NLP Assignment 3

Parisa Yalsavar

April 24, 2021

1 Machine Learning & Neural Networks

1.1 a

i.
Momentum keeps some fraction (β_1) of the previous gradients and adds them to the current update (Mathematically it takes an exponential average of the gradient steps). This makes the updates have rather a particular direction than just moving back and forth like in SGD. Low variance (reduced oscillation) makes the algorithm move more quickly towards the minima and helps the learning converge faster.
ii.
Adaptive learning rate uses estimations of second moments of gradient to adapt the learning rate for each parameter of the model. The parameters with smaller **v** values will get larger updates, these parameters are the ones that receive small or infrequent updates. Due to the accumulation of the squared gradients in **v**, the learning rates of parameters with large updates get smaller over time. This creates a normalization effect on the learning rate of the parameters.

1.2 b

i.
The expectation of **X** is defined as:

$$\mathbb{E}[X] = \sum_{i=1}^{k} x_i\, p_i = x_1 p_1 + x_2 p_2 + \cdots + x_k p_k.$$

We also know that for a matrix A, the expectation is:

$$\mathbb{E}[\mathbf{A}]_i = \mathbb{E}[\mathbf{A}i]$$

Rewriting the given formula:

$$\begin{aligned} \mathbb{E}_{p_{drop}}[h_{drop}]_i & \\ &= \mathbb{E}_{p_{drop}}[\gamma \mathbf{d} \circ \mathbf{h}]_i = h_i \\ &= \gamma \mathbb{E}_{p_{drop}}[d_i h_i] \\ &= \gamma[0 \times (p_{drop}) + 1 \times (1 - p_{drop})]h_i = h_i \\ \implies \gamma[1 \times (1 - p_{drop})] = 1 & \\ \gamma = \frac{1}{1 - p_{drop}} & \end{aligned}$$

ii.
Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs and thus reducing overfitting. At training we want the model to generalize better, so we use dropout. But at test/evaluation time, we want to measure the performance of the model and applying dropout will add noise to predictions that will damage the performance.

2 Neural Transition-Based Dependency Parsing

2.1 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]	[correctly]	parsed → sentence	`Right-Arc`
[ROOT, parsed, correctly]	[]		`Shift`
[ROOT, parsed]	[]	parsed → correctly	`Right-Arc`
[ROOT]	[]	ROOT → parsed	`Right-Arc`

2.2 b

Each word gets parsed in two steps. One for SHIFT and one for either of ARCs. If a sentence contains n words, it will be parsed in $2n$ steps. Algorithm has a time complexity of $\mathcal{O}(n)$.

2.3 e

dev UAS	**test UAS**
88.50	89.09

2.4 f

i.

- **Error type**: Verb Phrase Attachment Error
- **Incorrect dependency**: wedding ⟶ fearing
- **Correct dependency**: heading ⟶ fearing

ii.

- **Error type**: Coordination Attachment Error
- **Incorrect dependency**: rescue ⟶ and
- **Correct dependency**: rush ⟶ and

iii.

- **Error type**: Prepositional Phrase Attachment Error
- **Incorrect dependency**: named ⟶ Midland
- **Correct dependency**: guy ⟶ Midland

iv.

- **Error type**: Modifier Attachment Error
- **Incorrect dependency**: elements ⟶ most
- **Correct dependency**: crucial ⟶ most