CS 224N, Winter 2017
Practice Midterm #2 Solutions

Note: This practice midterm is based on the CS224D midterm given in Spring 2016.

---

1 TensorFlow and Backpropagation

A TensorFlow computation is described by a directed graph. Fig. 1 shows a typical TensorFlow computation graph. In the graph, each node has zero or more inputs and outputs. For example, the *MatMul* node in the left bottom has inputs **W** and **x** and it outputs **Wx**. Tensors (arbitrary dimensionality arrays such as vectors or matrices) flow along the edge of graph, e.g. the output of *MatMul* is fed into *Add*.

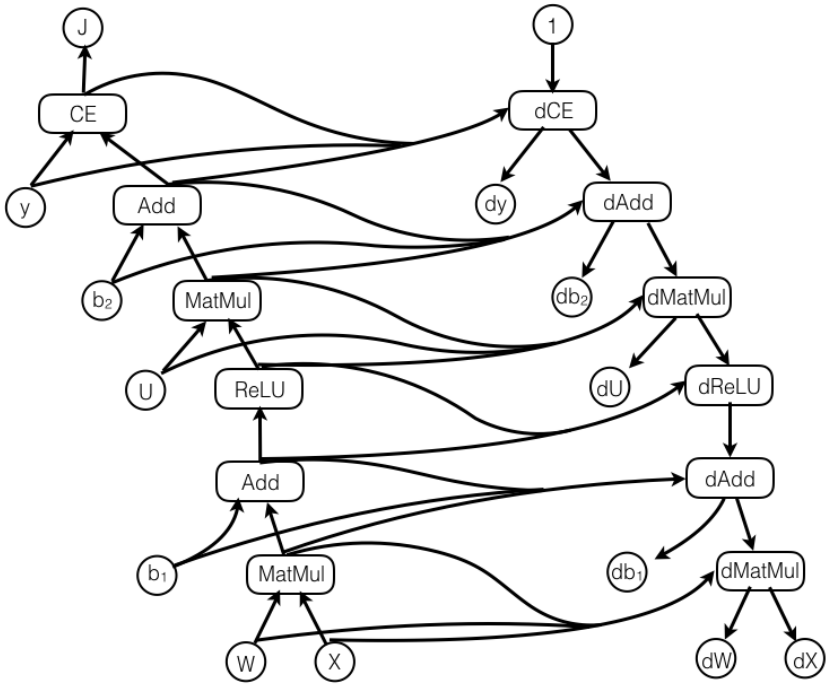


Figure 1: TensorFlow graph

The left part of the figure represents the computation graph of forward propagation. The

computation it represents is:

$$\mathbf{h} = \text{ReLU}(\mathbf{W}\mathbf{x} + \mathbf{b_1}) \tag{1}$$
$$\hat{\mathbf{y}} = softmax(\mathbf{U}\mathbf{h} + \mathbf{b_2}) \tag{2}$$
$$J = \text{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i \tag{3}$$

Here ReLU (rectified linear unit) performs element-wise rectified linear function:

$$\text{ReLU}(z) = \max(z, 0) \tag{4}$$

The dimensions of the matrices are:

$$\mathbf{W} \in \mathbb{R}^{m \times n} \quad \mathbf{x} \in \mathbb{R}^n \quad \mathbf{b_1} \in \mathbb{R}^m \quad \mathbf{U} \in \mathbb{R}^{k \times m} \quad \mathbf{b_2} \in \mathbb{R}^k \tag{5}$$

1) (8 points) Use backpropagation to calculate these four gradients

$$\frac{\partial J}{\partial \mathbf{b}_2} \quad \frac{\partial J}{\partial \mathbf{U}} \quad \frac{\partial J}{\partial \mathbf{b}_1} \quad \frac{\partial J}{\partial \mathbf{W}} \tag{6}$$

Hint: You can use the following notation:

$$1\{x > 0\} = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x \leq 0. \end{cases}$$

Using it on a matrix would perform an element-wise operation, e.g.

$$1\{\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} > 0\} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7}$$

$$z_2 = \mathbf{U}\mathbf{h} + \mathbf{b_2} \tag{8}$$
$$\boldsymbol{\delta_1} = \frac{\partial J}{\partial z_2} = \hat{\mathbf{y}} - \mathbf{y} \tag{9}$$
$$\frac{\partial J}{\partial \mathbf{b_2}} = \boldsymbol{\delta_1} \tag{10}$$
$$\frac{\partial J}{\partial \mathbf{U}} = \boldsymbol{\delta_1}\mathbf{h}^T \tag{11}$$
$$\boldsymbol{\delta_2} = \mathbf{U}^T\boldsymbol{\delta_1} \circ \mathbf{1}\{\mathbf{h} > 0\} \tag{12}$$
$$\frac{\partial J}{\partial \mathbf{b_1}} = \boldsymbol{\delta_2} \tag{13}$$
$$\frac{\partial J}{\partial \mathbf{W}} = \boldsymbol{\delta_2}\mathbf{x}^T \tag{14}$$

2) The right part of the figure represents the computation graph of backpropagation. Each *gradient operation node* in this part (e.g. *dMatMul*) takes as input not only the partial gradients computed already along the backpropagation path, but also, optionally, the inputs and outputs of the corresponding operation node in the forward propagation to calculate gradients. Here, the *MatMul* node in the bottom left corresponds to *dMatMul* node in the bottom right, and the *ReLU* node corresponds to the *dReLU* node. Briefly explain why the corresponding inputs and/or outputs of the forward operation node are sometimes needed when backpropagating to calculate gradient. Give an example for each case.

(1 point) Explanation:

Because they may be used in the expression for gradient.

(i) (2 points) Case 1: The input of the corresponding operation node is needed

If $\mathbf{h}' = \mathbf{W}\mathbf{x}$, and we already know $\boldsymbol{\delta} = \frac{\partial J}{\partial \mathbf{h}'}$. Then $\frac{\partial J}{\partial \mathbf{W}'} = \boldsymbol{\delta}\boldsymbol{x}^T$. So to calculate $\frac{\partial J}{\partial \mathbf{W}'}$ we need the value of $\mathbf{x}$, which is input to the *MatMul* node.

These answers are also correct: ReLU, CE

(ii) (2 points) Case 2: The output of the corresponding operation node is needed

Because $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, calculating backpropagation of sigmoid function would require the output of sigmoid node.

These answers are also correct: ReLU, tanh

3) (2 points) The *CE node* (softmax cross-entropy) in the graph expects unscaled inputs (they are not yet exponentiated), since it performs a softmax internally for efficiency. Give a reason why this is more efficient than a separate softmax and cross-entropy layer.

Recall that the gradient of J with respect to the unexponentiated inputs to CE node is $\hat{y} - y$. This is very easy to compute.

We also give full credits to these reasons:

1. Don't have to first take exp and then take log. Improve numerical stability.
2. y is one hot so we only need to compute $\hat{y}_k$ where k is the correct label.

2 Word2Vec

1) Recall the loss function for GloVe:

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

where P_{ij} (a scalar) is the probability that word j appears in the context of word i,
$f : \mathbb{R} \to \mathbb{R}$ is a function that gives a weight to each (i, j) pair based on its probability P_{ij},
u_i is a column vector of shape $(d \times 1)$ representing the output vector for word i, and
v_j is a column vector of shape $(d \times 1)$ representing the input vector for word j

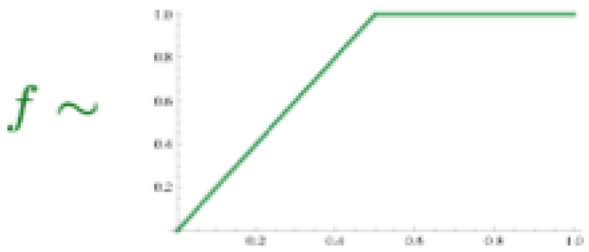
---

(i) (2 points) Calculate the gradient of the loss function with respect to input and output vectors: $\frac{\partial J(\theta)}{\partial u_i}$ and $\frac{\partial J(\theta)}{\partial v_j}$.

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

(ii) (2 points) Show what the function $f(x)$ looks like by drawing it.

Plot of $f(x)$:



(iii) (2 points) Explain in one or two sentences why should $f(x)$ have the form that you have drawn.

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

(iv) (2 points) Give one advantage of GloVe over Skipgram/CBOW models.

Fast training

2) (2 points) What are two ways practitioners deal with having two different sets of word vectors U and V at the end of training both Glove and word2vec?

1. Add them
2. Concatenate them.

3 DeepNLP in Practice

You are consulting for a healthcare company. They provide you with clinical notes of the first encounter that each patient had with their doctor regarding a particular medical episode. There are a total of 12 million patients and clinical notes. Figure 2 shows a sample clinical note. At the time that each clinical note was written, the underlying illnesses associated with the medical episode were unknown to the doctor. The company provides you with the true set of illnesses associated with each medical episode and asks you to build a model that can infer these underlying illnesses using only the current clinical note and all previous clinical notes belonging to the patient. The set of notes provided to you span 10 years; each patient therefore can have multiple clinical notes (medical episodes) in that period.

History
ROS: no change in bowel/urinary habits
Meds: no Rx or OTC.
All: NKDA.
FH: mother - schizophrenia
PMH: asthma, good control. No surgeries, traumas or hospital.
SxH: sex. active with multiple F and M partners, inconsistent use of condoms, no h/o STDs
SH: NO cig/eoth. Uses PCP and ecstasy x 1y, once/week, last intake yesterday. College student.

Physical Examination
Pt is in NAD. Speech fluent, talkative, mood euphoric, affect c/w mood,
behavior inappropriate. Cooperative. Appearance disheveled.
VS: WNL
HEENT: EOMI, PERRLA.
Neck: NL Thyroid Gland

Figure 2: Sample clinical note (source USMLE)

Each note can contain any number of tokens (see Figure 2). Some tokens (e.g. "Meds") occur more frequently than others in the collection of notes provided to you.

You call your former CS224D TA for advice. He tells you to first create a distributed representation of each patient note by combining the distributed representations of the words contained in the note.

---

1) (2 points) Given the sample note provided in Figure 2, how would you map the various tokens into a distributed vector representation?

Given that there is no proper sentence structure a bag of words approach would be most appropriate. We awarded full credit to students who recommended other word vector representation techniques (Skip-gram, GloVe, CBOW etc.). We awarded partial credit to students who did not explicit discuss which word vector representation technique should be applied.

2) (2 points) You have the option of representing each note as the summation of its constituent word vectors or as the average of its word vectors. Both seem reasonable. What's your best course of action? Briefly justify your selection.

Try both and see which one yields better results downstream. We awarded full credit to students who proposed either averaging or summing and provided a reasonable justification for their choice (e.g. summing captures the length of the note, averaging disregards the length etc.)

3) (2 points) Your former TA tells you that you must normalize (magnitude-wise) your word-vectors before you perform the operation you decided to do in 2). Assuming you might try a standard neural network model, for which nonlinearities might that matter more?

It will matter for non-linearities that flatten in higher regimes (softmax, sigmoid, tanh etc.). It will not matter for RELU. Large-magnitude word vectors will cause the non-linearity units to saturate. At saturation regimes the gradient is close to zero. Your model will not learn effectively as the error would not propagate. Partial credit was awarded to students who identified the non-linearities for which normalization is important, but did not provide a suitable justification thereof.

4) (2 points) You now have a distributed representation of each patient note (note-vector). You assume that a patient's past medical history is informative of their current illness. As such, you apply a recurrent neural network to predict the current illness based on the patient's current and previous note-vectors. Explain why a recurrent neural network would yield better results than a feed-forward network in which your input is the summation (or average) of past and current note-vectors?

RNNs allows you to capture temporal relationships (e.g. sequence of events). You would lose that information by summing your note-vectors.

5) (2 points) A patient may have any number of illnesses from a list of 70,000 known medical illnesses. The output of your recurrent neural network will therefore be a vector with 70,000 elements. Each element in this output vector represents the probability that the patient has the illness that maps to that particular element. Your former TA tells you that illnesses are not mutually exclusive i.e. having one illness does not preclude you from having any other illnesses. Given this insight, is it better to have a sigmoid non-linearity or a softmax non-linearity as your output unit? Why?

Sigmoid. In the softmax case, the presence of one disease would lower the probability of all other diseases. This contradicts our assumption that the diseases are not mutually exclusive.

6) (4 points) You try to figure out a better way to reduce the training and testing time of your model. You perform a run time analysis and observe that the computational bottleneck is in your output unit: the number of target illnesses is too high. Your former TA tells you that each illness in the list of 70,000 illnesses belongs to one of 300 classes (e.g. a

migraine belongs to the neurological disorder class). He shares with you a dictionary which maps each illness to its corresponding class. How can you use this information to reduce the time complexity of your model? Include your forward propagation equations in your answer.

Use the last hidden layer to predict the class of the disease. Take the K top classes and predict the actual disease from the same last hidden layer. In this way you are only ever predicting the relevant subset of the full disease distribution given your hidden layer with minimal loss of accuracy. You are also propagating your errors from both outputs to the same hidden unit. We awarded partial credit to students that said to predict the classes instead of the diseases. This approach would lead to a loss of accuracy.

$$\mathbf{c_l} = sigmoid(\sum_j \mathbf{h_j}(\mathbf{t})\mathbf{w_{lj}}) \tag{15}$$

$$\mathbf{y_c} = sigmoid(\sum_j \mathbf{h_j}(\mathbf{t})\mathbf{v_{cj}}) \tag{16}$$

7) (2 points) Your model now trains (and predicts) several times faster. You achieve a precision score of 72% on positive cases (true positives) and a precision score of 68% on negative cases (true negatives). Confident with your initial results, you decide to make a more complex model. You implement a bidirectional deep recurrent neural network over the chronologically ordered patient note-vectors. Your new results are stellar. Your positive precision is 95% and your negative precision is 92%. You boast to your TA that you have built an AI doctor. You coin the name Dr. AI for your model. Unfortunately, your TA tells you that you have made a mistake. What is the mistake?

Leakage. You are using future information to predict the current disease. We were generous with this question, and awarded full credit to students that argued that a measurement of recall was needed. We awarded partial credit to students that claimed the the network might have been over-fitting.

8) (2 points) Which level(s) of the linguistic hierarchy were you tackling here?

Semantic

4 LSTMs, GRUs, and Recursive Networks

1) Problematic Gradients

(a) (3 points) _____ suffer(s) from the vanishing gradient problem. Circle all that apply and JUSTIFY YOUR ANSWER.

(i) 1-Layer Feed-forward NN (i.e., the NN from problem set 1)

(ii) Very Deep Feed-forward NN

(iii) Recurrent NN

(iv) Word2vec CBOW

(v) Word2vec Skip-Gram

Very Deep Feed-forward NN, Recurrent NN. All of these models are "deep", involving chained matrix multiplication and possibly saturating non-linearities.

Circle True/False for the following and briefly JUSTIFY YOUR ANSWER:

(b) (2 points) (True/False) Adding more hidden layers will solve the vanishing gradient problem for a 2 layer neural network.

False. Making the network deeper will increase the chance of vanishing gradients.

(c) (2 points) (True/False) Adding $L2$-regularization will help with vanishing gradients.

False. This will pull the weights toward 0, which can make vanishing gradients worse.

(d) (2 points) (True/False) Clipping the gradient (cutting off at a threshold) will solve the exploding gradients problem.

True. Self-explanatory if the threshold choice is good. Points were awarded based on the clarity of the answer.

2) Here are the defining equations for a LSTM cell.

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right) \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right) \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right) \\
\widetilde{c_t} &= \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \widetilde{c_t} \\
h_t &= o_t \circ \tanh\left(c_t\right)
\end{aligned}
$$

Recall that $\circ$ denotes element-wise multiplication and that σ denotes the sigmoid function.

Circle True/False for the following and briefly JUSTIFY YOUR ANSWER:

(a) (2 points) (True/False) If x_t is the 0 vector, then $h_t = h_{t-1}$.

False. Try $x_t = 0$, it should be simple to verify that after transformations by Us and non-linearities that $h_t = h_{t-1}$ is false generally.

(b) (2 points) (True/False) If f_t is very small or zero, then error will not be back-propagated to earlier time steps.

False. i_t and $\tilde{c}_t$ depend on h_{t-1}.

(c) (2 points) (True/False) The entries of f_t, i_t, o_t are non-negative.

True. The range of sigmoid is (0,1).

(d) (2 points) (True/False) f_t, i_t, o_t can be viewed as probability distributions. (i.e., their entries are non-negative and their entries sum to 1.)

False. Sigmoid is applied element-wise. The entries of those vectors need not need to sum to 1.

3) (3 points) Here are the defining equations for a GRU cell.

$$
\begin{aligned}
z_t &= \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right) \\
r_t &= \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right) \\
\widetilde{h_t} &= \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right) \\
h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \widetilde{h_t}
\end{aligned}
$$

Recall that $\circ$ denotes element-wise multiplication and that σ denotes the sigmoid function. h_t and x_t are column vectors. Assume that the h_t are of dimension d_h and that the x_t are of dimensions d_x. What are the dimensions of $W^{(z)}$, $U^{(z)}$, $W^{(r)}$, $U^{(r)}$, W, and U? Define clearly which numbers are rows and columns.

$$
\begin{aligned}
W^{(z)} &: (d_h, d_x) \\
U^{(z)} &: (d_h, d_h) \\
W^{(r)} &: (d_h, d_x) \\
U^{(r)} &: (d_h, d_h) \\
W &: (d_h, d_x) \\
U &: (d_h, d_h)
\end{aligned}
$$

4) (3 points) Fig. 3 illustrates a deep recurrent network. Assume that each circle in this model denotes a GRU-cell (as in the previous question). Assume that GRU parameters are shared for each layer in the deep RNN.

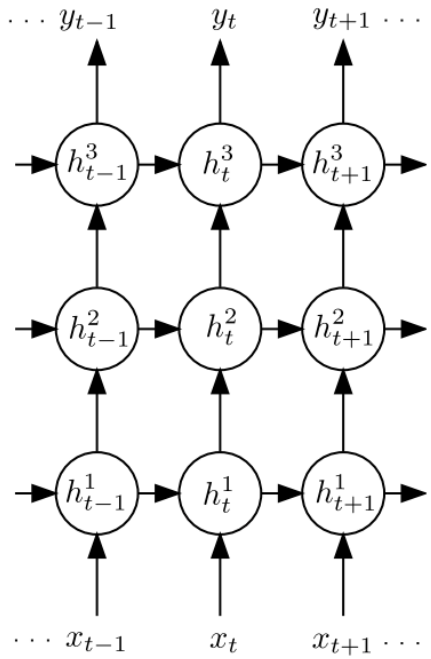


Figure 3: Deep RNN

Let $W_i^{(z)}, U_i^{(z)}, W_i^{(r)}, U_i^{(r)}, W_i, U_i$ denote the parameters for the i-th hidden layer (the h_t^i nodes above). Assume that x_t is of dimension d_x and that the h_t^i are of dimension d_h. What are the dimensions for $W_2^{(z)}$, $U_2^{(z)}$, $W_2^{(r)}$, $U_2^{(r)}$, W_2, and U_2 (note that these should **not all** be the same as those for the previous answer). Hint: the output of a lower level's cell becomes the input to the next higher layer.

$$
\begin{aligned}
W^{(z)} &: (d_h, d_h) \\
U^{(z)} &: (d_h, d_h) \\
W^{(r)} &: (d_h, d_h) \\
U^{(r)} &: (d_h, d_h) \\
W &: (d_h, d_h) \\
U &: (d_h, d_h)
\end{aligned}
$$

5 Hyper-Parameter Tuning

1) (2 points) In our objective functions, we usually regularize the weight parameters of the softmax but not its biases. Explain why.

The purpose of the bias term is help skew the prediction in the direction of the distribution of the classes and thus act as a non-uniform prior. Adding regularization would prevent this.

2) (3 points) In 2-3 short sentences, describe why we initialize the weights in our model to be i)*small* and ii)*random* numbers. Hint: Think of the softmax and tanh nonlinearities.

(i) small: to prevent saturation which would lead to low gradients and slow training

(ii) random: symmetry breaking

3) Your training error and cost are high and your validation cost and error are almost equal to it. Answer the following questions:

(i) (2 points) What does this mean for your model?

Underfitting

(ii) (1 point) What actions would you take?

Decrease regularization, increase model complexity/add parameters

4) (3 points) Explain why we use $\frac{f(x+h)-f(x-h)}{2h}$ and not $\frac{f(x+h)-f(x)}{h}$ to estimate the numerical gradient of f. Remember that for a smooth function, $f : \mathbb{R} \to \mathbb{R}$, its Taylor series expanded around a is $\sum_{n=0}^{\infty} \frac{f^{(n)}(a)\,(x-a)^n}{n!}$

$$\frac{f(x+h)-f(x-h)}{2h} = \frac{2f^{(1)}(x)h + \sum_{n=1}^{\infty} 2\frac{f^{(2n+1)}(x)\,(h)^{2n+1}}{(2n+1)!}}{2h} \tag{17}$$

$$= f^{(1)}(x) + O(h^2) \tag{18}$$

$$\frac{f(x+h)-f(x)}{h} = \frac{f^{(1)}(x)h + \sum_{n=1}^{\infty} \frac{f^{(n)}(x)\,(h)^{n}}{(n)!}}{h} \tag{19}$$

$$= f^{(1)}(x) + O(h) \tag{20}$$