**CSCE 823: Machine Learning**

**Summer 2019**

**HW4**

Due Wed, 14 Aug at 2359

Submit via Canvas

**(**This Homework is worth 10 points toward your final grade**)**

This homework assignment focuses on classifying sequential data. In particular, your goals for this task are to train a recurrent neural net (RNN) using several different text corpuses and then have the network answer the question: for this small excerpt of text, which corpus did it come from? Some text corpuses and several utilities are provided in the files and codebase for this assignment.

A simple character-sequence-learning many-to-many character-level RNN inspired by Karpathy is provided (<http://karpathy.github.io/2015/05/21/rnn-effectiveness/> ), **however the task solved by Karpathy’s network is different than your task**. Your goal is sequence style classification – given a sample of text, which source style did this kind of writing come from. You may use this provided network as a starting point, but keep in mind that it doesn’t do classification – it does sequence generation.

In Karpathy’s many-to-one network, given a sequence, the network’s goal is to generate the sequence of next characters in the sequence of the training data. To anthropomorphize what is happening, with a big enough training sample, and enough training, the goal is to learn the “style” of the writing (however, if over-trained, the network will overfit to the training data and is likely to just start copycatting / plagiarizing the original training text) In the provided application of Karpathy’s network, the goal was to create a text generator such that – for each character in the input sequence, it produces an output which is the next character in the sequence. More specifically, given characters 1‑through‑*n* of raw text, a zero-training-error response would be that when input sequence is characters 1‑through‑(*n*‑1), the output is exactly character sequence 2‑through‑*n*. For example, if the text corpus contained training text which included “hello world!” and you seeded the generator with “hel” the model should output “lo world!”

Your submission should include *at least one final trained model*. In addition to the ipython notebook, you should include any new data you use for evaluating the network, and your saved model(s). Although your notebook will contain all the code to train a model from scratch, when you submit it, your ipython notebook should be configured *to load and run (not train) your best model(s) for prediction and model performance comparison.* Use code to perform calculations or mathematical transformations, or provide code-generated graphs and figures or other evidence that explain how you determined model performance.

All code, notebooks, images, data and saved models and written documentation should be provided in a single directory (possibly with subdirectories as needed – but any subdirectories should be referenced in code via relative file paths).

Optional: This assignment can include a written report (MS Word or PDF) separate from your code, data, and models if that makes it easier for you to document your architecture description/design decisions. If you don’t include a separate document, then all of these architecture descriptions and design decisions must be in your jupyter notebook.

**Provided Codebase vs. Student Code**

Some starter code is included for you, including utilities for loading data, saving processed datafiles, saving and loading models, and showing confusion matrices. One character-sequence-RNN and an application of it (to generate a stream of text like the training sample) is also provided. Explore the existing code before building new code. This code is provided as-is.

**A Note on Training Time**

Note that you could use just the CPU (not GPU) for this assignment, but training time may be VERY long if you don’t have one – so plan accordingly. If you have access to a mediocre machine with a GPU it will likely outperform an awesome machine which doesn’t have one. One student found it took over 10 times longer to train the network with just a CPU than when the GPU was used.

One small sample network architecture (for a different task) has been included for you to play with and adapt to the task. Training time on a GPU for this small network with a small training text (a few KB) might take a few minutes. However, using large text (MB) and deeper models will likely push training time into hours. You should do some timing tests on a few epochs to estimate how long your training will take. Don’t set up training that will take more time than you have available.

Since you will be exploring architectures, you will likely be iterating coding different architectures and training/evaluating them. Keep in mind that you can train multiple architectures in sequence and save the models under different filenames, then load them later – this might be a way to train several architectures when you are doing something else (like sleeping).

**Student Model Submission**

In this HW, you must submit at least one **trained** model which is your best-performing model (the instructor won’t have time to train your untrained models). This model can be saved using the provided utilities. Note that you will have to rename the final model before submission. This best model should be named “keras\_RNN\_LASTNAME\_FIRSTNAME\_final\_model.h5”. The instructor will run evaluation code on your trained model to confirm its performance. You may submit other models, but the instructor may not have time to evaluate them.

**Student Tasks**

This is the final homework in the class. You will be expected to apply many of the skills you have learned over this quarter and in CSCE 623 in this homework. Examine your training/validation loss & accuracy curves as your network trains. Use early stopping and model checkpointing to capture the best validation-performance model during training. Consider using tensorboard for real time monitoring.

Each step listed below should correspond to code and/or text in your file. Make it easy for the instructor to find your work by using the step identifiers in both your code and in markdown cells in the notebook. Use keras with tensorflow backend on this assignment to make it possible for your instructor to grade the HW. You will also need packages hdf5 and h5py installed in your environment to load and save files even though they don’t appear as imports.

1. [UNGRADED/DON’T SUBMIT] Existing Architecture (Karpathy) Training Check / exploration: Using the provided code, practice training and testing the Karpathy model on each of the provided text samples dataset on a small number of epochs (e.g. 5 to 10). The number of training epochs required to get to a certain loss value will probably go down as the amount of available training text goes up. Use the generate\_text function to try some text generation with the fully trained model. The default setting for model training will save versions of the model after each training epoch. By exploring the output of models trained with different numbers epochs you will notice the evolution of what your model “knows”. Models trained on fewer epochs babble. Those trained on too-many epochs will plagiarize whole chunks of the source text. Somewhere in the middle, the model will appear to get the “gist” of the training text - it writes in the ‘style’ of the author without appearing to plagiarize.
2. Student Architecture Design: Your goal in this step is to design and document your proposed model.
   1. Design the model: Your model has a different ML task than Karpathy’s – you need to determine from which text a sequence of characters most likely came from. Your input observation will be a string of text (40 characters) drawn from the training text. Your output will be a class ID (0\_bible.txt; 1\_nietzsche.text; 2\_shakespeare.txt; 3\_warpeace.txt). You can provide a class ID for each character in the input sequence, but you must provide an overall class ID (0, 1, 2, or 3) for the whole input sequence. You could use one-hot encoding here.
   2. Document your design: Examine the layers of your model in the code, and develop a diagram (e.g. using Keras functions and/or Powerpoint) of the layers like you’ve seen in class or on the internet. In text, describe the architecture and its parameterization. Discuss: How many parameters are there in your whole model? In each layer? How did you decide on overall size/capacity? What other features are present in your model (like dropout) and why did you use them? How does your model differ from the Karpathy-like model?
   3. Implement your design in code
3. Design customized code to generate training / validation / test data: Modify the existing training code to match your assigned task (inputs are character sequences and outputs are class IDs). You will need to figure out how to read and vectorize each of the four text files and use them all to train your network. You will also need to split the data into train (40%) / validation (40%) / test (20%) samples. Do this in a way such that
   * The order of any characters in text is preserved,
   * That none of the characters in any *n*-character sequence are used in more than one partition of the train/val/test split
   * The text for each partition is interleaved so that all portions of the text are represented in each of the partitions (in other words, DON’T just cut the text by location in the file into beginning=training, middle = validation, end=test).
   * The same amount of data from each class for training / validation. Note that the text files for each class are different sizes, so be careful how you implement this. You will need to use less than the full amount of text from each file.

Assuming that the sequence length *n*=40, one way to do this split might be to take each chunk the text file into a sequence of 400-character segments (ignoring the final segment of less than 400 characters in the file) and then partition these segments into train, val, and test such that the first 160 chars are used for training, the second 160 chars are used for validation, and the last 80 chars are used for test. Then, within each group, use vectorize\_text to parse the text into sub-strings, and recombine the substrings into full sets for train; val; test sets. Also, make sure that you do this segmentation with respect for the classes where the text came from. A final caution: don’t use off-the-shelf train-val-test splitters available in other packages unless you first ensure they meet all of the requirements expressed in this step… and document how they achieve this. If you build your own code for this part, fully document how you did it in your report.

1. Design Validation-based Training Code: Modify the provided training code to enable examining additional information per epoch such as training loss and accuracy and validation loss and accuracy. You may want to implement early stopping based on the validation set, however since you will be training 1 epoch at a time, you could also just take performance measurements after each training epoch. If you are making decisions using the validation data (i.e. early stopping), remember to not use the same data to also evaluate the model’s performance. Include code to capture the validation performance so that it can be plotted (in a later HW step).
2. Design Testing code: The testing code provided in the HW is designed for the original Karpathy task. Your testing code should have the signature test\_model(model, observations, targets) and it should return a sklearn confusion matrix (cm). observations is an arbitrary-sized list of 40-character sequences to classify, and targets is a list of the correct classes of those sequences. A code shell for this function has been provided – but you will need to populate it with working code.
3. New Architecture Training Check: Note that this classification task may not take as many training epochs as past HW assignments have taken – but each epoch may take longer to train than in previous assignments. You may experience that overfitting begins earlier than in previous assignments. Train your new Keras model for 2 epochs. Confirm that training works (training loss lower in the second epoch) and track how long training takes. You may need to adjust your model architecture based on the expected training time for 20 epochs (don’t build something that you don’t have time to train). Report the estimated time for training 20 epochs.
4. New Architecture Training: Train your network incrementally by epoch, monitoring both training and validation loss and accuracy. Your goal is to design a network which could achieve a training accuracy of 100% and as high validation accuracy as possible. Since you are saving each epoch’s model, you can backtrack to the model that achieves the highest validation accuracy for use in test. In this way, you sidestep the danger of early stopping too early. Plot training and validation curves as you go and use them to help you decide whether to stop training, keep training, or go back and redesign your network.
   1. If your network training accuracy has not yet plateaued that means you should train for at least another epoch – maybe several more.
   2. If your network training accuracy plateaus for a few epochs and it is far below 100% accuracy and the loss is still improving, it may mean you just need to train it for more epochs (be patient).
   3. If your training *loss* plateaus then you may need to adjust the optimizer parameters such as learning rate or decay, AND/OR design a different network.
   4. Capacity check: Once your training gets close to 100% with loss close to zero, then this means your network has sufficient capacity to learn the task and you are on your way to overfitting. You are unlikely to be improving validation metrics if you keep training. Note that dropout may disturb this type of capacity-check – dropout may prevent training performance from ever getting much better than validation performance.

Your network still may not perform well in generalization (validation accuracy and loss) even if it has good training performance. Don’t get too hung up on trying to get great validation accuracy – if you are above 50% accuracy for a 4 class classification problem, you are doing decently better than chance. Provide plots of training and validation loss and accuracy and describe what happened in your decisionmaking process (especially if you decided to re-design your network). Consult chapter 11 in the deep learning book for additional guidance.

1. Final Architecture Performance Evaluation: Train your model on all training and validation data. Test your final model on 4-class text data. Capture the predicted AND actual output classes for each input, so you can spot-check them later Report standard classification metrics including loss and accuracy as well as a confusion matrix. Perform an residuals analysis: investigate some of your network’s mistakes, given what you know about the styles of text samples provided. Were they sections of text which were easily misclassified by a human? Give a conjecture about why your network didn’t get them right.

**Rules of Engagement for this Homework Assignment:**

**Using external sources:**

The use of pre-existing solutions to answer assignments is not allowed. This includes the use of other students’ answers, solution manuals, and any other source of information which does not reflect your own work.

You may get inspiration from the internet (see above) but your work should be your own. Cite all sources you used.

You may use the internet or get help from peers when determining basic things like “how do I add points to a plot in python?”

You may use any pseudocode or concepts learned in class to solve the problem.

The code you write must be original work unless otherwise specified by the instructor.

**Programming Conventions**

In the code chunks, good software engineering principles apply: self-documenting code (meaningful function & variable names), additional comments and whitespace should be the standard for all code you turn in. If your code is not understandable it may not receive full credit.

IMPORTANT: In your python notebook you should explain what you are doing in text as well as in the comments to a code cells. A rule of thumb is to have line-level comments in the code chunks and save the larger high level comments & observations for the text in your homework writeup. Include step numbers as comments in your code to make it easy to locate them. Make mention of the names of your function names in your writeup so it is clear how you are using them.

**Code & Model File Structure and Naming Conventions**

Ensure that your document, ipython notebook, python code, model and any other files are contained entirely in **one directory** (possibly with subdirectories). Your main homework code file(s) should be python code with the name: “LASTNAME\_FIRSTNAME\_HW4.ipynb”. Your final RNN model should be named 'LASTNAME\_FIRSTNAME\_final\_model.h5'. Name other code, model or data files appropriately, to include your name.

**Pre-submission Checklist:**

**Code/Data/Model:** Ensure all of your code, data, and model is located in a single directory structure (put your document in the root of the structure). Use relative pathnames from the current directory when loading data in scripts as your pathname will not be the same as the instructor’s pathname.

Make sure that your code is set up to load & evaluate the model using test data (do not leave it configured in training mode).  
Perform a final preflight code execution check by restarting the kernel, then ensure that your code still runs (due to time-constraints you should not re-train your model though…).

**What to Submit:** Submit your **zip** file of the directory containing the ipython file and your model files to Canvas.

Note that if you discover an error and change a problem solution and re-submit, keep in mind that your instructor will only review your *latest* submission on Canvas – make sure it is complete.