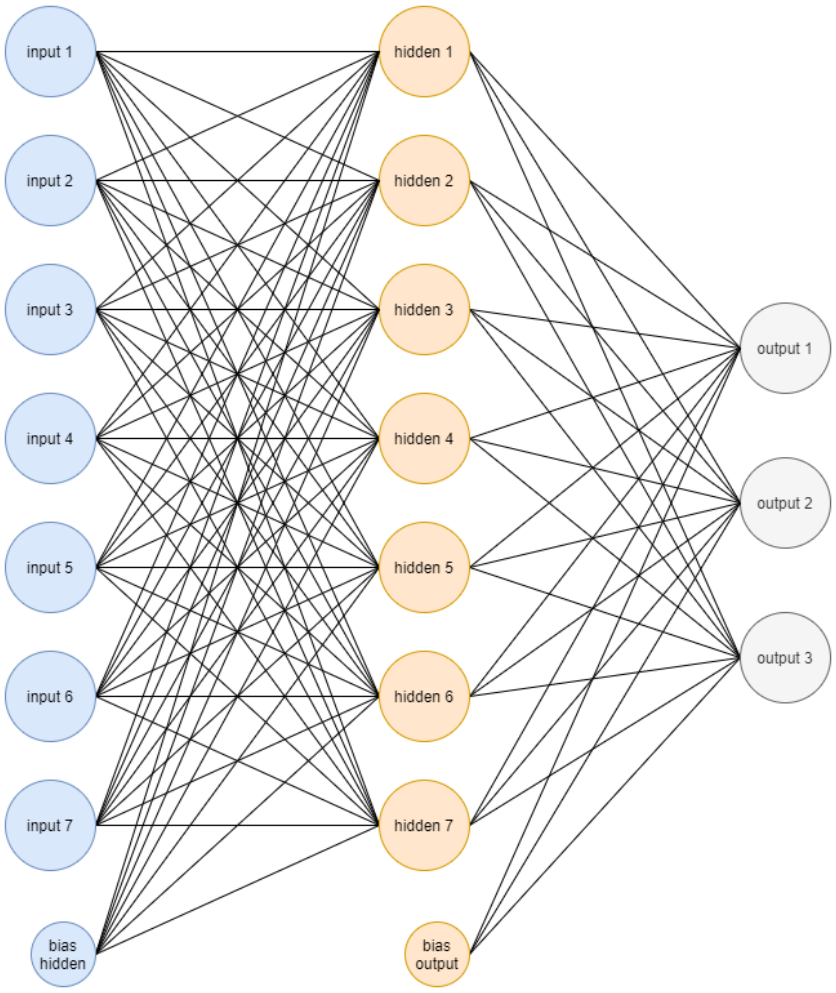
Yeshiva University

AIM 5007 Neural Networks and Deep Learning
Problem Set 3

Introduction

In the last problem set, we ran through the calculations for a simple neural network. In this problem set, we will start to develop code that accomplishes the same tasks. For this assignment, we will use the following network:



This is the architecture used in Bramer chapter 23 when analyzing the seeds data set in section 23.7. There are seven input variables, seven hidden layer nodes, and three output nodes. We will follow Bramer's lead in using a one-hot encoding for each of the three possible classes.

Through the following series of problem sets, we will build our calculations programmatically, and then put it all together at the end to create the ability to loop through our training set, develop our model, and make it possible to create predictions for new data.

In addition to the architecture shown above, we will specify the sigmoid function as our activation function at both the hidden layer and the output layer:

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

Your submission should come in the form of either a Jupyter notebook or something similar. (If you want to use something other than Python or R, please reach out to me first.)

As we work through this assignment, need to account for the following quantities:

- A vector f of input features (do not incorporate a 1 for the bias just yet)
- A matrix W_{input} of weights from the input layer to the hidden layer, such that the entry in the i^{th} row and j^{th} column is the weight from input node i to hidden node j (What size is this matrix? Don't forget the bias!)
- A vector h_{raw} of raw, incoming values for the hidden layer nodes (Bramer's $Whidj$ values)
- A vector $h_{activated}$ of transformed values for the hidden layer nodes (Bramer's $Thidj$ values) – these are the inputs to the next layer's calculations
- A matrix W_{hidden} of weights from the transformed values in the hidden layer to the raw output values in the output layer (What size is this matrix? Again, don't forget the bias!)
- A vector o_{raw} of raw, incoming values for the output layer nodes (Bramer's $Woutk$ values)
- A vector $o_{activated}$ of transformed values for the output layer nodes
- A vector t of true one-hot encodings for the observation
- A scalar E that gives the error value for the current pass
- A scalar α that is the learning rate for training the model

Forward Propagation

In this first set of problems, we will build out the forward propagation process for our network using the quantities above. You may rely on packages that allow creation of mathematical objects (e.g., NumPy) but you should code the different calculations directly rather than using a package.

Problem 0

Write a function that takes dimensions as inputs and creates an initial set of weights in a matrix of those dimensions. (For the example above, you will need to create an 8×7 matrix for W_{input} as well as another 8×3 matrix for W_{hidden}.) Each entry should be a randomly selected nonzero but small value, say in the range $(-0.5\ , 0.5)$. There is one caveat. The final row of the matrix will hold the weights for that layer's bias weights, and these need to be the same value for each entry.

After each pass (or batch, if we choose a batch update approach), we will be updating this set of weights, so the initialization will only need to be done once for each matrix. Hence, problem zero!

Problem 1

Write a function that takes a vector of length m as input and returns a vector of length $m + 1$, where the output vector is the original vector plus an extra 1 appended as the final entry. For example, your function, here called "append", should do the following:

$$append\left(\begin{pmatrix}2\\3\end{pmatrix}\right) = \begin{pmatrix}2\\3\\1\end{pmatrix}$$

Problem 2

Given the vector f of inputs, we need to use weights in W_{input} to obtain the raw, incoming values for the hidden layer nodes. In other words, we need to calculate:

$$h_{raw} = append(f)^T W_{input}$$

Write a function that takes the properly appended feature vector f and multiplies its transpose by the weight matrix W_{input} to get h_{raw}. (What are the dimensions of this resulting vector?)

Note that you should incorporate your function from problem 1 into the function for problem 2.

Problem 3

We next need to activate the neurons in our hidden layer. This means, loosely, we need:

$$h_{activated} = sigmoid(h_{raw})$$

Write a function that takes a vector of raw values and outputs a vector of transformed values, where the transformation is performed using the sigmoid function on each element of the vector individually.

What are the dimensions of the output vector with respect to the input vector?

Backward Propagation

It is harder to generalize the steps of backward propagation. To keep things simple, we will create the functions necessary only for our specific architecture above. This should suffice to give you a feel for the mechanics of backward propagation while not getting too bogged down in programming.

Problem 4

We must begin the backward propagation process by calculating our error. Write a function that takes the final, activated output vector and calculates its error with respect to the true vector of one-hot encodings for this observation. Use the squared error loss function we used before:

$$E = 0.5 * \sum_{k} (targk - Toutk)^2$$

In vector terms we can express this as:

$$E = 0.5 \cdot (o_{activated} - t)^T (o_{activated} - t)$$

Concept check: Why do I calculate $o_{activated} - t$ instead of $t - o_{activated}$? Does it matter?

Now that we can calculate the error, we need to perform our gradient descent. Our process here will be a bit inefficient for now. However, we want to be sure we understand each step. The next four functions implement the gradients.

Problem 5

Write a function that calculates the gradient of the weight from a hidden node to an output node. In Bramer's notation, this is $g(E, W_{jk})$. Note that you will need to make use of the various quantities that are defined above. Make sure you include all necessary inputs to your function.

Problem 6

Write a function that calculates the gradient of the weight from the bias term to the output nodes. In Bramer's notation, this is $g(E, biasO)$.

Problem 7

Write a function that calculates the gradient of the weight from an input node to a hidden node. In Bramer's notation, this is $g(E, w_{ij})$.

Problem 8

Write a function that calculates the gradient of the weight from the bias term to the hidden nodes. In Bramer's notation, this is $g(E, biasH)$.

Gradient Descent

Now that we can calculate the gradients, we need to use them to update our weights. For our architecture, we need two functions – one for each weight matrix.

Problem 9

Write a function that takes as inputs a learning rate (α) and updates the weights for the input-to-hidden-node step. Note that you need to update the bias weights so that it is the same weight for each node. You will probably want to incorporate your functions from problems 7-8 here.

Problem 10

Write a function that takes as inputs a learning rate (α) and updates the weights for the input-to-hidden-node step. Note that you need to update the bias weights so that it is the same weight for each node. You will probably want to incorporate your functions from problems 5-6 here.

Putting Everything Together (Almost)

Problem 11

We now should have all the pieces that we need to train our model. Write a function that performs one entire pass based on a single training instance, using your functions above. What should the output(s) of this function be?

Hint: What would we need from here to run our model on a new instance to make a prediction?

We won't develop a fully fledged training program just yet. Well, we will. But that will be problem set 5.