Chapter theory 3 solution

Artem Puzanov

2016-09-27

A note: this chapter is gargantuan in size, so A LOT of exercises and problems. Not my fault:)

1 activation derivative proof

Intuition Taking derivatives from exponents rather often ends in a kind of semi-recursive definitions, which come in handy for computational purposes.

Mathematical notation We want to prove that:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ This is a rather simple derivation, using chain rule:

$$\sigma'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

Let's take out $\sigma(z)$

$$\sigma'(z) = \frac{1}{(1+e^{-z})}\frac{e^{-z}}{1+e^{-z}}$$

$$\sigma'(z) = \sigma(z)\frac{e^{-z}+1-1}{1+e^{-z}}$$

$$\sigma'(z) = \sigma(z)(1 - \frac{1}{1+e^{-z}})$$

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Q.E.D.

2 entropy function exercies

2.1 entropy form memorization

Intuition It's easy to remember specific form, if you follow two rules: it's symmetrical, and logariphm is not defined at zero. Symmetricity gives: $xln(y)+(1-x)ln(1-y)$, and whether x or y is a or y is easy to decide, if we remember that true value can take values in $[0,1]$ interval, and activation can't reach 0 or 1. This means that only activation value can be under ln. No mathematical proof needed here.

2.2 proof of entropy minimization for regression-like problems

Intuition Nothing especially hard here, usual function-minimization proof. Take the first derivative, solve it relative to zero, show that this is minimizing solution.

Mathematical notation First let's show that $y = a$ is a solution for:

$$\frac{\partial C}{\partial a} = 0$$

$$\frac{\partial C}{\partial a} = -\frac{1}{n}\sum(\frac{y}{a} - \frac{1-y}{1-a}) = 0$$

Let's drop $-\frac{1}{n}$, and concetrate on the expression under sum:

$$\frac{y}{a} - \frac{1-y}{1-a} = 0$$

$$\frac{y}{a} = \frac{1-y}{1-a}$$

$$y(1-a) = (1-y)a$$

$$a = y$$

As neurons in the final layer are not intersecting, this holds true for all j Now let's show that the $\frac{\partial^2 C}{\partial^2 a} > 0$ for $y = a$, to prove that we have found minimum:

$$\frac{\partial^2 C}{\partial^2 a} = -\frac{1}{n}\sum(-\frac{y}{a^2} - \frac{1-y}{(1-a)^2})$$

$$-\frac{1}{n}\sum(-\frac{y}{a^2} - \frac{1-y}{(1-a)^2}) > 0$$

substitue $y = a$

$$-\frac{1}{n}\sum(-\frac{y}{y^2} - \frac{1-y}{(1-y)^2}) > 0$$

$$-\frac{1}{n}\sum(-\frac{1}{y} - \frac{1}{1-y}) > 0$$

$$\frac{1}{n}\sum(\frac{1}{y} + \frac{1}{1-y}) > 0$$

Given condition $0 <= y <= 1$, it's prettry obvious this inequality holds. So we have shown that $y = a$ is an extreme value, and that second derivative is larger than zero. This is sufficient to say that $y = a$ is a minimum. Q.E.D.

3 Learning saturation problems

3.1 Saturation for quadratic cost function and sigmoid neurons

Intuition Nothing interesting here, just direct derivation using chain rules

Mathematical notation we want to show that:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n}\sum_x a_k^{L-1}(a_j^L - y_j)\sigma'(z_j^L)$$

when C is quadratic loss function.

$$C = \frac{1}{n}\sum_x\sum_j \frac{(y_j - a_j)^2}{2}$$

Applying chain rule (I'm going to need an abbreviature for that),

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n}\sum_x (y_j - a_j^L)\sigma'(z_j^L)\frac{\partial z}{\partial w_{jk}^L}$$

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n}\sum_x (y_j - a_j^L)\sigma'(z_j^L)a_k^{L-1}$$

Note that we can nullify all other j only if there is no w_{jk}^L in other neurons, which is our case here. That wasn't hard, was it?)

3.2 Show error for cross entropy

Intuition Our only hope (Obi-wan Kenobi), is that ln-s shall produce smth to cancel the activation value denominator

Mathematical notation Here is our cost function:

$$C = -\frac{1}{n}\sum_x\sum_j y_j \ln a_j^L + (1 - y_j)\ln(1 - a_j^L)$$

Let's take our derivative:

$$\frac{\partial C}{\partial z_j} = -\frac{1}{n}\sum_x \frac{y_j}{a_j^L}\frac{\partial a_j^L}{\partial z} - \frac{1 - y_j}{1 - a_j^L}\frac{\partial a_j^L}{\partial z}$$

Now let's remember that $\sigma'(z) = \sigma(z)(1 - \sigma(z))$, and substitue it:

$$\frac{\partial C}{\partial z_j} = -\frac{1}{n}\sum_x \frac{y_j}{a_j^L}a_j^L(1 - a_j^L) - \frac{1 - y_j}{1 - a_j^L}a_j^L(1 - a_j^L)$$

$$\frac{\partial C}{\partial z_j} = -\frac{1}{n}\sum_x y_j(1 - a_j^L) - (1 - y_j)a_j^L$$

$$\frac{\partial C}{\partial z_j} = \frac{1}{n}\sum_x a_j^L - y_j$$

3.3 Show weight derivative for cross entropy

Intuition A usual derivative, just don't forget the $\sigma'(z) = \sigma(z)(1-\sigma(z))$ trick.

Mathematical notation Let's take our previous derivative, but remember that there is now $\frac{\partial z}{\partial w_{jk}}$ new link in a chain:

$$\frac{\partial C}{\partial w_{jk}} = -\frac{1}{n}\sum_x \frac{y_j}{a_j^L}\frac{\partial a_j^L}{\partial z}\frac{\partial z}{\partial w_{jk}} - \frac{1-y_j}{1-a_j^L}\frac{\partial a_j^L}{\partial z}\frac{\partial z}{\partial w_{jk}}$$

Using $\sigma'(z) = \sigma(z)(1-\sigma(z))$ trick we get:

$$\frac{\partial C}{\partial w_{jk}} = -\frac{1}{n}\sum_x y_j(1-a_j^L)\frac{\partial z}{\partial w_{jk}} - (1-y_j)a_j^L\frac{\partial z}{\partial w_{jk}}$$

$$\frac{\partial C}{\partial w_{jk}} = \frac{1}{n}\sum_x (a_j^L - y_j)\frac{\partial z}{\partial w_{jk}}$$

As $z_j = \sum_k w_{jk}a_k^{L-1} + b_j$, $\frac{\partial z}{\partial w_{jk}} = a_k^{L-1}$:

$$\frac{\partial C}{\partial w_{jk}} = \frac{1}{n}\sum_x (a_j^L - y_j)a_k^{L-1}$$

Q.E.D.

A note: for $\frac{\partial C}{\partial b_j}$ the process is the same

3.4 Quadratic cost and linear neurons in output

Now our cost function for one example is $C = \frac{(y-a^L)^2}{2}$

Intuition Nothing interesting here, except a note that we are talking about ouput layer, so learning slowdown may occur for hidden layers.

Mathematical notation Error for quaratic fucntion, given that $a^L = z^L$

$$C = \frac{(y-a^L)^2}{2}$$

$$\frac{\partial C}{\partial z} = \delta_L = y - a^L$$

Derivative for weights:

$$\frac{\partial C}{\partial w_{jk}} = \frac{1}{n}\sum_x \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial w_{jk}}$$

using our previous result for $\frac{\partial C}{\partial z}$, and taking derivative from z_j

$$\frac{\partial C}{\partial w_{jk}} = \frac{1}{n}\sum_x (y_j - a_j^L)a_k^{L-1}$$

Derivative for biases:

$$\frac{\partial C}{\partial b_j} = \frac{1}{n}\sum_x \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial b_j}$$

$$\frac{\partial C}{\partial w_{jk}} = \frac{1}{n}\sum_x (y_j - a_j^L)$$

Q.E.D.

3.5 Discussion on exclusion of x_j (previous neuron value) from parameter derivaties

Intuition The problem of x_j exclusion (or a^{l-1} for layer a^l) is equivalent to the question: Can we train a network when neuron in the next layer either doesn't depend on the previous layer, or depends on it in such a way that $\frac{\partial C}{\partial w_{jk}}$ has no dependecy on the previous layer? The answer is: we can, but there shall be no weigths parameter, only b_j. The explanation is that to have no dependency on x_j, it must enter the activation funtion in a way that $\frac{\partial C}{\partial w_{jk}}$ should give it as a constant in some form. In order for it to be a constant, it has to be separated from the w_{jk} in the activation. Without it, weigths are just an array of constants, so we can group them into one consant, hence b_j. A note: such a network can still be trained! - backprogation equations still make sense here. It's just the parameters for optimization shall include only constants on neurons. It shall still be capable of reacting to different inputs differently, but each neuron shall not have the option of reweighting each previous neuron individually.

No mathematical notation here, maybe later.

4 Softmax

4.1 Example of not 1 sum for usual network

Intuition Proving such statement requires no math whatsoever - there is literally nothing stopping us from assigning weights and biases in any way we like, in any neural network. The better question is this: Can a network to some sensible task generate an output layer, where activation valus shall not sum up to 1? The answer is again yes: let's rememeber binary encoding for numbers classification - each of the correct answers was either one or two neurons activated. For the numbers that were encoded by two activated bits, the trained network would ouput sum of ouput neurons larger than 1.

Again, no need for mathematical notation here.

4.2 Monothonity of softmax

Intuition Increase in numerator is in terms of percents always larger than increase in denominator for each positive δz_j. So first derivatve should be positive. $z_{k,k\neq j}$ exists only in denomiator, so it's increase should lead directly to decrease of a

Mathematical notation

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

First let's work on differenttation by z_j^L. Using differentiation rules for fractions:

$$\frac{\partial a_j^L}{\partial z_j^L} = \frac{e^{z_j^L}\sum_k e^{z_k^L} - e^{z_j^L}e^{z_j^L}}{(\sum_k e^{z_k^L})^2}$$

$$\frac{\partial a_j^L}{\partial z_j^L} = \frac{e^{z_j^L}\sum_k e^{z_k^L} - e^{z_j^L}e^{z_j^L}}{(\sum_k e^{z_k^L})^2}$$

$$\frac{\partial a_j^L}{\partial z_j^L} = \frac{e^{z_j^L}\sum_{k\neq j} e^{z_k^L}}{(\sum_k e^{z_k^L})^2}$$

Exponent is always positive, so $\frac{\partial a_j^L}{\partial z_j^L} > 0$ Now let's work on differenttation by z_k^L:

$$\frac{\partial a_j^L}{\partial z_k^L} = -\frac{e^{z_k^L}e^{z_j^L}}{(\sum_k e^{z_k^L})^2}$$

The denominator is positive, the numerator is positive, so the whole thingy is negative Q.E.D.

4.3 Inverting the softmax layer

Intuition Nothing interesting here.

Mathematical notaion

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

$$ln(a_j^L) = ln(e^{z_j^L}) - ln(\sum_k e^{z_k^L})$$

$$ln(a_j^L) = z_j^L - ln(\sum_k e^{z_k^L})$$

As we can see, $ln(\sum_k e^{z_k^L})$ is the same for all neurons, so we can replace it with C.

$$z_j^L = ln(a_j^L) + C$$

Q.E.D.

4.4 Differentiation of parameters for Loglikelihood

Intuition Again, nothing particulary interesting, except a way to include $y_j = 1$, but that works out rather naturally.

Mathematical notation

$$C \equiv -ln(a_y^L)$$

Let's strart with weight w_{jk}

$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{w_{jk}}$$

From the previous exercise we know $\frac{\partial a_j^L}{\partial z_j^L}$, so let's substitute it (along with other stuff):

$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{1}{a_j^L}\frac{e^{z_j^L}\sum_{k\neq j} e^{z_k^L}}{(\sum_k e^{z_k^L})^2}a_k^{L-1}$$

$$\frac{\partial C}{\partial w_{jk}^L} = -1\frac{\sum_{k\neq j} e^{z_k^L} + e^{z_j^L} - e^{z_j^L}}{(\sum_k e^{z_k^L})}a_k^{L-1}$$

$$\frac{\partial C}{\partial w_{jk}^L} = -1(1 - \frac{e^{z_j^L}}{\sum_k e^{z_k^L}})a_k^{L-1}$$

That fraction is acutally activation for j

$$\frac{\partial C}{\partial w_{jk}^L} = (a_j^L - 1)a_k^{L-1}$$

. (Strange stuff here, recheck:) for j, $y_j = 1$, so we can substitue that to get:

$$\frac{\partial C}{\partial w_{jk}^L} = (a_j^L - y_j)a_k^{L-1}$$

.

For b_j:

$$\frac{\partial C}{\partial b_j^L} = -\frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{b_j}$$

The solution is the same as for w_{jk}, except that $\frac{\partial z_j^L}{\partial b_j^L} = 1$, so there is no a_k^{L-1} part.

$$\frac{\partial C}{\partial b_j^L} = a_j^L - y_j$$

.

Q.E.D.

4.5 Softmax origination and naming reasons

Intuition About probability distribution: we have to show that for every event probability/density is non-negative, and that probabilities of all possible events sum up to 1.

About limiting value of activations: as we have the same value in the numerator and as part of the denomiator, it's pretty easy to see that this fraction is strictly non-negative, and can never be larger than 1.

The behavior of function for differnt c is not obivous, so we will have to check differentiantions.

Derivation of $\frac{\partial C}{\partial z_j}$ is straighforward, we kinda did before actually, for b_j and w_j.

Mathematical notation Let's show that all activations sum up to 1. We can take equation (79) for the texbook itself, and add c to it:

$$\sum_j a_j^L = \sum_j \frac{e^{Cz_j^L}}{\sum_k e^{Cz_k^L}} = 1$$

Non negativity follows directly from usage of e, as it is alwasys non-negative. These two conditions are sufficient (not very strict sense though) to use this ́flexibléśoftmax as a distribution function.

Let's differentiate softmax by c:

$$a_j^L = \frac{e^{cz_j^L}}{\sum_k e^{cz_k^L}}$$

$$\frac{\partial a_j^L}{\partial c} = \frac{z_j^L e^{cz_j^L} \sum_k e^{cz_k^L} - e^{cz_j^L} \sum_k z_k^L e^{cz_k^L}}{(\sum_k e^{cz_k^L})^2}$$

$$\frac{\partial a_j^L}{\partial c} = e^{cz_j^L} \frac{z_j^L \sum_k e^{cz_k^L} - \sum_k z_k^L e^{cz_k^L}}{(\sum_k e^{cz_k^L})^2}$$

$$\frac{\partial a_j^L}{\partial c} = e^{cz_j^L} \frac{\sum_k (z_j^L - z_k^L) e^{cz_k^L}}{(\sum_k e^{cz_k^L})^2}$$

Two facts are very easy to see now:

$$\forall_k z_j^L > z_k^L : \frac{\partial a_j^L}{\partial c} > 0$$

$$\forall_k z_j^L < z_k^L : \frac{\partial a_j^L}{\partial c} < 0$$

So, if our neuron is stronger than all others, c makes it (relatively) more powerful, and if it is weaker, c makes it even weaker. In terms of function form, this means that the larger c the steeper is shift from 0 to 1 as z_j increases.

And finally, error derivation:

$$\frac{\partial C}{\partial z_j^L} = -\frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}$$

Using approach from previous exercises:

$$\frac{\partial C}{\partial z_j^L} = -\frac{1}{a_j^L}\frac{e^{z_j^L}\sum_{k\neq j} e^{z_k^L}}{(\sum_k e^{z_k^L})^2}$$

$$\frac{\partial C}{\partial z_j^L} = -1\frac{\sum_{k\neq j} e^{z_k^L} + e^{z_j^L} - e^{z_j^L}}{(\sum_k e^{z_k^L})}$$

$$\frac{\partial C}{\partial z_j^L} = -1(1 - \frac{e^{z_j^L}}{\sum_k e^{z_k^L}})$$

$$\frac{\partial C}{\partial z_j^L} = (a_j^L - 1)$$

. (Strange stuff here, recheck:) for j, $y_j = 1$, so we can substitue that to get:

$$\frac{\partial C}{\partial z_j^L} = a_j^L - y_j$$

.

Q.E.D.

5 Artificial dataset expansion

The problem with arbitrarily large rotations is actually two-fold: 1) We make training examples look like óthertŕaining examples. 6 and 9 for example shall generate a lot of fun. To make that work, we shall have to use even larger set, which kinda defeats the point. 2) Overall teaching the model extremly-rare real-life cases seems like a good way to waste a lot of optimization time on very small part of real world problem. It's much easier to first classify whether the dataset has been reversed, and than classify digits.

6 Asymptotic efficiency of algorithms

This is a large topic/question, we'll return here later. Some sources:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3307431/

7 Initialization tricks

7.1 Standart deviation estimation

Intuition We are dealing with independent random variables, so this is pretty easy.

Mathematical notation

$$Var(z) = \sum_j Var(w_j)x_j^2 + Var(b)$$

$$Var(z) = \sum_j \frac{1}{n_{in}} x_j^2 + 1$$

Now $n_i n = 1000$, and 500 of $1000 x_j$-s is 1, so:

$$Var(z) = \frac{1}{1000} 500 + 1$$

$$Var(z) = \frac{3}{2}$$

$$\sigma(z) = \sqrt{\frac{3}{2}}$$

Tada.

8 Regularization heuristscs

8.1 First epochs dominated by weight decay

A note: *epochs*, as in $\ddot{\text{do}}$ this update n/m times: Had been stuck here, for some time, because thought it's about one update. Smooth, I know.

Intuition The main problem is that while logloss fixes neuron saturation in the output layer, all other layers still have $\sigma'(z)$ part in the $\frac{\partial C}{\partial z}$ equation. As hidden neurons are almost guaranteed to be saturated, $\sigma'(z)$ goes to zero.

Mathematical notation A note: I'm ignoring the behavior of the output layer, as it's behavior differs considerably from other layers. Probabily will fill it up later.

Let's remember update rule for hidden layers (for regularized case):

$$\delta^l = ((W^{l+1})^T \delta^{l+1})\sigma'(z)$$
$$\frac{\partial C}{\partial W^l} = a^{l-1}(\delta^l)^T + \frac{\eta\lambda}{n} W^l$$

As neurons are saturated, $(\delta^l)^T \to 0$, and update rule can be simplified to:

$$W^l \to (1 - \frac{\eta\lambda}{n})W^l$$

which means than as long as $\sigma'(z) \to 0$, only weight decay is active.

8.2 Weight decay speed

During the first epoch, we can assume n/m updates shall happen. Than, if $\sigma'(z) \to 0$, only weight decay is active. So we can write down result of weight decay at the end of the epoch as

$$W^l \to (1 - \frac{\eta\lambda}{n})^{n/m} W^l$$

Now if we interpret $\eta\lambda << n$ as $n \to \infty$, than

$$lim_{n\to\infty}(1 - \frac{\eta\lambda}{n})^{n/m} W^l = W^l exp(-\eta\lambda/m)$$

(By classic limit rule for $(1 - 1/x)^x$). This is the expected update ratio.

8.3 Weight decay tailoff

Weight decay will tail off (for L2) once the regularization component is close to constant (small one). Regularization component for mini-batch update is:

$$\frac{\eta\lambda}{2m}\sum_w w^2$$

It's easy to see that if we substitute $w = n^{-1/2}$, the value of L2 component shall be

$$\frac{\eta\lambda}{2m}1$$

The intuition is quite simple - for quadratic function derivative is linear, so close to zero the derivative starts gettting closer to zero. The concept of tailing-off is rather relative, but this particular value $n^{-1/2}$ is where L2 regularization for any network becomes independent of the number of weights.

8.4 Relation between tahn and sigmoid neurons

$$\frac{1}{2}(1 + \frac{e^{-z/2} + e^{z/2}}{e^{z/2} + e^{-z/2}}) == \frac{1}{1 + e^{-z}}$$

$$\frac{1}{2}(\frac{e^{-z/2} + e^{-z/2} + e^{-z/2} - e^{z/2}}{e^{z/2} + e^{-z/2}}) == \frac{1}{1 + e^{-z}}$$

$$\frac{e^{-z/2}}{e^{z/2} + e^{-z/2}} == \frac{1}{1 + e^{-z}}$$

$$\frac{1}{1 + e^{-z}} == \frac{1}{1 + e^{-z}}$$

Q.E.D.