TERADATA

7 steps for highly effective deep neural networks

Natalino Busa - Head of Data Science

Natalino Busa

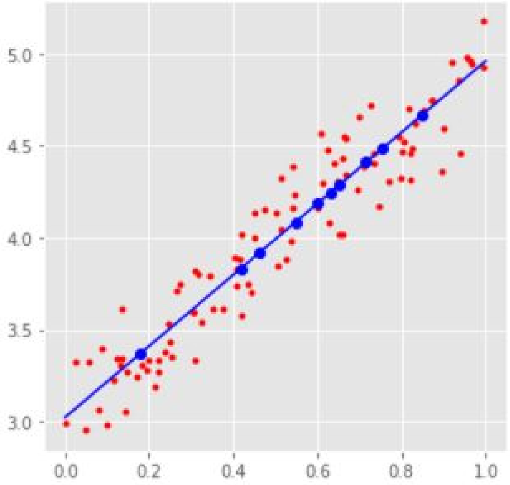
View Profile

Linkedin and Twitter:

@natbusa

O’Reilly	Author and Speaker
Teradata	EMEA Practice Lead on Open Source Technologies
Teradata	Principal Data Scientist
ING Group	Enterprise Architect: Cybersecurity, Marketing, Fintech
Cognitive Finance Group	Advisory Board Member
Philips	Senior Researcher, Data Architect

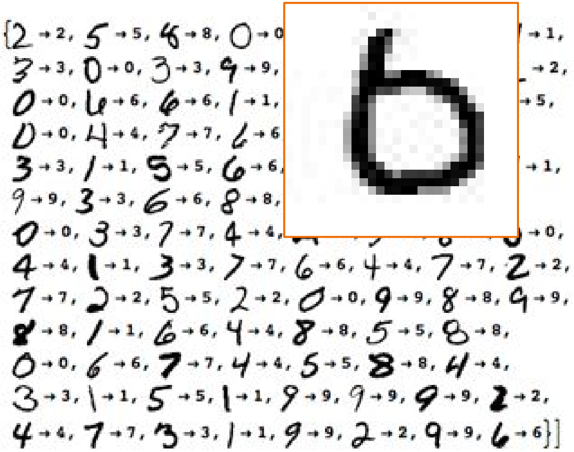
Linear Regression:
How to best fit a line to some data



X: independent variable
Y: dependent variable

Classification: Handwritten digits

28x28 pixels



input : 784 numbers
output: 10 classes

Sharing the (Not so) Secret Lore:

Keras & Tensorflow

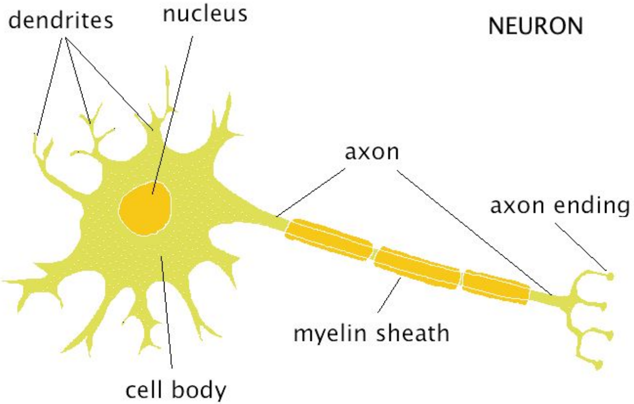
Some smaller projects:
[Tflearn], [Tensorlayers]

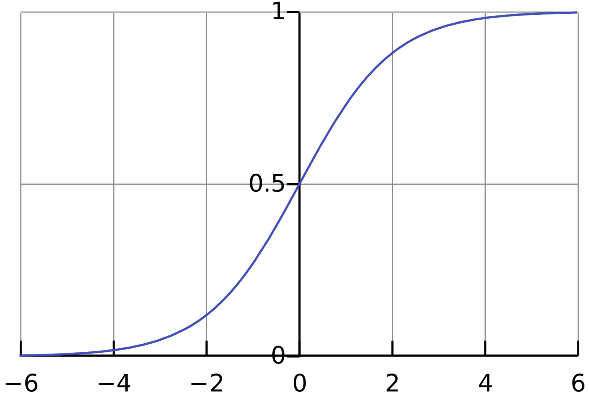
K http://keras.io/

TensorFlow™

Natalino Busa - @natbusa

1: Single Layer Perceptron





Activation function

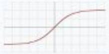
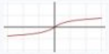
Axon's response

$$\theta_i = \frac{1}{1 + \exp\left[-\left(\beta_0 + \sum_{j=1}^{k} \beta_j x_{ij}\right)\right]}$$

"dendrites"

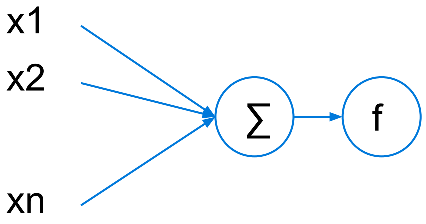
More activation functions:

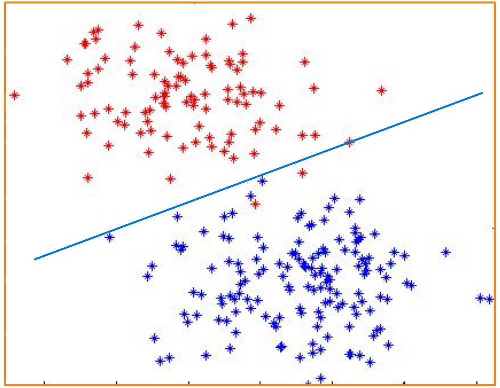
https://en.wikipedia.org/wiki/Activation_function

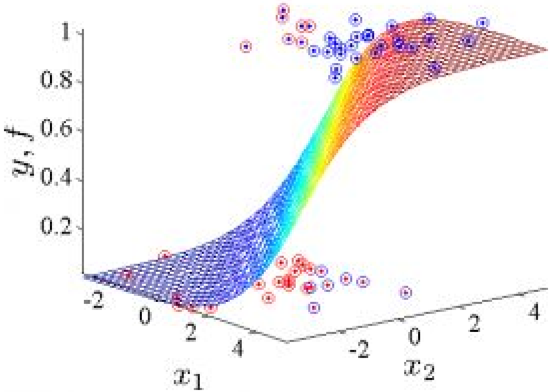
Name	Plot	Equation	Derivative (with respect to x)	Range	Order of continuity	Monotonic	Derivative Monotonic	Approximates identity near the origin
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	C^∞	Yes	Yes	Yes
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}	Yes	No	No
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	C^∞	Yes	No	No
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞	Yes	No	Yes
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞	Yes	No	Yes
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x \rvert}$	$f'(x) = \frac{1}{(1+\lvert x \rvert)^2}$	$(-1, 1)$	C^1	Yes	No	Yes
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$	C^0	Yes	Yes	No
Leaky rectified linear unit (Leaky ReLU)[10]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Parameteric rectified linear unit (PReLU)[11]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes iff $\alpha \geq 0$	Yes	Yes iff $\alpha = 1$
Randomized leaky rectified linear unit (RReLU)[12]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ [1]	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Exponential linear unit (ELU)[13]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\alpha, \infty)$	C^1 when $\alpha = 1$, otherwise C^0	Yes iff $\alpha \geq 0$	Yes iff $0 \leq \alpha \leq 1$	Yes iff $\alpha = 1$

1: Single Layer Perceptron (binary classifier)

Single Layer Neural Network
Takes: n-input features: Map them to a soft "binary" space



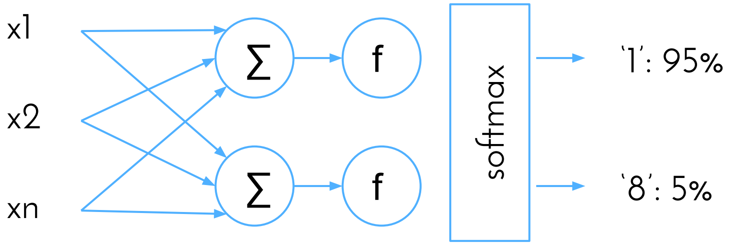




1: Single Layer Perceptron (multi-class classifier)

From soft binary space to predicting probabilities:

Take n inputs, Divide by the sum of the predicted values



$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Values between 0 and 1

Sum of all outcomes = 1

It produces an **estimate of a probability**!

1: Single Layer Perceptron

Minimize costs:

The cost function depends on:

- Parameters of the model
- How the model "composes"

Goal :

Reduce the mean probability error

modify the parameters to reduce the error!

Gradient Descent Algorithm :

$$\Theta_{n+1} = \Theta_n - \alpha \frac{\partial}{\partial \Theta_n} J(\Theta_n)$$

$\Theta \rightarrow$ Parameter Vector

$J \rightarrow$ Cost Function

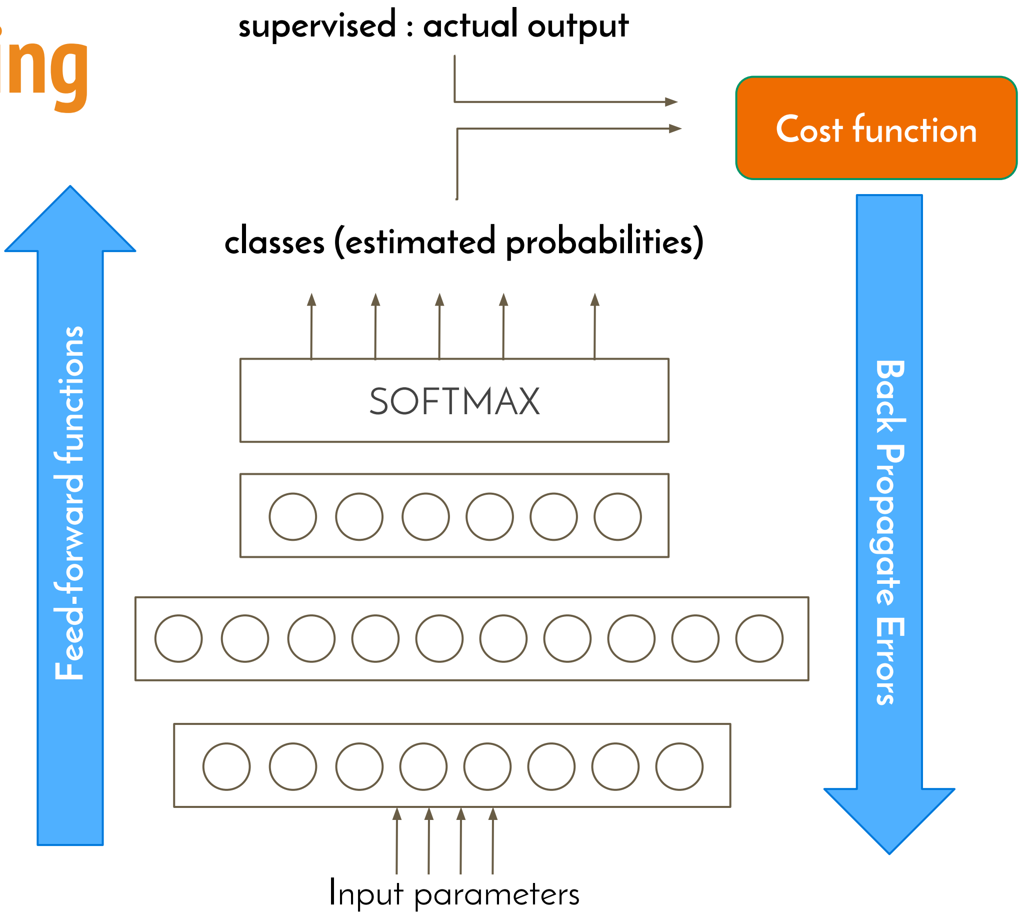
$\alpha \rightarrow$ Slope Parameter

Vintage math from last century

Supervised Learning

Stack layers of perceptrons

- Feed Forward Network
- Scoring goes
  from input to output
- Back propagate the error
  from output to input



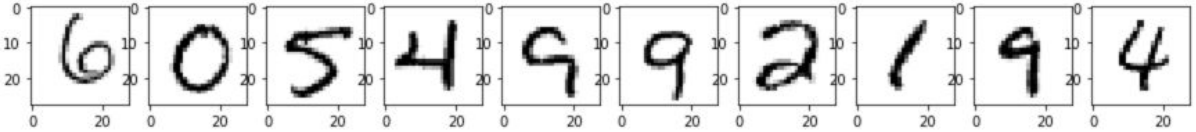
Let’s go!

```
In [7]: %matplotlib inline

        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [8]: from tensorflow.examples.tutorials.mnist import input_data
        mnist = input_data.read_data_sets("../mnist-data/", one_hot=True)

        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
```

```
In [9]: mnist.train.images.shape

Out[9]: (55000, 784)
```

```
In [10]: plt.figure(figsize=(15,5))
         for i in list(range(10)):
             plt.subplot(1, 10, i+1)
             pixels = mnist.test.images[i+100]
             pixels = pixels.reshape((28, 28))
             plt.imshow(pixels, cmap='gray_r')
         plt.show()
```



1. Single Layer Perceptron

```
In [11]: from keras.models import Sequential
         from keras.layers.core import Dense, Activation
```

```
In [12]: model = Sequential()
         model.add(Dense(10, input_shape=(784,)))
         model.add(Activation('softmax'))
```

```
In [13]: model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

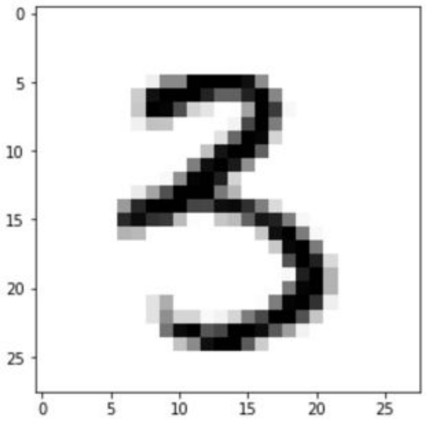
```
In [14]: model.fit(mnist.train.images, mnist.train.labels,
                   batch_size=500, nb_epoch=10, verbose=1,
                   validation_data=(mnist.test.images, mnist.test.labels))
```

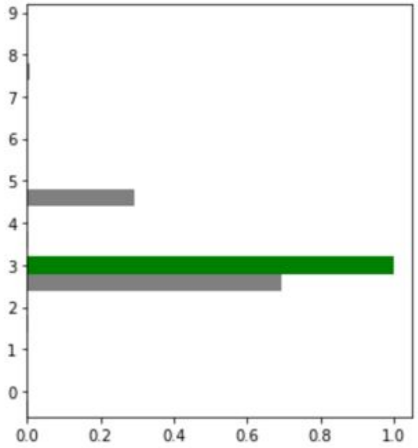
1. Single Layer Perceptron

```
In [10]: score = model.evaluate(mnist.test.images, mnist.test.labels, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         Test score: 0.293662216854
         Test accuracy: 0.9191

In [11]: # test item #100 is a six
         pixels = mnist.test.images[100]
         result = model.predict_on_batch(np.array([pixels]))
         dict(zip(range(10), result[0]))

Out[11]: {0: 0.0073514683,
          1: 0.0034446407,
          2: 0.047751036,
          3: 0.00301607,
          4: 0.0032819158,
          5: 0.00030964505,
          6: 0.92316657,
          7: 0.00083195668,
          8: 0.0075355168,
          9: 0.0033110771}
```

1. Single Layer Perceptron

In [17]:
```
import random
i = random.randint(0,mnist.test.images.shape[0])

pixels = mnist.test.images[i]
truth  = mnist.test.labels[i]
result = model.predict_on_batch(np.array([pixels]))[0]

test_render(pixels, result, truth)
```





Natalino Busa - @natbusa

1. Single Layer Perceptron

TensorFlow™

```
In [7]: # Set parameters
        learning_rate = 0.01
        training_iteration = 10
        batch_size = 250

        FLAGS = None
```

```
In [8]: # TF graph input
        x = tf.placeholder('float', [None, 784]) # mnist data image of shape 28*28=784
        y = tf.placeholder('float', [None, 10]) # 0-9 digits recognition => 10 classes
```

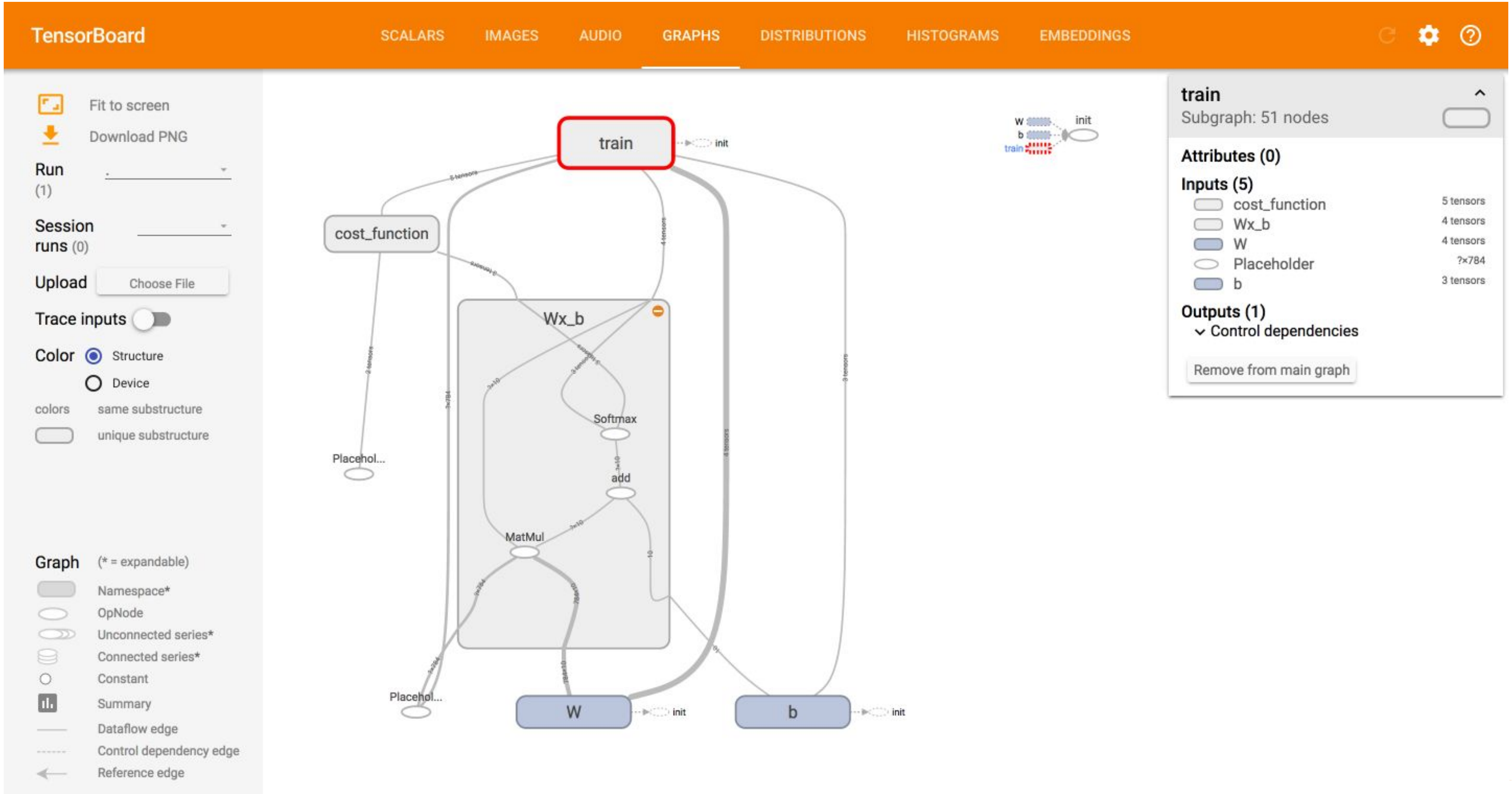
```
In [9]: # Set model weights
        W = tf.Variable(tf.zeros([784, 10]), name='W')
        b = tf.Variable(tf.zeros([10]), name='b')
```

```
In [10]: with tf.name_scope("Wx_b") as scope:
             # Construct a linear model
             y_hat = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

             # Add summary ops to collect data
             tf.summary.histogram("weights", W)
             tf.summary.histogram("biases", b)
```

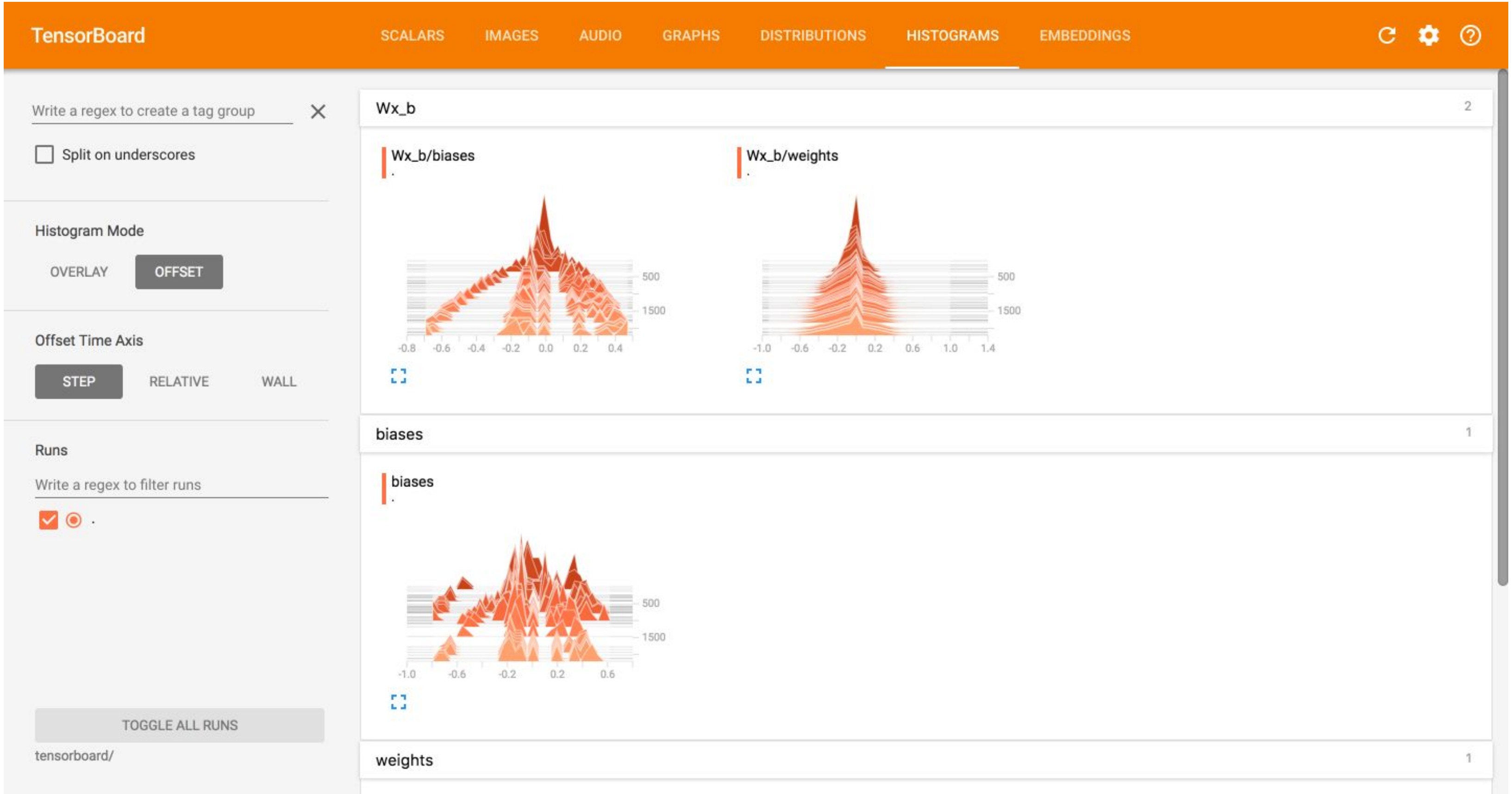
```
In [11]: # More name scopes will clean up graph representation
         with tf.name_scope("cost_function") as scope:
             # Minimize error using cross entropy
             # Cross entropy
             cost_function = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=y_hat))
             # Create a summary to monitor the cost function
             tf.summary.scalar("cost_function", cost_function)
```

```
In [12]: with tf.name_scope("train") as scope:
             # Gradient descent
             optimizer = tf.train.AdamOptimizer().minimize(cost_function)
```

Tensorboard

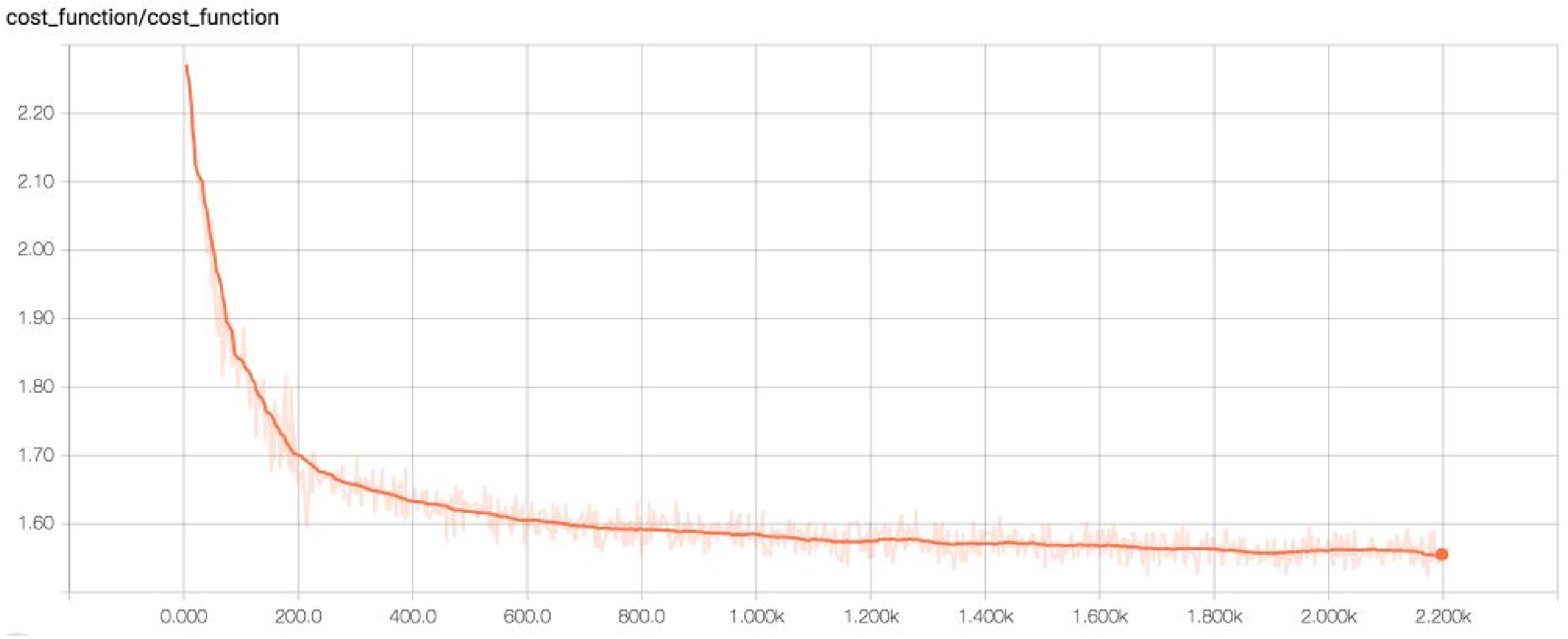


Tensorboard



Natalino Busa - @natbusa

Tensorboard



2. Multi Layer Perceptron

```
In [7]: # Set parameters
        learning_rate = 0.01
        training_iteration = 10
        batch_size = 250
        print_freq=5
```

```
In [8]: # TF graph input
        x = tf.placeholder('float', [None, 784]) # mnist data image of shape 28*28=784
        y = tf.placeholder('float', [None, 10]) # 0-9 digits recognition => 10 classes

        keep_rate = tf.placeholder(tf.float32)
```

```
In [9]: def weight_variable(shape):
          initial = tf.constant(0.0, shape=shape)
          return tf.Variable(initial)

        def bias_variable(shape):
          initial = tf.constant(0.1, shape=shape)
          return tf.Variable(initial)
```

```
In [10]: with tf.name_scope("hidden_1") as scope:

             # Set model weights
             W_layer1 = weight_variable([784, 512])
             b_layer1 = bias_variable([512])

             # Construct a dense linear model, with act=relu and dropout
             layer_1 = tf.nn.dropout(tf.nn.relu(tf.matmul(x, W_layer1) + b_layer1), keep_rate) # Relu, dropout

             # Add summary ops to collect data
             tf.histogram_summary("W1_weights", W_layer1)
             tf.histogram_summary("B1_biases", b_layer1)
```

2. Multi Layer Perceptron

TensorFlow™

In [11]:
```
with tf.name_scope("hidden_2") as scope:

    # Set model weights
    W_layer2 = weight_variable([512, 512])
    b_layer2 = bias_variable([512])

    # Construct a dense linear model, with act=relu and dropout
    layer_2 = tf.nn.dropout(tf.nn.relu(tf.matmul(layer_1, W_layer2) + b_layer2), keep_rate) # Relu, dropout

    # Add summary ops to collect data
    tf.histogram_summary("W2_weights", W_layer2)
    tf.histogram_summary("B2_biases", b_layer2)
```

In [12]:
```
with tf.name_scope("output") as scope:

    # Set model weights
    W_layer3 = weight_variable([512, 10])
    b_layer3 = bias_variable([10])

    # Construct a dense linear model, with act=relu and dropout
    layer_3 = tf.add(tf.matmul(layer_2, W_layer3), b_layer3)

    # Add summary ops to collect data
    tf.histogram_summary("W3_weights", W_layer3)
    tf.histogram_summary("B3_biases", b_layer3)
```

In [13]:
```
y_hat = layer_3

# More name scopes will clean up graph representation
with tf.name_scope("cost_function") as scope:
    # Minimize error using cross entropy
    # Cross entropy
    cost_function = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y_hat,y))
    # Create a summary to monitor the cost function
    tf.scalar_summary("cost_function", cost_function)
```

2. Multi Layer Perceptron

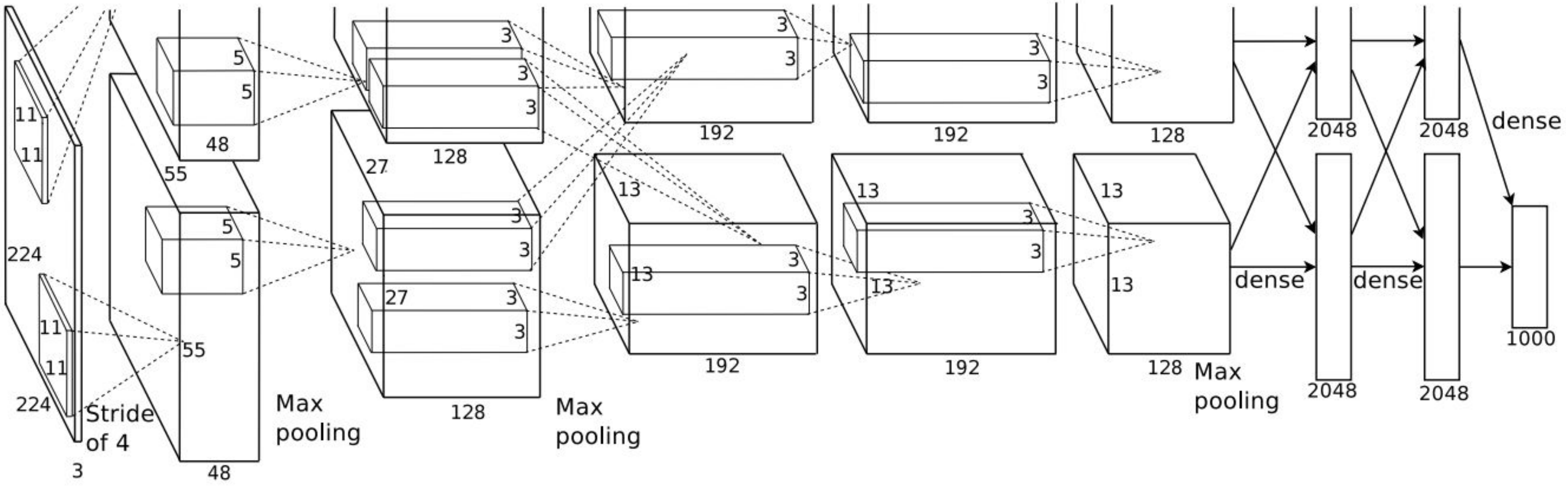
TensorFlow™

```
In [14]: with tf.name_scope("train") as scope:
             # Gradient descent
             optimizer = tf.train.AdamOptimizer().minimize(cost_function)

In [15]: # Initializing the variables
         init = tf.global_variables_initializer()

         # Merge all summaries into a single operator
         merged_summary_op = tf.merge_all_summaries()

In [16]: # Launch the graph
         sess = tf.InteractiveSession()
         sess.run(init)

In [17]: # Change this to a location on your computer
         summary_writer = tf.train.SummaryWriter('./tensorboard', graph=sess.graph)

In [18]: # Training cycle
         for iteration in range(training_iteration):
             avg_cost = 0.
             total_batch = int(mnist.train.num_examples/batch_size)
             # Loop over all batches
             for i in range(total_batch):
                 batch_xs, batch_ys = mnist.train.next_batch(batch_size)

                 # dropout placeholder
                 batch_kr = 0.50

                 # Fit training using batch data
                 sess.run(optimizer, feed_dict={x: batch_xs, keep_rate: batch_kr, y: batch_ys})

                 # Compute the average loss
                 avg_cost += sess.run(cost_function, feed_dict={x: batch_xs, keep_rate: batch_kr, y: batch_ys})/(total_batch+1)

                 # Write logs for each iteration
                 summary_str = sess.run(merged_summary_op, feed_dict={x: batch_xs, keep_rate:batch_kr, y: batch_ys})
                 summary_writer.add_summary(summary_str, iteration*total_batch + i)

             # Display logs per iteration step
             if iteration % print_freq ==0 :
                 print("Iteration:", '%04d' % (iteration), "cost=", "{:.9f}".format(avg_cost))

Iteration: 0000 cost= 0.636337465
Iteration: 0005 cost= 0.083090350
```

2. Multi Layer Perceptron

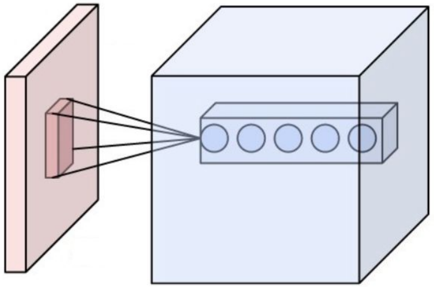
```
In [6]: from keras.models import Sequential
        from keras.layers.core import Dense, Activation, Dropout

        Using TensorFlow backend.

In [7]: model = Sequential()
        model.add(Dense(512, input_shape=(784,)))
        model.add(Activation('relu'))
        model.add(Dropout(0.25))
        model.add(Dense(512, activation='relu'))
        model.add(Activation('relu'))
        model.add(Dropout(0.25))
        model.add(Dense(10))
        model.add(Activation('softmax'))

In [8]: model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

In [*]: model.fit(mnist.train.images, mnist.train.labels,
                  batch_size=250, epochs=10, verbose=1,
                  validation_data=(mnist.test.images, mnist.test.labels))
```

3. Convolution

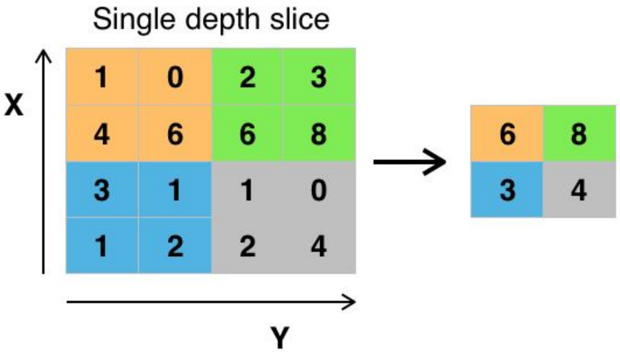


From Krizehvsky *et al.* (2012)

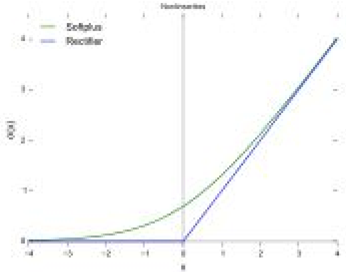
3. Convolution



convolution



Max pooling



RELU / ELU

diagrams:
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581
CC0, https://en.wikipedia.org/w/index.php?curid=48817276

3. Convolution

```
In [14]: from keras.models import Sequential
         from keras.layers import Dense, Activation
         from keras.layers import Dropout, Flatten, Reshape
         from keras.layers import Conv2D, MaxPooling2D
```

```
In [15]: from keras import backend as K

         #tensorflow default channel ordering
         input_shape = (28,28,1) #channel is third
```

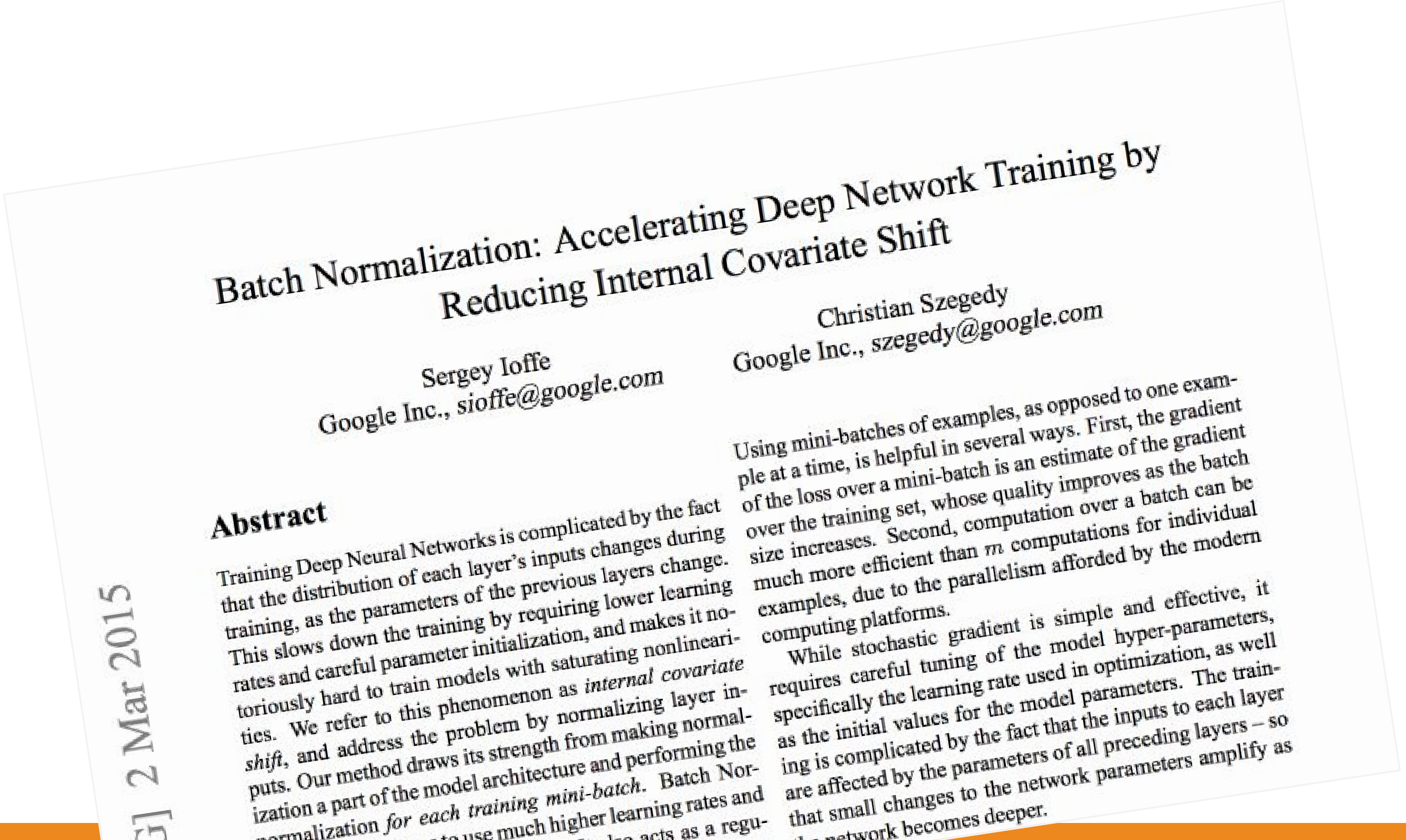
```
In [18]: model = Sequential()
         model.add(Reshape(input_shape, input_shape=(784,)))

         model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
         model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
         model.add(MaxPooling2D((2,2)))

         model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
         model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
         model.add(MaxPooling2D((2,2)))

         model.add(Flatten())
         model.add(Dropout(0.5))

         model.add(Dense(256, activation='relu'))
         model.add(Dropout(0.25))

         model.add(Dense(10, activation='softmax'))
```

3. Convolution

```
In [9]: from  functools import reduce

        for l in model.layers:
            print(l.name, l.output_shape, [reduce(lambda x, y: x*y, w.shape) for w in l.get_weights()])
```

```
reshape_1 (None, 28, 28, 1) []
convolution2d_1 (None, 28, 28, 32) [288, 32]
convolution2d_2 (None, 28, 28, 32) [9216, 32]
maxpooling2d_1 (None, 14, 14, 32) []
convolution2d_3 (None, 14, 14, 64) [18432, 64]
convolution2d_4 (None, 14, 14, 64) [36864, 64]
maxpooling2d_2 (None, 7, 7, 64) []
flatten_1 (None, 3136) []
dropout_1 (None, 3136) []
dense_1 (None, 256) [802816, 256]
dropout_2 (None, 256) []
dense_2 (None, 10) [2560, 10]
```

3. Batch Normalization



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., sioffe@google.com

Christian Szegedy
Google Inc., szegedy@google.com

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Nor-

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

3. Batch Normalization (example for MLP)

In [6]:
```
from keras.models import Model
from keras.layers import Input, Dense, Activation
from keras.layers import Dropout, Flatten, Reshape
from keras.layers import Convolution2D, MaxPooling2D
from keras.layers import BatchNormalization
```

```
Using TensorFlow backend.
```

In [8]:
```
def mlp(batch_normalization=False, activation='sigmoid'):
    _in = Input(shape=(784,))

    for i in range(5):
        x = Dense(128, activation=activation, input_shape=(784,))(x if i else _in)
        if batch_normalization:
            x = BatchNormalization()(x)

    _out = Dense(10, activation='softmax')(x)
    model = Model(_in, _out)

    return model
```

3. Batch Normalization (example for MLP)

Sigmoid activation function

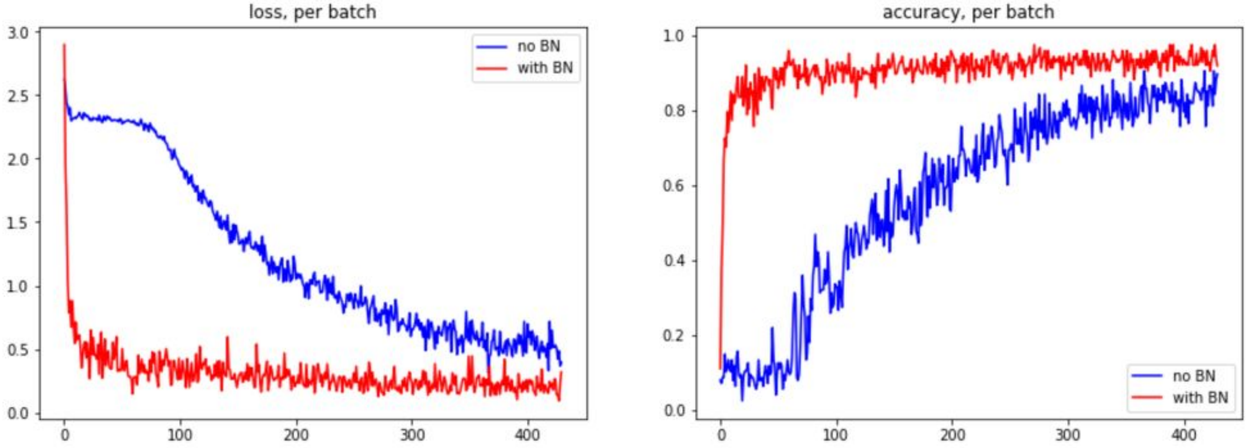
In [10]:
```
# see http://cs231n.github.io/neural-networks-3/

model = mlp(False, 'sigmoid')
print_layers(model)

bl_noBN = BatchLogger()

from keras.optimizers import Adam
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001), metrics=["accuracy"])

model.fit(mnist.train.images, mnist.train.labels,
          batch_size=128, nb_epoch=1, verbose=1, callbacks=[bl_noBN],
          validation_data=(mnist.test.images, mnist.test.labels))
```

```
input_1 (None, 784) []
dense_1 (None, 128) [100352, 128]
dense_2 (None, 128) [16384, 128]
dense_3 (None, 128) [16384, 128]
dense_4 (None, 128) [16384, 128]
dense_5 (None, 128) [16384, 128]
dense_6 (None, 10) [1280, 10]
```

3. Batch Normalization (example for MLP)

In [12]:
```
plt.figure(figsize=(15,5))
plt.subplot(1, 2, 1)
plt.title('loss, per batch')
plt.plot(bl_noBN.log_values['loss'], 'b-', label='no BN');
plt.plot(bl_BN.log_values['loss'], 'r-', label='with BN');
plt.legend(loc='upper right')
plt.subplot(1, 2, 2)
plt.title('accuracy, per batch')
plt.plot(bl_noBN.log_values['acc'], 'b-', label='no BN');
plt.plot(bl_BN.log_values['acc'], 'r-', label='with BN');
plt.legend(loc='lower right')
plt.show()
```

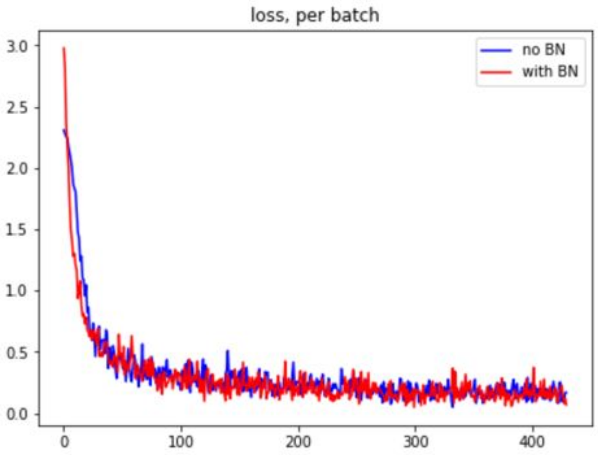
Activation function:
SIGMOID

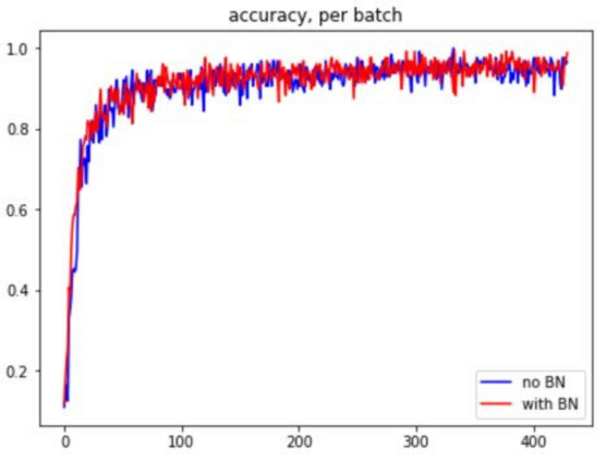


3. Batch Normalization (example for MLP)

In [15]:
```
plt.figure(figsize=(15,5))
plt.subplot(1, 2, 1)
plt.title('loss, per batch')
plt.plot(bl_noBN.log_values['loss'], 'b-', label='no BN');
plt.plot(bl_BN.log_values['loss'], 'r-', label='with BN');
plt.legend(loc='upper right')
plt.subplot(1, 2, 2)
plt.title('accuracy, per batch')
plt.plot(bl_noBN.log_values['acc'], 'b-', label='no BN');
plt.plot(bl_BN.log_values['acc'], 'r-', label='with BN');
plt.legend(loc='lower right')
plt.show()
```

Activation function:
RELU



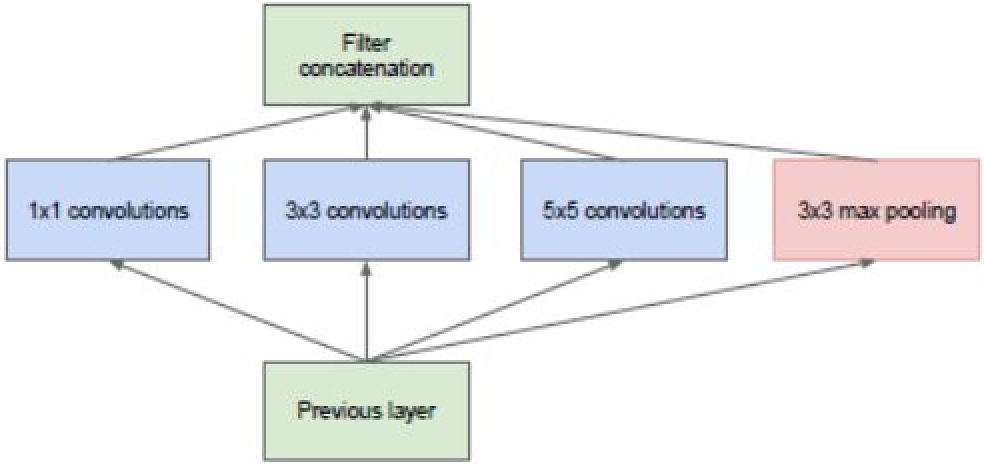


4. Regularization: Prevent overfitting in ANNs

- Batch Normalization
- RELU/ELU
- RESIDUAL / SKIP Networks
- DROP LAYER
- REDUCE PRECISION (HUFFMAN ENCODING)

In general ANN are parameters rich, constraining the parameter space usually produces better results and speed up the learning

5. Inception architectures



(a) Inception module, naïve version

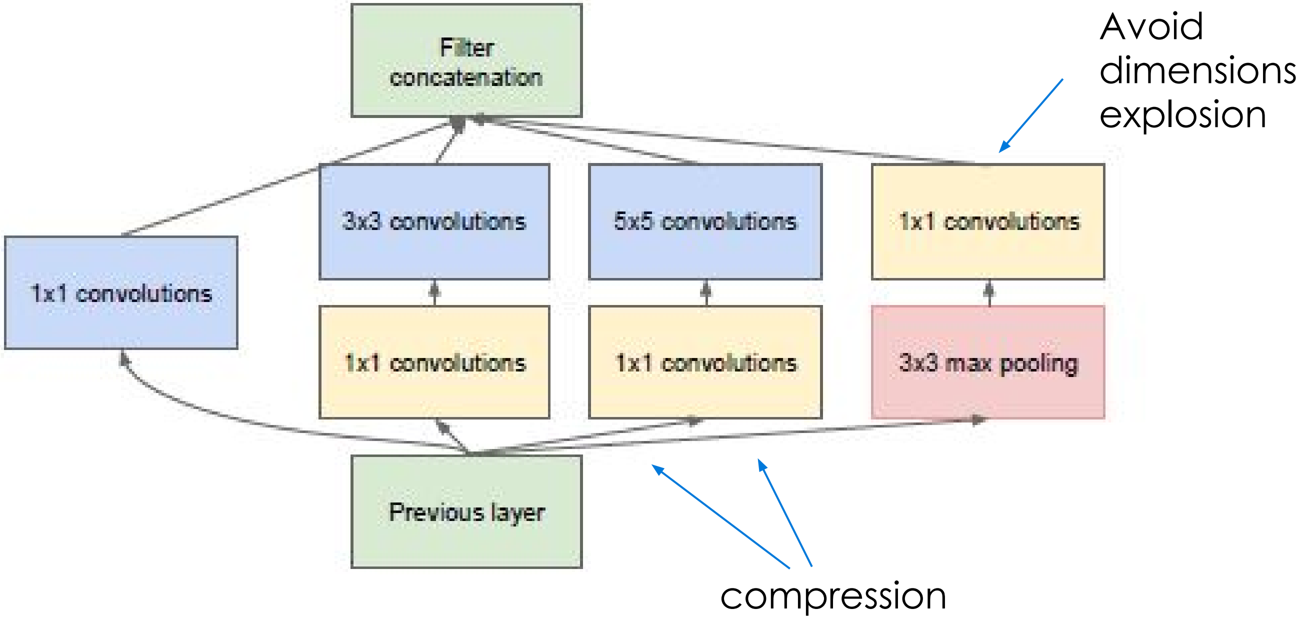
Cannot be stacked!

5. Inception architectures

```
In [15]: def BNConv(filters, nb_row, nb_col, subsample=(1, 1), padding="same"):
             def f(input):
                 conv = Conv2D(activation="relu", kernel_size=(nb_row, nb_col), filters=filters, strides=
                        padding=padding, kernel_initializer="he_normal")(input)
                 return BatchNormalization()(conv)
             return f
```

```
In [16]: def inception_naive_module(m=1):
             def f(input):

                 # Tower A
                 conv_a = BNConv(32*m, 1, 1)(input)

                 # Tower B
                 conv_b = BNConv(32*m, 3, 3)(input)

                 # Tower C
                 conv_c = BNConv(16*m, 5, 5)(input)

                 # Tower D
                 pool_d = MaxPooling2D(pool_size=(3, 3), strides=(1, 1), padding="same")(input)

                 return merge([conv_a, conv_b, conv_c, pool_d], mode='concat', concat_axis=3)

             return f
```

5. Inception architectures



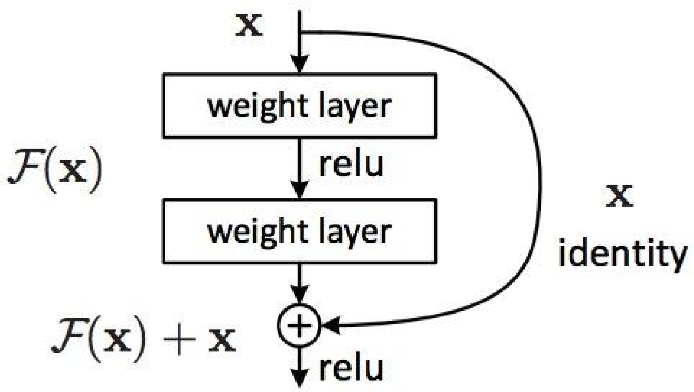
5. Inception architectures

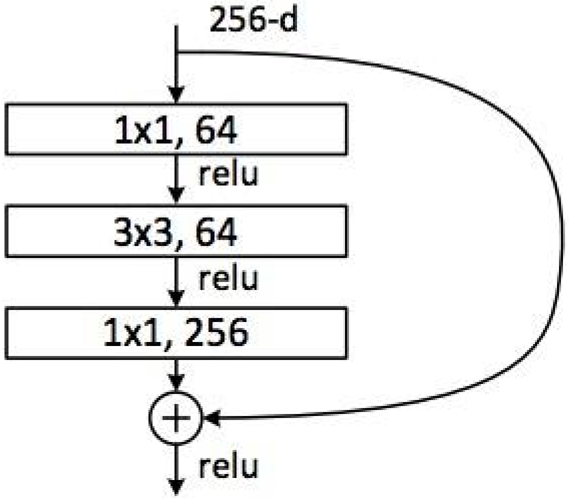
In [23]:
```python
def inception_dimred_module(m=1):
    def f(input):

        # Tower A
        conv_a = BNConv(32*m, 1, 1)(input)

        # Tower B
        conv_b = BNConv(16*m, 1, 1)(input)
        conv_b = BNConv(32*m, 3, 3)(conv_b)

        # Tower C
        conv_c = BNConv(4*m, 1, 1)(input)
        conv_c = BNConv(16*m, 5, 5)(conv_c)

        # Tower D
        # max pooling followed by compression
        pool_d = MaxPooling2D(pool_size=(3, 3), strides=(1, 1), padding="same")(input)
        conv_d = BNConv(16*m, 1, 1)(pool_d)

        return merge([conv_a, conv_b, conv_c, conv_d], mode='concat', concat_axis=3)

    return f
```

5. Inception architectures (top architecture)

```
In [49]: def inception(inception_module):

             #input in the right shape, tensorflow ordered
             _in = Input(shape=(784,))
             reshape_1 = Reshape((28,28,1))(_in)

             # go to 32 channels
             conv_0 = BNConv(32, 3, 3)(reshape_1)
             conv_0 = BNConv(32, 3, 3)(conv_0)
             pool_0 = MaxPooling2D((2, 2))(conv_0)

             # apply inception network (input: 14x14x32, output channels:96)
             module_1 = inception_module()(pool_0)

             # pool to 7x7x96
             pool_1 = MaxPooling2D((2, 2))(module_1)

             # apply inception network (input: 7x7x96, output channels:192)
             module_2 = inception_module(m=2)(pool_1)

             # pool to: 1x1x96 and flatten
             x = AveragePooling2D((7, 7))(module_2)
             x = Flatten()(x)
             x = Dropout(0.4)(x)

             # dense layer and normalization
             fc = Dense(128, activation='relu')(x)
             fc = BatchNormalization()(fc)

             _out = Dense(10, activation='softmax')(fc)
             model = Model(_in, _out)

             return model
```

6. Residual Networks





https://culurciello.github.io/tech/2016/06/04/nets.html

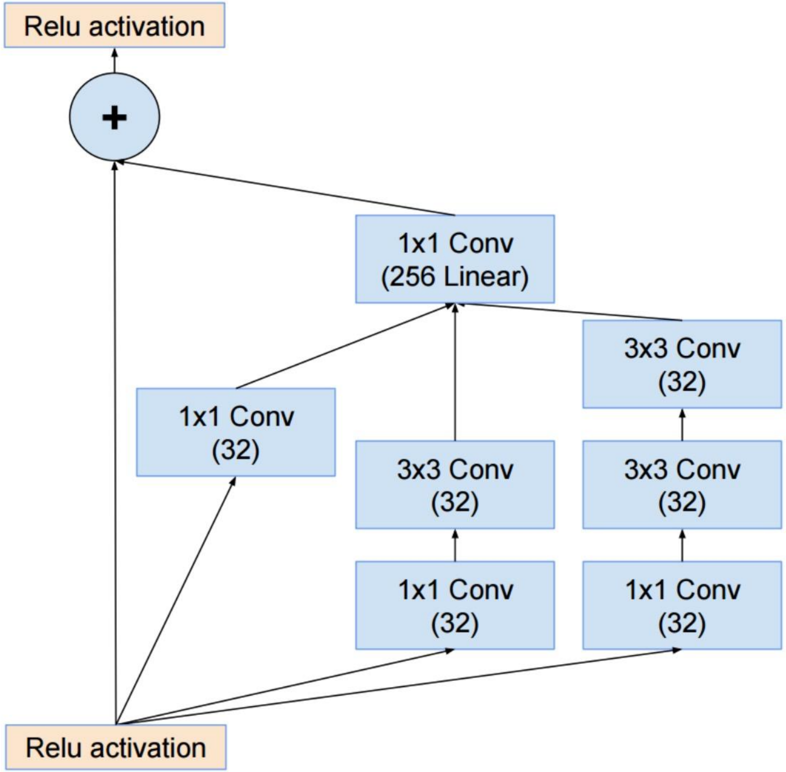
6. Residual Networks

```
In [8]: def residual_block(skip=True):
            def f(input):
                conv = Convolution2D(4,3,3,border_mode='same', activation='relu')(input)
                res  = merge([conv,input], mode='sum')
                return Activation('relu')(res) if skip else conv
            return f
```

```
In [9]: def resnet(skiplayers=3):
            #select inception module
            _in     = Input(shape=(784,))
            reshape = Reshape((28,28,1))(_in)
            res     = Convolution2D(4,3,3,border_mode='same')(reshape)

            for i in range(skiplayers):
                res = residual_block()(res)

            flat = Flatten()(res)
            flat = Dropout(0.4)(flat)

            _out = Dense(10, activation='softmax')(flat)
            model = Model(_in, _out)

            return model
```

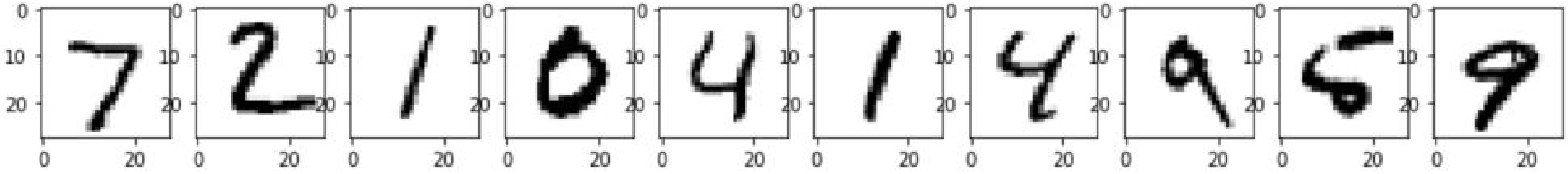
```
In [10]: model = resnet(10)
```

6. Residual + Inception Networks



7. LSTM on Images

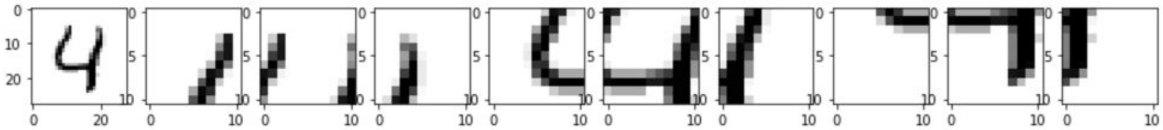
In [5]:
```
plt.figure(figsize=(15,5))
for i in list(range(10)):
    plt.subplot(1, 10, i+1)
    pixels = mnist.test.images[i]
    pixels = pixels.reshape((28, 28))
    plt.imshow(pixels, cmap='gray_r')
plt.show()
```



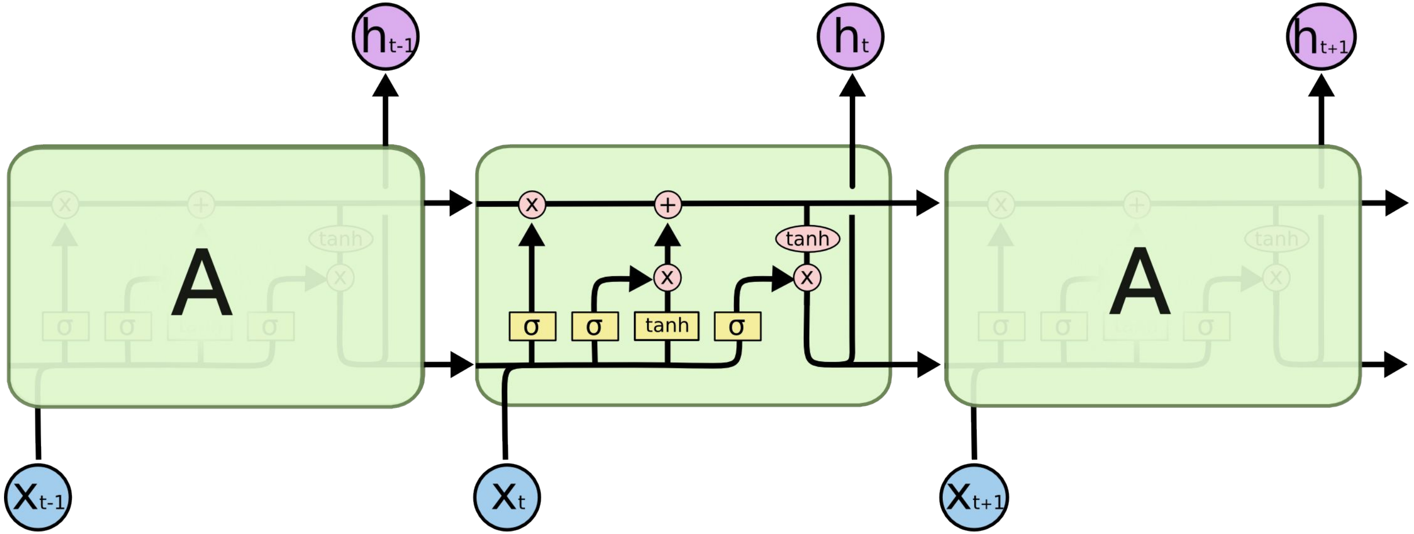
7. LSTM on Images

In [7]:
```
import math
def glimpses(pixels, n=1):
    g = []
    k = int(math.sqrt(n-1))//2
    r = list(range(-k, k+1))
    if type(n)==list:
        r = n
    for i in r:
        for j in r:
            g.append(glimpse(pixels,14+7*j,14+7*i,5))
    return np.array(g)
```

In [8]:
```
pixels = mnist.test.images[4]
plt.figure(figsize=(15,5))

# plot the full field
plt.subplot(1, 10, 1)
plt.imshow(glimpse(pixels,14,14,14), cmap='gray_r')

# plot 9 glimpses
i = 2
for g in glimpses(pixels,9):
    plt.subplot(1, 10, i)
    plt.imshow(g, cmap='gray_r')
    i += 1
plt.show()
```

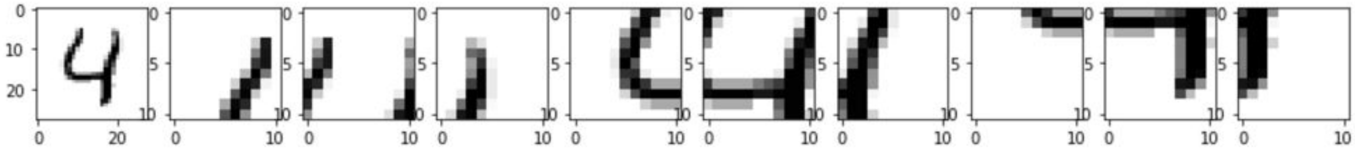


7. LSTM on Images



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

7. LSTM on Images



```
In [144]: from keras.models import Sequential
          from keras.layers import Dense
          from keras.layers import LSTM

          model = Sequential()
          model.add(LSTM(32, input_length=9, input_dim=121))
          model.add(Dense(10, activation='softmax'))

          from keras.optimizers import Adam
          model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.01), metrics=["accuracy"])
```

```
In [145]: from keras.utils.layer_utils import print_summary
          print_summary(model.layers)
```

7. LSTM on Images

In [146]:
```
def sequence(x,n):
    return glimpses(x,n).reshape((n*121))
```

In [147]:
```
# prepare the train/test input as a tensor of shape 55000, 9, 121
train_sequences = np.apply_along_axis(sequence, 1, mnist.train.images,9).reshape(-1,9,121)
test_sequences  = np.apply_along_axis(sequence, 1, mnist.test.images,9).reshape(-1,9,121)
```

In [148]:
```
model.fit(train_sequences, mnist.train.labels,
          batch_size=250, nb_epoch=5, verbose=1,
          validation_data=(test_sequences, mnist.test.labels))
```

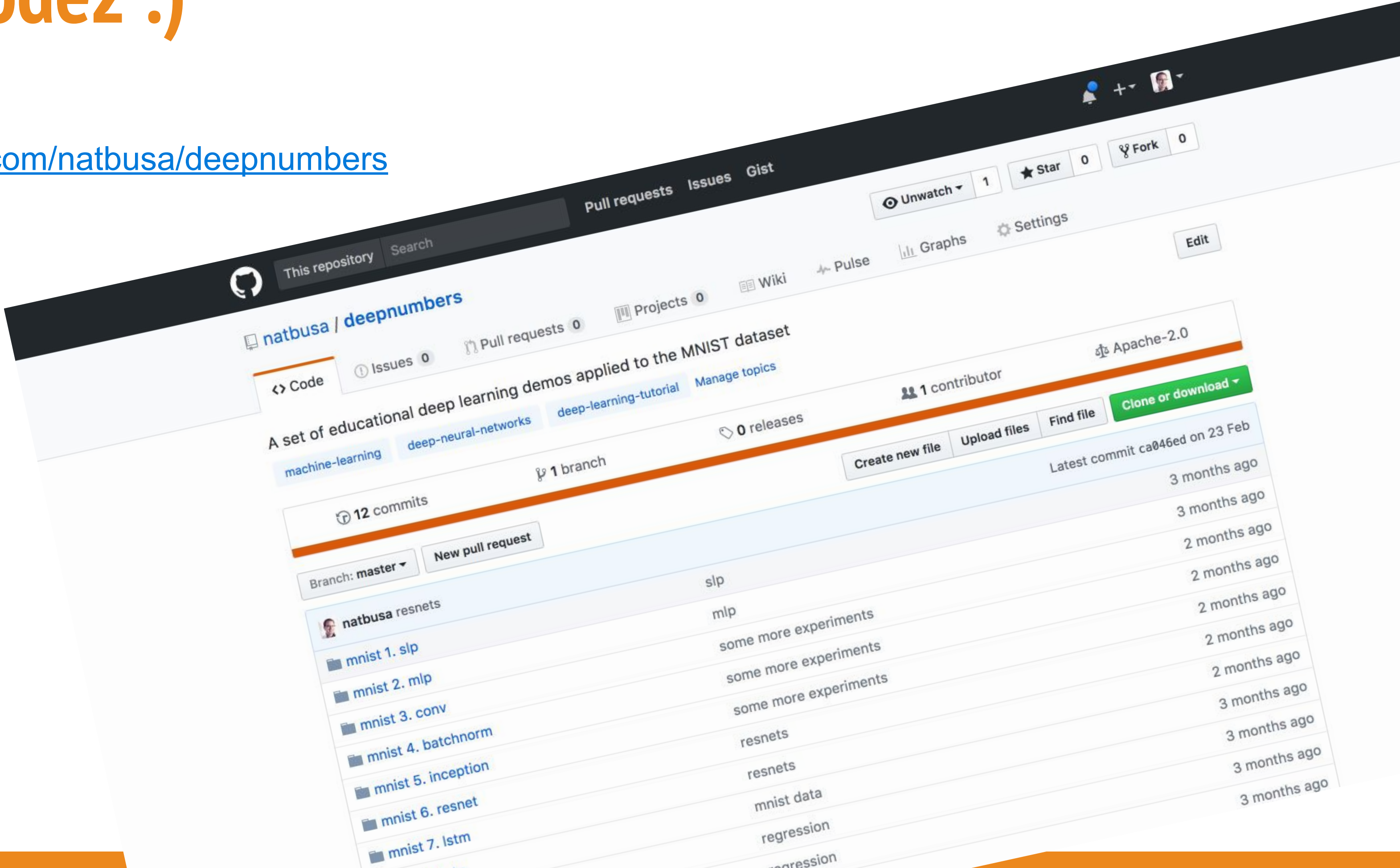
7. LSTM on ConvNets (bonus slide)

In [215]:

```
# Convolutional + Multilayer LSTM
from keras.models import Model

from keras.layers import TimeDistributed
from keras.layers import Flatten, Reshape
from keras.layers import Convolution2D, MaxPooling2D, BatchNormalization

def conv_net():
    def f(_in):
        # go to 32 channels
        layer = Convolution2D(16, 3, 3, border_mode="same", activation="relu")(_in)
        layer = Convolution2D(16, 3, 3, border_mode="same", activation="relu")(_in)
        layer = MaxPooling2D((2, 2))(layer)
        layer = Flatten()(layer)
        _out  = Dense(64, activation='relu')(layer)
        return _out
    return f

#create the conv_model
x = Input(shape=(11, 11, 1))
conv_model = Model(x,conv_net()(x))

# build the top model
model = Sequential()

#prep for convolution, keep the timestep as first dimension (after the implicit batch dim)
model.add(Reshape((9,11,11,1), input_shape=(9,121)))

# time distributed on the convolutional part
model.add(TimeDistributed(conv_model))

# temporal model (64 is de output dim of the conv_model)
model.add(LSTM(32, return_sequences=True, input_length=9, input_dim=64))
model.add(LSTM(32, return_sequences=True))
model.add(LSTM(32))

# last layer
model.add(Dense(10, activation='softmax'))

from keras.optimizers import Adam
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.01), metrics=["accuracy"])
```

All the codez :)

https://github.com/natbusa/deepnumbers



Meta- References

… just a few articles, but extremely dense in content.
A must read!

https://keras.io/
http://karpathy.github.io/neuralnets/

https://culurciello.github.io/tech/2016/06/04/nets.html
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b