Homework 7

The questions below are due on Sunday October 29, 2017; 11:00:00 PM.

You are not logged in.

If you are a current student, please Log In (https://introml.mit.edu/fall17/homework/hw07?loginaction=login) for full access to this page.

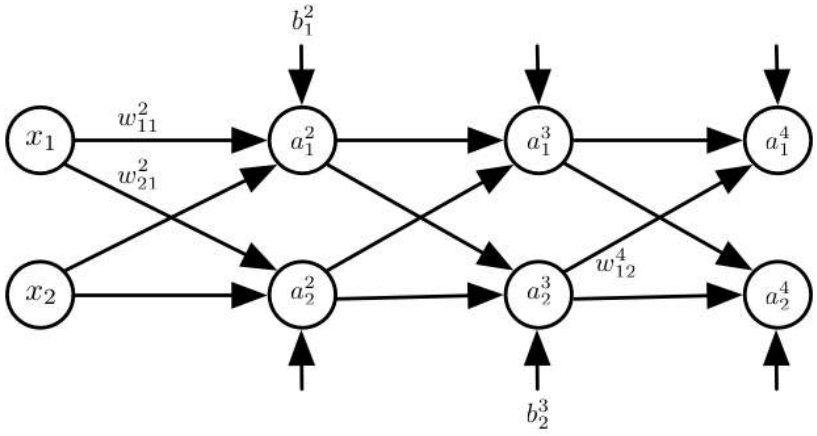
The notation for neural nets in the notes is for networks with at most one hidden layer; we'll need a notation for multilayer networks. We are going to use a more general neural network notation in some of the problems in this homework, even when it seems like overkill, to get us used to it. So:

- Input units are $x_1, \ldots, x_d$. They are also known as the activations in the first layer, so $a_1^1, \ldots, a_d^1$
- Number of layers is L
- There are m^l units in layer l
- Output units are $a_1^L \ldots a_{m^L}^L$
- Weight from unit k in layer $l-1$ to unit j in layer l is w_{jk}^l
- Constant offset for unit j in layer l is b_j^l
- Activation function at layer l is f^l; most of the time, we use one activation function in the output layer (f^L), and then the same activation function in the rest of the layers
- Activation of unit j at layer l is computed as:

$$a_j^l = f^l\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

- Cost function $C(a, y)$ measures the loss of output a when the target is y

Here is an illustrative picture:



$$a_j^1 = x_j \qquad a_j^l = f(z_j^l) = f\left(\sum_i w_{ji} a_i^{l-1} + b_j^l\right)$$

1) LOSS FUNCTIONS AND OUTPUT ACTIVATIONS: CLASSIFICATION

When doing classification, it's natural to think of the output values as being discrete: +1 and -1. But it is generally difficult to use optimization-based methods without somehow thinking of the outputs as being continuous (even though you will have to discretize when it's time to make a prediction).

1.1) Hinge loss, linear activation

When we looked at the SVM objective for classification, we did this:

- defined the output space to be $\mathbb{R}$
- developed a loss function

$$C(a, y) = L_h(ya) = \begin{cases} 0 & \text{if } ya > 1 \\ 1 - ya & \text{otherwise} \end{cases}$$

  where a is the continuous output (we're using a here to be consistent with the neural network terminology of *activation*) and y is the desired output
- tried to find parameters θ of our model to minimize loss summed over the training data

Consider a single "neuron" with a linear activation function; that is, where $a_1^L = \sum_k w_{1k}^L x_k + b_1^L$. In this case $L = 2$.

Write a short program to compute the gradient of the NLL with respect to the weight vector: $\nabla_{w^L} C(a_1^L, y)$ when $C(a, y) = L_h(ya)$.

- `x` is a column vector
- `y` is a number, a label
- `a` is a number, an activation

It should return a column vector.

```
1 def linear_hinge_grad(x, y, a):
2   pass
3
```

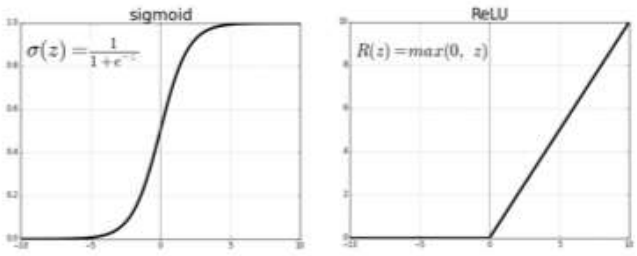
Ask for Help

1.2) Log loss, sigmoidal activation

Another way to make the output for a classifier continuous is to make it be in the range $[0, 1]$, which admits the interpretation of being a probability that the example is positive. A convenient way to make the activation of a unit be in the range $[0, 1]$ is to use a *sigmoid* function:

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

The figure below shows a sigmoid activation function on the left, with the rectified linear (ReLU) activation function used in the class notes for comparison.



1) What is an expression for the derivative of the sigmoid wrt to z, expressed as a function of z, its input:

Enter a Python expression (use ** for exponentiation) involving `e` and `z`:

```
e**-z/(1+e**-z)**2
```

Ask for Help

2) What is an expression for the derivative of the sigmoid wrt to z but expressed as a function of $o = \sigma(z)$, its output?

Hint: Think about the expression $1 - \frac{1}{1+e^{-z}}$.

Enter a Python expression (use ** for exponentiation) involving `o`:

```
o*(1-o)
```

Ask for Help

In this model, we will consider positive points to have label +1, and negative points to have label 0.

We need a loss function that works well when we are predicting probabilities. A good choice is to ask what probability is assigned to the correct label.

We will interpret the output value a as the probability that the example is positive.

So, if the output value is a and the true value is $+1$, then the probability assigned to the true value is a; on the other hand, if the true value is 0, then the probability assigned to the true value is $1 - a$. Because we actually will be interested in the probability of the predictions on the whole data set, we'd want to choose weights to maximize

$$\prod_t P(a^{(t)}, y^{(t)})$$

Using a notational trick (which turns an *if* expression into a product) that might seem unmotivated now, but will be useful later, we can write $P(a, y)$ as

$$a^y (1 - a)^{(1-y)}$$

.

What is the value of the expression above?

3) When $y = 0$?

Enter an expression for the expression above when $y = 0$ in terms of `a`:

1-a

Ask for Help

4) When $y = 1$?

Enter an expression for the expression above when $y = 1$ in terms of `a`:

a

Ask for Help

5) What is the value of $\log P(a, y)$?

Enter an expression in terms of `y` and `a`, you can use `log(.)`:

```
y*log(a) + (1-y)*log(1-a)
```

Ask for Help

In fact, because log is a monotonic function, the same weights that maximize the product of the probabilities will minimize the *negative log likelihood* ("likelihood" is the same as probability; we just use that name here because the phrase is an idiom in machine learning, abbreviated NLL):

$$\sum NLL(a^{(t)}, y^{(t)}) = -\sum_{t=1}^{n} y^{(t)} \log a^{(t)} + (1 - y^{(t)}) \log(1 - a^{(t)})$$

and that will be our objective function.

Now, we can think about a single unit with a sigmoidal activation function, trained to minimize NLL. So, $a_1^L = \sigma(\sum_k w_{1k}^L x_k + b_1^L)$. In this case $L = 2$.

6) Write a formula for the derivative of the NLL with respect to the first weight: $\frac{\partial}{\partial w_{11}^L} NLL(a_1^L, y)$ for a single training example.

Write an expression in terms of `x_1`, `a_1`, and `y`:

```
?
```

Ask for Help

7) Write a formula for the gradient of the cost with respect to the weight vector: $\nabla_{w^L} NLL(a_1^L, y)$.

Enter an expression in terms of `x`, `a_1`, and `y`:

```
?
```

Ask for Help

2) MULTICLASS CLASSIFICATION

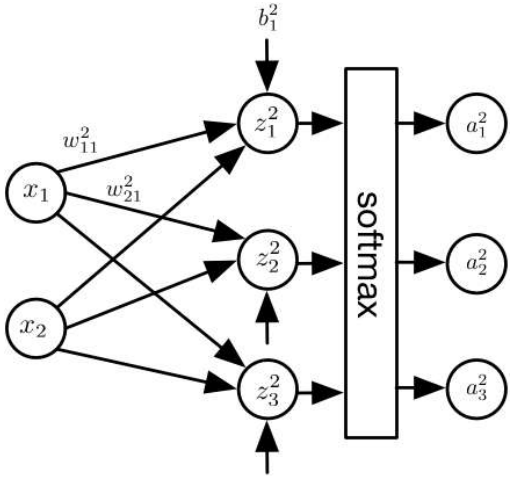
What if we needed to classify homework problems into three categories: enlightening, boring, impossible? We can do this by using a "one-hot" encoding on the output, and using three output units with what is called a "softmax" (SM) activation function. It's not a typical activation function, really, it's more of a layer on its own. It takes in m_L input values z_j^L in $\mathbb{R}$ and returns m_L output values $a_j^L \in [0, 1]$ such that $\sum_j a_j^L = 1$. This can be interpreted as representing a probability distribution over the possible categories.

The individual entries are computed as

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{m^L} e^{z_k}}$$

We'll describe the relationship of the vector a on the vector z as

$$a = \mathrm{SM}(z)$$



1) What probability distribution over the categories is represented by $z^L = [-1, 0, 1]$?

Enter a distribution, a list of numbers adding up to 1, for the three categories.

```
[0.09003057317038046, 0.24472847105479767, 0.66524
```

Ask for Help

Now we need a loss function $C(a, y)$ where now a is a discrete probability distribution and y is a one-hot vector encoding of a single output value. It makes sense to use negative log likelihood as a cost function for the same reasons as above. We'll extend the definition

$$NLL(a, y) = -\sum_{j=1}^{m^L} y_j \log a_j^L$$

.

2) If $a = [.3, .5, .2]$ and $y = [0, 0, 1]$, what is $NLL(a, y)$?

Enter an expression involving `log(.)` and constants. `-log(0.2)`

Ask for Help

Now, we can think about a single layer with a softmax activation function, trained to minimize NLL. The "pre-softmax" values are:

$$z_j^L = \sum_k w_{jk}^L x_k + b_j^L$$

and $a^L = SM(z^L)$.

To do gradient descent, we need to know $\frac{\partial}{\partial w_{jk}^L} NLL(a^L, y)$. We'll reveal the secret (that you might guess from Problem 1) that it has an awesome form! (Please consider deriving this, for fun and satisfaction!)

$$\frac{\partial}{\partial w_{jk}^L} NLL(a^L, y) = x_k(a_j^L - y_j)$$

And of course, it's easy to compute the whole matrix of these derivatives, $\nabla_{w^L} NLL(a^L, y)$ in one quick matrix computation.

3) If we have two input units and three possible output values,

$$w_L = \begin{bmatrix} 1 & -1 & -2 \\ -1 & 2 & 1 \end{bmatrix}$$

Assume the biases are zero, the input $x = (1, 1)^T$ and the target output $y = (0, 1, 0)^T$, what is the whole matrix $\nabla_{w^L} NLL(a^L, y)$?

Enter the matrix as a list of lists, one list for each row of the matrix.

`[[0.24472847, -0.33475904, 0.09003057],[ 0.2447284`

Ask for Help

4) What is the predicted probability that x is in class 1? (assume we have classes 0, 1, and 2)

Enter a number: `0.66524096`

Ask for Help

5) Using learning rate 0.5, what is w^L after one gradient update step?

Enter a list of lists corresponding to the rows of the matrix (accurate to two decimal places).

```
[[0.88, -0.83, -2.05],[-1.12, 2.17, 0.95]]
```

Ask for Help

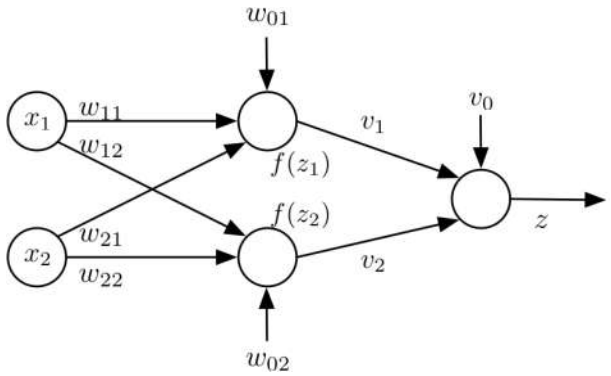
6) What is the predicted probability that x is in class 1, given the new weight matrix?

Enter a number: `0.77245284`

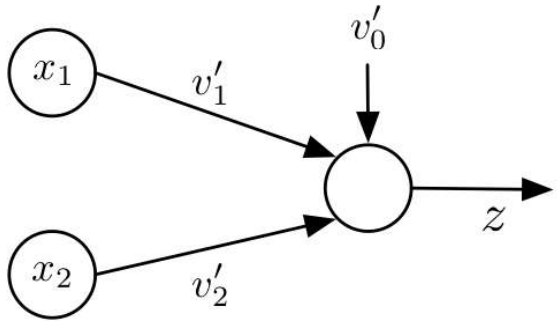
Ask for Help

3) LINEAR + LINEAR = LINEAR

Nur Alnet is trying to train the following network



where the activation function in all the units is the identity: $f(z) = z$. Lenny Arlayer comes by and says that it would be just as effective to use this network instead



Lenny is right. Supply expressions for v'_0, v'_1 and v'_2 in Lenny's network, in terms of weights w_{01}, w_{11}, w_{12}, w_{02}, w_{21}, w_{22}, v_0, v_1, and v_2 that will make Lenny's network equivalent to Nur's.

Enter expressions in terms of `w_01`, `w_11`, `w_12`, `w_02`, `w_21`, `w_22`, `v_0`, `v_1`, and `v_2`.

1)

$v_0' =$

```
w_01*v_1 + w_02*v_2 + v_0
```

Ask for Help

2)

$v_1' =$

```
w_11*v_1 + w_12*v_2
```

Ask for Help

3)

$v_2' =$

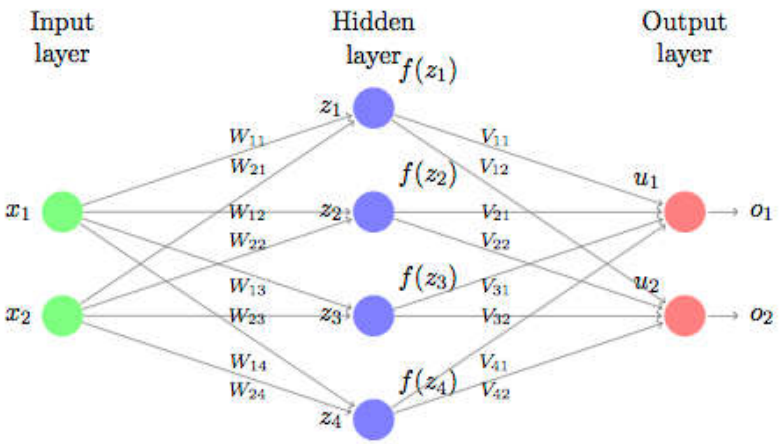
```
v_1*w_21 + v_2*w_22
```

Ask for Help

4) NEURAL NETWORKS

In this problem we will analyze a simple neural network to understand its classification properties. Consider the neural network given in the figure below, with ReLU activation functions (denoted by f) on all hidden neurons, and a softmax activation function in the output layer:

We're sorry about notational inconsistency in this problem; please look at the figure for the notation.



Given an input $x = [x_1, x_2]^T$, the hidden units in the network are activated in stages as described by the

following equations:

$$\begin{aligned} z_1 &= x_1 W_{11} + x_2 W_{21} + W_{01} & f(z_1) &= \max\{z_1, 0\} \\ z_2 &= x_1 W_{12} + x_2 W_{22} + W_{02} & f(z_2) &= \max\{z_2, 0\} \\ z_3 &= x_1 W_{13} + x_2 W_{23} + W_{03} & f(z_3) &= \max\{z_3, 0\} \\ z_4 &= x_1 W_{14} + x_2 W_{24} + W_{04} & f(z_4) &= \max\{z_4, 0\} \end{aligned}$$

$$\begin{aligned} u_1 &= f(z_1)V_{11} + f(z_2)V_{21} + f(z_3)V_{31} + f(z_4)V_{41} + V_{01} & f(u_1) &= \max\{u_1, 0\} \\ u_2 &= f(z_1)V_{12} + f(z_2)V_{22} + f(z_3)V_{32} + f(z_4)V_{42} + V_{02} & f(u_2) &= \max\{u_2, 0\}. \end{aligned}$$

The final output of the network is obtained by applying the *softmax* function to the last hidden layer,

$$\begin{aligned} o_1 &= \frac{e^{f(u_1)}}{e^{f(u_1)} + e^{f(u_2)}} \\ o_2 &= \frac{e^{f(u_2)}}{e^{f(u_1)} + e^{f(u_2)}}. \end{aligned}$$

In this problem, we will consider the following setting of parameters:

$$\begin{bmatrix} W_{11} & W_{21} & W_{01} \\ W_{12} & W_{22} & W_{02} \\ W_{13} & W_{23} & W_{03} \\ W_{14} & W_{24} & W_{04} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix},$$

$$\begin{bmatrix} V_{11} & V_{21} & V_{31} & V_{41} & V_{01} \\ V_{12} & V_{22} & V_{32} & V_{42} & V_{02} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ -1 & -1 & -1 & -1 & 2 \end{bmatrix}.$$

4.1) Output

Consider the input $x_1 = 3, x_2 = 14$.

1) What are the outputs of the hidden units, $(f(z_1), f(z_2), f(z_3), f(z_4))$.

Enter a Python list of 4 numbers.

```
[ 2, 13, 0, 0]
```

Ask for Help

2) What is the final output (o_1, o_2) of the network?

Enter a Python list of 2 numbers.

```
[9.99999694e-01, 3.05902227e-07]
```

Ask for Help

4.2) Unit decision boundaries

Let's characterize the *decision boundaries* in x-space, corresponding to the four hidden units. These are the regions where the input to the units z_1, z_2, z_3, z_4 are exactly zero. Hint: You should draw a diagram of the decision boundaries for each unit in the x-space and label the sides of the boundaries with 0 and $+$ to indicate whether the unit's output would be exactly 0 or positive, respectively.

1) What is the shape of the decision boundary for a single unit?

Choose one: Line

Ask for Help

2) Enter a 2 x 4 matrix where each column represents a (different) input vector $[x_1, x_2]^T$ each of which is on the decision boundary for the first unit, that is, for which $z_1 = 0$.

Enter a Python list of lists where each list is a row of the matrix.

```
[[1,1,1,1],[1,2,3,4]]
```

Ask for Help

3) Consider the following input vectors: $x^{(1)} = [0.5, 0.5]^T, x^{(2)} = [0, 2]^T, x^{(3)} = [-3, 0.5]^T$. Enter a matrix where each column represents the outputs of the hidden units $(f(z_1), \ldots, f(z_4))$ for each of the input vectors.

Enter a Python list of lists where each list is a row of the matrix.

```
[[0, 0, 0, 0], [0, 1, 0, 0], [0, 0, 2, 0]]
```

Ask for Help

4.3) Network outputs

A network with two softmax units is used to classify the input vector into one of two classes. For class 1, the first unit's output should be larger than the other's and for class 2, the second unit's output should be larger. This generalizes nicely to k classes by using k output units.

Let's characterize the region in x-space where this network's output indicates the first class (that is, o_1 is larger) or indicates the second class (that is, o_2 is larger). Your diagram from the previous part will be useful here.

What is the output value of the neural network in each of the following cases? Write your answer for o_i as expressions, you can use powers of e, for example, `e**2 + 1`; the exponents can be negative, `e**(-2) + 1`.

Case 1) For $f(z_1) + f(z_2) + f(z_3) + f(z_4) = 0$

a)

$o_1 =$ 0.11920292

Ask for Help

b)

$o_2 =$ 0.88079708

Ask for Help

c)

Which class is predicted? Class 2

Ask for Help

Case 2) For $f(z_1) + f(z_2) + f(z_3) + f(z_4) = 1$

a)

$o_1 =$ 0.5

Ask for Help

b)

$o_2 =$ 0.5

Ask for Help

c)

Which class is predicted? Boundary

Ask for Help

Case 3) For $f(z_1) + f(z_2) + f(z_3) + f(z_4) = 3$

a)

$o_1 =$ 0.98201379

Ask for Help

b)

$o_2 =$ 0.01798621]

Ask for Help

c)

Which class is predicted? Class 1

Ask for Help

4.4) Network boundaries

Case 1) For $f(z_1) + f(z_2) + f(z_3) + f(z_4) = 0$

a)

Does this case hold over a region or a (piecewise linear) curve of the input x-space? --

Ask for Help

b)

Enter a list of vertices (corners), each written as an $[x_1, x_2]$ list, that define the boundary of the region or define the (piecewise linear) curve.

Ask for Help

c)

What class is predicted for points just on the inside of this boundary (closer to the origin)?

--

Ask for Help

d)

What class is predicted for points just on the outside of this boundary (farther from the origin)? --

Ask for Help

Case 2) For $f(z_1) + f(z_2) + f(z_3) + f(z_4) = 1$

a)

Does this case hold over a region or a (piecewise linear) curve of the input x-space? Curve

Ask for Help

b)

Enter a list of vertices (corners), each written as an $[x_1, x_2]$ list, that define the boundary of the region or define the (piecewise linear) curve.

Ask for Help

c)

What class is predicted for points just on the inside of this boundary (closer to the origin)?

--

Ask for Help

d)

What class is predicted for points just on the outside of this boundary (farther from the origin)? --

Ask for Help

Case 3) For $f(z_1) + f(z_2) + f(z_3) + f(z_4) = 3$

a)

Does this case hold over a region or a (piecewise linear) curve of the input x-space? --

Ask for Help

b)

Enter a list of vertices (corners), each written as an $[x_1, x_2]$ list, that define the boundary of the region or define the (piecewise linear) curve.

Ask for Help

c)

What class is predicted for points just on the inside of this boundary (closer to the origin)?

--

Ask for Help

d)

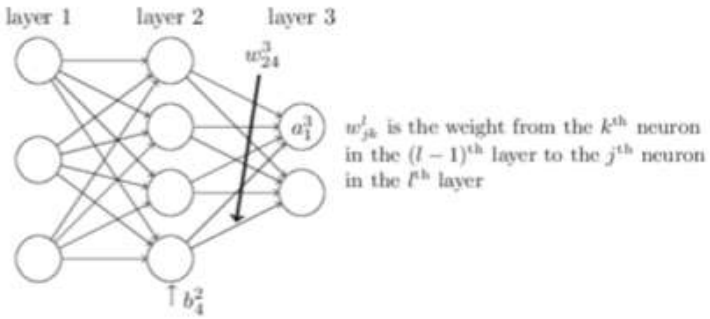
What class is predicted for points just on the outside of this boundary (farther from the origin)? --

Ask for Help

5) BACKPROPAGATION

We have seen in the notes how to train multi-layer neural networks as classifiers using stochastic gradient descent (SGD). One of the key steps in the SGD method is the evaluation of the gradient of the loss function with respect to the model parameters. In the notes the focus was on models that have at most one hidden layer. In this problem, you will derive the backpropagation method, now for a general L-layer neural network.

The setup, figure and notation for this problem are drawn from Neural Networks and Deep Learning (http://neuralnetworksanddeeplearning.com/) by Michael Nielsen.



We will use the notation introduced earlier: b^l_j is the bias of the j^{th} neuron in the l^{th} layer, a^l_j is the activation of the j^{th} neuron in the l^{th} layer, and w^l_{jk} denotes 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 to be minimized is C, then the equations describing the network are:

$$a^l_j = f\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right), \qquad \text{for } l = 1, \ldots, L$$
$$\text{Loss} = C(a^L).$$

Let's define the weighted input to the neurons in layer l by $z^l \equiv w^l a^{l-1} + b^l$ (as a result, $a^l = f(z^l)$). Note that z^l, a^l, and b^l are vectors and w^l is the matrix of layer weights w^l_{jk} weights. Let the error δ^l_j of neuron j in layer l be $\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$. Let δ^l denote the vector of errors associated with layer l.

The backpropagation algorithm will give us a way of computing δ^l for every layer, and then relating those errors to the quantities of real interest, the partial derivatives of the loss $\frac{\partial C}{\partial w^l_{jk}}$ and $\frac{\partial C}{\partial b^l_j}$.

The following four are the basic backpropagation equations:

$$(A) \quad \delta^L_j = \frac{\partial C}{\partial a^L_j} f'(z^L_j), \qquad (B) \quad \delta^l_j = \sum_k w^{l+1}_{kj} \delta^{l+1}_k f'(z^l_j).$$

$$(C) \quad \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l, \qquad (D) \quad \frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

We will now work through the derivation of these equations.

The critical steps are several applications of the chain rule.

- We know that the loss is a function of the activations in the output layer L, that is, $C(a^L)$, and those activations depend on the weighted inputs to the neurons in that layer, recall that $a^l = f(z^l)$. Focusing on the inputs to the j^{th} unit, we can apply the chain rule to get:

  $$\frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L}$$

  But, because the activation of the j^{th} unit depends only on the weighted input for that unit, we get:

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

- When considering the impact on the loss of a change in the input to hidden unit in layer l, things are a bit more complicated, but we can characterize the effect of that change on the units in the next $(l+1)$ layer, since that ultimately will determine the output of the network. So we have that C is a function of the inputs to all the units in the $l+1$ layer (let k range over those) and each of those inputs is affected by a change in the input to the j^{th} unit in layer l. So, applying the chain rule again, we have:

  $$(2) \quad \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l}$$

- Considering the impact of an individual weight on the loss:

  $$(3) \quad \frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l}$$

- Finally, the impact of an individual offset on the loss:

  $$(4) \quad \frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l}$$

Here are some other basic facts:

Because $f(z_j^l) = a_j^l$, we have that:

$$(5) \quad f'(z_j^l) = \frac{\partial a_j^l}{\partial z_j^l}$$

By our definition above:

$$(6) \quad \delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

From our definition of z^l above we get the component version:

$$(7) \quad z_k^l = \sum_j w_{kj}^l a_j^{l-1} + b_k^l$$

From which follows:

$$(8) \quad \frac{\partial z_j^l}{\partial w_{jk}^l} = a_j^{l-1} \qquad (9) \quad \frac{\partial z_j^l}{\partial b_j^l} = 1$$

Indicate for each of the following "proof" steps which of the facts above (if any) are needed to justify the step.

A) To prove:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L)$$

i) $\delta_j^L \equiv \frac{\partial C}{\partial z_j^L}$

Enter a python list of fact numbers.

Ask for Help

ii) $\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}$

Enter a python list of fact numbers.

Ask for Help

iii) $\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L)$

Enter a python list of fact numbers.

Ask for Help

This establishes the first equation.

B) To prove:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} f'(z_j^l)$$

i) $\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$

Enter a python list of fact numbers.

Ask for Help

ii) $\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}$

Enter a python list of fact numbers.

Ask for Help

iii) $\delta_j^l = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$

Enter a python list of fact numbers.

Ask for Help

We also have:

iv) $z_k^{l+1} = \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1}$

Enter a python list of fact numbers.

Ask for Help

Differentiating and removing zero terms:

v) $\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} f'(z_j^l)$

Enter a python list of fact numbers.

Ask for Help

Combining (iii) and (v):

vi) $\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} f'(z_j^l)$

Enter a python list of fact numbers.

Ask for Help

This establishes the second equation.

C) To prove:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

i) $\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l}$

Enter a python list of fact numbers.

Ask for Help

ii) $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

Enter a python list of fact numbers.

Ask for Help

This establishes the third equation.

D) To prove:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

i) $\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l}$

Enter a python list of fact numbers.

Ask for Help

ii) $\frac{\partial C}{\partial b_j^l} = \delta_j^l$

Enter a python list of fact numbers.

Ask for Help

This establishes the fourth equation.