Assignment 4: Neural Networks and Deep Learning

UVA CS 6316 :
Machine Learning (Fall 2019)

Due: Nov. 12th, Mon midnight 11:59pm, 2019 @ Collab

a *The assignment should be submitted in the PDF format through Collob. If you prefer hand-writing QA parts of answers, please convert them (e.g., by scanning or using an app like Genuis Scan) into PDF form.*

b *For questions and clarifications, please post on Piazza.*

c *Policy on collaboration:*

Homework should be done individually: each student must hand in their own answers. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that, with the honor code, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration.

d *Policy on late homework:*

Homework is worth full credit at the midnight on the due date. Each student has three extension days to be used at his or her own discretion throughout the entire course. Your grades would be discounted by 10% per day when you use these 3 late days. You could use the 3 days in whatever combination you like. For example, all 3 days on 1 assignment (for a maximum grade of 70%) or 1 each day over 3 assignments (for a maximum grade of 90% on each). After you've used all 3 days, you cannot get credit for anything turned in late.

e *Policy on grading:*

***1**: 30 points in total. 10 points for each question answered in your report.*
***2**: 60 points in total. 10 points for each question answered in your report (See "What to Submit").*
***3**: 10 points in total. 5/5 points for answering each question respectively*
***4**: 10 extra points (See Q1.4 and Q2.7). Plus another 5 extra points if you get into top 20 of leaderboard in Q2.*
The overall grade will be divided by 10 and inserted into the grade book. Therefore, you can earn 11.5 out of 10.

1 Neural Network Playground

TensorFlow Playground is an interactive tool for learning neural networks (more specifically, multi-layer-perceptron[1] networks). A customized version of TensorFlow Playground is available at http://www.cs.virginia.edu/yanjun/HW/playground/.

First, we will get familiar with the interface of the TensorFlow Playground. Let's get to it!

You can choose from four different datasets on the left side of the page in the DATA panel: Circle, Exclusive Or, Gaussian, and Spiral. The actual data point locations are available on the right side under the OUTPUT label. Do not change the ratio of training to test data or click the REGENERATE button throughout this question. Batch size indicates how many samples are used in your mini-batch gradient descent. You may

[1] A perceptron model typically uses a heaviside step function as its activation. Since we use a sigmoid (or other differentiable nonlinearity), our representation not exactly a perceptron model, but that is what we call it

change this parameter if you want.

The neural network model is in the middle of DATA and OUTPUT. This model is a standard "feed-forward" neural network, where you can vary: (1) the input features; (2) the number of hidden layers; and (3) the number of neurons at each layer. By default, it uses only the raw inputs X_1 and X_2 as features, and no hidden layers. You will need to change theses attributes later.

Several hyper-parameters are tunable at the top of the page, such as the learning rate, the activation (non-linearity) function of each neuron, the regularization norm as well as the regularization rate.

There are three buttons at the top left for you to control the training of a neural network: Reset, Run/Pause and Step (which steps through each mini-batch of samples at a time).

1.1 Hand-crafted Feature Engineering

Now that we are familiar with the Playground, we will start to build models to classify samples.

First you are going to earn some experience in hand-crafted feature engineering with a simple perceptron model with no hidden layers, sigmoid activations, no regularization. In other words, don't change the model you're given by default when loading the page.

A perceptron (single-layer neural network) with a sigmoid activation function is equivalent to logistic regression. As a linear model, it cannot fit some datasets like the Circle, Spiral, and Exclusive Or. To extend linear models to represent nonlinear functions of x, we can apply the linear model to a transformed input $\phi(x)$.

One option is to manually engineer ϕ. This can easily be done in the Playground since you are given 7 different features to choose from in the FEATURES panel.

Task: You are required to find the best perceptron models for the four datasets, **Circle, Exclusive Or, Gaussian and Spiral** by choosing different features. Try to select as few features as possible. For the best model of each dataset, you should report the selected features, iterations and test loss in a table. If you also change other hyper-parameters, e.g., the learning rate, you should include them in your report.

For each dataset (Circle, Exclusive Or, Gaussian, and Spiral), please include a **web page screenshot** of the result in your report and explain why this configuration works.

(Note: Don't worry if you cannot find a good perceptron for the Spiral dataset. For the other three datasets, the test loss of a good model should be lower than 0.001)

1.2 Regularization

Forget the best features you have found in last question. You should now select all the possible (i.e., all 7) features to test the regularization effect here.

You are required to work on the three datasets: **Circle, Exclusive Or, and Gaussian**.

Task A: Try both L1 and L2 regularization with different (non-zero) regularization rates. In the report, you are required to compare the decision boundary and the test loss over the three models trained with similar number of iterations: no regularization, L1 regularized, L2 regularized.

For each dataset (Circle, Exclusive Or, and Gaussian), please include a **web page screenshot** of the result in your report and explain why this configuration works.

Task B: We have learned that L1 regularization is good for feature selection. Take a look at the features with significantly higher weights. Are they the same as the ones you select in last question? Write down the

results you observe in your report. (You can get the feature weights by moving the mouse pointer over the dash lines.)

1.3 Automated Feature Engineering with Neural Network

While we were able to find different parameters that were able to make good predictions, the previous two sections required a lot of hand-engineering and regularization tweaking :(We will now explore the power of a neural network's ability to *automatically* learn good features :) Let's try it out on two datasets: the **Circle** and **Exclusive Or**.

Here we should only select X_1 and X_2 as features (since we are trying to automatically learn all other features from the network). As we have previously seen, a simple perceptron is not going to learn the correct boundaries since both datasets are not linearly separable. However, a more complex neural network should be able to learn an approximation of the complex features that we have selected in the previous experiments.

You can click the + button in the middle to add some hidden layers for the model. There is a pair of + and - buttons at the top of each hidden layer for you to change the number of the hidden units. Note that each hidden unit, or neuron, is the same neuron from Lecture 17 (possibly varying the sigmoid activation function).

Task: Find a set of neural network model parameters which allow the model to find a boundary which correctly separates the testing samples. Report the test loss and iterations of the best model for each dataset. If you modify the other parameters (e.g., activation function), please report them too. Can it beat or approach the result of your hand-crafted feature engineering?

For each dataset (Circle, Exclusive Or), please include a **web page screenshot** of the result in your report and explain why the configuration works.

1.4 Spiral Challenge (0.5 Extra credit)

Congratulations on your level up!

Task: Now try to find a model that achieves a test loss lower than 0.01 on the **Spiral** dataset. You're free to use other features in the input layer besides X_1 and X_2, but a simpler model architecture is preferred. Report the **input features, network architecture, hyper parameters, iterations and test loss** in a table. For simplicity, please represent your network architecture by the hidden layers **a-b-c-...**, where a, b, c are number of hidden units of each layer respectively.

Please include a **web page screenshot** of the result in your report and explain why this configuration works.

You are now a neural network expert (on the Playground)!

Figure 1: The first 300 images from fashion_train.csv

2 Deep Learning with Keras

In this section you will get familiar with training deep learning models for image classification using Keras, a user-friendly programming interface to the popular TensorFlow machine learning framework. Your task is to familiarize yourself with the problem described in section 2.1 below and then solve the problem by using Keras to implement:

1. A multilayer perceptron (MLP)
2. A convolutional neural network (CNN)
3. 0.5 extra credits will be given to those who achieve the top 20 in the leader-board.
4. 0.5 extra credits will be give to those who finish the extra PCA part.

As will be described below, you should experiment with a variety of architectures and hyperparameters to obtain the highest accuracy you can on the test set. This is a class-wide Kaggle-style competition and the top 20 students will get an extra credit point (you can get a score of up to 11.5/10 on this assignment!)

2.1 About the Data

The problem is to classify images of articles of clothing. The dataset is a modified version of the Fashion-MNIST dataset, which was created by Zalando, a German e-commerce site. There are 10 classes in the dataset:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Each image is 28 by 28 pixels (784 features total), with a single color channel. Each pixel is represented as a value between 0 (black) and 255 (white). The CSV files on Collab contain 1 column of labels followed by 784 columns of pixel values.

The training set is 60,000 labeled examples, while the test set is 10,000 unlabeled examples (dummy labels are included in the first column).

Note: It's possible to load Fashion-MNIST using the Keras Datasets library, but don't use this feature. We're using a different version of the data. Use the data on Collab.

2.2 Setting Up

We're using the deep learning library Keras. Most commonly, it is used as a programming interface to TensorFlow and this is how we'll be using it. In turn, TensorFlow is a deep learning library that handles tensor (a generalization of a matrix) computations and glues the components of a neural network model together. This section explains two ways to run Keras and TensorFlow.

Using Google Colab (Recommended for Experimenting)
Google Colab and the provided .ipynb coding template will likely be helpful for completing this assignment (unless you have access to a CUDA-enabled GPU). Colab is a Jupyter notebook environment/free cloud service on Google Drive, giving you access to a virtual environment with GPU-support. This option allows you to substantially speed up training time, making experimenting with different models much easier. Instructions:

- Go to your Google drive > New > More > Colaboratory. Once you have a new Colab open, go to File > upload notebook and select the ipynb template file from your machine.
- Next, go to "Runtime" (top of the screen) > "Change runtime type." Make sure the runtime type option says "Python 3" and hardware accelerator is "GPU" or "TPU" (Tensor Processing Unit). Save.
- To work with our dataset, upload the "fashion_train.csv" file to your Google Drive. In the main method, make sure the "train_file" variable contains the right path. The beginning of the path should remain "/content/drive/My Drive/", but add the correct path after this part. When you run your program, you'll be given a link to an authorization code and be prompted to enter it.
- From here, you should be good to go. A caveat of using Google Colab is that you can't remain connected to the same Colab instance for more than 12 hours. After your 12 hours are up, you'll be reconnected, but if you have a job running, it'll be canceled. Furthermore, this is a free publicly available resource. Performance is usually quite good, but because you'll be sharing resources with other people, it can potentially be slower than expected. Start early to budget enough to experiment thoroughly and avoid Colab slowdowns.
- When you're done, make sure it's ready to submit by downloading your work as a .py file and then running it with grading_mode = True. This should train your best model and then create the predictions.txt file. To make sure this works, you'll need to install Keras and TensorFlow locally (see below, but don't bother with setting up a GPU or anything of that nature).

Setting Up Locally (Required for Submission)
See the Keras getting started page and follow their installation instructions. Make sure to install TensorFlow first, as they state. Some notes:

- Keras requires TensorFlow and TensorFlow doesn't support Python 3.7 yet. If you have 3.7 and use conda, you can try "conda install python=3.6". Otherwise, we'd recommend a virtual environment, docker, etc.
- On that note, using a virtual environment (such as virtualenv) is always a good idea for projects requiring multiple installations. You can also use a Keras Docker container.

- If you have a CUDA-capable Nvidia GPU, you can make TensorFlow much faster. First, check if your GPU is CUDA-capable. Then see the TensorFlow GPU support instructions. As a warning, going through this process is a bit of a hassle and it can take several hours. There are some pitfalls to avoid, such as making sure you download versions of CUDA and cuDNN that are compatible with TensorFlow.

Other dependencies: install pandas if you want to use the "get_data" method that comes with the coding template.

2.3 Multilayer Perceptron (MLP)

Architecture: First define a network architecture in the "create_mlp" method by filling out the skeleton code and adding extra layers. For each layer, select the number of units and an activation. Think about the size of the input when selecting the number of units. For the hidden layers, you can experiment with various activation functions, such as tanh, relu, elu, sigmoid, etc. For the output layer, you should select an activation function with a probabilistic interpretation.

Optimizer and Objective Function: Define an optimizer and compile the model. SGD should work fine, but feel free to switch to another optimizer or experiment ("adaptive" optimizers like Adam are very popular and often perform quite well). We recommend starting with a learning rate of around 0.01 and adjusting it as needed. Other options include adding momentum, decay, or nesterov momentum. When you call "compile" you have to specify a loss/objective function. Hint: MSE isn't a good choice in this context.

Training: Call "model.fit" with a validation_split of at least 0.1 (alternatively, 6,000 samples). Select the number of epochs (you shouldn't very many to hit 80% validation accuracy).

What to Submit: Once you're happy with your model (and the validation accuracy is above 80%), include the following in your writeup:

- Call print(model.summary()) to view a tabular summary of your network. Include this table in your writeup (either by taking a screenshot or by making a new table with the values). Include the total number of params, trainable params, and non-trainable params. Briefly explain your model and why it works.
- Use the Keras history object returned by the fit function and matplotlib to create two plots: (1) training loss and validation loss vs epoch number; (2) training accuracy and validation accuracy vs epoch number. Do the plots show evidence of any problems with your model or how many epochs you're using? Briefly explain.
- Call the "visualize_weights" function provided in the template. This method creates visualizations of the weights feeding into the units of your first hidden layer by displaying greater weight values as lighter pixels and the lower weights as darker pixels (by default, the .py template will create a folder in your working directory and save the images there; the .ipynb file will just display them). Provide a "num_to_display" argument (such as 10 or 20) and look through them. Select two visualizations to include in your report. For each one, briefly explain what that unit is attempting to do. Hint: many of the visualizations will appear incomprehensible, but can you find any that seem to resemble articles of clothing?
- Why isn't MSE a good choice for a loss function in for this problem?
- If your best-performing model was an MLP, make sure this model automatically trains, gets predictions, and writes to a "predictions.txt" file before you submit your code.

2.4 Convolutional Neural Network (CNN)

Architecture: Define the network architecture in "create_cnn" by filling out the skeleton code. The minimum requirement is that you use 1 convolutional layer, 1 maxpooling layer, 1 flattening layer, and a dense output layer. This minimum requirement is sufficient to get an accuracy of more than 85% within 10 epochs.

Hint: sigmoid and tanh probably won't work well as activation functions for your convolution layers.

Optimizer and Objective Function: As in section 2.3, define the optimizer, loss function, learning rate, and any optimizer parameters you want to use, such as momentum, decay, and nesterov momentum. Select an objective function.

Training: Train the model using a validation split of at least 0.1 (alternatively, use a validation set of at least 6,000 samples). Tune your model by trying a variety of architectures, hyperparameters, etc. You may use scikit-learn's cross-validation and grid search methods, or simply define your own search through the parameter space.

What to Submit: After you've obtained a model you're happy with (and the validation accuracy is greater than 85%), include the following in your report:

- Call print(model.summary()) to view a tabular summary of your CNN. Include this table in your writeup (either by taking a screenshot or by making a new table with the values). Briefly explain your model and why it works.
- As in section 2.3, use the Keras history object returned by the fit function and matplotlib to create two plots: (1) training loss and validation loss vs epoch number; (2) training accuracy and validation accuracy vs epoch number. Do the plots show evidence of any problems with your model or how many epochs you're using? Briefly explain.
- How many matrices are outputted by your first convolutional layer when it receives a single testing image?
- What are the dimensions of these matrices?
- What are the dimensions of one of these matrices after it passes through your first maxpooling layer?
- If your best-performing model was a CNN, make sure this model automatically trains, gets predictions, and writes to a "predictions.txt" file before you submit your code.

2.5 Tips

Accuracy: Your MLP validation accuracy should be at least 80% and your CNN validation accuracy should be at least 85%. High 80s or 90s is very possible for both models.

Training time: Training time can vary wildly with model complexity/architecture, number of epochs, batch size, and system specs. You should expect it to take at least 5 minutes to train a single model (MLP or CNN). It wouldn't be unusual for training to take more than 20 minutes. **Start early** to make sure you have enough time to train your models.

Speeding up training: A few suggestions (certainly not exhaustive):

- If you have a (CUDA-enabled) NVIDIA GPU, make sure the TensorFlow backend is using it.
- Increase the batch size, which could increase CPU/GPU utilization.
- Simplify your models by using fewer layers or fewer units per layer (you don't necessarily need an extremely deep model to get high accuracy).
- Check the training loss numbers as your model trains. If they aren't noticeably decreasing (or are even increasing), halt training and make changes.
- If your model (MLP or CNN) is on the right track, it'll probably have an accuracy of more than 50% after one epoch.
- Many techniques that improve accuracy can also speed up training by allowing you to train for fewer epochs.

Improving accuracy: Many, many options, but a few brief pointers:

- Try using Keras Dropout layers, a regularization technique.
- Try an "adaptive" optimizer like Adam, using momentum, changing the learning rate, using Nesterov momentum, etc. Check out the Keras optimizers page.
- Plot the train and validation loss against epochs to check for overfitting.
- Make sure your architecture makes sense.

2.6 Evaluation

To get full credit:

1. Include each of the items under the "what to submit" sections above.
2. Submit a file called "fashion.py" containing the code for your MLP and CNN models. We will run this file from the command line using the command:

   python3 fashion.py path/to/fashion_train.csv path/to/fashion_test.csv

   When it runs, it should train your best performing model (just one model–if your best performing model was an MLP have it run that by default. Otherwise, have it run your best performing CNN). It should generate predictions for the fashion_test.csv file and write the predictions to a file called predictions.txt. There should be one predicted label per line. The labels should be numbers in {0, 1, ..., 9}. The predictions.txt file should be created in the same directory as fashion.py.
3. When we run your code, it does not need to generate anything besides the predictions.txt file (don't output any plots or visualizations)
4. **Update: Submit your predictions.txt file in addition to the code and writeup. We will grade it in case we have problems running your code or need to save time grading.**

2.7 Extra credit: Discussion of PCA and Logistic Regression

You can get 0.5 extra credits if you can finish the following requirement.

Principal component analysis allows us to reduce the dimensions of our data while retaining as much variance as possible, thus retaining information to separate our data points. Scikit learn provides multiple ways to perform dimension reduction with PCA. For this problem, sklearn.decomposition.RandomizedPCA is recommended.

- Please select three different values of PCs you choose for PCA and present your prediction error results when using the Scikit logistic regression classifier. Please draw a figure showing how the prediction error changes when varying the number of PCs? Please also include the error discussions about the logistic regression without PCA in the discussion and in the figure.
- Since the labels of the zip.test are fake, you surely should not use results from zip.test to discuss the performance.
- - Q: can we use predefined functions like logistic regression, or cross validation from scikit-learn?
  - A: Yes.
- - Q: how many classifiers should be made, and how to deal with using the best one but including all of them?
  - A: You need to submit all the codes about the methods and model selection of the methods you tried. And you need to make sure the main function providing the interface for TA to run as the best one that you make!

  - A: please discuss the results of what you have tried in the written part of the submission.

- - Q: Must we implement all the models suggests by the skeleton code?
  - A: No. The template is just a recommendation what you can try.
  - A. The minima requirement for this coding problem requires your discussion of running the LogRegression(LoG) and PCA+LoG on the datasets.

- - Q: Is testing data being released separately and if so when? - Q: If not, will you simply run the scripts as stated in the homework instructions? (without the parameter for model selection, students just only include the one they want run).
  - A. The feature part of the testing data has been included in the data.zip. We will release its labels when we release the result key of HW3.
  - A: Yes. We will simply run your submitted code as stated in the instruction.

Congratulations ! You have implemented a state-of-the-art machine-learning toolset for an important image labeling task !

3 Sample Questions:

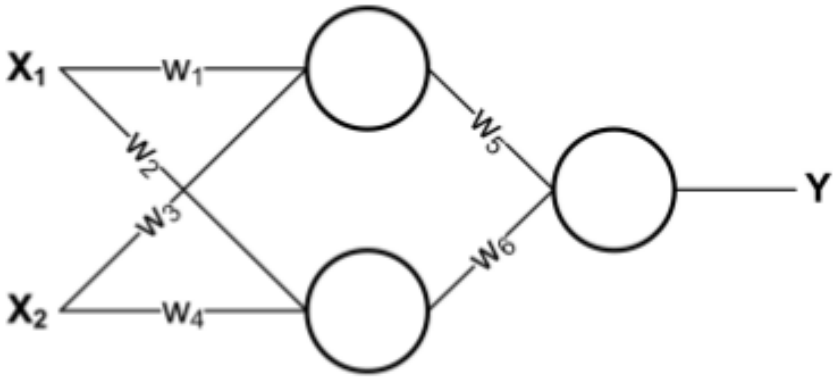
Each assignment covers a few sample exam questions to help you understand the material better.

Question 1. Neural Nets and Regression Consider a two-layer neural network to learn a function $f : X \to Y$ where $X = \langle X_1, X_2 \rangle$ consists of two attributes. The weights, $w_1, ..., w_6$, can be arbitrary. There are two possible choices for the function implemented by each unit in this network:

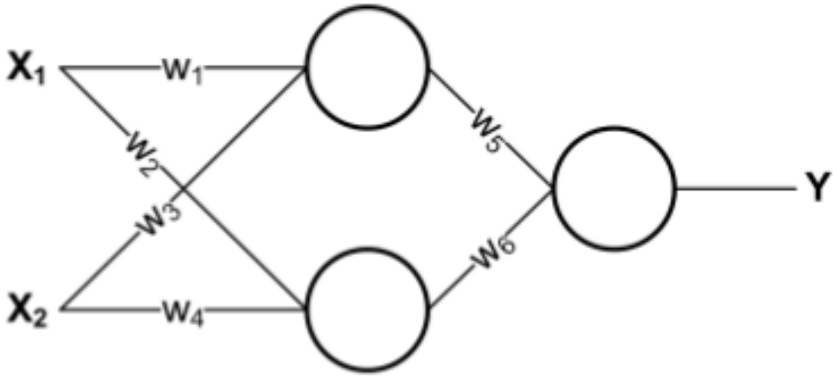
- **S:** signed sigmoid function $S(a) = sign[\sigma(a) - 0.5] = sign[\frac{1}{1+exp(-a)} - 0.5]$
- **L:** linear function $L(a) = ca$

Where in both cases $a = \sum_i w_i X_i$.

(a) Assign proper activation functions (**S** or **L**) to each unit in the following graph so this neural network simulates a linear regression: $Y = \beta_1 X_1 + \beta_2 X_2$.

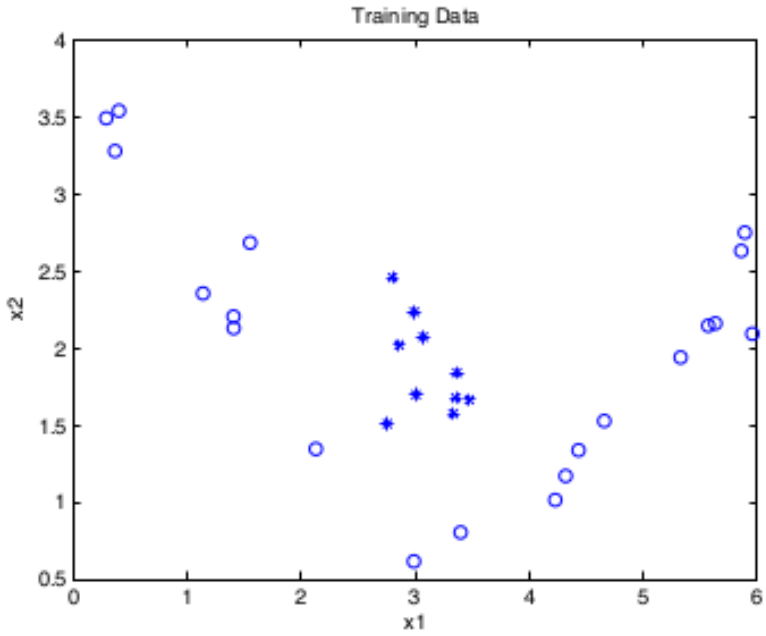


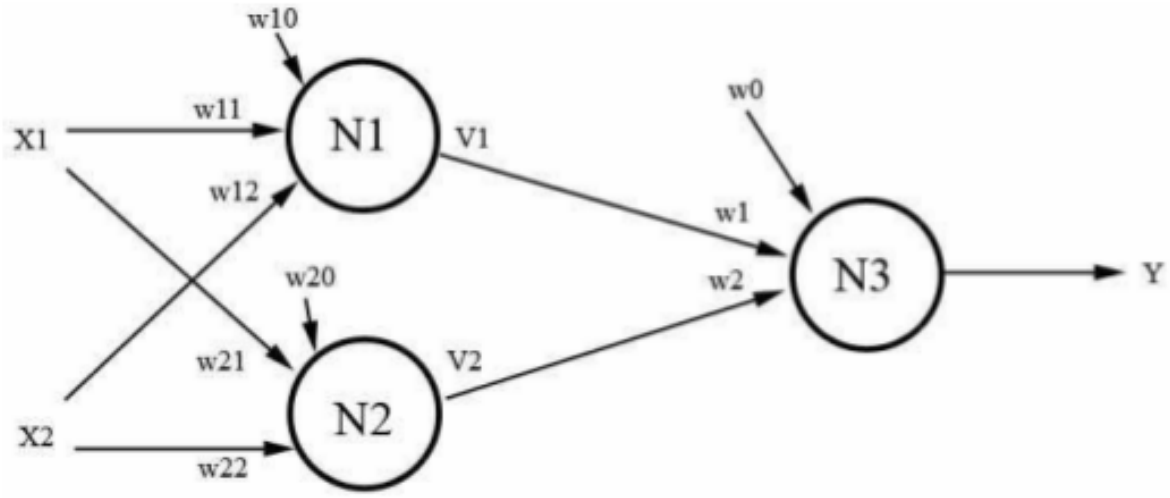
(b) Assign proper activation functions (**S** or **L**) to each unit in the following graph so this neural network simulates a binary logistic regression classifier: $Y = argmax_y P(Y = y|X)$, where $P(Y = 1|X) = \frac{exp(\beta_1 X_1 + \beta_2 X_2)}{1+exp(\beta_1 X_1 + \beta_2 X_2)}$, $P(Y = -1|X) = \frac{1}{1+exp(\beta_1 X_1 + \beta_2 X_2)}$.



(c) Following (b), derive β_1 and β_2 in terms of $w_1, ..., w_6$.

Question 2. Neural Nets Consider the following classification training data (where "*" = true or 1 and "O" = false or 0) and neural network model that uses the **sigmoid** response function ($g(t) = \frac{1}{1+e^{(-t)}}$).

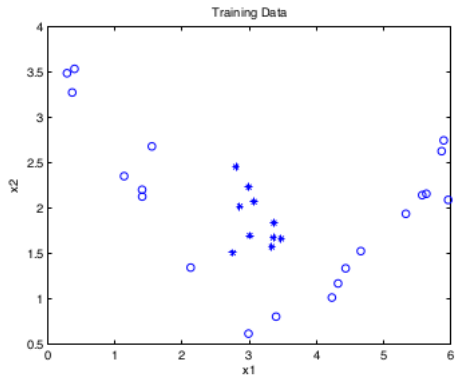




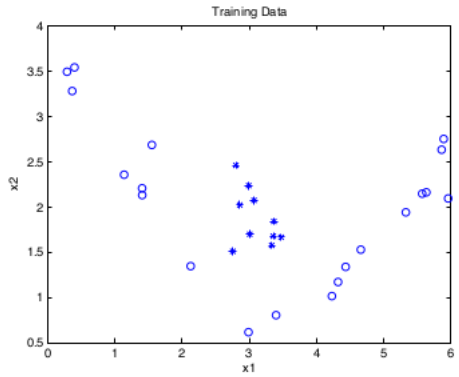
(a) We would like to set the weights (w) of the neural network so that it is capable of correctly classifying this dataset. Please plot the decision boundaries for N_1 and N_2 (e.g., for neuron N_1, the line where $w_{10} + w_{11}X_1 + w_{12}X_2 = 0$) on the first two graphs. In the third graph, which has axes V_2 and V_1, plot $V_1(x_1, x_2), V_2(x_1, x_2)$ for a few of the training points and provide a decision boundary so that the neural net will correctly classify the training data.

All graphs are on the following page!

N1 (2 points)



N2 (2 points)



N3 (4 points)