# CS224n - Assignment 2

By Dominique Paul

## Question 1

(c)
Placeholder Variables:
- Placeholder variables are empty Tensorflow variables which we use to tell Tensorflow that we will replace these values later on. We would not want to use constants in any model to make it useable for other data as well.

Feed Dictionaries:
- Feed Dictionaries are used for training for new variables to be fed in each iteration.

(d)
Automatic Differentiation
- Tensorflows method of building a graph before filling in values allows for a structure that knows how to calculate the backward pass for each mathematical operation as we construct our graph. Therefore, we don't have to take care of calculating the gradients ourselves.

## Question 2

(a)

| 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 → I | LEFT-ARC |
| [ROOT, parsed, this] | [sentence, correctly] | | SHIFT |
| [ROOT, parsed, this, sentence] | [correctly] | | SHIFT |
| [ROOT, parsed, sentence] | [correctly] | Sentence → this | LEFT-ARC |
| [ROOT, parsed, sentence] | [correctly] | Parsed → sentence | RIGHT-ARC |

| | | | |
|---|---|---|---|
| [ROOT, parsed, correctly] | [] | | SHIFT |
| [ROOT, parsed] | [] | Parsed → correctly | RIGHT-ARC |
| [ROOT] | [] | ROOT → parsed | RIGHT-ARC |

(b) A sentence containing n words will be parsed in how many steps (in terms of n)? Briefly explain why.

A sentence of n words will be parsed in 2*n steps. Each word has to be shifted at least once, and each word has to be assigned an arc operation to another word at least once.

(f) (2 points, written) We will regularize our network by applying Dropout[3]. During training this randomly sets units in the hidden layer $\boldsymbol{h}$ to zero with probability $p_{drop}$ and then multiplies $\boldsymbol{h}$ by a constant $\gamma$ (dropping different units each minibatch). We can write this as

$$\boldsymbol{h}_{drop} = \gamma \boldsymbol{d} \circ \boldsymbol{h}$$

where $\boldsymbol{d} \in \{0, 1\}^{D_h}$ ($D_h$ is the size of $\boldsymbol{h}$) is a mask vector where each entry is 0 with probability $p_{drop}$ and 1 with probability $(1 - p_{drop})$. $\gamma$ is chosen such that the value of $\boldsymbol{h}_{drop}$ in expectation equals $\boldsymbol{h}$:

$$\mathbb{E}_{p_{drop}}[\boldsymbol{h}_{drop}]_i = \boldsymbol{h}_i$$

for all $0 < i < D_h$. What must $\gamma$ equal in terms of $p_{drop}$? Briefly justify your answer.

Epsilon = 1 / (1 - p_drop)

It is stated that epsilon is chosen in a manner that the expected value of h_drop equals h. Therefore if we drop e.g. 90% of the values, then we would have to make the remaining 10% ten times as big to have an expected value of the sum of the values equal to the original sum.

(g) (4 points, written) We will train our model using the Adam[4] optimizer. Recall that standard SGD uses the update rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta})$$

where $\boldsymbol{\theta}$ is a vector containing all of the model parameters, $J$ is the loss function, $\nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta})$ is the gradient of the loss function with respect to the parameters on a minibatch of data, and $\alpha$ is the learning rate. Adam uses a more sophisticated update rule with two additional steps[5].

(i) First, Adam uses a trick called *momentum* by keeping track of $\boldsymbol{m}$, a rolling average of the gradients:

$$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1-\beta_1)\nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{m}$$

where $\beta_1$ is a hyperparameter between 0 and 1 (often set to 0.9). Briefly explain (you don't need to prove mathematically, just give an intuition) how using $\boldsymbol{m}$ stops the updates from varying as much. Why might this help with learning?

(ii) Adam also uses *adaptive learning rates* by keeping track of $\boldsymbol{v}$, a rolling average of the magnitudes of the gradients:

$$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1-\beta_1)\nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta})$$
$$\boldsymbol{v} \leftarrow \beta_2 \boldsymbol{v} + (1-\beta_2)(\nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta}) \circ \nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta}))$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \circ \boldsymbol{m}/\sqrt{\boldsymbol{v}}$$

where $\circ$ and $/$ denote elementwise multiplication and division (so $\boldsymbol{z} \circ \boldsymbol{z}$ is elementwise squaring) and $\beta_2$ is a hyperparameter between 0 and 1 (often set to 0.99). Since Adam divides the update by $\sqrt{\boldsymbol{v}}$, which of the model parameters will get larger updates? Why might this help with learning?

(i) The parameter m holds information on the past weight updates. If the current weight update has been close to zero, but past updates have been quite large, then the new update will nevertheless be of considerable size. The momentum factor m can help overcome local minima through its consideration of past updates and also ignores some of the noise caused by the batches.

(ii) The factor affects the updates by making those gradients with the on-average larger gradients smaller and those gradients with on-average smaller values bigger. This helps small updates from plateauing and getting stuck.

## Question 3

(3c) Cross entropy is defined as

$$CE = -\sum y_u \log(\hat{y}_u)$$

Considering that $y$ is one hot encoded and will only equal 1 once we can also write

$$CE = -\log \hat{y}_i$$

Equally we can write the Perplexity as:

$$PP^{(t)}(y^{(t)}, \hat{y}^{(t)}) = \frac{1}{\hat{y}_i^{(t)}}$$

Using the logarithm rules we can rewrite CE as

$$CE = \log \frac{1}{\hat{y}_i}$$

And therefore we can write CE using the perplexity

$$CE = \log PP^{(t)}(y^{(t)}, \hat{y}^{(t)})$$

Due to this logarithmic transformation minimizing the arithmetic mean of the cross entropy is equivalent to minimizing the geometric mean of the perplexity

The expected value for $\hat{y}_i$ would be $E(\hat{y}_i) = \frac{1}{|V|}$. This corresponds to an expected value of $E(PP) = 1/1/|V| = |V|$ and an expected Cross Entropy error of $\log(|V|)$, which for $|V| = 10.000$ would equal $\log(10.000) = 9.210$

(3B) First we ~~present~~ write some simplifications as follows

$$z = h^{(t-1)} H + e^{(t)} I + b_1$$
$$v = h^{(t)} U + b_2$$

Then:

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

$$\delta_2 = \frac{\partial J}{\partial z} = \delta_1 U^T \circ h \circ (1-h)$$

which we use to answer the original question:

$$\frac{\partial J}{\partial b_2} = \delta_1$$

$$\frac{\partial J}{\partial L_{x^{(t)}}} = \delta_2 I^T$$

$$\frac{\partial J}{\partial I} = (e^{(t)})^T \delta_2$$

$$\frac{\partial J}{\partial H} = h^{(t-1)T} \delta_2^{(t)}$$

$$\frac{\partial J}{\partial h^{t-1}} = \delta_2^{(t)} H^T$$

(3c)

$$\begin{tikzcd}
\hat{y}^{t-3} & \hat{y}^{t-2} & \hat{y}^{t-1} & \hat{y}^{t} & \\
h^{t-3} \arrow[u] \arrow[r] & h^{t-2} \arrow[u] \arrow[r] & h^{t-1} \arrow[u] \arrow[r] & h^{t} \arrow[u] \arrow[r] & h^{t+1} \\
x^{t-3} \arrow[u] & x^{t-2} \arrow[u] & x^{t-1} \arrow[u] & x^{t} \arrow[u] &
\end{tikzcd}$$

$$\frac{\partial J}{\partial L_{x^{t-1}}} = \delta^{t-1} \sigma'(v^{t-1}) I^T$$

$$\frac{\partial J}{\partial I} = e^{t-1^T} \delta^{t-1} \sigma'(v^{t-1})$$

$$\frac{\partial J}{\partial H} = h^{t-2^T} \delta^{t-1} \sigma'(v^{t-1})$$

where $\sigma'(v^{t-1}) = \mathrm{diag}(h^{t-1} \circ (1 - h^{t-1}))$

(3d) Forward propagation: $O(|V| D_h + d D_h + D_h^2)$

Backward propagation: $O(\tau(|V| D_h + d D_h + D_h^2))$

→ Back-prop takes $\tau$ times as long as forward propagation

→ if $|V| \gg D_h$ then $|V| D_h$ takes very long as well