Feed-forward neural networks

CS/DS 541 2020 – Jacob Whitehill (jrwhitehill@wpi.edu)

In Homework 4, you are tasked with implementing backpropagation for a multi-layer neural network. In these notes, a derivation for the special case of 3 layers is given. This may be helpful for implementing the more general case. **Notation**: superscripts denote either the *layer index* or the *example*; subscripts denote an *element* of a matrix or vector.

1 Computing partial derivatives in 3-layer neural network

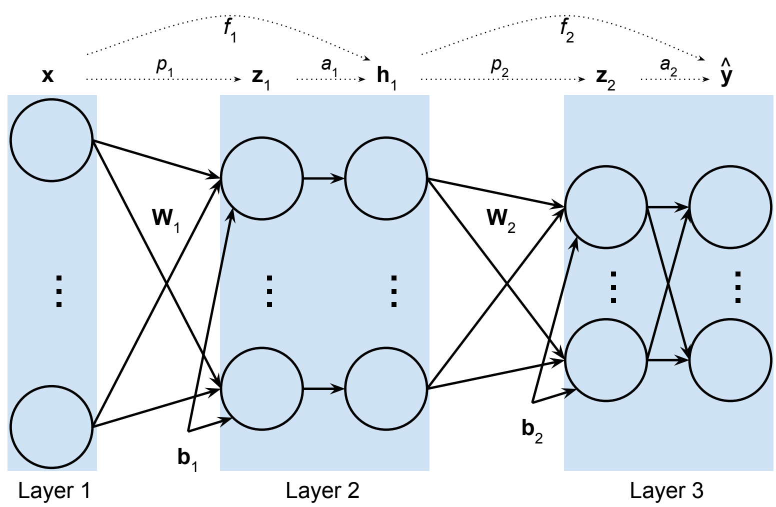
Consider a 3-layer neural network designed to classify images of hand-written digits from the MNIST dataset. Similarly to Homework 4, the input to the network will be a 28×28-pixel image (converted into a 784-dimensional vector); the output will be a vector of 10 probabilities (one for each digit). Specifically, the network you create should implement a function $f : \mathbb{R}^{784} \to \mathbb{R}^{10}$, where:

$$\begin{aligned}
\mathbf{z}^{(1)} &= p_1(\mathbf{x}) = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{z}^{(2)} &= p_2(\mathbf{h}^{(1)}) = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)} \\
\mathbf{h}^{(1)} &= f_1(\mathbf{x}) = a_1(p_1(\mathbf{x})) \\
\hat{\mathbf{y}} &= f_2(\mathbf{h}^{(1)}) = a_2(p_2(\mathbf{h}^{(1)})) \\
f(\mathbf{x}) &= f_2(f_1(\mathbf{x})) = a_2(p_2(a_1(p_1(\mathbf{x}))))
\end{aligned}$$

For the activation functions a_1, a_2 in your network, use:

$$\begin{aligned}
a_1(\mathbf{z}^{(1)}) &= \mathrm{relu}[\mathbf{z}^{(1)}] \\
a_2(\mathbf{z}^{(2)}) &= \mathrm{softmax}(\mathbf{z}^{(2)})
\end{aligned}$$

The network specified above is shown in the figure below:



As usual, the cross-entropy cost function should be

$$J(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{10} y_k^{(i)} \log \hat{y}_k^{(i)}$$

where n is the number of examples.

Note that the figure above is rather pedantic in its level of detail compared to most neural network texts, which typically omit $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}$. The reason for showing these terms is that it helps to make explicit how to compute the gradient terms for use in gradient descent. For example, to optimize $\mathbf{W}^{(1)}$, all we need to compute is the expression below (which is the matrix-product of multiple Jacobian matrices)

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \frac{\partial J}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}}$$

2 Matrix vectorization

As described during lecture, a common technique in matrix calculus is to "vectorize" a matrix into the concatenation of either its rows or its columns. For example, if a matrix is $m \times n$, then it can be vectorized into either a $1 \times (mn)$ row vector or a $(mn) \times 1$ column vector (depending on what is desired). This is very useful when computing the Jacobian of a vector-valued function with respect to a matrix input. For example, $\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}}$ would actually need to be a 3-dimensional tensor since it represents the gradient of a vector with respect to a matrix. However, by vectorizing $\mathbf{W}^{(1)}$, we can compute instead a Jacobian with dimensionality $n \times (km)$ (where $\mathbf{z}^{(1)}$ has dimensionality n and $\mathbf{W}^{(1)}$ is a $k \times m$ matrix). Using vectorization, we can rewrite the equation above as:

$$\frac{\partial J}{\partial \text{vec}[\mathbf{W}^{(1)}]} = \underbrace{\frac{\partial J}{\partial \hat{\mathbf{y}}}}_{1\times 10} \underbrace{\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(2)}}}_{10\times 10} \underbrace{\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(1)}}}_{10\times 30} \underbrace{\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{z}^{(1)}}}_{30\times 30} \underbrace{\frac{\partial \mathbf{z}^{(1)}}{\partial \text{vec}[\mathbf{W}^{(1)}]}}_{30\times(30*784)}$$

As an example, the equation above shows the dimensions of each Jacobian matrix assuming that the dimensionality of the hidden layer ($\mathbf{h}^{(1)}$) is 30. Once you have computed the gradient of J with respect to the (vectorized) vec$[\mathbf{W}^{(1)}]$, it is easy to reshape the vectorized gradient $\frac{dJ}{\partial \text{vec}[\mathbf{W}^{(1)}]}$ $(1 \times (30 * 784))$ into the gradient $\frac{dJ}{\partial \mathbf{W}^{(1)}}$ (30×784).

3 Deriving gradient terms for $\mathbf{W}^{(1)}$

Here we derive the gradient update for one particular weight matrix in the 3-layer neural network. The same idea can be used to obtain all the weight matrices and bias vectors.

$$
\begin{aligned}
\frac{\partial J}{\partial \hat{\mathbf{y}}} &= \begin{bmatrix} -\frac{y_1}{\hat{y}_1} & \cdots & -\frac{y_{10}}{\hat{y}_{10}} \end{bmatrix} \\
\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(2)}} &= \begin{bmatrix} \hat{y}_1(1-\hat{y}_1) & -\hat{y}_1\hat{y}_2 & \cdots & & -\hat{y}_1\hat{y}_{10} \\ -\hat{y}_2\hat{y}_1 & \hat{y}_2(1-\hat{y}_2) & -\hat{y}_2\hat{y}_3 & \cdots & -\hat{y}_2\hat{y}_{10} \\ -\hat{y}_3\hat{y}_1 & -\hat{y}_3\hat{y}_2 & \hat{y}_3(1-\hat{y}_3) & \cdots & -\hat{y}_3\hat{y}_{10} \\ \vdots & \cdots & & \ddots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \hat{y}_{10}(1-\hat{y}_{10}) \end{bmatrix} \\
\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(1)}} &= \begin{bmatrix} \frac{\partial (\mathbf{z}^{(2)})_1}{\partial (\mathbf{h}^{(1)})_1} & \frac{\partial (\mathbf{z}^{(2)})_1}{\partial (\mathbf{h}^{(1)})_2} & \cdots & \frac{\partial (\mathbf{z}^{(2)})_1}{\partial (\mathbf{h}^{(1)})_{30}} \\ \frac{\partial (\mathbf{z}^{(2)})_2}{\partial (\mathbf{h}^{(1)})_1} & \frac{\partial (\mathbf{z}^{(2)})_2}{\partial (\mathbf{h}^{(1)})_2} & \cdots & \frac{\partial (\mathbf{z}^{(2)})_2}{\partial (\mathbf{h}^{(1)})_{30}} \\ & \cdots & \vdots & \\ \frac{\partial (\mathbf{z}^{(2)})_{10}}{\partial (\mathbf{h}^{(1)})_1} & \frac{\partial (\mathbf{z}^{(2)})_{10}}{\partial (\mathbf{h}^{(1)})_2} & \cdots & \frac{\partial (\mathbf{z}^{(2)})_{10}}{\partial (\mathbf{h}^{(1)})_{30}} \end{bmatrix} \\
&= W_2 \\
\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{z}^{(1)}} &= \begin{bmatrix} \mathrm{relu}'[(\mathbf{z}^{(1)})_1] & 0 & \cdots & 0 \\ 0 & \mathrm{relu}'[(\mathbf{z}^{(1)})_2] & \cdots & 0 \\ 0 & \cdots & \ddots & 0 \\ 0 & \cdots & \cdots & \mathrm{relu}'[(\mathbf{z}^{(1)})_{30}] \end{bmatrix} \\
& \text{where} \quad \mathrm{relu}'[x] = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x \le 0 \end{cases} \\
\frac{\partial \mathbf{z}^{(1)}}{\partial \mathrm{vec}[\mathbf{W}^{(1)}]} &= \begin{bmatrix} \frac{\partial (\mathbf{z}^{(1)})_1}{\partial (\mathbf{W}^{(1)})_{1,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_1}{\partial (\mathbf{W}^{(1)})_{1,784}} & \frac{\partial (\mathbf{z}^{(1)})_1}{\partial (\mathbf{W}^{(1)})_{2,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_1}{\partial (\mathbf{W}^{(1)})_{2,784}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_1}{\partial (\mathbf{W}^{(1)})_{30,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_1}{\partial (\mathbf{W}^{(1)})_{30,784}} \\ \frac{\partial (\mathbf{z}^{(1)})_2}{\partial (\mathbf{W}^{(1)})_{1,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_2}{\partial (\mathbf{W}^{(1)})_{1,784}} & \frac{\partial (\mathbf{z}^{(1)})_2}{\partial (\mathbf{W}^{(1)})_{2,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_2}{\partial (\mathbf{W}^{(1)})_{2,784}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_2}{\partial (\mathbf{W}^{(1)})_{30,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_2}{\partial (\mathbf{W}^{(1)})_{30,784}} \\ & & & & & \vdots & & & & \\ \frac{\partial (\mathbf{z}^{(1)})_{30}}{\partial (\mathbf{W}^{(1)})_{1,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_{30}}{\partial (\mathbf{W}^{(1)})_{1,784}} & \frac{\partial (\mathbf{z}^{(1)})_{30}}{\partial (\mathbf{W}^{(1)})_{2,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_{30}}{\partial (\mathbf{W}^{(1)})_{2,784}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_{30}}{\partial (\mathbf{W}^{(1)})_{30,1}} & \cdots & \frac{\partial (\mathbf{z}^{(1)})_{30}}{\partial (\mathbf{W}^{(1)})_{30,784}} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}^\top & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{x}^\top & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{x}^\top & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{x}^\top \end{bmatrix}
\end{aligned}
$$

4 Analytical simplification

Now, watch what happens when we multiply the matrices together (and compare with Algorithm 6.4 from the book – note that below we are using no regularization and hence $\lambda = 0$):

$$\begin{aligned}
\frac{\partial J}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(2)}} &= \begin{bmatrix} \hat{y}_1 - y_1 & \cdots & \hat{y}_{10} - y_{10} \end{bmatrix} \\
&= (\hat{\mathbf{y}}^\top - \mathbf{y}^\top) \\
\frac{\partial J}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(2)}}\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(1)}} &= (\hat{\mathbf{y}}^\top - \mathbf{y}^\top)\mathbf{W}^{(2)} \\
\frac{\partial J}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(2)}}\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(1)}}\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{z}^{(1)}} &= (\hat{\mathbf{y}}^\top - \mathbf{y}^\top)\mathbf{W}^{(2)} \begin{bmatrix} \mathrm{relu}'[(\mathbf{z}^{(1)})_1] & 0 & \cdots & 0 \\ 0 & \mathrm{relu}'[(\mathbf{z}^{(1)})_2] & \cdots & 0 \\ 0 & \cdots & \ddots & 0 \\ 0 & \cdots & \ldots & \mathrm{relu}'[(\mathbf{z}^{(1)})_{30}] \end{bmatrix} \\
&= (\hat{\mathbf{y}}^\top - \mathbf{y}^\top)\mathbf{W}^{(2)} \odot \begin{bmatrix} \mathrm{relu}'[(\mathbf{z}^{(1)})_1] & \cdots & \mathrm{relu}'[(\mathbf{z}^{(1)})_{30}] \end{bmatrix} \\
&= (\hat{\mathbf{y}}^\top - \mathbf{y}^\top)\mathbf{W}^{(2)} \odot \mathrm{relu}' \left[{\mathbf{z}^{(1)}}^\top \right] \\
& \quad \text{where } \odot \text{ represents Hadamard product.}
\end{aligned}$$

To obtain the final expression, first define

$$\begin{aligned}
\mathbf{g}^\top &= (\hat{\mathbf{y}}^\top - \mathbf{y}^\top)\mathbf{W}^{(2)} \odot \mathrm{relu}' \left[{\mathbf{z}^{(1)}}^\top \right] \\
&= \begin{bmatrix} g_1 & g_2 & \cdots & g_{30} \end{bmatrix}
\end{aligned}$$

Then:

$$\begin{aligned}
\frac{\partial J}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(2)}}\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(1)}}\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{z}^{(1)}}\frac{\partial \mathbf{z}^{(1)}}{\partial \mathrm{vec}[\mathbf{W}^{(1)}]} &= \mathbf{g}^\top \begin{bmatrix} \mathbf{x}^\top & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{x}^\top & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{x}^\top & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{x}^\top \end{bmatrix} \\
&= \begin{bmatrix} g_1\mathbf{x}^\top & g_2\mathbf{x}^\top & \cdots & g_{30}\mathbf{x}^\top \end{bmatrix}
\end{aligned}$$

We can now "unvectorize" $\frac{\partial J}{\partial \mathrm{vec}[\mathbf{W}^{(1)}]}$ into a matrix with 30 rows and 784 columns:

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{W}^{(1)}} &= \begin{bmatrix} g_1\mathbf{x}^\top \\ g_2\mathbf{x}^\top \\ \vdots \\ g_{30}\mathbf{x}^\top \end{bmatrix} \\
&= \mathbf{g}\mathbf{x}^\top \quad \text{(outer product)}
\end{aligned}$$

5 Deriving gradient terms for $\mathbf{b}^{(1)}$

Let's derive $\frac{\partial J}{\partial \mathbf{b}^{(1)}}$. Since $\mathbf{b}^{(1)}$ is just a vector rather than a matrix, we won't have to vectorize it.

$$\frac{\partial J}{\partial \mathbf{b}^{(1)}} = \underset{1\times 10}{\frac{\partial J}{\partial \hat{\mathbf{y}}}} \quad \underset{10\times 10}{\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(2)}}} \quad \underset{10\times 30}{\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(1)}}} \quad \underset{30\times 30}{\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{z}^{(1)}}} \quad \underset{30\times 30}{\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{b}^{(1)}}}$$

Most of the terms above are the same as for $\frac{\partial J}{\partial \mathbf{W}^{(1)}}$ except the last one:

$$\begin{aligned}
\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{b}^{(1)}} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \\
&= \mathbf{I}
\end{aligned}$$

which is diagonal since each $(\mathbf{z}^{(1)})^{(i)}$ depends only on the ith component of $\mathbf{b}^{(1)}$. We can simplify the whole sequence of matrix products to become:

$$\frac{\partial J}{\partial \mathbf{b}^{(1)}} = \mathbf{g}$$

6 Deriving gradient terms for $\mathbf{W}^{(2)}$

It can be shown, using the same principle as illustrated above, that

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = (\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^{(1)\top}$$

7 Forward and backward propagation

As shown above, when computing the gradients of the cost function J with respect to each weight matrix and bias term, there is a lot of redundant computation (e.g., $(\hat{\mathbf{y}} - \mathbf{y})$, $\mathbf{g}$) – even after simplifying each gradient analytically. Rather than re-compute these matrices/vectors repeatedly, we can compute *all* gradient terms using a two-pass sweep: (1) forward propagation and (2) backward propagation.

7.1 Forward propagation

The job of forward propagation (through layers $1, 2, \ldots, l$) is to determine the value of each hidden layer $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \ldots$ (in the 3-layer neural network there is only one hidden layer, but this is only a special case) and also the final output layer $\hat{\mathbf{y}}$:

1. Let $\mathbf{h}^{(0)} = \mathbf{x}$.
2. For $i = 1, \ldots, l-1$ (where l is the number of layers, including input and output layers):
   (a) Let $\mathbf{z}^{(i)} = p_i(\mathbf{h}^{(i-1)}) = \mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}$.
   (b) Let $\mathbf{h}^{(i)} = a_i(\mathbf{z}^{(i)})$.
3. Let $\hat{\mathbf{y}} = \mathbf{h}^{(l-1)}$.

7.2 Back(ward)-prop(agation)

Once $\hat{\mathbf{y}}$ has been computed, we can compute $(\hat{\mathbf{y}} - \mathbf{y})$, which is known as the *error* since it expresses the difference between the ground-truth and the network's current predictions. By iteratively multiplying – layer-by-layer $(l-1, l-2, \ldots, 2, 1)$ – by the derivative of each layer's activation function as well as its weight matrices, we can compute and update our vector $\mathbf{g}$:

1. Let $\mathbf{g}^\top = \frac{\partial J}{\partial \hat{\mathbf{y}}}$.
2. Let $\mathbf{h}^{(l-1)} = \hat{\mathbf{y}}$.

3. For $i = l-1, \dots, 1$:

   (a) Let $\mathbf{g}^\top \leftarrow \mathbf{g}^\top \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{z}^{(i)}}$.

   (b) Let $\frac{\partial J}{\partial \mathbf{b}^{(i)}} = \mathbf{g}$.

   (c) Let $\frac{\partial J}{\partial \mathbf{W}^{(i)}} = \mathbf{g}{\mathbf{h}^{(i-1)}}^\top$.

   (d) Let $\mathbf{g}^\top \leftarrow \mathbf{g}^\top \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{h}^{(i-1)}} = \mathbf{g}^\top \mathbf{W}^{(i)}$.

7.2.1 Special case for NNs with softmax outputs with cross-entropy loss

In the special case such as Homework 4 in which (1) the network is trained using cross-entropy loss and (2) the network has a softmax output layer, the algorithm above can be simplified slightly. In particular, we can skip the first step (where we initialize $\mathbf{g}$ to $\frac{\partial J}{\partial \hat{\mathbf{y}}}$); instead, in the *first* iteration of the loop we replace step (a) with simply "Let $\mathbf{g}^\top = \hat{\mathbf{y}} - \mathbf{y}$". Where does this result come from? You proved this in Homework 3, problem 2.

8 Convexity and parameter initialization

In contrast to logistic or softmax regression, the cost function J of a non-linear 3-layer neural network is generally **not convex**. This means that, depending on the initial values for the weight matrices and bias terms, stochastic gradient descent (SGD) could end up in a different place. While weight initialization is an important topic in its own right, for now let's just examine a few important cases:

1. Suppose that you initialized all the parameters ($\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}$) in the 3-layer neural network in the example above to be 0. How will SGD perform?

   - If $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ are 0, then $\mathbf{h}^{(1)} = 0$, and hence $\mathbf{z}^{(2)} = 0$, so that $\hat{\mathbf{y}}$ is just the uniform distribution over c classes. Moreover, if relu$'$ is defined so that relu$'[0] = 0$, then the gradients with respect to all the weights and bias vectors will be 0, and the network will never learn.

2. Suppose that you initialized all the parameters ($\mathbf{W}^{(2)}, \mathbf{b}^{(2)}$) to be 0 (but $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$ are non-zero). How will SGD perform?

   - In this case, $\hat{\mathbf{y}}$ will *initially* just be the uniform distribution. However, assuming $\mathbf{h}^{(1)}$ is initially non-zero (and non-uniform), $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$ can still improve. Very soon, the outputs of the network will be non-uniform, and the network can train to good accuracy.