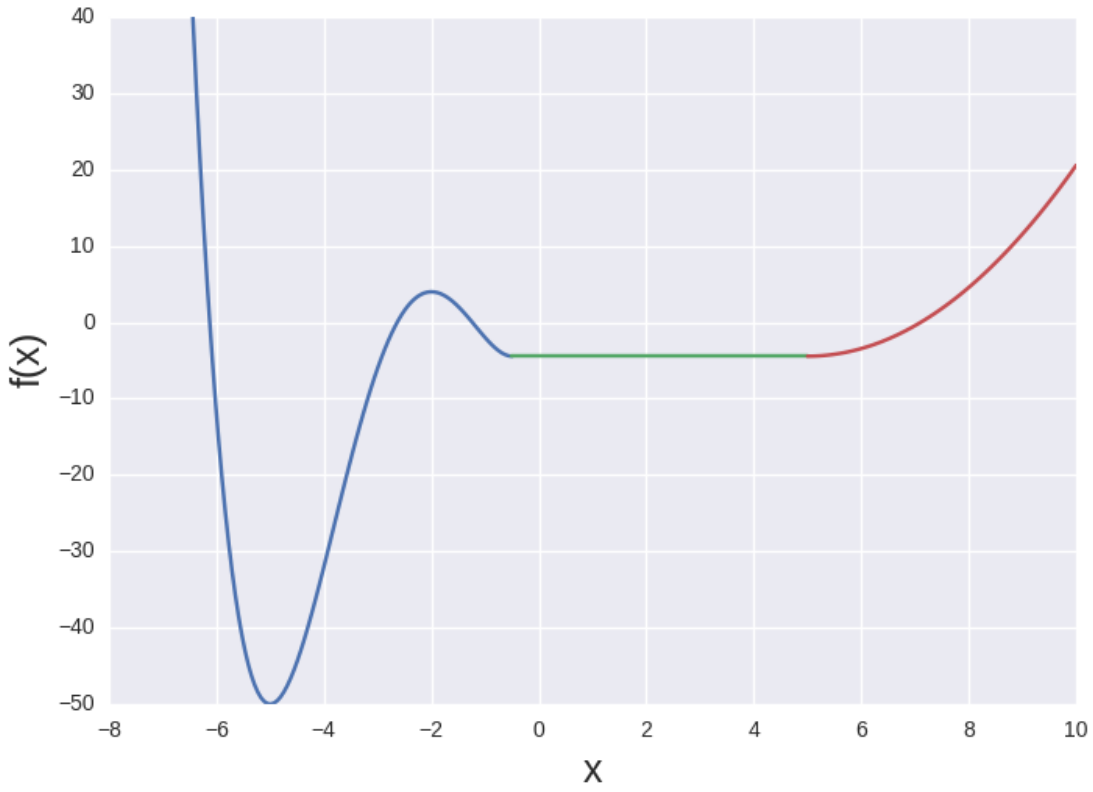
# CSE 571

# Problem Set 3

## Q1 Local Optima and Gradient Descent

After a busy year of chasing ghosts, Pacman and Paclady are planning to visit the Kakslauttanen Arctic Resort for their winter vacation. Paclady who is particularly fond of skiing, excitedly begins planning ahead. Pacman, who is apprehensive of skiing (when asked why, he rambles on about the Aspen Red Ghost Chase of 2012, but we won't get into that), reluctantly agreed to go skiing, but under one condition: Paclady must tell Pacman how steep the slopes are at several points of interest.

Paclady asks the resort for terrain details, and receives the following graph.




The resort says at any given location *x*, *f*(*x*) models the terrain height.

when $x \leq -\frac{1}{2}$, $f(x) = \frac{1}{2}x^4 + 5x^3 + \frac{27}{2}x^2 + 10x$

when $-\frac{1}{2} \leq x \leq 5$ , $f(x) = -\frac{71}{16}$

and when $x \geq 5$ , $f(x) = x^2 - 10x + \frac{329}{16}$

The local optima for $f$ lies at $x = -5$ and $x = -2$, with a plateau in the region $1/2 \leq x \leq 5$.

Show your workings for Q1.1, Q1.2 and Q1.3.

**Q1.1** Paclady decides to compute derivatives to measure how steep slopes are. Evaluate $f'(-6)$.

**Q1.2** Evaluate $f'(0)$.

**Q1.3** Evaluate $f'(8)$.

**Q1.4** Pacman and Paclady get to the resort, and have a fantastic time skiing, but get lost. Unfortunately, a blizzard kicks in right then, reducing visibility. As Pacman panics and brings up the Aspen Red Ghost Chase of 2012, Paclady remembers that their glass igloo cabin is located at the global minimum elevation point of the resort ($x = -5$). The blizzard complicates things, since they can't ski due to the reduced visibility for safety. After thinking for a minute, Pacman says, *"Aha! We can get home in that case by following gradient descent, as long as we employ a small step size -- once we hit a gradient of 0, we know we're home!"* Paclady pauses and says, *"Your algorithm almost works, but it depends on where in the resort we currently are."*

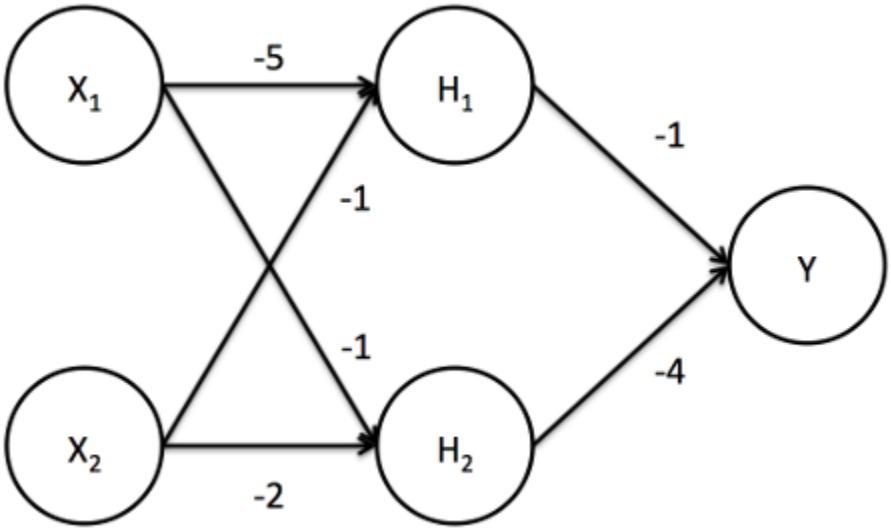
What are the regions where Pacman and Paclady can be, and still find their igloo, assuming that they employ gradient descent with a small step size and stop walking when they encounter a gradient of 0.

**Q1.5** While slowly trudging to their igloo via gradient descent, Pacman and Paclady get into an argument. Pacman complains that trudging down a hill is tiresome, and that they instead should have gotten an igloo closer to $x = 3$. Paclady says that Pacman's previous gradient descent algorithm wouldn't lead them to the igloo in this case, unless they were already at the igloo. Why is this the case?

## Q2 Neural Networks and Logic Gates

As you probably know, Pacumus Maximus Corporation (PMC) is the most well-known company that manufactures the gears that let Pacman open and close its mouth. Recently, the Vice President of Science in PMC decided to replace all of its low-level NAND, AND, and NOR gates with mini neural networks. Unfortunately, there was a Pacman uprising, where the Pacmen took over the factory, eating all of the neural network documentation. The scientists were able to salvage the following neural networks weights, but don't remember which gates these neural networks corresponded to. They've hired you to help them recover this information.

Here is the first network that you're given:




Above, the nodes $H_1, H_2$, and $Y$ have sigmoid activations and each have biases of $0.5$. The inputs are placed in $X_1$ and $X_2$. To convert the output $Y$ into a boolean value, we round $Y$. This will be the case for all problems in this section.

Concretely, we have the following, where $w_{AB}$ denotes the weight between nodes $A$ and $B$:

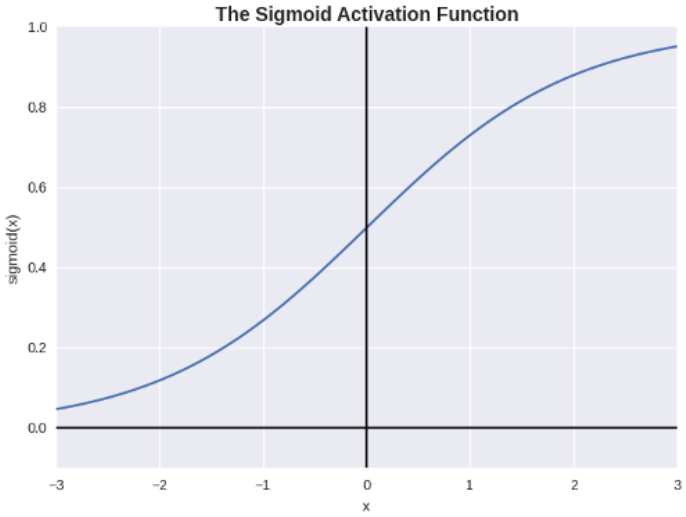
$$H_1(x) = \sigma\left(w_{H_1X_1} \cdot X_1 + w_{H_1X_2} \cdot X_2 + 0.5\right)$$

$$H_2(x) = \sigma\left(w_{H_2X_1} \cdot X_1 + w_{H_2X_2} \cdot X_2 + 0.5\right)$$

$$Y(x) = \text{round}\{\sigma\left(w_{YH_1} \cdot H_1 + w_{YH_2} \cdot H_2 + 0.5\right)\}$$

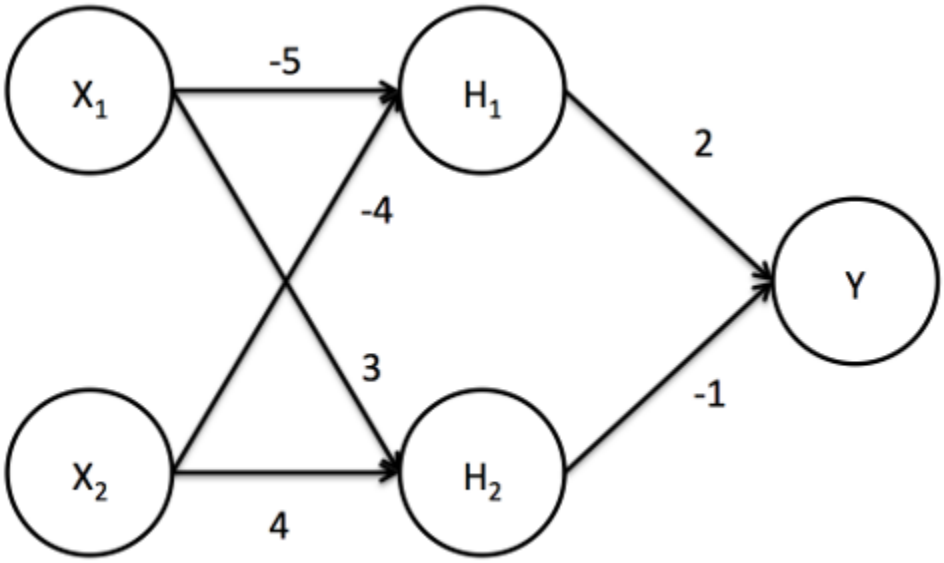
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Recall that the sigmoid function, $\sigma(x)$, looks like this:



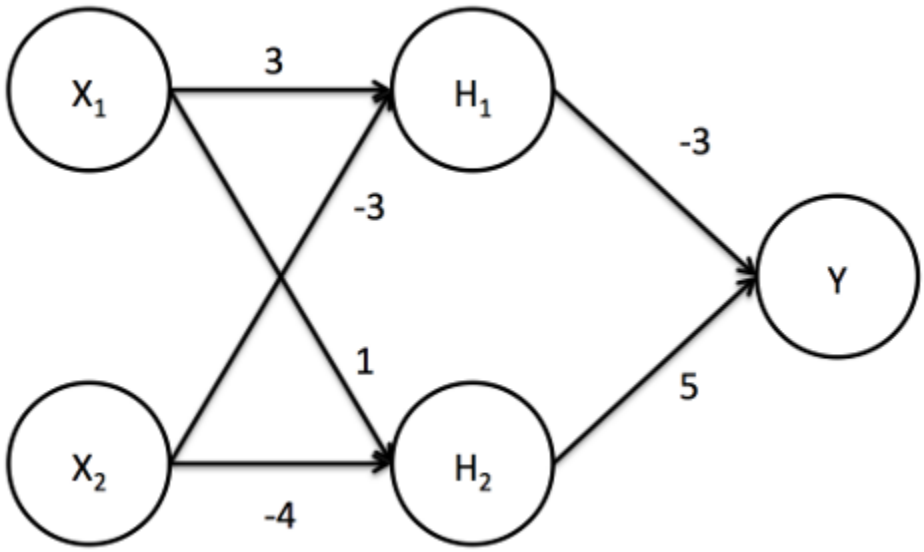

**Q2.1** What kind of logic gate does the first network correspond to? Justify.

**Q2.2** Here is the second network that you're given



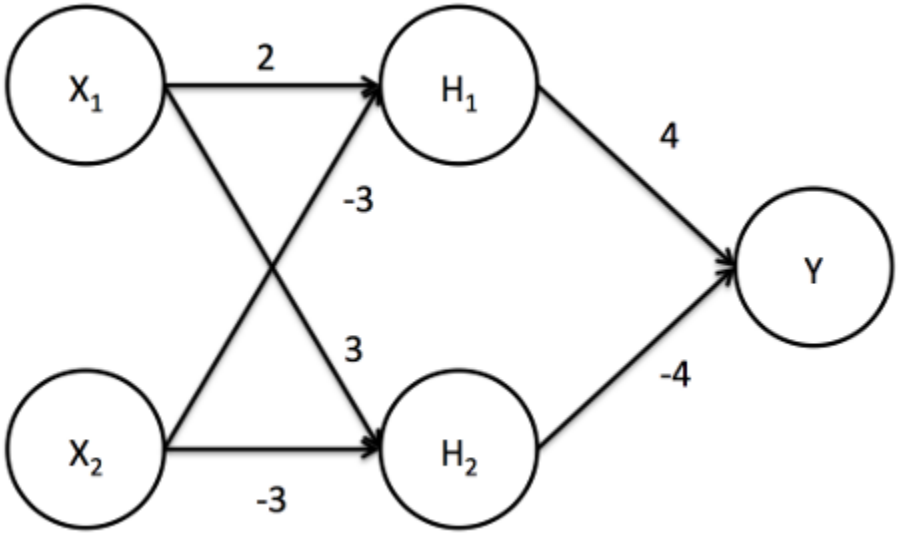
What does this network correspond to? Justify.

**Q2.3** Here is the third network you are given




What does this network correspond to? Justify.

**Q2.4** Here is the final network you are given:




What does this network correspond to? Justify.

## Q3 Multiclass Perceptron

In this problem, we will train a Multi-class perceptron on data of the form $(f(X) \in R^2, Y \in \{A, B, C\})$. In particular, we will use training data to update three weight vectors, $W_y \in R^2, y = A, B, C$.

We begin with the following set of randomly-initialized weight vectors:

| $y$ | $W_{y,1}$ | $W_{y,2}$ |
|---|---|---|
| A | -0.82 | -0.02 |
| B | -1.63 | -0.88 |
| C | 0.39 | 0.65 |

Note: For a multi-class perceptron, the label, $Y$, is chosen for a data point, $X$, as

$Y = argmax_y W_y \cdot f(X)$.

When training a perceptron, if the label chosen by the perceptron matches the label provided with the training data, the weights do not change.

However, if the label differs, say the perceptron classified the data point as $Y$, but it should have been some other label, $Y^*$, then the weights must be updated. This update is performed as:

$$W_y = W_y - f(X)$$

and

$$W_{y^*} = W_{y^*} + f(X)$$

**Q3.1** We will now incorporate the training data point $f(X) = (-1.06, 0.95); Y = C$.

What are the resulting weight-feature dot products?

a. $W_A . f(X)$
b. $W_B . f(X)$
c. $W_C . f(X)$

**Q3.2**

Now update the weight values as necessary for the training point from part 1.

a. new $W_{A,1}$
b. new $W_{A,2}$
c. new $W_{B,1}$
d. new $W_{B,2}$
e. new $W_{C,1}$
f. new $W_{C,2}$

**Q3.3** We will now incorporate the training data point $f(X) = (0.09, 1.48)$; $Y = A$. Find the resulting weight-feature dot product, and update the weight values as necessary (same as the two questions above).

**Q3.4** We took over from here and ran the perceptron algorithm till convergence. In case you're curious, this data set consisted of 50 data points, and the perceptron algorithm converged after 722 steps. Of these steps, 103 changed the weight vector.

At convergence, we have the following weight vectors:

| $y$ | $W_{y,1}$ | $W_{y,2}$ |
|---|---|---|
| A | 3.12 | 0.96 |
| B | 3.11 | -0.97 |
| C | -8.29 | -0.24 |

Use the converged perceptron to classify the new data point $f(X) = (-1.35, 0.42)$. Fill in the weight-feature dot product for each value of y.

**Q3.5** What is the predicted label?

## Q4 Perceptron Algorithm

Your perceptron-like algorithm is unable to reach zero errors on your training data. Which of the following techniques you think would help improve (or not improve) the performance on the training data? Provide reasons for each.

**Q4.1**. Running the perceptron algorithm for a longer period of time; since the perceptron algorithm is guaranteed to keep equal or reduce the number of errors at each time step, we are guaranteed to eventually reach zero errors.

**Q4.2**. Add higher-order features to our list of six features, e.g., pairwise products, and run our perceptron algorithm with this newly constructed dataset. New features increase the dimensionality of the space, and improve the chances that the data is separable.

**Q4.3**. Removing some of the features from the training data, and training the perceptron on this subset of data. Too many features increases the chance of overfitting on the training data, which would decrease performance on the training data.

**Q4.4**. Collect a larger set of data, so the perceptron algorithm does a better job of fitting to the data distribution; with a small training set, the perceptron cannot fully learn w, causing it to produce errors on the training data.

## Q5 True or False

**Note: You don't need to provide a reason for your answer. Just the answer would suffice.**

**Q5.1** Generalization is possible when using an atomic representation of states.

**Q5.2** When using features to represent states, two different states could have exactly same values for their features.

**Q5.3** Learning with feature-based state representation can be faster than learning with atomic representation of states.

**Q5.4** When using online Q-learning with linear function approximation for the Q-function, the Q-update for each observed transition can potentially modify Q-values for multiple states.

**Q5.5** Given a machine learning task, where we are given data $\{X_1, Y_1\}, \{X_2, Y_2\}, \ldots \{X_n, Y_n\}$ and we are learning a function from $X \rightarrow Y$. Then reduction in the training error (by choosing a different hypothesis for the function), always guarantees that the test error will be reduced.

**Q5.6** It is not possible to do function optimization(finding a minima for the function), without an externally supplied hyper-parameter for the learning rate(the rate at which its parameter values are updated).

**Q5.7** Using gradient descent guarantees to reach the global minima of a function.

**Q5.8** Any given data can be made linearly separable if we increase the dimensionality enough.

**Q5.9** Just like A* search, gradient descent also can consume more and more memory the longer it searches.

**Q5.10** Say we are given a perceptron with no hidden layer and an identity function, applied at the output node, to the weighted sum of the input features. If we use a squared error for the loss function, then this perceptron corresponds to linear regression.

**Q5.11** A perceptron can be applied to regression as well as classification tasks.

**Q5.12** The complexity of a machine learning algorithm is determined by the time it takes to train on a input of size $N$

**Q5.13** The last layer of a convolutional neural network which does classification is often a logistic perceptron.

**Q5.14** The learning curves of two learners used on the same training dataset can be used to compare both how fast they learn as well as the capacity of the learners (how well they can learn the true function asymptotically).

**Q5.15** If we keep training a neural network with a regularization term in its loss function even after its training error has become $0$ , there can be improvements in the test error.

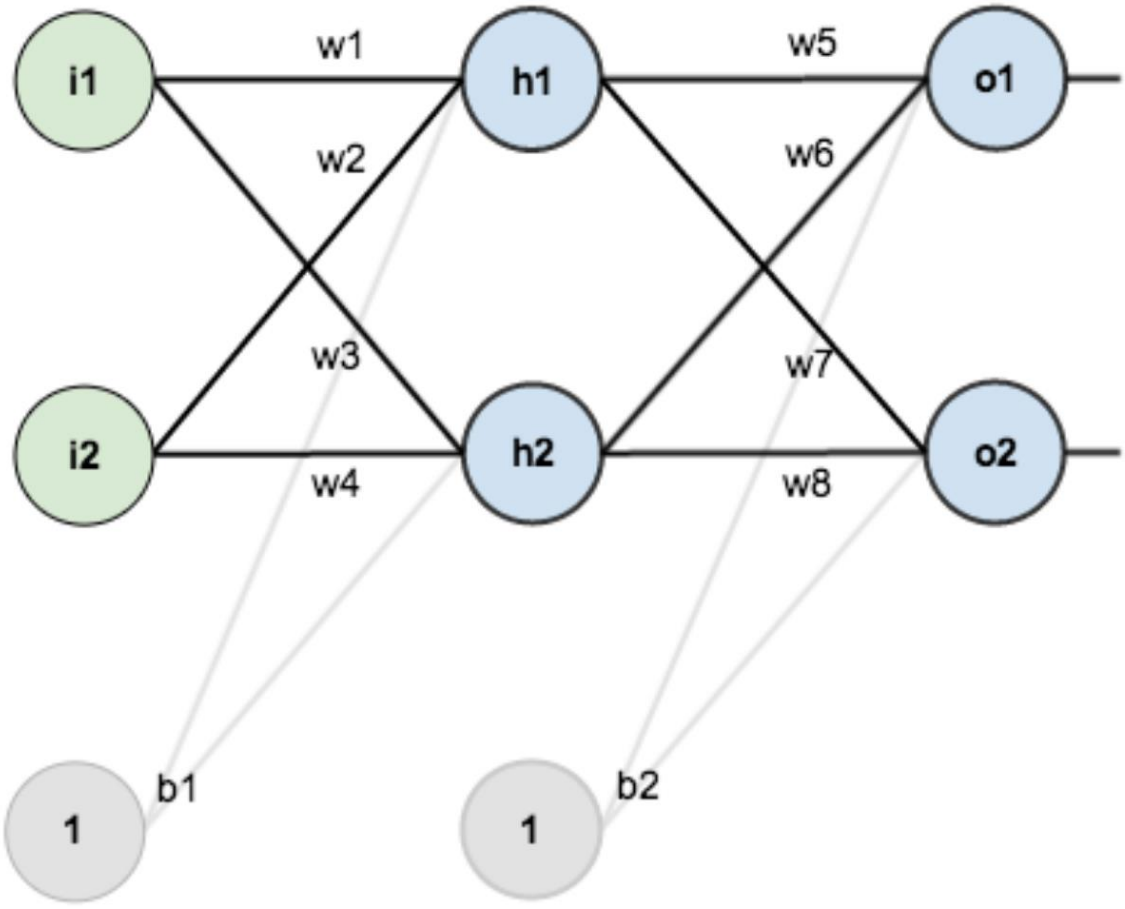
**Q5.16** The number of weights in a neural network cannot be greater than the number of training examples or their dimensionality.

**Q5.17** Scaling of the input to a network to have zero mean and unit variance is done to prevent overfitting by the network.

**Q5.18** Given a training dataset, only a unique network configuration can provide a training error of 0.

**Q5.19** Both regularization and batch-norm can be used in a neural network that uses backpropagation to minimize its loss function.

## Q6 Weight Update NN



Assume you are given the neural network above. The initial weight values, the bias and the learning rate $\alpha$ are as follows:

$w1 = 0.15$

$w2 = 0.30$

$w3 = 0.12$

$w4 = 0.28$

$w5 = 0.50$

$w6 = 0.55$

$w7 = 0.45$

$w8 = 0.40$

$b1 = 0.35$

$b2 = 0.6$

$\alpha = 0.5$

You are given a training datapoint with input ($i1 = 0.05,\ i2 = 0.1$) and corresponding output ($o1 = 0.1,\ o2 = 0.99$). For the following questions you will need to perform one iteration of backpropagation step by step and eventually update the weights $w5, w6, w7$ and $w8$.

Assume that each node has a sigmoid activation function.

A good reference on how to solve the question can be found here: *https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/* . This was also shared during lecture.

**Q6.1** Apply the input to the neural network and calculate the output at each layer(both hidden and output layer). Recall that the output at any node is the activation function applied to the sum of bias and weighted inputs.

For example, the output at node h1, $out_{h1}$ would be given by:

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}}$$

where $net_{h1} = i1 * w1 + i2 * w2 + b1$

**Q6.2** Recall that the total error is sum of squared errors at each node in the output layer:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

What is the total error for the current iteration?

**Q6.3** Now for the weights $w5$, $w6$, $w7$ and $w8$, perform a single step of gradient descent for each weight and give their updated values.

Recall that we use chain rule to find the gradient of the error w.r.t any weight.

For example, the gradient w.r.t $w\_5$ is given by

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{01}}{\partial w_5}$$

Once we know the gradient, we can update the weight as:

$w_5^{updated} = w_5 - \alpha * \frac{\partial E_{total}}{\partial w_5}$

Remember $\alpha = 0.5$