CS224N: Natural Language Processing with Deep Learning Winter 2017 Midterm Exam

This examination consists of 14 printed sides, 5 questions, and 100 points. The exam accounts for 17% of your total grade. Please write your answers on the exam paper in the spaces provided. You may use the back of a page if necessary but **please mark this**. 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 14 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): ____

Question	Points	Score
Multiple choice	20	
Short questions	24	
Feedforward neural network language models	18	
Autoencoders	18	
Recurrent neural networks	20	
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 (20 points)

For each of the following questions, circle the letter of your choice. *There is only ONE correct choice.* No explanations are required.

(a) (2 points) If your training loss increases with number of epochs, which of the following could be a possible issue with the learning process?

A. Regularization is too low and model is overfitting

B. Regularization is too high and model is underfitting

C. Step size is too large

D. Step size is too small

Solution: C. The train loss always decreases whether the model is overfitting or underfitting. If the step size is too small, the convergence is too slow, but the training loss will still go down. If the step size is too large, it may cause a bouncing effect because we skip and overshoot the optimal solution. This leads to increase in training loss and decrease in training accuracy.

(b) (2 points) In which of the following cases will there be no update when doing SGD with a max margin loss, $J = \max(0, 1 - s + s_c)$ where s is the true window score and s_c is the corrupt window score?

A. $(s, s_c) = (1, 0.5)$

B. $(s, s_c) = (2, 0.5)$

C. $(s, s_c) = (1, 1.5)$

D. $(s, s_c) = (2, 1.5)$

Solution: B. The max margin loss is 0 when $s - s_c > 1$, which is only true for option B.

(c) (2 points) The biggest advantage of neural transition-based dependency parsers over non-neural transition-based dependency parsers is that neural parsers:

A. Choose transitions using more words in the stack and buffer

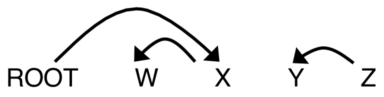
B. Generate a larger class of dependency parses

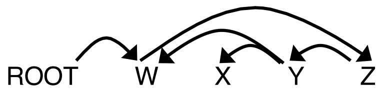
C. Model a grammar whereas traditional parsers do not

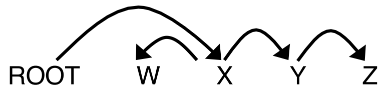
D. Rely on dense feature representations

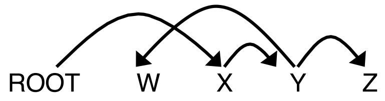
Solution: D. The main advantage of neural dependency parsers is that they offer a dense representation instead of a spare representation of the parser. Neural and traditional parsers are not different in what input information they can use, or what kinds of parses they can output (both can output any parse), but they differ in their representation of the features they use.

(d) (2 points) Which one of the following is a valid projective dependency parse?

(A)


(B)


(C)


(D)


Solution: C. In (A) Z has no head, in (B) W has two heads, and (D) is not projective because ROOT$\rightarrow$X and Y$\rightarrow$W cross.

(e) (2 points) We have learned that dense word vectors learned through word2vec or GloVe have many advantages over using sparse one-hot word vectors. Which of the following is NOT an advantage dense vectors have over sparse vectors?

A. Models using dense word vectors generalize better to unseen words than those using sparse vectors.

B. Models using dense word vectors generalize better to rare words than those using sparse vectors.

C. Dense word vectors encode similarity between words while sparse vectors do not.

D. Dense word vectors are easier to include as features in machine learning systems than sparse vectors.

Solution: A. Just like sparse representations, word2vec or GloVe do not have representations for unseen words and hence do not help in generalization.

(f) (2 points) Which of the following statements is INCORRECT?

A. Recurrent neural networks can handle a sequence of arbitrary length, while feedforward neural networks can not.

B. Training recurrent neural networks is hard because of vanishing and exploding gradient problems.

C. Gradient clipping is an effective way of solving vanishing gradient problem.

D. Gated recurrent units (GRUs) have fewer parameters than LSTMs.

Solution: C. Gradient clipping is only a solution for solving exploding gradient problems, not vanishing gradient ones.

(g) (2 points) In order for the following Tensorflow code to run, which of the following MUST be contained as keys of `feed_dict`?

```
import tensorflow as tf
W = tf.Variable(tf.random_normal((200, 20)))
b = tf.Variable(tf.zeros((20,)))
x = tf.placeholder(tf.float32, (32, 200))
y = tf.matmul(x, W) + b
prediction = tf.nn.softmax(y)

label = tf.placeholder(tf.float32, (32, 20))
cross_entropy = tf.reduce_mean(-tf.reduce_sum(label * tf.log(
    prediction), reduction_indices=[1]))
train_op = tf.train.GradientDescentOptimizer(0.5).minimize(
    cross_entropy)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
feed_dict = ____________________________
sess.run([prediction], feed_dict=feed_dict)
```

A. `x`, `W`, `b`
B. only `x`
C. only `prediction`
D. `x` and `label`

Solution: B. Only the placeholder `x` is required to compute `prediction`. `label` is only required to compute `cross_entropy` and `train_op`. `W`, `b` and `prediction` are not placeholders.

(h) (2 points) A multi-layer neural network model trained using stochastic gradient descent on the same dataset with different initializations for its parameters is guaranteed to learn the same parameters. A. True B. False

Solution: False. The loss function for a multi-layer neural network is non-convex and hence SGD is only guaranteed to converge to a local optimum.

(i) (2 points) If it parses a sentence correctly, a transition-based dependency parser must use a `RIGHT-ARC` transition at least once. A. True B. False

Solution: True. A `RIGHT-ARC` must be used to link ROOT to a dependent.

(j) (2 points) Stochastic gradient descent results in a smoother convergence plot (loss vs epochs) as compared to batch gradient descent. A. True B. False

Solution: False. SGD results in noisier convergence plots compared to batch gradient descent.

2. Short questions (24 points)

Please write answers to the following questions in a sentence or two. Each part is worth 6 points.

(a) Dependency Parsing

i. (2 points) Can the neural dependency parser we learned in class correctly parse the sentence "John saw a dog yesterday which was a Yorkshire terrier."? Explain.

Solution: No. The sentence has a non-projective parse: yesterday is a dependent of saw, while the phrase "which was a Yorkshire terrier" is a dependent of "dog".

ii. (2 points) What is the ambiguity in parsing the sentence "There's an awful cost to getting a PhD that no one talks about"?

Solution: The phrase "that no one talks about" could either be referring to the cost, "No one talks about the awful cost of getting a PhD" or to the PhD, "Getting a PhD that no one talks about is awfully costly".

iii. (2 points) Name at least two types of features that would be helpful to provide as input to a neural dependency parser and explain why.

Solution: Here are some useful features that are indicative of transition decisions:

- Taking word vectors for n_s words from the top of the stack. Helps identify
- Taking word vectors for n_b words from the top of the buffer.
- Taking vector representations for words' POS tags. Helps disambiguate the role of the word or word sense.
- Taking vector representations for words' arc-labels.

(b) (6 points) Adagrad is a modification to the stochastic gradient descent algorithm used to update parameters. Updates are performed using the following formula:

$$\begin{aligned}\text{cache}_i &= \text{cache}_i + (\nabla_{\theta_i} L)^2 \\ \theta_i &= \theta_i - \eta \nabla_{\theta_i} L / (\sqrt{\text{cache}_i} + \epsilon)\end{aligned}$$

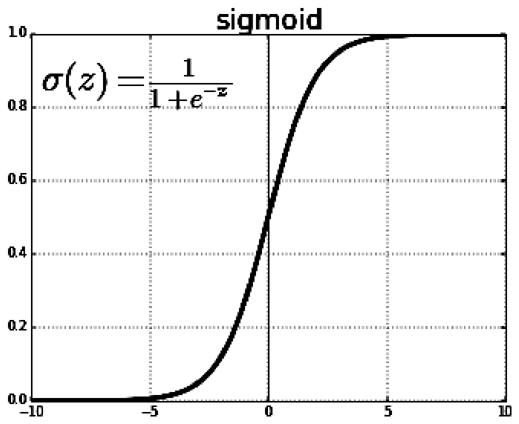
where θ_i is the parameter to be updated, $\nabla_{\theta_i} L$ is the gradient of the loss (cost) with respect to θ_i, ϵ is a hyperparameter usually between 10^{-8} and 10^{-4} and η is a hyperparameter similar to step size in SGD. cache$_i$ is initialized to zero at the start of the algorithm.

(i) State two differences between AdaGrad and SGD in terms of step size and (ii) describe what effect they would have.

Solution:

- Adagrad has a decreasing step-size. This allows parameters to change quickly at first and then converge.
- Adagrad has different per-element stepsizes. This allows parameters with small gradients to change more quickly than those with large step-sizes.

(c) Consider the sigmoid activation function below:



i. (2 points) What would the gradient of the sigmoid be with respect to an input that is very large?

Solution: Because the sigmoid is almost flat for very large values, its gradient will be almost 0.

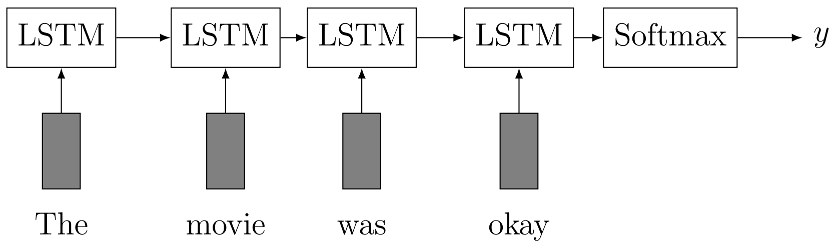
ii. (2 points) Why might this be a problem for training neural networks?

Solution: If weights become saturated during a particular training epoch, the small gradients make it very hard to change weights thereafter: in other words, it makes training much much slower.

iii. (2 points) How does the ReLU activation $\text{ReLU}(z) = \max(0, z)$ solve this problem?

Solution: Because ReLU activations are linear, they do not saturate for large (positive) values, and hence freely allow gradients to change weights in the network. However, they do not solve the problem for large negative weights and ReLU units can "die" if a gradient update moves parameters into the flat region.

(d) A popular model used for sentiment classification is an LSTM model:



This model inputs word vectors to the LSTM model at each time step and uses the last hidden state vector to predict the sentiment label (y).

Recall that in assignment 1 we used a simple "bag-of-vectors" model for sentiment classification: we used the average of all the word vectors in a sentence to predict the sentiment label.

i. (3 points) Name at least one benefit of the LSTM model over the bag-of-vectors model.

Solution: The LSTM model is able to integrate information from word ordering, e.g. "this was not an amazing fantastic movie" while the bag-of-vectors model can not.

ii. (3 points) If we chose to update our word vectors when training the LSTM model on sentiment classification data, how would these word vectors differ from ones not updated during training? Explain with an example. Assume that the word vectors of the LSTM model were initialized using GloVe or word2vec.

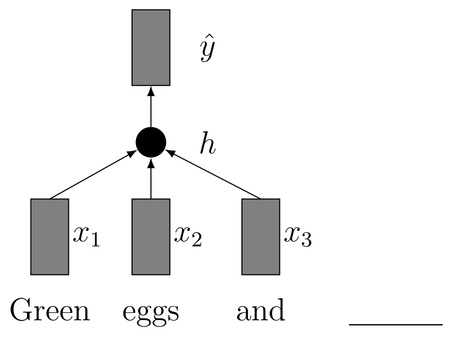
Solution: The word vectors learned using this method can capture more sentiment information. In the word2vec models that depend on co-occurrence counts, words like 'good' or 'bad' have very similar representations; representations learned through a sentiment classifier would learn different embeddings.

3. Feedforward neural network language models (18 points)

A feedforward neural network language model (NNLM) can be used as another architecture for training word vectors. This model tries to predict a word given the N words that precede it. To do so, we concatenate the word vectors of N previous words and send them through a single hidden layer of size H with a tanh nonlinearity and use a softmax layer to make a prediction of the current word. The size of the vocabulary is V. The model is trained using a cross entropy loss for the current word.

Let the word vectors of the N previous words be $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, each a column vector of dimension D, and let $\mathbf{y}$ be the one-hot vector for the current word. The network is specified by the equations below:

The dimensions of our parameters and variables are $\mathbf{x} \in \mathbb{R}^{(N \cdot D)}, W \in \mathbb{R}^{H \times (N \cdot D)}, \mathbf{b} \in \mathbb{R}^{H}, \mathbf{h} \in \mathbb{R}^{H}, U \in \mathbb{R}^{V \times H}, \mathbf{d} \in \mathbb{R}^{V}, \hat{\mathbf{y}} \in \mathbb{R}^{V}$.



$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}$$

$$\mathbf{h} = \tanh(W\mathbf{x} + \mathbf{b})$$
$$\hat{\mathbf{y}} = \text{softmax}(U\mathbf{h} + \mathbf{d})$$
$$J = \text{CE}(\mathbf{y}, \hat{\mathbf{y}})$$
$$CE = -\sum_i y_i \log(\hat{y}_i).$$

(a) (4 points) State two important differences between NNLM the CBOW model we learned in class. Explain how each might affect the word vectors learned.

Solution:

- The CBOW is trained to predict a center word given a context window that extends on both sides, while word vectors learned by NNLM do not capture the context to the right of the word. Thus, the NNLM may not differentiate "word (carefully)" from "word (puzzle)".
- The CBOW model simply uses the sum of context words, while the NNLM model combines context words non-linearly. Thus the NNLM can learn to treat "not good to" differently from "good to not", etc.

(b) i. (3 points) What is the complexity of forward propagation in an NNLM for a *single* training example?

Solution: The forward propagation complexity for an NNLM is $N \times D$ for concatenating the word vectors, $N \times D \times H$ to compute h and $H \times V$ to compute $\hat{y}$ from h: in total, $O(NDH + HV)$. Typically, $V \gg ND$, so the latter term dominates the forward propagation computation.

ii. (3 points) Describe at least one way to change the model that would reduce this complexity.

Solution: The complexity can be reduced by using negative sampling to compute the softmax or using the hierarchical softmax.

(c) (8 points) What is the gradient of J with respect to $\mathbf{x}$, $\frac{\partial J}{\partial \mathbf{x}}$? Hint: $\tanh(z)' = 1 - \tanh(z)^2$.

Solution: Let $z_1 = \mathbf{x}W + b$ and $z_2 = hU + d$.

$$\begin{aligned}
\delta_1 &= \frac{\partial CE}{\partial \hat{y}} = (\hat{y} - y) \\
\delta_2 &= \frac{\partial CE}{\partial h} = \delta_1 \frac{\partial z_2}{\partial h} = U^T \delta_1 \\
\delta_3 &= \frac{\partial CE}{\partial z_1} = \delta_2 \frac{\partial h}{\partial z_1} = \delta_2 \circ (1 - \tanh(z_1)^2) \\
\frac{\partial CE}{\partial \mathbf{x}} &= \delta_3 \frac{\partial z_1}{\partial \mathbf{x}} = W^T \delta_3
\end{aligned}$$

4. Autoencoders (18 points)

In class, we've learned that one way to combine word vectors is to simply add them together. In this question, we'll explore another approach: using an autoencoder.

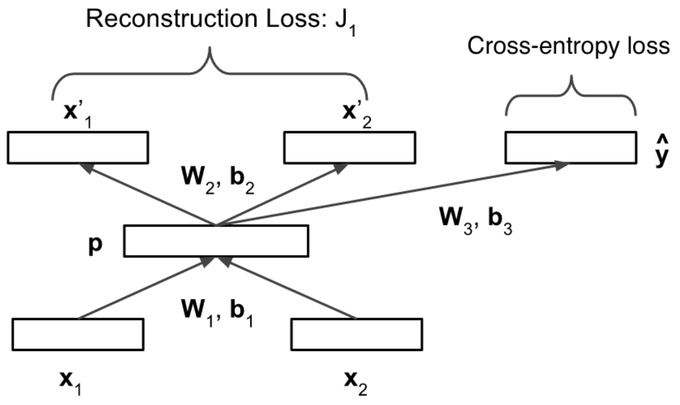


Figure 2: Illustration of an autoencoder unit to combine two vectors

In the autoencoder, two input word vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{D_x \times 1}$ are first concatenated into a single vector $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \in \mathbb{R}^{2D_x \times 1}$, and the parent vector $\mathbf{p}$ can be computed as:

$$\mathbf{p} = \mathrm{ReLU}(W_1 \mathbf{x} + \mathbf{b}_1) \;\in \mathbb{R}^{D_p \times 1} \qquad\qquad \mathrm{ReLU}(x) = \max(0, x),$$

where W_1 can be decomposed as:

$$W_1 = \begin{bmatrix} W_{11} & W_{12} \end{bmatrix} \qquad\qquad W_1 \mathbf{x} = W_{11}\mathbf{x}_1 + W_{12}\mathbf{x}_2.$$

During training, we use the parent vector $\mathbf{p}$ to *reconstruct the input vectors*:

$$\mathbf{x}' = \begin{bmatrix} \mathbf{x}'_1 \\ \mathbf{x}'_2 \end{bmatrix} = W_2 \mathbf{p} + \mathbf{b}_2 \;\in \mathbb{R}^{2D_x \times 1}$$

where $\mathbf{x}'_1, \mathbf{x}'_2 \in \mathbb{R}^{D_x \times 1}$ are the reconstructions. Correspondingly, a *reconstruction loss* that computes the Euclidean distance between inputs and reconstructions is used during training:

$$J_1 = \frac{1}{2}||\mathbf{x}' - \mathbf{x}||^2 \quad \in \mathbb{R}.$$

For sentiment classification, the parent vector is used to predict a sentiment class $\hat{\mathbf{y}}$ for the pair of the input words:

$$\hat{\mathbf{y}} = W_3\mathbf{p} + \mathbf{b}_3 \quad \in \mathbb{R}^{D_c \times 1},$$

where $D_c = 3$ is the number of sentiment classes ('positive', 'neutral' and 'negative'). Here, the network is also trained using a cross-entropy loss:

$$J_2 = CE(\mathbf{y}, \hat{\mathbf{y}}) \quad \in \mathbb{R},$$

where $\mathbf{y} \in \mathbb{R}^{D_c \times 1}$ is the one-hot label vector with $\mathbf{y}_k = 1$.

In total, the network is trained using the loss $J = J_1 + J_2$.

(a) (2 points) Give at least one example of how the sentiment predictions made by an autoencoder model would differ from one made using a bag-of-vectors model like the one in assignment 1.

Solution: The autoencoder model described above allows us to predict that "not bad" is actually neutral or positive, while the bag-of-vectors model would predict a more negative class.

(b) (2 points) How do the reconstructed vectors and reconstruction loss help in learning the parent representation?

Solution: The reconstruction loss prevents the network from ignoring the contribution of one of x_1 or x_2. In other words, it forces the network to capture the joint 'meaning' of x_1 and x_2.

(c) (2 points) i. What is the shape of each weight and bias term: $W_1, W_2, W_3, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$?

ii. How many parameters does the model have in total?

Solution: Shapes of all terms:

$$\begin{aligned} W_1 &\in \mathbb{R}^{D_p \times 2D_x} & b_1 &\in \mathbb{R}^{D_p \times 1} \\ W_2 &\in \mathbb{R}^{2D_x \times D_p} & b_2 &\in \mathbb{R}^{2D_x \times 1} \\ W_3 &\in \mathbb{R}^{D_c \times D_p} & b_3 &\in \mathbb{R}^{D_c \times 1} \end{aligned}$$

Number of parameters $= 4D_xD_p + D_p + 2D_x + D_pD_c + D_c$

(d) Compute the following gradients for the *reconstruction loss*:
Hint: You can use the following notation

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

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

$$\mathbb{1}\left\{\begin{bmatrix} 1 & 0 \\ -1 & 3 \end{bmatrix} > 0\right\} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

i. (2 points) $\delta_1 = \frac{\partial J_1}{\partial \mathbf{p}} =$ ____________

ii. (2 points) $\delta_2 = \frac{\partial J_1}{\partial \mathbf{h}} =$ ____________ (where $\mathbf{h} \overset{\text{def}}{=} W_1\mathbf{x} + \mathbf{b}_1$)

iii. (2 points) $\frac{\partial J_1}{\partial W_1} =$ ____________

iv. (2 points) $\frac{\partial J_1}{\partial \mathbf{b}_1} =$ ____________

v. (2 points) $\frac{\partial J_1}{\partial \mathbf{x}_1} =$ ____________

You can use the space below as scratch space.

Solution: Let $\mathbf{h} = \mathbf{x}W_1 + \mathbf{b}_1$. $\circ$ denotes element-wise multiplication.

$$J_1 = \frac{1}{2}||\mathbf{x}' - \mathbf{x}||^2$$

$$\delta_1 = \frac{\partial J_1}{\partial p} = W_2^T(\mathbf{x}' - \mathbf{x})$$

$$\delta_2 = \frac{\partial J_1}{\partial h} = \delta_1 \circ \mathbb{1}\{\mathbf{h} > 0\}$$

$$\frac{\partial J_1}{\partial W_1} = \delta_2 \mathbf{x}^T, \quad \frac{\partial J_1}{\partial \mathbf{b}_1} = \delta_2, \quad \frac{\partial J_1}{\partial \mathbf{x}_1} = (W_1^T \delta_2)_{[1:D_x]} = W_{11}^T \delta_2$$

(e) (2 points) How would you construct a network using *only* copies of the autoencoder above to predict a sentiment label for a whole sentence, not just a pair of words? Remember that each sentence could be of arbitrary length.

Solution: We can combine the vectors recursively by combining every subsequent pair of words, and then combining each pair of parent vectors, etc. We can also combine the vectors linearly by combining the first two words to get a parent vector, followed by combining every subsequent word with the parent vector.

We considered answers like averaging each parent vector to be incorrect because this adds mean-pooling layer to the network. However, we considered answers like averaging the *predictions* made on every pair of words to still be correct (though not preferable) because such a network does not use any other element than the autoencoders themselves.

5. Recurrent neural networks (20 points)

Recall that a recurrent neural network (RNN) takes in an input vector x_t and a state vector h_{t-1} and returns a new state vector h_t and an output vector y_t:

$$h_t = f(w_1 x_t + w_2 h_{t-1} + b_2)$$
$$y_t = g(w_3 h_t + b_3),$$

where f and g are activations functions.

(a) (8 points) The following diagram depicts a single RNN unit, where x_t, h_{t-1}, h_t and y_t are all scalars, as a state machine:

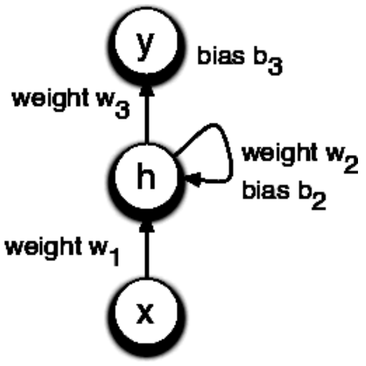
$w_1 =$ ________

$w_2 =$ ________

$b_2 =$ ________

$w_3 =$ ________

$b_3 =$ ________



Suppose that f is a binary threshold unit and g is a linear unit:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$
$$g(x) = x.$$

Fill in weights (w_1, w_2, w_3), biases (b_1, b_2) so that the RNN initially outputs 0, but as soon as it receives an input of 1, it switches to outputting 1 for all subsequent time steps. For instance, the input 0001010 produces the output 0001111. The hidden unit has an initial value of 0.

You don't need to provide an explanation, but doing so may help you receive partial credit.

Hint: In one possible solution, the hidden unit has an activation $h_t = 0$ until there's an input $x_t = 1$, at which point it switches to maintaining an activation of 1 forever. The output unit always predicts the same as the hidden unit, i.e. $y_t = h_t$.

Solution: There are many possible solutions, but only one main scenario: the output unit must predict the same as the hidden state.

This requires that:

$$\begin{aligned} b_3 &= 0 \\ w_3 + b_3 &= 1 \\ b_2 &< 0 \\ w_2 + b_2 &\geq 0 \\ w_1 + b_2 &\geq 0, \end{aligned}$$

so values like $w_1 = 1$, $w_2 = 1$, $b_2 = -1$, $w_3 = 1$, $b_3 = 0$ would be considered correct.

Unfortunately, the original hint we provided was wrong given that the initial value of the hidden state is 0: there is no value of h_t for which the unit can flip from 0 to 1 at the start, when the input is 0 AND flip from 1 to 0 when input is 1 and stay there.

When grading this question, we have considered parameters that agree with the hint, i.e. $w_3 = -1$, $b_3 = 1$, to also be correct. The remaining parameters were partially marked if (a) you stated that $h_0 = 1$ and had consistent parameters:

$$\begin{aligned} b_3 &= 1 \\ w_3 + b_3 &= 0 \\ b_2 &< 0 \\ w_2 + b_2 &\geq 0 \\ w_1 + w_2 + b_2 &< 0, \end{aligned}$$

OR (b) reasoned that no values of w_1, w_2, b_2 are possible.

(b) Now suppose that f is a sigmoid unit and that the input is always 0. The following diagram shows how the next state h_{t+1} changes with the current state h_t when $w_2 = 3$ and $b_2 = -1$, i.e. $h_{t+1} = \sigma(3h_t - 1)$. The diagram also shows that $h_{t+1} = h_t$ when $h_t = 0.80$.

i. (2 points) Would h_{t+1} be greater than or less than h_t if $h_t = 0.5$? **greater than**

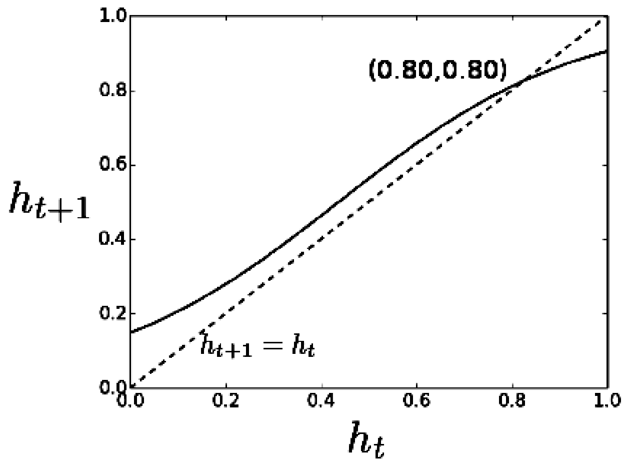
ii. (2 points) Would h_{t+1} be greater than or less than h_t if $h_t = 0.9$? **less than**

iii. (2 points) What would h_{t+1} look like as we take increasingly longer sequences, i.e. $t \to \infty$?

Solution: h_{t+1} would converge to 0.80 as $t \to \infty$.

iv. (1 point) For what range of initial values h_0, would this network experience exploding gradients for long sequences? **None**

v. (1 point) For what range of initial values h_0, would this network experience



vanishing gradients for long sequences? **All**

Solution: The plot above shows how h_t is mapped to h_{t+1}. Whenever $h_t < 0.8$, the graph shows that h_{t+1} is greater than h_t. Similarly, whenever $h_t > 0.8$, the graph shows that h_{t+1} is less than h_t. This shows that as the $t \to \infty$, h_t will converge to h_{t+1}.
In the long term, the gradients experienced by the network, will converge to the gradients at $h_t = 0.8$, which is clearly less than 1 (because it cuts the line $h_t = h_{t+1}$ with a slope less than 1). At each time step, the networks' gradients will be multiplied by this value and hence the network will experience vanishing gradients irrespective of where it started from.

vi. (4 points) We have learned in class that ReLUs are good at preserving gradients in deep neural networks. Would it be wise to replace the sigmoid activation above with a ReLU activation unit? Explain.

Solution: No. When $h_t > \frac{1}{3}$, then ReLU unit has a slope of 3 and would experience exploding gradients. When $h_t < \frac{1}{3}$, the network would experience vanishing gradients.