Deep Learning: Homework 3

Name: Yifan Shen

SN: 15015762

Start date: *12th Feb 2018*

Due date: *12nd March 2018, 11:55 pm*

How to Submit

When you have completed the exercises and everything has finsihed running, click on 'File' in the menu-bar and then 'Download .ipynb'. This file must be submitted to Moodle named as **studentnumber_DL_hw3.ipynb** before the deadline above.

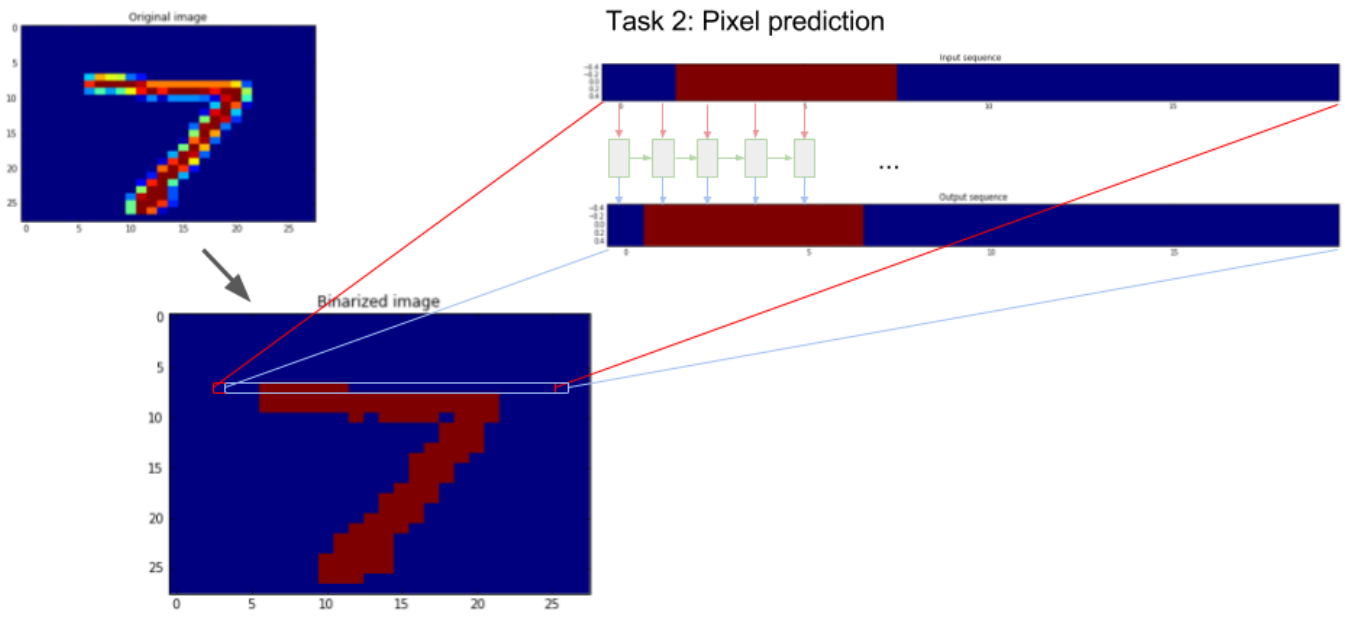
Also send a **sharable link** to the notebook at the following email: ucl.coursework.submit@gmail.com. You can also make it sharable via link to everyone, up to you.

Please compile all results (table in Q2) and all answers to the understanding/analysis results questions (Q1 and Q3), into a PDF. Name convention: **studentnumber_DL_hw3.pdf**. Do not include any of the code (we will use the notebook for that).

Page limit: 7 pg (w/o the bonus question).

MNIST as a sequence

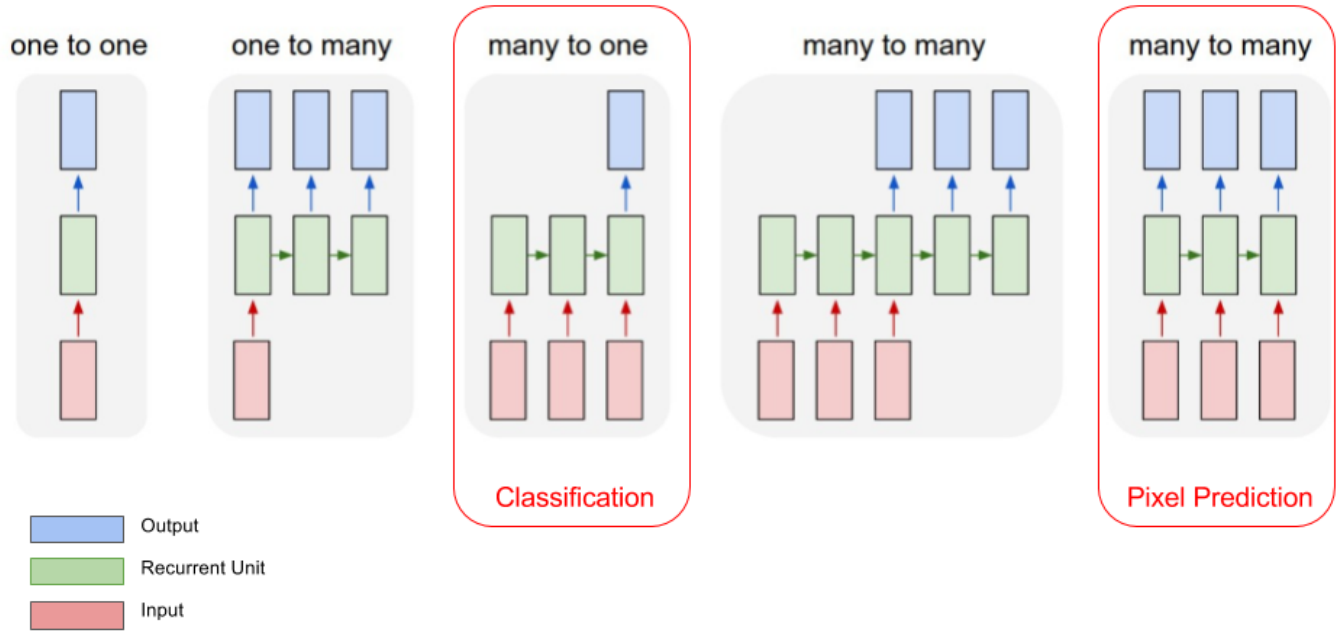
In this assignment we will be using the MNIST digit dataset (https://yann.lecun.com/exdb/mnist/). The dataset contains images of hand-written digits ($0-9$), and the corresponding labels. The images have a resolution of 28×28 pixels. This is the same dataset as in Assignment 1, but we will be using this data a bit differently this time around. Since this assignment will be focusing on recurrent networks that model sequential data, we will be looking at each image as a sequence: the networks you train will be "reading" the image one row at a time, from top to bottom (we could even do pixel-by-pixel, but in the interest of time we'll do row-by-row which is faster). Also, we will work with a binarized version of MNIST -- we constrain the values of the pixels to be either 0 or 1. You can do this by applying the method `binarize`, defined below, to the raw images.



- We take the MNIST images, binarise them, and interpret them as a sequence of pixels from top-left to bottom-right. ("Task 2" refers to the next homework, wherein you will be using the sequence for pixel prediction).

Recurrent Models for MNIST

As discussed in the lectures, there are various ways and tasks for which we can use recurrent models. A depiction of the most common scenarios is available in the Figure below. In this assignment and the following one we will look at two of these forms: **many-to-one** (sequence to label/decision) and the **many-to-many** scenario where the model receives an input and produces an output at every time step. You will use these to solve the following tasks: i) classification (*this homework*), ii) pixel prediction (*next homework*) and iii) in-painting (*next homework*).



- (Figure adapted from Karpathy's The Unreasonable Effectiveness of Recurrent Neural Networks (http://karpathy.github.io/2015/05/21/rnn-effectiveness)). You will be implementing variants of *many-to-one* for classification (in this homework), and *many-to-many* for prediction (in the next homework).

In [0]:

```
1
```

18## Q1: Understanding LSTM vs GRU (30 pts) Before going deeper into your practical tasks, take some time to revise and make sure you understand the two major types of recurrent cells you will be using in this assignment: Long-Short Term Memory Units (LSTM) first introduced by Hochreiter and Schmidhuber [1997] and the more recent Gated Recurrent Units (GRU) by Cho et al. [2014]. Once you have done this, answer the following questions:

Vanilla RNN:

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1})$$

LSTM:

$$\text{Input gate: } i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1})$$

$$\text{Forget gate: } f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1})$$
$$\text{Output gate: } o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1})$$
$$\text{Memory cell: } \tilde{c}_t = \tanh(W^{(c)}x_t + U^{(c)}h_{t-1})$$
$$\text{Final memory: } c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$\text{Hidden state: } h_t = o_t \circ \tanh(c_t)$$

GRU:

$$\text{Update gate: } z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

$$\text{Reset gate: } r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$
$$\text{Intermediate memory content: } \tilde{h_t} = \tanh(Wx_t + r_t \circ Uh_{t-1})$$
$$\text{Hidden state: } h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h_t}$$

1. Can LSTMs (and respectively GRUs) just store the current input in the state (c_t for LSTM and h_t for GRU, in the class notation) for the next step? If so, give the gates activation that would enable this behaviour. If not, explain why not. [10 pts]

   Answer 1: Both LSTM and GRU can do this. For LSTM, $i_t > 0$ and $f_t = 0$. For GRU, $z_t = 1$ and $r_t = 0$
2. Can LSTMs (and respectively GRUs) just store a previous state into the current state and ignore the current input? If so, give the gates' activation that would enable this. If not, explain why not. [10 pts]

   Answer 2: Both LSTM and GRU can do this. For LSTM, $i_t = 0$ and $f_t > 0$. For GRU, $z_t = 0$ and $r_t > 0$
3. Are GRUs a special case of LSTMs? If so, give the expression of the GRU gates in term of LSTM's gates (o_t, i_t, f_t). If not, give a counter-example. Assume here the same input. [10 pts]

   Answer 3: GRUs are not a special case of LSTMs.

   We propose a special case of GRU that cannot attained by LSTM.

   Let $z_i = 1, \forall i \in 1, 2, \ldots, t$, then we have $h_t = h_{t-1} = h_{t-2} = \ldots = h_1$. If we want match the same result from LSTM side, we need $h_t = o_t \circ \tanh(f_t \circ c_{t-1} + i_t \circ \tanh(W^{(c)}x_t + U^{(c)}h_{t-1})) = h_{t-1}$. Thus, in theory it could be possible for the LSTM attain this result $h_t = h_{t-1} = \ldots = h_1$ by tuning the weights of the two tanh layers so that they operate an identy transformation, and by forgetting everything that is in the cell state and replacing it with the last output. Though, such tuning would be valid only for a specifcic value of the output. Therefore, in general, the LSTM cannot decide, at some point, to start reproducing the last output.

Word limit: 1000 words or less

Q2: Implementation. Line-by-Line MNIST Classification (50 pts)

In this part you will train a number of many-to-one recurrent models that takes as input: an image (or part of an image) as a sequence (row by row) and after the last input row produces, as output, a probability distribution over the 10 possible labels ($0 - 9$). The models will be trained using a cross-entropy loss function over these output probabilities.

Optimization

Use the Adam optimizer (with default settings other than the learning rate) for training.

[Optional] Sometimes dropout has been shown to be beneficial in training recurrent models, so feel free to use it or any other form of regularization that seems to improve performance. It might be also worth trying out batch-normalization. Reference (https://arxiv.org/pdf/1603.09025.pdf).

Models: Your models will have the following structure:

1. [(Red Block)] The *input* (current binarised row of pixels) can be fed directly into the recurrent connection without much further pre-processing. The only thing you need to do is have an affine transformation to match the dimensionality of the recurrent unit, i.e. one of $(32, 64, 128)$.

2. [(Blue Block)] The *output* (probabilities over the 10 classes) is produced by looking at the last output of the recurrent units, transforming them via an affine transformation.
3. [(Green Block)] For the *recurrent* part of the network, please implement and compare the following architectures:

    - LSTM with 32, 64, 128 units. [15 pts]
    - GRU with 32, 64, 128 units. [15 pts]
    - stacked LSTM: 3 recurrent layers with 32 units each. [10 pts]
    - stacked GRU: 3 recurrent layers with 32 units each. [10 pts]

Your network should look like:

$$\text{Input} \Rightarrow \text{RNN cell} \Rightarrow \text{Relu} \Rightarrow \text{Fully connected} \Rightarrow \text{Relu} \Rightarrow \text{Fully connected} \Rightarrow \text{Output}$$

You might find the function `tf.nn.dynamic_rnn` useful.

Hyper-parameters

For all cases train the model with these hyper-parameter settings:

- *num_epochs*=10, *learning_rate*=0.001, *batch_size*=256, *fully_connected_hidden_units=64*

With these hyper-parameters you should be comfortably above 95% test set accuracy on all tasks. (Feel free to try other settings, there are certainly better choices, but please report the results with these exact hyper-parameters). Please report the *cross-entropy* and the *classification accuracy* for the *test set* of the models trained. Use the `plot_summary_table` method below to format the table.

##Q3: Analyse the results (20 pts + 10 pts)

1. How does this compare with the results you obtained in the first assignment(DL1), when training a model that "sees" the entire image at once? Explain differences. [5 pts]

Answer: The results found by LSTM and GRU are slightly better than the normal neural networks (first three settings in the first assignment) but slightly worse than CNN.

Reason: Normal neural networks "sees" the entrie image at once while RNNs "sees" a row of the image each time. Therefore RNNs captures more information by "looking" the image step by step (a row per time, for 28 times), resulting better performance than normal neural networks. For CNNs, it "sees" a 3x3 window of the image everytime, and all pixels have been "seen" for several times. Therefore CNNs stores much even more information than RNNs, resulting the best performance.

1. Let us take a look closer look at one of the trained models: say GRU (32). Plot the outputs of the RNN layer and hidden state over time. In particular, look at the first 3-5 time steps. Plot the input image along side. You can use `python plt.imshow(output_GRU_over_time)` for these, where `output_GRU_over_time.shape` is (T=28,hidden_units) dimensional. What do you observe? Show at least one pair of these plots to support your observation(s). [5 pts]

Answer: In general, the stripes of "output_GRU_over_time" plot is vague at the very beginning, but becomes clear with time steps. At the very beginning, the input are all 0s (have not reach the true number yet), which are meaningless. Therefore, it is nor surprising that the model cannot distinguish the digit at the beginning and resulting very small valus in "output_GRU_over_time".

1. Now, look at the last 3-5 time steps. What do you observe? When is the classification decision made? To validate your answer to the second part of the question, provide the classification predictions for the last 5 time steps -- that is, pretend whatever the output of the GRU is at that time step is in fact the last output in the computation, and feed that into the classification mapping. [10 pts]

Answer: The stripes of "output_GRU_over_time" of the last 3-5 times becomes very clear, which means the classification has been made. In the final part of my code, I have printed out the decision made each time. In this case, our label is 5, we can see that the decision is made at the 18th step. I have tried a couple of different examples, the decisions are usually made at the later steps (around 16 to 25)

Word limit: 500 words or less

[Bonus] Let's looking inside the computation. Take one of the previous LSTM models (for simplicity pick one of the one-layer recurrence models) and track the status of the gates (o_t, i_t, f_t) over time. Note that if you used the provided RNNCell wrappers (BasicLSTMCell and co) in tensorflow, these will keep this information hidden, so you will need to implement your own version of this recurrence layer, or mirror the one in tensorflow, but now exposing these hidden variables. A bit of warning: this is not trivial and will require some thinking on the coding side, but it will also provide you with a more informative way of visulazing the inner computation. [10 pts]

Word limit: 300 words or less

Imports and utility functions (do not modify!)

In [0]:

```
# Import useful libraries.
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np

# Binarize the images
def binarize(images, threshold=0.1):
  return (threshold < images).astype('float32')

# Import dataset with one-hot encoding of the class labels.
def get_data():
  return input_data.read_data_sets("MNIST_data/", one_hot=True)

# Placeholders to feed train and test data into the graph.
# Since batch dimension is 'None', we can reuse them both for train and eval.
def get_placeholders():
  x = tf.placeholder(tf.float32, [None, 784])
  y = tf.placeholder(tf.float32, [None, 10])
  return x, y

# Generate summary table of results. This function expects a dict with the
# following structure: keys of 'LSTM' or 'GRU' and the values for each key are a
# list of tuples consisting of (test_loss, test_accuracy), and the list is
# ordered as the results from 32 units, 64 units, 128 units, 3 x 32 units, i.e.
# {
#  'LSTM': [(loss,acc), (loss, acc), (loss, acc), (loss, acc)]
#  'GRU': [(loss,acc), (loss, acc), (loss, acc), (loss, acc)]
# }
def plot_summary_table(experiment_results):
  # Fill Data.
  cell_text = []
  columns = ['(1 layer, 32 units)', '(1 layer, 64 units)', '(1 layer, 128 units
             , '(3 layers, 32 units)']
  for k, v in experiment_results.items():
    rows = ['Test loss', 'Test accuracy']
    cell_text=[[],[]]
    for (l, _) in v:
      cell_text[0].append(str(l))
    for (_, a) in v:
      cell_text[1].append(str(a))

    fig=plt.figure(frameon=False)
    ax = plt.gca()
    the_table = ax.table(
      cellText=cell_text,
      rowLabels=rows,
      colLabels=columns,
      loc='center')
    the_table.scale(2, 8)
    # Prettify.
    ax.patch.set_facecolor('None')
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    ax.text(-0.73, 0.9, k, fontsize=18)
```

Train Models

Generate summary table of results. This function expects a dict with the following structure: keys of 'LSTM' or 'GRU' and the values for each key are a list of tuples consisting of (test_loss, test_accuracy), and the list is ordered as the results from 32 units, 64 units, 128 units, 3 x 32 units, i.e. expected dictionary (final performace only):

```
{
  'LSTM': [(loss,acc), (loss, acc), (loss, acc), (loss, acc)]
  'GRU': [(loss,acc), (loss, acc), (loss, acc), (loss, acc)]
}
```

In [0]:

```
# Your models here

num_epochs = 10
learning_rate = 0.001
batch_size = 256
fully_connected_hidden_units = 64
num_classes = 10

initializer = tf.contrib.layers.xavier_initializer()

def RNN(RNN_type, num_hidden_unit, num_layers):
  time_steps = 28
  num_units = 28
  tf.reset_default_graph()

  mnist = get_data()
  eval_mnist = get_data()
  x, y_ = get_placeholders()
  x_reshape = tf.reshape(x, [-1, 28, 28])

  if RNN_type == 'LSTM':
    RNN_layers = [tf.contrib.rnn.BasicLSTMCell(size) for size in \
                  ([num_hidden_unit] * num_layers)]
  elif RNN_type == 'GRU':
    RNN_layers = [tf.contrib.rnn.GRUCell(size) for size in \
                  ([num_hidden_unit] * num_layers)]

  cell = tf.contrib.rnn.MultiRNNCell(RNN_layers)

  rnn_outputs, state = tf.nn.dynamic_rnn(cell, x_reshape, dtype=tf.float32) \
  # rnn_outputs: batch_size * 28 * num_hidden
  # choose the last one by using -1 (many to one)
  rnn_outputs_unstack = tf.unstack(rnn_outputs, time_steps, 1)[-1] \
  # rnn_outputs_unstack: batch_size * num_hidden

  W_1 = tf.Variable(initializer([num_hidden, fully_connected_hidden_units]))
  b_1 = tf.Variable(initializer([fully_connected_hidden_units]))
  hid_1 = tf.nn.relu(tf.matmul(rnn_outputs_unstack, W_1) + b_1)

  W_2 = tf.Variable(initializer([fully_connected_hidden_units, \
                                 fully_connected_hidden_units]))
  b_2 = tf.Variable(initializer([fully_connected_hidden_units]))
  hid_2 = tf.nn.relu(tf.matmul(hid_1, W_2) + b_2)

  W_3 = tf.Variable(initializer([fully_connected_hidden_units, num_classes]))
  b_3 = tf.Variable(initializer([num_classes]))
  y = tf.matmul(hid_2, W_3) + b_3

  # loss
  cross_entropy = \
  tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y_, logits = y

  # update
  train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

  # evaluation metric
  correct_prediction = tf.equal(tf.argmax(tf.nn.softmax(y),1), tf.argmax(y_,1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

```

```
  i, train_accuracy, test_accuracy = 0, [], []

  with tf.train.MonitoredSession() as sess:

    while mnist.train.epochs_completed < num_epochs:

      i += 1
      batch_xs, batch_ys = mnist.train.next_batch(batch_size)

      sess.run(train_step, feed_dict = {x: binarize(batch_xs), y_: batch_ys})

      if i % (100) == 0:

        test_acc = accuracy.eval(session = sess, feed_dict= \
                        {x: binarize(mnist.test.images), y_: mnist.test.labels})
        loss = sess.run(cross_entropy, \
                        feed_dict = {x: binarize(batch_xs), y_: batch_ys})
        print('%d th test acc: %s loss: %s' %(i, test_acc, loss))

  return (loss, test_acc)
```

In [41]:

```
settings = [\
            ['LSTM',32,1],\
            ['LSTM',64,1],\
            ['LSTM',128,1],\
            ['LSTM',32,3],\
            ['GRU',32,1],\
            ['GRU',64,1],\
            ['GRU',128,1],\
            ['GRU',32,3]]

experiment_results = {'LSTM':[], 'GRU':[]}

for RNN_type, num_hidden, num_layers in settings:

  print(RNN_type, num_hidden, num_layers)

  experiment_results[RNN_type].append(RNN(RNN_type, num_hidden, num_layers))

print(experiment_results)


```

```
LSTM 32 1
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
100 th test acc: 0.5623 loss: 1.2627294
200 th test acc: 0.7923 loss: 0.60181373
300 th test acc: 0.854 loss: 0.45864093
400 th test acc: 0.8835 loss: 0.44273466
500 th test acc: 0.9014 loss: 0.28687698
600 th test acc: 0.9156 loss: 0.29344687
700 th test acc: 0.9226 loss: 0.2072212
```

#Results

In [42]:

```
1 plot_summary_table(experiment_results)
```

LSTM

	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layers, 32 units)
Test loss	0.12693101	0.09858341	0.074853815	0.0649623
Test accuracy	0.9634	0.9772	0.9765	0.9709

GRU

	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layers, 32 units)
Test loss	0.09399286	0.027650137	0.017552938	0.05128681
Test accuracy	0.9696	0.9781	0.9837	0.9766

Analysis of results

In [41]:

```
num_epochs = 10
learning_rate = 0.001
batch_size = 256
fully_connected_hidden_units = 64
num_classes = 10

initializer = tf.contrib.layers.xavier_initializer()

num_hidden_unit = 64
num_layers = 1

RNN_type = 'GRU'

time_steps = 28
num_units = 28
tf.reset_default_graph()

mnist = get_data()
eval_mnist = get_data()
x, y_ = get_placeholders()
x_reshape = tf.reshape(x, [-1, 28, 28])

if RNN_type == 'LSTM':
  RNN_layers = [tf.contrib.rnn.BasicLSTMCell(size) for size in \
                ([num_hidden_unit] * num_layers)]
elif RNN_type == 'GRU':
  RNN_layers = [tf.contrib.rnn.GRUCell(size) for size in \
                ([num_hidden_unit] * num_layers)]

cell = tf.contrib.rnn.MultiRNNCell(RNN_layers)

rnn_outputs, state = tf.nn.dynamic_rnn(cell, x_reshape, dtype=tf.float32) \
# rnn_outputs: batch_size * 28 * num_hidden
# choose the last one by using -1 (many to one)
rnn_outputs_unstack = tf.unstack(rnn_outputs, time_steps, 1)[-1]\
# rnn_outputs_unstack: batch_size * num_hidden

W_1 = tf.Variable(initializer([num_hidden_unit, fully_connected_hidden_units]))
b_1 = tf.Variable(initializer([fully_connected_hidden_units]))
hid_1 = tf.nn.relu(tf.matmul(rnn_outputs_unstack, W_1) + b_1)

W_2 = tf.Variable(initializer([fully_connected_hidden_units, \
                               fully_connected_hidden_units]))
b_2 = tf.Variable(initializer([fully_connected_hidden_units]))
hid_2 = tf.nn.relu(tf.matmul(hid_1, W_2) + b_2)

W_3 = tf.Variable(initializer([fully_connected_hidden_units, num_classes]))
b_3 = tf.Variable(initializer([num_classes]))
y = tf.matmul(hid_2, W_3) + b_3

# loss
cross_entropy = \
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y_, logits = y)

# update
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

# evaluation metric
```

```
correct_prediction = tf.equal(tf.argmax(tf.nn.softmax(y),1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

i, train_accuracy, test_accuracy = 0, [], []

with tf.train.MonitoredSession() as sess:

  while mnist.train.epochs_completed < num_epochs:

    i += 1

    batch_xs, batch_ys = mnist.train.next_batch(batch_size)

    sess.run(train_step, feed_dict = {x: binarize(batch_xs), y_: batch_ys})

    if i % (2149) == 0: # the last training step

      test_acc = accuracy.eval(session = sess, \
            feed_dict= {x: binarize(mnist.test.images), y_: mnist.test.labels})
      loss = sess.run(cross_entropy, \
                      feed_dict = {x: binarize(batch_xs), y_: batch_ys})
      print('%d th test acc: %s loss: %s' %(i, test_acc, loss))
      rnn_outputs_run = sess.run(rnn_outputs, feed_dict = \
                                 {x: binarize(batch_xs), y_: batch_ys})
      output_GRU_over_time = rnn_outputs_run[1,:,:]
      binarized_input_image = binarize(batch_xs[1]).reshape(28,28)

      # feed "output_GRU_over_time" to the trained model:

      for j in range(output_GRU_over_time.shape[0]): # loop through time steps

        decision_result = sess.run(y, \
       feed_dict = {rnn_outputs_unstack: output_GRU_over_time[j].reshape(1,-1)})
        decision = np.argmax(decision_result)

        print(j + 1,'th time step, decision result is', decision)

```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
2149 th test acc: 0.9779 loss: 0.10354345
1 th time step, decision result is 5
2 th time step, decision result is 5
3 th time step, decision result is 5
4 th time step, decision result is 5
5 th time step, decision result is 2
6 th time step, decision result is 9
7 th time step, decision result is 7
8 th time step, decision result is 7
9 th time step, decision result is 7
10 th time step, decision result is 7
11 th time step, decision result is 7
```

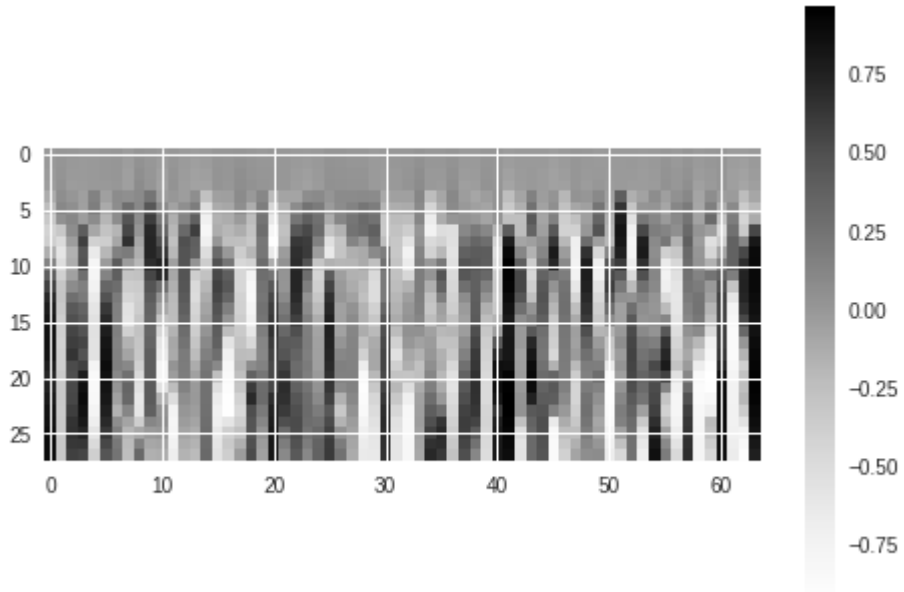
```
12 th time step, decision result is 0
13 th time step, decision result is 0
14 th time step, decision result is 0
15 th time step, decision result is 0
16 th time step, decision result is 0
17 th time step, decision result is 0
18 th time step, decision result is 5
19 th time step, decision result is 5
20 th time step, decision result is 5
21 th time step, decision result is 5
22 th time step, decision result is 5
23 th time step, decision result is 5
24 th time step, decision result is 5
25 th time step, decision result is 5
26 th time step, decision result is 5
27 th time step, decision result is 5
28 th time step, decision result is 5
```

In [42]:

```
1 plt.imshow(output_GRU_over_time)
2 plt.colorbar()
```

Out[42]:

```
<matplotlib.colorbar.Colorbar at 0x7f27893bd588>
```

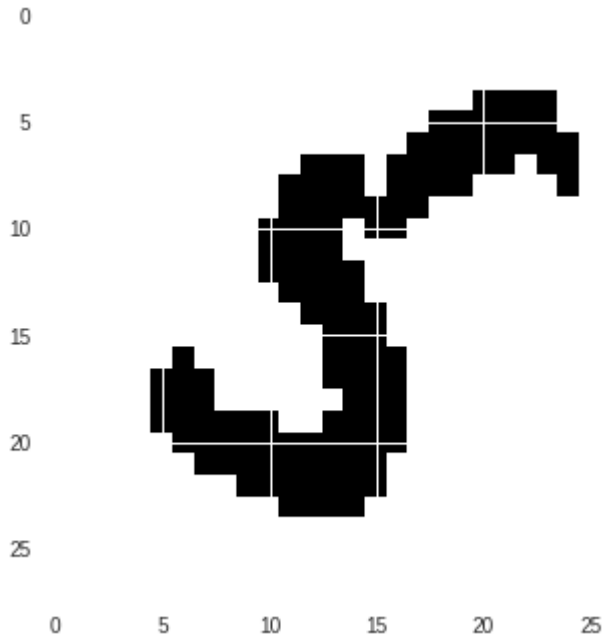


In [43]:

```
1 plt.imshow(binarized_input_image)
```

Out[43]:

```
<matplotlib.image.AxesImage at 0x7f2789dec400>
```

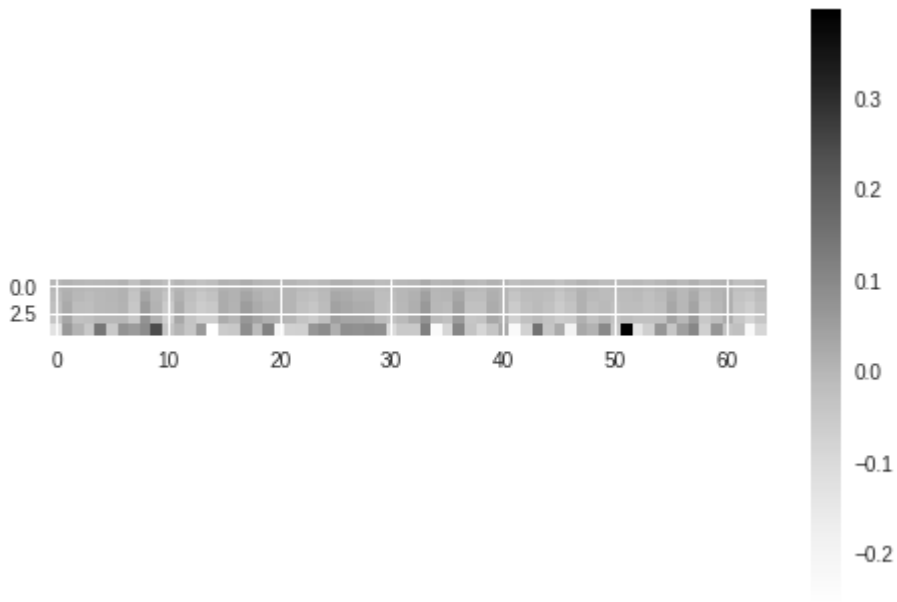


In [44]:

```
1 plt.imshow(output_GRU_over_time[:5,:])
2 plt.colorbar()
```

Out[44]:

```
<matplotlib.colorbar.Colorbar at 0x7f2789f52dd8>
```

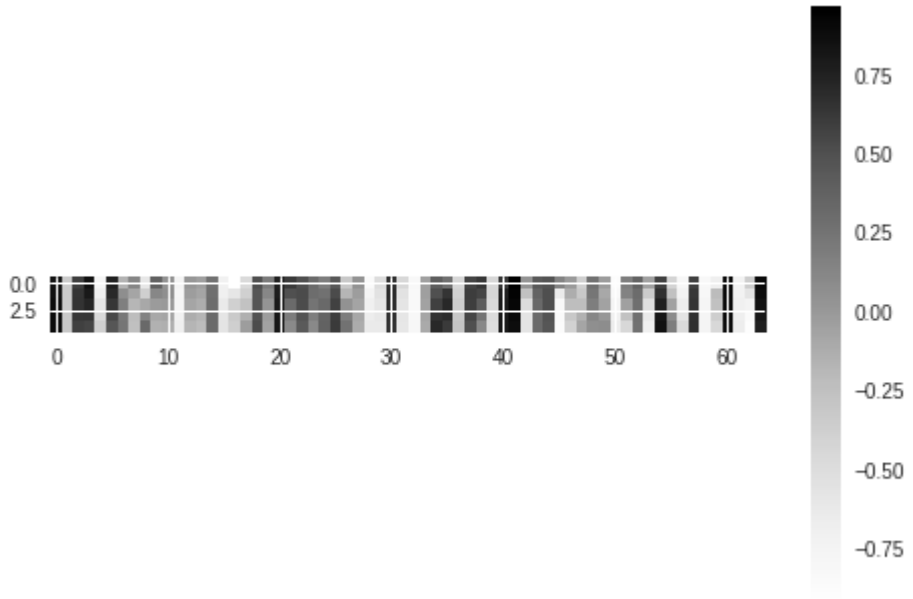


In [45]:

```
plt.imshow(output_GRU_over_time[-5:,:])
plt.colorbar()
```

Out[45]:

```
<matplotlib.colorbar.Colorbar at 0x7f27899c8470>
```



Bonus Question:

In [0]:

```
class LSTM_cell(object):

    """
    LSTM cell object which takes 3 arguments for initialization.
    input_size = Input Vector size
    hidden_layer_size = Hidden layer size
    target_size = Output vector size

    """

    def __init__(self, input_size, hidden_layer_size , x_batch_input):

        # Initialization of given values
        self.input_size = input_size
        self.hidden_layer_size = hidden_layer_size
        self.x_batch_input = x_batch_input

        self.processed_input = self.x_batch_input

        # Weights and Bias for input and hidden tensor
        self.Wi = tf.Variable(tf.zeros(
            [self.input_size, self.hidden_layer_size]))
        self.Ui = tf.Variable(tf.zeros(
            [self.hidden_layer_size, self.hidden_layer_size]))
        self.bi = tf.Variable(tf.zeros([self.hidden_layer_size]))


        self.Wf = tf.Variable(tf.zeros(
            [self.input_size, self.hidden_layer_size]))
        self.Uf = tf.Variable(tf.zeros(
            [self.hidden_layer_size, self.hidden_layer_size]))
        self.bf = tf.Variable(tf.zeros([self.hidden_layer_size]))


        self.Wog = tf.Variable(tf.zeros(
            [self.input_size, self.hidden_layer_size]))
        self.Uog = tf.Variable(tf.zeros(
            [self.hidden_layer_size, self.hidden_layer_size]))
        self.bog = tf.Variable(tf.zeros([self.hidden_layer_size]))


        self.Wc = tf.Variable(tf.zeros(
            [self.input_size, self.hidden_layer_size]))
        self.Uc = tf.Variable(tf.zeros(
            [self.hidden_layer_size, self.hidden_layer_size]))
        self.bc = tf.Variable(tf.zeros([self.hidden_layer_size]))

        '''
        Initial hidden state's shape is [1,self.hidden_layer_size]
        In First time stamp, we are doing dot product with weights to
        get the shape of [batch_size, self.hidden_layer_size].
        For this dot product tensorflow use broadcasting. But during
        Back propagation a low level error occurs.
        So to solve the problem it was needed to initialize initial
        hiddden state of size [batch_size, self.hidden_layer_size].
        So here is a little hack !!!! Getting the same shaped
        initial hidden state of zeros.
        '''
```

```
60
61          self.initial_hidden = self.x_batch_input[0, :, :]
62          self.initial_hidden= tf.matmul(
63              self.initial_hidden, tf.zeros([input_size, hidden_layer_size]))
64
65
66          self.initial_hidden=tf.stack([self.initial_hidden,self.initial_hidden,\
67                                        self.initial_hidden,self.initial_hidden,\
68                                        self.initial_hidden])
69
70      # Function for LSTM cell.
71      def Lstm(self, previous_hidden_memory_tuple, x):
72          """
73          This function takes previous hidden state and memory tuple with input a
74          outputs current hidden state.
75          """
76
77          previous_hidden_state, c_prev,_,_,_=tf.unstack(previous_hidden_memory_t
78
79          #Input Gate
80          i= tf.sigmoid(
81              tf.matmul(x,self.Wi)+tf.matmul(previous_hidden_state,self.Ui) \
82              + self.bi
83          )
84
85          #Forget Gate
86          f= tf.sigmoid(
87              tf.matmul(x,self.Wf)+tf.matmul(previous_hidden_state,self.Uf) \
88              + self.bf
89          )
90
91          #Output Gate
92          o= tf.sigmoid(
93              tf.matmul(x,self.Wog)+tf.matmul(previous_hidden_state,self.Uog) \
94              + self.bog
95          )
96
97          #New Memory Cell
98          c_= tf.nn.tanh(
99              tf.matmul(x,self.Wc)+tf.matmul(previous_hidden_state,self.Uc) \
100             + self.bc
101         )
102
103         #Final Memory cell
104         c= f*c_prev + i*c_
105
106         #Current Hidden state
107         current_hidden_state = o*tf.nn.tanh(c)
108
109         return tf.stack([current_hidden_state,c,i,f,o]) # stack everything toge
110
111     # Function for getting all hidden state.
112     def get_states(self):
113         """
114         Iterates through time/ sequence to get all hidden state
115         """
116         # Getting all hidden state throuh time
117         all_hidden_states = tf.scan(self.Lstm,
118                                     self.x_batch_input,
119                                     initializer=self.initial_hidden,
120                                     name='states')
```

```
121 
122         return all_hidden_states
123 
124 # Function to convert batch input data to use scan ops of tensorflow.
125 def process_batch_input_for_RNN(batch_input):
126     """
127     Process tensor of size [5,3,2] to [3,5,2]
128     """
129     batch_input_ = tf.transpose(batch_input, perm=[2, 0, 1])
130     X = tf.transpose(batch_input_)
131 
132     return X
133 
```

In [60]:

```
num_epochs = 10
learning_rate = 0.001
batch_size = 256
fully_connected_hidden_units = 64
num_classes = 10

initializer = tf.contrib.layers.xavier_initializer()

num_hidden_unit = 32
num_layers = 1

time_steps = 28
num_units = 28
tf.reset_default_graph()

mnist = get_data()
eval_mnist = get_data()
x, y_ = get_placeholders()
x_reshape = tf.reshape(x, [-1, 28, 28])
x_processed = process_batch_input_for_RNN(x_reshape)

single_lstm_cell = LSTM_cell(num_units, 32, x_processed)

rnn_outputs_full = single_lstm_cell.get_states() # rnn_outputs_full: 28 * 5 * 2

rnn_outputs = rnn_outputs_full[:,0,:,:]  # rnn_outputs: 28 * 256 * 32

rnn_outputs_unstack = rnn_outputs[-1] # rnn_outputs_unstack: 256 * 32

W_1 = tf.Variable(initializer([num_hidden_unit, fully_connected_hidden_units]))
b_1 = tf.Variable(initializer([fully_connected_hidden_units]))
hid_1 = tf.nn.relu(tf.matmul(rnn_outputs_unstack, W_1) + b_1)

W_2 = tf.Variable(initializer([fully_connected_hidden_units, \
                               fully_connected_hidden_units]))
b_2 = tf.Variable(initializer([fully_connected_hidden_units]))
hid_2 = tf.nn.relu(tf.matmul(hid_1, W_2) + b_2)

W_3 = tf.Variable(initializer([fully_connected_hidden_units, num_classes]))
b_3 = tf.Variable(initializer([num_classes]))
y = tf.matmul(hid_2, W_3) + b_3

# loss
cross_entropy = \
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y_, logits = y)

# update
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

# evaluation metric
correct_prediction = tf.equal(tf.argmax(tf.nn.softmax(y),1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

i, train_accuracy, test_accuracy = 0, [], []

with tf.train.MonitoredSession() as sess:

  while mnist.train.epochs_completed < num_epochs:

```

```
    i += 1

    batch_xs, batch_ys = mnist.train.next_batch(batch_size)

    sess.run(train_step, feed_dict = {x: binarize(batch_xs), y_: batch_ys})

    if i % (2149) == 0: # the last training step

      test_acc = accuracy.eval(session = sess, \
             feed_dict= {x: binarize(mnist.test.images), y_: mnist.test.labels}
      loss = sess.run(cross_entropy, \
                      feed_dict = {x: binarize(batch_xs), y_: batch_ys})
      print('%d th test acc: %s loss: %s' %(i, test_acc, loss))
      rnn_outputs_full_run = sess.run(rnn_outputs_full, \
                              feed_dict = {x: binarize(batch_xs), y_: batch_ys}
      which_image = 0
      binarized_input_image = binarize(batch_xs[which_image]).reshape(28,28)
      lstm_i = rnn_outputs_full_run[:,2,which_image,:]
      lstm_f = rnn_outputs_full_run[:,3,which_image,:]
      lstm_o = rnn_outputs_full_run[:,4,which_image,:]

```

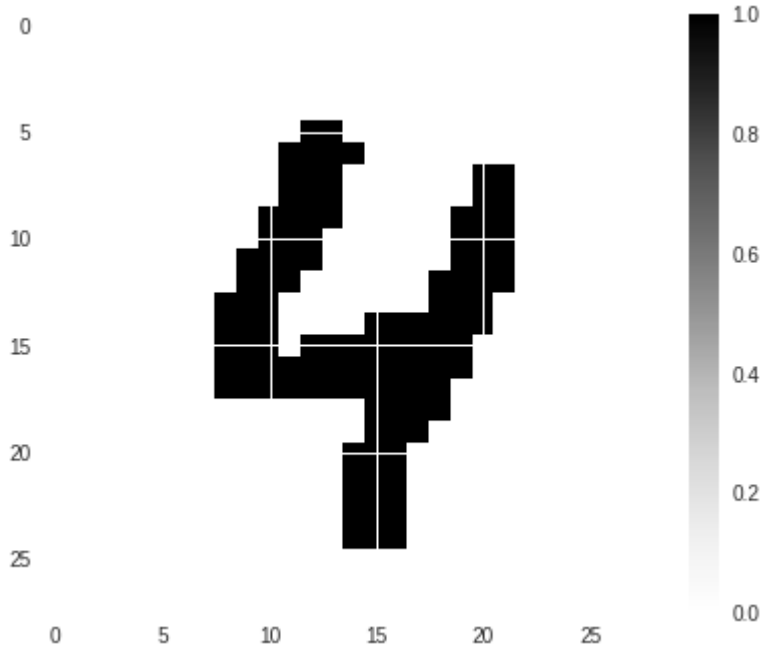
```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
2149 th test acc: 0.9515 loss: 0.12104231
```

In [61]:

```
1 plt.imshow(binarized_input_image)
2 plt.colorbar()
```

Out[61]:

```
<matplotlib.colorbar.Colorbar at 0x7fe5f2936518>
```

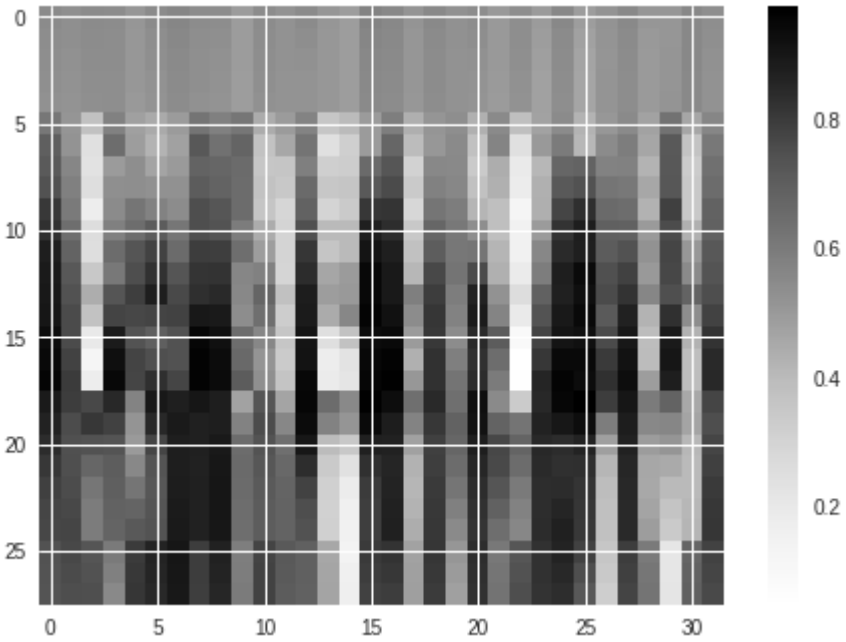


In [62]:

```
1 plt.imshow(lstm_i)
2 plt.colorbar()
```

Out[62]:

```
<matplotlib.colorbar.Colorbar at 0x7fe5f2f7cda0>
```

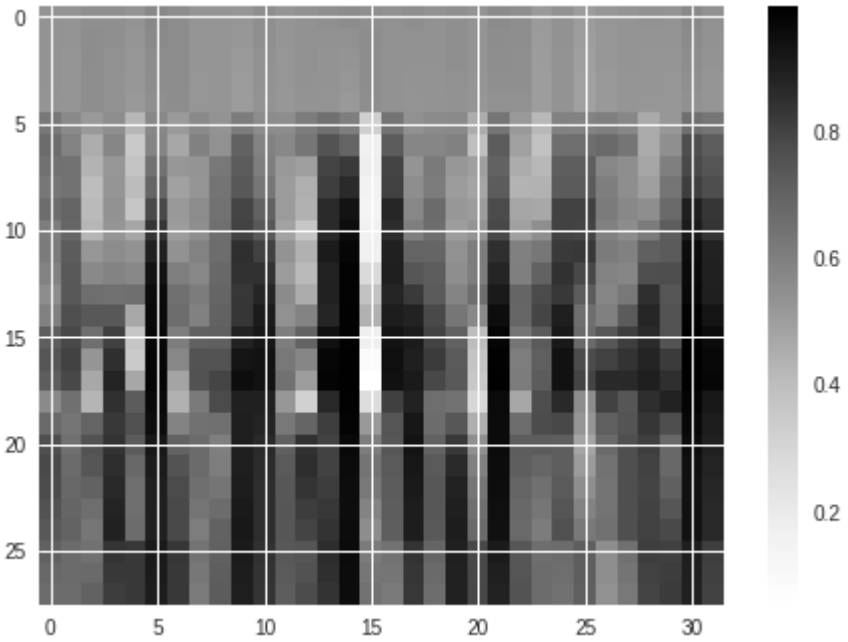


In [63]:

```
1 plt.imshow(lstm_f)
2 plt.colorbar()
```

Out[63]:

```
<matplotlib.colorbar.Colorbar at 0x7fe5f24dc9b0>
```

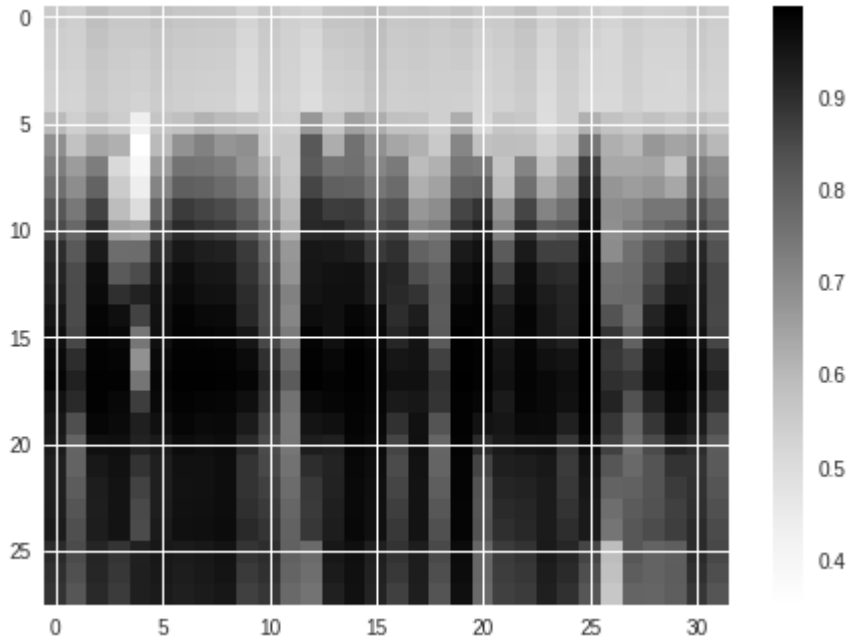


In [64]:

```
1 plt.imshow(lstm_o)
2 plt.colorbar()
```

Out[64]:

```
<matplotlib.colorbar.Colorbar at 0x7fe5f5ad4668>
```



In [0]:

```
1
```