Microsoft

A Quick Primer on TensorFrames Apache Spark and TensorFlow Together

Tom Drabas
Data Scientist II

Denny Lee
Principal Program Manager

Session Goals

- What is Deep Learning?
- A primer on feature learning
- What is feature engineering?
- What is TensorFlow?
- Build a Deep Learning model to recognize objects in images
- Introducing TensorFrames
- TensorFrames – quick start

Tom Drabas

Data Scientist II, Core Data Science (WDG)

All-around geek and nerd

Close to 15 years of industry experience: analytics, operations research and data science.

Worked in airlines, telecommunications, logistics and technology

Interested in big data, algorithms, neural networks, and… vintage electronics

Denny Lee

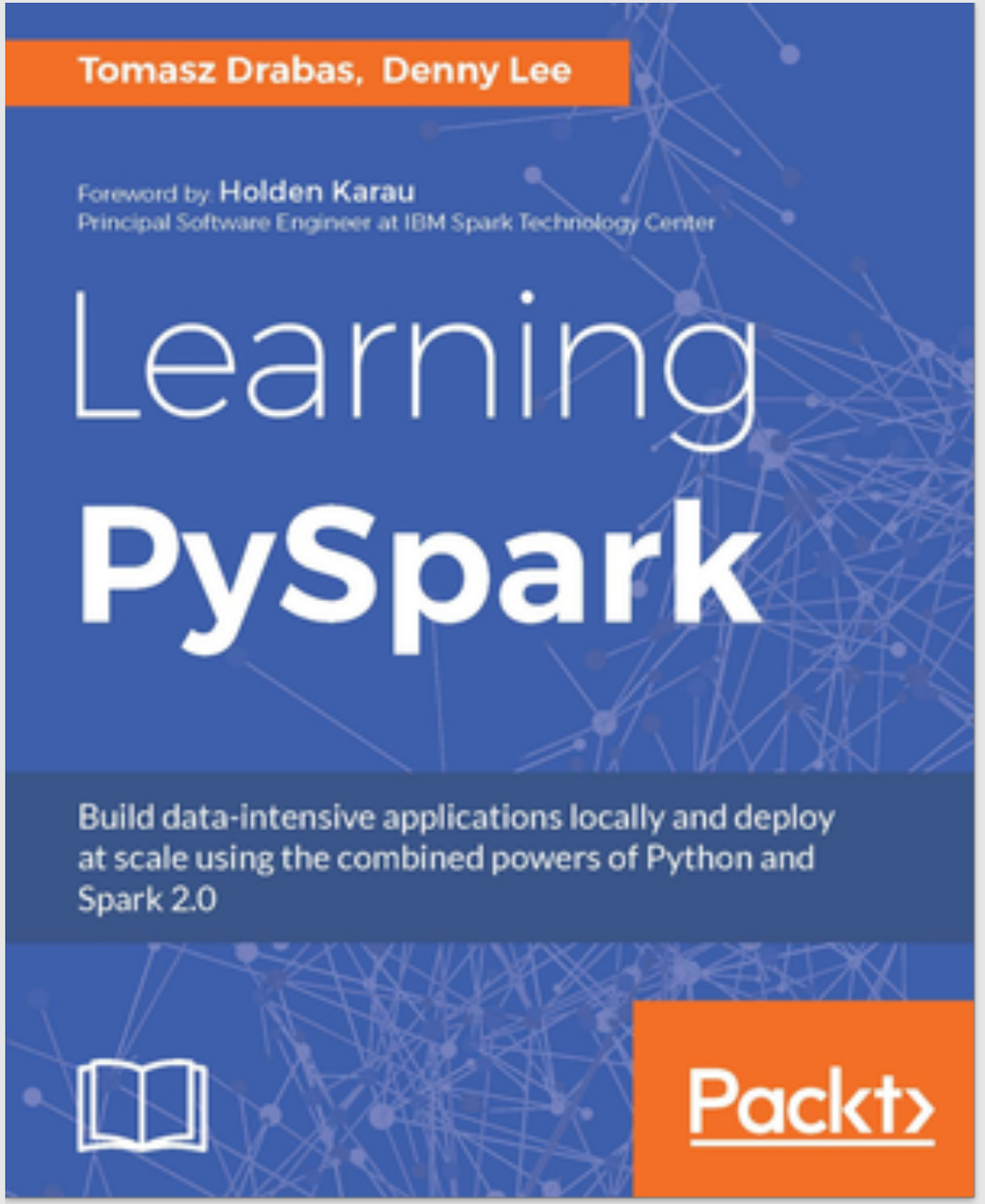
Principal Program Manager, Azure Cosmos DB

Former:

- Technology Evangelist @ Databricks
- Senior Director of Data Sciences Engineering at Concur (now part of SAP)
- Principal Program Manager at Microsoft

Hands-on Data Engineer, Architect more than 15y developer internet-scale infrastructure for both on-premises and cloud including Bing's Audience Insights, Yahoo's 24TB SSAS cube, and Isotope Incubation Team (HDInsight).

Jump Start into Python and Apache Spark



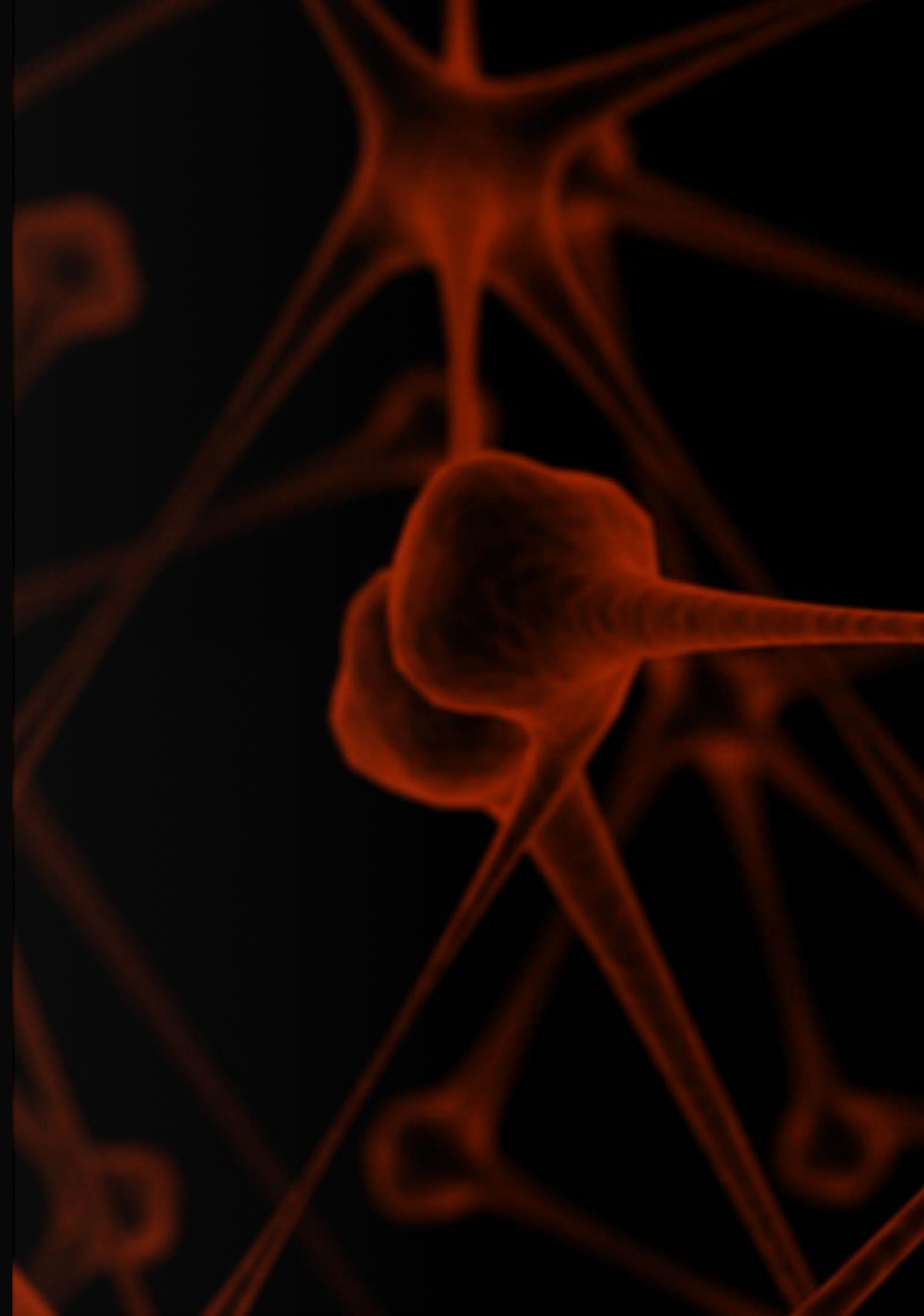
Many Thanks!

Tim Hunter
Software Engineer
Databricks

Joy Qiao,
Senior Solution Architect
Microsoft Algorithms and Data Science Solutions

Bilal Obeidat
Solution Architect
Databricks

Prerequisties



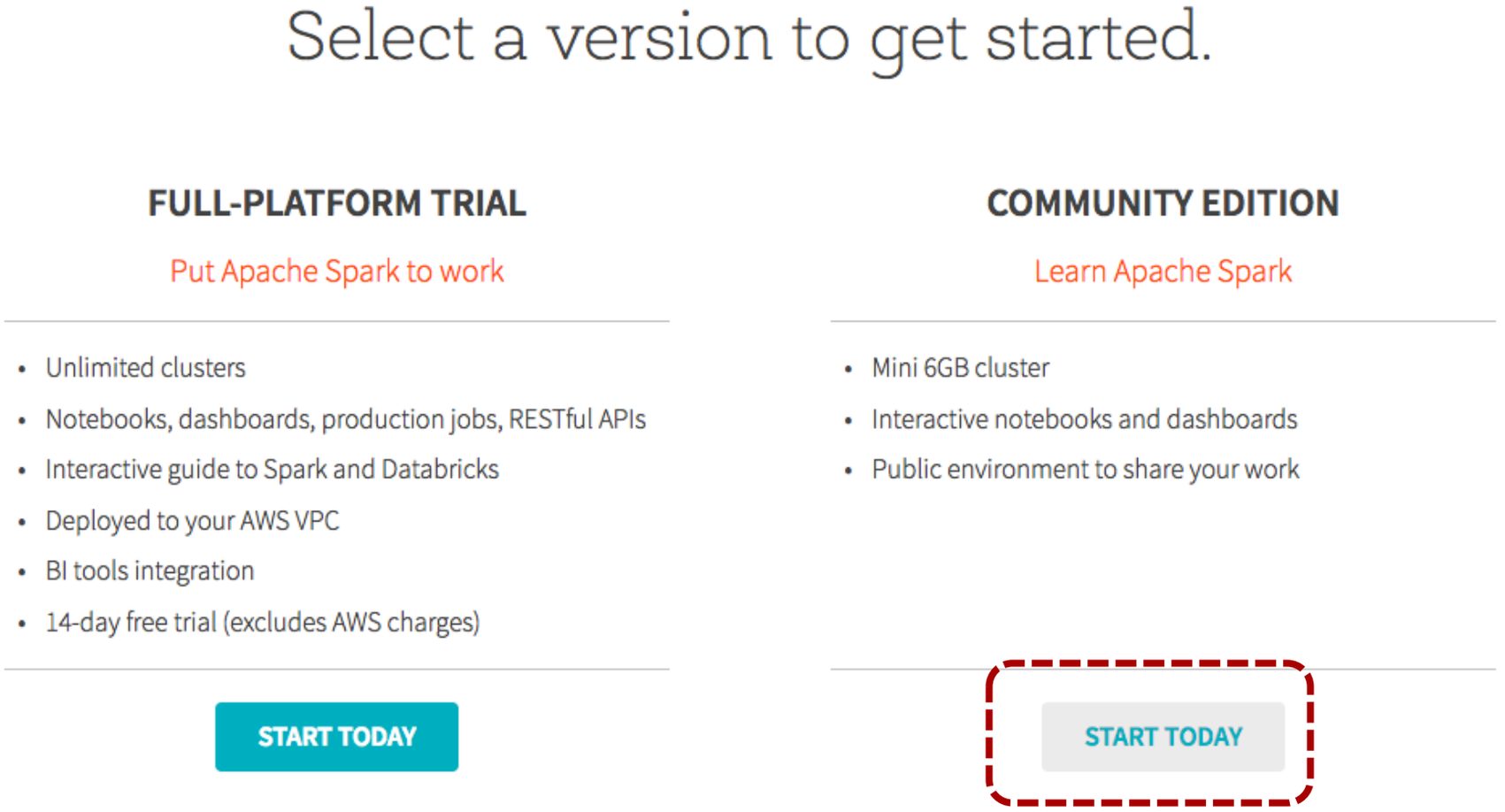
Tutorial Prerequisites

- PySpark
  - As of Spark 2.2-ish, pip install pyspark

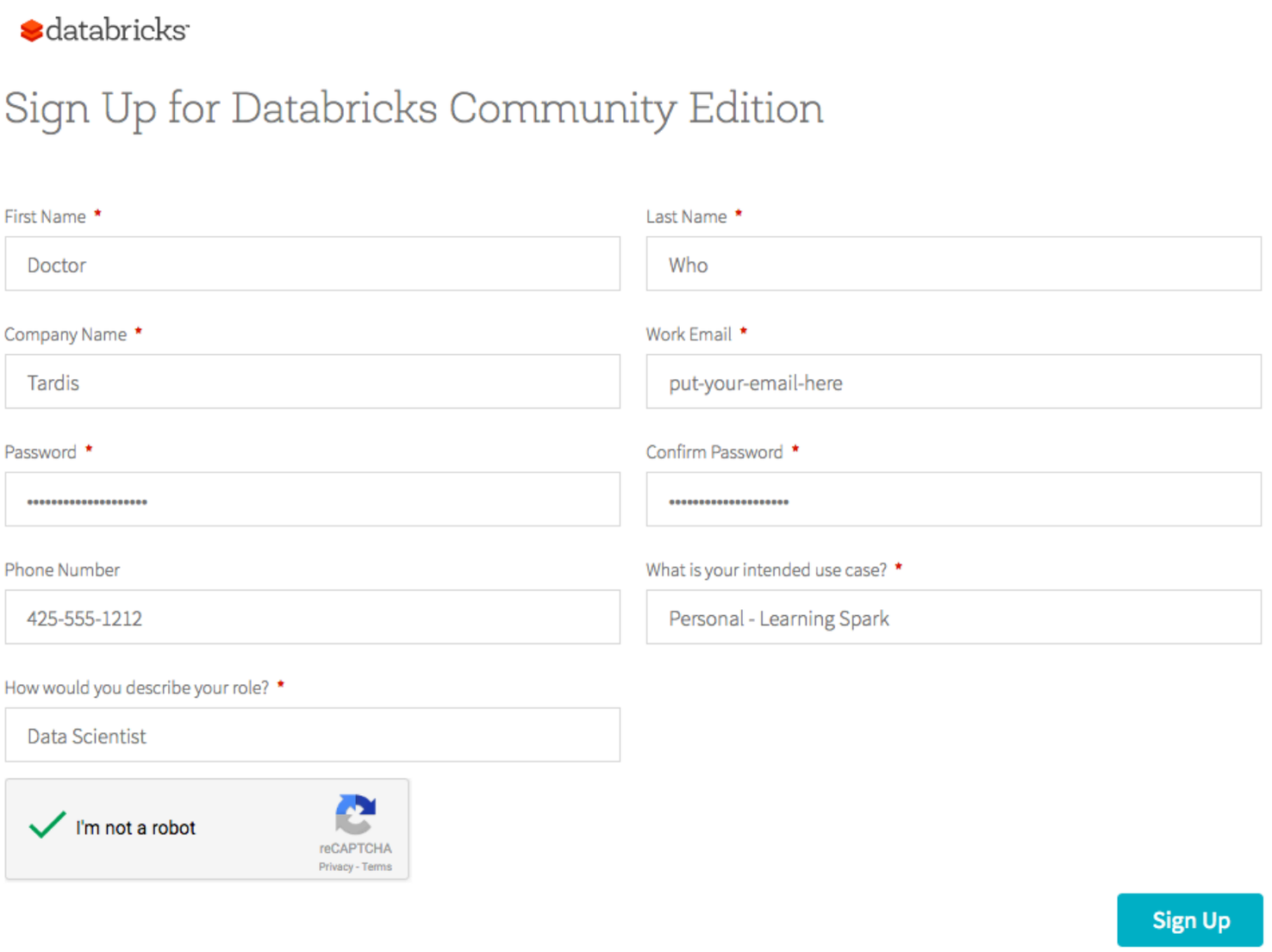
- Databricks Community Edition
  - https://databricks.com/try-databricks

Get Databricks Community Edition

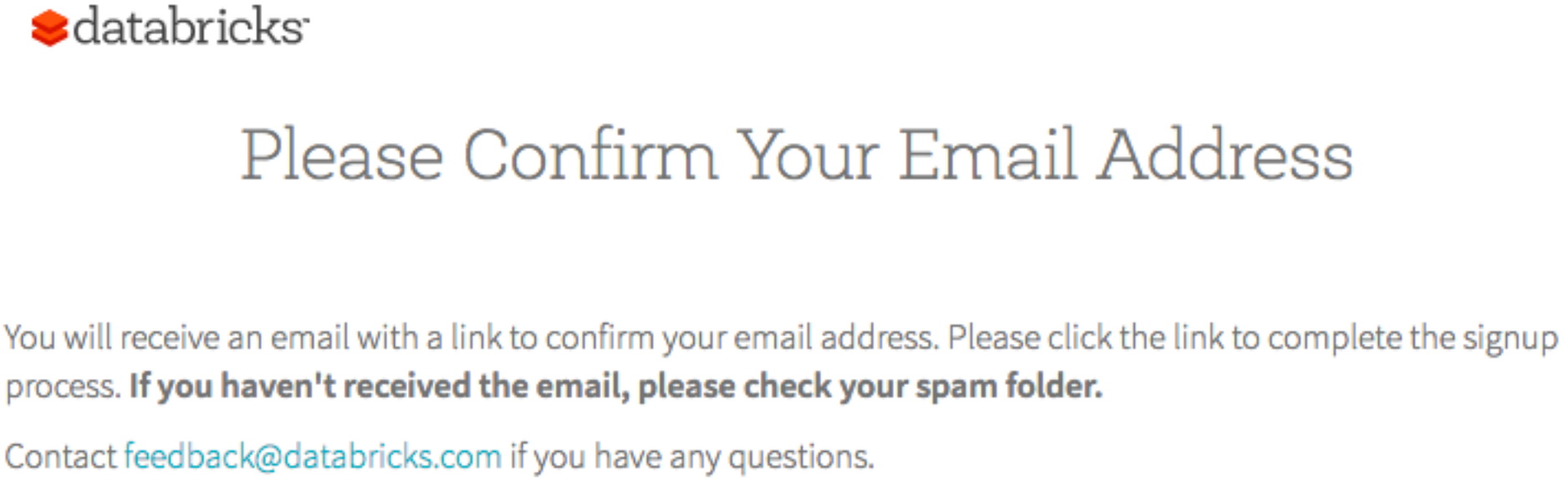
http://databricks.com/try-databricks



Sign Up for Databricks Community Edition

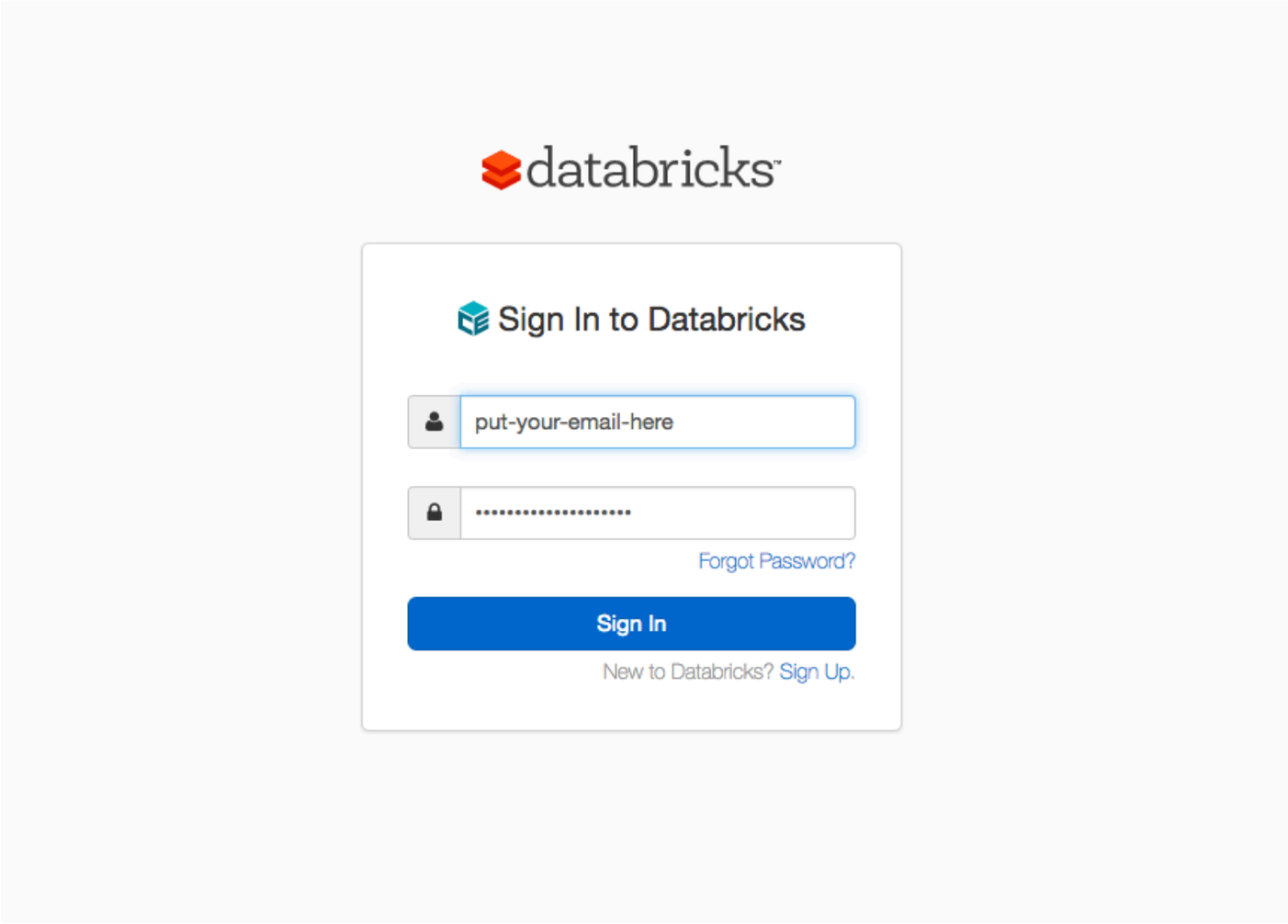


Confirm your email address



Sign in!

https://community.cloud.databricks.com



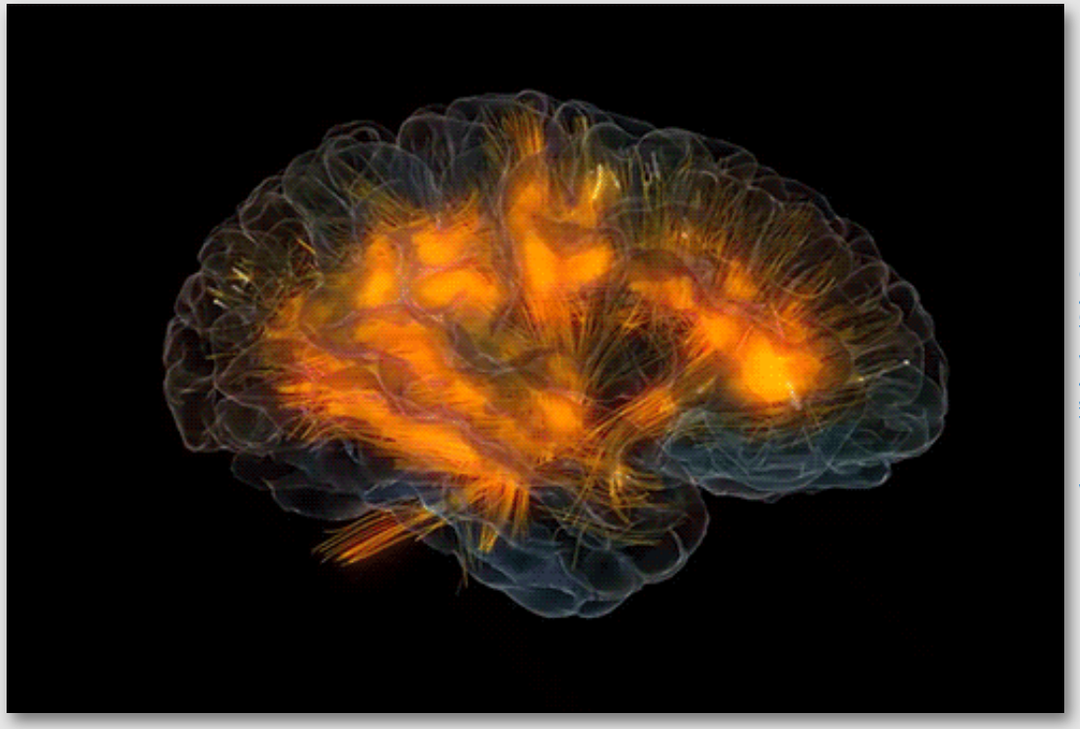
What are artificial neural networks?

source: http://bit.ly/2jjzFyt

Artificial Neural Networks

An imitation of human's brain

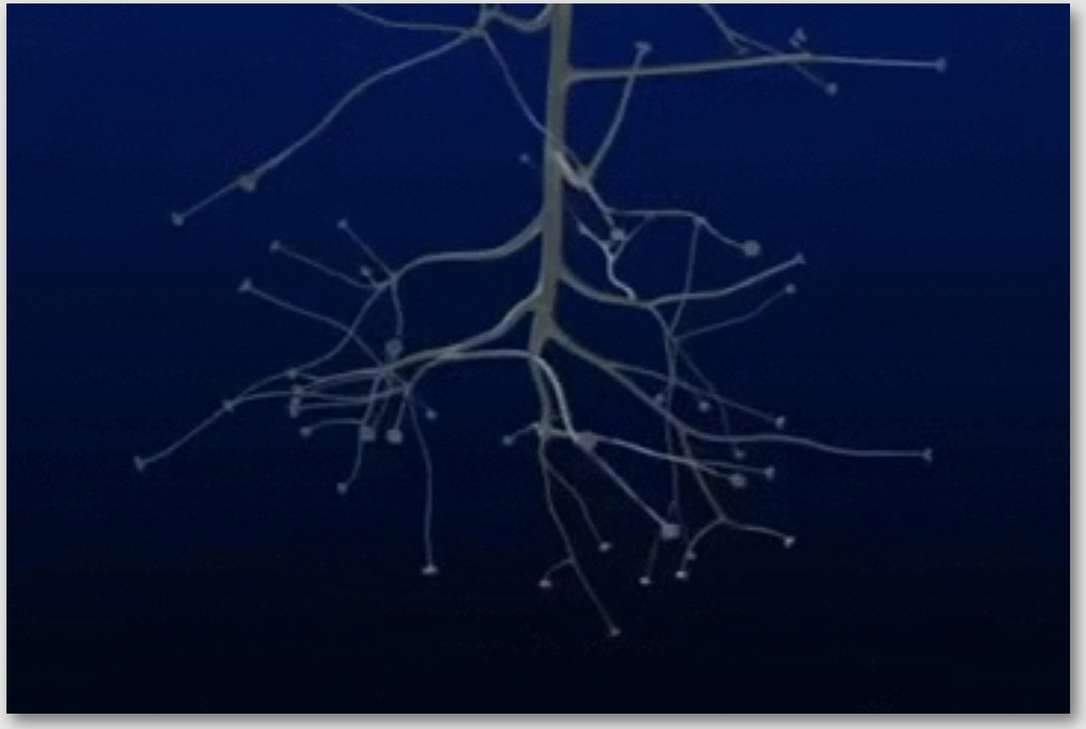
- Dense net of simple structures
- Around 100 billion neurons
- Each connected to ~10k other neurons
- 10^{15} synaptic connections



source: http://bit.ly/2jc9mYw

A neuron at work

- *Dendrites* receive signals
- Neuron's *cell body* acts as an accumulator
- If energy level in the neuron's body exceeds certain level it fires a short pulse through the *axon* ended with *synaptic terminals*



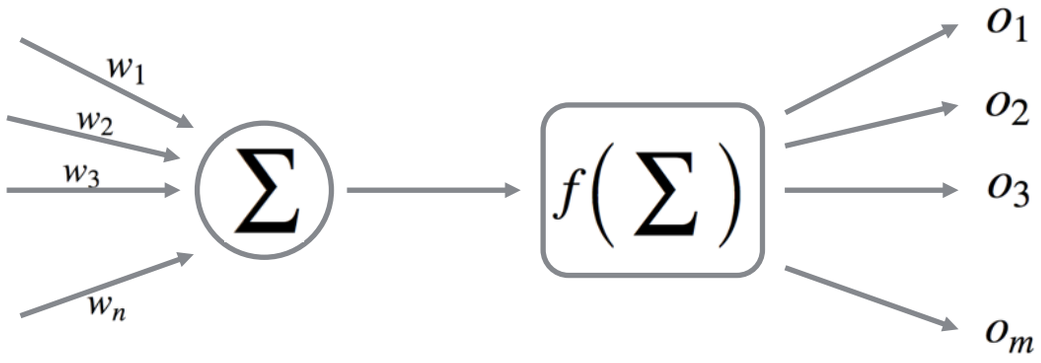
source: http://bit.ly/2jjGlNd

Conceptual mathematical model

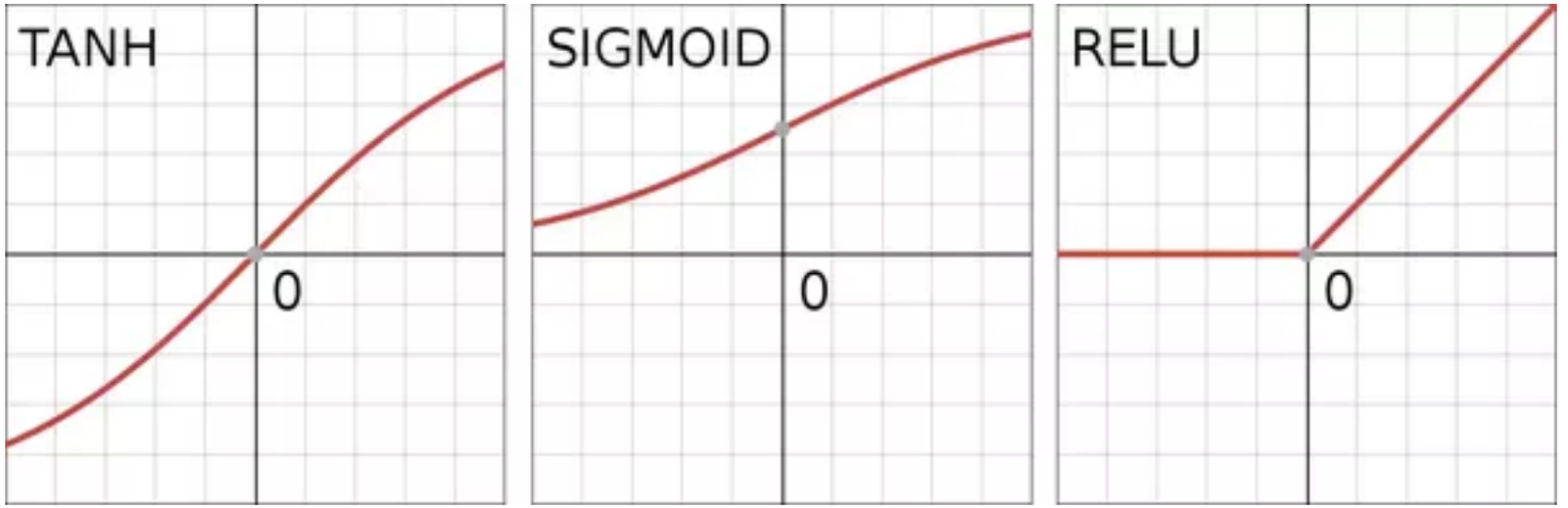
- Receives input from *n* sources
- Computes weighted sum

$$h_1 = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$$

- Passes through an *activation function*
- Sends the signal to *m* succeeding neurons



Activation functions



- Bias (threshold) activation function was proposed first
- Sigmoid and tanh introduce non-linearity with different codomains
- ReLu is the latest and greatest due to backprop (more later)

Source: http://bit.ly/2tKhbMS

Artificial Neural Network

- A layer is a set of neurons
- Typically consists of input, hidden and output layers
- Input layer acts as an interface
- The neurons in hidden layer(s) and the output modify data
- A black-box model

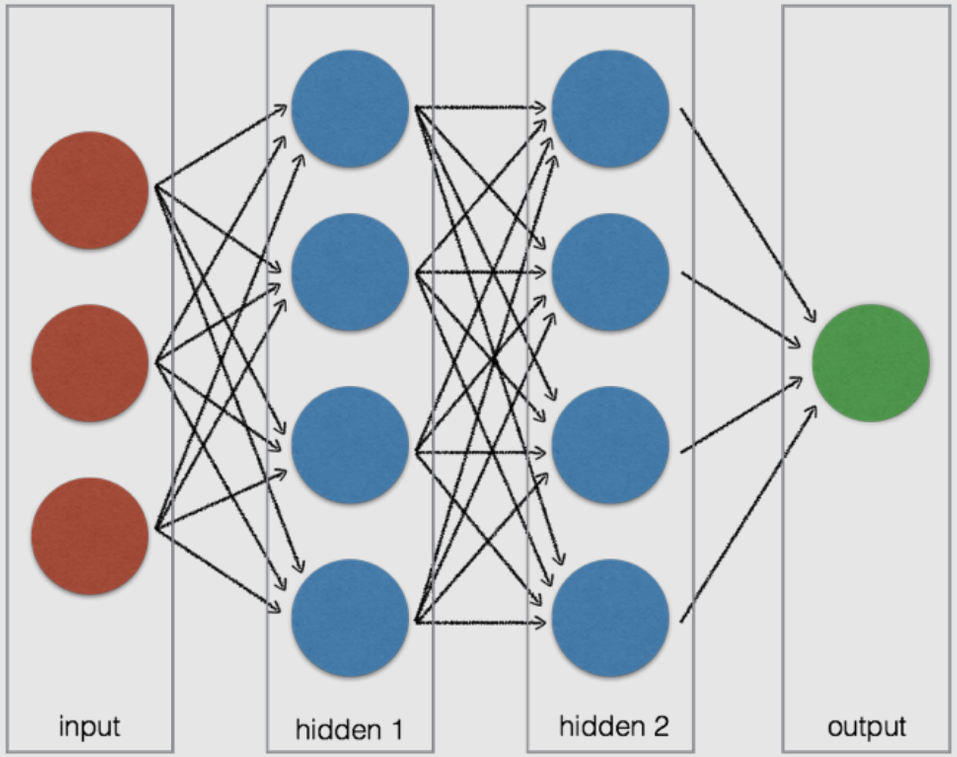
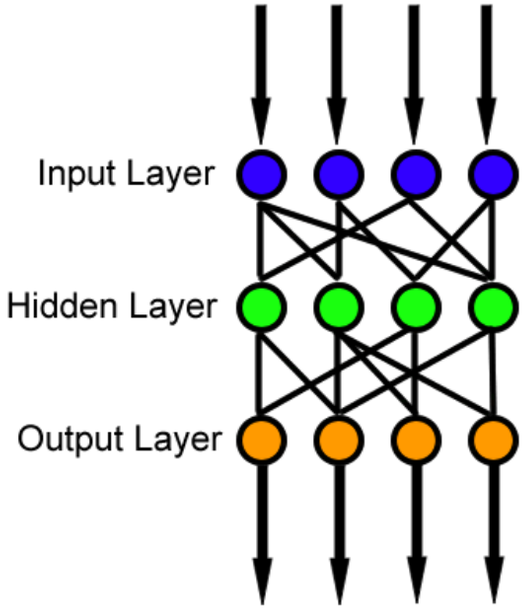


Image sources linked

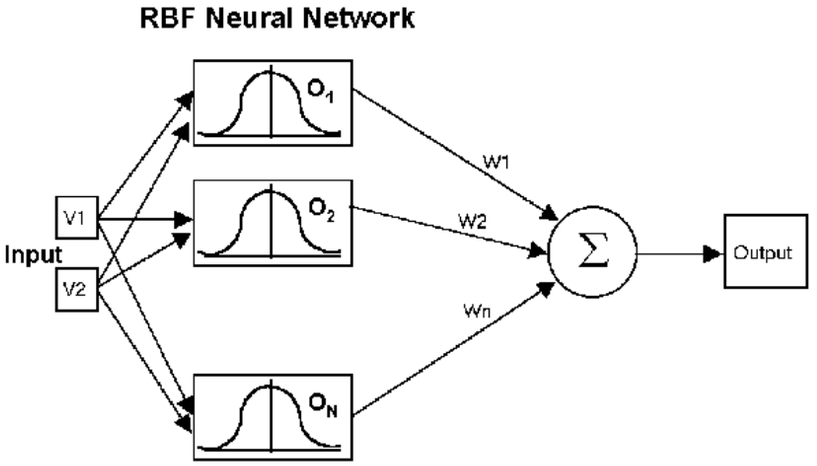
Types of neural networks

Some examples

Single- or multi-layer [feedforward networks]



[Radial Basis Function]



[Recurrent Neural Networks]

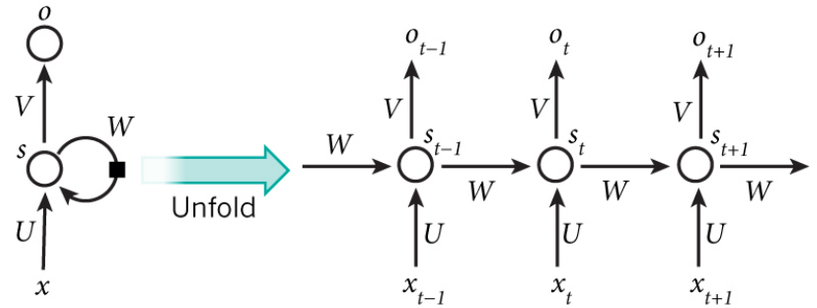
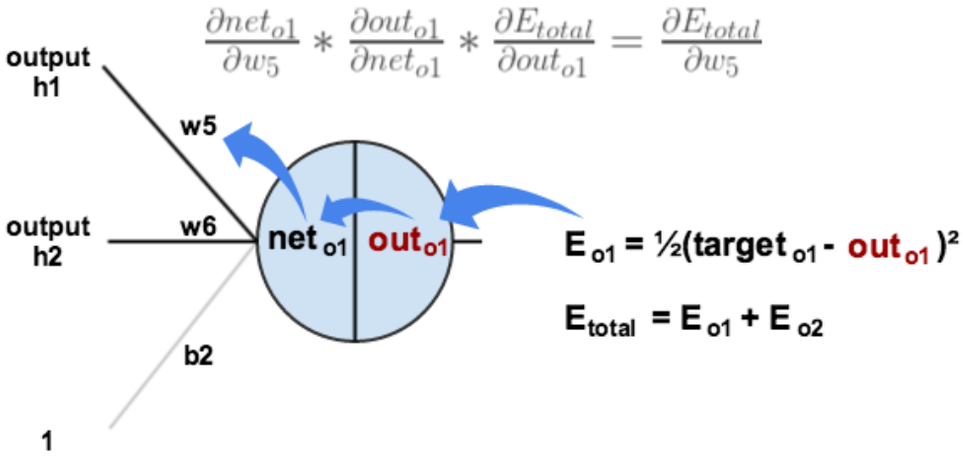


Image sources linked

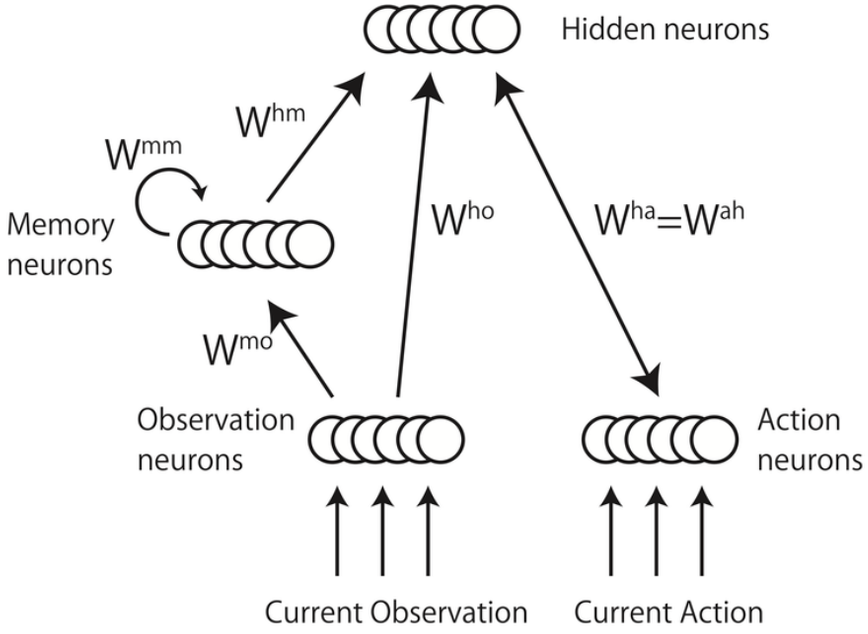
Learning

Some examples

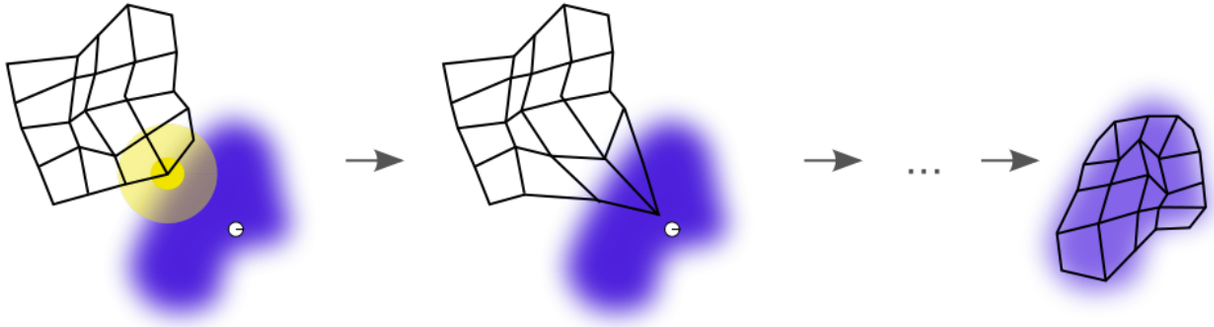
Backpropagation



Memory based learning

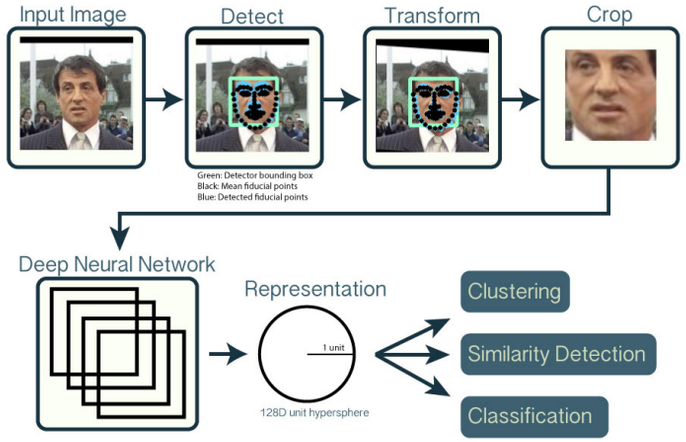


Self-organizing maps (competitive learning)

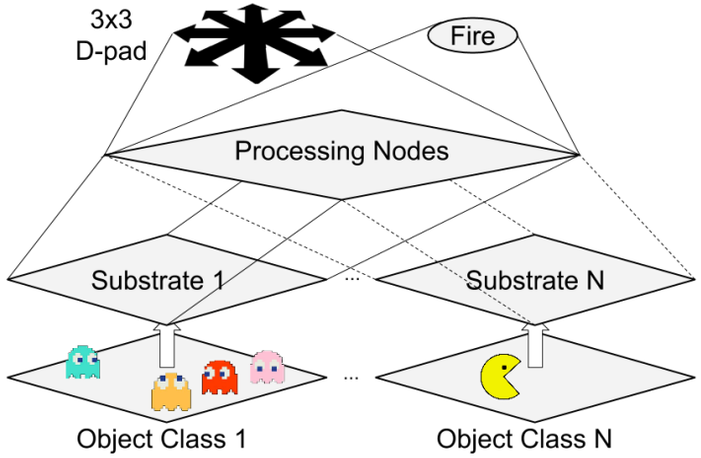


Applications of neural networks

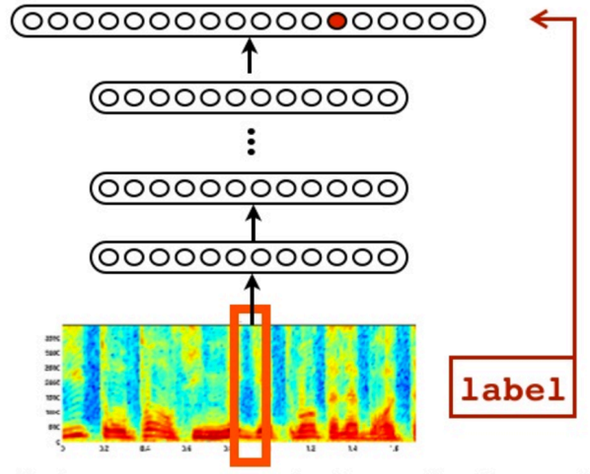
Object Recognition: Facial



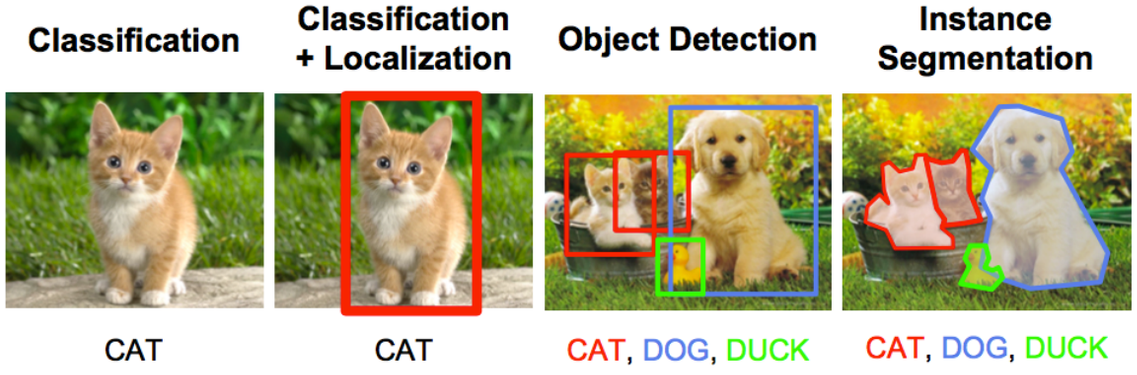
Game Playing



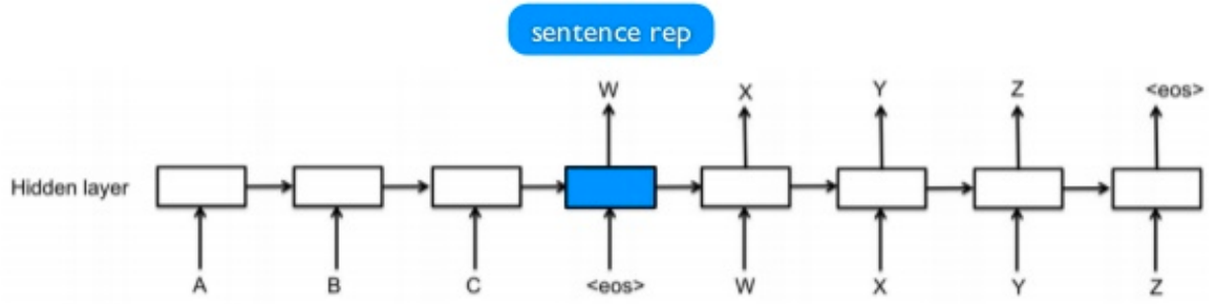
Object Recognition: Speech



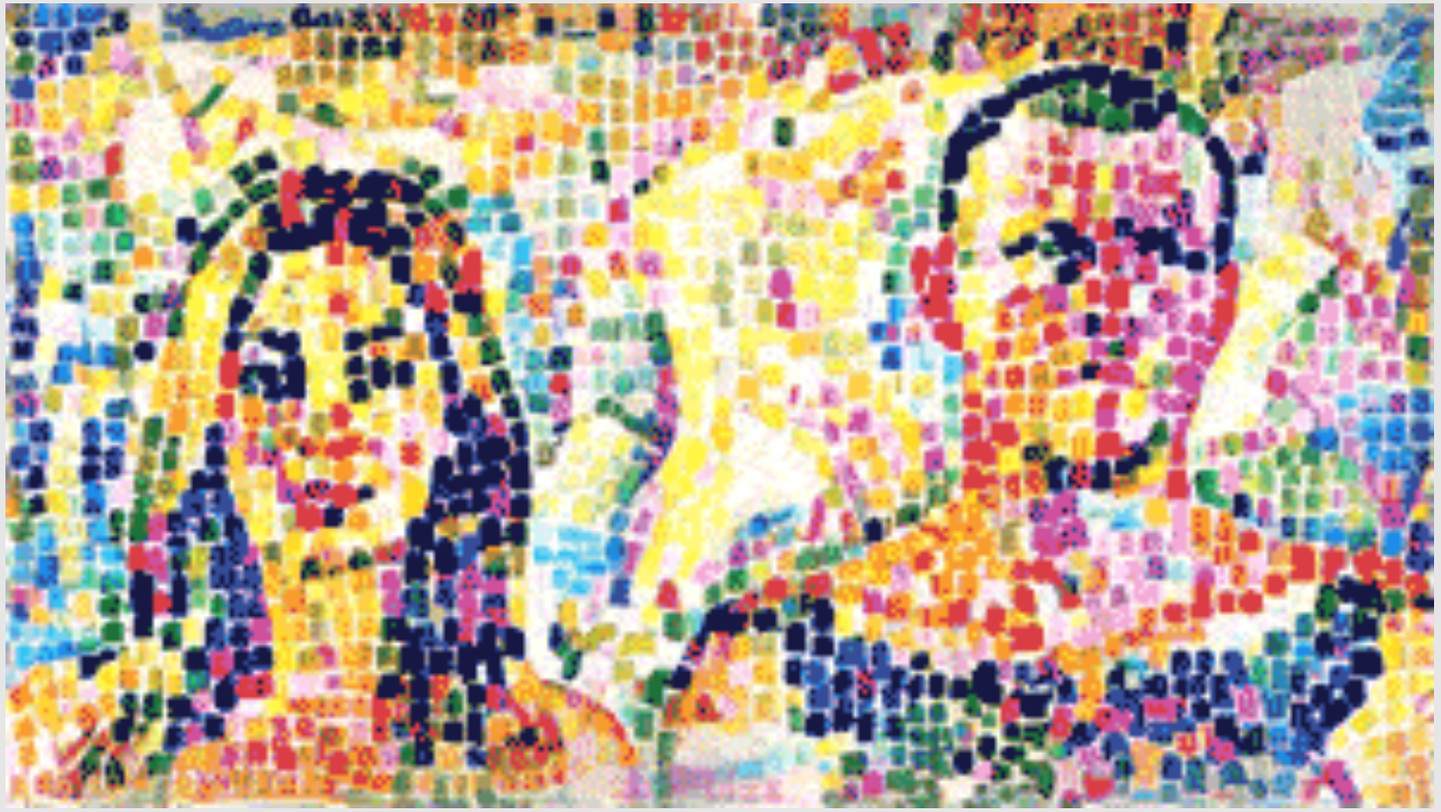
Object Classification



Language Translation



What is Deep Learning?



source: http://bit.ly/2jcquh5

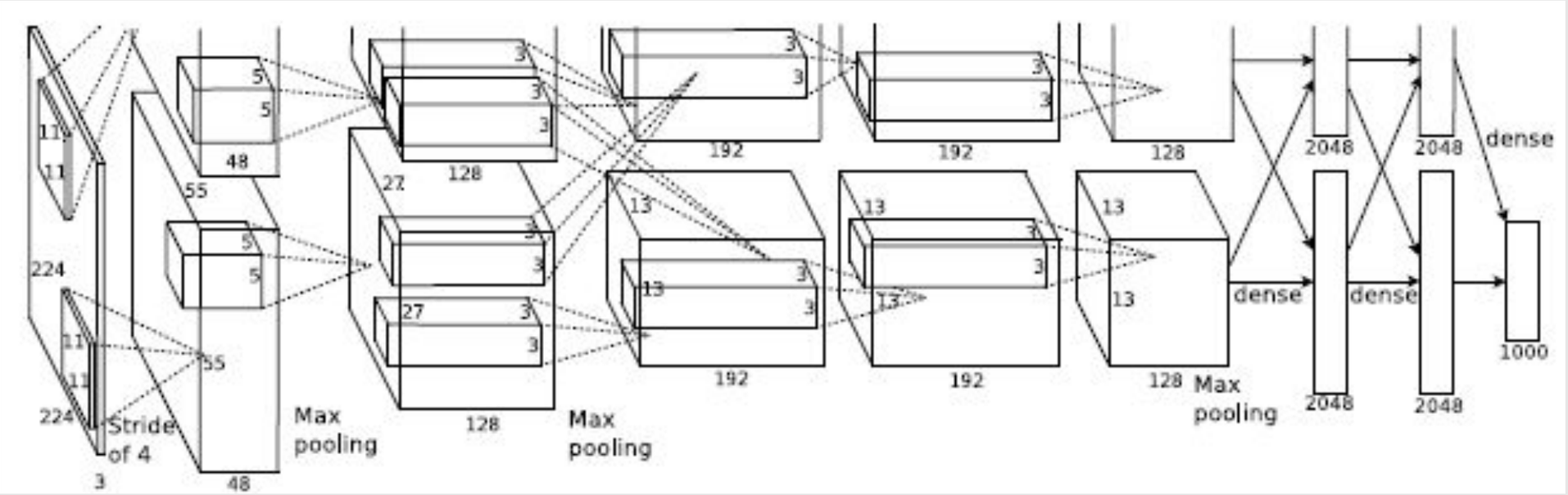
Deep Learning

The next step in AI

Deep learning

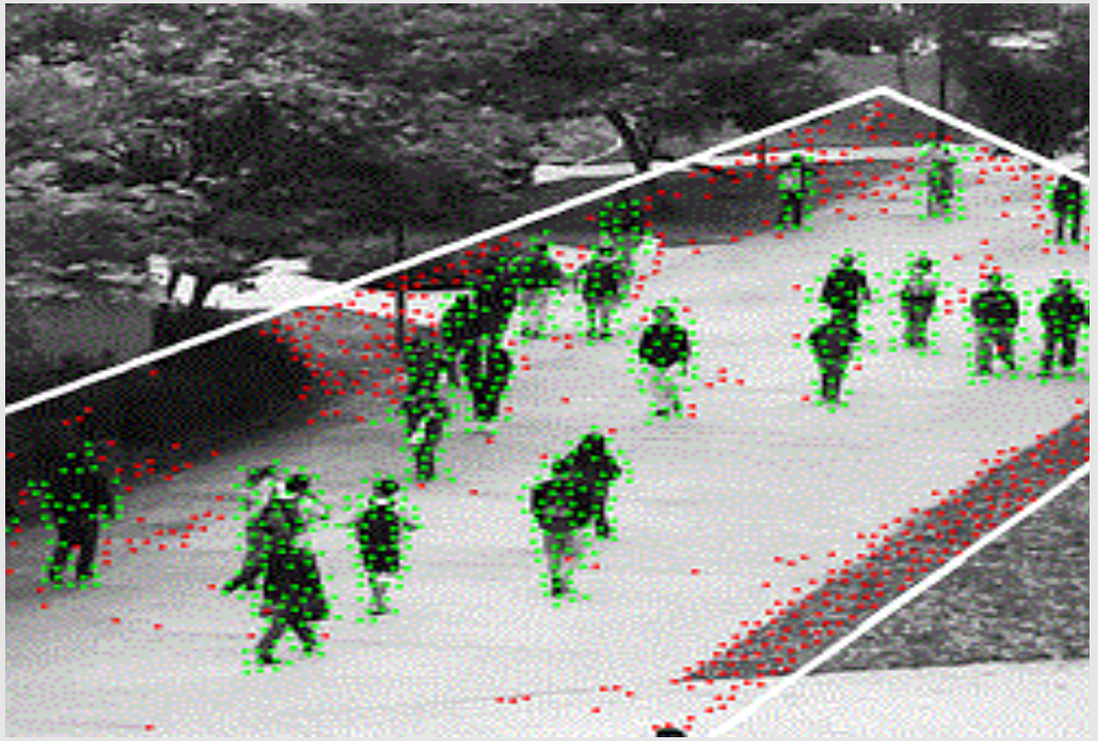
- Vast and sophisticated ANN structure
- Tens, hundreds, thousands… of hidden layers
- Ability to not only map the input to the output but recognize what inputs are needed for the task

Example: AlexNet (ImageNet 2012 winner); Source: http://bit.ly/2tNa3Ps



(Automated) Feature learning

- Determine which features to use for a given task
- Ability to (implicitly) perform feature transformations and dimensionality reduction
- Automated feature engineering and learning eliminates the need for human intervention

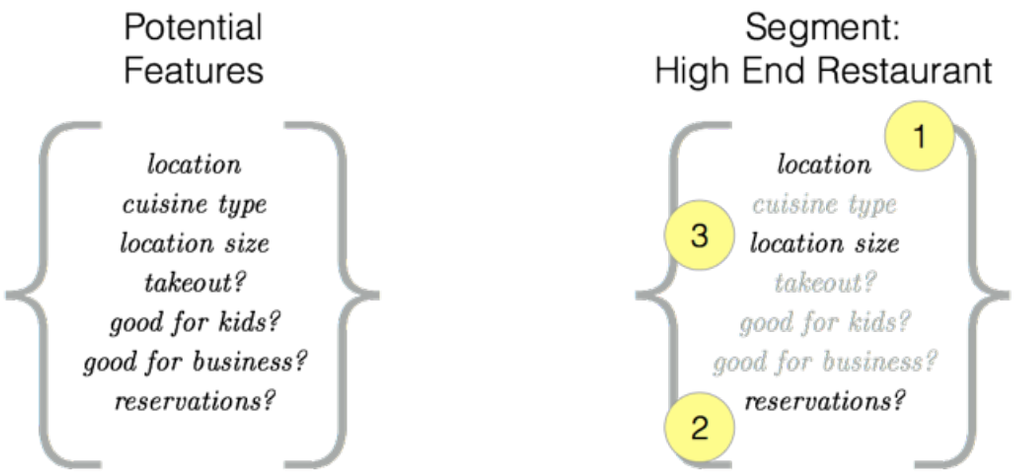


source: http://bit.ly/2jwytoN

In practice

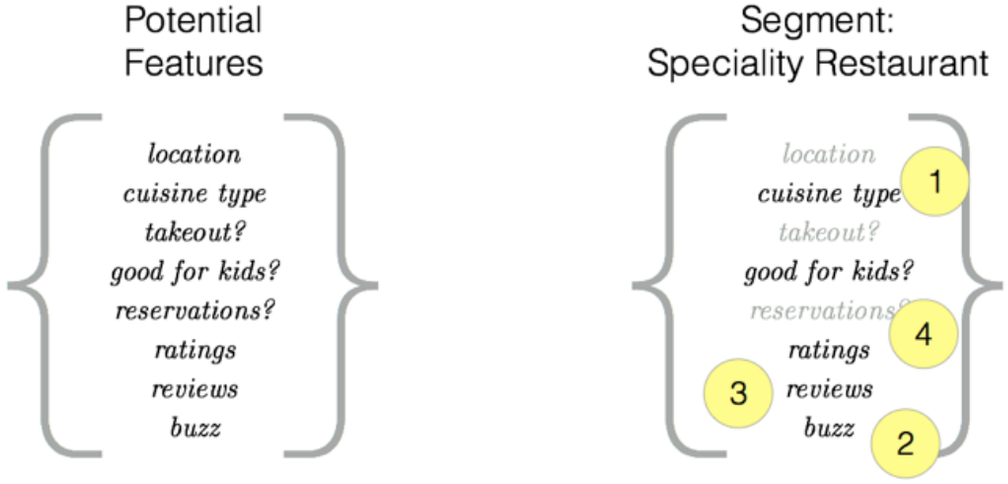
High-end restaurant

Key features are often related to location, ability to make reservations for large parties, and diversity of wine list

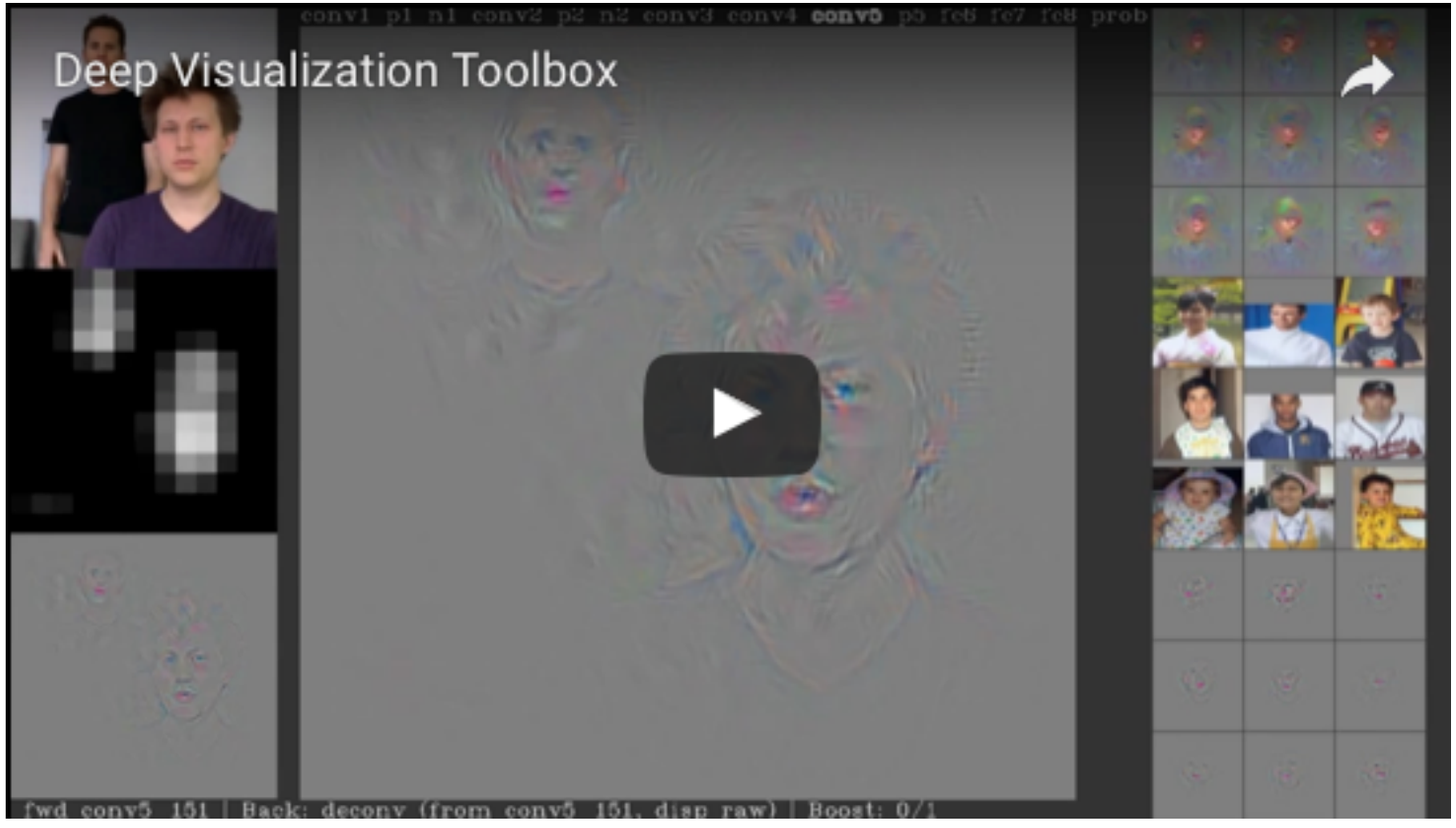


Specialty restaurant

Focus is on the reviews, ratings, social media buzz, and possibly if the restaurant is good for kids



Deep Visualization Toolbox



Demonstrate different layers of the conv net neuron activation

https://github.com/yosinski/deep-visualization-toolbox

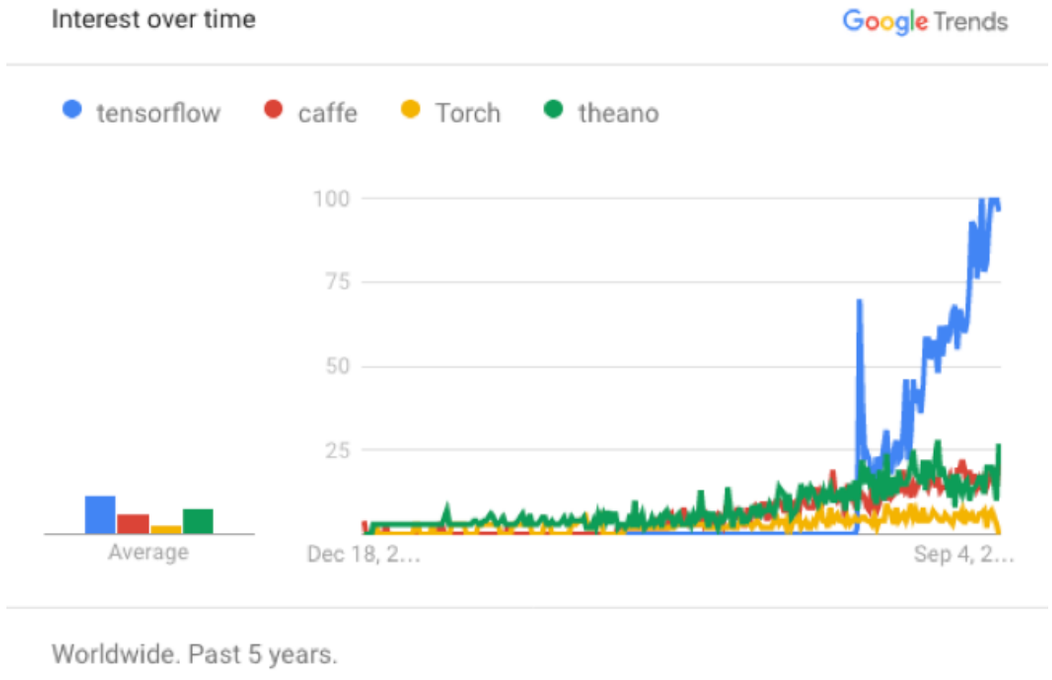
TensorFlow

Technology from Google

source: http://bit.ly/2itbqJT

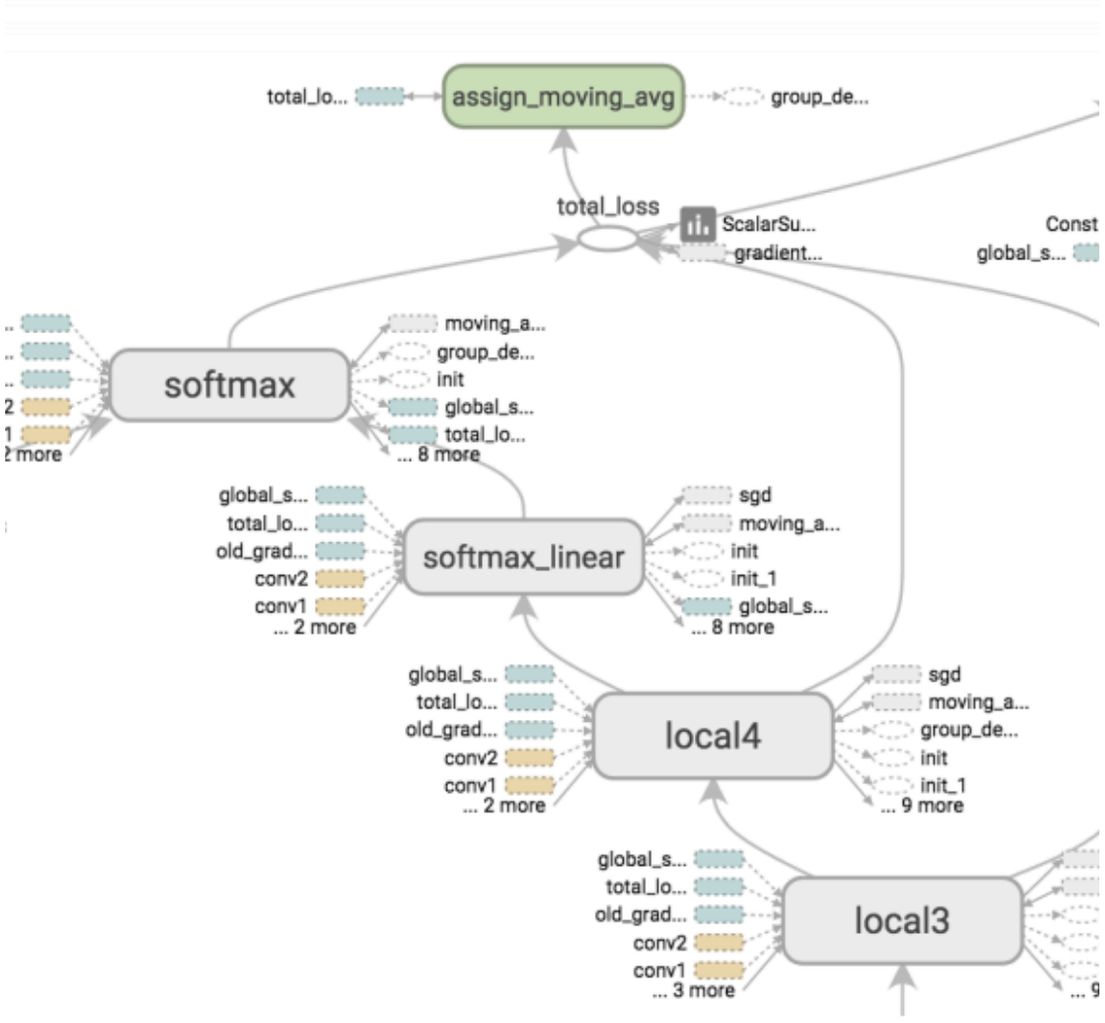
What is TensorFlow

- Open source software library for numerical computation using data flow graphs
- Based loosely on neural networks
- Built in C++ with a Python interface
- Quickly gained high popularity
- Supports GPU computations



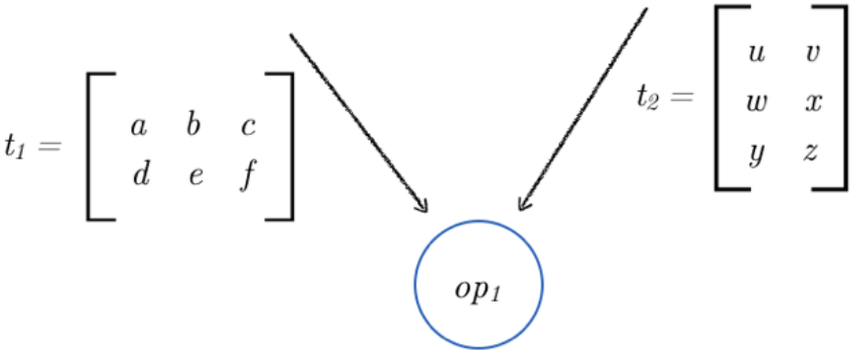
Data Flow Graph

- Nodes represent mathematical operations
- Edges are multidimensional arrays (tensors)
- Thus, TensorFlow is a flow of multidimensional arrays (tensors) between the mathematical operations (nodes)
- i.e. flow of *tensors*



Concept

- t1 is a 2x3 matrix
- t2 is a 3x2 matrix
- The op1 is a multiplication operator



- The above can be represented in a matrix form

$$op_1 = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} * \begin{bmatrix} u & v \\ w & x \\ y & z \end{bmatrix}$$

Example (pure Python)

1. Import TensorFlow

2. Prepare the tensors and operator

3. Feed the data

4. And get…

```
In [1]: import tensorflow as tf

In [2]: t1 = tf.placeholder(tf.float32)
        t2 = tf.placeholder(tf.float32)

        op = tf.matmul(t1, t2)

In [3]: m1 = [[3., 2., 1.]]
        m2 = [[-1.], [2.], [1.]]

        with tf.Session() as s:
            print(s.run([op], feed_dict={t1: m1, t2: m2}))
```

```
[array([[ 2.]], dtype=float32)]
```

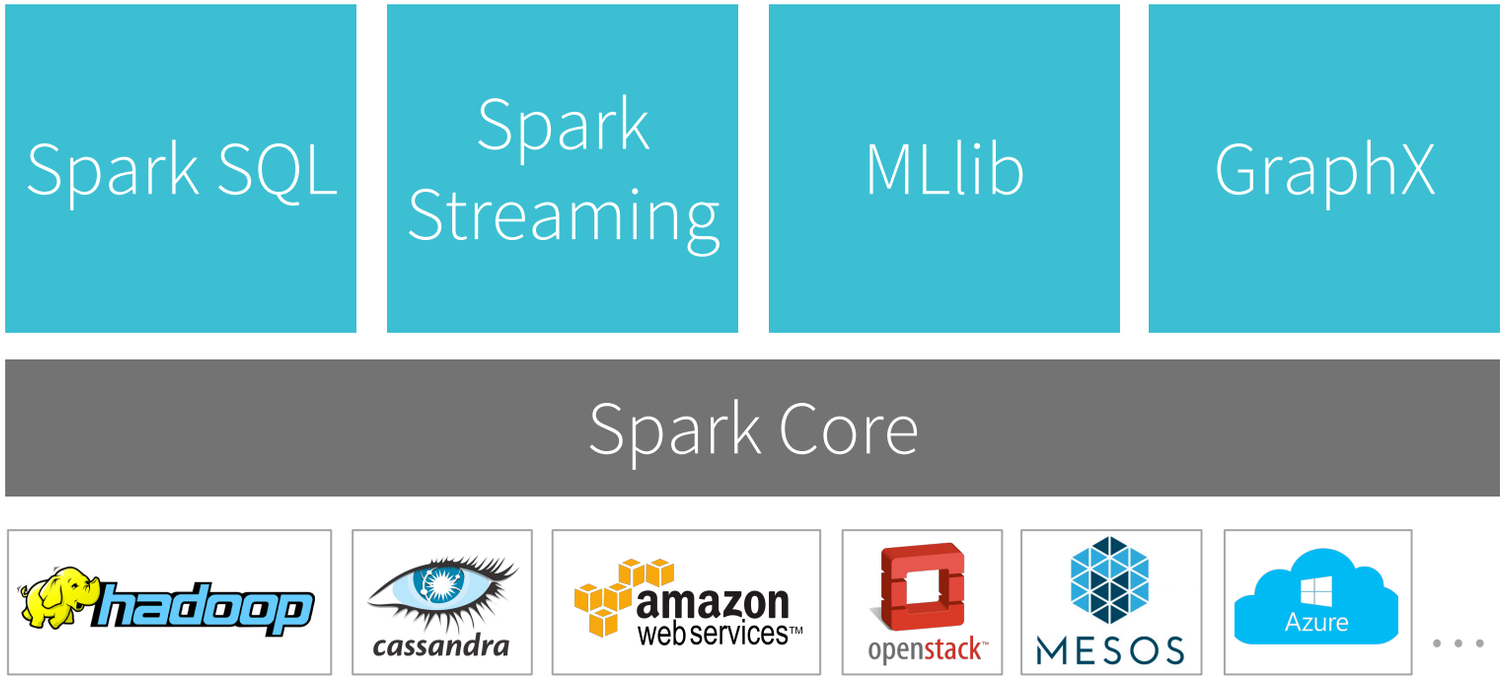
Microsoft

source: http://bit.ly/2jRN1mj

TensorFrames (experimental)

Built for speed!

Apache Spark Engine



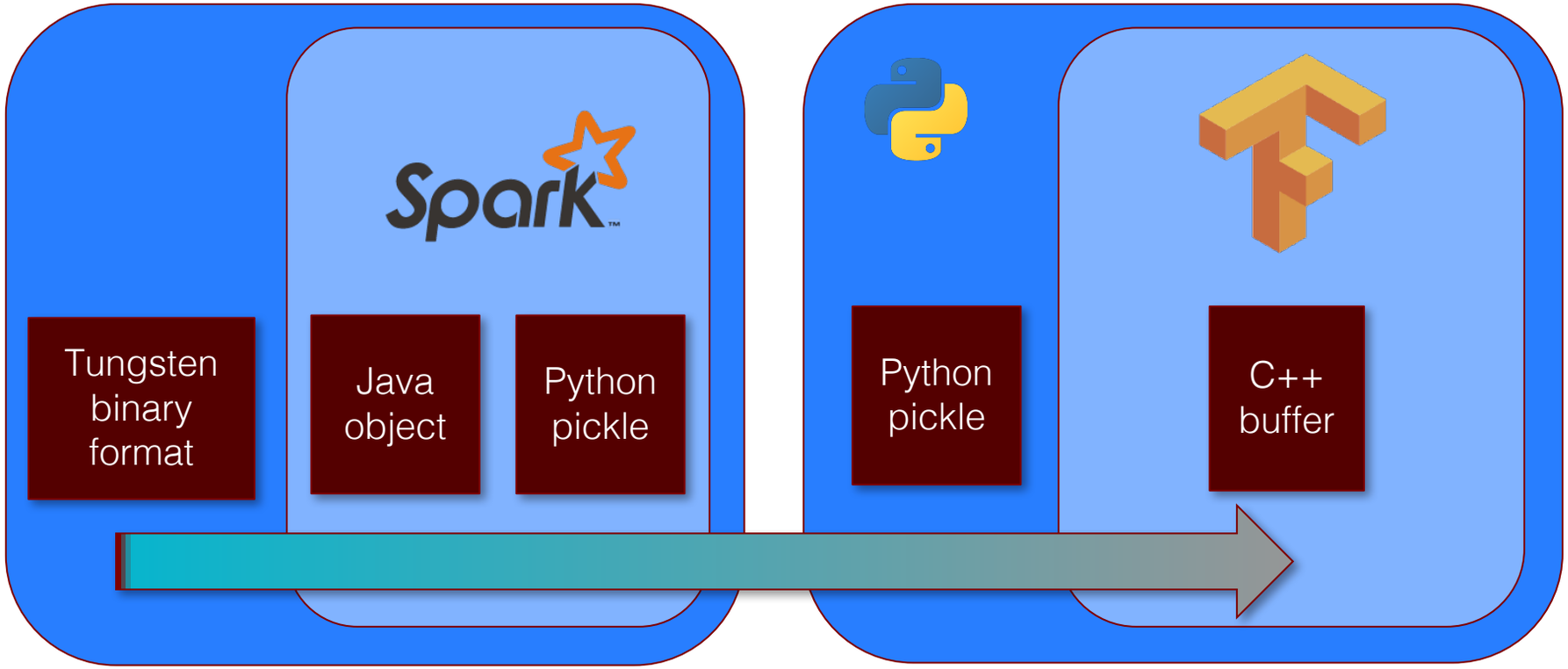
Scale out, fault tolerant

Python, Java, Scala, and R APIs

Standard libraries

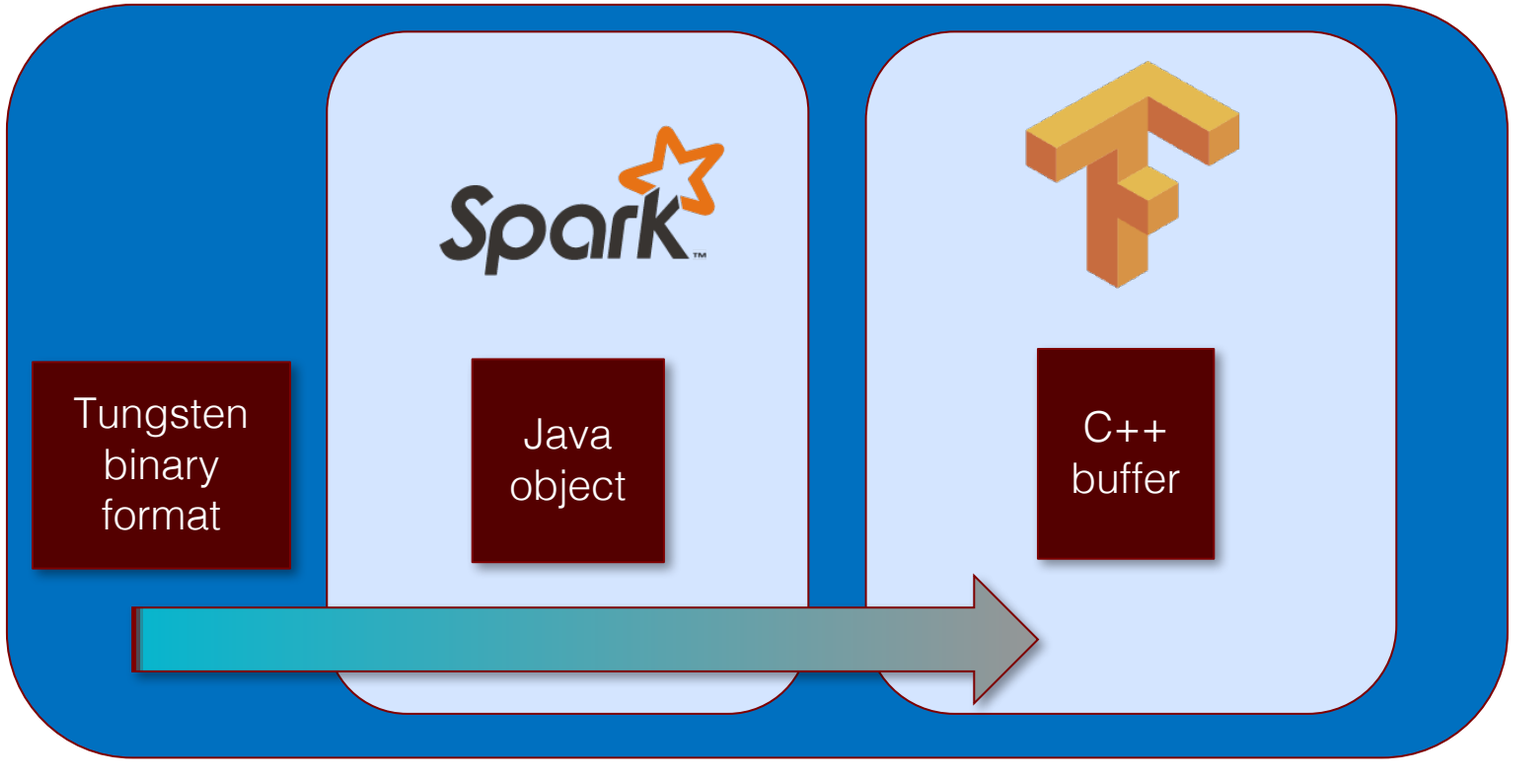
Unified engine across diverse workloads & environments

Spark to TensorFlow Communication Issues



Source: TensorFrames: Google TensorFlow with Apache Spark by Tim Hunter

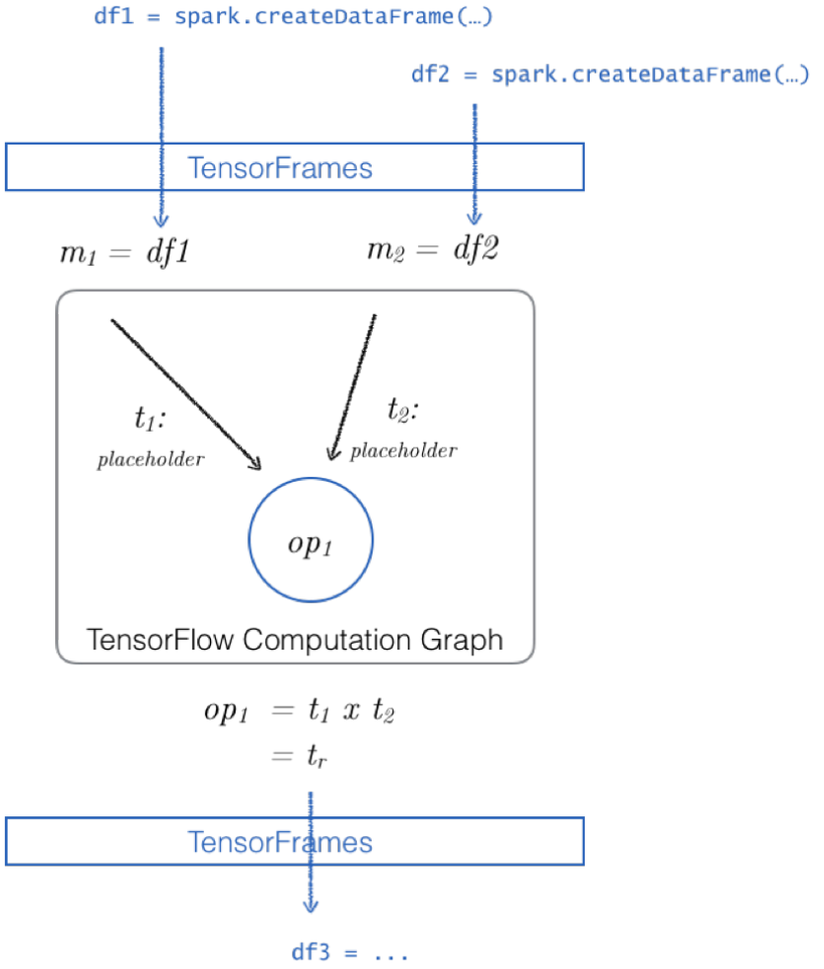
TensorFrames: Native Embedding of TensorFlow



Source: TensorFrames: Google TensorFlow with Apache Spark by Tim Hunter

An intermediary

- TensorFrames are built for speed*
- Spark DataFrames as input
- TensorFrames act as a bridge
- Can return a DataFrame



http://bit.ly/2iVy8vm

Why TensorFrames?

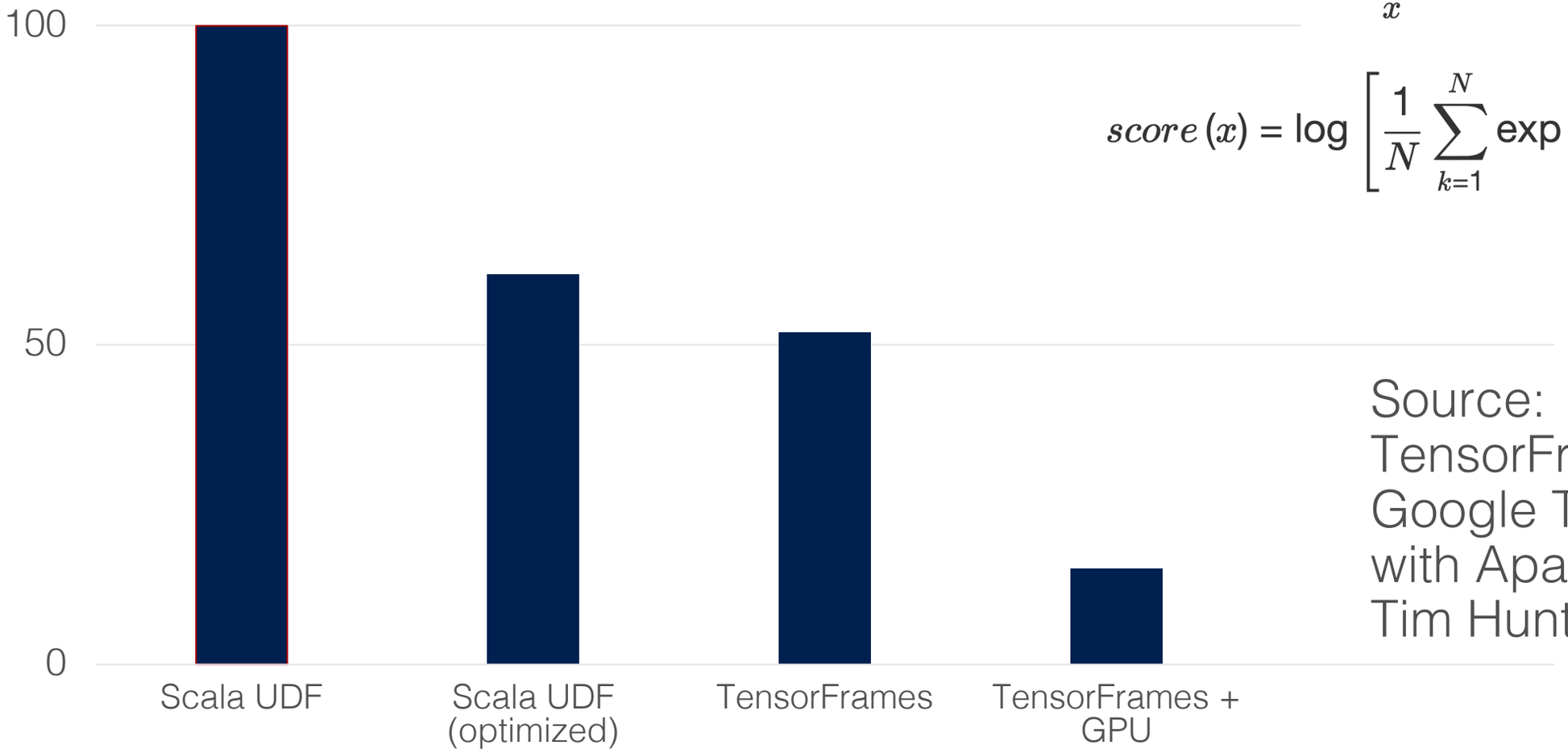
- Utilize TensorFlow with your data:
  - Allows data scientists to expand their analytics, streaming, graph, and ML capabilities to include Deep Learning via TensorFlow

- Parallel training to determine optimal hyperparameters:
  - Determine optimal learning rate: If rate is high, it will learn quickly but may not take into account of high variability
  - Number of neurons in each layer: too many results in noisy estimates, too few and the network will not learn well.

Comparing Kernel Density Scoring

$$\frac{1}{L}\sum_{x} score\,(x)$$

$$score\,(x) = \log\left[\frac{1}{N}\sum_{k=1}^{N}\exp\left(-\frac{(x - z_k)^2}{\sqrt{2\pi}b}\right)\right]$$



Source: TensorFrames: Google TensorFlow with Apache Spark by Tim Hunter

Getting ready

From CLI run

```
$SPARK_HOME/bin/pyspark \
--packages databricks:tensorframes:0.2.8-s_2.11
```

or (preferred) in Free Community Edition on Databricks (instructions shortly)

Start!

1. Import modules
2. Create some fake data
3. Create the graph
4. Block the variable
5. Add 3 to x and put in z
6. Return as df2

```
> # Import TensorFlow, TensorFrames, and Row
  import tensorflow as tf
  import tensorframes as tfs
  from pyspark.sql import Row

  # Create RDD of floats and convert into DataFrame `df`
  rdd = [Row(x=float(x)) for x in range(10)]
  df = sqlContext.createDataFrame(rdd)
```

```
> # Run TensorFlow program executes:
  #   The `op` performs the addition (i.e. `x` + `3`)
  #   Place the data back into a DataFrame
  with tf.Graph().as_default() as g:
      # The TensorFlow placeholder that corresponds to column 'x'.
      # The shape of the placeholder is automatically inferred from the DataFrame.
      x = tfs.block(df, "x")

      # The output that adds y to x
      z = tf.add(x, 3, name='z')

      # The resulting dataframe
      df2 = tfs.map_blocks(z, df)

  # Note that `z` is the tensor output from the `tf.add` operation
  print z
```

Element-wise sum and min

1. Create a DataFrame of lists
2. Analyze
3. Copy y
4. Create graph
5. Block the variables
6. Create the nodes
7. Get the results back

```
# Build a DataFrame of vectors
data = [Row(y=[float(y), float(-y)]) for y in range(10)]
df = sqlContext.createDataFrame(data)
```

```
+-----------+
|          y|
+-----------+
| [0.0, 0.0]|
|[1.0, -1.0]|
|[2.0, -2.0]|
|[3.0, -3.0]|
|[4.0, -4.0]|
|[5.0, -5.0]|
|[6.0, -6.0]|
|[7.0, -7.0]|
|[8.0, -8.0]|
|[9.0, -9.0]|
+-----------+
```

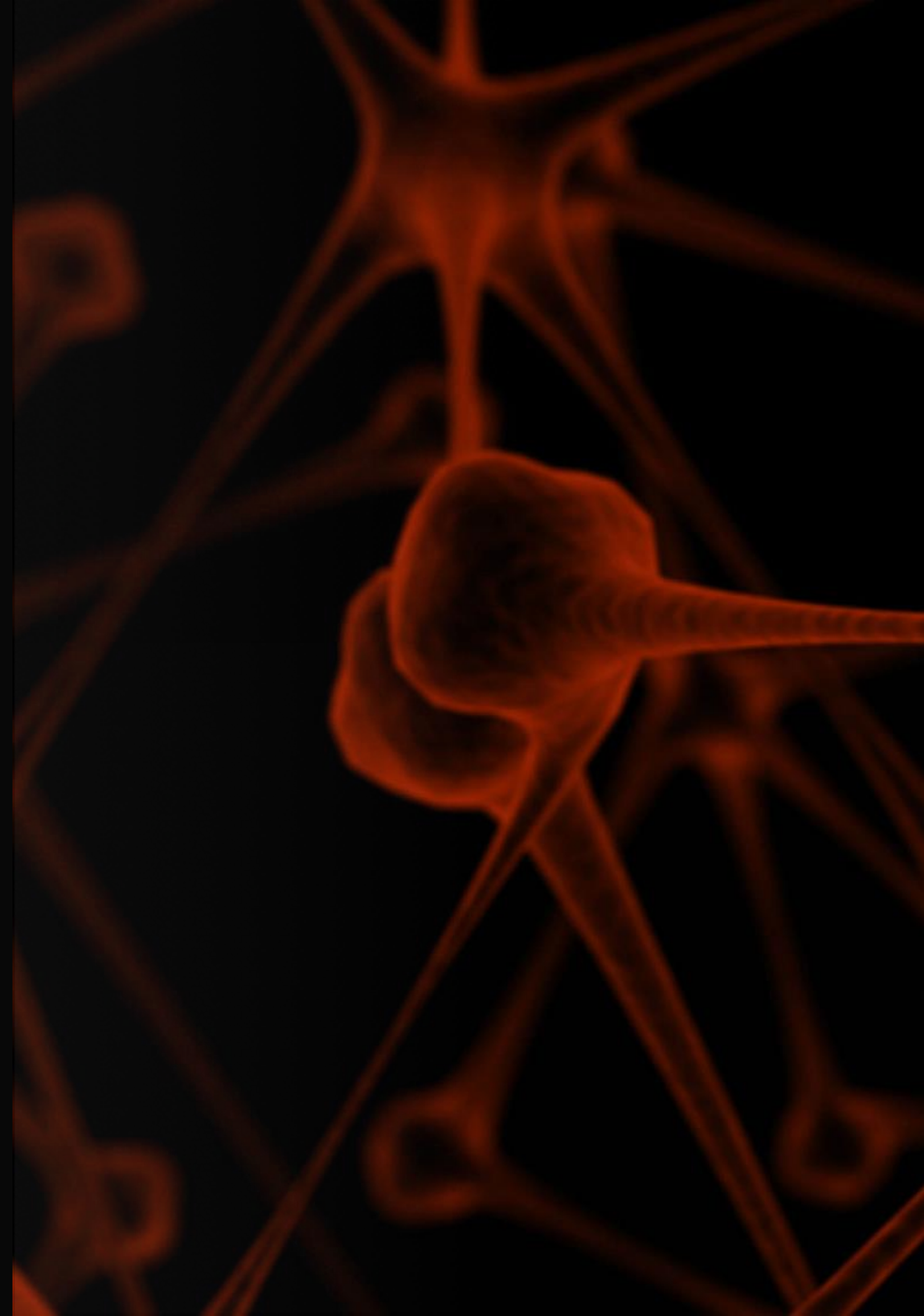
```
# Because the dataframe contains vectors, we need to analyze it first to find the
# dimensions of the vectors.
df2 = tfs.analyze(df)
```

```
# Note: First, let's make a copy of the 'y' column. This will be very cheap in Spark 2.0+
df3 = df2.select(df2.y, df2.y.alias("z"))

# Execute the Tensor Graph
with tf.Graph().as_default() as g:
    # The placeholders. Note the special name that end with '_input':
    y_input = tfs.block(df3, 'y', tf_name="y_input")
    z_input = tfs.block(df3, 'z', tf_name="z_input")

    # Perform elementwise sum and minimum
    y = tf.reduce_sum(y_input, [0], name='y')
    z = tf.reduce_min(z_input, [0], name='z')

    # The resulting dataframe
    (data_sum, data_min) = tfs.reduce_blocks([y, z], df3)
```

```
# The final results are numpy arrays:
print "Elementwise sum: %s and minimum: %s " % (data_sum, data_min)
```

```
Elementwise sum: [ 45. -45.] and minimum: [ 0. -9.]
```

Tutorial Setup



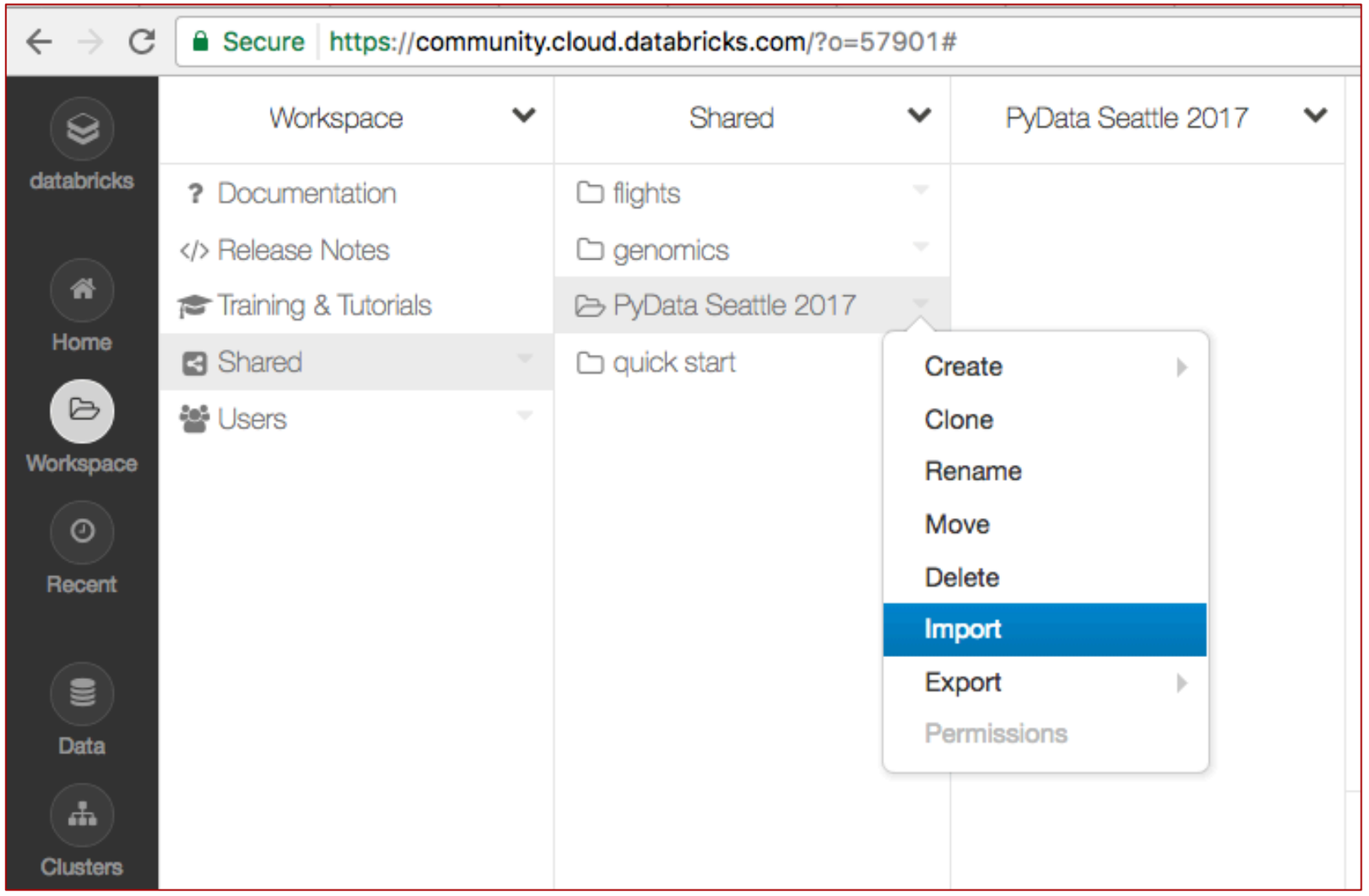
Tutorial Prerequisites

- Log into Databricks Community Edition (https://community.cloud.databricks.com)
- Import notebooks: https://aka.ms/pydatatfs
- Start cluster
- Install libraries
- Let’s go!

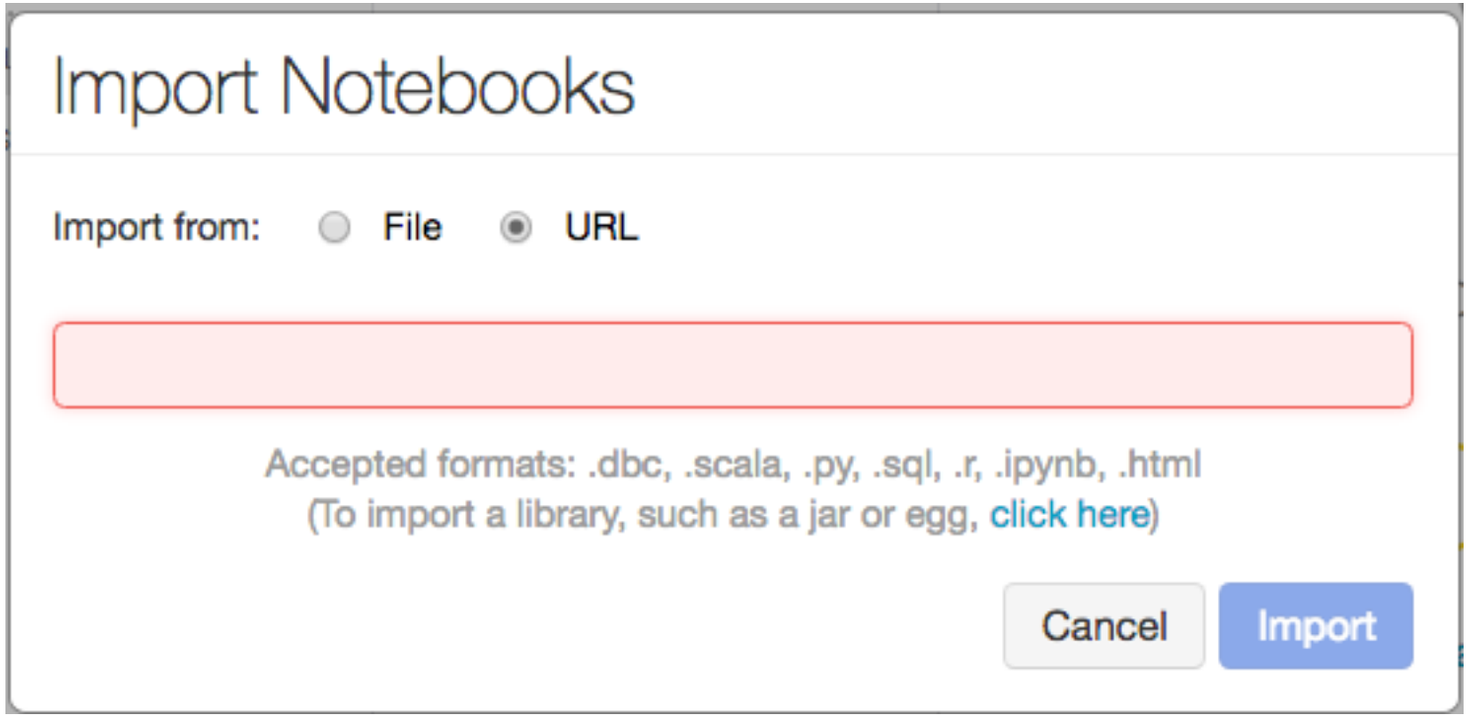
Import Notebooks

Steps to Import notebooks
Repeat

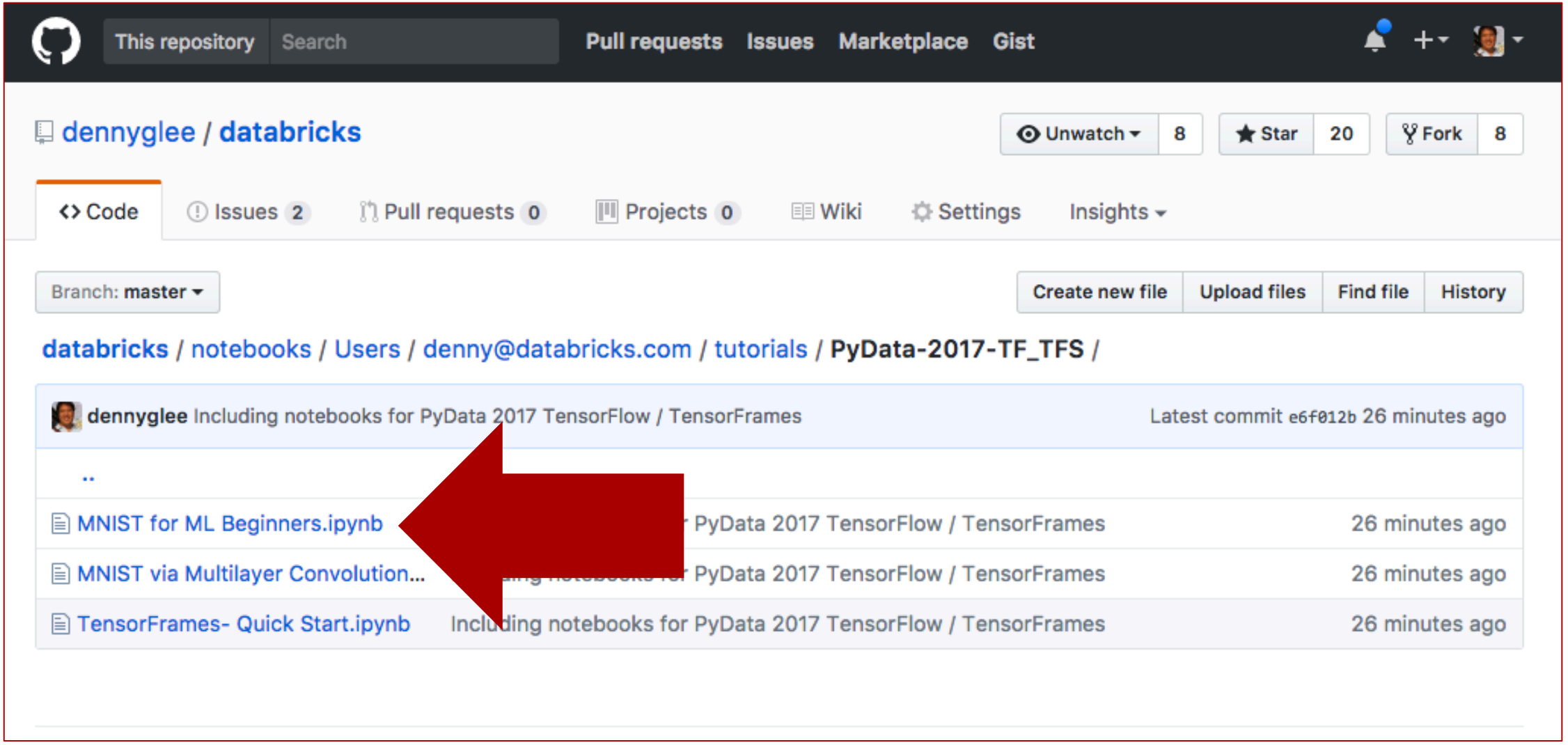
Import notebook



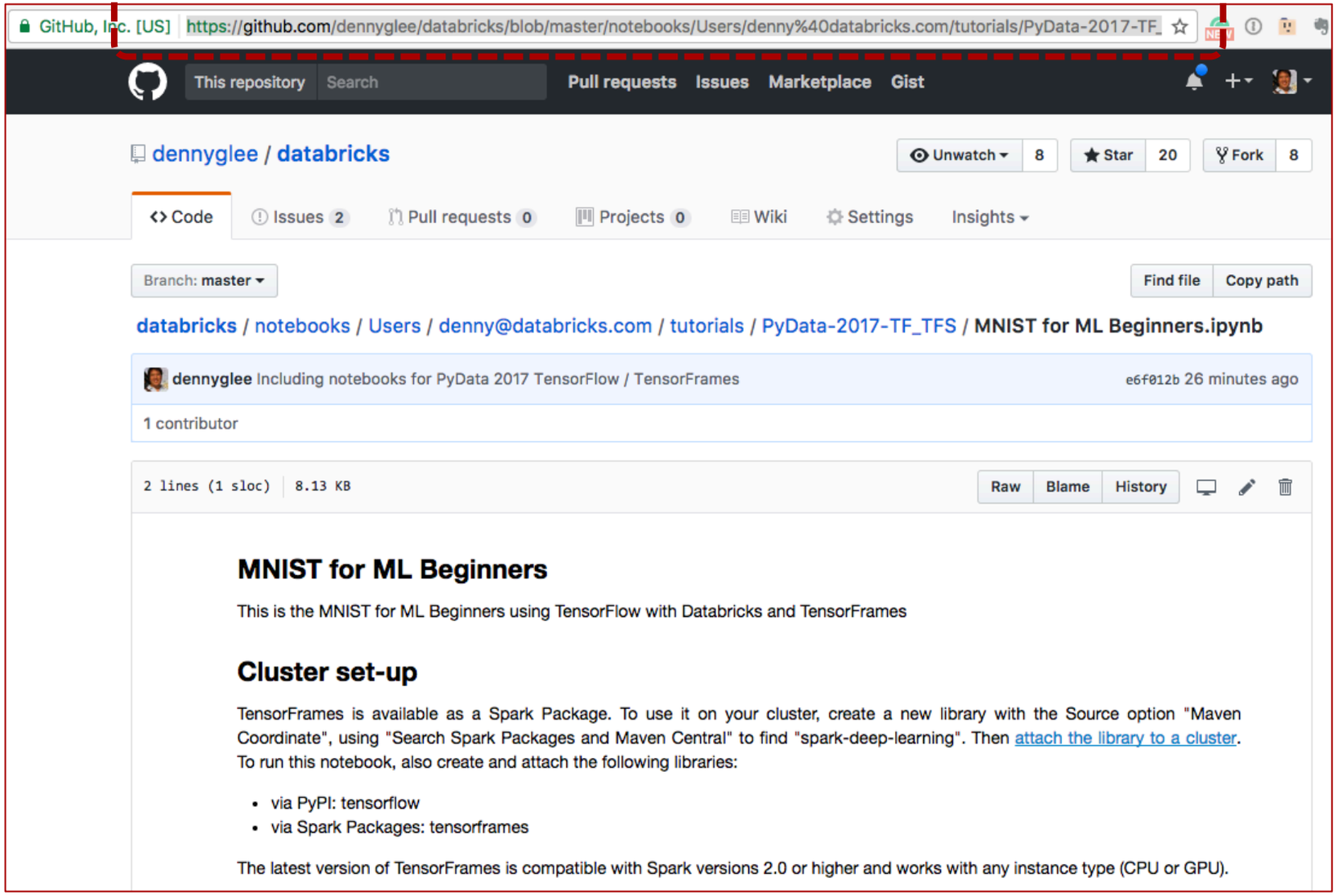
Import Notebook Dialog



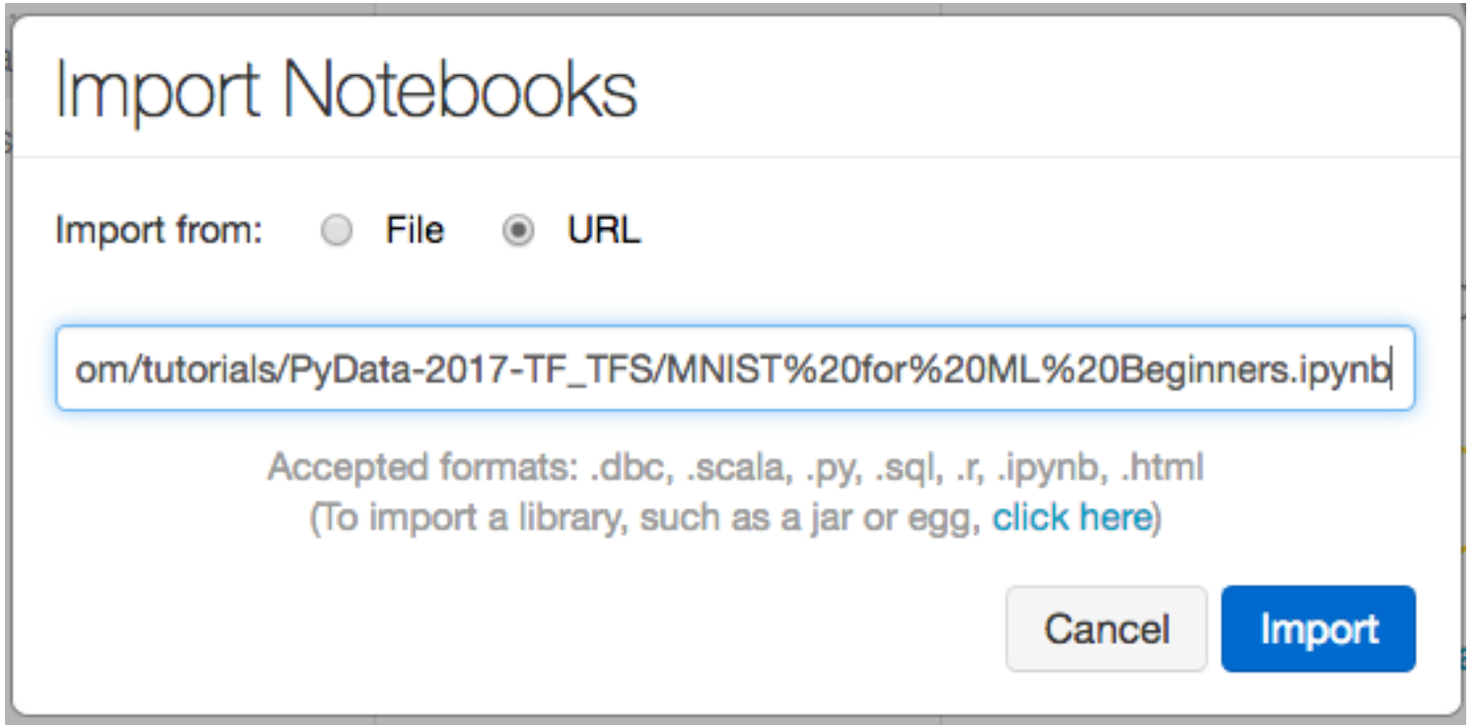
Goto https://aka.ms/pydata-sea-2017-tfs



Open a notebook > Get URL



Paste URL into Import Notebooks dialog



Do it for all three notebooks

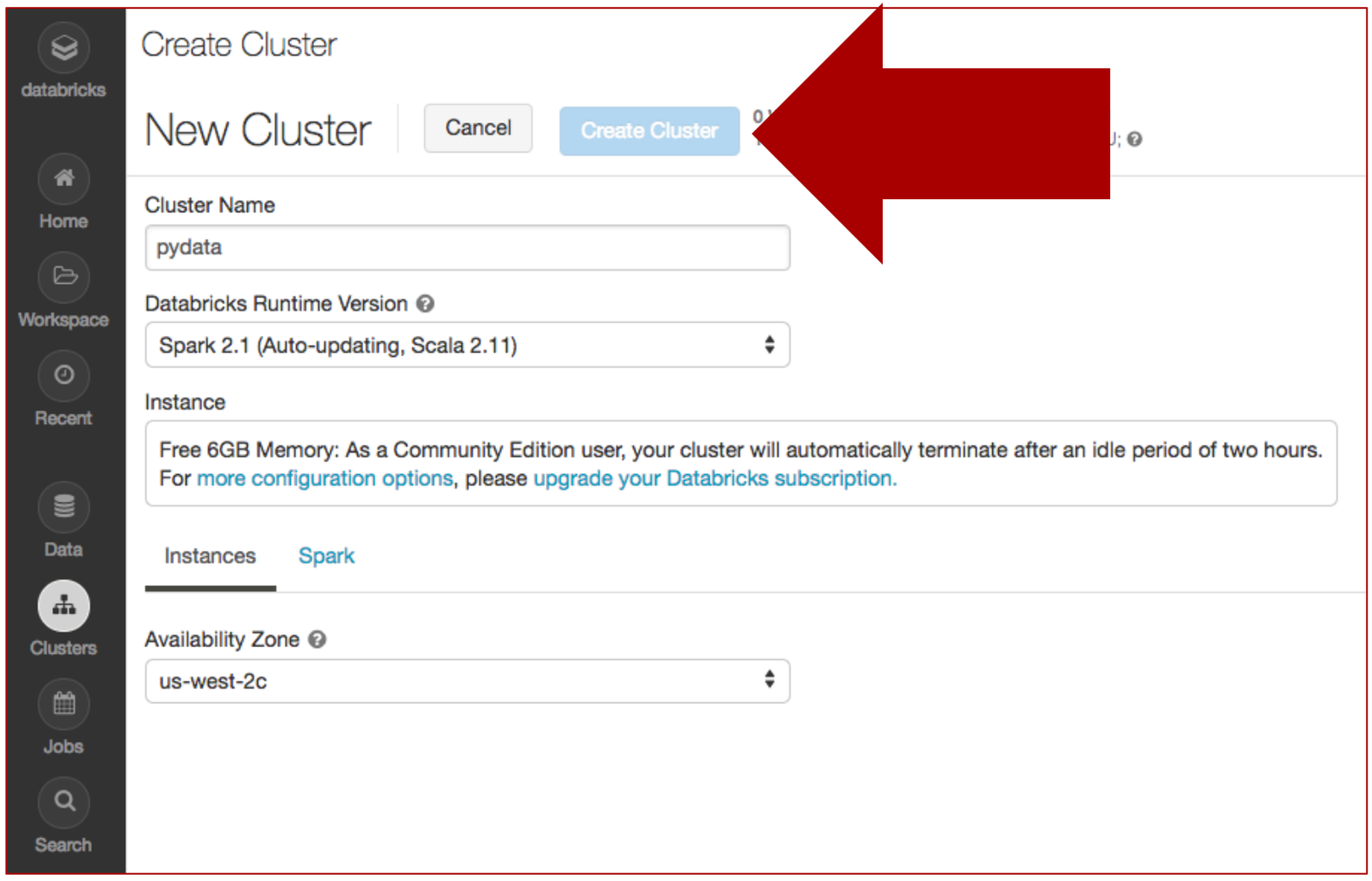
Start a cluster

Three steps to start a Spark cluster

Start a cluster

1. Clusters > Create Cluster

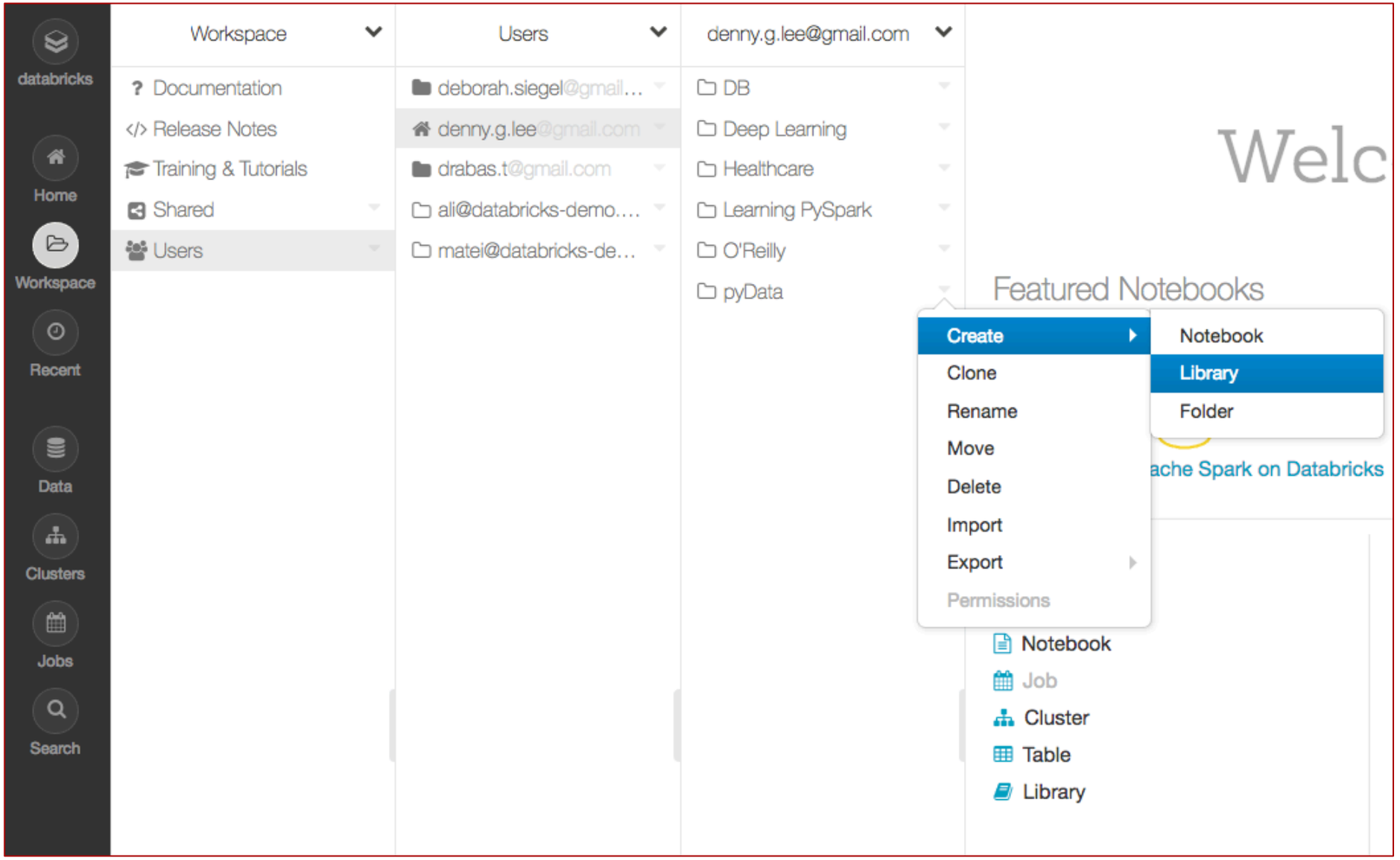
2. Fill in name and Run-time version: Spark 2.1 (Scala 2.11)



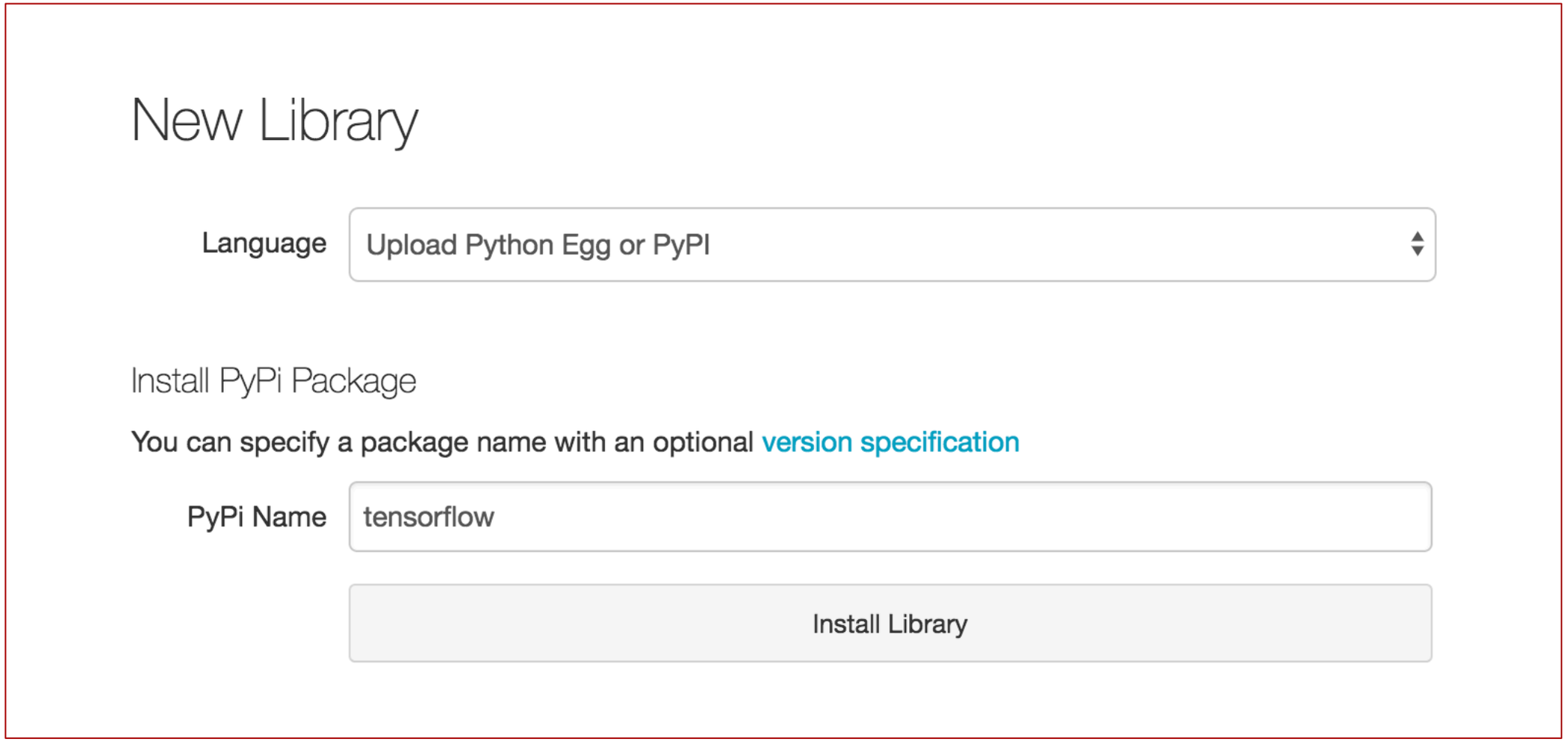
Install libraries

Installing TensorFlow via PyPi
Installing TensorFrames via Spark Packages

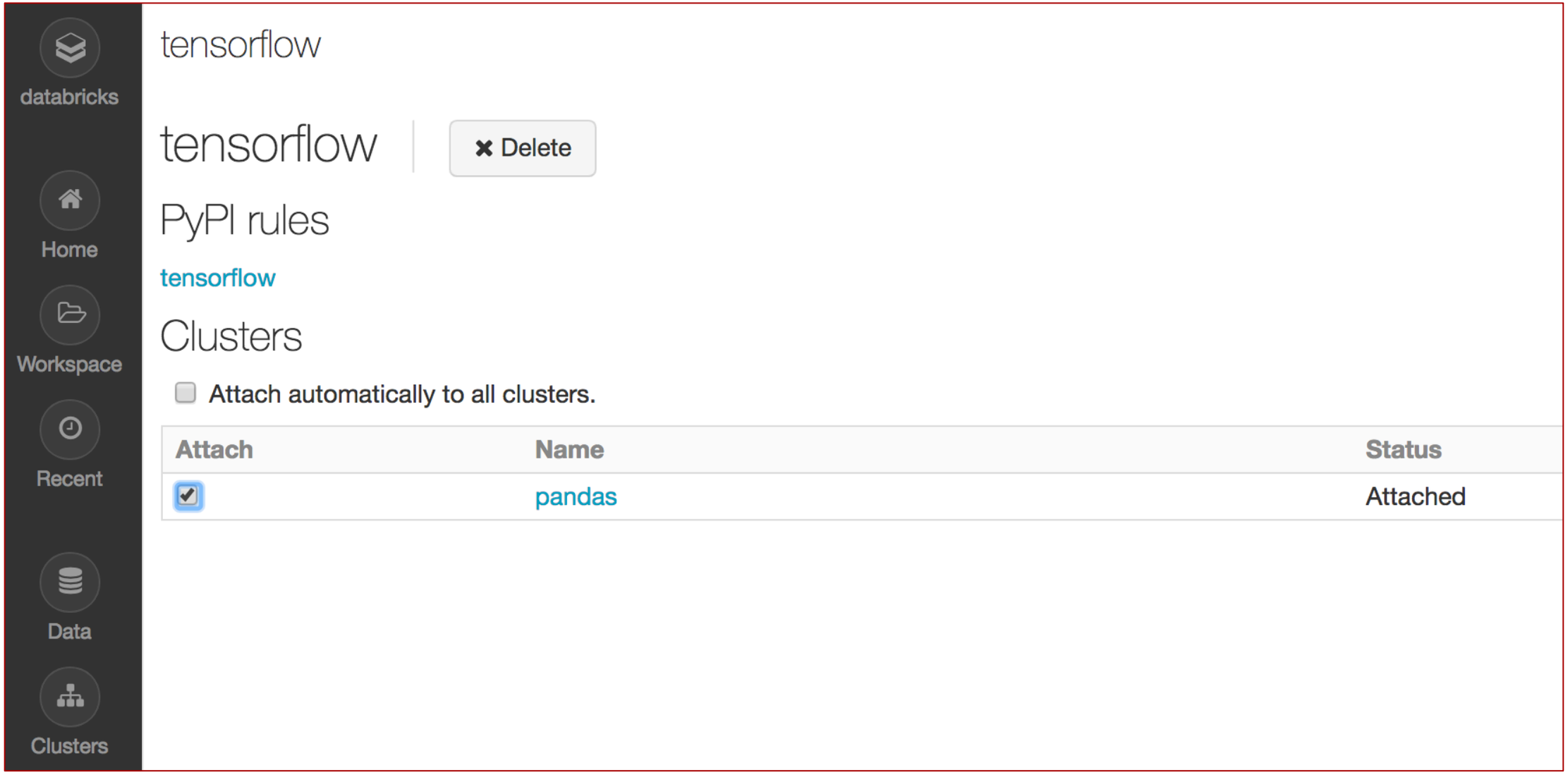
Create Library



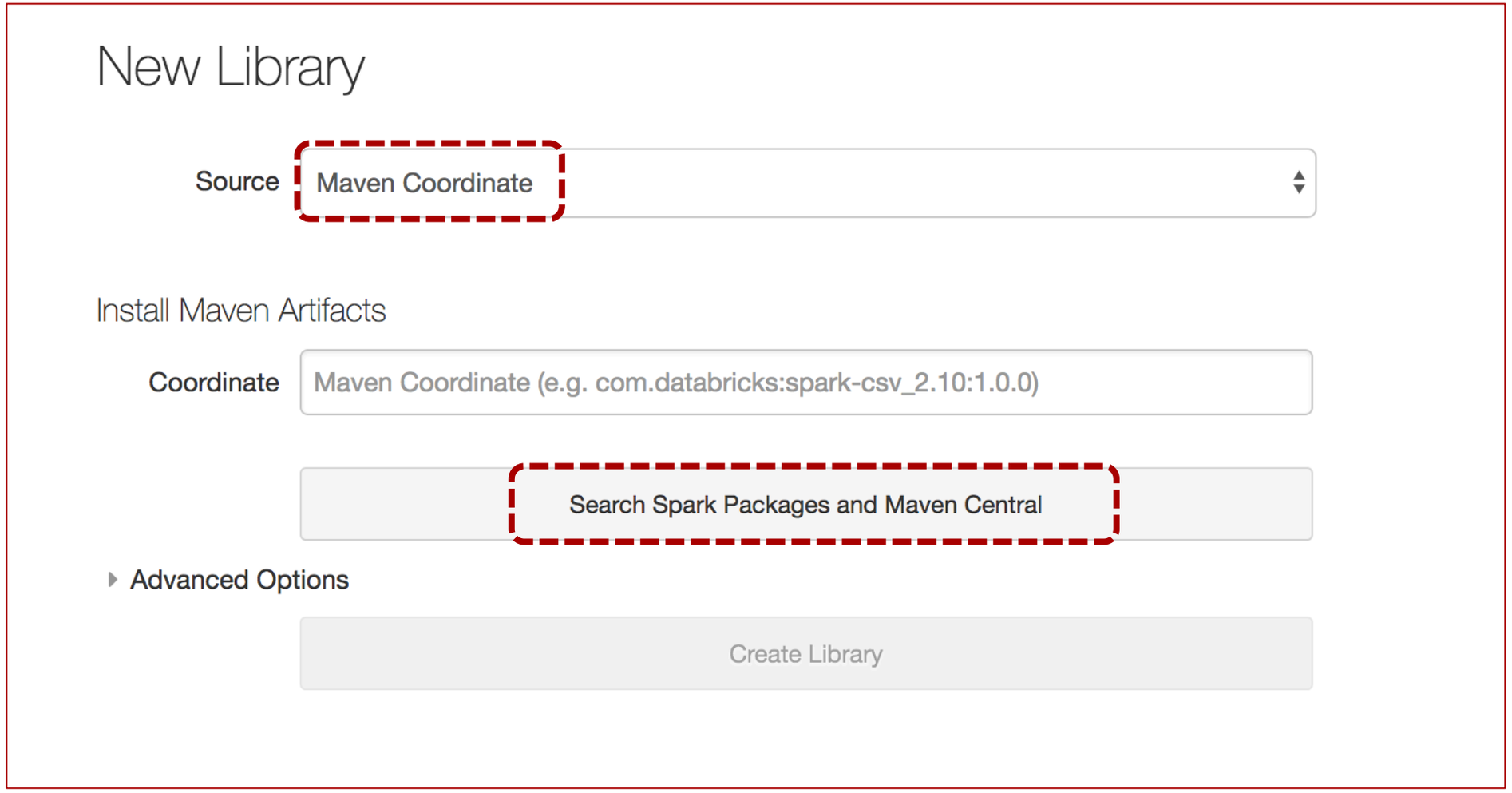
Install TensorFlow via PyPi



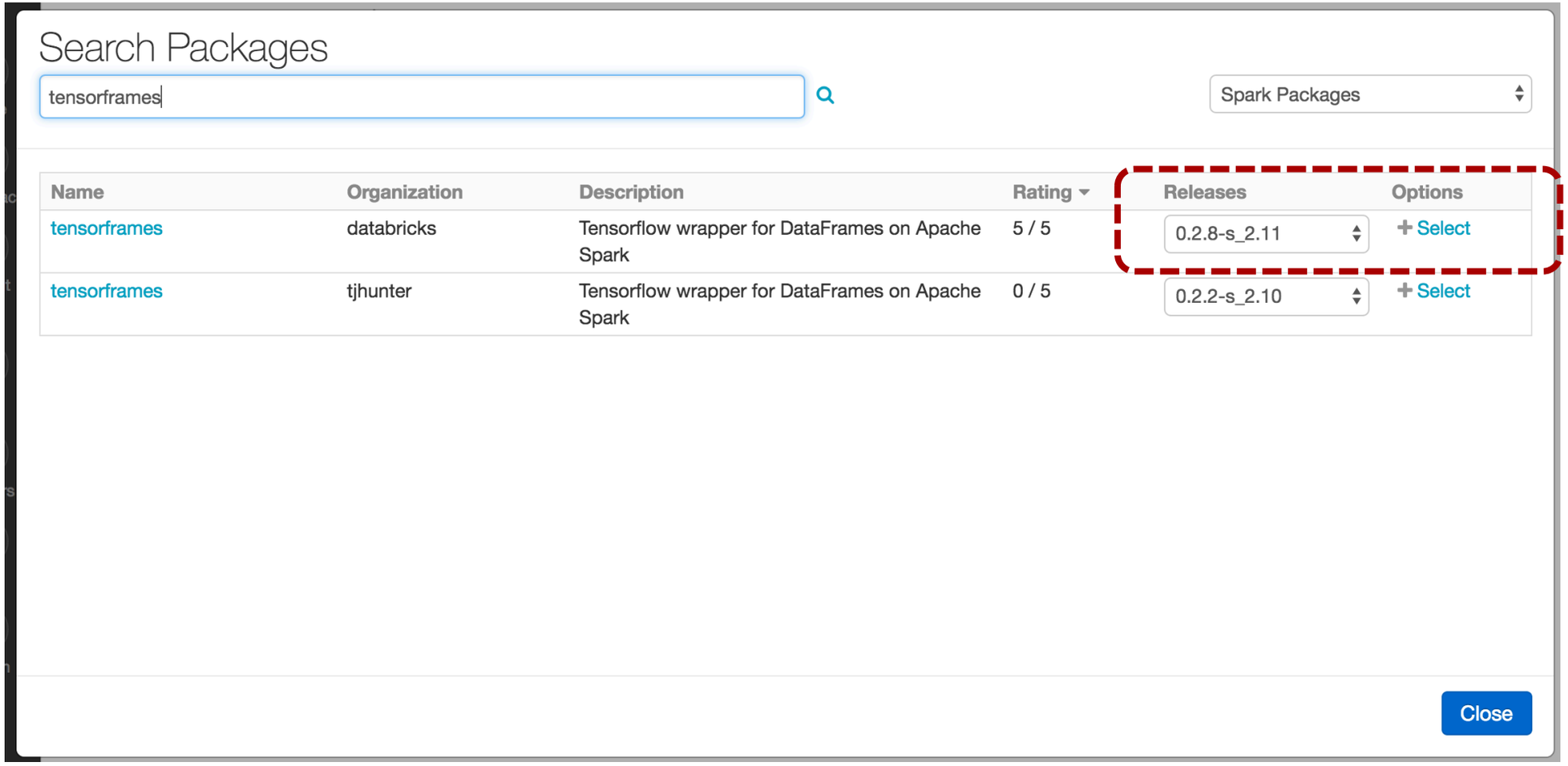
Attach TensorFlow to your cluster



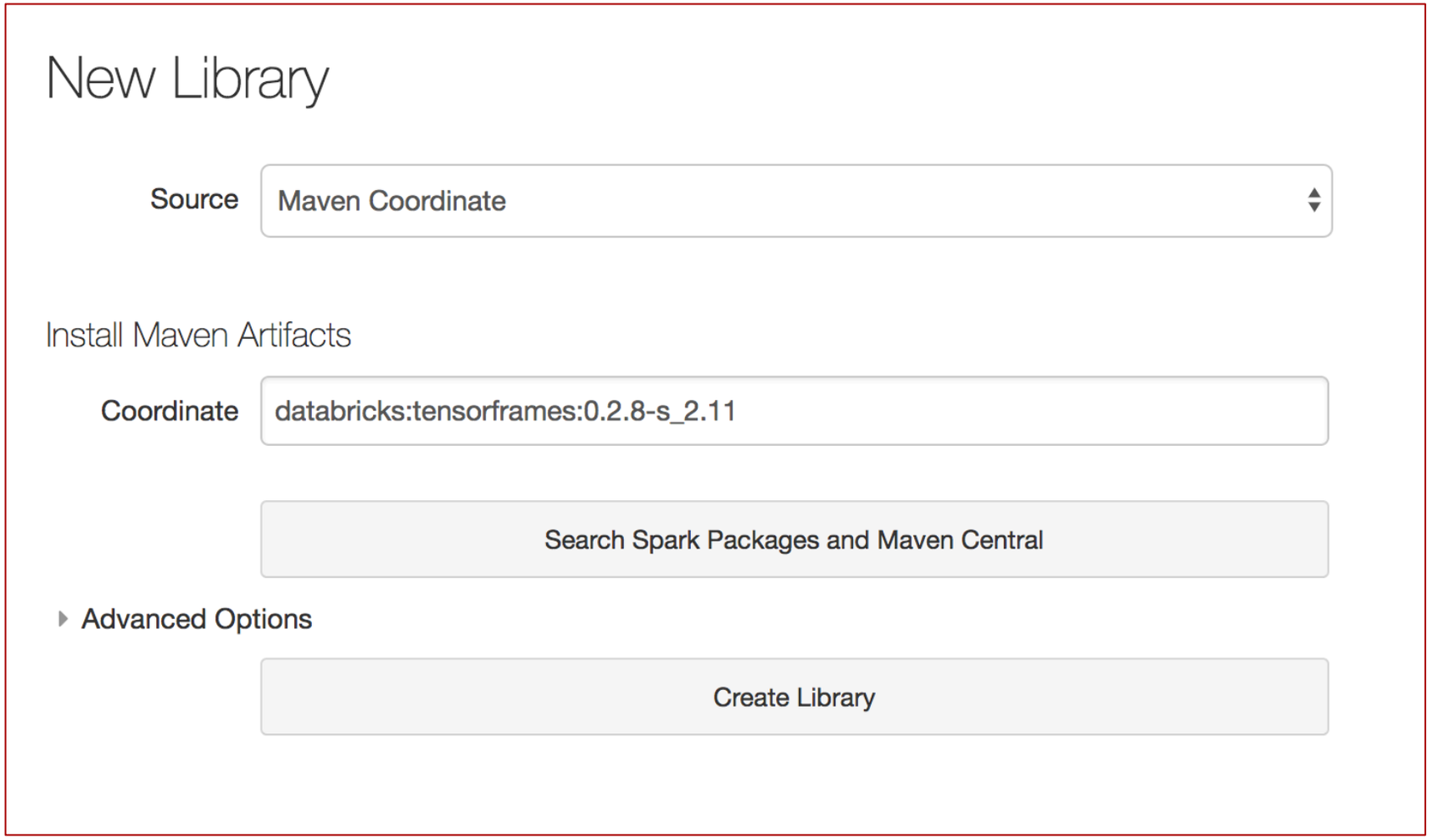
Install TensorFrames via Spark Packages



Find TensorFrames



Create TensorFrames Library … Almost!



Attach TensorFrames Library

tensorframes-0.2.8-s_2.11

tensorframes-0.2.8-s_2.11

✖ Delete

Artifacts

scalactic_2.11-3.0.0.jar
commons-proxy-1.0.jar
commons-lang3-3.4.jar
scala-logging-slf4j_2.11-2.1.2.jar
scala-logging-api_2.11-2.1.2.jar
libtensorflow-1.1.0-rc1.jar
slf4j-api-1.7.7.jar
tensorflow-1.1.0-rc1.jar
tensorframes-0.2.8-s_2.11.jar
libtensorflow_jni-1.1.0-rc1.jar

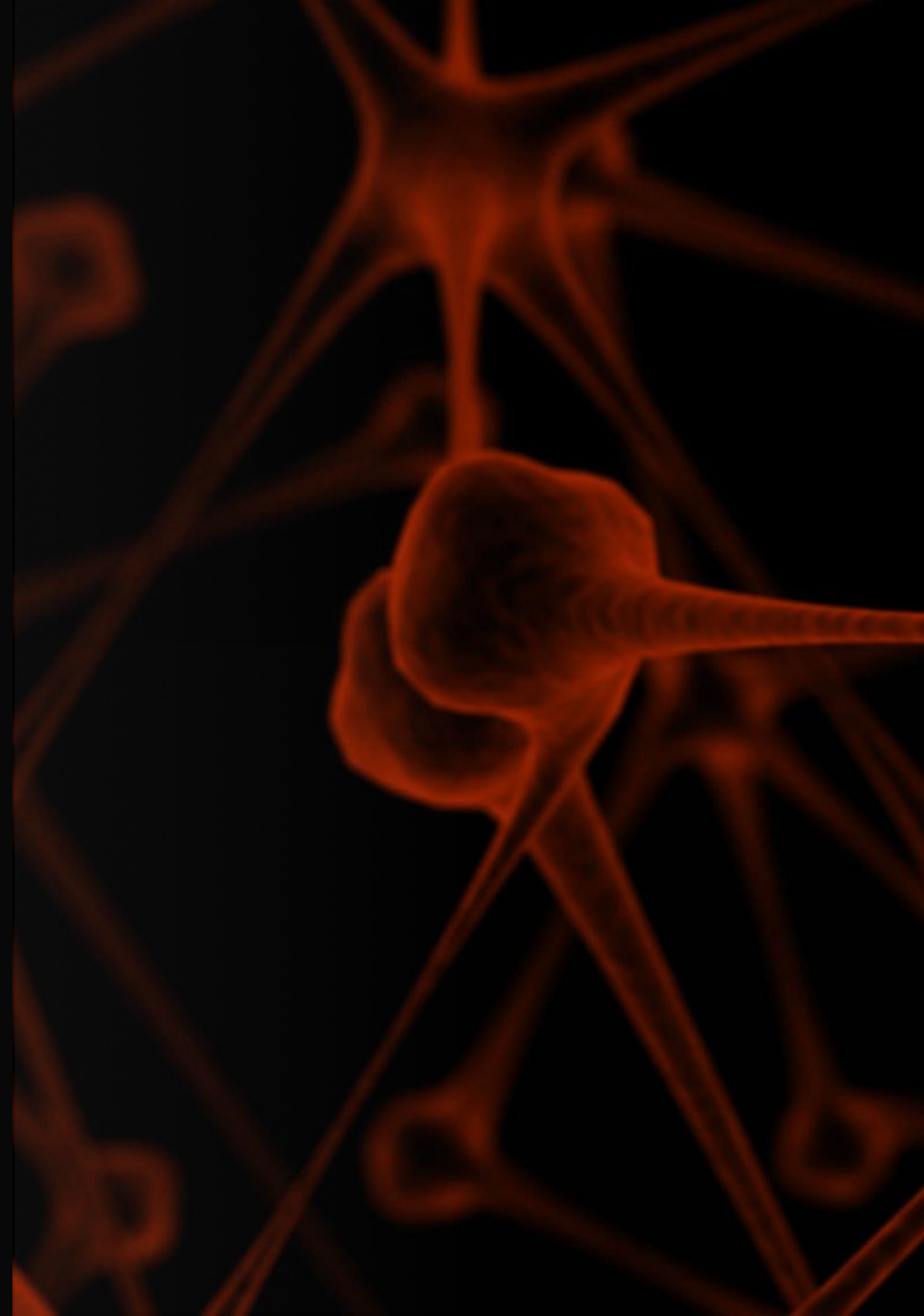
Clusters

☐ Attach automatically to all clusters.

Attach	Name	Status
☑	pandas	Attached

Let’s go!

Handwritten Digit Recognition using Softmax



Softmax Function (Linear Classifier)

Softmax function

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W, b)$$

Loss function

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

input image

stretch pixels into single column

matrix multiply + bias offset

0.01	-0.05	0.1	0.05
0.7	0.2	0.05	0.16
0.0	-0.45	-0.2	0.03

W

-15
22
-44
56

x_i

+

0.0
0.2
-0.3

b

cross-entropy loss (Softmax)

-2.85	exp →	0.058	normalize → (to sum to one)
0.86		2.36	
0.28		1.32	

0.016	cat
0.631	dog
0.353	car

- log(0.353) = **0.452**

$f(x_i; W, b)$

unnormalized log probabilities

unnormalized probabilities

probabilities

Source: http://bit.ly/2tNRZpg | Demo: http://stanford.io/2tJCtd7

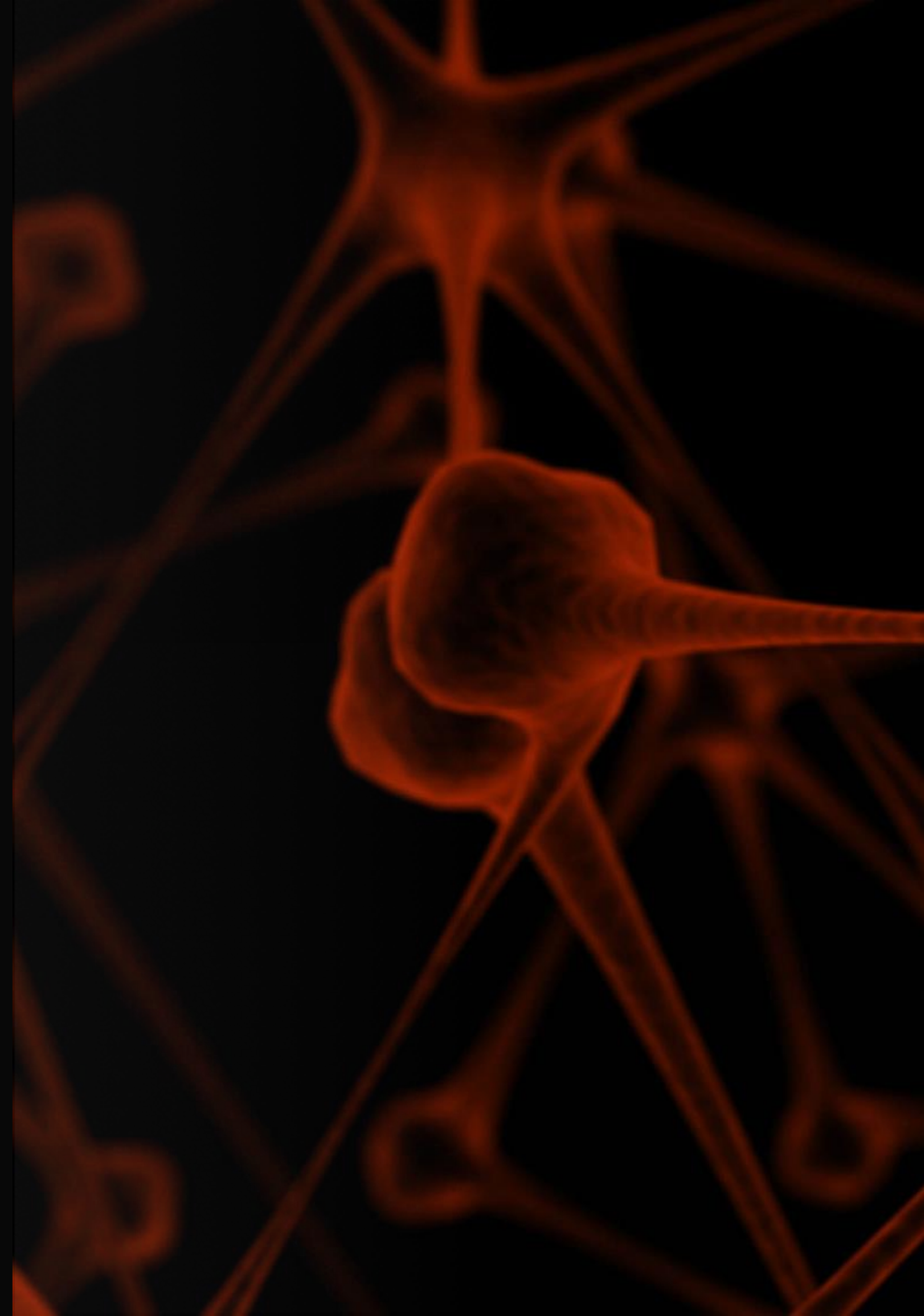
CNN accuracy: ~92

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
#print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

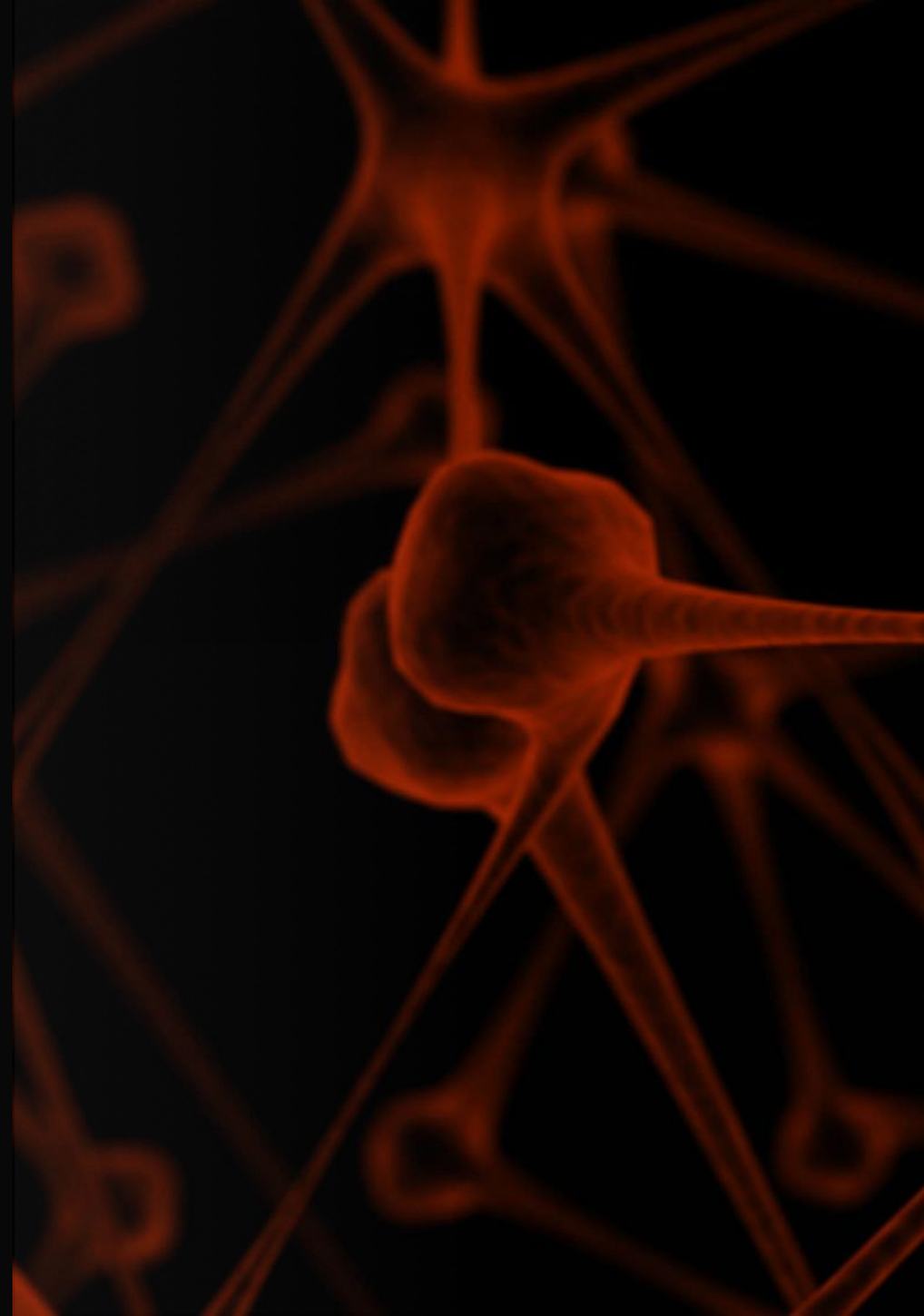
```
0.9071
```

ConvNetJS MNIST Demo

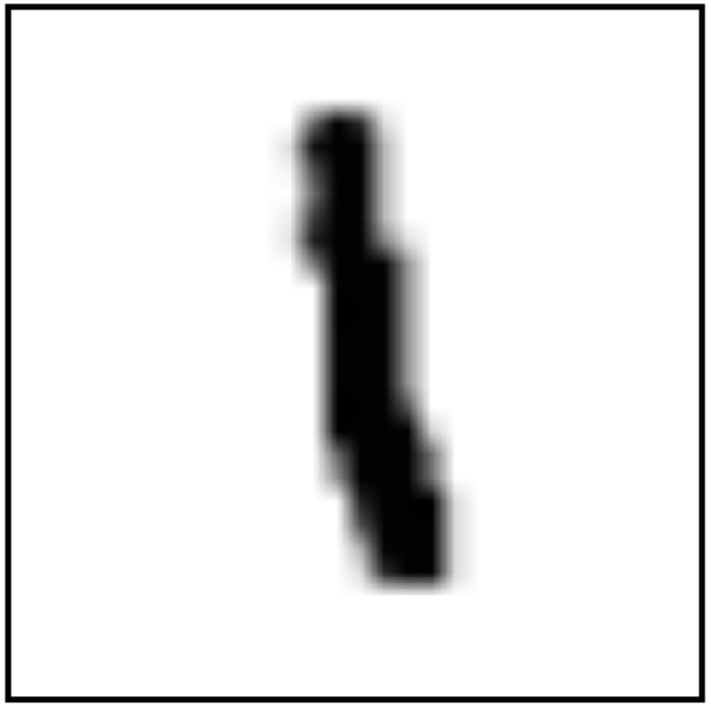
Andrej Karpathy's MNIST Demo:
https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html



Handwritten Digit Recognition using CNNs



Handwritten Digit Recognition



≈

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	.6	.8	0	0	0	0	0	0
0	0	0	0	0	0	.7	1	0	0	0	0	0	0
0	0	0	0	0	0	.7	1	0	0	0	0	0	0
0	0	0	0	0	0	.5	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	1	.7	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	.9	1	.1	0	0	0	0
0	0	0	0	0	0	0	.3	1	.1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Source: http://bit.ly/2sbFW0R

MNIST Data

- Set of handwritten digits based on the National Institute of Science and Technology – i.e. a modified dataset or MNIST dataset – via LeCun et. al

- Original source is NIST Special Database 19 Handprinted Forms and Characters Database comprised of:
  - Images: Each digit is a scanned 28px x 28px flattened into an array of size 784
  - Labels: One-Hot vector representing the actual number associated with the image

Great References

Andrej Karparthy's ConvNetJS MNIST Demo

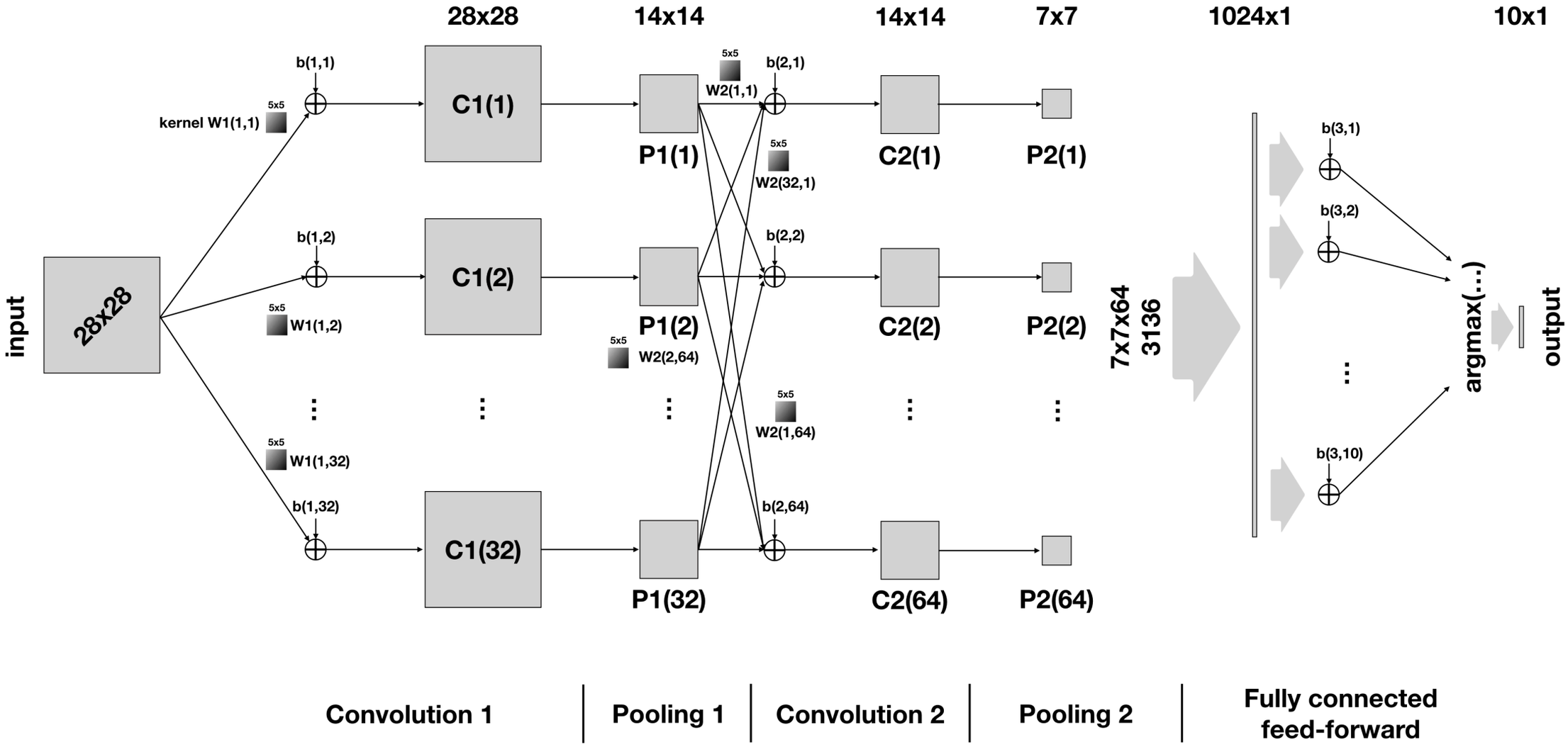
CS231n: Convolutional Neural Networks for Visual Recognition

Syllabus and Slides | Course Notes | YouTube

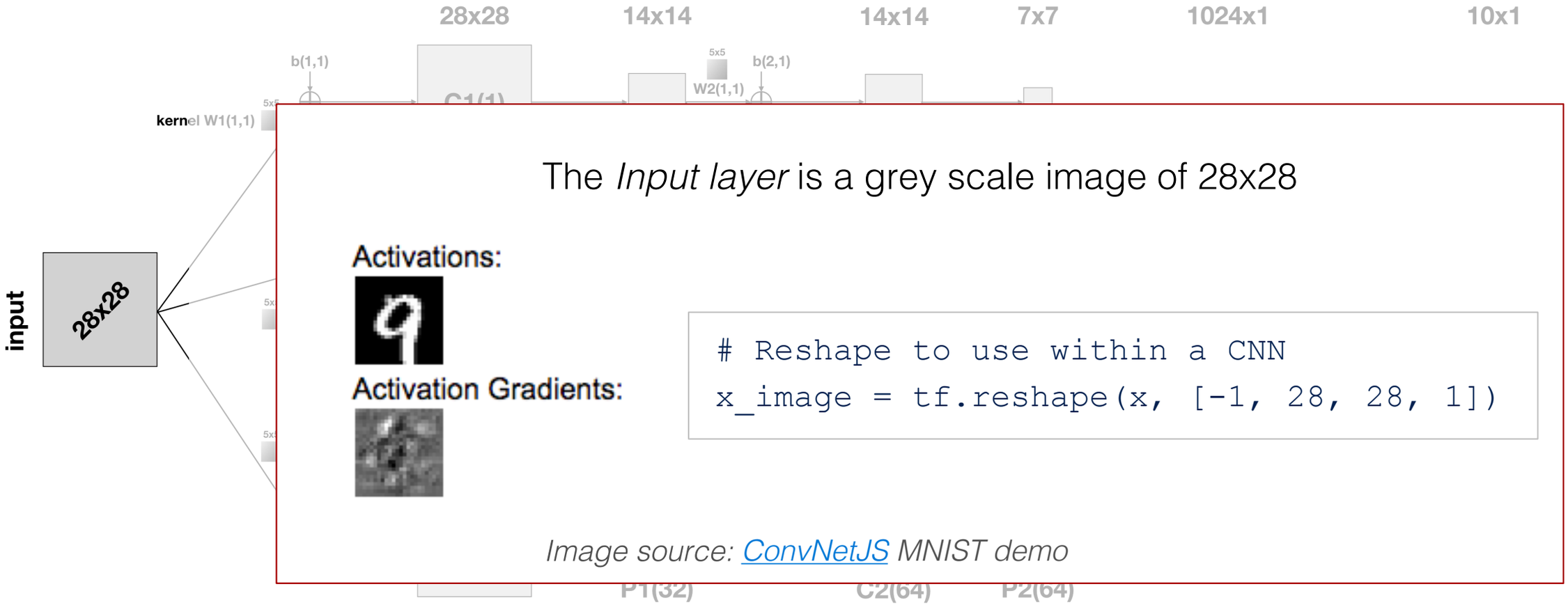
- With particular focus on CS231n: Lecture 7: Convolution Neural Networks

Upcoming webinar: Build, Scale, and Deploy Deep Learning Pipelines with Ease

CNN Structure

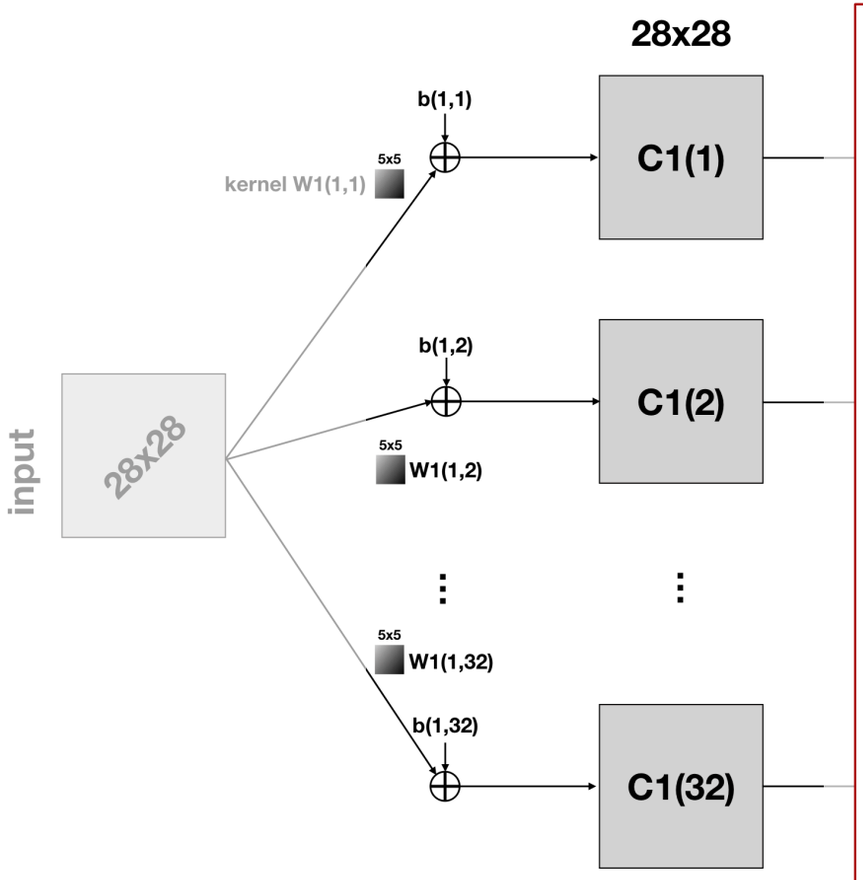


CNN Structure



Convolution 1 | **Pooling 1** | **Convolution 2** | **Pooling 2** | **Fully connected feed-forward**

CNN Structure: Convolution Layer 1

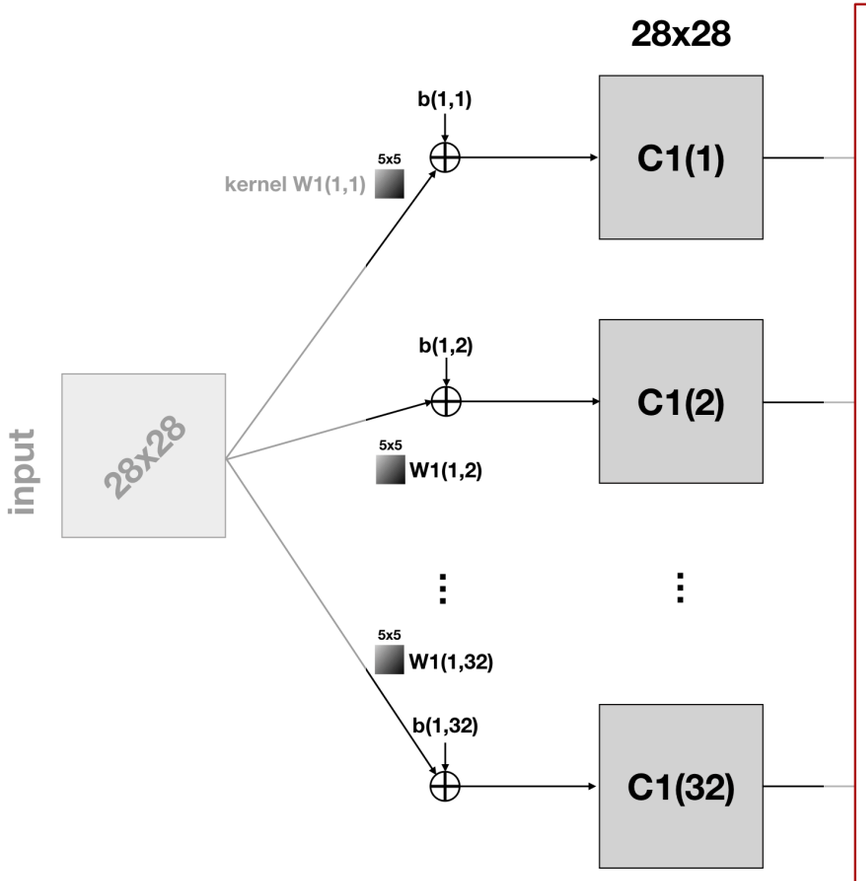


Convolution layer 1 maps one grayscale image to 32 feature maps using 5x5 kernels using ReLU (rectifier) activation function

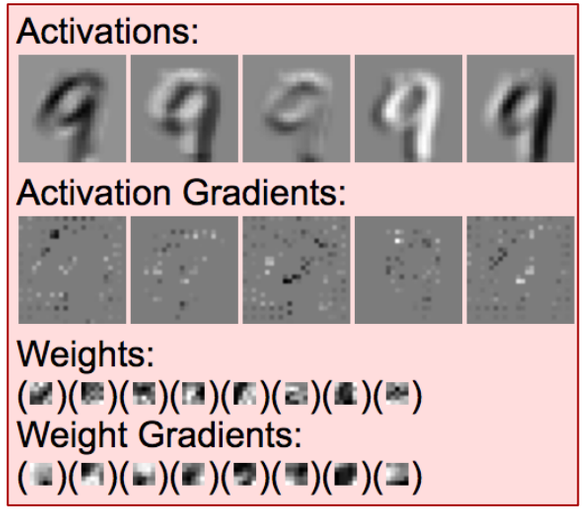
```
# First convolutional layer:
# maps one grayscale image to 32 feature maps.
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) +
b_conv1)
```

Convolution 1 | **Pooling 1** | **Convolution 2** | **Pooling 2** | **Fully connected feed-forward**

CNN Structure: Convolution Layer 1



Convolution layer 1 maps one grayscale image to 32 feature maps using 5x5 kernels using ReLU (rectifier) activation function



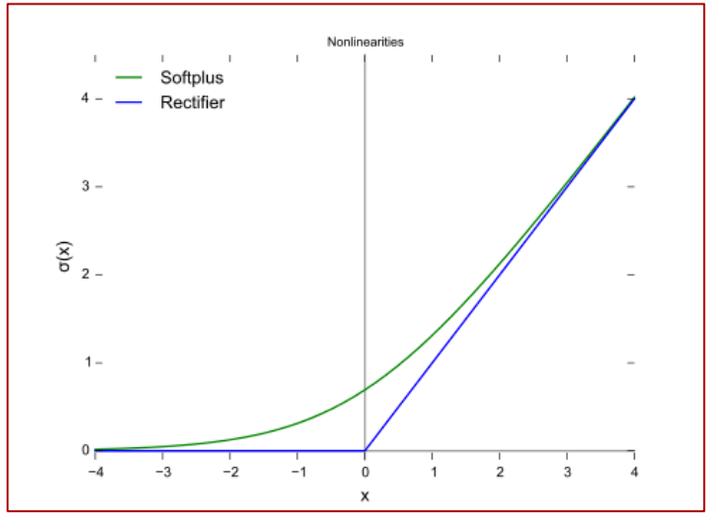
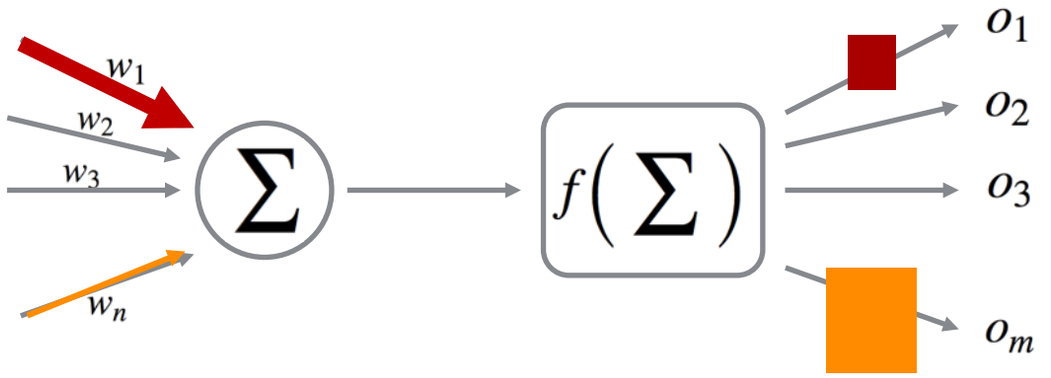


Image source: ConvNetJS MNIST demo

Convolution 1 | **Pooling 1** | **Convolution 2** | **Pooling 2** | **Fully connected feed-forward**

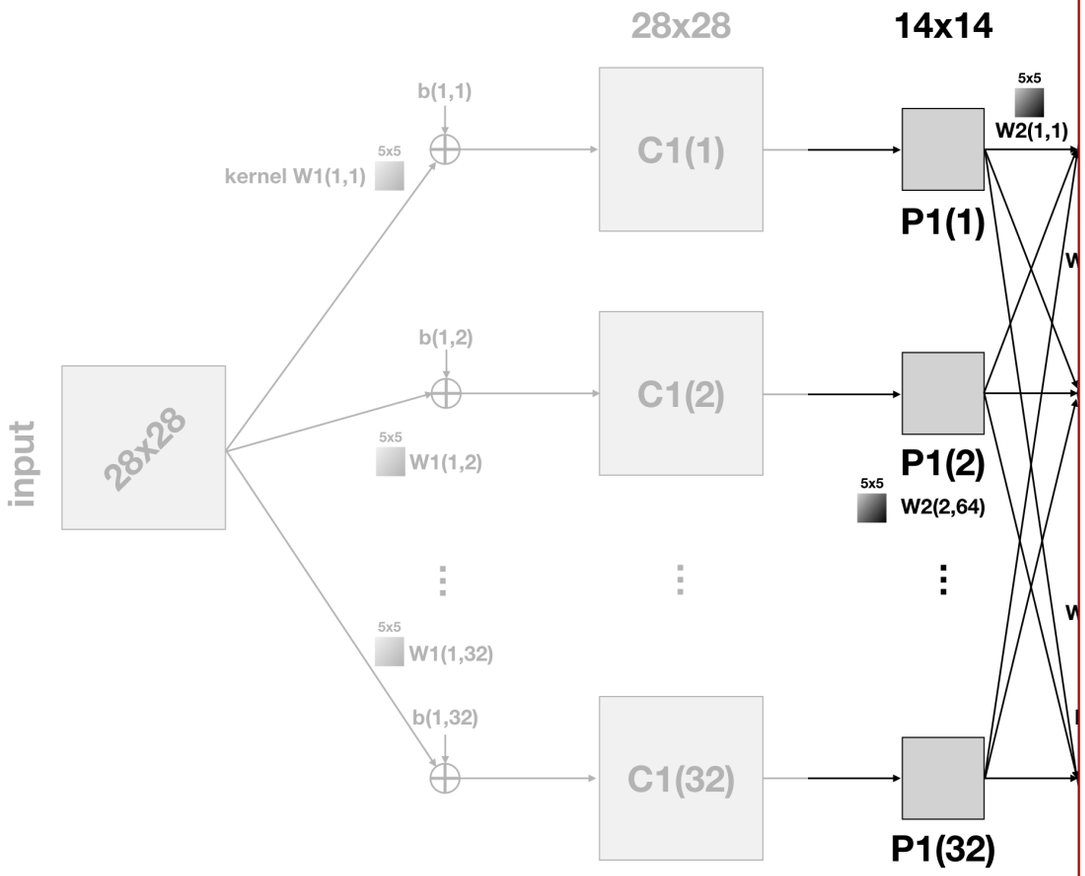
Why normalize? (continued)

- Normalization changes the range of pixel intensity
- Also known as contrast stretching or histogram stretching
- Bring image values to the same range



"If we don't scale our input training vectors, the ranges of distribution of those feature vectors would likely be different for each vector – i.e. learning rate would cause corrections, e.g. over compensating one and under compensating another"
[Source: http://bit.ly/2rsc2b1]

CNN Structure: Pooling 1



Pooling layer 1 down samples image by 2x so you have a 14x14 matrix

```
# Pooling layer - downsamples by 2X.
h_pool1 = max_pool_2x2(h_conv1)
```

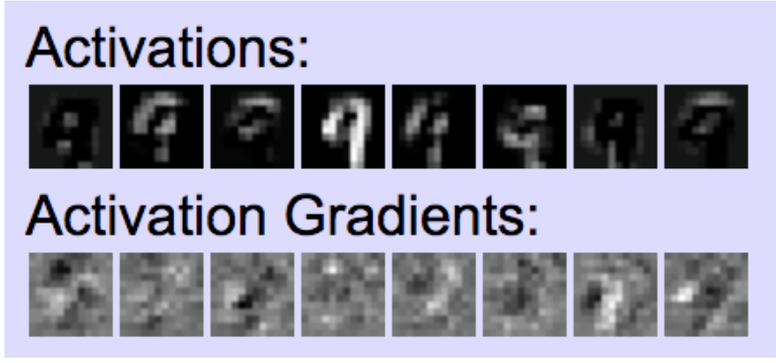
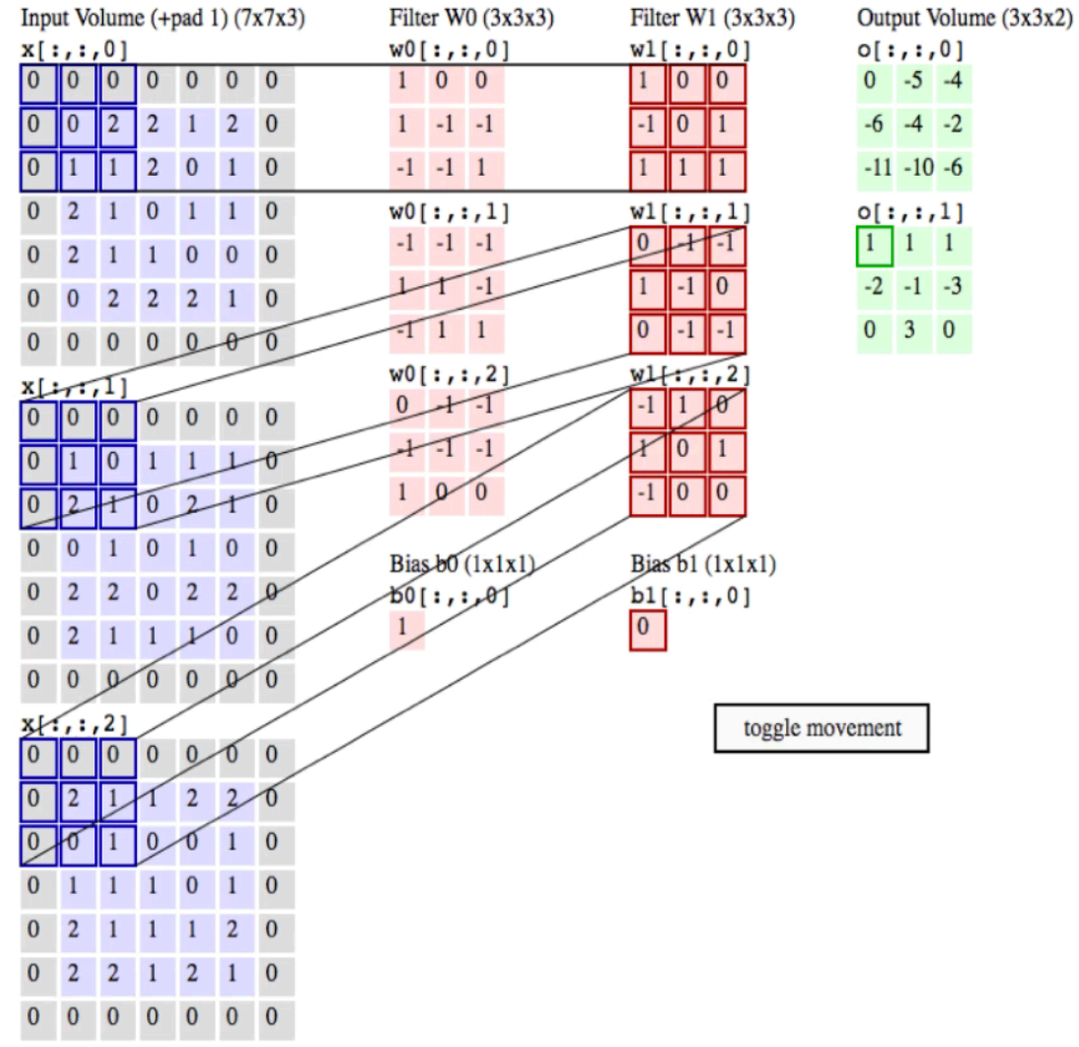


Image source: ConvNetJS MNIST demo

Convolution 1 | **Pooling 1** | **Convolution 2** | **Pooling 2** | **Fully connected feed-forward**

What's happening here? (example)



Credit for this excellent animation goes to Andrej Karpathy.

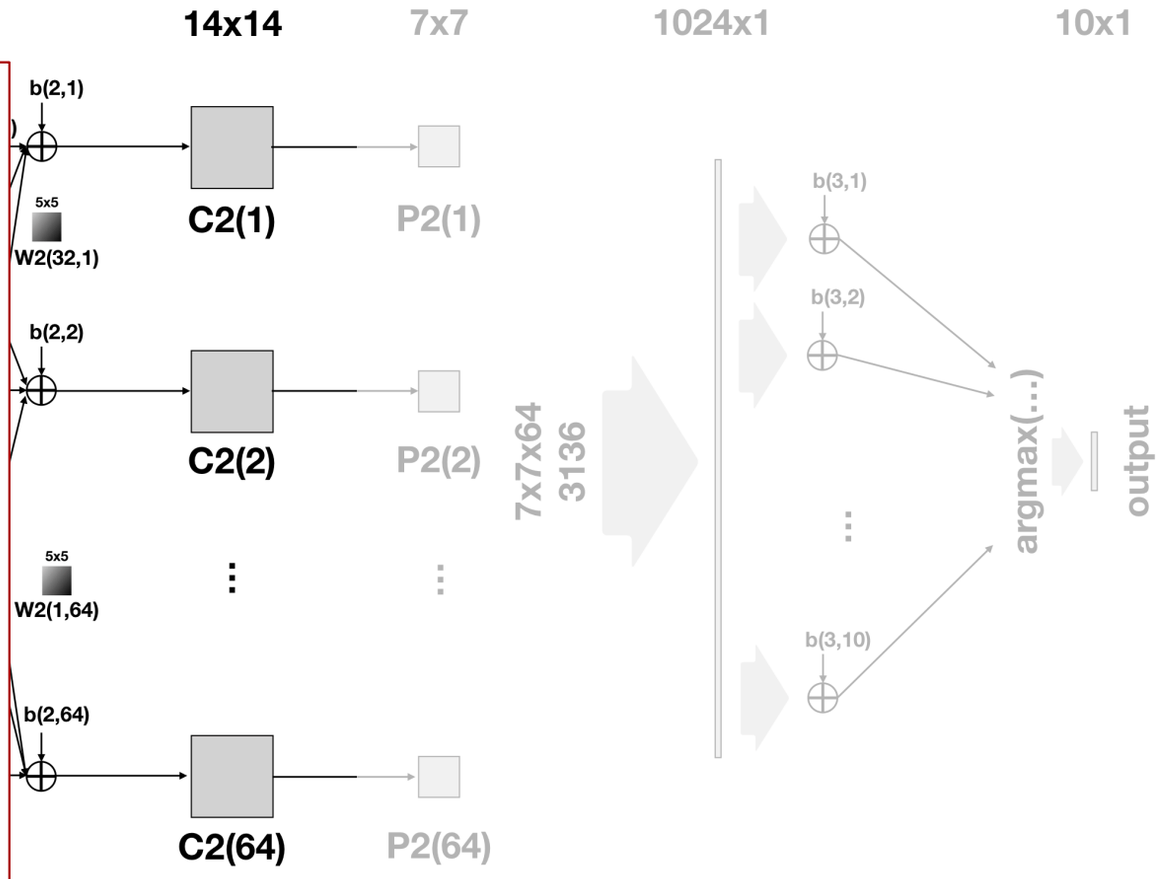
- CNN takes a square of patches of pixels (vs. individual pixels)
- Filters them into a smaller square matrix AND equal to size of the patch (or kernel)
- Job of filter is to find patterns in the pixels

Source: https://deeplearning4j.org/convolutionalnets.html

CNN Structure: Convolution 2

Convolution layer 2 maps 32 feature maps to 64 using ReLU (rectifier) activation function

```
# Second convolutional layer
# maps 32 feature maps to 64.
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1,
W_conv2) + b_conv2)
```



Convolution 1 | Pooling 1 | Convolution 2 | Pooling 2 | Fully connected feed-forward

CNN Structure: Convolution 2

Convolution layer 2 maps 32 feature maps to 64 using ReLU (rectifier) activation function

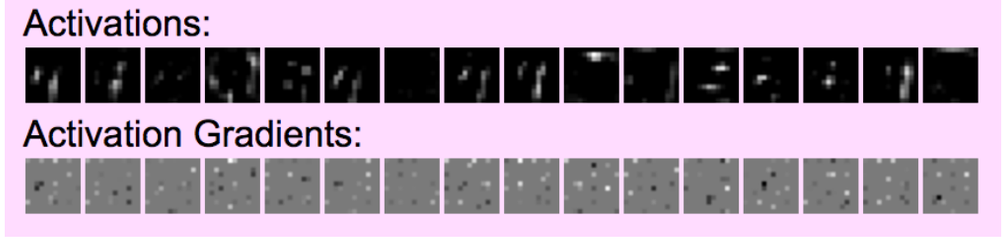
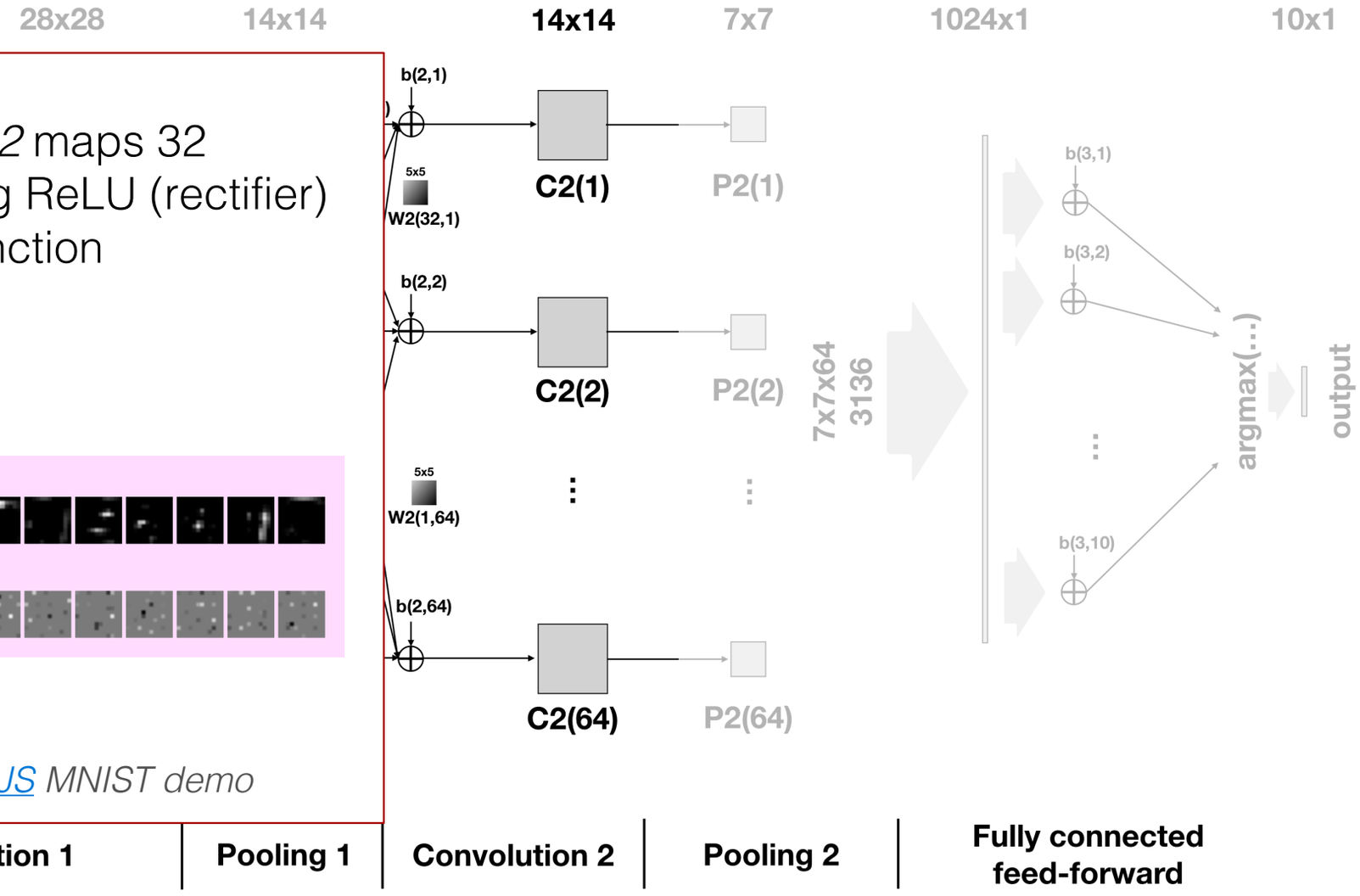
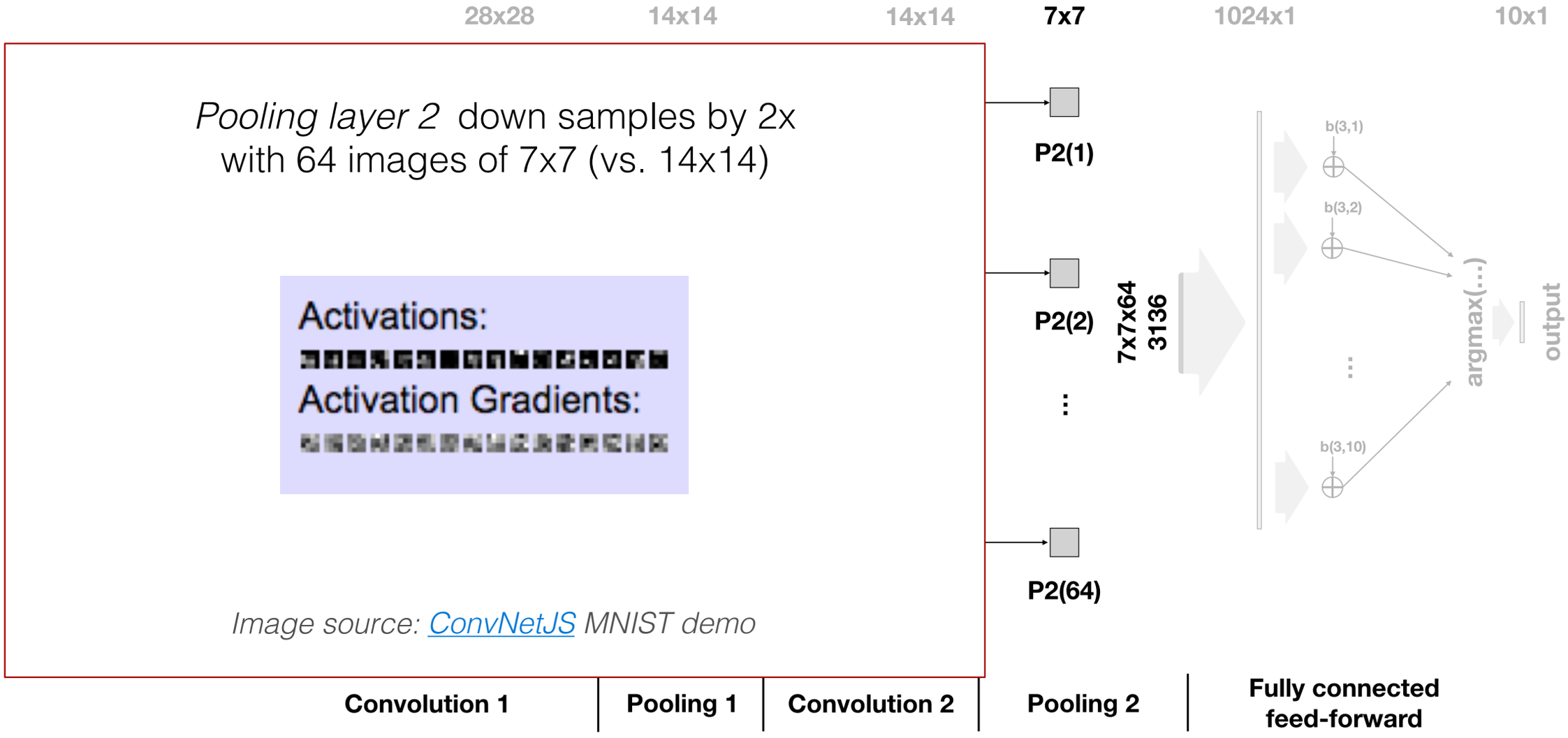


Image source: [ConvNetJS](#) MNIST demo



CNN Structure: Pooling 2



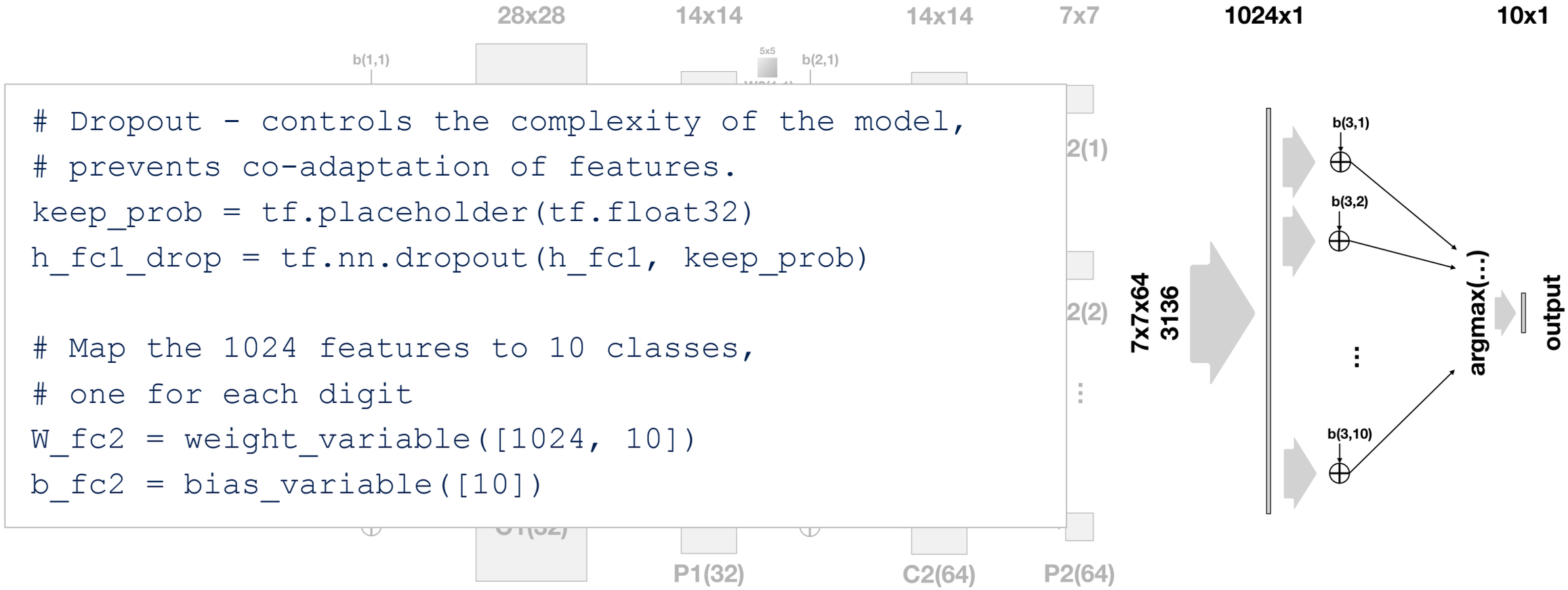
CNN Structure: Fully connected feed-forward

```
# Fully connected layer 1
# after 2 round of downsampling, our 28x28 image
# is down to 7x7x64 feature maps
# -- maps this to 1024 features.
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) +
b_fc1)
```

Convolution 1 | Pooling 1 | Convolution 2 | Pooling 2 | Fully connected feed-forward

CNN Structure: Dropout

```
# Dropout - controls the complexity of the model,
# prevents co-adaptation of features.
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Map the 1024 features to 10 classes,
# one for each digit
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
```



Convolution 1 | Pooling 1 | Convolution 2 | Pooling 2 | Fully connected feed-forward

CNN accuracy: ~98.95

```
1  # Take first 500 images
2  print('test accuracy %g' % accuracy.eval(feed_dict={x: mnist.test.images[:500, :784], y_: mnist.test.labels[:500,:10], keep_prob: 1.0}))
```

```
test accuracy 0.966
```

Command took 0.32 seconds -- by denny.g.lee@gmail.com at 6/26/2017, 8:41:30 PM on unknown cluster

Cmd 8

In fact, the accuracy could be much higher if we set it to higher iterations (e.g. 20000) and tested it against the full test dataset

```
step 19500, training accuracy 1
step 19600, training accuracy 1
step 19700, training accuracy 1
step 19800, training accuracy 1
step 19900, training accuracy 1
test accuracy 0.9895
```

Great References

Andrej Karparthy's ConvNetJS MNIST Demo

What is back propagation in neural networks?

CS231n: Convolutional Neural Networks for Visual Recognition
Syllabus and Slides | Course Notes | YouTube

- With particular focus on CS231n: Lecture 7: Convolution Neural Networks

Neural Networks and Deep Learning

TensorFlow

Great References

Deep Visualization Toolbox

Back Propagation with TensorFlow

TensorFrames: Google TensorFlow with Apache Spark

Integrating deep learning libraries with Apache Spark

Upcoming webinar: Build, Scale, and Deploy Deep Learning Pipelines with Ease

Microsoft

Q&A