CS224N: Natural Language Processing with Deep Learning Winter 2018 Midterm Exam

This examination consists of 17 printed sides, 5 questions, and 100 points. The exam accounts for 20% of your total grade. Please write your answers on the exam paper in the spaces provided. You may use the 16^{th} page if necessary but **you must make a note on the question's answer box**. You have **80 minutes** to complete the exam. Exams turned in after the end of the examination period will be either penalized or not graded at all. The exam is closed book and allows *only a single page of notes*. You are **not allowed** to: use a phone, laptop/tablet, calculator or spreadsheet, access the internet, communicate with others, or use other programming capabilities. You must disable all networking and radios ("airplane mode").

If you are taking the exam **remotely**, please send us the exam by **Tuesday, February 13 at 5:50 pm PDT** as a scanned PDF copy to `scpd-distribution@lists.stanford.edu`.

Stanford University Honor Code: I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

SUNet ID: ______________________ Signature: ______________________

Name (printed): ______________________________________ ☐ SCPD student

Question	Points	Score
Multiple Choice	18	
Short Questions	32	
Word Vectors	12	
Backpropagation	17	
RNNs	21	
Total:	100	

The standard of academic conduct for Stanford students is as follows:

1. The Honor Code is an undertaking of the students, individually and collectively: a. that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading; b. that they will do their share and take an active part in seeing to it that they as well as others uphold the spirit and letter of the Honor Code.

2. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

3. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

1. Multiple Choice (18 points)

For each of the following questions, color *all the circles* you think are correct. No explanations are required.

(a) (2 points) Which of the following statement about Skip-gram are correct?

- ○ It predicts the center word from the surrounding context words
- √ **The final word vector for a word is the average or sum of the input vector v and output vector u corresponding to that word**
- ○ When it comes to a small corpus, it has better performance than GloVe
- ○ It makes use of global co-occurrence statistics

(b) (2 points) Which of the following statements about dependency trees are correct?

- ○ Each word is connected to exactly one dependent (i.e., each word has exactly one outgoing edge)
- ○ A dependency tree with crossing edges is called a "projective" dependency tree.
- √ **Assuming it parses the sentence correctly, the last transition made by the dependency parser from class and assignment 2 will always be a `RIGHT-ARC` connecting ROOT to some word.**
- ○ None of the above

(c) (2 points) Which of the following statements is true of language models?

- ○ Neural window-based models share weights across the window
- ○ Neural window-based language models suffer from the sparsity problem, but n-gram language models do not
- ○ The number of parameters in an RNN language model grows with the number of time steps
- √ **Neural window-based models can be parallelized, but RNN language models cannot**

Solution: D: Gradients must flow through each time step for RNNs whereas for neural window-based models can perform its forward- and back-propagation in parallel.

(d) (2 points) Assume x and y get multiplied together element-wise $x * y$. S_x and S_y are the shapes of x and y respectively. For which S_x and S_y will Numpy / Tensorflow throw an error?

- ○ $S_x = [1, 10], S_y = [10, 10]$
- ○ $S_x = [10, 1], S_y = [10, 1]$
- √ $S_x = [10, 100], S_y = [100, 10]$
- ○ $S_x = [1, 10, 100], S_y = [1, 1, 1]$

(e) (2 points) Suppose that you are training a neural network for classification, but you notice that the training loss is much lower than the validation loss. Which of the following can be used to address the issue (select all that apply)?

[✓ **Use a network with fewer layers**]
○ Decrease dropout probability
[✓ **Increase L2 regularization weight**]
○ Increase the size of each hidden layer

Solution: A, C – the model is now overfitting, and all of these are valid techniques to address it. However, since dropout is a form of regularization, then decreasing it (B) will have the opposite effect, as will increasing network size (D).

(f) (2 points) Suppose a classifier predicts each possible class with equal probability. If there are 10 classes, what will the cross-entropy error be on a single example?

○ $-\log(10)$
○ $-0.1\log(1)$
[✓ $-\log(0.1)$]
○ $-10\log(0.1)$

Solution: C. Cross-Entropy loss simplifies to negative log of the predicted probability for the correct class. At the start of training, we have approximately uniform probabilities for the 10 classes, or 0.1 for each class. So, that means -log(0.1) is what the loss should approximately be at the start.

(g) (2 points) Suppose we have a loss function $f(\mathbf{x}, \mathbf{y}; \theta)$, defined in terms of parameters θ, inputs $\mathbf{x}$ and labels $\mathbf{y}$. Suppose that, for some special input and label pair ($\mathbf{x_0}$, $\mathbf{y_0}$), the loss equals zero. True or false: it follows that the gradient of the loss with respect to θ is equal to zero.

○ True [✓ **False**]

(h) (2 points) During backpropagation, when the gradient flows backwards through the sigmoid or tanh non-linearities, it cannot change sign.

[✓ **True**] ○ False

Solution: A. True. The local gradient for both of these non-linearites is always positive.

(i) (2 points) Suppose we are training a neural network with stochastic gradient descent on minibatches. True or false: summing the cost across the minibatch is equivalent to averaging the cost across the minibatch, if in the first case we divide the learning rate by the minibatch size.

[✓ **True**] ○ False

Solution: A. True. There is only a constant factor difference between averaging and summing, and you can simply multiply the learning rate to get the same update.

2. Short Questions (32 points)

Please write answers to the following questions in a sentence or two.

(a) Suppose we want to classify movie review text as (1) either positive or negative sentiment, and (2) either action, comedy or romance movie genre. To perform these two related classification tasks, we use a neural network that shares the first layer, but branches into two separate layers to compute the two classifications. The loss is a weighted sum of the two cross-entropy losses.

$$\begin{aligned} \mathbf{h} &= \text{ReLU}(\mathbf{W_0 x} + \mathbf{b_0}) \\ \hat{\mathbf{y}}_1 &= \text{softmax}(\mathbf{W_1 h} + \mathbf{b_1}) \\ \hat{\mathbf{y}}_2 &= \text{softmax}(\mathbf{W_2 h} + \mathbf{b_2}) \\ J &= \alpha \text{CE}(\mathbf{y_1}, \hat{\mathbf{y}}_1) + \beta \text{CE}(\mathbf{y_2}, \hat{\mathbf{y}}_2) \end{aligned}$$

Here input $\mathbf{x} \in \mathbb{R}^{10}$ is some vector encoding of the input text, label $\hat{\mathbf{y}}_1 \in \mathbb{R}^2$ is a one-hot vector encoding the true sentiment, label $\hat{\mathbf{y}}_2 \in \mathbb{R}^3$ is a one-hot vector encoding the true movie genre, $\mathbf{h} \in \mathbb{R}^{10}$ is a hidden layer, $\mathbf{W}_0 \in \mathbb{R}^{10\times10}, \mathbf{W}_1 \in \mathbb{R}^{2\times10}, \mathbf{W}_2 \in \mathbb{R}^{3\times10}$ are weight matrices, and α and β are scalars that balance the two losses.

i. (4 points) To compute backpropagation for this network, we can use the multivariable chain rule. Assuming we already know:

$$\frac{\partial \hat{\mathbf{y}}_1}{\partial \mathbf{x}} = \mathbf{\Delta}_1, \qquad \frac{\partial \hat{\mathbf{y}}_2}{\partial \mathbf{x}} = \mathbf{\Delta}_2, \qquad \frac{\partial J}{\partial \hat{\mathbf{y}}_1} = \delta_3^T, \qquad \frac{\partial J}{\partial \hat{\mathbf{y}}_2} = \delta_4^T$$

What is $\frac{\partial J}{\partial \mathbf{x}}$?

Solution: $\delta_3^T \mathbf{\Delta}_1 + \delta_4^T \mathbf{\Delta}_2$(row convention), or $\mathbf{\Delta}_1 \delta_3^T + \mathbf{\Delta}_2 \delta_4^T$(column convention)

If you are using row convention(numerator-layout notation), the dimensions of parameters in the questions are δ_3^T: 1x2, $\mathbf{\Delta}_1$: 2x10, δ_4^T: 1x3, $\mathbf{\Delta}_2$: 3x10
If you are using column convention(denominator-layout notation), the dimensions of parameters in the questions are δ_3^T: 2x1, $\mathbf{\Delta}_1$: 10x2, δ_4^T: 3x1, $\mathbf{\Delta}_2$: 10x3

ii. (3 points) When we train this model, we find that it underfits the training data. Why might underfitting happen in this case? Provide at least one suggestion to reduce underfitting.

Solution: The model is too simple (just two layers, hidden layers' dimension is only 10, input feature dimension is only 10).
Anything increasing the complexity of the model would be accepted, including:

- Increasing dimensions of the hidden layer

- Adding more layers
- Splitting the model into two (with more overall parameters)

(b) Practical Deep Learning Training

i. (2 points) In assignment 1, we saw that we could use gradient check, which calculates numerical gradients using the central difference formula, as a way to validate the accuracy of our analytical gradients. Why don't we use the numerical gradient to train neural networks in practice?

Solution: Since calculating the complete numerical gradient for a single update requires iterating through all dimensions of all the parameters and computing two forward passes for each iteration, this becomes too expensive to use for training in practice.

ii. (3 points) In class, we learned how the ReLU activation function ($\text{ReLU}(z) = \max(0, z)$ could "die" or become saturated/inactive when the input is negative. A friend of yours suggests the use of another activation function $f(z) = \max(0.2z, 2z)$ which he claims will remove the saturation problem. Would this address the problem? Why or why not?

Solution: Yes, – it addresses the saturation problem of ReLU and also is not a purely linear function. In fact, this is just an instance of the Leaky ReLU non-linearity, frequently used to train neural nets in practice.

iii. (3 points) The same friend also proposes another activation function $g(z) = 1.5z$. Would this be a good activation function? Why or why not?

Solution: No, this is not a good idea – this is no longer a non-linear function, so the neural network boils down to a simple linear predictor.

iv. (4 points) There are several tradeoffs when choosing the batch size for training. Name one advantage for having very large batch sizes during training. Also, name one advantage for having very small batch sizes during training.

Solution: Larger batch sizes give you updates that are closer to the overall gradient on the dataset; small batches give you less exact but more frequent updates.

v. (2 points) Suppose we are training a feed-forward neural network with several hidden layers (all with ReLU non-linearities) and a softmax output. A friend suggests that you should initialize all the weights (including biases) to zeros. Is this a good idea or not? Explain briefly in 1-2 sentences.

Solution: No – all neurons will receive the same update if they all start off with the same initialization due to symmetry. In fact, because everything is zeros to start off with, then there will be no gradient flow to any of the lower layers of the network (since the upstream gradients get multiplied by W_i's and h_i's, which are all 0). (Full credit was also given to students who mentioned saturation/dead ReLUs, or thinking about non-differentiability at 0 for ReLU.)

(c) (4 points) Word2Vec represents a family of embedding algorithms that are commonly used in a variety of contexts. Suppose in a recommender system for online shopping, we have information about co-purchase records for items $x_1, x_2, \ldots, x_n$ (for example, item x_i is commonly bought together with item x_j). Explain how you would use ideas similar to Word2Vec to recommend similar items to users who have shown interest in any one of the items.

Solution: We can treat items that are copurchased with x to be in the 'context' of item x (1 point). We can use those copurchase records to build item embeddings akin to Word2Vec. (2 points, you need to mention that item embeddings are created). Then we can use a similarity metric such as finding the items with the largest cosine similarity to the average basket to determine item recommendations for users (1 point).

(d) In lectures and Assignment 2 you learned about transition-based dependency parsing. Another model for parsing sentences is a *graph-based* dependency parser. It takes as input a sentence $[w_1, w_2, ..., w_T]$. First, it encodes the sentence as a sequence of d-dimensional vectors $[\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T]$ (usually using a bidirectional LSTM, but the particular details don't matter for this question). Next, it assigns a score $s(i, j)$ to each possible dependency $i \to j$ going from word w_i to word w_j (in this question, the model only predicts which edges are in the dependency graph, not the types of the edges). The score is computed as $s(i, j) = \mathbf{h}_i^T \mathbf{A} \mathbf{h}_j$ where $\mathbf{A}$ is a $d \times d$ weight matrix. There is also a score for having an edge going from `ROOT` to a word w_j given as $s(\texttt{ROOT}, j) = \mathbf{w}^T \mathbf{h}_j$ where $\mathbf{w}$ is a d-dimensional vector of weights. Lastly, the model assigns each word j the head whose edge scores the highest: $\text{argmax}_{i \in [1,...,T]}\, s(i, j)$.

i. (3 points) Is there a kind of parse tree this model can produce, but a transition-based dependency parser can't? If so, what is this kind of tree?

Solution: Unlike a transition-based parser, this model can parse sentences with non-projective dependency trees (i.e., with crossing edges).

ii. (2 points) Suppose the model instead scored each edge with a simple dot product: $s(i, j) = \mathbf{h}_i^T \mathbf{h}_j$. Why would this not work as well?

Solution: Any of the following is correct

- The score of an edge $i \to j$ will be the same as the score of an edge $j \to i$, which doesn't make sense.
- $s(i, i)$ will be high, so words may get linked to themselves.
- Similar words will score highly, but often similar words should not be connected in a dependency tree.
- A allows the model to express more complex interactions between the elements of the h vectors.

iii. (2 points) What is one disadvantange of graph-based dependency parsers compared to transition-based dependency parsers?

Solution: Any of the following is correct:

- Graph-based dependency parsers are slower (quadratic time instead of linear in the length of the sentence).
- Graph-based dependency parsers are not constrained to produce valid parse trees, so they may be more likely to produce invalid ones.
- Graph-based dependency parsers make parsing decisions independently so they can't use features based on previous parsing decisions.

3. Word Vectors (12 points)

(a) (3 points) Although pre-trained word vectors work very well in many practical downstream tasks, in some settings it's best to continue to learn (i.e. 'retrain') the word vectors as parameters of our neural network. Explain why retraining the word vectors may hurt our model if our dataset for the specific task is too small.

Solution: See TV/television/telly example from lecture 4. Word vectors in training data move around; word vectors not in training data don't move around. Destroys structure of word vector space. Could also phrase as an overfitting or generalization problem.

(b) (2 points) Give 2 examples of how we can evaluate word vectors. For each example, please indicate whether it is intrinsic or extrinsic.

Solution: Intrinsic: Word Vector Analogies ; Word vector distances and their correlation with human judgments.

Extrinsic: Name entity recognitions: finding a person, location or organization and so on.

(c) (4 points) In lectures, we saw how word vectors can alternatively be learned via co-occurrence count-based methods. How does Word2Vec compare with these methods? Please briefly explain one advantage and one disadvantage of the Word2Vec model.

Solution:

Advantage of word2vec: scale with corpus size; capture complex patterns beyond word similarity

Disadvantage: slower than occurrence count based model; efficient usage of statistics

(d) (3 points) Alice and Bob have each used the Word2Vec algorithm to obtain word embeddings for the same vocabulary of words V. In particular, Alice has obtained 'context' vectors $\mathbf{u}_w^A$ and 'center' vectors $\mathbf{v}_w^A$ for every $w \in V$, and Bob has obtained 'context' vectors $\mathbf{u}_w^B$ and 'center' vectors $\mathbf{v}_w^B$ for every $w \in V$.

Suppose that, for every pair of words $w, w' \in V$, the inner product is the same in both Alice and Bob's model: $(\mathbf{u}_w^A)^T \mathbf{v}_{w'}^A = (\mathbf{u}_w^B)^T \mathbf{v}_{w'}^B$. Does it follow that, for every word $w \in V$, $\mathbf{v}_w^A = \mathbf{v}_w^B$? Why or why not?

Solution: No. Word2Vec model only optimizes for the inner product between word vectors for words in the same context.

One can rotate all word vectors by the same amount and the inner product will still be the same. Alternatively one can scale the set of context vectors by a factor of k and the set of center vectors by a factor of $1/k$. Such transformations preserves inner product, but the set of vectors could be different.

Note that degenerate solutions (all zero vectors etc.) are discouraged.

4. Backpropagation (17 points)

In class, we used the sigmoid function as our **activation function**. The most widely used activation function in deep learning now is **ReLU**: Rectified Linear Unit:

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

In this problem, we'll explore using the ReLU function in a neural network. Throughout this problem, you are allowed (and highly encouraged) to use variables to represent intermediate gradients (i.e. δ_1, δ_2, etc.).

(a) (2 points) Compute the gradient of the ReLU function, i.e. derive $\frac{\partial}{\partial z}\text{ReLU}$.

Hint: You can use the following notation:

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

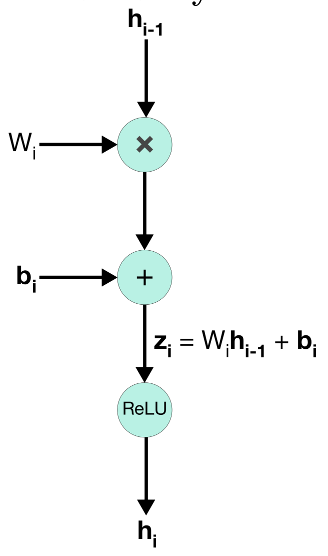
Solution:

$$\frac{\partial}{\partial z}\text{ReLU} = 1\{z > 0\} \tag{1}$$

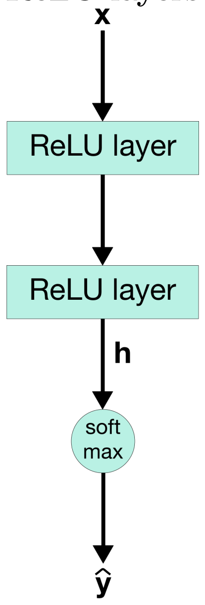
(b) Now, we'll use the ReLU function to construct a multi-layer neural network.

Below, the figure on the left shows a computation graph for a single ReLU hidden layer at layer i. The second figure shows the computation graph for the full neural network we'll use in this problem. The network is specified by the equations below:

ReLU layer i



Neural network with 2 ReLU layers



$$
\begin{aligned}
\mathbf{z_1} &= \mathbf{W}_1\mathbf{x} + \mathbf{b_1} \\
\mathbf{h_1} &= \mathrm{ReLU}(\mathbf{z_1}) \\
\mathbf{z_2} &= \mathbf{W}_2\mathbf{h_1} + \mathbf{b_2} \\
\mathbf{h_2} &= \mathrm{ReLU}(\mathbf{z_2}) \\
\hat{\mathbf{y}} &= \mathrm{softmax}(\mathbf{h_2}) \\
J &= \mathrm{CE}(\mathbf{y}, \hat{\mathbf{y}}) \\
\mathrm{CE}(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_i y_i \log(\hat{y}_i)
\end{aligned}
$$

The dimensions of our parameters and variables are $\mathbf{x} \in \mathbb{R}^{D_x \times 1}$, $\mathbf{W_1} \in \mathbb{R}^{H \times D_x}$, $\mathbf{b_1} \in \mathbb{R}^{H}$, $\mathbf{W_2} \in \mathbb{R}^{D_y \times H}$, $\mathbf{b_2} \in \mathbb{R}^{D_y}$, $\hat{\mathbf{y}} \in \mathbb{R}^{D_y \times 1}$. Note: $\mathbf{x}$ is a single *column* vector.

i. (4 points) Compute the gradients $\frac{\partial J}{\partial \mathbf{W}_2}$ and $\frac{\partial J}{\partial \mathbf{b_2}}$.

Hint: Recall from PA1 that $\frac{\partial J}{\partial \theta} = \hat{\mathbf{y}} - \mathbf{y}$, *where* θ *is the inputs of softmax.*

Solution:

$$
\begin{aligned}
\delta_1 &= \frac{\partial J}{\partial h_2} = \hat{y} - y \\
\delta_2 &= \frac{\partial J}{\partial z_2} = \delta_1 \frac{\partial h_2}{\partial z_2} = \delta_1 \circ 1\{z_2 > 0\} \\
\frac{\partial J}{\partial b_2} &= \delta_2 \\
\frac{\partial J}{\partial W_2} &= \delta_2 h_1^T
\end{aligned}
$$

ii. (4 points) Compute the gradients $\frac{\partial J}{\partial \mathbf{W}_1}$ and $\frac{\partial J}{\partial \mathbf{b}_1}$. You may use gradients you have already derived in the previous part.

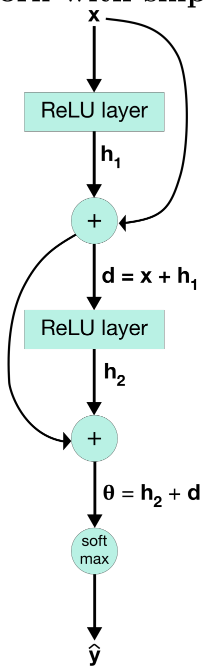
Solution:

$$\delta_3 = \frac{\partial J}{\partial h_1} = \delta_2 \frac{\partial z_2}{\partial h_1} = W_2^T \delta_2$$
$$\delta_4 = \frac{\partial J}{\partial z_1} = \delta_3 \frac{\partial h_1}{\partial z_1} = \delta_3 \circ 1\{z_1 > 0\}$$
$$\frac{\partial J}{\partial b_1} = \delta_4$$
$$\frac{\partial J}{\partial W_1} = \delta_4 x^T$$

(c) (7 points) When neural networks become very deep (i.e. have many layers), they become difficult to train due to the vanishing gradient problem – as the gradient is back-propagated through many layers, repeated multiplication can make the gradient extremely small, so that performance plateaus or even degrades.

An effective approach, particularly in computer vision applications, is ResNet. The core idea of ResNet is **skip connections** that skip one or more layers. See the computation graph below:

Neural network with skip connections



$$\mathbf{z_1} = \mathbf{W}_1\mathbf{x} + \mathbf{b_1}$$
$$\mathbf{h_1} = \mathrm{ReLU}(\mathbf{z_1})$$
$$\mathbf{d} = \mathbf{h_1} + \mathbf{x}$$
$$\mathbf{z_2} = \mathbf{W}_2\mathbf{d} + \mathbf{b_2}$$
$$\mathbf{h_2} = \mathrm{ReLU}(\mathbf{z_2})$$
$$\theta = \mathbf{h_2} + \mathbf{d}$$
$$\hat{\mathbf{y}} = \mathrm{softmax}(\theta)$$
$$J = \mathrm{CE}(\mathbf{y}, \hat{\mathbf{y}})$$

For this part, the dimensions from part B still apply, but assume $D_y = D_x = H$. **Compute the gradient $\frac{\partial J}{\partial \mathbf{x}}$.** Again, you are allowed (and highly encouraged) to use variables to represent intermediate gradients (i.e. δ_1, δ_2, etc.)

*Hint: Use the computational graph to compute the upstream and local gradients for each node. Recall that downstream = upstream * local. Alternatively, compute each of these gradients in order to build up your answer:* $\frac{\partial J}{\partial \theta}, \frac{\partial J}{\partial \mathbf{h_2}}, \frac{\partial J}{\partial \mathbf{z_2}}, \frac{\partial J}{\partial \mathbf{d}}, \frac{\partial J}{\partial \mathbf{x}}$. *Show your work so we are able to give partial credit!*

Please write your answer in the box provided on the next page.

Solution:

$$\begin{aligned}
\delta_1 &= \frac{\partial J}{\partial \theta} = \hat{y} - y \\
\delta_2 &= \frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial \theta}\frac{\partial \theta}{\partial h_2}\frac{\partial h_2}{\partial z_2} = \delta_1 \circ 1\{z_2 > 0\} \\
\delta_3 &= \frac{\partial J}{\partial d} = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial d} + \frac{\partial J}{\partial \theta}\frac{\partial \theta}{\partial d} = W_2^T \delta_2 + \delta_1 \\
\frac{\partial J}{\partial x} &= \frac{\partial J}{\partial d}\frac{\partial d}{\partial h_1}\frac{\partial h_1}{\partial x} + \frac{\partial J}{\partial d}\frac{\partial d}{\partial x} = W_1^T(\delta_3 \circ 1\{z_1 > 0\}) + \delta_3
\end{aligned}$$

5. RNNs (21 points)

RNNs are versatile! In class, we learned that this family of neural networks have many important advantages and can be used in a variety of tasks. They are commonly used in many state-of-the-art architectures for NLP.

(a) For each of the following tasks, state how you would run an RNN to do that task. In particular, specify how the RNN would be used at **test** time (not training time), and specify

1. how many outputs i.e. number of times the softmax $\hat{\mathbf{y}}^{(t)}$ is called from your RNN. If the number of outputs is not fixed, state it as *arbitrary*.
2. what each $\hat{\mathbf{y}}^{(t)}$ is a probability distribution over (e.g. distributed over all species of cats)
3. which inputs are fed at each time step to produce each output

The inputs are specified below.

i. (3 points) Named-Entity Recognition: For each word in a sentence, classify that word as either a person, organization, location, or none.
Inputs: A sentence containing n words.

Solution: Number of Outputs: n outputs, one per input word at each time step.

Each $\hat{\mathbf{y}}^{(t)}$ is a probability distribution over 4 NER categories.

Each word in the sentence is fed into the RNN and one output is produced at every time step corresponding to the predicted tag/category for each word.

ii. (3 points) Sentiment Analysis: Classify the sentiment of a sentence ranging from negative to positive (integer values from 0 to 4).
Inputs: A sentence containing n words.

Solution: Number of Outputs: 1 output. n outputs is also acceptable if they say for instance to take average of all outputs.

Each $\hat{\mathbf{y}}^{(t)}$ is a probability distribution over 5 sentiment values.

Each word in the sentence is fed into the RNN and one output is produced from the hidden states (by either taking only the final, max, or mean across all states) corresponding to the sentiment value of the sentence.

iii. (3 points) Language models: generating text from a chatbot that was trained to speak like you by predicting the next word in the sequence.
Input: A single start word or token that is fed into the first time step of the RNN.

Solution: Number of Outputs: arbitrary

Each $\hat{\mathbf{y}}^{(t)}$ is a probability distribution over the vocabulary.

The previous output is fed as input for the next time step and produces the next output corresponding to the next predicted word of the generated sentence.
As a detail, the first input can also be a designated $< START >$ token and the first output would be the first word of the sentence.

(b) You build a sentiment analysis system that feeds a sentence into a RNN, and then computes the sentiment class between 0 (very negative) and 4 (very positive), based only on the **final** hidden state of the RNN.

i. (2 points) What is one advantage that an RNN would have over a neural window-based model for this task?

Solution: There are multiple answers: We can process arbitrary length inputs. It can encode temporal information.('Take ordering into consideration' is only partial correct, because theoretically window-based model also can, although hard to). Shared weights. Less parameters. The number of parameters would increase proportional to the input size of the neural window-based network whereas it would stay constant for RNNs since weights are shared at every time-step.

ii. (2 points) You observe that your model predicts very positive sentiment for the following passage:

```
Yesterday turned out to be a terrible day.
I overslept my alarm clock, and to make matters worse,
my dog ate my homework. At least my dog seems happy...
```

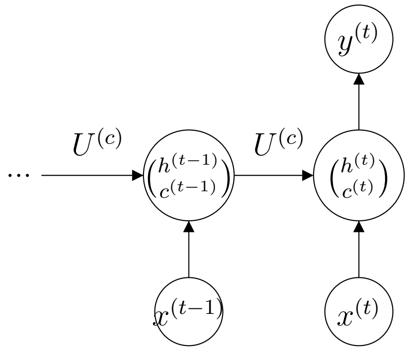
Why might the model misclassify the appropriate sentiment for this sentence?

Solution: The final word in the sentence is 'happy' which has very positive sentiment. Since we only use the final hidden state to compute the output, the final word would have too much impact in the classification. In addition, because the sentence is quite long, information from earlier time steps may not survive due to the vanishing gradient problem.

iii. (4 points) Your friend suggests using an LSTM instead. Recall the units of an LSTM cell are defined as

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

where the final output of the last lstm cell is defined by $\hat{y}_t = \text{softmax}(h_t W + b)$. The final cost function J uses the cross-entropy loss. Consider an LSTM for two time steps, t and $t-1$.



Derive the gradient $\frac{\partial J}{\partial U^{(c)}}$ in terms of the following gradients: $\frac{\partial h_t}{\partial h_{t-1}}$, $\frac{\partial h_{t-1}}{\partial U^{(c)}}$, $\frac{\partial J}{\partial h_t}$, $\frac{\partial c_t}{\partial U^{(c)}}$, $\frac{\partial c_{t-1}}{\partial U^{(c)}}$, $\frac{\partial c_t}{\partial c_{t-1}}$, $\frac{\partial h_t}{\partial c_t}$, and $\frac{\partial h_t}{\partial o_t}$. *Not all of the gradients may be used.* You can leave the answer in the form of chain rule and do not have to calculate any individual gradients in your final result.

Solution:

$$\frac{\partial J}{\partial U^{(c)}} = \sum_{i=t-1}^{t} \left.\frac{\partial J}{\partial U^{(c)}}\right|_i$$

$$\frac{\partial J}{\partial U^{(c)}} = \frac{\partial J}{\partial h_t}\left(\frac{\partial h_t}{\partial c_t}\left(\frac{\partial c_t}{\partial U^{(c)}} + \frac{\partial c_t}{\partial c_{t-1}}\frac{\partial c_{t-1}}{\partial U^{(c)}}\right) + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial U^{(c)}}\right)$$

Because $\frac{\partial c_t}{\partial U^{(c)}}$ is ambiguous, the following solution is also accepted:

$$\frac{\partial J}{\partial U^{(c)}} = \sum_{i=t-1}^{t} \left.\frac{\partial J}{\partial U^{(c)}}\right|_i$$

$$\frac{\partial J}{\partial U^{(c)}} = \frac{\partial J}{\partial h_t}\left(\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial U^{(c)}} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial U^{(c)}}\right)$$

We must consider the gradient going through the current hidden state h_t to the memory cells, c_t, c_{t-1}, and to the previous hidden state h_{t-1}

iv. (2 points) Which part of the gradient $\frac{\partial J}{\partial U^{(c)}}$ allows LSTMs to mitigate the effect of the vanishing gradient problem? Explain in two sentences or less how this would help classify the correct sentiment for the sentence in part b).

Solution: $\frac{\partial c_t}{\partial c_{t-1}}$. Since $\frac{\partial c_t}{\partial c_{t-1}} = f_t$, the forget gate, which can act as the identify function when $f_t = 1$, this allows the gradients entering the current cell to pass to the previous cell. This would help the model take into consideration words that appear many time steps away. From the sentences in part b), it would take more into account the negative sentiment at the beginning.

v. (2 points) Rather than using the last hidden state to output the sentiment of a sentence, what could be a better solution to improve the performance of the sentiment analysis task?

Solution: We can take advantage of every cell by taking a max pool/average/sum of all hidden states.