CSC321 Tutorial 3:
Assignment 1 - Exploring a simple neural language model

Yue Li
Email: yueli@cs.toronto.edu

Wed 11-12 Jan 29
Fri 10-11 Jan 31

In this assignment, you will run code that trains a simple neural language model on a dataset of sentences that were culled from a large corpus of newspaper articles so that the culled sentences would have a highly restricted vocabulary of only 250 words. (word source: `rawSentences.txt`)

Download matlab files from here:
`http://www.cs.toronto.edu/~bonner/courses/2014s/csc321/assignments/hw1_matlab.zip`

1. 4grams.mat: training, validation, test data and vocabulary;
2. loadData.m: load "4grams.mat", make minibatch of size `mbsz` cases, keep the first `batchesToKeep` training cases (`mbsz`=100; `batchesToKeep`=1000);
3. prepMinibatch.m: split off the inputs and targets and do a 1-of-K softmax encoding for the targets;
4. fprop.m: forward propagation to compute the output activity as the probability of next possible word given the input words;
5. bprop.m: backward propagation to compute derivatives of network connection weights in order to update the weights;
6. computeCrossEntropy.m: given the learned weights, compute cross-entropy per case of a whole dataset such as a test dataset that has **NOT** been used to train the network;
7. train.m: the "main" file that does the following:

    7.1 **train** the networks on the training data (4grams.mat) with user-defined embedding dimension (`d`) and number of hidden units (`numHid`) using the Matlab scripts in the order listed above,
    7.2 **validate** the network at the end of each training iteration/epoch using the validation data,
    7.3 **stop** the network training when the validation error starts to increase and saves the results,
    7.4 **test** the final trained network using the test data (used once and only once!);

8. wordDistance.m: compute the Euclidean distance (aka: L2 norm) between two words based on the hidden representation of the words (more details in a moment);
9. zeroOneLoss.m: compare the predicted word with the highest probability with the true answer and output the number of incorrect predictions.

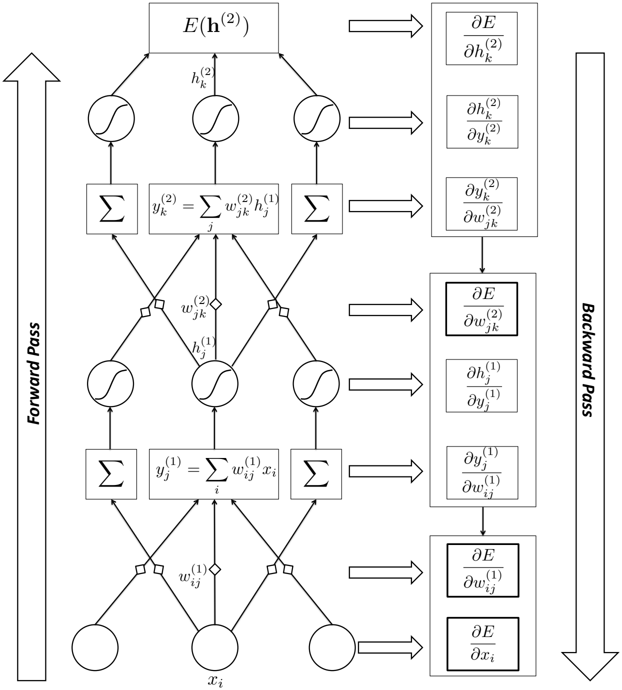
Please go through each script by yourselves.

- Before you can run the code, you will need to load the data it expects to find in the workspace. To do that, execute the following command:

```
[trainbatches, validbatches, testbatches, vocab] =
loadData(100, 1000);
```

- The line above creates the training, validation, and test data variables as well as a variable holding the 250 words in the vocabulary. It will load a training set with 1000 minibatches of 100 cases each.
- Let's take a look at the data and a few 4-gram instances ...

- The model you will train on these data produces a (1-of-K softmax) distribution over the next word given the previous three words as input.
- Since the neural network will be trained on 4-grams extracted from isolated sentences, this means that it will never be asked to predict any of the first three words in a sentence.
- The neural network learns a *d-dimensional embedding* for the (input) words in the vocabulary and has a hidden layer with `numHid` hidden units fully connected to the single output softmax unit.
- If we are so inclined, we can view the embedding as another (earlier) hidden layer with weights shared across the three words.



$E(\mathbf{h}^{(2)})$
$h_k^{(2)}$
$y_k^{(2)} = \sum_j w_{jk}^{(2)} h_j^{(1)}$
$w_{jk}^{(2)}$
$h_j^{(1)}$
$y_j^{(1)} = \sum_i w_{ij}^{(1)} x_i$
$w_{ij}^{(1)}$
x_i
Forward Pass
Backward Pass
$\frac{\partial E}{\partial h_k^{(2)}}$
$\frac{\partial h_k^{(2)}}{\partial y_k^{(2)}}$
$\frac{\partial y_k^{(2)}}{\partial w_{jk}^{(2)}}$
$\frac{\partial E}{\partial w_{jk}^{(2)}}$
$\frac{\partial h_j^{(1)}}{\partial y_j^{(1)}}$
$\frac{\partial y_j^{(1)}}{\partial w_{ij}^{(1)}}$
$\frac{\partial E}{\partial w_{ij}^{(1)}}$
$\frac{\partial E}{\partial x_i}$

- After you have loaded the data, you can set d and numHid like so:

```
d = 10; numHid = 50;
```

- and then run the main training script,

```
train;
```

- which will train the neural network using the embedding dimensionality and number of hidden units specified.

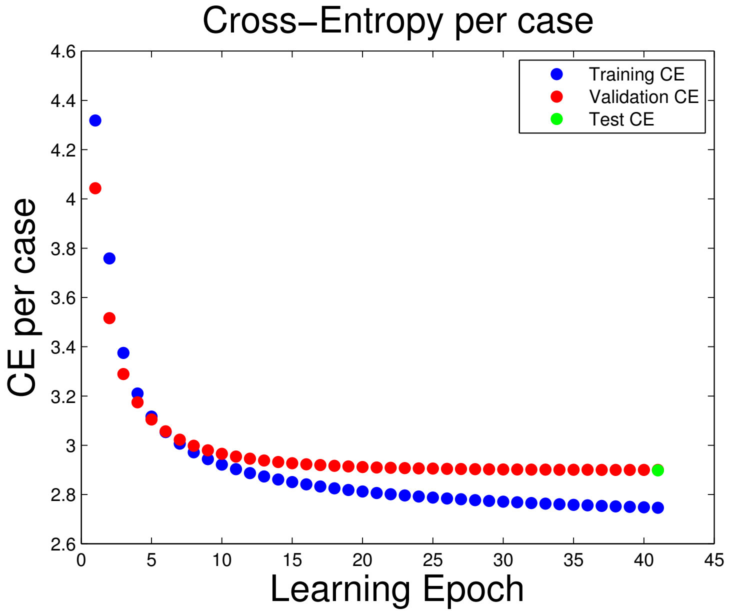
Algorithm 1 Pseudocode for Training, Validation, and Testing

```
for ep=1 to Epochs do
    for m=1 to numBatches do
        Train on $m^{th}$ batch of trainbatches
    end for
    Validate using validbatches
    if validation error increases then
        Stop training (or the outer loop)
    end if
end for
Test using testbatches
```

- The training script monitors the *cross-entropy error* (CE) on the validation data:

$$CE = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

  and uses that information to decide when to stop training.
- Training stops as soon as the validation error increases and the final weights are set to be the weights from immediately before this increase.
- This procedure is a form of “early stopping” and is a common method for avoiding overfitting in neural network training.
- **Recommandation**: Modify the script to record training and validation error at each epoch and plot them on the same plot for comparison. Clearly describe where and what you have modified in your report.


Cross–Entropy per case
CE per case
Learning Epoch
Training CE
Validation CE
Test CE
4.6
4.4
4.2
4
3.8
3.6
3.4
3.2
3
2.8
2.6
0
5
10
15
20
25
30
35
40
45

Here is a list of the variables of interest saved in the workspace:

- wordRepsFinal - the learned word embedding;
- repToHidFinal - the learned embedding-to-hidden unit weights;
- hidToOutFinal - the learned hidden-to-output unit weights;
- hidBiasFinal - the learned hidden biases;
- outBiasFinal - the learned output biases;
- epochsBeforeVErrUptick - the number of epochs of training before validation error increased. In other words, the number of training epochs used to produce the final weights above;
- finalTrainCEPerCase - The per case cross entropy error on the training set of the weights with the best validation error;
- finalValidCEPerCase - The per case cross entropy error on the validation set of the final weights (this will always be the best validation error the training script has seen);
- finalTestCEPerCase - The per case cross entropy error on the test set of the final weights.

You must train the model four times, trying all possible combinations of `d=10, d=40` and `numHid=50,numHid=200`:

1. `d=10, numHid=50`;
2. `d=10, numHid=200`;
3. `d=40, numHid=50`;
4. `d=40, numHid=200`.

For each of these runs, you must record:

1. `finalTrainCEPerCase` - the final cross entropy error on the training;
2. `finalValidCEPerCase` - the final cross entropy error on the validation;
3. `finalTestCEPerCase` - the final cross entropy error on the test sets;
4. `epochsBeforeVErrUptick` - the number of epochs before a validation error increase.

- Select the best configuration that you ran.
- The function `wordDistance` has been provided for you so that you can compute the distance between the learned representations of two words.
- The `wordDistance` function takes two strings, the `wordReps` matrix that you learned (use `wordRepsFinal` unless you have a good reason not to) and the vocabulary.
- For example, if you wanted to compute the distance between the words "and" and "but" you would do the following (after training the model of course):

```
wordDistance('and', 'but', wordRepsFinal, vocab)
```

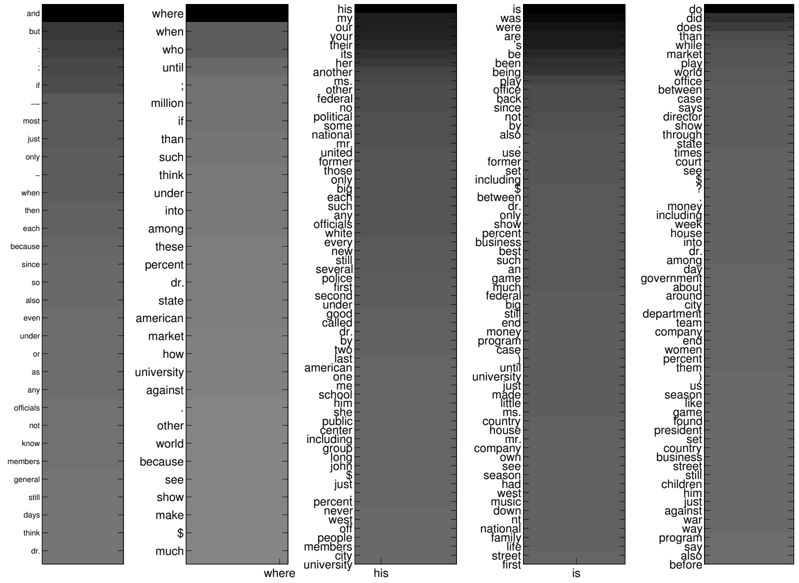
Formally, the `wordDistance` function simply takes the feature vector corresponding to each word and computes the *L2 norm* of the difference vector:

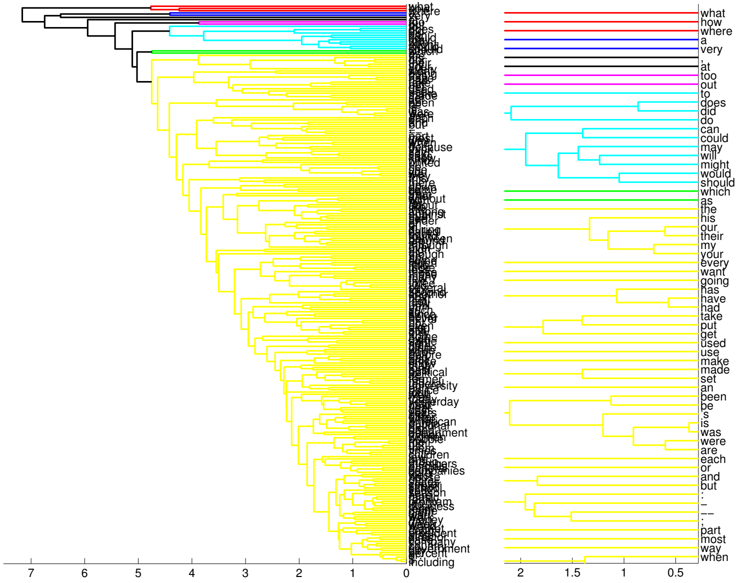
$$
\begin{aligned}
&d = 10\\
&\mathbf{w_i} = (h_{i,1},\ h_{i,2},\ h_{i,3}, \ldots,\ h_{i,10})\\
&\mathbf{w_j} = (h_{j,1},\ h_{j,2},\ h_{j,3}, \ldots,\ h_{j,10})\\
&d(\mathbf{w_i}, \mathbf{w_j}) = \sqrt{(h_{i,1} - h_{j,1})^2 + (h_{i,2} - h_{j,2})^2 + \ldots + (h_{i,10} - h_{j,10})^2}
\end{aligned}
$$

- You can only meaningfully compare the relative distances between two pairs of words and discover things like “the word ‘and’ is closer to the word ‘but’ than it is to the word ‘or’ in the learned embedding.”
- If you are especially enterprising, you can compare the distance between two words to the average distance to each of those words.
- Remember that if you want to enter a string that contains the single quote character in matlab you must escape it with another single quote. So the string apostrophe s, which is in the vocabulary, would have to be entered in matlab as: `'''s'`

- Compute the distances between a few words and look for patterns.
- See if you can discover a few interesting things about the learned word embedding by looking at the distances between various pairs of words.
- What words would you expect to be close together? Are they?
- Think about what factors contribute to words being given nearby feature vectors.
- **Recommandation**: Write a nested loop to compute distances between every pair of words (including a pair of the same words). The result will be a square and symmetrical distance matrix with diagonal entries all equal to zero (why?).

Sorted dist of “and”, “where”, “his”, “is”, “do” from other words (NB: Your results may differ slightly):





NB: Your results may differ slightly

What to hand in:

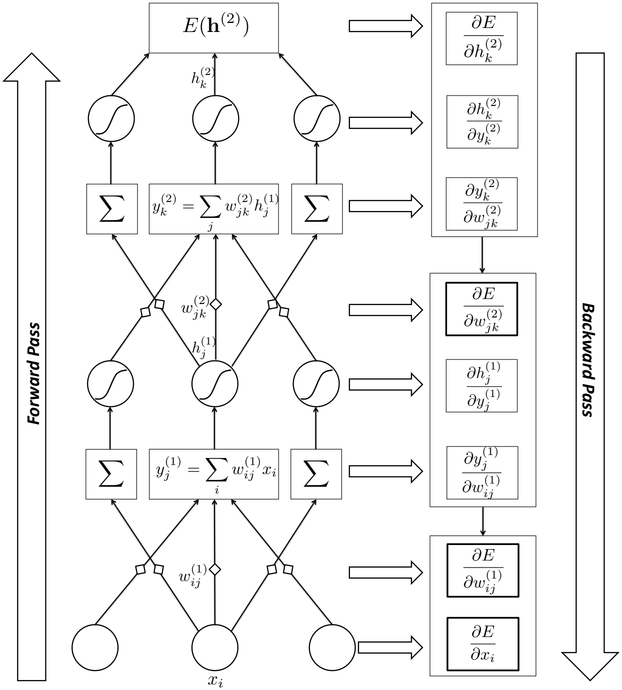
- You should hand in, along with the results you were instructed to record above, some clear, well-written, and concise prose providing a brief commentary on the results.
- **At most two pages will be graded** (excluding figures), so anything beyond two pages will be ignored during marking.
- *Figures (such as the illustrations above) are highly recommended to support your conclusion.*

Here are a few suggestions on what you should comment on, but they are certainly not exhaustive or meant to limit your discussion. You will be expected to offer some insights beyond what is mentioned below. The questions below are merely to help guide your thinking, please make your analysis go beyond them.

- You should give your reasoning about why the `d` and `numHid` you found to be the best actually is the best and how you defined "best".
- Comment a bit on everything you were asked to record and also what seemed to be happening during training.
- Explain all your observations.
- What did you discover about the learned word embedding?
- Why did these properties of the embedding arise?
- What things can cause two words to be placed near each other?
- What can you say about overfitting and generalization in light of your results?

A1 due on **Tuesday Feb 4 at 3pm** by electronic submission

Derivation of backpropagation (help you understand f/bprop.m)



$E(\mathbf{h}^{(2)})$
$h_k^{(2)}$
$y_k^{(2)} = \sum_j w_{jk}^{(2)} h_j^{(1)}$
$w_{jk}^{(2)}$
$h_j^{(1)}$
$y_j^{(1)} = \sum_i w_{ij}^{(1)} x_i$
$w_{ij}^{(1)}$
x_i
Forward Pass
Backward Pass
$\frac{\partial E}{\partial h_k^{(2)}}$
$\frac{\partial h_k^{(2)}}{\partial y_k^{(2)}}$
$\frac{\partial y_k^{(2)}}{\partial w_{jk}^{(2)}}$
$\frac{\partial E}{\partial w_{jk}^{(2)}}$
$\frac{\partial h_j^{(1)}}{\partial y_j^{(1)}}$
$\frac{\partial y_j^{(1)}}{\partial w_{ij}^{(1)}}$
$\frac{\partial E}{\partial w_{ij}^{(1)}}$
$\frac{\partial E}{\partial x_i}$

Formally, the forward pass computes the following activities, softmax, and cross-entropy (E):

$$y_j^{(1)} = \sum_i w_{ij}^{(1)} x_i \tag{1}$$

$$h_j^{(1)} = \sigma(y_j^{(1)}) = \frac{1}{1 + \exp(-y_j^{(1)})} \tag{2}$$

$$y_k^{(2)} = \sum_j w_{jk}^{(2)} h_j^{(1)} \tag{3}$$

$$h_k^{(2)} = \sigma(y_k^{(2)}) = \frac{\exp(y_k^{(2)})}{\sum_k \exp(y_k^{(2)})} \tag{4}$$

$$E = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln h_{nk}^{(2)} \tag{5}$$

Check `fprop.m`.

The backward pass computes the following partial derivatives:

$$\frac{\partial E}{\partial w_{jk}^{(2)}} = \sum_{n=1}^{N} (t_{n,k} - h_{n,k}^{(2)}) h_{n,j}^{(1)} \qquad \left(\text{use (2) and (4)}\right) \tag{6}$$

The gradient $\frac{\partial E}{\partial w_{jk}^{(2)}}$ in (6) is needed to adjust the connection weights $w_{jk}^{(2)}$ between the output and hidden layers.
In `bprop.m`, `dhidToOut` represents (6).

$$\frac{\partial E}{\partial y_k^{(2)}} = t_k - h_k^{(2)} \tag{7}$$

$$\frac{\partial y_k^{(2)}}{\partial h_j^{(1)}} = w_{jk}^{(2)} \tag{8}$$

$$\frac{\partial E}{\partial h_j^{(1)}} = \sum_k \frac{\partial E}{\partial y_k^{(2)}} \frac{\partial y_k^{(2)}}{\partial h_j^{(1)}} \qquad \Big(\text{use (7) and (8)}\Big) \tag{9}$$

$$\frac{\partial h_j^{(1)}}{\partial y^{(1)}} = h_j^{(1)}(1 - h_j^{(1)}) \tag{10}$$

$$\frac{\partial y_j^{(1)}}{\partial w_{ij}^{(1)}} = x_i \tag{11}$$

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{\partial E}{\partial h_j^{(1)}} \frac{\partial h_j^{(1)}}{\partial y_j^{(1)}} \frac{\partial y_j^{(1)}}{\partial w_{ij}^{(1)}} \qquad \Big(\text{use (9), (10), (11)}\Big) \tag{12}$$

The gradient $\frac{\partial E}{\partial w_{ij}^{(1)}}$ in (12) is needed to adjust $w_{ij}^{(1)}$ between the input and hidden layers. In `bprop.m`, `drepToHid` represents (12).

$$\frac{\partial E}{\partial y_j^{(1)}} = \frac{\partial E}{\partial h_j^{(1)}} \frac{\partial h_j^{(1)}}{\partial y_j^{(1)}} \qquad \Big(\text{use (9) and (10)}\Big) \tag{13}$$

$$\frac{\partial y_j^{(1)}}{\partial x_i} = w_{ij}^{(1)} \tag{14}$$

$$\frac{\partial E}{\partial x_i} = \sum_j \frac{\partial E}{\partial y_j^{(1)}} \frac{\partial y_j^{(1)}}{\partial x_i} \qquad \Big(\text{use (13) and (14)}\Big) \tag{15}$$

The gradient $\frac{\partial E}{\partial x_i}$ in (15) is needed to update the word representation of the d-dimension embedding. In `bprop.m`, `dwordReps` represents (15).