep Learning

- To train a traditional neural network, we feed it with actually observed patterns $\left(x_1^{(p)}, \ldots, x_n^{(p)}, y^{(p)}\right)$.
- Then, we find the values of the corresponding weights that match all these patterns.
- As a result, the trained network usually works well:
  - not only for the original patterns, but
  - also for modified versions of these patterns – e.g., when we add some noise.
- For deep learning, we do not have automatic success on noised patterns.

33. How to Deal with Sensitivity (cont-d)

- So, to achieve such success, it is reasonable:
  - to artificially add noise to the patterns and
  - to add such simulated-noise modification to the original patterns when training a network.
- We can also add noise to the inputs.
- This idea seems to work reasonably well.

34. Acknowledgments

This work was supported in part by the National Science Foundation via grants:

- 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science),
- and HRD-1242122 (Cyber-ShARE Center of Excellence).