This assignment is due before 11:59PM on Tuesday, November 2, 2021.

Neural Language Models : Assignment 4

In this assginment, we'll be moving on from traditional n-gram based language models to more advanced forms of language modeling using *neural networks*. Specifically, we'll be setting up a character-level *recurrent neural network*, known as a char-rnn for short.

Andrej Karpathy, previously a researcher at OpenAI, has written an excellent blog post about using RNNs for language models, which you should read before beginning this assignment. The title of his blog post is The Unreasonable Effectiveness of Recurrent Neural Networks (http://karpathy.github.io/2015/05/21/rnn-effectiveness/).

Karpathy shows how char-rnns can be used to generate texts for several fun domains:

- Shakespeare plays
- Essays about economics
- LaTeX documents
- Linux source code
- Baby names

In this assignment you will follow a PyTorch tutorial to implement your own char-rnn, and then test it on a dataset of your choice. You will also train on our provided training set, and submit to the leaderboard.

Here are the materials that you should download for this assignment:

- training data for text classification task (cities_train.zip).
- dev data for text classification task (cities_val.zip).
- test file for leaderboard (cities_test.txt)
- skeleton files (skeleton.zip)
- notebook (hw6_skeleton.ipynb)

Note

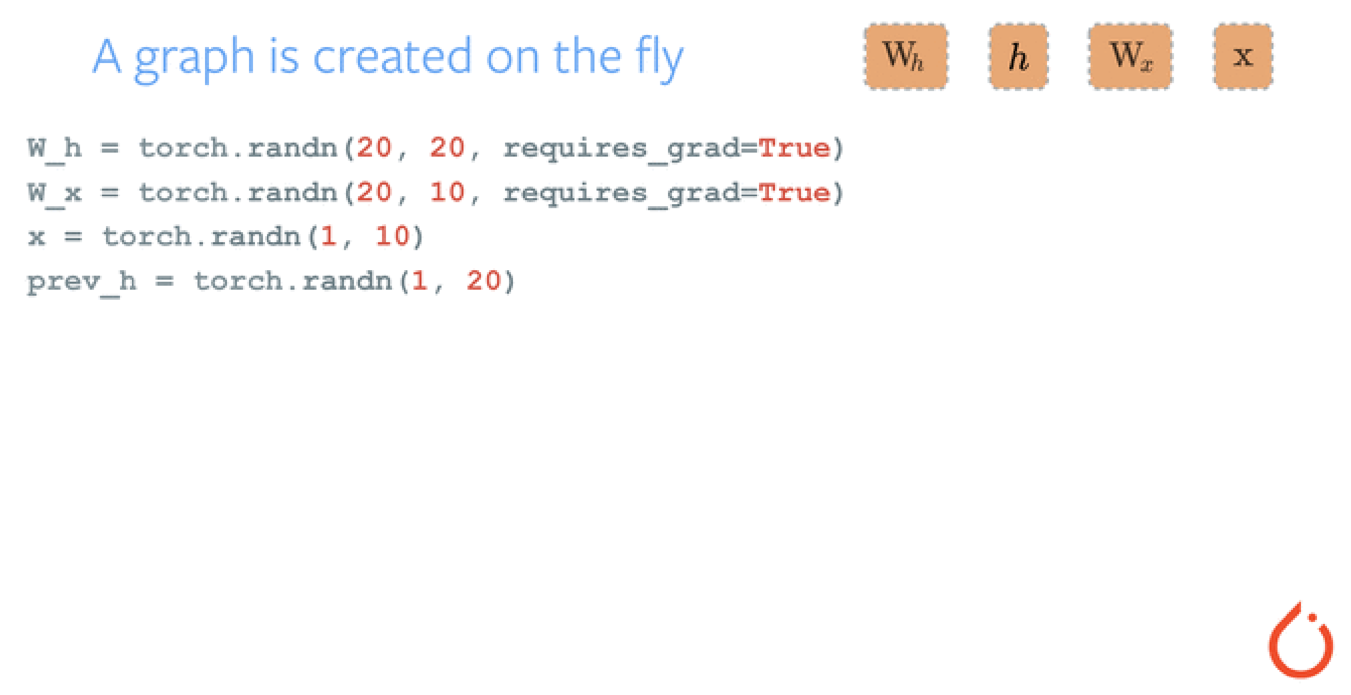
Please look at the FAQ section before you start working.

Part 1: Preamble and Setup

PyTorch is one of the most popular deep learning frameworks in both industry and academia, and learning its use will be invaluable should you choose a career in deep learning. You will be using PyTorch for this assignment, we ask you to build off a couple PyTorch tutorials.

About PyTorch

PyTorch abstracts the back-propogation process from us, allowing us to define neural network structures and use a generic `.backward()` function to compute the gradients that are later used in gradient descent (PyTorch also implements such optimization algorithms for us).



```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

PyTorch does all of this for us by maintaining a computational graph, which allows differentiation to happen automatically! Don't worry if you don't remember your chain rules from MATH 114. Another nice thing about PyTorch is that it makes strong use of both object-oriented and functional programming paradigms, which makes reading and writing PyTorch code very accessible to previous programmers.

Google Colab Set-Up

1. Download the skeleton notebook (hw6_skeleton.ipynb).
2. Upload the notebook on Colab (https://colab.research.google.com/).
3. Set hardware accelerator to `GPU` under `Change Runtime Type` in the `Runtime` menu.
4. Run the first cell to set up the environment.

Part 2: Character-Level Recurrent Neural Networks

Follow the tutorial code

Read through the tutorial here (http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html) that builds a char-rnn that is used to classify baby names by their country of origin. It is recommended that you can reproduce the tutorial's results on the provided name dataset before moving on, since the neural network architectures remain largely the same. Make sure you try your best to understand the dimensions of each layer (e.g. which ones can stay the same, and which are hyperparameters for us to tweak).

Switch to city names dataset

Download the city names dataset:

- training sets (cities_train.zip)
- validation set (cities_val.zip)
- test file for leaderboard (cities_test.txt)

Modify the tutorial code to instead read from the city names dataset. The tutorial code problematically used the same text file for both training and evaluation. We learned in class about how this is not a great idea. For the city names dataset we provide you separate train and validation sets, as well as a test file for the leaderboard.

All training should be done on the train set and all evaluation (including confusion matrices and accuracy reports) on the validation set. You will need to change the data processing code to get this working. In addition, to handle unicode, you might need to replace calls to `open` with calls to `codecs.open(filename, "r",encoding='utf-8', errors='ignore')`.

Warning: you'll want to lower the learning rating to 0.002 or less or you might get NaNs when training.

Experimentation and Analysis

Complete the following analysis on the city names dataset, and include your finding in the report.

1. Write code to output accuracy on the validation set. Include your best accuracy in the report. (For a benchmark, the TAs were able to get accuracy above 50% without any hyperparameter optimization) Discuss where your model is making mistakes. Use a confusion matrix plot to support your answer.
2. Periodically compute the loss on the validation set, and create a plot with the training and validation loss as training progresses. Is your model overfitting? Include the plot in your report.
3. Experiment with the learning rate. You can try a few different learning rates and observe how this affects the loss. Another common practice is to drop the learning rate when the loss has plateaued. Use plots to explain your experiments and their effects on the loss.
4. Experiment with the size of the hidden layer or the model architecture How does this affect validation accuracy?

Leaderboard

Write code to make predictions on the provided test set. The test set has one unlabeled city name per line. Your code should output a file `labels.txt` with one two-letter country code per line. Extra credit will be given to the top 5 leaderboard submissions. Here are some ideas for improving your leaderboard performance:

- Try dropout if your model is overfitting
- Experiment with different loss functions, optimizers
- Test out label smoothing
- Compare the different types of RNNs - RNN, LSTM, GRU units.
- Use a different initalization for the weights, for example, small random values instead of 0s

In your report, describe your final model and training parameters.

Bookkeeping

Another tip for experimenting with neural network hyperparameters is to maintain notes (e.g. a spreadsheet or text-file) with different parameters and their resulting accuray. As you can imagine, there is a combinatorial explosion of the possible hyperparameter space so navigating it efficiently is best done by remembering your past experiments. Feel free to include this in your report as well.

Part 3: Text Generation

In this section, you will be following more PyTorch tutorial code in order to reproduce Karpathy's text generation results. Read through the tutorial here (http://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html) to get an understanding for the neural architecture, and then download this iPython notebook (https://github.com/spro/practical-pytorch/tree/master/char-rnn-generation) to base your own code on (it's a bit easier to follow than the former).

You will notice that the code is quite similar to that of the classification problem. The biggest difference is in the loss function. For classification, we run the entire sequence through the RNN and then impose a loss only on the final class prediction. For the text generation task, we impose a loss at each step of the RNN on the predicted character. The classes in this second task are the possible characters to predict.

Experimenting with your own dataset

Be creative! Pick some dataset that interests you. Here are some ideas:

- ABC music format (https://raw.githubusercontent.com/rdeese/tunearch-corpus/master/all-abcs.txt)
- Donald Trump speeches (https://github.com/ryanmcdermott/trump-speeches)
- Webster dictionary (http://www.gutenberg.org/cache/epub/29765/pg29765.txt)
- Jane Austen novels (http://www.gutenberg.org/files/31100/31100.txt)

Potential extra credit will be given for creative and impressive methods of curating a text dataset for generation!

In your report:

Include a sample of the text generated by your model, and give a qualitative discussion of the results. Where does it do well? Where does it seem to fail? Recall a good way to evaluate language models is perplexity.

For a validation sentence $W = w_1 w_2 \dots w_N$:

$$Perplexity(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$
$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$
$$= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \dots w_{i-1})}}$$

Report perplexity on a couple validation texts that are similar and different to the training data.

Deliverables

Here are the deliverables that you will need to submit:

- writeup.pdf
- Saved models[model_classify, model_generate] - Your trained PyTorch models. Please put it in the same path as your code.
- code[main_classify.py, main_generate.py, models.py, README.txt] - It should be written in Python 3. README should include instructions to generate sentences.
- `labels.txt` predictions for leaderboard.

Please include the following in your report

For Part 2:

- A description of what you tried, and the final model you settled on
- A table with validation results for the models you tried
- A plot of validation accuracy as your best model is training

- An error analysis, including a confusion matrix and a dicussion of what is happening.

For Part 3:

- A description of the text you are training on
- A description of the model you are running
- A table of example generations
- Perplexity results on two types of held out data: (a) similar to what you trained on and (b) dissimilar to what you trained on

FAQs

Before submitting, please make sure to review the checklist on Piazza!

Submission Guidelines

Autograders can be finicky, and Gradescope doesn't let us change visibility of the error log. If you follow all instructions, you shouldn't have an issue. Most of the issues come down to one of the following:

1. Not turning in the saved model (model_classify).
2. Not splitting the notebook into `models.py` , `main_classify.py` , `main_generate.py`
3. Not having default values for the arguments of `__init__` in `CharRNNClassify` (IMPORTANT!)
4. Importing libraries not included in the skeleton
5. Saving model with CUDA. We don't have CUDA on the autograder. Save the final model with CPU.
6. Your default parameters model and the saved model have different dimensions. Please verify that you can load the model into `CharRNNClassify()` without any arguments.
7. Do not change the names of the default skeleton code. Feel free to add helper functions but do not alter the original code's structure.

To debug, print out the shapes of your tensors! This usually is a good sanity check that your architecture is correct and that you are performing the right computations.

How do I save a PyTorch model?

Use the command below. Please ensure that your model can be used for **inference**.

```
torch.save(model.state_dict(), PATH)
```

How do I load a PyTorch model?

Use the command below.

```
model = CharRNNClassify()
model.load_state_dict(torch.load(PATH))
model.eval() #To predict
```

I'm unfamiliar with PyTorch. How do I get started?

If you are new to the paradigm of computational graphs and functional programming, please have a look at this tutorial (https://hackernoon.com/linear-regression-in-x-minutes-using-pytorch-8eec49f6a0e2) before getting started.

How do I convert a Jupyter notebook to a python script?

```
jupyter nbconvert --to script notebook.ipynb
```

How do I beat the threshold for the test cases?

The TA's model, which passed all the testcases, had the following configuration:

- Optimizer: `torch.optim.SGD(model.parameters(), lr=0.005)`
- Criterion: `nn.NLLLoss()`
- Iterations: 250k
- RNN layers: 1 LSTM cell followed by softmax

How do I speed up training?

Send the model and the input, output tensors to the GPU using `.to(device)`. Refer the PyTorch docs (https://pytorch.org/docs/stable/notes/cuda.html) for further information.

Why are some of the cities mislabeled in the training and development datasets?

Noisy data is common when data is harvested automatically like the cities dataset (https://www.maxmind.com/en/geoip-demo). The onus is on the data scientist to ensure that their data is clean. However, for this assignment, you are not required to clean the dataset.

Recommended readings

Neural Nets and Neural Language Models. (https://web.stanford.edu/~jurafsky/slp3/8.pdf) Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd edition draft) .
The Unreasonable Effectiveness of Recurrent Neural Networks. (http://karpathy.github.io/2015/05/21/rnn-effectiveness/) Andrej Karpathy. Blog post. 2015.
A Neural Probabilistic Language Model (longer JMLR version). (http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf) Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin. Journal of Machine Learning Research 2003.

This assignment is based on The Unreasonable Effectiveness of Recurrent Neural Networks (http://karpathy.github.io/2015/05/21/rnn-effectiveness/) by Andrej Karpathy. The city names dataset is derived from Maxmind (https://dev.maxmind.com/geoip/geoip2/geolite2/)'s dataset. Daphne Ippolito, John Hewitt, and Chris Callison-Burch adapted their work into a homework assignment for UPenn's CIS 530 class in Spring 2018. Updated in Spring 2020 by Arun Kirubarajan.