Lecture 10-1
ReLU: Better non-linearity

Sung Kim <hunkim+mr@gmail.com>
http://hunkim.github.io/ml/
NN for XOR

Activation function

\[ W_1 = \begin{bmatrix} 5 & -7 \\ 5 & -7 \end{bmatrix}, \quad b_1 = \begin{bmatrix} -8 \\ 3 \end{bmatrix} \]

\[ W_2 = \begin{bmatrix} -11 \\ -11 \end{bmatrix}, \quad b_2 = 6 \]
NN for XOR

```
W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([2]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)
```
Let's go deep & wide!

```python
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)

hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```
9 hidden layers!

```python
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0), name = "Weight1")
W2 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight2")
W3 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight3")
W4 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight4")
W5 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight5")
W6 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight6")
W7 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight7")
W8 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight8")
W9 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight9")
W10 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight10")
W11 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0), name = "Weight11")

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([5]), name="Bias2")
b3 = tf.Variable(tf.zeros([5]), name="Bias3")
b4 = tf.Variable(tf.zeros([5]), name="Bias4")
b5 = tf.Variable(tf.zeros([5]), name="Bias5")
b6 = tf.Variable(tf.zeros([5]), name="Bias6")
b7 = tf.Variable(tf.zeros([5]), name="Bias7")
b8 = tf.Variable(tf.zeros([5]), name="Bias8")
b9 = tf.Variable(tf.zeros([5]), name="Bias9")
b10 = tf.Variable(tf.zeros([5]), name="Bias10")
b11 = tf.Variable(tf.zeros([1]), name="Bias11")
```
9 hidden layers!

W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0), name = "Weight1")
W2 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight2")
W3 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight3")
W4 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight4")
W5 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight5")
W6 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight6")
W7 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight7")
W8 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight8")
W9 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight9")
W10 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight10")
W11 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0), name = "Weight11")

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([5]), name="Bias2")
b3 = tf.Variable(tf.zeros([5]), name="Bias3")
b4 = tf.Variable(tf.zeros([5]), name="Bias4")
b5 = tf.Variable(tf.zeros([5]), name="Bias5")
b6 = tf.Variable(tf.zeros([5]), name="Bias6")
b7 = tf.Variable(tf.zeros([5]), name="Bias7")
b8 = tf.Variable(tf.zeros([5]), name="Bias8")
b9 = tf.Variable(tf.zeros([5]), name="Bias9")
b10 = tf.Variable(tf.zeros([5]), name="Bias10")
b11 = tf.Variable(tf.zeros([1]), name="Bias11")

# Our hypothesis
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)
L5 = tf.sigmoid(tf.matmul(L4, W5) + b5)
L6 = tf.sigmoid(tf.matmul(L5, W6) + b6)
L7 = tf.sigmoid(tf.matmul(L6, W7) + b7)
L8 = tf.sigmoid(tf.matmul(L7, W8) + b8)
L9 = tf.sigmoid(tf.matmul(L8, W9) + b9)
L10 = tf.sigmoid(tf.matmul(L9, W10) + b10)

hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
9 hidden layers!

```python
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0), name = "Weight1")
W2 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight2")
W3 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight3")
W4 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight4")
W5 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight5")
W6 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight6")
W7 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight7")
W8 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight8")
W9 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight9")
W10 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight10")
W11 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0), name = "Weight11")

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([5]), name="Bias2")
b3 = tf.Variable(tf.zeros([5]), name="Bias3")
b4 = tf.Variable(tf.zeros([5]), name="Bias4")
b5 = tf.Variable(tf.zeros([5]), name="Bias5")
b6 = tf.Variable(tf.zeros([5]), name="Bias6")
b7 = tf.Variable(tf.zeros([5]), name="Bias7")
b8 = tf.Variable(tf.zeros([5]), name="Bias8")
b9 = tf.Variable(tf.zeros([5]), name="Bias9")
b10 = tf.Variable(tf.zeros([5]), name="Bias10")
b11 = tf.Variable(tf.zeros([1]), name="Bias11")

# Our hypothesis

with tf.name_scope("layer1") as scope:
    L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
with tf.name_scope("layer2") as scope:
    L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
with tf.name_scope("layer3") as scope:
    L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
with tf.name_scope("layer4") as scope:
    L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)
with tf.name_scope("layer5") as scope:
    L5 = tf.sigmoid(tf.matmul(L4, W5) + b5)
with tf.name_scope("layer6") as scope:
    L6 = tf.sigmoid(tf.matmul(L5, W6) + b6)
with tf.name_scope("layer7") as scope:
    L7 = tf.sigmoid(tf.matmul(L6, W7) + b7)
with tf.name_scope("layer8") as scope:
    L8 = tf.sigmoid(tf.matmul(L7, W8) + b8)
with tf.name_scope("layer9") as scope:
    L9 = tf.sigmoid(tf.matmul(L8, W9) + b9)
with tf.name_scope("layer10") as scope:
    L10 = tf.sigmoid(tf.matmul(L9, W10) + b10)

with tf.name_scope("last") as scope:
    hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
```
Tensorboard visualization
Poor results?

196000
[0.69314718, array([[ 0.49999988],
[ 0.50000006],
[ 0.49999982],
[ 0.5    ]], dtype=float32)]

198000
[0.69314718, array([[ 0.49999988],
[ 0.50000006],
[ 0.49999982],
[ 0.5    ]], dtype=float32), array([[ 0.],
[ 1.],
[ 0.],
[ 1.]], dtype=float32)]

Accuracy: 0.5
Tensorboard
Cost & Accuracy

cost

accuracy
Backpropagation
lec 9-2: Backpropagation (chain rule)
Vanishing gradient (NN winter2: 1986-2006)
Geoffrey Hinton’s summary of findings up to today

• Our labeled datasets were thousands of times too small.
• Our computers were millions of times too slow.
• We initialized the weights in a stupid way.
• We used the wrong type of non-linearity.

Sigmoid!
Sigmoid!
ReLU: Rectified Linear Unit
ReLU: Rectified Linear Unit

\[ \text{max}(0, X) \]

\[ L1 = \text{tf.nn.relu}(\text{tf.matmul}(X, W1) + b1) \]

\[ L1 = \text{tf.sigmoid}(\text{tf.matmul}(X, W1) + b1) \]
# Our hypothesis

```python
with tf.name_scope("layer1") as scope:
    L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
with tf.name_scope("layer2") as scope:
    L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
with tf.name_scope("layer3") as scope:
    L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
with tf.name_scope("layer4") as scope:
    L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
with tf.name_scope("layer5") as scope:
    L5 = tf.nn.relu(tf.matmul(L4, W5) + b5)
with tf.name_scope("layer6") as scope:
    L6 = tf.nn.relu(tf.matmul(L5, W6) + b6)
with tf.name_scope("layer7") as scope:
    L7 = tf.nn.relu(tf.matmul(L6, W7) + b7)
with tf.name_scope("layer8") as scope:
    L8 = tf.nn.relu(tf.matmul(L7, W8) + b8)
with tf.name_scope("layer9") as scope:
    L9 = tf.nn.relu(tf.matmul(L8, W9) + b9)
with tf.name_scope("layer10") as scope:
    L10 = tf.nn.relu(tf.matmul(L9, W10) + b10)

with tf.name_scope("last") as scope:
    hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
```
Works very well

196000 [2.6226094e-06, array([[ 2.59195826e-06],
[ 9.99999642e-01],
[ 9.99994874e-01],
[ 2.43454133e-06]], dtype=’float32’)]
198000 [2.607708e-06, array([[ 2.55822852e-06],
[ 9.99999642e-01],
[ 9.99994874e-01],
[ 2.40260101e-06]], dtype=’float32’)]

[array([[ 2.52509381e-06],
[ 9.99999642e-01],
[ 9.99994874e-01],
[ 2.37124709e-06]], dtype=’float32’), array([[ 0.],
[ 1.],
[ 1.],
[ 0.]], dtype=’float32’)]

Accuracy: 1.0
Works very well
Cost function

- sigmoid
- ReLU

Graph showing the cost function with different activation functions: sigmoid and ReLU.
Activation Functions

**Sigmoid**

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh**

\[ \tanh(x) \]

**ReLU**

\[ \max(0, x) \]

**Leaky ReLU**

\[ \max(0.1x, x) \]

**Maxout**

\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**

\[ f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \]
Activation functions on CIFAR-10

<table>
<thead>
<tr>
<th></th>
<th>maxout</th>
<th>ReLU</th>
<th>VLReLU</th>
<th>tanh</th>
<th>Sigmoid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>93.94</td>
<td>92.11</td>
<td>92.97</td>
<td>89.28</td>
<td>n/c</td>
</tr>
<tr>
<td>93.78</td>
<td>91.74</td>
<td></td>
<td>92.40</td>
<td>89.48</td>
<td>n/c</td>
</tr>
<tr>
<td>-</td>
<td>91.93</td>
<td></td>
<td>93.09</td>
<td></td>
<td>n/c</td>
</tr>
<tr>
<td>91.75</td>
<td>90.63</td>
<td>92.27</td>
<td></td>
<td>89.82</td>
<td>n/c</td>
</tr>
<tr>
<td>n/c†</td>
<td>90.91</td>
<td>92.43</td>
<td></td>
<td>89.54</td>
<td>n/c</td>
</tr>
</tbody>
</table>

[Mishkin et al. 2015]
Next

Weight initialization
Lecture 10-2

Initialize weights in a smart way

Sung Kim <hunkim+mr@gmail.com>
http://hunkim.github.io/ml/
Vanishing gradient
Geoffrey Hinton’s summary of findings up to today

• Our labeled datasets were thousands of times too small.
• Our computers were millions of times too slow.
• We initialized the weights in a stupid way.
• We used the wrong type of non-linearity.

Set all initial weights to 0
Need to set the initial weight values wisely

- Not all 0's
- Challenging issue
- Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets"
  - Restricted Boatman Machine (RBM)
RBM STRUCTURE

Visible Layer

Hidden Layer

RESTRICTION = NO CONNECTIONS WITHIN A LAYER
RECREATE INPUT

Forward

Backward
RECREATE INPUT

Forward

Backward
How can we use RBM to initialize weights?

- Apply the RBM idea on adjacent two layers as a pre-training step
- Continue the first process to all layers
- This will set weights
- Example: Deep Belief Network
  - Weight initialized by RBM
Deep Belief Network (DBN)

Unsupervised, layer-wise, greedy pre-training
PRE-TRAINING
Good news

• No need to use complicated RBM for weight initializations

• Simple methods are OK
Xavier/He initialization

- Makes sure the weights are ‘just right’, not too small, not too big
- Using number of input (fan_in) and output (fan_out)

```python
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)

# He et al. 2015
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2)
```

[http://cs231n.stanford.edu/](http://cs231n.stanford.edu/)
prettytensor implementation

def xavier_init(n_inputs, n_outputs, uniform=True):
    """Set the parameter initialization using the method described. This method is designed to keep the scale of the gradients roughly the same in all layers.
Xavier Glorot and Yoshua Bengio (2010):

Args:
    n_inputs: The number of input nodes into each output.
    n_outputs: The number of output nodes for each input.
    uniform: If true use a uniform distribution, otherwise use a normal.

Returns:
    An initializer.
    """
    if uniform:
        # 6 was used in the paper.
        init_range = math.sqrt(6.0 / (n_inputs + n_outputs))
        return tf.random_uniform_initializer(-init_range, init_range)
    else:
        # 3 gives us approximately the same limits as above since this replicates values greater than 2 standard deviations from the mean.
        stddev = math.sqrt(3.0 / (n_inputs + n_outputs))
        return tf.truncated_normal_initializer(stddev=stddev)

http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow
Activation functions and initialization on CIFAR-10

<table>
<thead>
<tr>
<th>Init method</th>
<th>maxout</th>
<th>ReLU</th>
<th>VLReLU</th>
<th>tanh</th>
<th>Sigmoid</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSUV</td>
<td>93.94</td>
<td>92.11</td>
<td>92.97</td>
<td>89.28</td>
<td>n/c</td>
</tr>
<tr>
<td>OrthoNorm</td>
<td>93.78</td>
<td>91.74</td>
<td>92.40</td>
<td>89.48</td>
<td>n/c</td>
</tr>
<tr>
<td>OrthoNorm-MSRA scaled</td>
<td>–</td>
<td>91.93</td>
<td>93.09</td>
<td>–</td>
<td>n/c</td>
</tr>
<tr>
<td>Xavier</td>
<td>91.75</td>
<td>90.63</td>
<td>92.27</td>
<td>89.82</td>
<td>n/c</td>
</tr>
<tr>
<td>MSRA</td>
<td>n/c†</td>
<td>90.91</td>
<td>92.43</td>
<td>89.54</td>
<td>n/c</td>
</tr>
</tbody>
</table>

[Mishkin et al. 2015]
Still an active area of research

- We don’t know how to initialize perfect weight values, yet
- Many new algorithms
  - Batch normalization
  - Layer sequential uniform variance
  - …
Geoffrey Hinton’s summary of findings up to today

- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- We initialized the weights in a stupid way.
- We used the wrong type of non-linearity.

Next dropout and model ensemble
Lecture 10-3
NN dropout and model ensemble

Sung Kim <hunkim+mr@gmail.com>
Overfitting
Am I overfitting?

- Very high accuracy on the training dataset (eg: 0.99)
- Poor accuracy on the test data set (0.85)

http://cs224d.stanford.edu/syllabus.html
Solutions for overfitting

• More training data!
• Reduce the number of features
• Regularization
Regularization

- Let’s not have too big numbers in the weight
Regularization

• Let’s not have too big numbers in the weight

\[ \text{cost} + \lambda \sum W^2 \]

\[ \text{l2reg} = 0.001 \times \text{tf.reduce_sum(tf.square(W))} \]
Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]
Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”

(a) Standard Neural Net  
(b) After applying dropout.

[Srivastava et al., 2014]
Waaaait a second...
How could this possibly be a good idea?

Forces the network to have a redundant representation.

- has an ear
- has a tail
- is furry
- has claws
- mischievous look

Fei-Fei Li & Andrej Karpathy & Justin Johnson
Lecture 6 - 53
25 Jan 2016
TensorFlow implementation

```python
dropout_rate = tf.placeholder("float")
_L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
L1 = tf.nn.dropout(_L1, dropout_rate)
```

**TRAIN:**
```python
sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys, dropout_rate: 0.7})
```

**EVALUATION:**
```python
print "Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels, dropout_rate: 1})
```
What is Ensemble?

http://www.slideshare.net/sasasiapacific/ibp-improving-the-models-predictive-power-with-ensemble-approaches
Lecture 10-4
NN LEGO Play

Sung Kim <hunkim+mr@gmail.com>
Feedforward neural network
Fast forward
Recurrent network
'The only limit is your imagination'

http://itchyi.squarespace.com/thelatest/2012/5/17/the-only-limit-is-your-imagination.html