Assignment 3: Transition Parsing with Neural Networks

Sabir Ismail

SBU ID: 111734933

I run the code in the Python 3.6, so I have made few changes in the Util.py and DependencyTree.py also.

1. **Results of the experiments:** The following table contains the configuration of the model and results.

Configurations	Accuracy on the dev set

`# Cube non-linearity`
`# mean = 0.1`
`h = tf.pow(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input), 3)`
`pred = tf.add(tf.matmul(weights_output, h), self.b2)`
	`Testing on dev set at step  1000`
	`UAS: 42.58294488620784`
	`UASnoPunc: 45.111060871531116`
	`LAS: 37.14883964404118`
	`LASnoPunc: 39.069123382128524`
	
	`UEM: 2.5294117647058822`
	`UEMnoPunc: 2.6470588235294117`
	`ROOT: 27.058823529411764`

`# Cube non-linearity`
`h = tf.pow(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input), 3)`
`pred = tf.matmul(weights_output, h)`
	`Testing on dev set at step  1000`
	`UAS: 56.19313507989132`
	`UASnoPunc: 59.99547843780026`
	`LAS: 49.48774833611686`
	`LASnoPunc: 52.896625784208446`
	
	`UEM: 3.1176470588235294`
	`UEMnoPunc: 3.588235294117647`
	`ROOT: 30.764705882352942`

`# Sigmoid non-linearity`
`h = tf.nn.sigmoid(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input))`
`pred = tf.add(tf.matmul(weights_output, h), self.b2)`
	`Testing on dev set at step  1000`
	`UAS: 19.2013360919311`
	`UASnoPunc: 19.73944497823998`
	`LAS: 9.736520677019717`
	`LASnoPunc: 10.939354546995988`
	
	`UEM: 0.8235294117647058`
	`UEMnoPunc: 0.8235294117647058`
	`ROOT: 2.6470588235294117`

`# Sigmoid non-linearity`
`h = tf.nn.sigmoid(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input))`
`pred = tf.matmul(weights_output, h)`
	`Testing on dev set at step  1000`
	`UAS: 18.632998479447615`
	`UASnoPunc: 19.066862601028657`
	`LAS: 8.87404342298776`
	`LASnoPunc: 9.947436839428022`
	
	`UEM: 0.8235294117647058`
	`UEMnoPunc: 0.8235294117647058`
	`ROOT: 2.823529411764706`

`# Cube non-linearity`
`h = tf.pow(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input), 3)`
`pred = tf.add(tf.matmul(weights_output, h), self.b2)`
`mean = 0.0`
	`Testing on dev set at step  1000`
	`UAS: 60.91183288880026`
	`UASnoPunc: 64.7600746057763`
	`LAS: 55.42039534361991`
	`LASnoPunc: 59.00921268298197`
	
	`UEM: 4.9411764705882355`
	`UEMnoPunc: 5.235294117647059`
	`ROOT: 32.94117647058823`

`# Tanh non-linearity`
`h = tf.nn.tanh(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input))`
`pred = tf.add(tf.matmul(weights_output, h), self.b2)`
	`Testing on dev set at step  1000`
	`UAS: 50.08849116334721`
	`UASnoPunc: 53.30639235855988`
	`LAS: 41.062392501931846`
	`LASnoPunc: 43.695246707737525`

<table>
<tr><td></td><td>UEM: 2.764705882352941
UEMnoPunc: 3.0
ROOT: 24.647058823529413</td></tr>
<tr><td># Tanh non-linearity
h = tf.nn.tanh(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input))
pred = tf.matmul(weights_output, h)</td><td>Testing on dev set at step 1000
UAS: 43.29087419298552
UASnoPunc: 46.741649239812354
LAS: 29.847695490689734
LASnoPunc: 31.97874865766122

UEM: 1.7647058823529411
UEMnoPunc: 1.7647058823529411
ROOT: 14.117647058823529</td></tr>
<tr><td># Relu non-linearity
h = tf.nn.relu(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input))
pred = tf.add(tf.matmul(weights_output, h), self.b2)</td><td>Testing on dev set at step 1000
UAS: 49.78936610414537
UASnoPunc: 53.077488272198046
LAS: 40.676022633796144
LASnoPunc: 43.33069575538349

UEM: 2.5294117647058822
UEMnoPunc: 2.6470588235294117
ROOT: 23.470588235294116</td></tr>
<tr><td># Relu non-linearity
h = tf.nn.relu(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input))
pred = tf.matmul(weights_output, h)</td><td>Testing on dev set at step 1000
UAS: 40.685993469102876
UASnoPunc: 44.042841801842535
LAS: 31.60256250467383
LASnoPunc: 34.3808285762731

UEM: 1.7647058823529411
UEMnoPunc: 1.7647058823529411
ROOT: 11.294117647058824</td></tr>
<tr><td># two layers
#layer - 1
z1 = tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input)
h1 = tf.pow(z1, 3)

mean = 0.1
stddev = math.sqrt(1.0 / parsing_system.numTransitions())
weights_input_2 = tf.Variable(tf.random_normal([Config.hidden_size, Config.hidden_size], mean=mean, stddev=stddev))
biases_input_2 = tf.Variable(tf.zeros([Config.hidden_size, 1]))

#layer - 2
z2 = tf.add(tf.matmul(weights_input_2, h1), biases_input_2)
h2 = tf.pow(z2, 3)

#output layer
pred = tf.matmul(weights_output, h2)</td><td>Testing on dev set at step 1000
UAS: 16.466834509060998
UASnoPunc: 16.28610184818855
LAS: 2.9264401625246155
LASnoPunc: 3.3176962640592325

UEM: 0.5882352941176471
UEMnoPunc: 0.5882352941176471
ROOT: 2.588235294117647</td></tr>
<tr><td># three layer
#layer - 1
z1 = tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input)
h1 = tf.pow(z1, 3)

mean = 0.1
stddev = math.sqrt(1.0 / parsing_system.numTransitions())

weights_input_2 = tf.Variable(tf.random_normal([Config.hidden_size, Config.hidden_size], mean=mean, stddev=stddev))
biases_input_2 = tf.Variable(tf.zeros([Config.hidden_size, 1]))

#layer - 2
z2 = tf.add(tf.matmul(weights_input_2, h1), biases_input_2)
h2 = tf.nn.relu(z2)</td><td>Testing on dev set at step 1000
UAS: 16.466834509060998
UASnoPunc: 16.28610184818855
LAS: 2.9264401625246155
LASnoPunc: 3.3176962640592325

UEM: 0.5882352941176471
UEMnoPunc: 0.5882352941176471
ROOT: 2.588235294117647</td></tr>
</table>

weights_input_3 =
tf.Variable(tf.random_normal([Config.hidden_size,
Config.hidden_size], mean=mean, stddev=stddev))
biases_input_3 = tf.Variable(tf.zeros([Config.hidden_size,
1]))
 	
#layer - 3
z3 = tf.add(tf.matmul(weights_input_3, h2),
biases_input_3)
h3 = tf.nn.tanh(z3)
 	
#output layer
pred = tf.matmul(weights_output, h3)	

#layer - 1
z1 = tf.add(tf.matmul(weights_input, tf.transpose(embed)),
biases_input)
h1 = tf.pow(z1, 3)
 	
mean = 0.1
stddev = math.sqrt(1.0 / parsing_system.numTransitions())
 	
weights_input_2 =
tf.Variable(tf.random_normal([Config.hidden_size,
Config.hidden_size], mean=mean, stddev=stddev))
biases_input_2 = tf.Variable(tf.zeros([Config.hidden_size,
1]))
 	
#layer - 2
z2 = tf.add(tf.matmul(weights_input_2, h1),
biases_input_2)
h2 = tf.pow(z2, 3)
 	
weights_input_3 =
tf.Variable(tf.random_normal([Config.hidden_size,
Config.hidden_size], mean=mean, stddev=stddev))
biases_input_3 = tf.Variable(tf.zeros([Config.hidden_size,
1]))
 	
#layer - 3
z3 = tf.add(tf.matmul(weights_input_3, h2),
biases_input_3)
h3 = tf.pow(z3, 3)
 	
#output layer
pred = tf.matmul(weights_output, h3)
	Testing on dev set at step 1000
	UAS: 26.323005209761448
	UASnoPunc: 29.370372463686202
	LAS: 0.019941670613455642
	LASnoPunc: 0.022607810998700052
	
	UEM: 0.47058823529411764
	UEMnoPunc: 0.47058823529411764
	ROOT: 0.47058823529411764

# Cube non-linearity 	
# without gradient cliping 	
h = tf.pow(tf.add(tf.matmul(weights_input,
tf.transpose(embed)), biases_input), 3)
pred = tf.add(tf.matmul(weights_output, h), self.b2
	NAN

# Cube non-linearity 	
# change the learning rate, 0.1 to 0.05 	
h = tf.pow(tf.add(tf.matmul(weights_input,
tf.transpose(embed)), biases_input), 3)
pred = tf.add(tf.matmul(weights_output, h), self.b2
	Testing on dev set at step 1000
	UAS: 60.51299947653115
	UASnoPunc: 63.50816707172328
	LAS: 55.86160480594262
	LASnoPunc: 58.288588707398404
	
	UEM: 5.176470588235294
	UEMnoPunc: 5.647058823529412
	ROOT: 53.1764705882353

# Cube non-linearity 	
# learning rate, 0.05 	
# mat_iter, 2001 	
h = tf.pow(tf.add(tf.matmul(weights_input,
tf.transpose(embed)), biases_input), 3)
pred = tf.add(tf.matmul(weights_output, h), self.b2
	Testing on dev set at step 2000
	UAS: 76.15225465513373
	UASnoPunc: 78.96908381845928
	LAS: 72.7447216890595
	LASnoPunc: 75.16814559430283
	
	UEM: 15.882352941176471
	UEMnoPunc: 16.764705882352942

	ROOT: 70.17647058823529

# Cube non-linearity 	
# learning rate, 0.05 	
# mat_iter, 3001 	
h = tf.pow(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input), 3)
pred = tf.add(tf.matmul(weights_output, h), self.b2
	Testing on dev set at step 3000
	UAS: 77.2216267417803
	UASnoPunc: 79.81122477816085
	LAS: 73.99356881122716
	LASnoPunc: 76.18549708924434
	
	UEM: 17.11764705882353
	UEMnoPunc: 18.529411764705884
	ROOT: 76.47058823529412

# Cube non-linearity # learning rate, 0.05 # mat_iter, 3001 # no gradient clipping h = tf.pow(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input), 3) pred = tf.add(tf.matmul(weights_output, h), self.b2	NAN
# Cube non-linearity # learning rate, 0.05 # mat_iter, 3001 # mean 0.0 h = tf.pow(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input), 3) pred = tf.add(tf.matmul(weights_output, h), self.b2	Testing on dev set at step 3000 UAS: 77.18423610938007 UASnoPunc: 79.50319335330357 LAS: 73.66453124610514 LASnoPunc: 75.56378228678008 UEM: 16.176470588235293 UEMnoPunc: 17.88235294117647 ROOT: 75.82352941176471
# Cube non-linearity # learning rate, 0.1 # mat_iter, 3001 # mean 0.0 h = tf.pow(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input), 3) pred = tf.add(tf.matmul(weights_output, h), self.b2	Testing on dev set at step 3000 UAS: 79.78163870678266 UASnoPunc: 82.07200587803086 LAS: 76.75798290001745 LASnoPunc: 78.67518227547617 UEM: 19.0 UEMnoPunc: 20.88235294117647 ROOT: 78.70588235294117

2. **Observations**:
    - I tried with the cube non-linearity, sigmoid, tanh, relu, 2 layers (cube and tanh), 3 layers (all cube) and 3 layers (cube, relu, tanh).
    - Best results I achieved with the cube non-linearity.
    - I also tried to change some other parameters, including learning rate, mean and number of iterations.
    - The tanh, sigmid and relu sometimes gives better average loss, but in the dev set results are not good.
    - For the multiple layers, average loss is also much larger, then the single layer.
    - With cube non-linearity and change of the mean, learning rate and number of iterations, affect the results most.

3. **Best Model Configurations:**

```
UNKNOWN = "UNK"
ROOT = "ROOT"
NULL = "NULL"
NONEXIST = -1

max_iter = 3001
batch_size = 10000
hidden_size = 200
embedding_size = 50
learning_rate = 0.1
display_step = 100
validation_step = 400
n_Tokens = 48
lam = 1e-8

mean = 0.0
```

```
h = tf.pow(tf.add(tf.matmul(weights_input, tf.transpose(embed)), biases_input), 3)
pred = tf.matmul(weights_output, h)
```

4. **Results:**

```
Testing on dev set at step  3000
UAS: 79.78163870678266
UASnoPunc: 82.07200587803086
LAS: 76.75798290001745
LASnoPunc: 78.67518227547617

UEM: 19.0
UEMnoPunc: 20.88235294117647
ROOT: 78.70588235294117
```

5. **Gradient clipping:** When gradients are being propagated back in the time, they can vanish because they are being multiplied by the number less then one, which is called the Vanishing Gradient problem. The opposite can also happen, when the gradient is being multiplied by the numbers larger then 1. This is known as Exploding Gradients. Gradient clipping clip the gradients between two numbers to prevent them form being vanish or being explode. I run the code without gradient clipping (in DependecyParser.py, enable line 118 and disable line 121-123) but got nan after step 0. I tried with different model and configurations, but results are same (nan).