CS 533: Natural Language Processing **(Due: 02/18/20)**

Assignment 2

Name: Keya Desai, *NetID:* kd706 *Students discussed with:*

Problem 1: Log-Linear Models

((2 + 2 + 2 + 2 + 4 + 2 + 2 + 2 (+10) = 18 (+10) points))

1. The implementation of basic_features1_suffix3 that extracts features in basic_features1 plus suffixes of length up to 3 of window[-2] is:

```
def basic_features1_suffix3(window):
    features = basic_features1(window)
    for i in range(len(window[-2])):
        if(i>=3):
            break
        # appending the new features to the dictionary
        features.update({'c-1s%d=%s^w=%s'
                    % (i,window[-2][-(i+1):], window[-1]): True})

    return features
```

2. Number of feature types extracted using the three feature extractors, using 10% of the training data and 10% of the validation data are as follows:

Feature extractor	**#feature types**
basic_features1	61,144
basic_features1_suffix3	171,688
basic_features2	233,888

3. The formula of softmax used is:

$$softmax_i(v) = \frac{e^{v_i - v_{max}}}{\sum_j e^{v_j - v_{max}}} \tag{1}$$

The implementation of Eq. 1 is:

```
def softmax(v):
    v_max = np.max(v)
    v_exp = np.exp(v-v_max)
    summ = np.sum(v_exp)

    return v_exp/summ
```

The numerator and the denomination of Eq. 1 can be written as:

$$e^{v_i - v_{max}} = \frac{e^{v_i}}{e^{v_{max}}} \tag{2}$$

$$\sum_j e^{v_j - v_{max}} = \frac{\sum_j e^{v_j}}{e^{v_{max}}} \tag{3}$$

Dividing Eq. 2 by Eq. 3, gives the usual definition of softmax:

$$softmax_i(v) = \frac{e^{v_i}}{\sum_j e^{v_j}} \tag{4}$$

Hence, Eq. 1 is same as the usual definition of softmax Eq. 4.

4. The entry of the score vector q_ corresponding to target word y_:

$$[q_]_{ind(y)} = w^T_{\phi(x,y)} = \sum_{i=1:\phi_i(x,y)=1}^{d} w_i \tag{5}$$

The implementation is:

```
def compute_probs(self, x):
        q_ = np.zeros(len(self.token_to_idx))
        # list of all words following x
        y = list(self.x2ys[tuple(x)])

        for word in y:
            window = x.copy()
            window.append(word)
            # list of feature indices corresponding to the given window
            idx = self.fcache[tuple(window)]
            summ = 0
            for i in idx:
                summ += self.w[i]
            q_[self.token_to_idx[word]] = summ

        return softmax(q_)
```

5. The gradient update in function do_epoch:

```
    window = x.copy()
    window.append(y)
    idx = self.fcache[tuple(window)]
    all_y_following_x = list(self.x2ys[tuple(x)])

    for i in idx:
        summ = 0
        for word in all_y_following_x:
            window = x.copy()
            window.append(word)
            inds = self.fcache[tuple(window)]

            for j in inds:
                if(j == i):
                    summ += q[self.token_to_idx[word]]
                else:
                    self.w[j] -= self.lr * q[self.token_to_idx[word]]

        self.w[i] += self.lr * (1-summ)
```

6. The best validation perplexity in 10 epochs with basic_features1 is **149.570**2 for lr=2.

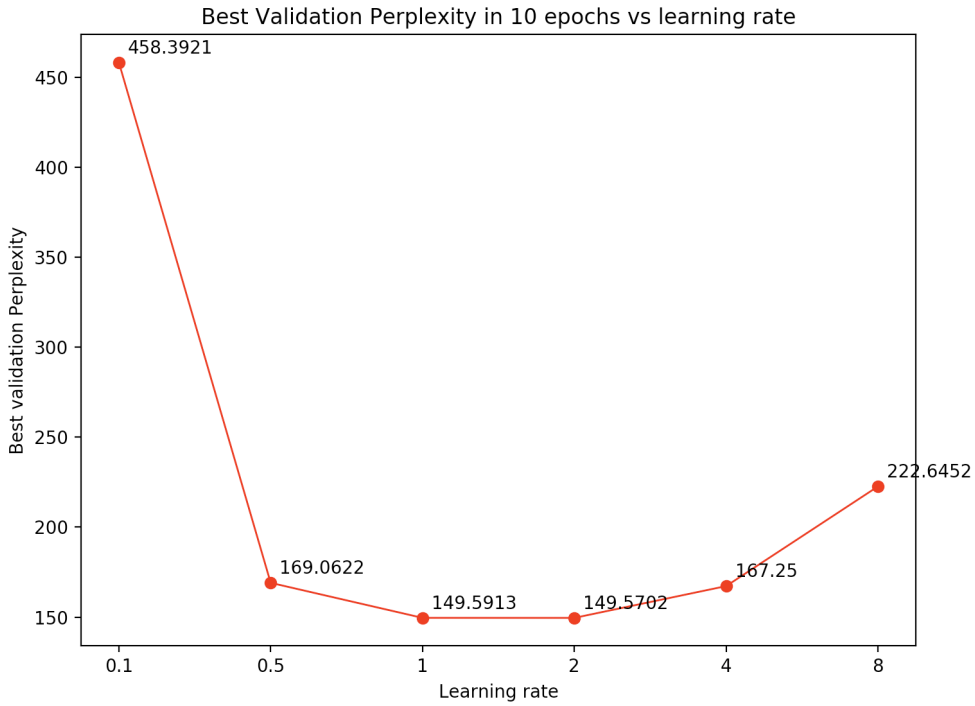

Figure 1. Validation perplexity vs learning rate

7. Top-10 feature types that have been assigned the highest weight with basic_features1 and lr = 2 are shown in Fig. 7. The weights of a feature in a log linear model are associated with the probability of the words in it appearing together. basic_features1 extracts a bigram feature for a window of size 3. More the number of times a bigram appears together, more the probability of it appearing one after the other in test corpus. The result, thus, makes sense. 'prime minister', 'barack obama', 'laurent gbagbo' are proper nouns and hence have the highest probabilities. Other bigrams too are often seen together.

```
1030: c-1=prime^w=minister                ( 16.2011)
1474: c-1=according^w=to                  ( 15.7474)
 690: c-1=barack^w=obama                  ( 15.3850)
 528: c-1=laurent^w=gbagbo                ( 15.3850)
2213: c-1=end^w=of                        ( 15.2720)
4546: c-1=ahead^w=of                      ( 15.2557)
3167: c-1=part^w=of                       ( 14.8111)
2092: c-1=such^w=as                       ( 14.7670)
 768: c-1=able^w=to                       ( 14.7210)
2054: c-1=members^w=of                    ( 14.6220)
```

Figure 2. Top 10 features for lr = 2 with basic_features1

The best model that I get, however, is using basic_features1_suffix3 for lr = 0.50, 79.5003. But the top-10 features for this model (shown in Fig. 7) are not interpretable.

```
 133: c-1s0=s^w=,                         (  7.3121)
2140: c-1s0=s^w=of                        (  7.2979)
 119: c-1s0=s^w=.                         (  7.2358)
1007: c-1s0=s^w=and                       (  7.1594)
  35: c-1s0=e^w=of                        (  7.0818)
 394: c-1s0=s^w=in                        (  7.0399)
 705: c-1s0=s^w=that                      (  7.0280)
4135: c-1s0=a^w=,                         (  7.0189)
13005: c-1s0=6^w=,                        (  6.9992)
2811: c-1s0=s^w=for                       (  6.9792)
```

Figure 3. Top 10 features for lr = 2 with basic_features

8. For 10 random seeds, at learning rate = 1, the mean and standard deviation of the perplexity are as follows:

Mean	147.319
Standard Deviation	4.537

9. [BONUS] Before using the entire training and validation data, I tested different models on 10% of the data. Out of all the feature functions, basic1suffix3 gave the minimum perplexity of 79.5 for lr = 0.25. Hence, I trained the model on the entire training data using basic1suffix3 and lr = 1 and lr = 0.25. The table below gives the validation perplexity at the end of each epoch for the two models.

	Epoch				
lr	**1**	**2**	**3**	**4**	**5**
1	243.0453	252.5182	102.8083	**101.1708**	103.2816
0.25	122.7985	106.4633	**99.1144**	-	-

For lr = 0.5, I got better validation perplexities than for lr = 1 in fewer epochs. Training the entire data takes a lot of time. Due to this reason, I could not complete the training. But this combination of hyper parameters gives the best perplexity.

The command for running the results:
python assignment2.py –lr 1 –train_fraction 1 –val_fraction 1
python assignment2.py –lr 0.25 –train_fraction 1 –val_fraction 1

(I collaborated with my fellow classmates (phj15, tn268) for the bonus question only.)

Problem 2: Feedforward Neural Language Model (2 + 2 + 2 + 2 + 2 + 2 = 12 points)

1. input dimension = context size * word embedding size
   output dimension = vocab size
   Hence, the correct initialization of self.FF is:

```
self.FF = FF(dim_input = wdim*self.nhis, dim_hidden = hdim,
             dim_output = self.V, num_layers = nlayers)
```

2. In the Batch-version of forward in FFLM, logits will be defined as follow:

```
# self.E(X) gives the embedding correspoing to each word in X.
# view reshapes the vector from X × self.nhis × self.wdim
# to B × (self.nhis * self.wdim)
logits = self.FF.forward(self.E(X).view(X.shape[0],-1))
```

3. For nlayers = 0, batch_size = 16, epochs = 10, varying wdim, hdim and lr, changes the optimised perplexity as follows:

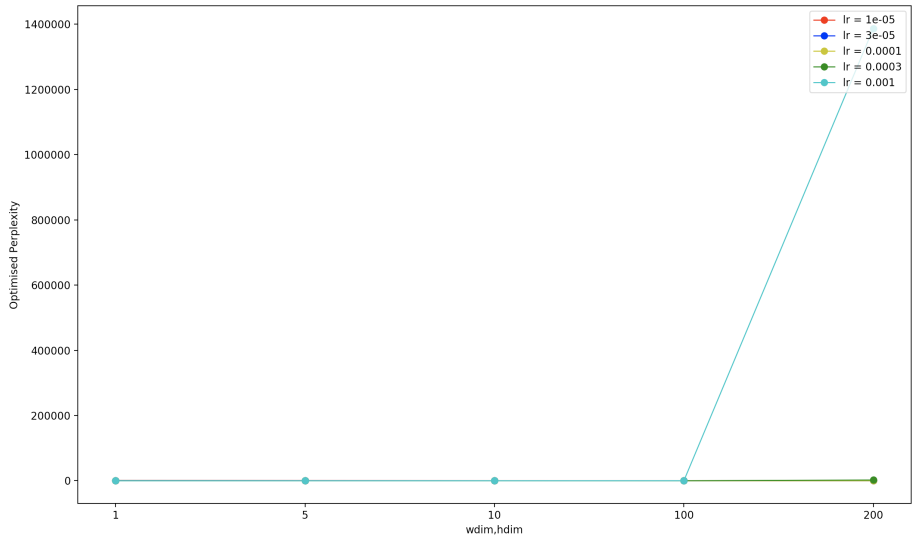

Figure 4. Optimised Perplexity vs wdim=hdim = [1,5,10,100,200] for learning rates = [0.00001,0.00003,0.0001,0.0003,0.001], nlayers = 0

For lr = 0.001, the perplexity is huge for wdim=hdim=200. To visualise the distribution of perplexity better, I made the same plot but excluding lr = 0.001.

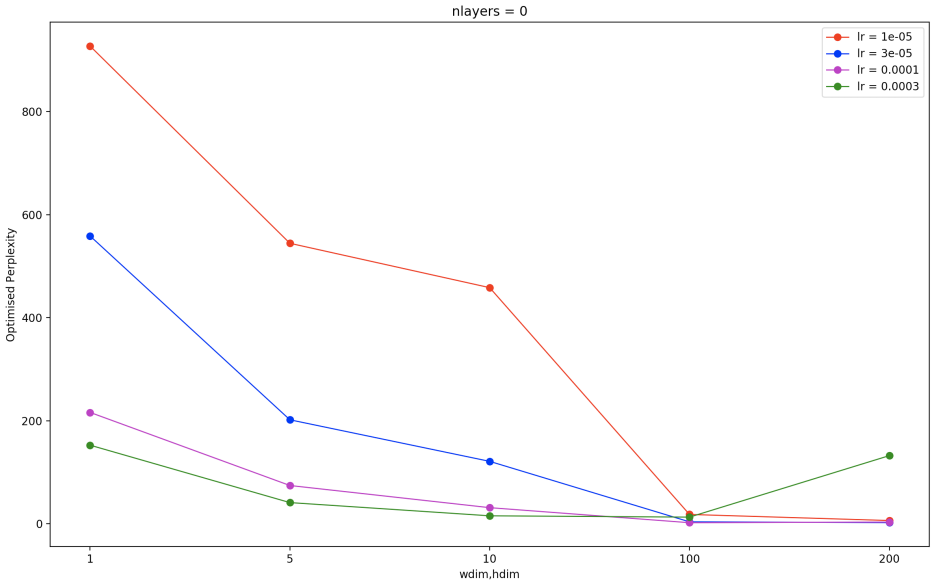

Figure 5. Optimised Perplexity vs wdim=hdim = [1,5,10,100,200] for learning rates = [0.00001,0.00003,0.0001,0.0003], nlayers = 0

From the plot, it can be inferred that for all learning rates, the optimised perplexity improves with increase in wdim and hdim, becoming almost equal at wdim=hdim=100. Overall, the minimum perplexity is achieved at learning rate = 0.0003

4. Repeating the same exercise with nlayers = 1, gives the following result (excluding result for lr=0.001 for better visualisation):

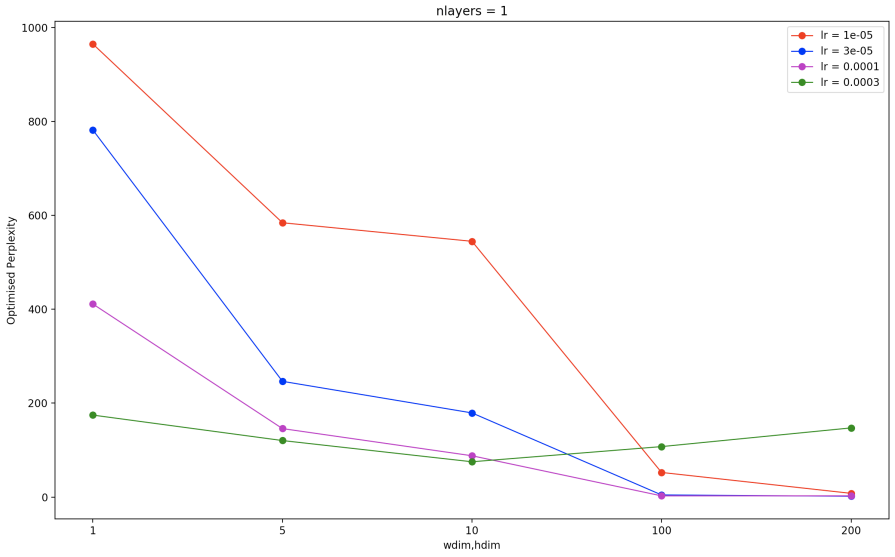

Figure 6. Optimised Perplexity vs wdim=hdim = [1,5,10,100,200] for learning rates = [0.00001,0.00003,0.0001, 0.0003], nlayers = 1

For nlayers=1, the trend in the perplexity with changin wdim,hdim and lr remains the same as that with nlayers=1. However, for most cases, minimum perplexity is achieved by keeping nlayers=0.

5. For epochs = 1000, setting wdim = hdim = 30 and learning rate = 0.0003, running the model with nlayers = 0 and nlayers = 1, gives the following result:

	nlayers = 0	**nlayers = 1**
Epochs to convergence	176	123
Training loss	0.605	1.087
Optimised Perplexity	1.765	2.738

The hypothesis stated is that 'it takes bigger models to converge, but when they do they can achieve a smaller training loss'. But, we get the opposite result. The smaller linear model (nlayers = 0), takes more epochs to converge to a smaller training loss than the nonlinear bigger model (nlayers = 1). I plotted training loss vs epochs. For every epoch, the nonlinear model has bigger training loss.

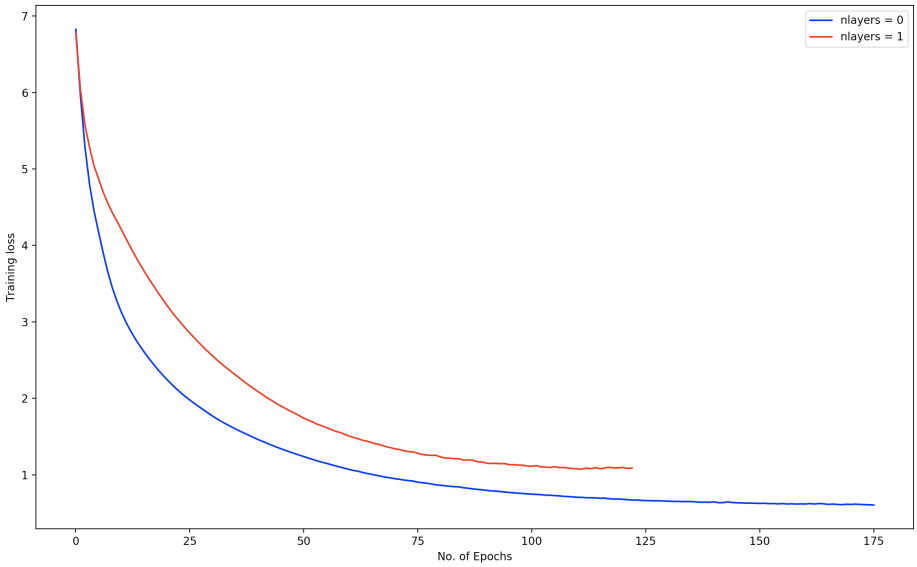

Figure 7. Training loss vs No. of epochs for nlayers = 0 and nlayers = 1

6. Below are snippets of the nearest neighbours I got for two cases:

```
lr = 0.0003, epochs = 10, Optimized Perplexity: 45.670284, nlayer = 0
player: 2-1 included plane bring ceremony winter discuss wishes fernandez if
participants: medals heat dalai regulations well fourth cuba name williams angry
smoot: rochette qualify npc smoke burger immediate part gala office package
commitment: get leg michelle my among shape that fourth . 1
article: even talk hearing won top role war of relations gone
miller: goals full gala gas set olympics stricter football penalty work
agenda: bid winning clinton interest forgiven easy fed territory begin proposal
carlos: election website pollution regional package farmers disguise legislation republicans emissions
boost: - against confident around big at criticized draw and majority
winner: state rochette kohn erekat monday team 6-4 gap process coalition
```

Figure 8. Nearest neighbours of trained word embeddings for lr = 0.0003, epochs = 10

```
lr = 0.0003, epochs = 1000, Optimized Perplexity: 2.250153
won: cheeseburgers farmers after left attend george 2006 into miss open
message: here surgery tournament firms statement online captain podium kick yin
minister: three trade new bilateral everything 17 stricter structure market rican
ministers: team step 29 dinner goals full gold stronger very annual
visit: bet verbeek seconds seat cholesterol gala ramsey leg nation programme
along: friendly weapons bet number career 28 cooperation speaker relief athletes
chelsea: it villa 2010 left think tax she croatia capello parties
world: although representatives goals little forgiven league up ministers 7-5 relationship
formal: our former parliamentary quarterfinals sen. such other during meet some
structure: natural minister such new under cyber last play coalition scoring
```

Figure 9. Nearest neighbours of trained word embeddings for lr = 0.0003, epochs = 1000

Most of the word embedding are sensible words. But there a few words like 'gibbs' and 'afb' which do not make any sense (neighbours of words 'article' in first snapshot).

Some of the nearest neighbours do make sense.

- For example, consider the nearest neighbours of 'carlos':
  carlos: election website pollution regional package farmers disguise legislation republicans emissions
  From the neighbouring words, it can be assumed the carlos is a name of a political figure, given the words 'legislation','republicans', 'election','regional'. The person might be associated with some 'pollution' or 'farmers' related 'legislation'. (Link to article realted to this topic: republican-carlos-curbelo-pitches-carbon-tax-climate-change)
- miller: goals full gala gas set olympics stricter football penalty work
  (Found a fun read relating the words 'miller' with 'olympics' 'stricter' 'football' 'penalty': baylor-notebook-miller-flagged-for-third-targeting-penalty-will-miss)
- chelsea: it villa 2010 left think tax she croatia capello parties
  'chelsea' here might be the place or name of a person. So some of it makes sense.
  'capello' is a famous Italian football player. Maybe he has some relation with 'chelsea'.

But most other pairings of neighbours look gibberish to me. Some examples:

- won: cheeseburgers farmers after left attend george 2006 into miss open
- message: here surgery tournament firms statement online captain podium kick yin
- commitment: get leg michelle my among shape that fourth . 1
- women: challenge steinberg everything ' focus week house north infrastructure stoke

One possible reason for the neighbours not making sense might be beacuse of the training corpus. The training corpus might not contain many instances of a particular word, for eg 'women', in context to something which is exclusively associate with we 'women'. Hence, the nearest neighbours will not make sense to us without looking at the context in which it appears in the training corpus.