COGS 185: Assignment #3

Due on Monday, May 9, 2015

April 25, 2016

In this assignment, we are going to explore text modeling with Hidden Markov Model (HMM) and (character-level) Recurrent Neural Networks (RNN). For hidden Markov model, we will be using the Matlab Statistics Toolbox. For the character recurrent neural networks, we'll be using Torch7, a lua-based deep learning library. This time we will break the assignment into many (very) simple tasks. Don't be scared by the number of tasks, most of them are as easy as just one line of code!

Task #1: MATLAB HMM toolbox

Read and understand the documentation for each of the following functions:

- hmmgenerate – Generates a sequence for a hidden Markov model
- hmmestimate – Estimates the parameters for a Markov model
- hmmtrain – Calculates the maximum likelihood estimate of hidden Markov model
- hmmdecode – Calculates the posterior state probabilities of a sequence
- hmmviterbi – Calculates the most likely state path for a hidden Markov model sequence

Task #2: Sampling from an HMM model

Use hmmgenerate to generate 50 sequences with length 10 from a 2 state HMM model specified below:

```
transitionProb = [0.85, 0.15; 0.10, 0.90];
emissionProb = [1/3, 1/4, 5/12; 1/4, 1/4, 1/2;];
```

store the resulted sequence and states variable to cells.

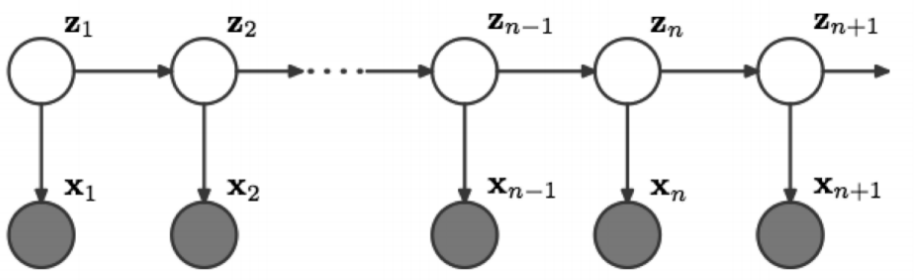

Figure 1: The graphical model for an HMM. Each emission X_t is conditioned on the corresponding hidden state Z_t. The latent variables form a Markov chain.

```
hints: loop from 1 to 50, run [seqs_train{i}, states_train{i}]  =
hmmgenerate(length, transitionProb, emissionProb);
```

Task #3: Estimate HMM parameters

Now, for this task, pick some sequences and their corresponding states, generated from Task2, and use **hmmestimate** to estimate the transition probability matrix and emission probability matrix.
Compare the parameters you learned with the real transitionProb, emissionProb, (provided in Task2). Do they agree with each other? Do you always get a 2x2 transition matrix? What will happen if only one state is visited in the sequence/states data you pick?

```
hints: [estimatedTrans, estimatedEmission] = hmmestimate(seqs_train{2}, states_train{2})
```

Task #4: Most probable path of states

hmmviterbi is useful to calculate the most probable path of states for a sequence, given the sequence observations, and the HMM model (transitionProb and emissionProb). Now pick some sequences you generated in Task 2 and apply viterbi algorithm on it.

```
hint: viterbiStates = hmmviterbi(seqs_train{2}, transitionProb, emissionProb);
compare viterbiStates with states_train{2}
```

Task #5: Forward-backward algorithm for posterior probabilities

Use **hmmdecode** to determine the probability of being in either state along the sequence (use the same sequence as in Task 4). Print the Posterior state probabilities, the Forward probabilities and the Backward probabilities returned by **hmmdecode**.
Is the most probable state equal to the state of the Viterbi path at all times? (This question is NOT required to be answered/explained in your report.)

```
hint:
[PSTATESS,logpseq,FORWARD,BACKWARD,S] = hmmdecode(seqs_train{2}, transitions, emissions)
```

You can choose to do Task 6-10, or Task 11-14, or all of them for extra points!

Task #6: HMM inference for real: Stock Market Prediction

Your TA is considering investing his next TA paycheck on S&P500 stock market index. He needs an HMM expert (yes, you) to help him predict the market in the future.
TERM OF SERVICE: THIS HMM MODEL AND TRAINING DATA PROVIDED HERE ARE TOO SIMPLE AND NAIVE. TRADING ACTIVITY MAY RESULT IN LOSSES THAT CAN EXCEED 100% OF YOUR INITIAL CAPITAL. YOU ARE SOLELY RESPONSIBLE FOR ANY LOSSES IN YOUR ACCOUNT IF YOU ARE TRADING WITH THE MODEL PROVIDEDIN THIS HOMEWORK ASSIGNMENT.

You will use a simple HMM with Q = 3 hidden states and a sequence length of T = 100 trading days. You can roughly think of the three hidden states as "bullish", "bearish", and "stable".
The emission model is a multinomial distribution with O = 5 observed symbols.

- 1: large drop
- 2: small drop
- 3: no change
- 4: small rise
- 5: large rise

To make your task easier, we will omit the training part of the HMM. The parameters of this HMM have already been estimated for you using the EM algorithm on a much longer portion of data. You will use this fully parametrized model to carry out inference over a sequence of 100 trading days, and then you will perform prediction of the output values (observations) over the next 28 days to help your TA.
All of the data used in this problem was downloaded from Yahoo! finance. (problem design credit: Professor Eric Poe Xing)
We provide a "**sp500.mat**" file where you will find the following parameters:

- transition: the transition probability matrix, where transition(i, j) = $P(Z_{t+1} = j | Z_t = i)$
- prior: the prior distribution over Z_1, where prior(i) = $P(Z_1 = i)$
- emission: the emission probability matrix, where emission(i, j) = $P(X_t = j | Z_t = i)$
- price_change: the observations labeled from 1 to 5

Task #7: Change Initial State Distribution

Read this section `http://www.mathworks.com/help/stats/hidden-markov-models-hmm.html#f10328`
By default, Statistics and Machine Learning Toolbox hidden Markov model functions begin in state 1. In other words, the distribution of initial states has all of its probability mass concentrated at state 1. Here we have the prior probability matrix, thus we want to assign a different prior distribution of probabilities.

```
hint:
TRANS_HAT = [0 prior'; zeros(size(transition,1),1) transition];
EMIS_HAT = [zeros(1,size(emission,2)); emission];
```

Task #8: Infer posterior probability distributions

Use hmmdecode on the observations for time points t = 1, ..., 100, given the transition and emission probability matrix. (Recall what you have done in Task 5) Report the inferred distributions over the hidden states at $t = 1, ..., 100$ by plotting (using **plot** in MATLAB) the posterior probabilities PSTATES ($P(Z_t = i|X_1, ..., X_{100})$) for $i = 1, 2, 3$ over $t = 1, ..., 100$. Make sure you label the 3 time series PSTATES(i, :) (one for each hidden state) in your plot.

Task #9: Infer most probable hidden state paths

Use **hmmviterbi** on the observations for time points $t = 1, ..., 100$, given the transition and emission probability matrix. Report the most likely hidden states over $t = 1, ..., 100$ by plotting (using plot in matlab) these values as a time series.

Task #10: Using an HMM model to predict observations

Here comes the exciting part where you are going to make predictions for **observations** (large drop, small drop, no change etc.) over the next 28 days. (Yes, your TA really needs your help.)
You should compare your predictions with the ground truth observations at time points t = 101, ..., 128. Whats the percentage of these values that your model predicts correctly? Report the average and variance over 100 runs.
As always, you will be using MATLAB functions and here we provide you a basic framework, **prediction_framework.m** to achieve the goal.

Task #11: Generating text using an HMM

In this exercise you will train a hidden Markov model from text data. The observations are the words in the text, and the hidden states are the parts-of-speech tags for each word. Here is a short example:

```
''Tom jumped over the dog.''
Observations: {'tom' 'jumped' 'over' 'the' 'dog' '.'}
Hidden states: {'name' 'verb' 'preposition' 'article' 'noun' 'punctuation'}
```

Your task: Find a short story. Create a training file with the observations and hidden states (youll have to do the part-of-speech labeling for each word). Run the MATLAB function hmmestimate using your training data to create the hidden-state transition probability matrix and the observations emission probability matrix. Then run the MATLAB function hmmgenerate using these matrices to probabilistically generate a new story of varying lengths.
Hints:
Don't spend too much time on labeling a super long training data unless you really want to.
Actually we provide a simple story in **story.mat**, and labeled the POS tag for you, so you can spend more time working on the next (RNN) Task.
After loading the data, you will find a cell called training, where

- training(:,1) are the words of a story (observations)

- training(:,2) are the POS tags (hidden states)
- The number of hidden states is 11.

hmmestimate and hmmgenerate only takes numerical sequences, so you need to write a mapping function to map every word string to a unique integer index.
You can then feed the data into MATLAB function calls just as you have done for previous tasks.

```
hints:
voc_dict = unique(lower(training(:,1)))
tag_dict = unique(upper(training(:,2)))
for i = 1:size(training,1),training{i,3} = find(strcmp(voc, lower(training{i,1}))),end
for i = 1:size(training,1),training{i,4} = find(strcmp(voc_dict, upper(training{i,2}))),end
```

Report the sequences generated by hmmgenerate with varying lengths. Try to test more and report top 5 sequences that make most sense.

Task #12: RNN - Get Prepared!

Don't get frustrated if you cannot get something exciting from Task 11. This part of the assignment will be really fun, because you are going to explore **The Unreasonable Effectiveness of Recurrent Neural Networks**.
First read the very insightful, detailed and popular blog post from Andrej Karpathy if haven't yet: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`
In this task, you are going to install Torch, a widely-used and cutting-edge deep learning library based on the programming language Lua.
Basically:

- 1) `http://www.psychocats.net/ubuntu/virtualbox` - Follow these directions to set up VirtualBox with 64-bit Ubuntu. Allocate at least 2GB RAM and as much Hard Drive Space as you can spare. (Skip this step if you're already in Ubuntu or OSX)
- 2) `http://torch.ch/docs/getting-started.html` - Follow these directions in your terminal to download and install torch
- 3) `https://github.com/karpathy/char-rnn` - Download RNN from here Alternatively you may want to use the rebased code developed by Justin Johnson `https://github.com/jcjohnson/torch-rnn` Basically they have the same interface. Compared to char-rnn, torch-rnn is up to 1.9x faster and uses up to 7x less memory
- 4) Look at the github README page and follow the installation instructions.
- 5) Run the Shakespear example (provided in the char-rnn github repo) and see if your char-RNN is all set.

Task #13: RNN - Be Creative!

Compared to an HMM model which usually requires the user to manually specify some explicit hidden states, RNN memorizes and learns from scratch, and the training data needed for a char-rnn is almost unconstrained: it is an **input.txt** file! That's it!
Now you are going to design your own **input.txt**! As you have already read in the blog post (Fun with RNNs section), the input to char-rnn could be:

- Concatenation of Paul Graham's essays.
- Shakespeak works
- (structured) Wikipedia pages
- Algebraic Geometry (Latex code)
- Linux Source Code

Here are some more examples from the Internet:

- If you like play card games: Generate Magic the Gethering cards text
- Generate Music using Char-rnn: `http://www.jianshu.com/p/cf3123c914da`
- Obama-RNN Machine generated political speeches `https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0#.ebowwzxoc`
- And probably the most hilarious one: Machine generated TED-Talks `https://www.youtube.com/watch?v=-0odHtJ1saY`

Collect YOUR OWN data, make sure to cite the source of your data and tell us how you collect them. And then run the char-rnn on your own data to see what your machine can generate.

Be creative and have fun!

Task #14: Play with RNN hyperparameters

There are many things you can tweak and play with for an RNN. Read the blog post and "Tips and Tricks" section in the char-rnn github repo.

- rnn_size - The default is 200. You can make it larger to increase the model complexity of your RNN. Of course this will depend on your hardware and how long you're willing to wait.
- num_layers - Defaults to 2. Like rnn_size will depend on your hardware and patience
- dropout - normal choice is 0.5. Dropout is a simple yet effective technique that regularize a deep neural network. Decreasing the dropout rate to see if your network is more likely to overfit the data.
- length - Default is 2000 characters. This will depend on your application.
- temperature - how "risky" the network. Definitely play with the temperature of the Softmax during sampling. Decreasing the temperature from 1 to some lower number (e.g. 0.5) makes the RNN more confident, but also more conservative in its samples. Conversely, higher temperatures will give more diversity but at cost of more mistakes.

Task #15: Submit your assignment

Congratulations! You have completed this assignment. Now summarize your answers, code and the outputs. Informative introduction section with necessary mathematical formations is encouraged.
If you choose to do the RNN part of the assignment. You need to also submit the data you collected, examples of generated results with different hyper-parameter settings. If the file size is large, upload it to Dropbox and send a pointer. Discuss the results.