Bonus Questions

ECE 461/661: Introduction to Machine Learning
Prof. Yuejie Chi and Prof. Beidi Chen
Due: Saturday Dec 16th, 2023 at 8:59 PM PT / 11:59PM ET

Please remember to show your work for all problems and to write down the names of any students that you collaborate with. The full collaboration and grading policies are available on the course website: https://18661.github.io/. The bonus part will be weighted 5% as extra credit. You can choose to attempt either none, some, or all of the bonus questions.

Your solutions should be uploaded to Gradescope (https://www.gradescope.com/) in PDF format by the deadline. We will not accept hardcopies. Please include your code for each question in the PDF submission. If you choose to hand-write your solutions, please make sure the uploaded copies are legible. Gradescope will ask you to identify which page(s) contain your solutions to which problems, so make sure you leave enough time to finish this before the deadline. We will give you a 30-minute grace period to upload your solutions in case of technical problems. **For these Bonus Questions, no late submissions will be accepted (you cannot use a Late Day).**

1 Decision Tree for Spotify Data *[18 points]*

In this problem, you will implement a Decision Tree classifier using scikit-learn and use it to classify the Spotify data set. In this data set, an individual has generated a list of songs, each with a set of features, and whether the individual liked or disliked the song. The goal of this problem is to create a decision tree classifier to predict whether this individual would like or dislike a song based on a list of features. We encourage you to use `sklearn`'s `DecisionTreeClassifier` class for this problem.

1.1 Import Data *[3 points]*

(a) Import the data into a Pandas dataframe. Pandas is a data analysis library that is very useful for machine learning projects. Examine the data. Which features, if any, appear to not be useful for classification and should be removed? Print the final list of the feature names that you believe to be useful. Code Write-up

(b) Of the remaining features which you believe may be useful for classification, which feature(s) do you estimate will be the most important? Which feature(s) will be the least important? Briefly explain your answers. Write-up

(c) Create a Pandas dataframe with just the useful features you have selected, and a separate data series for the targets (labels) of each sample. Code

(d) Divide the full dataset into a training set and testing set, with 80% of the data used for training. Consider using the `train_test_split` function for this step. Code

1.2 Training the Model *[7 points]*

(a) Determine the best hyper-parameters for your decision tree using cross-validation with at least 5 folds. Search across at least 3 hyper-parameters for Decision Trees. It is recommended to look at 'criterion', 'max_depth', and 'class_weight', but you are welcome to explore additional or alternative hyper-parameters. The `GridSearchCV` module may be helpful here. Report which hyper-parameters you searched over and the best hyper-parameter values. Code Write-up

(b) Train your model with the best hyper-parameters found in Q1.2a. Run it on the test data to generate predictions for the test data. Your final accuracy may vary, but expect it to be around 70%. Code

1.3 Evaluating the Model *[8 points]*

(a) Generate the precision, recall, accuracy, and F1-score for your predictions from Q1.2b. These metrics are all refinements of the classification accuracy. The `sklearn.metrics` modules may help with this. What are your results? Code Write-up

(b) Generate a confusion matrix to visualize your predictions (Figure 1 has an example; note that your matrix may have very different values). The `sklearn.metrics` module may also be useful here. Code Write-up

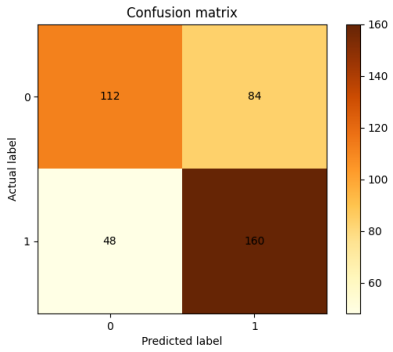

Figure 1: Confusion Matrix Example

(c) Generate a representation of your decision tree from Q1.2b using the `graphviz` and `export_graphviz` functions. Figure 2 shows an example decision tree output (note that your results may look very different from this example). Code Write-up

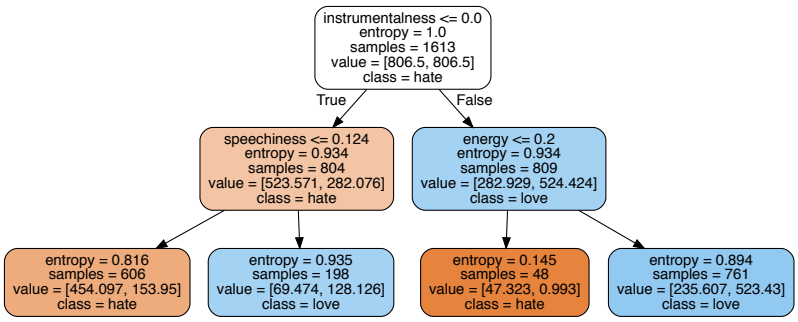

Figure 2: Decision Tree Visualization Example

(d) Determine the relative importance of each feature for the tree you trained in Q1.2b. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. How well do these results match your initial (qualitative) estimates of each feature's importance in Q1.1a? Print the importance of each feature. Code Write-up

Hint: This quantity is computed as you train your model, and thus should not require additional computations on your part.

2 Learning to classify the "classy" digits *[50 points]*

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

The goal of this task to leverage the power of Machine Learning, coupled with the comfort of PyTorch, to implement an end-to-end Multi-Layered Perceptron based system which can achieve a high classification accuracy on the Fashion-MNIST dataset.

2.1 Building a Vanilla Classifier *[30 points]*

2.1.1 Dependencies

Recommended Configuration: (i) `Python` 3.6 (ii) `PyTorch` 1.3.1 (iii) `Tensorboard` 1.14.0 (iv) `Numpy` 1.16.6 (v) `termcolor` 1.1.0

We have provided the code template in the form of a Jupyter notebook. This should enable you to run the code on Google Colab which has the recommended configuration preinstalled. You can of course run the code on your personal machine as well.

2.1.2 Starting with the data *[5 points]*

PyTorch is an effective tool to handle large datasets in parallel, using the `torch.utils.data.dataloader` class. The dataloader class, can automatically divide the dataset into batches, while parallelising the tasks amongst multiple processes. To create a dataloader, first you have to create a "Dataset" class (inheriting the `torch.utils.data.Dataset` class), and overriding the `__len__` and `__getitem__` methods. **Please**

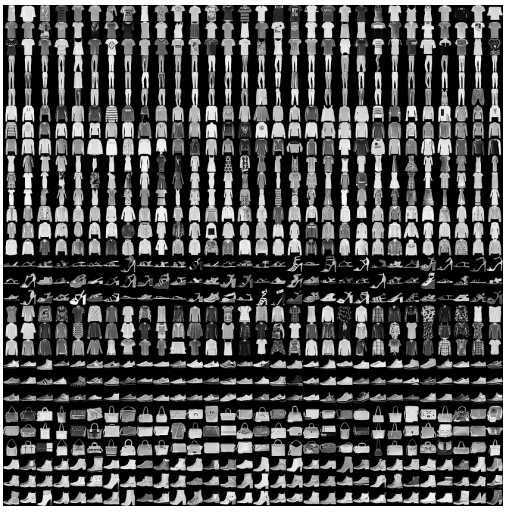

Figure 3: Fashion-MNIST

note that we have provided our own specialized dataset , and thus you can't use the Fashion MNIST loader directly from PyTorch.

(a) Explain the purpose of the `__len__` and `__getitem__` methods in the dataloader class. Write-up

(b) In the cell containing the class `FMNIST(Dataset)` complete the methods provided, ensuring that the images are scaled to $[0, 1]$. Please ensure that you use the hyperparameters from the `cfg` dictionary. Code

2.1.3 Designing the architecture *[5 points]*

(a) Define the architecture in the class `Network`. We will use an architecture containing one hidden layers ($784 \to 100 \to 10$) with a `ReLU` activation after the first layer and a `Softmax` after the final logits to get probability scores. Note that the input images are sized 28×28 and thus the input is obtained by flattening the images, making it sized 784. The architecture has been visually delineated in Figure 4. **Note that we will be using the PyTorch cross entropy loss criterion which applies `Softmax` on the output logits before computing the cross-entropy loss and therefore you do not need to explicitly apply `Softmax` in this case.**

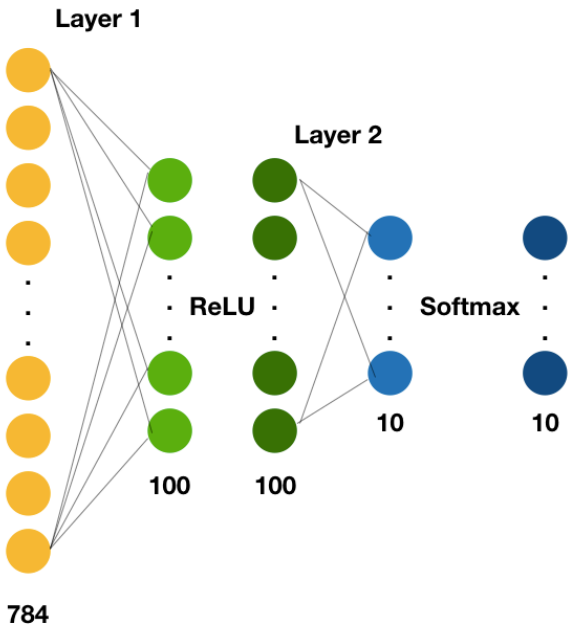

Figure 4: Architecture for Task 1

(b) Write code for saving and loading the model in the `load` and `save` methods of `Network`, which can be used to stop the training in an intermediate epoch and load it later to resume training. Code

2.1.4 A holistic view of the training *[10 points]*

TensorBoard provides the visualization and tooling needed for machine learning experimentation, by exposing an extremely easy to use interface for plotting scalars, images, histograms while training. Plot the following things after every epoch (see below for settings to train the model) using tensorboard and include in your writeup:

(a) Training Loss

(b) Test Accuracy

(c) Current learning rate

2.1.5 Training the model *[10 points]*

It's finally time to train your model on the specialized Fashion-MNIST dataset. In the code cell containing the function `train`, complete the following functionalities in the code:

(a) Create a cross entropy loss `criterion` object, which defines the objective function of our model. Code

(b) Create an `optimizer` **(Stochastic Gradient Descent)**, using the hyperparameters provided via the `cfg` dictionary. Code

(c) Complete the backpropogation by resetting the gradients, doing a backward pass, and then updating the parameters of your model. Code

(d) Learning Rate Scheduling - Use the `ReduceLROnPlateau` scheduler to anneal the learning rate based on the test set accuracy. Read about the `patience` and `factor` parameters of this scheduler, and explain their role. Use the hyper-parameters from the `cfg` dictionary Code Write-up

(e) Complete the `run` method, which takes in the network, and runs it over all the data points. Code

(f) Finally, train the model using the hyper-parameters in the `cfg` dictionary. As a milestone, you should be able to achieve around 75% accuracy on the test set, within the 20 epochs. Include the tensorboard screenshots in the writeup and report the final train loss and test accuracy Code Write-up

2.2 Improving our Vanilla Classifier *[20 points]*

Complete the following steps **(in order)**. Also, please note that the steps consist of ablation experiments for our training. We will fix the prior choices while performing the next experiment, for our comparisons. Please refer to the TL;DR section for the final set of results to be reported:

(a) Initialize the weights of your Multi-Layered Perceptron in the `init_weights` function using the following strategies:

  (a) All weights and biases zeros

  (b) Using Xavier Normal initialization for the weights and zeros for the biases

  At the end, report performance on the test set using the two initialization strategies and explain the difference in performance, if any. Code Write-up

(b) Data Augmentation has been a very useful technique for effective training of Deep Learning models. Give two examples of how data augmentation can be useful (In **any** task of your choice). Write-up

(c) Now, think of how we can do data augmentation for our task. Please note that we have modified our dataset, and it is **strongly recommended** to visually inspect both the training and test images. Now train your classifier with data augmentation and report the test accuracy after 20 epochs. Also, explain the augmentation that you have added, and the reason for the change in performance, if any. *[Note: Use the Xavier Normal initialization from the previous experiment]* Code Write-up

(d) Read about Dropout and how it affects training of deep networks. Now, add dropout to your network and include in the write-up the effect of dropout in the performance of your model. Explain any effects observed and explain if dropout helps (or not). *[Note: Use Xavier Normal initialization, and data augmentation from the prior parts]* Code Write-up

TL;DR for reporting results

To make sure you have not missed any part of the question, results and write-ups for the following questions have to be reported:

(a) 2.1.2 (a) Purpose of `__len__` and `__getitem__` methods in the dataloader class

(b) 2.1.5 (d) Role of `patience` and `factor` parameters in learning rate scheduler

(c) 2.1.5 (f) Tensorboard screenshot, final train loss and final test accuracy of Vanilla Classifier (VC)

(d) 2.2 (a) (a) Tensorboard screenshot, final train loss and final test accuracy of VC + Zero weights initialization.

(e) 2.2 (a) (b) Tensorboard screenshot, final train loss and final test accuracy of VC + Xavier normal weights initialization (XNW)

(f) 2.2 (a) Explain performance difference between zero initialization and Xavier initialization.

(g) 2.2 (b) Two examples of how data augmentation is useful

(h) 2.2 (c) Tensorboard screenshot, final train loss and final test accuracy of VC + XNW + Data Augmentation (DA). Explain any change in perfomance.

(i) 2.2 (d) Tensorboard screenshot, final train loss and final test accuracy of VC + XNW + DA + Dropout. Explain any difference in performance.

Please remember to attach your code for this question in your gradescope submission.

3 Implement Reinforcement Learning Algorithms *[32 points]*

In this problem, you will implement value iteration in Frozen-lake environment provided by OpenAI Gym. Frozen lake is a 4×4 or 8×8 grid-world environment. The agent can move up, down, left and right to reach the goal and also avoid falling into the holes. In frozen lake environment, if the agent reaches the goal, it gets +1 reward. If the agent falls into the hole, it gets -1 reward. In all other cases the agent gets -0.1 reward. Figure 5 illustrate the 4×4 frozen lake environment.

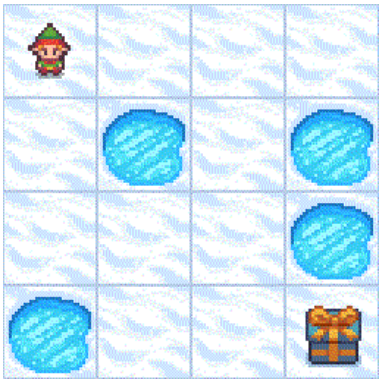

Figure 5: Frozen lake environment illustration

3.1 Environment Setup

The first step is to install OpenAI gym and register the frozen lake environment. We will use the gym 0.22 version. Simply run this command to install:

```
pip3 install gym=0.22
```

We will be using the deterministic version of the frozen lake.

3.2 Implement value iteration

Next we will implement value iteration in Frozen lake 4×4 and 8×8 environments. The algorithm of value iteration is shown as following. Report your final computed value functions and policies in both environments.

Algorithm 1: Value Iteration

Output: optimal value function $V_*(s)$ and optimal policy $\pi_*(a|s)$, $\forall s \in \mathcal{S}$

Initialize value function $V(s)$ and policy $\pi(a|s)$, $\forall s \in \mathcal{S}$
while $\Delta > \theta$ **do**
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $V_{prev}(s) \leftarrow V(s)$
    $Q(s,a) = \sum_{s',r} p(s',r|s,a)\left[r_t + \gamma V(s')\right]$
    $V(s) \leftarrow \max_a Q(s,a)$
    $\Delta \leftarrow \max(\Delta, |V_{prev}(s) - V(s)|)$
  end
end
$V_*(s) \leftarrow V(s), \forall s \in \mathcal{S}$
$\pi_*(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V_*(s')\right]$

3.3 Implement SARSA

In Value iteration, we assume the environment transition model $p(s', r|s, a)$ is known. However it is not always the case, when the model is unknown, we need another type of method called model-free RL. The Q-learning algorithm introduced in the lecture is one type of model-free learning. In this question, we will implement another type of model-free method called SARSA. The algorithm of SARSA is shown as following. Report your final policies in both environments. Compare with the policy from value iteration, are the the same? Provide some analysis or hypothesis.

Algorithm 2: SARSA

Initialize $Q(s,a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
$\alpha \in (0,1]$, small $\epsilon \in (0,1)$
for *all episodes* **do**
  initialize state s
  $a = \epsilon$-greedy$(Q(s))$
  for $t = 1, 2, \cdots, T$ **do**
    $s', r =$ take_step(s,a)
    $a' = \epsilon$-greedy$(Q(s'))$
    $Q(s,a) = Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s' = s$
  end
end