

Question 1 [1 pt]

The following figure shows a quadratic function $y=2x^2-4x+1$. Assume we are at the point (2,1), and is searching for the next movement to find the minimum value of the quadratic function using gradient descent (the learning rate is 0.1).

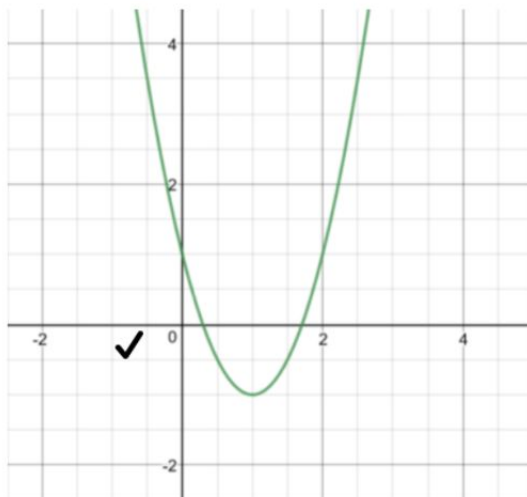


Figure 1

What is the gradient at point (2,1)? (Show your solutions) [0.5 pt]

$$\frac{\partial y}{\partial x} = \frac{\partial(2x^2 - 4x + 1)}{\partial x} = 4x - 4$$

At point (2,1):

$$\frac{\partial y}{\partial x} = 4 \cdot 2 - 4 = 4$$

Following gradient descent principle, find the next movement towards the global minimum [0.5pt]

The next movement is the opposite of the gradient:

$$\eta \cdot \left(-\frac{\partial y}{\partial x}\right) = 0.1 \cdot (-4) = -0.4$$

The next point $x(k+1)$ is: $x(k) + (-0.4) = 2 + (-0.4) = 1.6$

Question 2 [1 pt]

Please derive Backpropagation weight update rules for a neural network with one hidden layer and one output layer (please show BP rules for both hidden nodes and output nodes). (You can borrow derivations in the course lectures, but please add detailed explanations)


The general form (for hidden and output nodes) of the backpropagation weight update rules is

$$w_{ij} = w_{ij} + \Delta w_{ij}, \quad \text{where} \quad \Delta w_{ij} = -\eta \frac{\partial E(W)}{\partial w_{ij}} \quad (1)$$

The partial derivative of Δw_{ij} can be solved more easily with the chain rule:

$$\frac{\partial E(W)}{\partial w_{ij}} = \frac{\partial E(W)}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}} \quad (2)$$

The term v_j represents the output of a neuron:

$$v_j = \sum_{i=0 \dots m} w_{ij} x_{ij} \quad (3)$$


Whose partial derivative in respect to w_{ij} is easily calculated as

$$\frac{\partial \sum_{i=0 \dots m} w_{ij} x_{ij}}{\partial w_{ij}} = x_{ij} \quad (4)$$

Replacing (2) and (4) in Δw_{ij} from (1):

$$\Delta w_{ij} = -\eta \frac{\partial E(W)}{\partial w_{ij}} = -\eta \frac{\partial E(W)}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}} = -\eta \frac{\partial E(W)}{\partial v_j} \frac{\partial \sum_{i=0 \dots m} w_{ij} x_{ij}}{\partial w_{ij}} = -\eta \frac{\partial E(W)}{\partial v_j} x_{ij} \quad (5)$$

We define the partial derivative of $E(W)$ in (5) as the **local gradient** of neuron j as:

$$\delta_j = -\frac{\partial E(W)}{\partial v_j} \quad (6)$$

Replacing (6) in (5) we have the general definition of Δw_{ij}

$$\Delta w_{ij} = \eta \delta_j x_{ij} \quad (7)$$

To update the weight we need to compute the local gradient δ_j for two different cases:

- The output neurons
- The hidden neurons

Output neurons

$$\delta_j = -\frac{\partial E(W)}{\partial v_j} = -\frac{\partial E(W)}{\partial e_j} \frac{\partial e_j}{\partial o_j} \frac{\partial o_j}{\partial v_j} \quad (8)$$

Where:

$$E(W) = \frac{1}{2}(e_1^2 + e_2^2 + \dots + e_n^2) \implies \frac{\partial E(W)}{\partial e_j} = e_1 + \dots + e_n \quad (n = \text{number of output neurons})$$

$$e_j = d_j - o_j \implies \frac{\partial e_j}{\partial o_j} = -1 \quad (9)$$

$$o_j = \varphi(v_j) = \frac{1}{1 + e^{-av_j}} \implies \frac{\partial o_j}{\partial v_j} = \varphi'(v_j) = o_j(1 - o_j) \quad (9a)$$

Substituting in (8):

$$\delta_j = -e_j(-1)\varphi'(v_j) = e_j\varphi'(v_j) \quad (10)$$

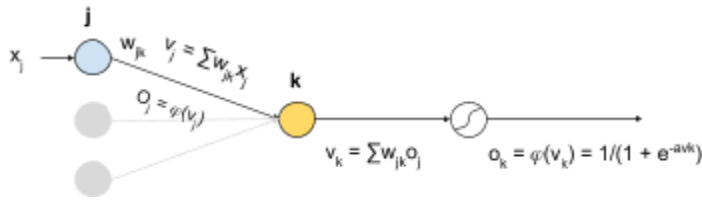
Substituting (10) in the definition of Δw_{ij} (7), using (9) to expand $e_j = (d_j - o_j)$ and (9a) to expand φ' :

$$\Delta w_{ij} = \eta \delta_j x_{ij} = \eta e_j \varphi'(v_j) x_{ij} = \eta \underbrace{(d_j - o_j)}_{e_j} \cdot \underbrace{a \cdot o_j(1 - o_j)}_{\varphi'(v_j)} \cdot x_{ij}$$

Where a is a factor to control the shape of the sigmoid function.

Hidden neurons

Visualization of some terms used in the following equations:



j = a hidden node

k = an output node

v_x = weighted output of x^{th} node (before applying activation [sigmoid] function)

o_x = output of x^{th} node (after applying activation [sigmoid] function)

$$\delta_j = -\frac{\partial E(W)}{\partial v_j} = -\sum_{k \in C} \frac{\partial E(W)}{\partial v_k} \frac{\partial v_k}{\partial v_j} \quad \text{where } C \text{ is the set of neurons of the output layer} \quad (11)$$

From (8):

$$\frac{\partial E(W)}{\partial v_k} = -\delta_k \quad (12)$$

And using the chain rule:

$$\frac{\delta v_k}{\delta v_j} = \frac{\delta v_k}{\delta o_j} \frac{\delta o_j}{\delta v_j} \quad (13)$$

Where:

$$v_k = \sum_{j=0, \dots} w_{jk} o_j \implies \frac{\delta v_k}{\delta o_j} = w_{jk} \quad (14)$$

Substituting (9a) and (14) in (13):

$$\frac{\delta v_k}{\delta v_j} = w_{jk} \varphi'(v_j) \quad (15)$$

Then we can use (12) and (15) in (13) to get:

$$\delta_j = -\sum_{k \in C} \frac{\partial E(W)}{\partial v_k} \frac{\partial v_k}{\partial v_j} = \varphi'(v_j) \sum_{k \in C} \delta_k w_{jk} = o_j \cdot (1 - o_j) \sum_{k \in C} \delta_k w_{jk} \quad (16)$$

Substituting (16) in the definition of Δw_{ij} from (7) and using (9a) to expand φ' :

$$\Delta w_{ij} = \eta \delta_j x_{ij} = \eta \cdot a \cdot x_{ij} \cdot o_j (1 - o_j) \sum_{k \text{ in next layer}} \delta_k w_{jk}$$

Where a is a factor to control the shape of the sigmoid function.

Question 3 [2 pts]

Given the following feedforward neural network with one hidden layer and one output layer,

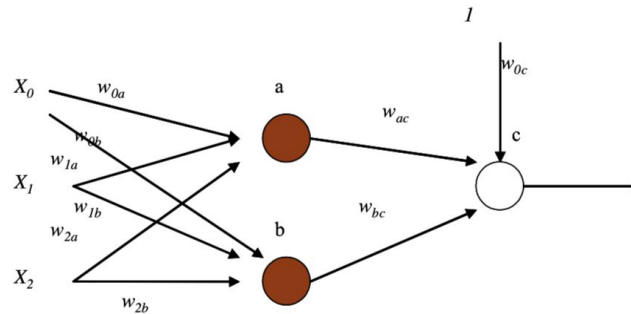


Figure 2

assuming the network initial weights are

$$\begin{bmatrix} w_{0a} \\ w_{1a} \\ w_{2a} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}; \quad \begin{bmatrix} w_{0b} \\ w_{1b} \\ w_{2b} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}; \quad \begin{bmatrix} w_{0c} \\ w_{ac} \\ w_{bc} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

All nodes use a sigmoid activation function with value $a=1.0$, and the learning rate η is set to 0.5. The expected output is 1 if an instance's class labels is "True", otherwise, the expected output is 0.

Given the following three instances I_1 , I_2 , I_3 which are labeled as "True", "False", and "True", respectively,

$$I_1 = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.5 \end{bmatrix}; \quad I_2 = \begin{bmatrix} 1.0 \\ 0.0 \\ 1.0 \end{bmatrix}; \quad I_3 = \begin{bmatrix} 1.0 \\ 0.5 \\ 0.5 \end{bmatrix}$$

Please calculate actual outputs of each instance from the network? [0.5 pt]

Instance I_1 : (1.0, 1.0, 0.5) $\sigma_x = 1/(1 + e^{-x})$ (for $a = 1.0$)

Hidden layer		Output layer
Neuron a	Neuron b	Neuron c
$v_a = w_{0a}x_0 + w_{1a}x_1 + w_{2a}x_2$ $= 1(1) + 1(1) + 1(0.5) = \mathbf{2.5}$ $o_a = x_{ac} = 1/(1 + e^{-v_a}) = \mathbf{0.924142}$	$v_b = w_{0b}x_0 + w_{1b}x_1 + w_{2b}x_2$ $= 1(1) + 1(1) + 1(0.5) = \mathbf{2.5}$ $o_b = x_{bc} = 1/(1 + e^{-v_b}) = \mathbf{0.924142}$	$v_c = w_{ac}o_a + w_{bc}o_b + w_{0c}(1)$ $= 1(0.993) + 1(0.993) + 1(1) = \mathbf{2.848}$ $o_c = 1/(1 + e^{-v_c}) = \mathbf{0.945230}$

Instance I_2 : (1.0, 0.0, 1.0) $\sigma_x = 1/(1 + e^{-x})$ (for $a = 1.0$)

Hidden layer		Output layer
Neuron a	Neuron b	Neuron c
$v_a = w_{0a}x_0 + w_{1a}x_1 + w_{2a}x_2$ $= 1(1) + 1(0) + 1(1) = \mathbf{2}$ $o_a = x_{ac} = 1/(1 + e^{-v_a}) = \mathbf{0.880797}$	$v_b = w_{0b}x_0 + w_{1b}x_1 + w_{2b}x_2$ $= 1(1) + 1(0) + 1(1) = \mathbf{2}$ $o_b = x_{bc} = 1/(1 + e^{-v_b}) = \mathbf{0.880797}$	$v_c = w_{ac}o_a + w_{bc}o_b + w_{0c}(1)$ $= 1(0.982) + 1(0.982) + 1(1) = \mathbf{2.762}$ $o_c = 1/(1 + e^{-v_c}) = \mathbf{0.940565}$

Instance I₃: (1.0, 0.5, 0.5) $o_x = 1/(1 + e^{-x})$ (for $a = 1.0$)

Hidden layer		Output layer
Neuron a	Neuron b	Neuron c
$V_a = W_{0a}X_0 + W_{1a}X_1 + W_{2a}X_2$ $= 1(1) + 1(0.5) + 1(0.5) = 2$	$V_b = W_{0b}X_0 + W_{1b}X_1 + W_{2b}X_2$ $= 1(1) + 1(0.5) + 1(0.5) = 2$	$V_c = W_{ac}O_a + W_{bc}O_b + W_{oc}(1)$ $= 1(0.982) + 1(0.982) + 1(1) = 2.762$
$O_a = x_{ac} = 1/(1 + e^{-V_a}) = \mathbf{0.880797}$	$O_b = x_{bc} = 1/(1 + e^{-V_b}) = \mathbf{0.880797}$	$O_c = 1/(1 + e^{-V_c}) = \mathbf{0.940565}$

What is the mean squared error of the network with respect to the three instances? [0.5 pt]

$$E(W) = \frac{1}{N} \sum_n (d(n) - o(n))^2$$

$$E(W) = \frac{1}{3} ((1 - 0.945230)^2 + (0 - 0.940565)^2 + (1 - 0.940565)^2) = 0.2971$$

Assuming instance I₁ is fed to the network to update the weight, please update the network weight (using backpropagation rule) by using this instance (list major steps and results). [1.0 pt]

Instance I₁: {(1.0, 1.0, 0.5), 1} $\eta = 0.5$

← Read from right (backpropagation)		←	←
Hidden layer		Output layer	
Neuron a	Neuron b	Neuron c	
$\delta_h = O_h(1 - O_h) \sum_k W_{hk} \delta_k$ (k = outputs)	$\delta_h = O_h(1 - O_h) \sum_k W_{hk} \delta_k$ (k = outputs)	$\delta_k = O_k(1 - O_k)(d_k - O_k)$	
$\delta_a = O_a(1 - O_a) W_{ac} \delta_c$ $= 0.924 \cdot (1 - 0.924) \cdot 1 \cdot 0 = \mathbf{1.99E-04}$	$\delta_b = O_b(1 - O_b) W_{bc} \delta_c$ $= 0.924 \cdot (1 - 0.924) \cdot 1 \cdot 0 = \mathbf{1.99E-04}$	$\delta_c = O_c(1 - O_c)(d_c - O_c)$ $= 0.945(1 - 0.945)(1 - 0.945) = \mathbf{2.84E-03}$	
$\Delta W_{ij} = \eta \delta_j X_{ij}$	$\Delta W_{ij} = \eta \delta_j X_{ij}$	$\Delta W_{ij} = \eta \delta_j X_{ij}$	
$\Delta W_{0a} = \eta \delta_a X_{0a} = 0.5 \cdot \delta_a \cdot 1 = \mathbf{9.94E-05}$	$\Delta W_{0b} = \eta \delta_b X_{0b} = 0.5 \cdot \delta_b \cdot 1 = \mathbf{9.94E-05}$	$\Delta W_{0c} = \eta \delta_c X_{0c} = 0.5 \cdot \delta_c \cdot 1 = \mathbf{1.42E-03}$	
$\Delta W_{1a} = \eta \delta_a X_{1a} = 0.5 \cdot \delta_a \cdot 1 = \mathbf{9.94E-05}$	$\Delta W_{1b} = \eta \delta_b X_{1b} = 0.5 \cdot \delta_b \cdot 1 = \mathbf{9.94E-05}$	$\Delta W_{ac} = \eta \delta_c X_{ac} = 0.5 \cdot \delta_c \cdot 0.993 = \mathbf{1.31E-03}$	
$\Delta W_{2a} = \eta \delta_a X_{2a} = 0.5 \cdot \delta_a \cdot 0.5 = \mathbf{4.97E-05}$	$\Delta W_{2b} = \eta \delta_b X_{2b} = 0.5 \cdot \delta_b \cdot 0.5 = \mathbf{4.97E-05}$	$\Delta W_{bc} = \eta \delta_c X_{bc} = 0.5 \cdot \delta_c \cdot 0.993 = \mathbf{1.31E-03}$	
$W_{ij}(k+1) = W_{ij}(k) + \Delta W_{ij}$	$W_{ij}(k+1) = W_{ij}(k) + \Delta W_{ij}$	$W_{ij}(k+1) = W_{ij}(k) + \Delta W_{ij}$	
$W_{0a}(k+1) = 1 + \Delta W_{0a} = \mathbf{1.0000993884}$	$W_{0b}(k+1) = 1 + \Delta W_{0b} = \mathbf{1.0000993884}$	$W_{0c}(k+1) = 1 + \Delta W_{0c} = \mathbf{1.0014177335}$	
$W_{1a}(k+1) = 1 + \Delta W_{1a} = \mathbf{1.0000993884}$	$W_{1b}(k+1) = 1 + \Delta W_{1b} = \mathbf{1.0000993884}$	$W_{ac}(k+1) = 1 + \Delta W_{ac} = \mathbf{1.0013101869}$	
$W_{2a}(k+1) = 1 + \Delta W_{2a} = \mathbf{1.0000496942}$	$W_{2b}(k+1) = 1 + \Delta W_{2b} = \mathbf{1.0000496942}$	$W_{bc}(k+1) = 1 + \Delta W_{bc} = \mathbf{1.0013101869}$	

Question 4 [2 pts]

In order to reduce the risk of neural network overfitting, one solution is to add a penalty term for the weight magnitude. For example, add a term to the squared error E that increases with the magnitude of the weight vector. This causes the gradient descent search to seek weight vectors with small magnitudes, thereby reduces the risk of overfitting. Assuming the redefined square error E is defined by

$$E(w) = \frac{1}{2} \sum_{n=1}^N \sum_{j \in \text{output nodes}} [d_j(n) - o_j(n)]^2 + \gamma \sum_{i,j} w_{i,j}^2$$

Where N denotes the total number of training instances, $d_j(n)$ denotes the desired output of the n^{th} instance from the j^{th} output node. $o_j(n)$ is the actual output of the n^{th} instance observed from the j^{th} output node. $w_{i,j}$ is the i^{th} weight value of the j^{th} output node. Assuming an output node j is using sigmoid activation function

$\phi(v_j) = \frac{1}{1 + e^{-av_j}}$, please derive the backpropagation weight updating rule for the output node j . [Hint: use gradient descent]

The weight update rule takes a step in the opposite direction of the E :

$$w_{ij} = w_{ij} + \Delta w_{ij}, \text{ where } \Delta w_{ij} = -\eta \frac{\partial E(W)}{\partial w_{ij}}$$

The first part of the given $E(w)$ equation is the usual weight update rule. To simplify the expressions below, we will call it $E_0(W)$:

$$E_0(W) = \frac{1}{2} \sum \sum ((d_j(n) - o_j(n))^2 \quad (1)$$

$E(W)$ derivative in relation to w_{ij} can be written as:

$$\frac{\partial E(W)}{\partial w_{ij}} = \frac{\partial E_0(W)}{\partial w_{ij}} + \frac{\partial \gamma \sum w_{i,j}^2}{\partial w_{ij}} \quad (2)$$

The second part of this equation can be developed as:

$$\frac{\partial \gamma \sum w_{i,j}^2}{\partial w_{ij}} = \gamma \frac{\partial \sum w_{i,j}^2}{\partial w_{ij}} = 2\gamma w_{i,j} \quad (3)$$

Using (1), (2) and (3):

$$\Delta w_{ij} = -\eta \frac{\partial E(W)}{\partial w_{ij}} = -\eta \left(\frac{\partial E_0(W)}{\partial w_{ij}} + 2\gamma w_{ij} \right)$$

The first part of Δw_{ij} (in terms of $E_0(W)$) is the usual weight update rule.

And the weight update then becomes:

$$w_{ij} = w_{ij} + \Delta w_{ij} = w_{ij} - \eta \frac{\partial E(W)}{\partial w_{ij}} = w_{ij} - \eta \left(\frac{\partial E_0(W)}{\partial w_{ij}} + 2\gamma w_{ij} \right)$$

Question 5 [3 pts]

The “nine.instances.txt” file in the Canvas is a binary class (i.e., two class) dataset containing nine instances. The nine instances (showing in Figure 3) are not linearly separable.

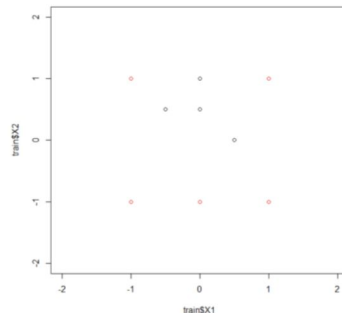
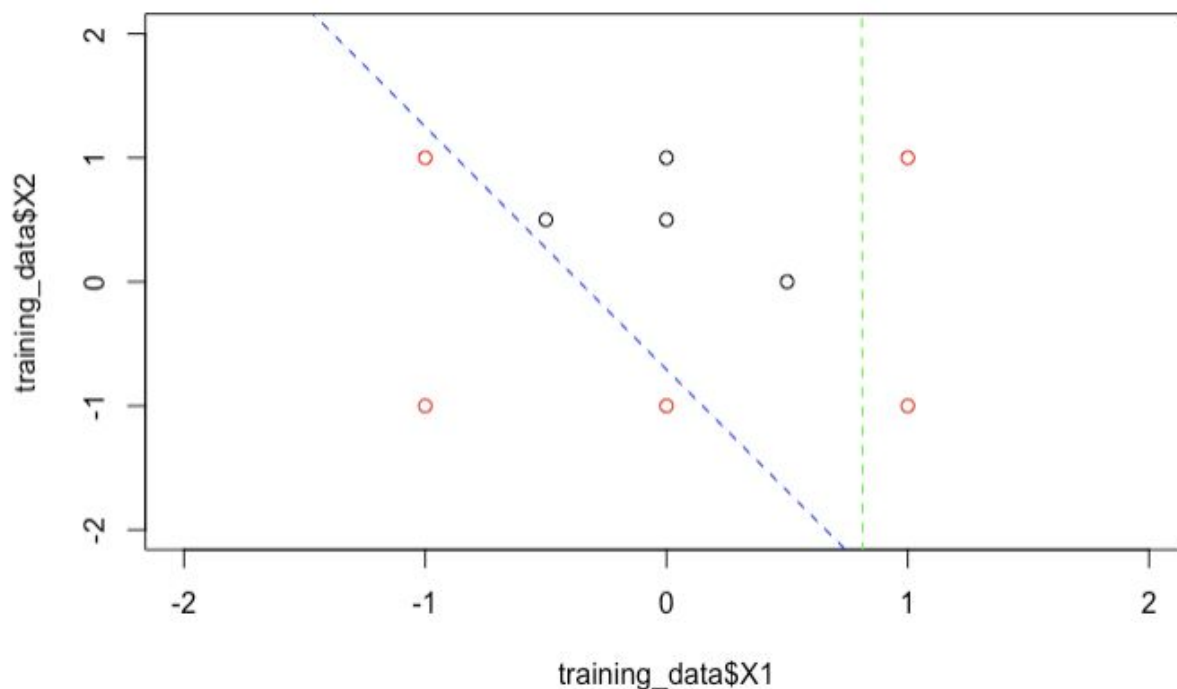


Figure 3

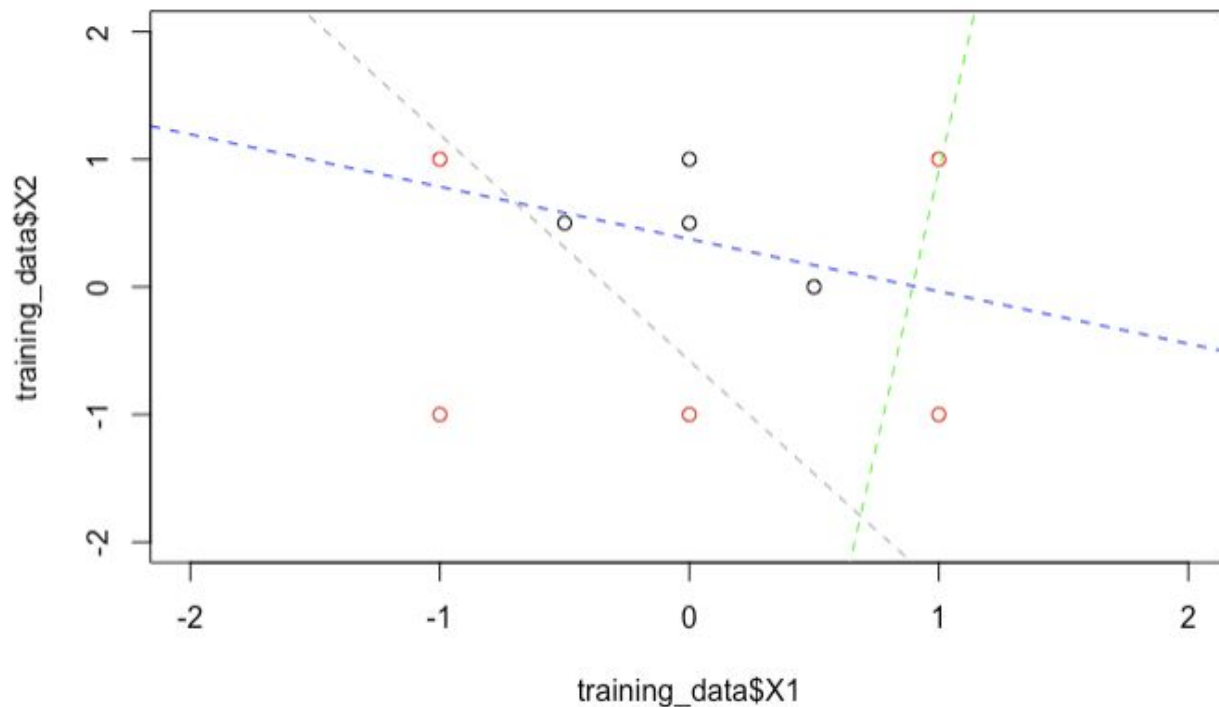
Please revise the R node in the [Neuralet Package Multi-Layer Network R Notebook \[html\]](#) to implement following tasks

Please use Neuralet to train a network with one hidden layer and two hidden nodes. Report decision surface of each hidden node (i.e., the line determined by its weight value). Please show the 9 instances in a plot (show data points in different colors according to their class label) and report the decision surfaces of all hidden nodes in the same plot [1 pt].

Code for this question is in cap6619-homework2-question5.Rmd and cap6619-homework2-question5.nb.html.



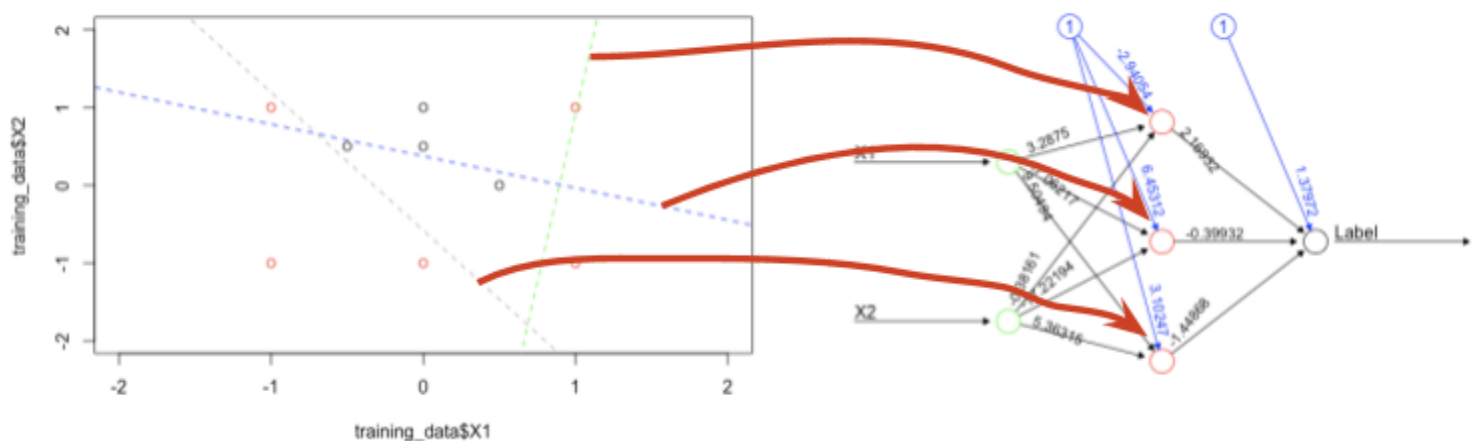
Please use Neuralet to train a network with one hidden layer and three hidden nodes. Report decision surface of each hidden node (i.e., the line determined by its weight value). Please show the 9 instances in a plot (show data points in different colors according to their class label) and report the decision surfaces of all hidden nodes in the same plot [1 pt].



Please explain the role of each hidden node in a one-hidden layer multi-layer neural network, and explain which of the above case learn a better network, and why. [1 pt]

Each hidden node creates a line (a hyperplane) in the sample space. The combination of these lines enables the classification of the data. See illustration below.

The second network, with three hidden neurons, learns a better network, as measured by error (SSE error, to be precise - the default error measure in neuralnet). This network will likely have higher accuracy when classifying unseen data because it can partition the sample space in more decision lines.



Question 6 [4 pts]

The [Multi-Layer NN for Face Recognition R Notebook \[html\]](#) in the Canvas includes R code for training multi-layer neural networks from pgm files to classify faces into different categories. The R notebook explains detailed procedures about how to load pgm files to train Neural Network classifiers. In addition, the CMU Face Recognition website (<http://www.cs.cmu.edu/~tom/faces.html>) includes faces with different directions, facial expression etc. (<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-8/faceimages/faces/>). Please design and implement following face recognition task using Neural Networks.

Please download at least 100 face images from CMU website (you can also download images from Canvas or from other sites) to build a two class classification task (50 images for each class). You must specify faces in the positive and negative class (e.g., faces turning left are positive, and faces turning right are negative). Please explain your classification task and show one example of the face in each class (using R) [0.5 pt].

Code for this question is in `cap6619-homework2-question6-part1.Rmd` and `cap6619-homework2-question6-part1.nb.html`.

The notebook is divided in four sections:

1. Helper functions, e.g. load data, create and train the neural network, etc.
2. Reproduce the class example, as a baseline and warm up exercise
3. Classification of two classes of faces: looking straight to the camera and looking up
4. Experiment with two other classes that are (for humans, at least) easier to classify: looking right and looking left. This was an attempt to understand why the classification accuracy of the first set (looking straight/up) was low.

This classification task separates two classes of images. One set is facing straight to the camera and the other is facing up. Each set contains images of several emotional states, e.g. sad, happy, angry, neutral.

Class 1 - looking straight into the camera



Class 2 - looking up



Please randomly select 40% of images as training samples to train neural networks with one hidden layer but different number of hidden nodes (3, 5, 7, 9, 11 hidden nodes, respectively). Please show the classification accuracies of the neural networks on the remaining 60% of images (which are not selected as training samples). Please report R code (e.g., using R notebook), and also use a table to summarize the classification accuracy of the neural networks with respect to different number of hidden nodes [1 pt].

Note: while experimenting with the code for this question I noticed that we were using the first repetition of the training iterations to classify the test set. As an exercise to understand better how the `neuralnet(...)` package works, I also looked into selecting the best repetition of the training iterations, to compare against the first one. In this context "best" is the repetition with the smallest training error (calculated with SSE). This is not necessarily the best model overall (we would need something more sophisticated, e.g. cross-validation, to select the best model).

Nodes	Accuracy	
	First repetition	Best repetition (smallest error)
3	0.4833	0.6333
5	0.4833	0.5333
7	0.4833	0.8000
9	0.4833	0.6167
11	0.6667	0.7500

Even with eleven neurons, the accuracy of this task is not very high. A possible explanation is that the classes are relatively close together, therefore harder to separate with the network configurations we used.

As an experiment, I tried the same network configuration on two classes that are easier to separate: looking left and looking right (not the set used in class - used the same people selected for the straight/up set above).

The table below show the classes and the classification results. It has a higher accuracy, as hypothesized, but because the code is not using cross-validation to select the best network, the results are at most preliminary at this point.

Class 1 - looking left



Class 2 - looking right



Nodes	Accuracy	
	First repetition	Best repetition (smallest error)
3	0.8833	0.9000
5	0.9000	0.8167
7	0.8667	0.8667
9	0.9333	0.8500
11	0.9000	0.9500

Please add a third category images to your dataset, and build a classification task with three classes (e.g., classifying images as facing left, right, or straight). The third category should also have at least 50 images. Explain your classification task and show one example of the face in each class [0.5 pt]

Please refer to following link for training a multi-class neural network:
<http://www.learnbymarketing.com/tutorials/neural-networks-in-r-tutorial/>

Code for this question is in cap6619-homework2-question6-part2.Rmd and cap6619-homework2-question6-part2.nb.html.

This classification task separates three classes of images. One set is facing straight to the camera, a second set is facing right and the other is facing directly into the camera. Each set contains images of several emotional states, e.g. sad, happy, angry, neutral.

Class 1 - looking left



Class 2 - looking right



Class 3 - looking straight



Please randomly select 60% of images as training samples to train neural networks with one hidden layer (9 hidden nodes) and one output layer (three output nodes). Please show the classification accuracies of the neural networks on the remaining 40% of images (which are not selected as training samples). Please report R code (e.g., using R notebook) and the classification accuracy [2 pts].

	Accuracy	
	First repetition	Best repetition (smallest error)
9	0.8167	0.7500

The following sections are not part of the homework. They are items I stumbled upon while working on the questions. These sections document the findings to resume their investigations in the future (time permitting).

Effect of label values in neuralnet's accuracy

This section is not part of the homework. I came across it when preparing the code and would like to document it here, with the hope that I can get back to this topic when more time is available.

The issue: when I first attempted to reproduce the class example I used labels 1 and 0, instead of the labels 1 and -1 used in class (for no good reason, other than I used similar values in other experiments). I was getting results significantly different than the ones in the class example. After I changed the labels to match the class example (1 and -1), the results improved.

These variations cannot be explained by differences in random values used in the different runs. The experiment was configured to get repeatable results by setting the seed for the random generator (`set.seed()`). This was verified by inverting the order of the runs, i.e. running with labels set to 1 and -1 first or set to 1 and 0 first doesn't change the results.

At this point I'm not able to explain the different results and in real life, when selecting a model with cross-validation, it may not make much of a difference. It's more of a curiosity to understand how the `neuralnet` library works internally. This notebook was created to show that effect and perhaps revisit it later, when I have time to dig into the `neuralnet` code.

The table below shows the results when using different class labels with the same set of images and same network configurations used in class.

The code for this experiment is in `cap6619-homework2-question6-label-value-effect.Rmd` and `cap6619-homework2-question6-label-value-effect.nb.html`.

		1 hidden layer, 2 neurons	2 hidden layers, 4 and 3 neurons
First repetition	Labels used in class (1, -1)	0.950	1.000
	Label (1, 0)	0.950	0.525
Best repetition	Labels used in class (1, -1)	1.000	0.975
	Label (1, 0)	0.925	1.000

Graphing iterations for question 5

This section is also not part of the homework. It was done as an attempt to gain better understanding of the `neuralnet()` function using a small data set.

The goal was to repeat the experiment in question 5 and compare the lines created in each repetition.

The code for this section is in `cap6619-homework2-question5-comparison.Rmd` and `cap6619-homework2-question5-comparison.nb.html`.

The picture below shows the lines for each iteration side by side, for easier comparison of each iteration. The notebook also shows each iteration in a separate graph, for easier visualization of each iteration.

Iteration number four is the one with the smallest error (as seen in question 5).

Some items that need some research later (time permitting) to understand better:

- Iterations 3 and 4 have similar error rates, even though iteration 4 clearly separates the samples better. Since we are using the default SSE function to calculate the error rate, I was expecting the difference in the error rate to be larger (the green line in iteration 4 is closer to the black samples, while still not far from the red samples on the right). Perhaps it's my wrong understanding of how SSE is calculated?
- The error rate in iteration 5 is significantly larger than the error for iteration 3, even though their lines aren't that different. Similar questions for SSE as in the point above.

