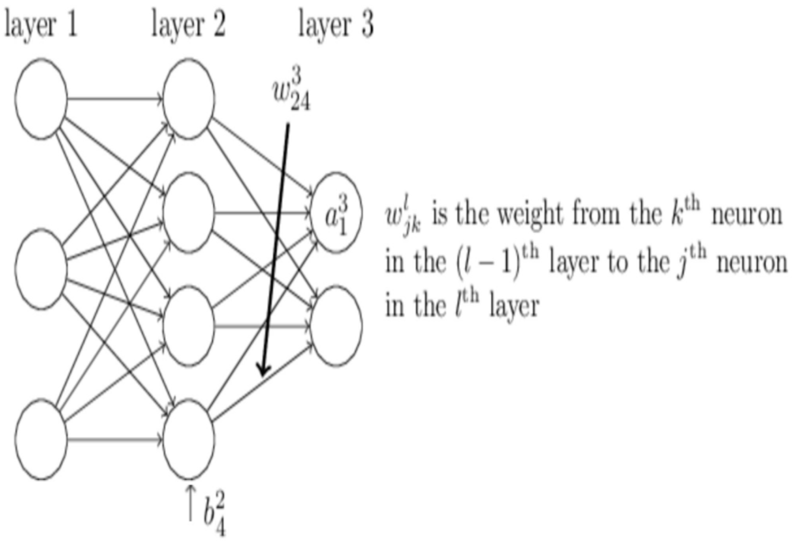
3. Backpropagation

One of the key steps for training multi-layer neural networks is stochastic gradient descent. We will use the back-propagation algorithm to compute the gradient of the loss function with respect to the model parameters.

Consider the L-layer neural network below:



In the following problems, we will the following notation: b^l_j is the bias of the j^{th} neuron in the l^{th} layer, a^l_j is the activation of j^{th} neuron in the l^{th} layer, and w^l_{jk} is the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.

If the activation function is f and the loss function we are minimizing is C, then the equations describing the network are:

$$\begin{aligned} a_j^l &= f\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \\ \text{Loss} &= C\left(a^L\right) \end{aligned}$$

Note that notations without subscript denote the corresponding vector or matrix, so that a^l is activation vector of the l^{th} layer, and w^l is the weights matrix in l^{th} layer.

For $l = 1, \ldots, L$.

Computing the Error

2/2 points (graded)

Let the weighted inputs to the d neurons in layer l be defined as $z^l \equiv w^l a^{l-1} + b^l$, where $z^l \in \mathbb{R}^d$. As a result, we can also write the activation of layer l as $a^l \equiv f\left(z^l\right)$, and the “error" of neuron j in layer l as $\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$. Let $\delta^l \in \mathbb{R}^d$ denote the full vector of errors associated with layer l.

Back-propagation will give us a way of computing δ^l for every layer.

Assume there are d outputs from the last layer (i.e. $a^L \in \mathbb{R}^d$). What is δ_j^L for the last layer?

● $\frac{\partial C}{\partial a_j^L} f'\left(z_j^L\right)$

○ $\sum_{k=1}^{d} \frac{\partial C}{\partial a_k^L} f'\left(z_j^L\right)$

○ $\frac{\partial C}{\partial a_j^L}$

○ $f'\left(z_j^L\right)$

✓

What is δ_j^l for all $l \neq L$?

● $\sum_k w_{kj}^{l+1} \delta_k^{l+1} f'\left(z_j^l\right)$

○ $\delta_k^{l+1} f'\left(z_j^l\right)$

○ $\sum_k w_{jk}^{l-1} \delta_j^{l-1} f'\left(z_j^l\right)$

○ $\sum_k w_{kj}^{l+1} \delta_k^{l+1} f\left(z_j^l\right)$

✓

Solution:

We make use of the chain rule.

1. By definition, $\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} f'\left(z_j^L\right)$.

2. We have:

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l}\\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}\\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}\end{aligned}$$

Then we have $z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} f\left(z_j^l\right) + b_k^{l+1}$. Taking the derivative of this with respect to z_j^l gives $w_{kj}^{l+1} f'\left(z_j^l\right)$.

Combining the two gives the final answer: $\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} f'\left(z_j^l\right)$.

Submit

You have used 2 of 2 attempts

Answers are displayed within the problem

Parameter Derivatives

2/2 points (graded)

During SGD we are interested in relating the errors computed by back-propagation to the quantities of real interest: the partial derivatives of the loss with respect to our parameters. Here that is $\frac{\partial C}{\partial w_{jk}^l}$ and $\frac{\partial C}{\partial b_j^l}$.

What is $\frac{\partial C}{\partial w_{jk}^l}$? Write in terms of the variables a_k^{l-1}, w_j^l, b_j^l, and δ_j^l if necessary.

Example of writing superscripts and subscripts:

$delta\_j\backslash\hat{}l$ for δ_j^l

$w\_\left(jk\right)\backslash\hat{}l$ for w_{jk}^l

$\frac{\partial C}{\partial w^l_{jk}} =$ a_k^(l-1)*delta_j^l ✓ **Answer:** a_k^(l-1)*delta_j^l

$a^{l-1}_k \cdot \delta^l_j$

What is $\frac{\partial C}{\partial b^l_j}$? Write in terms of the variables a^{l-1}_k, w^l_j, b^l_j, and δ^l_j if necessary.

$\frac{\partial C}{\partial b^l_j} =$ delta_j^l ✓ **Answer:** delta_j^l

δ^l_j

STANDARD NOTATION

Solution:

1. $\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$

2. $\frac{\partial C}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial b^l_j} = 1 * \delta^l_j$

Submit You have used 1 of 5 attempts

Answers are displayed within the problem

Activation Functions: Sigmoid

4/4 points (graded)

Recall that there are several different possible choices of activation functions f. Let's get more familiar with them and their gradients.

What is the derivative of the sigmoid function, $\sigma(z) = \frac{1}{1+e^{-z}}$? Please write your answer in terms of e and z:

e^(-z)/(1+e^(-z))^2

✓ **Answer:** e^(-z) / (1 + e^(-z))^2

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

Which of the following is true of $\sigma'(z)$ as $||z||$ gets large?

○ Its magnitude becomes large.

● Its magnitude becomes small.

○ It suffers from high variance.

✓

What is the derivative of the ReLU function, $\mathrm{ReLU}(z) = \max(0, z)$ for $z > 0$?

1

✓ **Answer:** 1

1

For $z < 0$?

0

✓ **Answer:** 0

0

STANDARD NOTATION

Solution:

$\sigma'(z) = \sigma(z)(1 - \sigma(z))$. As z gets large in magnitude, the sigmoid function saturates, and the gradient approaches zero.

ReLU is a simple activation function. Above zero, it has a constant gradient of 1.

Below zero, it is always zero.

Submit

You have used 1 of 5 attempts

Answers are displayed within the problem

Simple Network

4/4 points (graded)

Consider a simple 2-layer neural network with a single neuron in each layer. The loss function is the quadratic loss: $C = \frac{1}{2}(y-t)^2$, where y is the prediction and t is the target.

Starting with input x we have:

- $z_1 = w_1 x$
- $a_1 = \mathrm{ReLU}\left(z_1\right)$
- $z_2 = w_2 a_1 + b$
- $y = \sigma\left(z_2\right)$
- $C = \frac{1}{2}(y-t)^2$

Consider a target value $t = 1$ and input value $x = 3$. The weights and bias are $w_1 = 0.01$, $w_2 = -5$, and $b = -1$.

Please provide numerical answers accurate to at least three decimal places.

What is the loss?

0.2884

✓ **Answer:** 0.28842841648243966

What are the derivatives with respect to the parameters?

$\frac{\partial C}{\partial w_1} =$ 2.0809 ✓ **Answer:** 2.0809165621704553

$\frac{\partial C}{\partial w_2} =$ -0.00416 ✓ **Answer:** -0.00416183312434091

$\frac{\partial C}{\partial b} =$ -0.1387 ✓ **Answer:** -0.13872777081136367

STANDARD NOTATION

Solution:

Using the chain rule, we have:

- $\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} = (y-t)\,y\,(1-y)\,w_2\mathbf{1}\{z_1>0\}x$
- $\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial w_2} = (y-t)\,y\,(1-y)\,a_1$
- $\frac{\partial C}{\partial b} = (y-t)\,y\,(1-y)$

Submit You have used 2 of 5 attempts

ℹ Answers are displayed within the problem

SGD

1/1 point (graded)

Referring to the previous problem, what is the update rule for w_1 in the SGD algorithm with step size η? Write in terms of w_1, η, and $\frac{\partial C}{\partial w_1}$; enter the latter as `(partialC)/(partialw_1)`, noting the lack of space in the variable names:

Next $w_1 =$ w_1 - eta * (partialC)/(partia ✓

Answer: w_1 - eta * (partialC)/(partialw_1)

STANDARD NOTATION

Solution:

The definition of the simple SGD update rule is new_parameter = old_parameter - learning_rate * derivative of loss w.r.t old parameter.

Submit You have used 1 of 5 attempts

Answers are displayed within the problem

Discussion

Hide Discussion

Topic: Unit 3 Neural networks (2.5 weeks):Homework 4 / 3. Backpropagation

Add a Post

Show all posts by recent activity

Matrix calculus
If someone is interested in learning a little more about matrix calculus, the below link is good...
3

? [Staff] Parameter Derivatives Work Around No Longer works
Hi, I saw in the forum that folks were using alpha_k^(l-1) as a work around because the listed ...
4

Multivariable chain rule, simple version
I believe this can be usable at some point on this page: https://www.khanacademy.org/math/...
4

Numerical/computational solution to the "Simple network" problem
Community TA
1

[STAFF] SGD Question
I used this expression "w_1 - eta * (partialC)/(partialw_1)" but the system didn't accept it.
2

[Staff] Question: Simple Network - issue on the gradient of C with respect to weights
Hi all, Sorry for asking that but I already wasted 3 tries in the question and don want to try an...
14

? [STAFF] Could you check the SDG answer for me.
@Staff, I'm pretty sure I'm using the valid notation but always getting parsing error. Could yo... 6

? [STAFF] Rounding for simple network
Apologies if this has already been addressed but I'm wondering how to interpret the instructi... 3

☑ How does software handles differentiation of non-differentiation functions?
In the exercise "Activation Functions: Sigmoid" we differentiated non-differentiable functions,... 2

💬 Comparing partial derivatives of loss w.r.t weights
Trying to build the intuition for partial derivatives. In this toy example with simple NN, If we r... 2

☑ Simple network - Meaning of sigma
What's the meaning of sigma in the expression y = sigma(z2)? Thanks in advance! 4

? Any hints regarding Parameter Derivatives?
As the title said, I am quite lost in all those notations and do not have a clear understanding o... 6