Homework 1

Joshua Michalenko
ELEC 677: Deep Learning
Dr. Ankit Patel

10/04/16

1 Problem 1: Backpropogation in a simple Nueral Network

1.1 Part A: Dataset

The plot of the Two Moons dataset is displayed in Figure 1

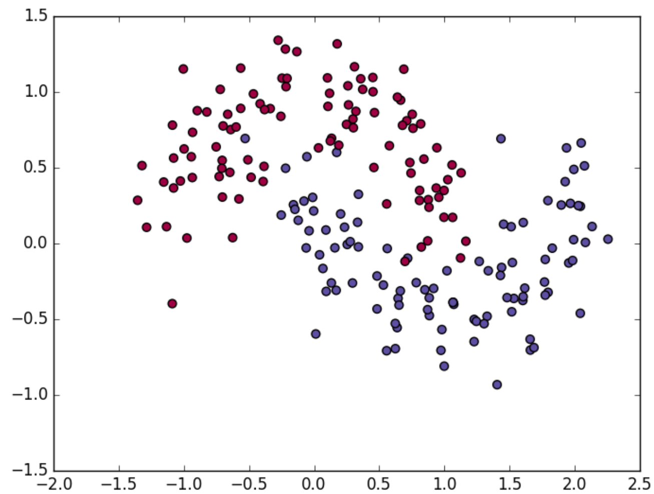

Figure 1: Two moons dataset displayed in matplotlib

1.2 Part B - Activation Functions

1.2.1 Part B1 - Implement activation functions

If you look in my code you can see I implemented the activations function.

1.2.2 Part B2 - Derivative Derivations

Part B2.1 - Derivative for Sigmoid function

$$
\begin{aligned}
\sigma(z) = f(z) = \frac{1}{1+e^{-z}} &= (1+e^{-z})^{-1} \\
\frac{df}{dz} := -1 * (1+e^{-z})^{-2} * -e^{-z} &= e^{-z}(1+e^{-z})^{-2} \quad \text{(by chain rule)} \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1-e^{-z}+1}{(1+e^{-z})^2} \quad \text{(add and subtract a 1)} \\
&= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2} \quad \text{(split terms)} \\
&= \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} \\
&= \sigma(z) - \frac{1}{(1+e^{-z})^2} \\
&= \sigma(z) - \left(\frac{1}{(1+e^{-z})}\right)^2 \\
&= \sigma(z) - \sigma(z)^2 \\
&\boxed{f'(z) = \sigma(z)(1-\sigma(z))}
\end{aligned}
$$

Part B2.2 - Derivative for Tanh function

$$
\begin{aligned}
f(z) = \tanh(z) &= \frac{\sinh(z)}{\cosh(z)} \\
&= \frac{e^z - e^{-z}}{e^z + e^{-z}} \\
\frac{df}{dz} : &= \frac{\cosh(z) * \sinh'(z) - \sinh(z) * \cosh'(z)}{(\cosh(z))^2} \quad \text{(by quotient rule)} \\
&= \frac{\cosh(z)^2 - \sinh(z)^2}{(\cosh(z))^2} \\
&= \frac{\cosh(z)^2}{\cosh(z)^2} - \frac{\sinh(z)^2}{\cosh(z)^2} \\
&\boxed{f'(z) = 1 - \tanh^2(z)}
\end{aligned}
$$

Part B2.3 - Derivative for ReLu function

$$
\mathrm{ReLu}(z) = f(z) = \max(0, z)
$$

$$
\boxed{\frac{df}{dz} := \begin{cases} 0 & z \leq 0 \\ 1 & z \geq 0 \end{cases}} \text{(I think the dirivitive here is pretty obvious)}
$$

1.2.3 Part B3 - Implement activation function gradiants

If you look in my code you can see I implemented the activations function gradiants.

1.3 Part C - Build the Neural Network

If you look in my code you can see I implemented the feedforward and loss functions

1.4 Part D - Backpropogation Derivations

1.4.1 Derivations of Backpropogation Equations

Keep in mind that the process for a single feedforward pass of the network is defined by the following equations with $x \in \Re^p$ being a single data sample with p features, $\mathbf{W_1} \in \Re^{n_1 \times p}$ is a weight matrix transitioning from the input later with p features to the number of units in hidden layer 1 n_1, $b_1 \in \Re^{n_1}$ is a bias vector, $z_1 \in \Re^{n_1}$ are a vector of potentials for layer 1, $a_1 \in \Re^{n_1}$ the corresponding activations, and $\hat{y}$ are the resulting probabilities. $\mathbf{W_2} \in \Re^{n_2 \times n_1}$, $a_1 \in \Re^{n_1}$, $z_2 \in \Re^{n_2}$

$$
\begin{aligned}
z_1 &= \mathbf{W_1} x + b_1 \\
a_1 &= \text{actFun}(z_1) \\
z_2 &= \mathbf{W_2} a_1 + b_2 \\
a_2 = \hat{y} &= \text{softmax}(z_2)
\end{aligned}
$$

Where the softmax function is given by:

$$
\text{softmax}(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{d=1}^{C} e^{z_d}} = \frac{e^{z_c}}{\Sigma_C} \quad \text{for } c = 1 \cdots C
$$

With a cross entropy loss function of :

$$
L(y, \hat{y}) = -\frac{1}{N} \sum_{n \in N} \sum_{j \in C} y_{n,j} \log(\hat{y}_{n,j})
$$

With y being a one hot vector of the correct label, N being the number of training examples and C being the number of classes.

For the following derivations, the derivative of the softmax $\frac{\partial \hat{y}_i}{\partial z_j}$ is useful to know. It is calculated below. It is the derivative of $\hat{y}$ with respect to one of the elements of the input vector z.

$$\hat{y} = \text{softmax}(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{d=1}^{C} e^{z_d}} = \frac{e^{z_c}}{\Sigma_C} \quad \text{for } c = 1 \cdots C$$

$$\begin{aligned}
\text{if } i = j : \quad \frac{\partial y_i}{\partial z_i} &= \frac{\partial \frac{e^{z_i}}{\Sigma_C}}{\partial z_i} \\
&= \frac{e^{z_i}\Sigma_C - e^{z_i}e^{z_i}}{\Sigma_C^2} \\
&= \frac{e^{z_i}}{\Sigma_C}\frac{\Sigma_C - e^{z_i}}{\Sigma_C} \\
&= \frac{e^{z_i}}{\Sigma_C}(1 - \frac{e^{z_i}}{\Sigma_C}) \\
&= y_i(1 - y_i) \\
\text{if } i \neq j : \quad \frac{\partial y_i}{\partial z_j} &= \frac{\partial \frac{e^{z_i}}{\Sigma_C}}{\partial z_j} \\
&= \frac{0 - e^{z_i}e^{z_j}}{\Sigma_C^2} \\
&= -\frac{e^{z_i}}{\Sigma_C}\frac{e^{z_j}}{\Sigma_C} \\
&= -y_i y_j
\end{aligned}$$

The derivative of the cross entropy loss with respect to the second layer inputs are also a useful quantity to calculate, it is derived as follows. Note that I used the cross entropy for one example. Taking out the summation here makes things easier but I'll add it back in later.

$$\begin{aligned}
\frac{\partial L}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_{2_i}} = \frac{\partial L}{\partial z_{2_i}} &= -\sum_{j=1}^{C} \frac{\partial y_j log(\hat{y}_j)}{\partial z_i} \\
&= -\sum_{j=1}^{C} y_j \frac{\partial log(\hat{y}_j)}{\partial z_i} \\
&= -\sum_{j=1}^{C} y_j \frac{1}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_i} \\
&= -\frac{y_i}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i} - \sum_{j \neq i}^{C} \frac{y_j}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_i} \quad \text{substitution from above derivation} \\
&= -\frac{y_i}{\hat{y}_i}\hat{y}_i(1 - \hat{y}_i) - \sum_{j \neq i}^{C} \frac{y_j}{\hat{y}_j}(-\hat{y}_j\hat{y}_i) \\
&= -y_i + y_i\hat{y}_i + \sum_{j \neq i}^{C} y_j\hat{y}_i \\
&= -y_i + \sum_{j=1}^{C} y_j\hat{y}_i \\
&= -y_i + \hat{y}_i\sum_{j=1}^{C} y_j \\
&= \hat{y}_i - y_i
\end{aligned}$$

$\frac{dL}{dW_2}$ **Derivation** We can rewrite the loss function with substituting in the feed-forward equations above as follows.

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{n\in N}\sum_{j\in C} y_{n,j}\log(\hat{y}_{n,j})$$

$$\frac{\partial L}{\partial W_2} := \frac{\partial L}{\partial \hat{y}_{n,j}}\frac{\partial \hat{y}_{n,j}}{\partial z_2}\frac{\partial z_2}{\partial W_2}$$

Since j-th element of z_2 is given by:

$$z_{2_j} = \sum_{k=0}^{n_1} w_{j,k}a_{1_k} + b_{2_j}$$

The derivitive of the j-th element of z_2 w.r.t. weight $w_{j,k}$ is then:

$$\frac{\partial z_{2_j}}{\partial w_{j,k}} = a_{1_k}$$

Using the derivation above for $\frac{\partial L}{\partial z_{2_j}}$

$$\frac{\partial L}{\partial w_{2_{j,k}}} = (\hat{y}_j - y_j)a_{1_k}$$

$$\therefore$$

$$\frac{dL}{dW_2} = (\mathbf{\hat{y}} - \mathbf{y})\mathbf{a_1^T} \in \Re^{n_2\times n_1}$$

The above equation only takes into consideration on data point

We extend this to multiple samples by plugging in the summation we originally took out.

$$\boxed{\frac{dL}{dW_2} = \frac{1}{N}\sum_{n\in N}(\mathbf{\hat{y}_n} - \mathbf{y_n})\mathbf{a_1^T} \in \Re^{n_2\times n_1}}$$

$\frac{dL}{db_2}$ **Derivation** This derivation will be fairly similar to the previous section except now the gradient looks as follows:

$$\frac{\partial L}{\partial b_2} := \frac{\partial L}{\partial \hat{y}_{n,j}} \frac{\partial \hat{y}_{n,j}}{\partial z_2} \frac{\partial z_2}{\partial b_2}$$

Since j-th element of z_2 is given by:

$$z_{2_j} = \sum_{k=0}^{n_1} w_{j,k} a_{1_k} + b_{2_j}$$

The derivitive of the j-th element of z_2 w.r.t. bias b_{2_k} is then:

$$\frac{\partial z_{2_j}}{\partial b_{2_k}} = 1$$

Using the derivation above for $\frac{\partial L}{\partial z_{2_j}}$

$$\frac{\partial L}{\partial b_{2_k}} = (\hat{y}_j - y_j)$$

$$\therefore$$

$$\frac{dL}{db_2} = \hat{\mathbf{y}} - \mathbf{y} \in \Re^{n_2}$$

The above equation only takes into consideration on data point.

We extend this to multiple samples by plugging in the summation we originally took out.

$$\boxed{\frac{dL}{db_2} = \frac{1}{N} \sum_{n \in N} \hat{\mathbf{y}}_{\mathbf{n}} - \mathbf{y}_{\mathbf{n}} \in \Re^{n_2}}$$

$\frac{dL}{dW_1}$ **Derivation** We add one more layer to our previous derivation of $\frac{dL}{dW_2}$ for this derivation. The derivative can be expanded as follows

$$\frac{\partial L}{\partial W_1} := \frac{\partial L}{\partial \hat{y}_{n,j}} \frac{\partial \hat{y}_{n,j}}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial W_1}$$

So we need to figure out the term $\frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial W_1}$ by piecing to out. The j-th element of z_1 is given by:

$$z_{1_j} = \sum_{i=0}^{p} w_{1_{j,i}} x_i + b_{1_j}$$

So the derivative w.r.t. $w_{1_{j,i}}$ is then:

$$\frac{\partial z_{1_j}}{\partial w_{1_{j,i}}} = x_i$$

We know from part b that $\frac{\partial a_1}{\partial z_1}$ is dependent on the type of activation function but we derived all the derivatives of the three activation functions above so stating the derivatives is redundant. See the above part B.

$$\frac{\partial a_1}{\partial z_1} = \text{actFun}'(z_1) \qquad \text{(See part B above)}$$

The last part we need to derive is the $\frac{\partial z_2}{\partial a_1}$ term.

Since the j-th element of z_2 is given by:

$$z_{2_j} = \sum_{k=0}^{n_1} w_{2_{j,k}} a_{1_k} + b_{2_j}$$

we can conclude that the derivative w.r.t. a_{1_k} is given by

$$\frac{\partial z_{2_j}}{\partial a_{1_k}} = w_{2_{j,k}}$$

From above, we know that the other partial derivatives are

$$\frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_{2_i}} = \frac{\partial L}{\partial z_{2_i}} = \hat{y}_i - y_i$$

So if we piece this all together we come up with:

$$\frac{\partial L}{\partial w_{1_{k,p}}} := \frac{\partial L}{\partial \hat{y}_{n,c}} \frac{\partial \hat{y}_{n,c}}{\partial z_{2_c}} \frac{\partial z_{2_c}}{\partial a_{1_k}} \frac{\partial a_{1_k}}{\partial z_{1_k}} \frac{\partial z_{1_k}}{\partial w_{1_{k,p}}}$$

Which I changed some of the indexing to make the most sense. In this format $n \in N$, $c \in C$, , $k \in \{0, ..., n_1 - 1\}$, $p \in \{0, ..., P - 1\}$ with P input features. The resulting derivative with respect to one weight in the first layer $w_{1_{k,p}}$ is then:

$$\frac{\partial L}{\partial w_{1_{k,p}}} = \sum_{c \in C} (\hat{y}_c - y_c) w_{2_{c,k}} \text{actFun}'(z_{1_k}) x_p$$

and the extention to matrix form is then

$$\frac{\partial L}{\partial \mathbf{W_1}} = (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{W_2} \odot \text{actFun}'(\mathbf{z_1}) \mathbf{x^T} \in \Re^{n_1 \times p}$$

If there is morethan one sample (which there always is)

then we averageover all of the samples and the final solution is then

$$\boxed{\frac{\partial L}{\partial \mathbf{W_1}} = \frac{1}{N} \sum_{n \in N} (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{W_2} \odot \text{actFun}'(\mathbf{z_1}) \mathbf{x^T} \in \Re^{n_1 \times p}}$$

$\frac{dL}{db_1}$ **Derivation** The derivation is very similar to the $\frac{dL}{dW_1}$ except the last part of the chain of partial derivatives is changed. The gradient equation of the Loss function with respect to a single bias term in the first layer is

$$\frac{\partial L}{\partial b_{1_k}} := \frac{\partial L}{\partial \hat{y}_{n,c}} \frac{\partial \hat{y}_{n,c}}{\partial z_{2_c}} \frac{\partial z_{2_c}}{\partial a_{1_k}} \frac{\partial a_{1_k}}{\partial z_{1_k}} \frac{\partial z_{1_k}}{\partial b_{1_k}}$$

So we only need to calculate $\frac{\partial z_{1_k}}{\partial b_{1_k}}$ first. Let's look at how the k th value is calculated.

$$z_{1_k} = \sum_{i=0}^{p} w_{1_{k,i}} x_i + b_{1_k}$$

So the derivative w.r.t. b_{1_k} is then:

$$\frac{\partial z_{1_j}}{\partial b_{1_k}} = 1$$

The partial derivative $\frac{\partial z_{1_k}}{\partial b_{1_k}} = 1$ simplified the equation for the partial derivative of the loss function w.r.t. b_{1_k} as

$$\frac{\partial L}{\partial b_{1_k}} = \sum_{c \in C} (\hat{y}_c - y_c) w_{2_{c,k}} \text{actFun}'(z_{1_k})$$

and the resulting matrix form is then

$$\frac{\partial L}{\partial \mathbf{b_1}} = (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{W_2^T} \text{actFun}'(\mathbf{z_1}) \in \Re^{n_1}$$

If there is more than one sample (which there always is) then we

have to sum up and average over all samples and the gradient becomes

$$\boxed{\frac{\partial L}{\partial \mathbf{b_1}} = \frac{1}{N} \sum_{n \in N} (\hat{\mathbf{y}} - \mathbf{y}) \mathbf{W_2^T} \odot \text{actFun}'(\mathbf{z_1}) \in \Re^{n_1}}$$

1.4.2 Implementation of Derivations of Backpropogation Equations

If you look in my code you can see I implemented the backpropogation equations correctly.

1.5 Time to Have Fun - Training!

The decision boundaries are shown for tanh, sigmoid, and relu functions for 3, 10 and 20 hidden hidden units. Using the default 3 hidden units with different activation fuctions, we get Figures 2,3, and 4.

Increasing the number of hidden units to 10 gives us figure 5, 6 and 7

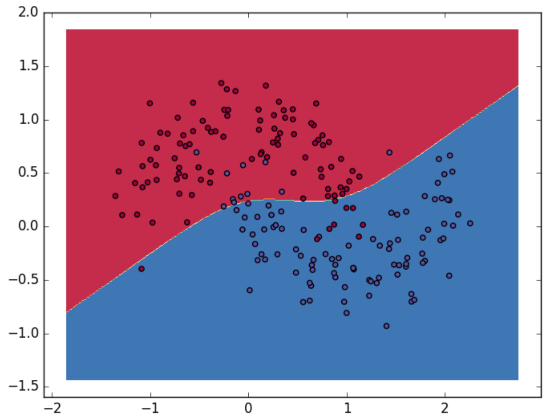

Figure 2: Default settings, 3 hidden units with tanh activation fuctions. Loss = .260

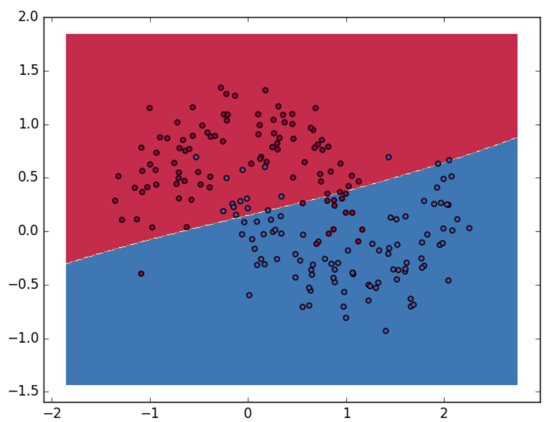

Figure 3: 3 hidden units with sigmoid activation fuctions. Loss = .304

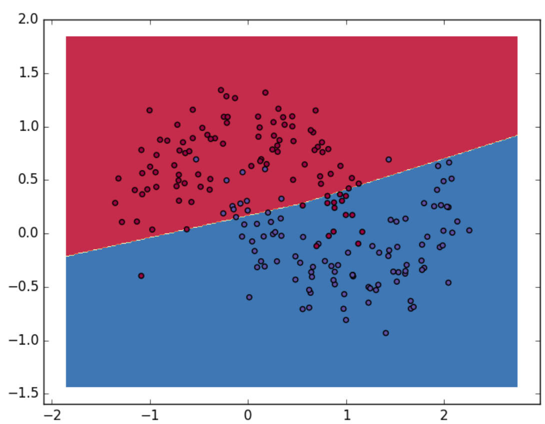

Figure 4: 3 hidden units with ReLU activation fuctions. Loss = .305

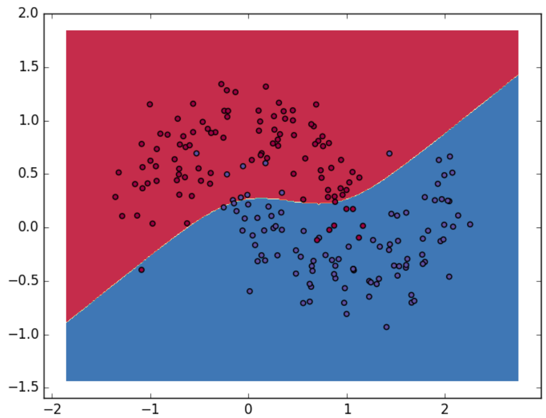

Figure 5: 10 hidden units with tanh activation fuctions. Loss = .246

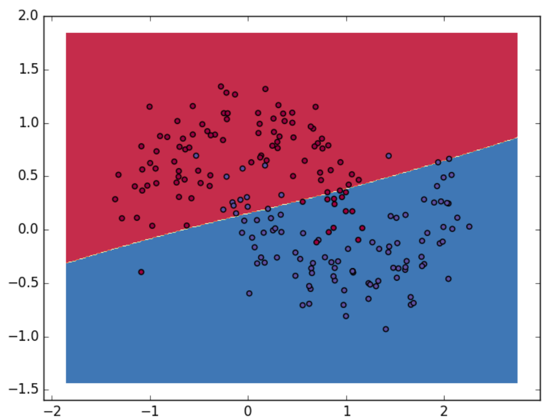

Figure 6: 10 hidden units with sigmoid activation fuctions. Loss = .301

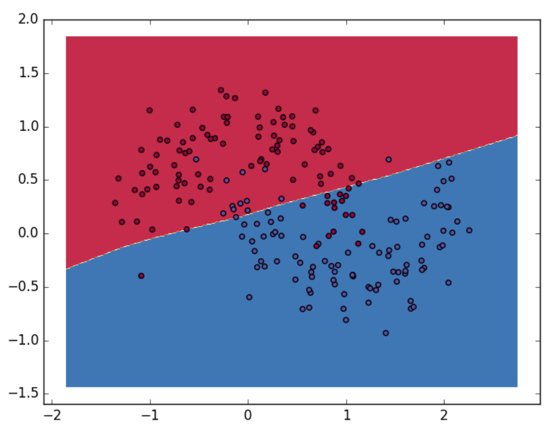

Figure 7: 10 hidden units with ReLU activation fuctions. Loss = .302