CS224N - Stanford University

ZIAUL HASAN HASHMI

Assignment #2

(1) Tensorflow Softmax

(a) Please check code

(b) Please check code

(c) Placeholder variables allow us to assign data at a later stage hence create operations to build the computational graph without requiring the data beforehand. Feed dictionaries make it possible to feed data such as training data into the computational graph.

Please check code for implementation

(d) Please check code for implementation

(e) Please check code for implementation part

When train_op is called Tensorflow automatically computes the gradients of all the variables (weights & biases) in the computational graph by minimizing the loss via gradient descent algorithm.

(2) Neural Transition-Based Dependency Parsing

(a)

stack	buffer	new dependency	transition
[ROOT]	[I, parsed, this, sentence correctly]		Initial Config.
[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]	[correctly]	Parsed → sentence	Right-ARC
[ROOT, Parsed, Correctly]	ϕ		Shift
[ROOT, Parsed]	ϕ	Parsed-correctly	Right-ARC
[ROOT]	ϕ	ROOT-Parsed	Left-ARC

(b) Since each word is processed twice; once to move to the stack and once to create dependency, so for a sentence containing n words we need "$2n$" steps.

(c) Please check code implementation

(d) Please check code implementation

(e) Please check code for implementation

(f) $E_{p_{drop}}[h_{drop}]_i$

$$= r\left[p_{drop}(0) + (1 - p_{drop})(h_i)\right]$$

$$= r(1 - p_{drop})h_i$$

If $E_{p_{drop}}[h_{drop}]_i = h_i$

$$\Rightarrow \quad r = \frac{1}{1 - p_{drop}}$$

(g) (i) Each "m" will approximately be the same as previous one because $(1-\beta_1)$ is small. Therefore updates won't vary that much.

Since minibatch gradients are random approximations of the true gradient, momentum calculates a rolling average of gradients and stops the oscillations. This helps SGD to accelerate in the right direction. Essentially stops parameters from bouncing around too much.

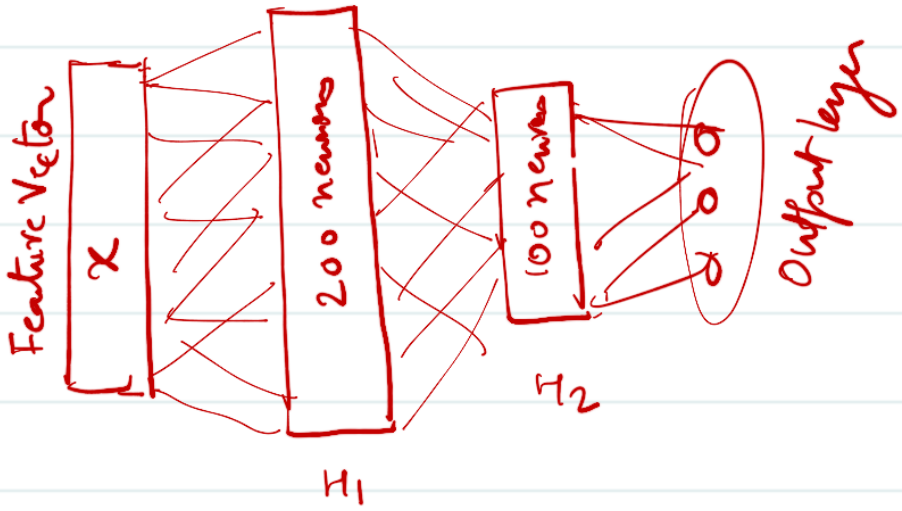
(ii) The model parameters with smaller average gradients will get larger updates. This should help to move such parameters for which loss doesn't change much, effectively moving them from flat portions (plateaus).

(h)

- Please check the implementation of my parser in "q2_parser_model.py"
- dev set UAS = 88.66

  test set UAS = 84.24
- Please check q2_test.predicted.pkl file.

(i) I added one additional hidden layer with 100 units (with Dropouts)



I also reduced the dropout probability to 0.2 so keep probability is 0.8 & increased epochs to 12.
Evaluation result:

- ★ dev set UAS = 88.98
- ★ test set UAS = 85.39

Adding additional layer helps to compute more complex features that can help learn the dependency structure. Also, I increased epochs to train the parameters even more and reduced dropouts to reduce regularization effect. Perhaps we can play with parameters such as learning rate and hidden layer size to further improve the results.

(3) Recurrent Neural Networks: Language Modelling

(a) (i) If $y^{(t)}$ is one hot, assume $y_i^{(t)} = 1$ and zero everywhere else, then

$$CE(y^{(t)}, \hat{y}^{(t)}) = -\log \hat{y}_i^{(t)} = \log \frac{1}{\hat{y}_i^{(t)}}$$

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

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

$$\Rightarrow \boxed{PP^{(t)}(y^{(t)}, \hat{y}^{(t)}) = e^{CE(y^{(t)}, \hat{y}^{(t)})}}$$

(ii)

$$\arg\min \frac{1}{T} \sum_{t=1}^{T} CE\left(y^{(t)}, \hat{y}^{(t)}\right)$$

$$= \arg\min \frac{1}{T} \sum_{t=1}^{T} \log pp^{(t)}\left(y^{(t)}, \hat{y}^{(t)}\right)$$

$$= \arg\min \sum_{t=1}^{T} \log\left[pp^{(t)}\left(y^{(t)}, \hat{y}^{(t)}\right)\right]^{\frac{1}{T}}$$

$$= \arg\min \log \prod_{t=1}^{T} \left(pp^{(t)}\left(y^{(t)}, \hat{y}^{(t)}\right)\right)^{\frac{1}{T}}$$

$$= \arg\min \prod_{t=1}^{T} pp^{(t)}\left(y^{(t)}, \hat{y}^{(t)}\right)^{\frac{1}{T}}$$

Q. E. D

(iii)

Perplexity $PP(y^{(t)}, \hat{y}^{(t)})$

$$= \frac{1}{1/|V|}$$

$$= |V|$$

Perplexity would therefore be $|V|$ for a single word.

Cross entropy $CE(y^{(t)}, \hat{y}^{(t)}) = \log(\text{perplexity})$

$$= \log |V|$$

for $|V| = 10000$

$CE = 9.21$ (assuming natural log)

(b) $e^{(t)} = E x^{(t)}$

$h^{(t)} = \text{sigmoid}\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$

$\hat{y}^{(t)} = \text{softmax}\left(U h^{(t)} + b_2\right)$

As per these equations

$h^{(t)}, y^{(t)}, \hat{y}^{(t)}, e^{(t)}$ are assumed to be column vectors. We assume gradients are row vectors so appropriate transpose is required during gradient updates

Assume $\boxed{\begin{aligned} z^{(t)} &= W_h h^{(t-1)} + W_e e^{(t)} + b_1 \\ \theta^{(t)} &= U h^{(t)} + b_2 \end{aligned}}$

$$\delta_1^{(t)} = \frac{\partial J^{(t)}}{\partial \theta^{(t)}} = \left(\hat{y}^{(t)} - y^{(t)}\right)^T \in \mathbb{R}^{1 \times |V|}$$ [From Assignment #1]

$$\delta_2^{(t)} = \frac{\partial J^{(t)}}{\partial z^{(t)}} = \delta_1^{(t)} \frac{\partial \theta^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial z^{(t)}}$$ [since $\sigma'(x) = \sigma(x)(1-\sigma(x))$]

$$= \delta_1^{(t)} U \circ h^{(t)} \circ \left(1 - h^{(t)}\right)$$

$$\in \mathbb{R}^{1 \times D_h}$$

$$\therefore \quad \frac{\partial J^{(t)}}{\partial U} = \left(\hat{y}^{(t)} - y^{(t)}\right)\left(h^{(t)}\right)^T \in \mathbb{R}^{|V| \times D_h}$$

$|V| \times 1$ $\quad$ $1 \times D_h$

(We derived similar derivative in Assignment #1)

$$\frac{\partial J^{(t)}}{\partial e^{(t)}} = \delta_2^{(t)} \cdot W_e \in \mathbb{R}^{1 \times d}$$

$1 \times D_h$ $\quad$ $D_h \times d$ $\quad$ (Dimension check)

$$\left.\frac{\partial J^{(t)}}{\partial W_e}\right|_t = \left(\delta_2^{(t)}\right)^T \cdot \left(e^{(t)}\right)^T \in \mathbb{R}^{D_h \times d}$$

$(D_h \times 1)$ $\quad$ $(1 \times d)$ $\quad$ (Dimension check)

$$\left.\frac{\partial J^{(t)}}{\partial W_h}\right|_t = \left(\delta_2^{t}\right)^T \cdot \left(h^{(t-1)}\right)^T \in \mathbb{R}^{D_h \times D_h}$$

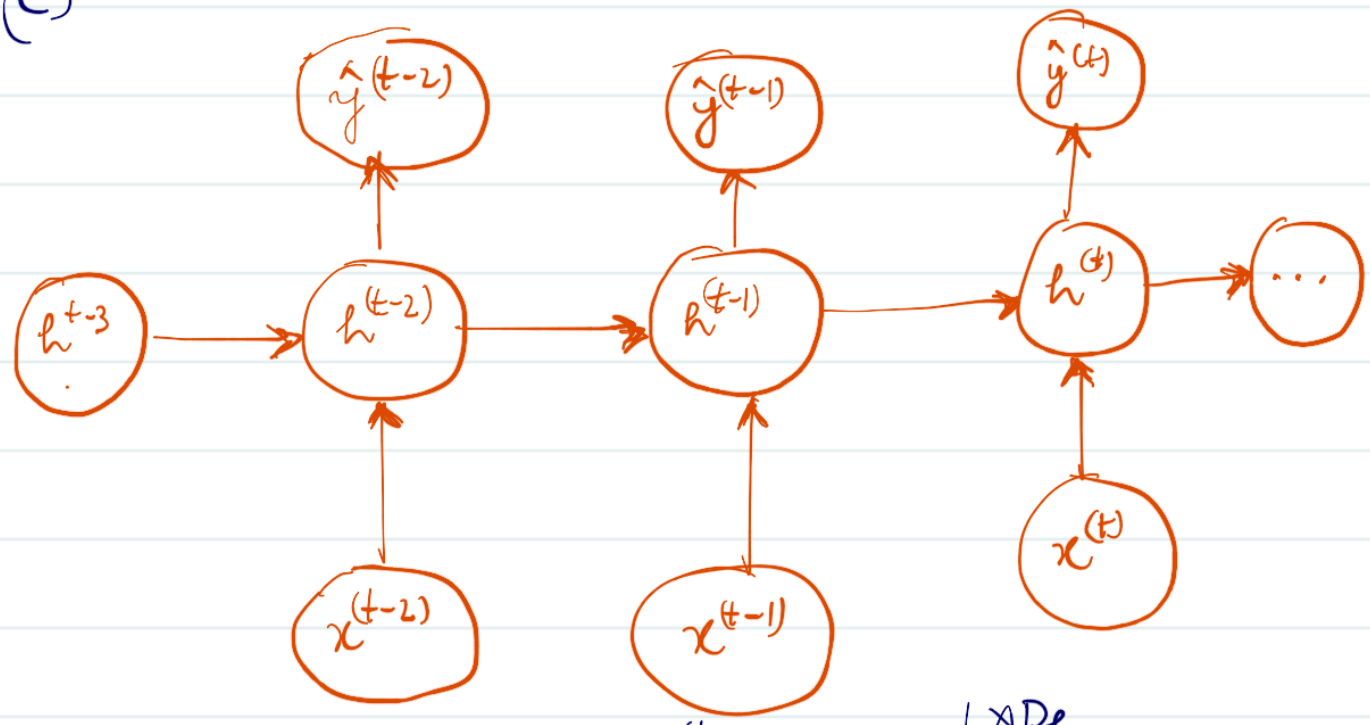
$D_h \times 1$ $\quad$ $1 \times D_h$ $\quad$ (Dimension check)

Finally,

$$\frac{\partial J^{(t)}}{\partial h^{(t-1)}} = \delta_2^{(t)} \cdot W_h \in \mathbb{R}^{1 \times D_h}$$

$1 \times D_h$ $\quad$ $D_h \times D_h$ $\quad$ (Dimension check)

(c)



Lets say $r^{(t-1)} = \frac{\partial J^{(t)}}{\partial h^{(t-1)}} \in \mathbb{R}^{1 \times D_h}$

$h^{(t-1)} = \text{Sigmoid}(z^{(t-1)})$

where $z^{(t-1)} = W_h h^{(t-2)} + W_e c^{(t-1)} + b_1$

$\frac{\partial h^{(t-1)}}{\partial z^{(t-1)}} = \text{diag}(h^{(t-1)} \circ (1 - h^{(t-1)}))$

$\delta_2^{(t-1)} = \frac{\partial J^{(t)}}{\partial z^{(t-1)}} = r^{(t-1)} \circ h^{(t-1)} \circ (1 - h^{(t-1)}) \in \mathbb{R}^{1 \times D_h}$

$\therefore \quad \frac{\partial J^{(t)}}{\partial c^{(t-1)}} = \delta_2^{(t-1)} \cdot W_e$

$$\frac{\partial J^{(t)}}{\partial e^{(t-1)}} = \left(\gamma^{(t-1)} \circ h^{(t-1)} \circ \left(1 - h^{(t-1)}\right)\right) \cdot W_e$$

$$\left.\frac{\partial J^{(t)}}{\partial W_e}\right|_{t-1} = \left(\delta_2^{(t-1)}\right)^T \cdot \left(e^{(t-1)}\right)^T$$

Using results from (b)

$$\Rightarrow \left.\frac{\partial J^{(t)}}{\partial W_e}\right|_{t-1} = \left(\gamma^{(t-1)} \circ h^{(t-1)} \circ \left(1 - h^{(t-1)}\right)\right)^T \cdot \left(e^{(t-1)}\right)^T$$

Similarly,

$$\left.\frac{\partial J^{(t)}}{\partial W_h}\right|_{(t-1)} = \left(\delta_2^{(t-1)}\right)^T \cdot \left(h^{(t-2)}\right)^T$$

(Using results from (b))

or

$$\left.\frac{\partial J^{(t)}}{\partial W_h}\right|_{(t-1)} = \left(\gamma^{(t-1)} \circ h^{(t-1)} \circ \left(1 - h^{(t-1)}\right)^T\right)^T \cdot \left(h^{(t-2)}\right)^T$$

(d)

Complexity for $\frac{\partial J^{(t)}}{\partial U} \approx O(|V| \times D_h)$

Complexity for $\frac{\partial J^{(t)}}{\partial e^{(t)}} \approx O(D_h \times d)$

Complexity for $\left.\frac{\partial J^{(t)}}{\partial W_e}\right|_{(t)} \approx O(D_h \times d)$

Complexity for $\left.\frac{\partial J^{(t)}}{\partial W_h}\right|_{(t)} \approx O(D_h \times D_h)$

Therefore overall complexity is $O(|V| D_h + d D_h + D_h^2)$

(c) If we do backpropagation for T time steps for one word, the complexity would be:

$$O(|V| D_h + T(d D_h + D_h^2))$$

And if we hence do backpropagation through time for each of sequence of T words separately, then it would hence take

$$O(T|V| D_h + T^2(d D_h + D_h^2))$$

Essentially, here we would be passing error signal for each word all the way to $t=0$ (average T times). However, we don't need to since all the terms are common except error term. We just need to multiply error term once for each common term, which grows as we move forward. Hence we can reduce the net complexity to

$$O(T(|V| h + d D_h + D_h^2))$$

(f) The term $O(|V| D_h)$ to compute $\frac{\partial J^{(t)}}{\partial U}$ is the largest since $|V| >> D_h$. The computational bottleneck is between the hidden and output layers. This term corresponds to the gradient updates of the output layer. The output layer computes the probability distribution of the next words.

Bonus Question:

The bottleneck is due to the huge vocabulary size. One idea to speed up the computation could be to group the words in a smaller number of classes such that each word belongs to one class only (based on frequency), then we obtain two probability distributions via softmax, one for class prediction and one for word within that class