Midterm Examination

1. Tensorflow and Backpropagation

(1) Using some new notation:

$$z_1 = Wx + b_1$$
$$h = ReLU(z_1)$$
$$z_2 = Uh + b_2$$
$$\hat{y} = softmax(z_2)$$
$$J = CE(y, \hat{y})$$

from lectures,

$$\frac{\partial J}{\partial z_2} = \delta_2 = (\hat{y} - y)$$

Backpropagating this error:

$$\boxed{\frac{\partial J}{\partial b_2} = \delta_2} \qquad \boxed{\frac{\partial J}{\partial U} = \delta_2 h^T}$$

$$\frac{\partial J}{\partial h} = U^T \delta_2$$

$$\frac{\partial J}{\partial z_1} = \delta_1 = ReLU'(z_1) \odot U^T \delta_2$$

$$\delta_1 = \mathbb{1}\{z_1 > 0\} \odot U^T \delta_2$$

$$\delta_1 = \mathbb{1}\{W x_1 + b_1 > 0\} \odot U^T \delta_2$$

$$\boxed{\frac{\partial J}{b_1} = \delta_1} \qquad \boxed{\frac{\partial J}{\partial W} = \delta_1 x_1^T}$$

(2) They might be required to compute the gradient. for eg. to calculate gradient of sigmoid fun^n, the o/p of fun^n is needed.

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

Similary let $a = bc$, then to compute gradient $\frac{\partial a}{\partial b}$, the i/p c is needed.

$$\frac{\partial a}{\partial b} = c$$

(3) Doing separate softmax will require computing exponents for potentially large numbers, which might result in NaN for the programming language.
On the other hand, CE node doesn't actually compute exponents of unscaled inputs to compute the loss. Therefore it is more efficient.

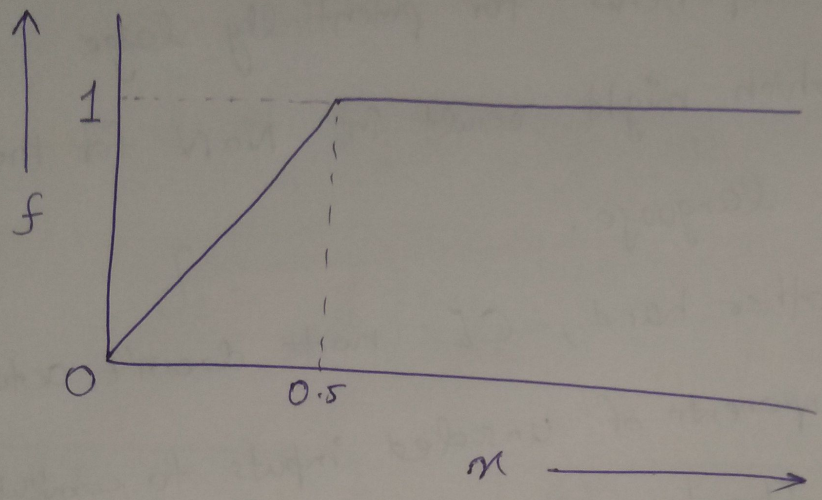
2. Word2Vec

(1) (i) $J(\theta) = \frac{1}{2} \sum_{i=1}^{W} \sum_{j=1}^{W} f(P_{ij}) (u_i^T v_j - \log P_{ij})^2$

$$\frac{\partial J(\theta)}{\partial u_i} = \sum_{j=1}^{W} f(P_{ij}) (u_i^T v_j - \log P_{ij}) v_j$$

$$\frac{\partial J(\theta)}{\partial v_j} = \sum_{i=1}^{W} f(P_{ij}) (u_i^T v_j - \log P_{ij}) u_i$$

(ii) $f(x)$ looks like this



(iii) $f(x)$ should have this form since we want that pairs with high frequency be given higher weightage. We clip it at a maximum value since we don't want most of the weightage given only to those few pairs with very high frequency.

(iv) Unlike Skipgram/CBOW, GloVe provide statistical properties. Also fast to train.

(2) – Concatenate both vector of a word $u_w \in v_w$ to form the representation $[u_w; v_w]$

– Add the vectors

3. Deep NLP in Practice

(1) We will take the distributed representation of each token in the note and concatenate them. Since there are not many tokens, this will not blow up very large.

(2) Adding the word vectors seems a good choice. If a word occurs only few times (eg. disease name), then averaging would reduce its weights for very large notes. While adding the vectors would preserve such info.

(3) For sigmoid ~~[illegible]~~, normalization is very important. This is because if a component is too high (or low) then the corresponding linear transformation of NN will result in

(or very small)
a large numbers. This will result in exploding/ vanishing gradient problems for sigmoid & ReLU nonlinearities.

(4) Because RNN learns how the illness of patient has evolved, in a natural way and based on that, it not ~~only~~ learns only the average of past vectors, but where patient vectors stands now.

(5) Sigmoid. Since in the softmax case, the presence of one disease would lower the probability of all other diseases. This contradicts our assumption that diseases are not mutally exclusive.

(6) We will now use 300 neuron units instead of 70,000 in the second last layer & k neurons in a final layer. Here k is the maximum no. of diseases

under a class (on an average 233).
So, the NN will first predict a class, and then the disease from that class.

$$z_i^{(n-1)} = f_1(W z^{(n-2)} + b_i^{(1)}) \quad i \in [300]$$

$$z_i^{(n)} = f_2(U z^{(n-1)} + b_2) \quad i \in [K]$$

$$W \in \mathbb{R}^{N \times 300}$$

$$U \in \mathbb{R}^{K \times 300}$$

(7) A bidirectional model would use the future information of patient notes which was not available in current time step to make the prediction at current time step.

(8) Semantic

4. LSTMs, GRUs, and Recursive Networks

(1) (a) Very deep feed-forward NN, Recurrent NN & Recursive NNs suffer from vanishing gradient problem since all of them backpropage errors to very farther in the network. And for ~~networks with~~ small errors, ~~it becomes~~ the gradient becomes zero as it propagetes through the network.

(b) False. Adding more layers will only increase the vanishing gradient problem.

(c) False, L_2 regularizer will push weights of neurons towards zero. Which will cause gradients to become zero.

(d) True. Clipping gradients will ensure that no large gradients are ~~forced to next~~ backpropagated.

(2) (a) False. After multiplication with U and non-linearities, $h_t = h_{t-1}$ is generally false.

(b) True. If f_t is small (or zero), then c_{t-1} will not have much contribution for calculating c_t & hence, h_t.

(c) True. Sigmoid is always b/w 0 & 1.

(d) False. Sigmoid only guarantees that o/p is b/w 0 & 1. But entries of resulting vector may not sum to 1.

(3) $h_t \in \mathbb{R}^{d_h \times 1}$ [Assuming real numbers]

$x_t \in \mathbb{R}^{d_x \times 1}$

$W^{(2)} \in \mathbb{R}^{d_h \times d_x}$, $W^{(r)} \in \mathbb{R}^{d_h \times d_x}$,

$U^{(2)} \in \mathbb{R}^{d_h \times d_h}$, $U^{(r)} \in \mathbb{R}^{d_h \times d_h}$,

$W \in \mathbb{R}^{d_h \times d_x}$, $U \in \mathbb{R}^{d_h \times d_h}$

(4)

$W_2^{(z)} \in \mathbb{R}^{d_h \times d_h}$

$U_2^{(z)} \in \mathbb{R}^{d_h \times d_h}$

$W_2^{(r)} \in \mathbb{R}^{d_h \times d_h}$

$U_2^{(r)} \in \mathbb{R}^{d_h \times d_h}$

$W_2 \in \mathbb{R}^{d_h \times d_h}$

$U_2 \in \mathbb{R}^{d_h \times d_h}$

5. Hyper-Parameter Tuning

(1) Bias term acts as the translation & moves the prediction in the direction of original distribution. Adding regularization would prevent this.

(2) (i) Small weights helps in reducing overfitting. Also, very large weights might cause exploding gradient problem for some deep NNs.

(ii) Initializing weights randomly helps in faster training of the model.

(3) (i) Underfitting

(ii) Make model more complex

(4) $\dfrac{f(x+h) - f(x-h)}{2h} = \sum_{n=0}^{\infty} \dfrac{f^{(n)}(x+h)^n - (-h)^n}{n!} - \dfrac{f^{(n)}(x-h)(h)^n}{n!}$ $\cdot 2h$

61/77