CSC 578 Quiz#3 Sample Solutions

1. NNDL code

```
# backward pass
delta = self.cost_derivative(activations[-1], y) *
                sigmoid_prime(zs[-1])  # line (1)
nabla_b[-1] = delta  # line (2)
nabla_w[-1] = np.dot(delta, activations[-2].transpose())  # line (3)
```

<u>ANSWER</u>:

Line (1) is computing the error at the output layer δ_j^L (for all component *j*, in a column vector of the output layer). The error is defined in the NNDL chapter 2, $\delta_j^L \equiv \frac{\partial C}{\partial a_j^L} \cdot \sigma'\left(z_j^L\right)$ … (BP1). The first term $\frac{\partial C}{\partial a_j^L}$ is obtained by calling the function cost_derivative() with two necessary arguments (the activations of the components in the output layer (activations[-1]) and the target output y). The second term $\sigma'\left(z_j^L\right)$ is straight-forwardly obtained by calling the derivative of the activation function sigmoid (sigmoid_prime()) passing in the z's of the components of the output layer. The returned values of both calls are a column vector, of shape (n, 1) where n is the number of nodes on the output layer. They are component-wise multiplied (by *), as consistent with the definition of δ_j^L.

Line (2) is propagating the error (delta, δ_j^L) to the **bias** on the output layer. Biases are not connected to any weights, so the error of the node/component directly corresponds to delta -- $\frac{\partial C}{\partial b_j^L} = \delta_j^L$ … (BP3).

Line (3) is propagating the error to the **weights** between the output layer (with *n* nodes) and the layer before it (with *m* nodes). By matrix multiplication of the delta (whose shape is (n, 1)) by the transpose of the activations of the 2nd to last layer (whose shape is (1, m)), the resulting 2D array of shape (n, m) would contain the error propagated back to all weights between the two layers (from m nodes to n nodes, transposed). For a single weight w_{ji} (which connects the jth node in the output layer and the ith node in the previous layer), the amount of error propagated back is the error at the jth output node (δ_j^L) weighted by the activation of the ith node in the previous layer (a_i^{L-1}) -- $\frac{\partial C}{\partial w_{ji}} = a_i^{L-1} \cdot \delta_j^L$ … (BP4). The notation · in the formula implies a matrix multiplication.

Then the lines (2) and (3) together are computing the gradient of the error of the cost function.

2. NNDL Exercise

In the single-neuron discussion at the start of this section, I argued that the cross-entropy is small if $\sigma(z) \approx y$ for all training inputs. The argument relied on y being equal to either 0 or 1. This is usually true in classification problems, but for other problems (e.g., regression problems) y can sometimes take values intermediate between 0 and 1. Show that the cross-entropy is still minimized when $\sigma(z)=y$ for all training inputs. When this is the case the cross-entropy has the value:

$$C = -\frac{1}{n}\sum_x [y \ln y + (1-y)\ln(1-y)]$$

<u>ANSWER</u>: For this quiz, it suffices to show this is the case for a couple of given values of y. For instance, for y = 0.3, the cost is minimum for a = y = 0.3 (compared to other values of a).

y	a	-y*ln(a)	-(1-y)*ln(1-a)	C	
0.3	0.1	0.690776	0.073752361	0.764528	
	0.2	0.482831	0.156200486	0.639032	
	0.3	0.361192	0.249672461	0.610864	<= min
	0.4	0.274887	0.357577937	0.632465	
	0.5	0.207944	0.485203026	0.693147	
	0.6	0.153248	0.641403512	0.794651	
	0.7	0.107002	0.842780963	0.949783	
	0.8	0.066943	1.126606539	1.19355	
	0.9	0.031608	1.611809565	1.643418	

You can also show this formally (although it was not required). Here I give a sketch/explanation of a proof.

In Information Theory, cross entropy is a measure of divergence between two probability distributions. It applies to any probability distribution, which takes any values between 0 and 1 (and not just 0 and 1). For distributions P and Q, cross entropy measures how much Q, which is an approximation of the **true** distribution P, differs from P. The general form of cross entropy is

$$H(p,q) = -\sum_x p(x) \cdot \log q(x)$$

When we apply cross entropy to classification, where the target output y is approximated by $\hat{y}$, cross entropy indicates how well the approximation came close to the target.

$$H(y,\hat{y}) = -\sum_x y \cdot \log \hat{y}$$

Then for a binary classification problem where a variable takes on two values (only), $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$, the formula becomes

$$H(p,q) = -y \cdot \log \hat{y} - (1-y) \cdot \log(1-\hat{y}).$$

The rest is explained by the proven properties of cross entropy.

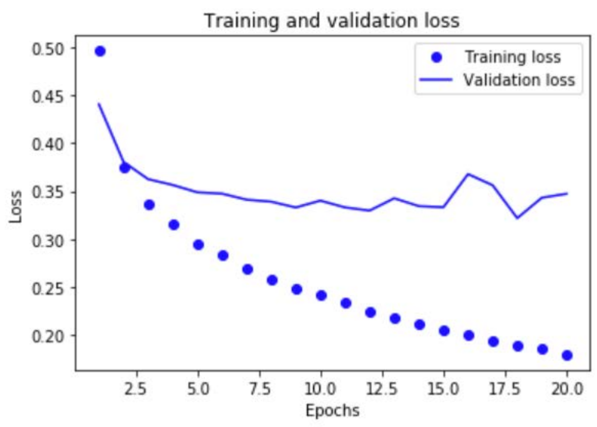
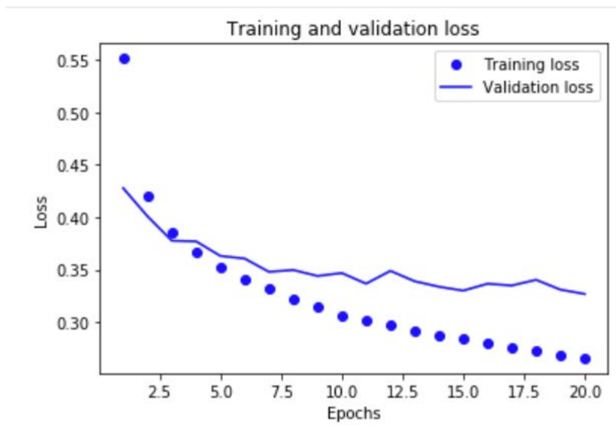
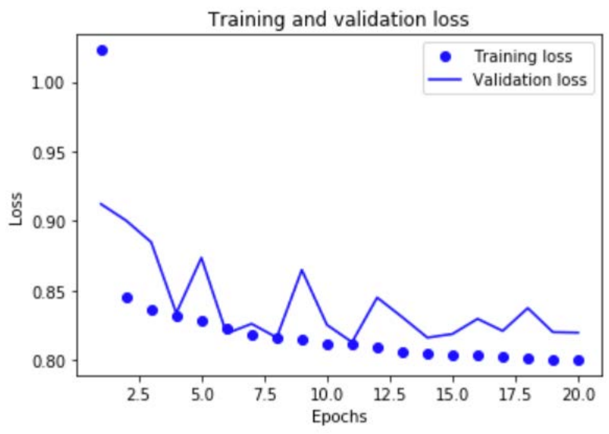
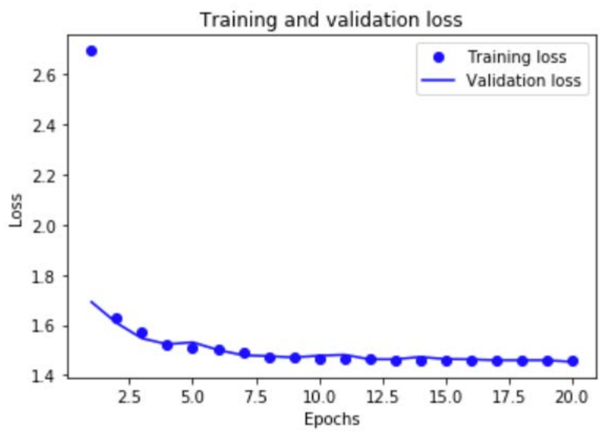
- Cross Entropy $H(p,q) = H(p) + D_{KL}(p||q)$, where
    - $H(p)$ is the entropy of p (the true probability distribution) $H(p) = -\sum_x p(x) \cdot \log p(x)$
    - $D_{KL}(p||q)$ is the KL divergence of q from p, $D_{KL}(p||q) = \sum_x p(x) \cdot \log p(x) - \sum_x p(x) \cdot \log q(x)$.
- Both $H(p) \geq 0$ and $D_{KL}(p||q) \geq 0$.
- $D_{KL}(p||q) = 0$ when p and q are identical – when the approximation was perfect, i.e., $p = q$. And that is when cross entropy minimizes.

3. Tensorflow Tutorial ("basic_classification.ipynb")

Answer will vary. Any good attempt is fine, as long as the experiment was well thought out and to the point, and clearly described (in at least 0.5 pages).

I did a very quick experiment, focusing on reducing overfitting rather than achieving better accuracy or minimizing loss. I experimented with regularization (by Keras Regularizer) and dropout layer to reduce overfitting, since the original setting was already overfitting.

Since the data instances are randomized during learning, we cannot make a fair comparison between experiment runs. But here are some result graphs (with 20 epochs for each run).

(1) Original setting [784, 128, 10] 	(2) 0.3 **Dropout** (for the hidden layer)  Yes, less overfitting (as expected)
(3) **L2 regularizer** with lmbda 0.01 for both hidden and output layers  Less overfitting. But noticeable fluctuation for validation loss (but as expected).	(4) **L1 regularizer** with lmbda 0.01 for both hidden and output layers  !!! Validation loss is much smoother.. but it needs more experiments to draw any conclusion.