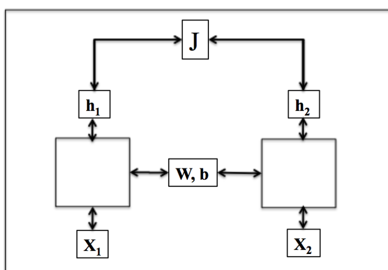
CS 224N, Winter 2017
Practice Midterm #1 Solutions

Note: This practice midterm contains a subset of questions from the CS224D midterm given in 2015, since some topics were not covered in CS224N.

---

1 Siamese Nets

Siamese neural nets have an interesting architecture– the same parameters and functions are used to evaluate 2 inputs. As one might expect, Siamese nets are useful to train *similarity metrics*, evaluations of how "close" inputs are. These nets have been applied to facial recognition tasks with a good deal of success, but in this example, we'll see how to train Siamese nets to learn a distance metric for word vectors. One might imagine training a net to map word vectors across languages, discover synonyms or antonyms, etc.



Here is one such model to evaluate how similar two input words are using Euclidean distance. There are two input word vectors x_1, $x_2 \in \mathbb{R}^n$, shared parameters $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, and a single hidden layer associated with *each* input:

$$h_1 = \sigma(Wx_1 + b)$$

$$h_2 = \sigma(Wx_2 + b)$$

We evaluate the distance between the two activations h_1, h_2 using Euclidean distance as our similarity metric. The model objective J is

$$J = \frac{1}{2}\|h_1 - h_2\|_F^2 + \frac{\lambda}{2}\|W\|_F^2$$

where λ is a given regularization parameter. (The Frobenius norm $\|.\|_F$ is a matrix norm defined by $\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} |A_{ij}|^2}$)

1) (7 points) Calculate $\nabla_W J$ and $\nabla_b J$.

Take the arguments of the functions for h_1 and h_2 to be z_1 and z_2, respectively.

$$\delta_1 = \nabla_{h_1} J = (h_1 - h_2)$$

$$\delta_2 = \nabla_{h_2} J = -(h_1 - h_2)$$

$$\delta_3 = \nabla_{z_1} J = \delta_1 \circ h_1 \circ (1 - h_1)$$

$$\delta_4 = \nabla_{z_2} J = \delta_2 \circ h_2 \circ (1 - h_2)$$

$$\nabla_W J = \delta_3 x_1^T + \delta_4 x_2^T + \lambda W$$

$$\nabla_b J = \delta_3 + \delta_4$$

2) (3 points) Write out the (vanilla) gradient descent update rules for the model parameters for a single training example (with arbitrary step size $\alpha > 0$).

$$W_{t+1} = W_t - \alpha \nabla_W J$$

$$b_{t+1} = b_t - \alpha \nabla_b J$$

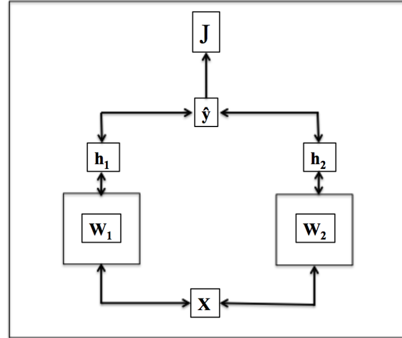
3) (3 points) If $W \in \mathbb{R}^{10\times 5}$ and $b \in \mathbb{R}^{10\times 1}$, how many parameters does the model have?

W,b are the model parameters. 50 +10 = 60. (Note: we don't address hyperparameter tuning in this question, so λ is not updated.)

4) (2 points) How does this model compare to the skip-gram model you implemented in Assignment 1? Give one qualitative similarity or one difference.

Many possible answers, some examples:
- Diff: Not classification, but regression problem
- Diff: Compares word vectors with each other instead of predicting context words.
- Sim: Tries to learn word vector representations (similar input)

In class, we've discussed models for learning word vector representations. Now imagine you wanted to see how ReLU/sigmoid nonlinearities might affect training on single word inputs. But instead of training *two* separate nets, you want to train a psuedo-Siamese net like the one below.



Our model is:

$$h_1 = \sigma(W_1 x + b_1)$$

$$h_2 = \text{relu}(W_2 x + b_2)$$

$$\hat{y} = \text{softmax}(W_3(h_1 + h_2) + b_3)$$

where $x \in \mathbb{R}^n$, W_1, $W_2 \in \mathbb{R}^{m \times n}$, $W_3 \in \mathbb{R}^{k \times m}$, b_1,$b_2 \in \mathbb{R}^m$, and $b_3 \in \mathbb{R}^k$. We evaluate this model for N examples and k classes with cross entropy loss

$$J = -\frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{k} y_j^i log(\hat{y}_j^i)$$

where y_j is the one-hot vector for example j with all probability mass on the correct class and $\hat{y}_j$ are the softmax scores for example j.

---

5) (7 points) Find $\nabla_{h_1} J$, $\nabla_{h_2} J$, and $\nabla_x J$.

Take the arguments of the functions for h_1, h_2, and $\hat{y}$ to be z_1, z_2, and z_3, respectively.

$$\delta_1 = \nabla_{z_3} J = \hat{y} - y$$

$$\delta_2 = \nabla_{h_1} J = \nabla_{h_2} J = W_3^T \delta_1$$

Solutions for the gradient w.r.t a single word vector or w.r.t. a batch of word vectors were accepted.

$$\delta_3 = \nabla_{z_1} J = \delta_2 \circ h_1 \circ (1 - h_1)$$

$$\delta_4 = \nabla_{z_2} J = \delta_2 \circ 1\{z_2 > 0\}$$

$$\nabla_x J = W_1^T \delta_3 + W_2^T \delta_4$$

6) (3 points) Which is likely to train faster, W_1 or W_2? Explain.

W_2: since the ReLU function gradient does not saturate at high values of the input, vanishing gradient is not as much of an issue for this parameter. Since we are training these parameters together, computational efficiency has little effect on training efficiency.

2 Adaptive regularization

Dropout. Dropout is a stochastic regularization technique where with probability p a neuron in a neural network is kept alive. Neurons are dropped during training (during both forward and back propagation). To be more thorough, if you have a hidden layer h

$$h = f(Wx + b)$$

then dropout is an element-wise multiplication of a binary mask matrix m where $m_j \sim Bernoulli(p)$

$$\hat{h} = m \circ f(Wx + b)$$

---

1) (3 points) For the *tanh* ($f(x) = \frac{exp(x)-exp(-x)}{exp(x)+exp(-x)}$) and *ReLU* ($f(x) = \max\{0, x\}$) functions, we can rewrite the equation above as

$$\tilde{h} = f(m \circ (Wx + b))$$

Why is this true? (Hint: This is *not* true if f is the *sigmoid* function $f(x) = \frac{1}{1+exp(-x)}$.)

True since $f(0) = 0$ for tanh, ReLU and not for non-centered sigmoid.

2) (4 points) Suppose you were given $\frac{\partial L}{\partial \tilde{h}}$, the gradient of the loss L with respect to the hidden layer $\tilde{h}$. How would you use this to calculate $\frac{\partial L}{\partial W}$ if $f(x) = \tanh(x)$? Remember, $\tanh'(x) = 1 - (\tanh(x)^2)$.

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \tilde{h}} \frac{\partial \tilde{h}}{\partial W}$$

$$\frac{\partial \tilde{h}}{\partial W} = ((1 - \tilde{h}^2) \circ m)x^T$$

$$\frac{\partial L}{\partial W} = (\frac{\partial L}{\partial \tilde{h}} \circ (1 - \tilde{h}^2) \circ m)x^T$$

Relationship between L2-penalty and SGD. In class, we learned how to update parameter values w with respect to a loss function J via SGD with step size α_t.

$$\text{SGD update: } w_{t+1} = w_t - \alpha_t \nabla_{w_t} J(w_t)$$

$$\text{Linear, L2-penalized update: } w_{t+1} = \underset{w}{\operatorname{argmin}}\{J(w_t) + \nabla_{w_t} J(w_t)^T (w - w_t) + \frac{1}{2\alpha_t}\|w - w_t\|_2^2\}$$

The L2-penalized update involves finding w to minimize the linear approximation of the loss function at w_t (first two terms) with a L2-regularizer term (last term) which mandates that the distance between the current weights and update weights be small. Interestingly, both dropout and Adagrad can be framed as adaptive methods with different regularizer terms. For more detail, see Wager et al.[1]

---

3) (5 points) Show that the L2-penalized update is equivalent to the SGD update.

$$\nabla_W (J(w_t) + \nabla_{w_t} J(w_t)^T (w - w_t) + \frac{1}{2\alpha_t}\|w - w_t\|_2^2)$$
$$= \nabla_{w_t} J(w_t) + \frac{1}{\alpha_t}(w - w_t) = 0$$
$$w = w_{t+1} = w_t - \alpha_t \nabla_{w_t} J(w_t)$$

3 Recurrent nets

GRUs. In class, we learned about RNNs and an extension— *Gated Recurrent Units*. GRUs can adaptively reset or update its "memory" of previous states. The feedforward computation for a GRU is given by

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$
$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1})$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

---

[1] Wager, Stefan, Sida Wang, and Percy S. Liang. "Dropout training as adaptive regularization." Advances in Neural Information Processing Systems. 2013.

1) (3 points) Show that for the sigmoid function $\sigma(x) = \frac{1}{1+exp(-x)}$, $\sigma(-x) = 1 - \sigma(x)$

There are many correct ways to show this—here is one way:

$$\sigma(-x) = \frac{1}{1+exp(x)} = \frac{1/exp(x)}{exp(-x)+1} = \frac{exp(-x)}{1+exp(-x)} = \frac{1+exp(-x)}{1+exp(-x)} - \frac{1}{1+exp(-x)} = 1 - \sigma(x)$$

2) (1 point) True/False. If the update gate z_t is close to 0, the net does not update its state significantly.

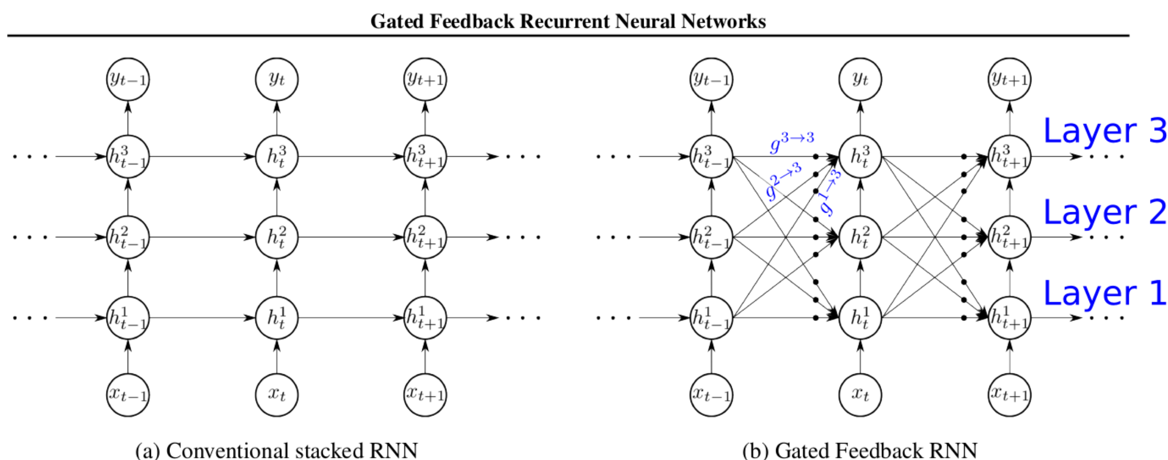
True. In this case, $h_t \approx h_{t-1}$

3) (1 point) True/False. If the update gate z_t is close to 1 and the reset gate r_t is close to 0, the net remembers the past state very well.

False. In this case, h_t depends strongly on input x_t and not on h_{t-1}.

Gated Feedback RNNs (GF-RNNs). GF-RNNs are a new model of stacked RNNs which allow the network to learn long-and short-term dependencies. Unlike in a typical stacked RNN, where information flows from inputs up to higher layers but not back to lower layers, GF-RNNs allow for the back-flow of information from higher to lower layers. Typically, information flows to lower levels at a finer timescale than to higher levels; GF-RNNs allow information in higher layers to be updated at a finer timescale.

Unlike GRUs which have a reset gate for *each* unit, GF-RNNs have a *global reset gate* which controls signal propagation from all layers at some time step $t-1$ to the next time step t. The figure below (lifted directly from Chung et. al[2]) illustrates the difference between conventional stacked RNNs and GF-RNNs.



(a) Conventional stacked RNN (b) Gated Feedback RNN

Figure 1. Illustrations of (a) conventional stacking approach and (b) gated-feedback approach to form a deep RNN architecture. Bullets in (b) correspond to global reset gates. Skip connections are omitted to simplify the visualization of networks.

The global reset gate between layers i and j is given by

$$g^{i\to j} = \sigma(W_g^{i\to j} h_t{}^{j-1} + U_g^{i\to j} \mathbf{h_{t-1}})$$

where $W_g^{i\to j}$ is the weight **vector** for the input states (with $h_t^{j-1} = x_t$ for $j=1$) and $U_g^{i\to j}$ is the weight **vector** for the hidden states for **all** layers at time $t-1$.

Imagine we have each hidden layer neuron compute the tanh function. We have

$$h_t^j = \tanh(W^{j-1\to j} h_t^{j-1} + \sum_{i=1}^{L} g^{i\to j} U^{i\to j} h_{t-1}^i)$$

where we have L layers. (The only difference between this and a typical stacked RNN is that the previous hidden states in *all* layers are controlled by the global reset gates for the layer.)

[2] Chung, Junyoung, et al. "Gated Feedback Recurrent Neural Networks." arXiv preprint arXiv:1502.02367 (2015).

Model:

$$g^{i\to j} = \sigma(W_g^{i\to j} h_t{}^{j-1} + U_g^{i\to j}\mathbf{h_{t-1}})$$

$$h_t^j = \tanh(W^{j-1\to j} h_t^{j-1} + \sum_{i=1}^{L} g^{i\to j} U^{i\to j} h_{t-1}^i)$$

Remember: $g^{i\to j}$ is a scalar.

---

4) (10 points) Given J_t, the loss at time t, and $\nabla_{h_t^j} J_t$ the gradient of the loss with respect to the jth hidden unit at time t, h_t^j , calculate $\nabla_{g^{i\to j}} J_t$ and $\nabla_{W_g^{i\to j}} J_t$. Remember, $\tanh'(x) = 1 - (\tanh(\mathrm{x})^2)$.

Defining backprop through time recursively:

$$\frac{\partial J_t}{\partial g^{i\to j}} = \frac{\partial J_t}{\partial h_t^j}\frac{\partial h_t^j}{g^{i\to j}} = (\frac{\partial J_t}{\partial h_t^j} \circ (1 - (h_t^j)^2))^T (U^{i\to j} h_{t-1}^i + g^{j\to j} U^{j\to j} \frac{\partial h_{t-1}^j}{\partial g^{i\to j}})$$

Here, we do not keep the entire sum in the last term for simplicity, since for all $i \neq j$, $\frac{\partial h_{t-1}^i}{\partial g^{i\to j}} = 0$.

Now that we have the gradient of the loss w.r.t. $g^{i\to j}$, we can use this term to define $\frac{\partial J_t}{\partial W_g^{i\to j}}$.

$$\frac{\partial J_t}{\partial W_g^{i\to j}} = \frac{\partial J_t}{\partial h_t^j}\frac{\partial h_t^j}{\partial g^{i\to j}}\frac{\partial g^{i\to j}}{W_g^{i\to j}} = [\frac{\partial J_t}{\partial g^{i\to j}} g^{i\to j}(1-g^{i\to j})][h_t^{j-1^T} + U_g^{i\to j}[hj : h(j+1)]\frac{\partial h_{t-1}^j}{\partial g^{i\to j}}\frac{\partial g^{i\to j}}{\partial W_g^{i\to j}}]$$