1 Definitions

(a) **Convexity**: A function $f : \mathbb{R}^M \to \mathbb{R}$ is convex if and only if $\forall \alpha \in [0, 1]$, i.e., $0 \leq \alpha \leq 1$,

$$f(\alpha \mathbf{x}_1 + (1 - \alpha)\mathbf{x}_2) \leq \alpha f(\mathbf{x}_1) + (1 - t)f(\mathbf{x}_2)$$

You can think of a convex function as a curve or surface that opens upward like a smiley face. It could also be completely flat or flat in some places, but it definitely can't curve/bend downward.

(b) **Multivariate Chain Rule:** Let $f : \mathbb{R}^N \to \mathbb{R}$. Let $g_i : \mathbb{R} \to \mathbb{R}$ for all $i \in \{1, 2, \ldots, N\}$. Let $x \in \mathbb{R}$, $z_i = g_i(x)$, and $y = f(g_1(x), g_2(x), \ldots, g_N(x))$. Then, the multivariate chain rule states that:

$$\frac{d}{dx} f(g_1(x), g_2(x), \ldots, g_N(x)) = \sum_{i=1}^{N} \frac{dy}{dz_i} \frac{dz_i}{dx}$$

If instead, we combined all of the g_i functions into one function $\mathbf{g} : \mathbb{R} \to \mathbb{R}^N$, $\mathbf{z} = \mathbf{g}(x)$, we effectively have the same thing but now using vectors (and partial derivatives). Given $y = f(\mathbf{g}(x))$

$$\frac{\partial}{\partial x} f(\mathbf{g}(x)) = \sum_{i=1}^{N} \frac{\partial y}{\partial z_i} \frac{\partial z_i}{\partial x} = \frac{\partial y}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial x}$$

Note that the above chain rule is written according to numerator layout. If specified to use denominator layout, the order of the derivatives is opposite that of numerator layout.

	Numerator layout	Denominator layout	Notes
$\frac{d}{dt}\ f\left(g(t), h(t)\right)$	$\frac{df}{dg}\frac{dg}{dt} + \frac{df}{dh}\frac{dh}{dt}$	Same	$f : (\mathbb{R} \times \mathbb{R}) \to \mathbb{R}$, $t \in \mathbb{R}$ $g : \mathbb{R} \to \mathbb{R}$, $h : \mathbb{R} \to \mathbb{R}$
$\frac{d}{dt}\ f\left(g_1(t), \ldots, g_N(t)\right)$	$\sum_{i=1}^{N} \frac{df}{dg_i}\frac{dg_i}{dt}$	Same	$h : (\mathbb{R} \times \cdots \times \mathbb{R}) \to \mathbb{R}$, $t \in \mathbb{R}$ $f_i : \mathbb{R} \to \mathbb{R}\ \ \forall i \in \{1, \ldots, N\}$
$\frac{d}{dt}\ f(\mathbf{g}(t))$	$\frac{\partial f}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial t}$	$\frac{\partial \mathbf{g}}{\partial t}\frac{\partial f}{\partial \mathbf{g}}$	$f : \mathbb{R}^N \to \mathbb{R}$, $\mathbf{g} : \mathbb{R} \to \mathbb{R}^N$ $t \in \mathbb{R}$
$\frac{\partial}{\partial \mathbf{v}}\ f(\mathbf{g}(\mathbf{v}))$	$\frac{\partial f}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{v}}$	$\frac{\partial \mathbf{g}}{\partial \mathbf{v}}\frac{\partial f}{\partial \mathbf{g}}$	$f : \mathbb{R}^N \to \mathbb{R}$, $\mathbf{g} : \mathbb{R}^M \to \mathbb{R}^N$ $\mathbf{v} \in \mathbb{R}^M$
$\frac{\partial}{\partial \mathbf{v}}\ f(\mathbf{g}(\mathbf{v}), \mathbf{h}(\mathbf{v}))$	$\frac{\partial f}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{v}} + \frac{\partial f}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{v}}$	$\frac{\partial \mathbf{g}}{\partial \mathbf{v}}\frac{\partial f}{\partial \mathbf{g}} + \frac{\partial \mathbf{h}}{\partial \mathbf{v}}\frac{\partial f}{\partial \mathbf{h}}$	$h : \mathbb{R}^K \times \mathbb{R}^N \to \mathbb{R}$, $\mathbf{v} \in \mathbb{R}^M$ $f : \mathbb{R}^M \to \mathbb{R}^K$, $\mathbf{g} : \mathbb{R}^M \to \mathbb{R}^N$

(c) **Neural Network:** A machine learning model that aims to approximate some function through the composition of both linear and nonlinear functions. There are two parts of neural networks: forward pass and backpropagation.

(a) **Forward Pass**: The process of calculating the predicted output of your network (and corresponding loss), given data, weights, and network structure. Given the input data $\mathbf{x}$, we can 1) transform the data using the weights associated with the first layer, W, then 2) apply the corresponding activation function to the output, and then 3) pass the result to the next layer and repeat. The forward pass does not involve taking derivatives and proceeds from the input layer to the output layer.

(b) **Backpropagation**: Given a neural network and a corresponding loss function, backpropagation gives us the gradient of the loss function with respect to the weights of the neural network. The method is called backward propagation because to efficiently apply the chain rule repeatedly, we calculate the dervatives of the final layer first, then proceed backward to the first layer.

(d) **Activation Function**: A nonlinear function, that is added to a neural network in order to help the network learn more complex patterns in the data. A few common ones are listed below.

Name	Function Definition	Derivative
logistic (sigmoid)	$g(z) = \frac{1}{1+e^{-z}}$	$g'(z) = g(z)(1 - g(z))$
tanh	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$1 - g^2(z)$
ReLU	$g(z) = \max(0, z)$	$g'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \end{cases}$

2 Chain Rule Is All You Need

Note: Just generic math notation here. So, for example, $h(x)$ definitely doesn't refer to a hypothesis function.

Let $m : \mathbb{R}^2 \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$, and $h : \mathbb{R} \to \mathbb{R}$ be functions defined as follows:

- $m(x_1, x_2) = x_1 x_2$ Note: we used m for multiply :)
- $g(x) = \sin(x)$
- $h(x) = x^2$

Suppose $x \in \mathbb{R}$ is given. We define the composite function $y = m(u, v) = m(g(x), h(x))$, where $u = g(x)$ and $v = h(x)$.

1. Apply multivariate chain rule to write $\frac{dy}{dx}$ in terms of $\frac{dy}{du}$, $\frac{dy}{dv}$, $\frac{du}{dx}$, and $\frac{dv}{dx}$.

2. Find $\frac{dy}{dx}$ using the equation from the previous part.

3. Rewrite the equation from part 1 to show that $\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$.

4. Note that the statement proven in part 3 is the product rule. Now, try to prove the quotient rule in a similar way. Let $q : \mathbb{R} \to \mathbb{R}$ and $y \in \mathbb{R}$ such that $q(x_1, x_2) = \frac{x_1}{x_2}$ and $y = q(u, v) = q(g(x), h(x))$. Prove the following equation:

$$\frac{dy}{dx} = \frac{v\frac{du}{dx} - u\frac{dv}{dx}}{v^2}$$

3 Neural Networks Are Fun

Assume we have the following data point $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$$

with corresponding binary label:

$$y = 1$$

which is part of a larger dataset X with binary labels $\mathbf{y}$.

Pat has tasked you with creating a neural network to solve this classification problem. For the loss function, he wants you to use the squared error: $\ell(y, \hat{y}) = (y - \hat{y})^2$. (Pat would like to note that using squared error for classification problems is very strange!) He also requests you to use stochastic gradient descent to train the network by minimizing the loss. Finally, he specifies that the neural network should have only one hidden layer with two neurons, a tanh activation function at the hidden layer, a sigmoid activation function at the output layer, and all bias terms should be included.

3.1 Forward Pass

1. Draw what the neural network will look like.

2. We often group weights into a single matrix for each layer of neurons. How many weight matrices are there in the neural network?

3. What will be the shape of each weight matrix W_l? (Hint: we view $\mathbf{x}$ as a column vector that includes an extra one so we can pack the bias terms into the weight matrix and compute the product as $W\mathbf{x}$.)

Assume our weights are as follows with $W_{l,i,j}$ where l represents the layer and i, j represent the row and column index, respectively:

$$W_1 = \begin{bmatrix} W_{1,1,0} & W_{1,1,1} & W_{1,1,2} \\ W_{1,2,0} & W_{1,2,1} & W_{1,2,2} \end{bmatrix} = \begin{bmatrix} -1 & -3 & 1 \\ 2 & 1 & 3 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} W_{2,1,0} & W_{2,1,1} & W_{2,1,2} \end{bmatrix} = \begin{bmatrix} 1 & 2 & -4 \end{bmatrix}$$

Note the extra column at the left in each weight matrix that will act as the bias term (hence the weird column index starting at zero.

Here are some intermediate values that we'll use going forward (haha, "forward" and "backward"):

Assume we add an extra one to the start of $\mathbf{x}$ to account for the bias term

$\mathbf{a} = W_1\mathbf{x}$

$\mathbf{z} = \tanh(\mathbf{a})$

Assume we add an extra one to the start of $\mathbf{z}$ to account for the bias term

$b = W_2\mathbf{z}$

$\hat{y} = \sigma(b)$

4. What are the values in $\mathbf{a}$, the vector being passed into our hidden neurons? Recall we need to add an extra one to the start of $\mathbf{x}$ vector to account for the bias term in this layer!

5. What is $\mathbf{z}$, the output values of our hidden neurons?

6. What is the value being passed into our output layer?

7. What is our final output $\hat{y}$?

8. What does our model classify $\mathbf{x}$ as?

9. What is the loss $\ell(y, \hat{y})$ on this example? (Note: recall that Pat admitted that using squared error with classification is strange, but we can still do it.)

Objective Function

When we are considering all of the training data, as we do in gradient descent, our objective function is:

$$J(W_1, W_2; \mathbf{y}, X) = \frac{1}{N} \sum_{i=1}^{N} \ell\left(y^{(i)}, \hat{y}^{(i)}\right) \tag{1}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \sigma\left(W_2 \tanh\left(W_1 \mathbf{x}^{(i)}\right)\right)\right)^2 \tag{2}$$

However, when we are using stochastic gradient descent, our objective function is with respect to just one training point, $\mathbf{x}, y$, at a time and thus, the output of the loss, ℓ, is our objective function for that one point:

$$J(W_1, W_2; y, \mathbf{x}) = \ell\left(y, \hat{y}\right) \tag{3}$$

$$= \left(y - \sigma\left(W_2 \tanh\left(W_1 \mathbf{x}\right)\right)\right)^2 \tag{4}$$

3.2 Backpropagation

Since our goal is to find the best weights that optimize our neural network, now we'll do backpropagation to update the weights we were given. You will need to calculate the derivative of J with respect to the weights W_1 and W_2, then you use gradient descent to update them.

In the interest of time, let's just calculate the derivatives: $\frac{\partial J}{\partial W_{2,1,1}}$ and $\frac{\partial J}{\partial W_{1,1,2}}$.

Let's start with $\frac{\partial J}{\partial W_{2,1,1}}$, where $W_{2,1,1}$ is the $1,1$ entry in W_2 weight matrix.

1. Using the multivariate chain rule, write the derivative chain expression for $\frac{\partial J}{\partial W_{2,1,1}}$.
   Hint: think about writing out the chain rule as d-out/d-in for each layer: $\frac{\partial out}{\partial in}\frac{\partial out}{\partial in}\ldots\frac{\partial out}{\partial weight}$

2. What is $\frac{\partial \ell}{\partial \hat{y}}$? Write in terms of y and $\hat{y}$

3. What is $\frac{\partial \hat{y}}{\partial b}$? Write in terms of b (but then simplify it to write in terms of just $\hat{y}$).

4. What is $\frac{\partial b}{\partial W_{2,1,1}}$?

5. Finally, what is $\frac{\partial J}{\partial W_{2,1,1}}$? Hint: remember the chain rule that we wrote for this above.

Now we will calculate the derivative with respect to $W_{1,1,2}$: $\frac{\partial J}{\partial W_{1,1,2}}$, where $W_{1,1,2}$ is the $1,2$ entry in W_1 weight matrix.

6. What is the derivative chain expression for $\frac{\partial J}{\partial W_{1,1,2}}$? Reminder: d-out/d-in, d-out/d-in, ...

We already have the first two derivatives calculated from the previous question.

7. What is $\frac{\partial b}{\partial z_1}$?

8. What is $\frac{\partial z_1}{\partial a_1}$?

9. What is $\frac{\partial a_1}{\partial W_{1,1,2}}$?

10. Finally, what is $\frac{\partial J}{\partial W_{1,1,2}}$?

11. What is our updated $W_{2,1,1}$ and $W_{1,1,2}$ if we use learning rate $\eta = 2$?