1. Tensorflow Softmax
   (a) Coding
   (b) Coding
   (c) Placeholder variables are placeholders for input data. They are used so that the computational graph is somewhat indpendent from the data that flows through it. Feed dictionaries provide the actual input data corresponding to each placeholder at session run time.
   (d) Coding
   (e) Coding
   (f) Tensorflow defines computations over a directed acyclic graph where nodes are operations. Edges represent data flowing between operations. As long as we know how to compute operations and local gradients at every node, automatic differentiation automatically does forward propogation through the graph and then backpropogates the loss through the nodes using the chain rule of differentiation. As a result, we never have to define the overall gradient explicitly.

2. Neural Transition-Based Dependency Parsing
   (a) Go through the sequence of transitions needed for parsing the sentence "I parsed this sentence correctly."

stack	buffer	new dependency	transition
[ROOT]	[I, parsed, this, sentence, correctly]		Initial Configuration
[ROOT, I]	[parsed, this, sentence, correctly]		SHIFT
[ROOT, I, parsed]	[this, sentence, correctly]		SHIFT
[ROOT, parsed]	[this, sentence, correctly]	$parsed \rightarrow I$	LEFT-ARC
[ROOT, parsed, this]	[sentence, correctly]	$parsed \rightarrow I$	SHIFT
[ROOT, parsed, this, sentence]	[correctly]	$parsed \rightarrow I$	SHIFT
[ROOT, parsed, sentence]	[correctly]	$parsed \rightarrow I$ $sentence \rightarrow this$	LEFT-ARC
[ROOT, parsed]	[correctly]	$parsed \rightarrow I$ $sentence \rightarrow this$ $parsed \rightarrow sentence$	RIGHT-ARC
[ROOT, parsed, correctly]	[]	$parsed \rightarrow I$ $sentence \rightarrow this$ $parsed \rightarrow sentence$	SHIFT
[ROOT, parsed]	[]	$parsed \rightarrow I$ $sentence \rightarrow this$ $parsed \rightarrow sentence$ $parsed \rightarrow correctly$	RIGHT-ARC
[ROOT]	[]	$parsed \rightarrow I$ $sentence \rightarrow this$ $parsed \rightarrow sentence$ $parsed \rightarrow correctly$ $ROOT \rightarrow parsed$	RIGHT-ARC

   (b) Each word in a sentence is moved twice during a transition-based dependency parse - once from the *buffer* to the *stack*, and next from the *buffer* to the *dependency list*. Each transition moves one word from the *buffer* to the *stack* or from the *stack* to the *dependency list*.
   So, given a sentence with n words, it will be parsed in $2n$ steps.
   (c) Coding
   (d) Coding
   (e) Coding
   (f) What must γ equal in terms of p_{drop} ?

$$
\begin{aligned}
E_{p_{drop}}[h_{drop}]_i &= h_i \\
E_{p_{drop}}[\gamma * d_i * h_i] &= h_i \\
p_{drop} * \gamma * 1.0 * h_i + (1 - p_{drop}) * \gamma * 0.0 * h_i &= h_i \\
p_{drop} * \gamma * 1.0 * h_i &= h_i \\
\gamma &= \frac{1}{p_{drop}}
\end{aligned}
$$

   (g) i. Briefly explain (you don't need to prove mathematically, just give an intuition) how using m stops the updates from varying as much. Why might this help with learning?

Ans: Update on m retains β_1 of the previous value of m. So, if β_1 is high enough, the update doesn't change the value of m dramatically. This helps with learning because the gradient is now a rolling average over multiple minibatches and closer to the true gradient value for the whole dataset. It also prevents the updates from changing too much and oscillating around the local minima.

ii. Which of the model parameters will get larger updates? Why might this help with learning?
Ans: The parameters with smaller absolute rolling average of gradients will get larger updates (due to the division by magnitude of rolling average of gradients). This may help with learning because parameters with low gradients (rolling average) which are perhaps stuck on in a plateau region will get a larger update and could jump away from the plateau.

(h) UAS on test set = 88.60 (10 epochs)

3. Recurrent Neural Networks: Language Modeling

(a)

$$PP\left(y^{(t)}, \hat{y}^{(t)}\right) = \frac{1}{\hat{y}_j^{(t)}} \quad where\ y_j^{(t)} = 1$$

$$CE\left(y^{(t)}, \hat{y}^{(t)}\right) = -ln(\hat{y}_j^{(t)})$$

$$PP\left(y^{(t)}, \hat{y}^{(t)}\right) = \exp\left(CE\left(y^{(t)}, \hat{y}^{(t)}\right)\right)$$

So, minimizing the arithmetic mean of the cross entropy loss also minimizes the geometric mean of the perplexity across the training set.
For $|V| = 10000$,

$$\hat{y}^{(t)} = \frac{1}{|V|} = 10^{-4}$$

$$PP\left(y^{(t)}, \hat{y}^{(t)}\right) = 10^4$$

$$CE\left(y^{(t)}, \hat{y}^{(t)}\right) = -ln(10^{-4}) \approx 9.21$$

(b)

$$\delta_1^{(t)} = \frac{\partial J}{\partial z^{(t)}} = \hat{y}^{(t)} - y^{(t)}$$

$$z^{(t)} = h^{(t)}U + b_2$$

$$\boldsymbol{\frac{\partial J^{(t)}}{\partial b_2} = \delta_1^{(t)}}$$

$$\delta_2^{(t)} = \frac{\partial z^{(t)}}{\partial h^{(t)}} = U^T$$

$$w^{(t)} = h^{(t-1)}H + e^{(t)}I + b_1$$

$$h^{(t)} = sigmoid(w_{(t)})$$

$$\delta_3^{(t)} = \frac{\partial h^{(t)}}{\partial w^{(t)}} = h^{(t)} \circ (1 - h^{(t)})$$

$$\boldsymbol{\frac{\partial J^{(t)}}{\partial H} = (h^{(t-1)})^T(\delta_1^{(t)} * \delta_2^{(t)} \circ \delta_3^{(t)})}$$

$$\boldsymbol{\frac{\partial J^{(t)}}{\partial I} = (e^{(t)})^T(\delta_1^{(t)} * \delta_2^{(t)} \circ \delta_3^{(t)})}$$

$$\delta_4^{(t)} = \frac{\partial w^{(t)}}{\partial e^{(t)}} = I^T$$

$$e_{(t)} = L_{x(t)}$$

$$\boldsymbol{\frac{\partial J^{(t)}}{\partial L_{x(t)}} = (\delta_1^{(t)} * \delta_2^{(t)} \circ \delta_3^{(t)}) * I^T}$$

(c)

$$\delta^{(t-1)} = \delta_1^{(t)} * \delta_2^{(t)} \circ \delta_3^{(t)} * H^T$$

$$\frac{\partial h^{(t-1)}}{\partial w^{(t-1)}} = h^{(t-1)} \circ (1 - h^{(t-1)})$$

Derivatives contributed by $t-1$ step -

$$\frac{\partial J^{(t)}}{\partial H} = \left(h^{(t-2)}\right)^T * (\delta^{(t-1)} \circ h^{(t-1)} \circ (1 - h^{(t-1)}))$$

$$\frac{\partial J^{(t)}}{\partial I} = \left(e^{(t-1)}\right)^T * (\delta^{(t-1)} \circ h^{(t-1)} \circ (1 - h^{(t-1)}))$$

$$\frac{\partial J^{(t)}}{\partial L_{x(t-1)}} = \frac{\partial J^{(t)}}{\partial e^{(t-1)}} = (\delta^{(t-1)} \circ h^{(t-1)} \circ (1 - h^{(t-1)})) * I^T$$

(d) Forward Propagation:

$$\begin{aligned} J(\theta) &=> O(1) \\ \hat{y} &=> O(|V|D_h) \\ h &=> O({D_h}^2 + dD_h) \\ e &=> O(1) \\ ForwardPropagation &=> O(|V|D_h + {D_h}^2 + dD_h) \end{aligned}$$

Backward Propagation:

$$\begin{aligned} O(|V|Dh + {D_h}^2 + dD_h) \\ For\ \tau\ time\ steps: \\ O(\tau(|V|Dh + {D_h}^2 + dD_h)) \end{aligned}$$

Slow Step: $O(|V|D_h)$ because $|V| >> D_h$ and $|V| >> d$ Calculating the softmax at each step is the most expensive part