CS 224n Assignment 3.

Abhishek Goswami.

January 28, 2019

1. Machine Learning & Neural Networks

   (a) Adam Optimizer.

      i. $\boldsymbol{m}$ is rolling average of the gradients. β_1 is a hyper parameter between 0 and 1.

$$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} \boldsymbol{J}_{\text{minibatch}}(\boldsymbol{\theta}). \tag{1}$$

      The fact that β_1 is set to a high value (0.9 often) suggests that $\boldsymbol{m}$ is changing very little at each step. This in turn ensures that the model parameters do not bounce around when moving towards the local optimum.

      ii. $\boldsymbol{v}$ is the rolling average of the magnitude of the gradients.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \bigodot \boldsymbol{m} / \sqrt{\boldsymbol{v}}. \tag{2}$$

      Since we are dividing by $\sqrt{\boldsymbol{v}}$ it means that the larger gradients will get smalller updates. Conversely, the smaller gradients will get larger updates. This will help the learning algorithm to move off flat areas.

   (b) Dropout.

      i. γ in terms of p_{drop}

$$\begin{aligned} h_i &= \mathbb{E}_{p_{drop}}[\mathbf{h}_{drop}]_i \\ &= \mathbb{E}_{p_{drop}}[\gamma d_i h_i] \\ &= p_{drop}(0) + (1 - p_{drop})\gamma h_i \\ &= (1 - p_{drop})\gamma h_i \end{aligned} \tag{3}$$

      So,

$$\gamma = \frac{1}{1 - p_{drop}} \tag{4}$$

$$\tag{5}$$

      ii. Dropout is a regularization technique that we want to use during training to prevent overfitting on the training data.
      At test time, we should not do dropout, else it would result in randomness in predictions. One thing we should do during test time is to scale the predictions appropriately to account for the expectation term $\mathbb{E}_{p_{drop}}[\mathbf{h}_{drop}]_i$. Alternatively, we can use **inverted dropout** so we do the scaling at training time itself, and leave the code untouched during test time.

Stack	Buffer	New dependency	Transition
[ROOT]	[I, parsed, this, sentence, correctly]		start
[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

Table 1: Transitions for sentence "*I parsed this sentence correctly*".

2. Neural Transition-Based Dependency Parsing

   (a) Table 1 shows the sequence of transitions needed for parsing the sentence : "*I parsed this sentence correctly*"

   (b) A sentence containing n words will be parsed in $2n$ steps. It is either (a) SHIFT or (b) one of $\{LEFT-ARC|RIGHT-ARC\}$

   (c) Coding exercise.

   (d) Coding exercise.

   (e) dev UAS: 88.36. test UAS: 88.90

   (f) Example dependency parses.

      i. • **Error type** item 1
         • **Incorrect dependency** item 2
         • **Correct dependency** item 3

      ii. • **Error type** item 1
         • **Incorrect dependency** item 2
         • **Correct dependency** item 3

      iii. • **Error type** item 1
         • **Incorrect dependency** item 2
         • **Correct dependency** item 3

      iv. • **Error type** item 1
         • **Incorrect dependency** item 2
         • **Correct dependency** item 3