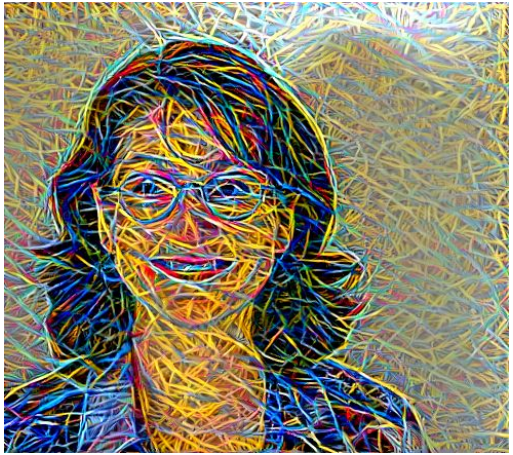
Diving into machine learning through TensorFlow

Amy Unruh, Eli Bixby, Julia Ferraioli

Google Cloud Platform

Slides: http://bit.ly/tf-workshop-slides
GitHub: https://github.com/amygdala/tensorflow-workshop

Your guides



Amy

Eli

Julia

What you'll learn about TensorFlow

How to:

- Build TensorFlow graphs
  - Inputs, variables, ops, tensors, sessions...
- Run/evaluate graphs, and how to train models
- Save and later load learned variables and models
- Use TensorBoard
- Intro to the distributed runtime

What we'll do from a ML perspective

(This is not really a ML tutorial. But…)

What we'll do from a ML perspective

- Look at a simple "MNIST" example
- Train a model that learns vector representations of words ("word2vec")
  - Use the results to determine how words relate to each other
- (if time) Use the learned vector representations to initialize a Convolutional NN for text classification
- Run a distributed training session

Agenda

- Welcome and logistics
- Setup
- Brief intro to machine learning
- What's TensorFlow?
- Diving in deeper with *word2vec*
- Using word embeddings from *word2vec* with a CNN for text classification
- Using the TensorFlow distributed runtime with Kubernetes

Setup -- install all the things!

- Clone or download this repo: https://github.com/amygdala/tensorflow-workshop
- Follow the installation instructions in that repo.
  You can run the workshop exercises in a Docker container, or alternately install and use a Conda virtual environment.
- If you're having trouble getting the bandwidth to download the data files, don't worry: most are for optional exercises.

(Very) Brief intro to NN concepts

What is Machine Learning?

data

algorithm

insight

What is Machine Learning?

“Field of study that gives computers the ability to learn without being explicitly programmed".

data

algorithm

insight

What is Machine Learning?

But: http://research.google.com/pubs/pub43146.html

("Machine Learning: The High Interest Credit Card of Technical Debt")

data

algorithm

insight

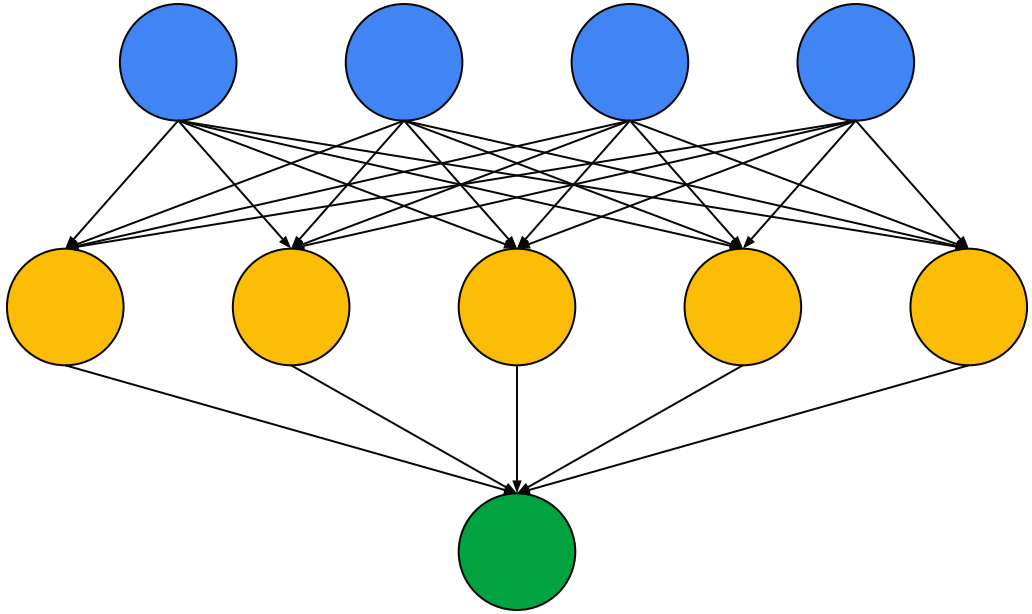
["this", "movie", "was", "great"]

Input →

Hidden →

Output

(label) →



["POS"]

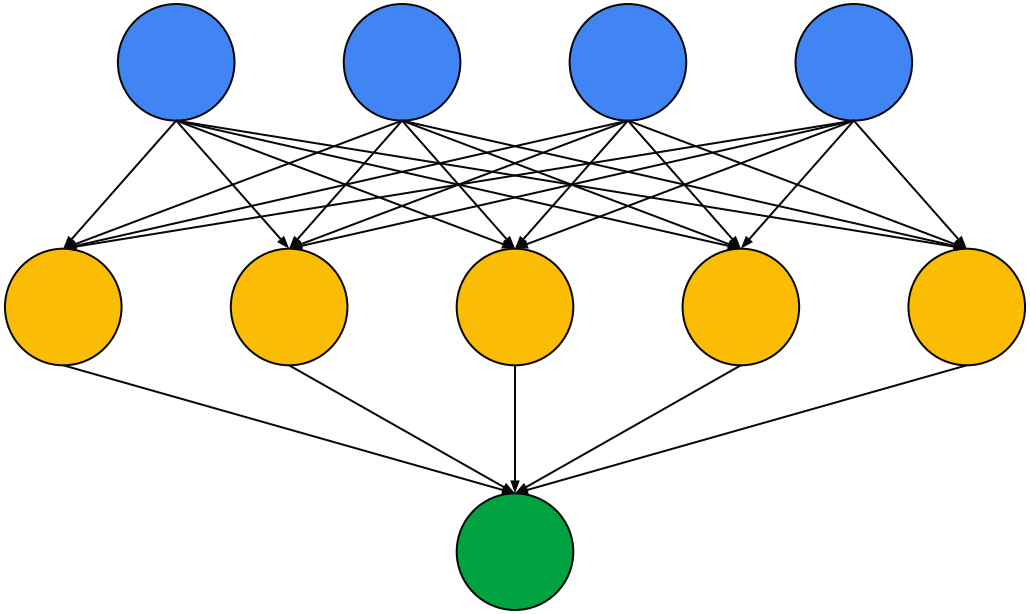
["this", "movie", "was", "great"]

Input →

Hidden →

Output

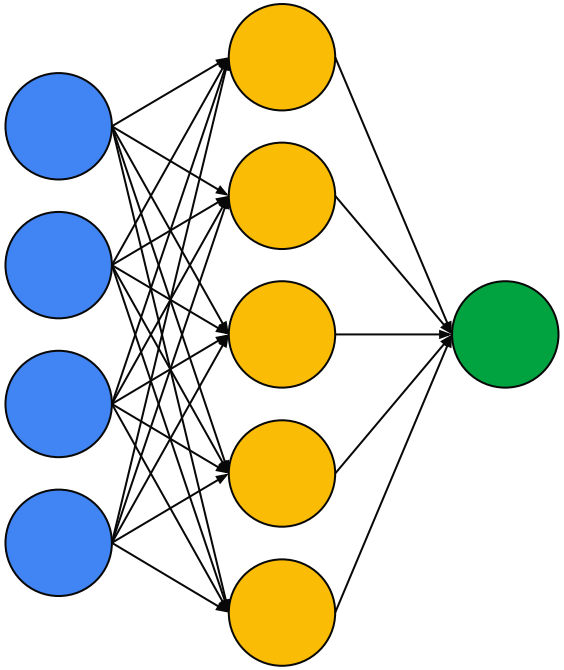
(score) →



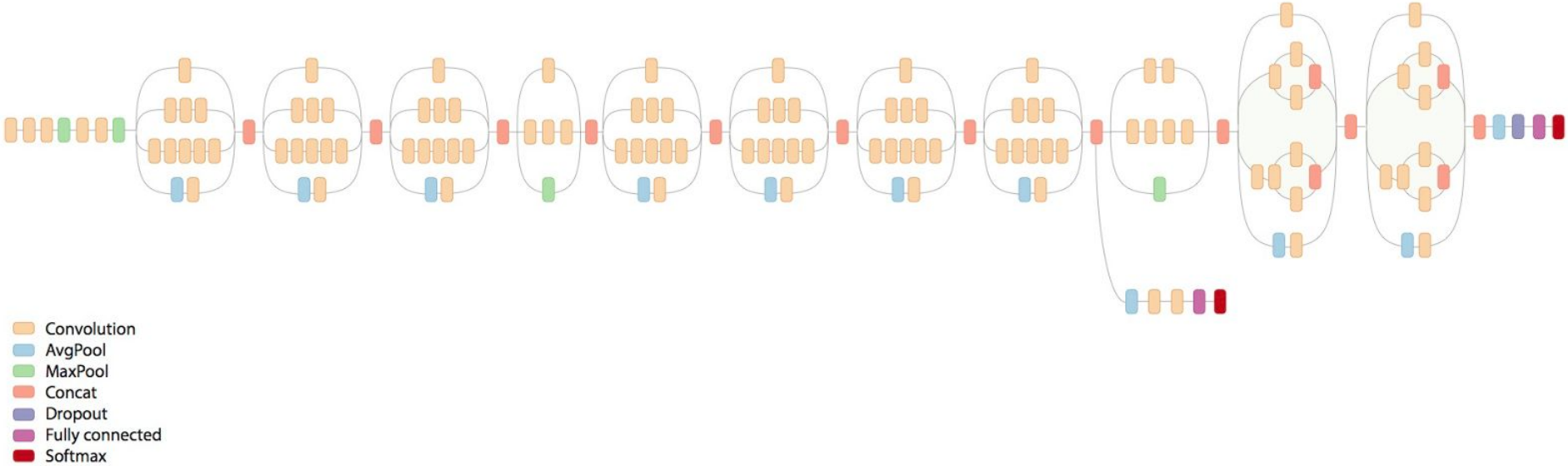
[.7]

Input Hidden Output(label)

pixels()



["cat"]



From: http://googleresearch.blogspot.com/2016_03_01_archive.html

Related concepts / resources

- https://www.tensorflow.org/versions/r0.8/api_docs/python/nn.html
- Introduction to Neural Networks: http://bit.ly/intro-to-ann
- Logistic versus Linear Regression: http://bit.ly/log-vs-lin
- Curse of Dimensionality: http://bit.ly/curse-of-dim
- A Few Useful Things to Know about Machine Learning: http://bit.ly/useful-ml-intro

What’s TensorFlow?

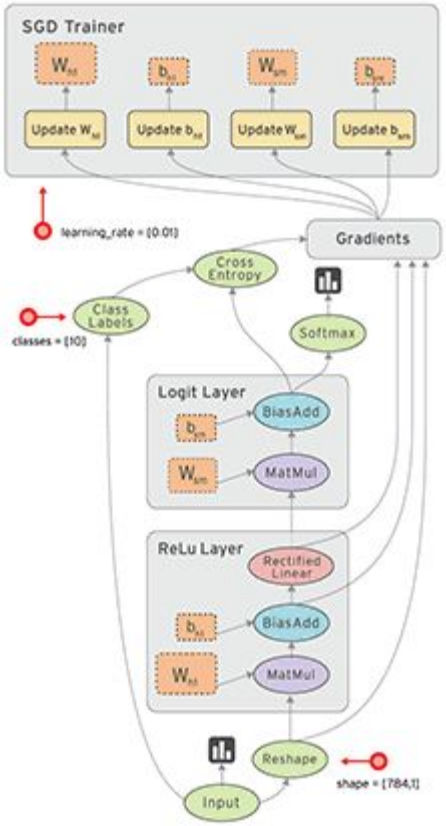
(and why is it so great for this stuff?)

TensorFlow

Operates over **tensors**: *n-dimensional arrays*

Using a **flow graph**: *data flow computation framework*

- Flexible, intuitive construction
- automatic differentiation
- Support for threads, queues, and asynchronous computation; [distributed runtime]
- Train on CPUs, GPUs
- Run wherever you like

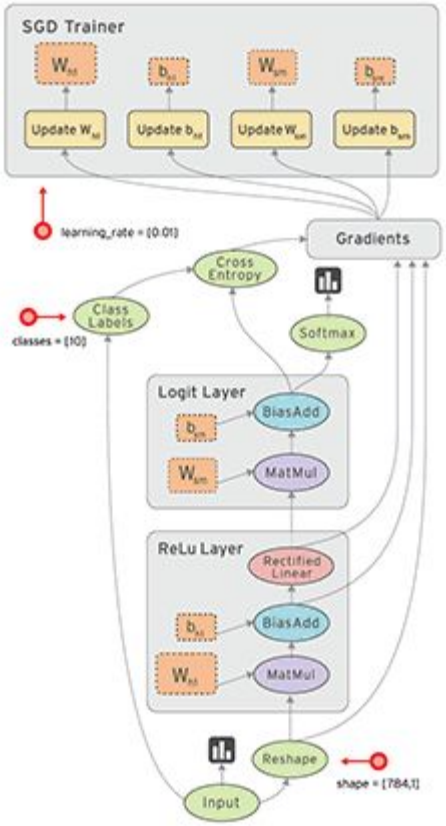


TensorFlow

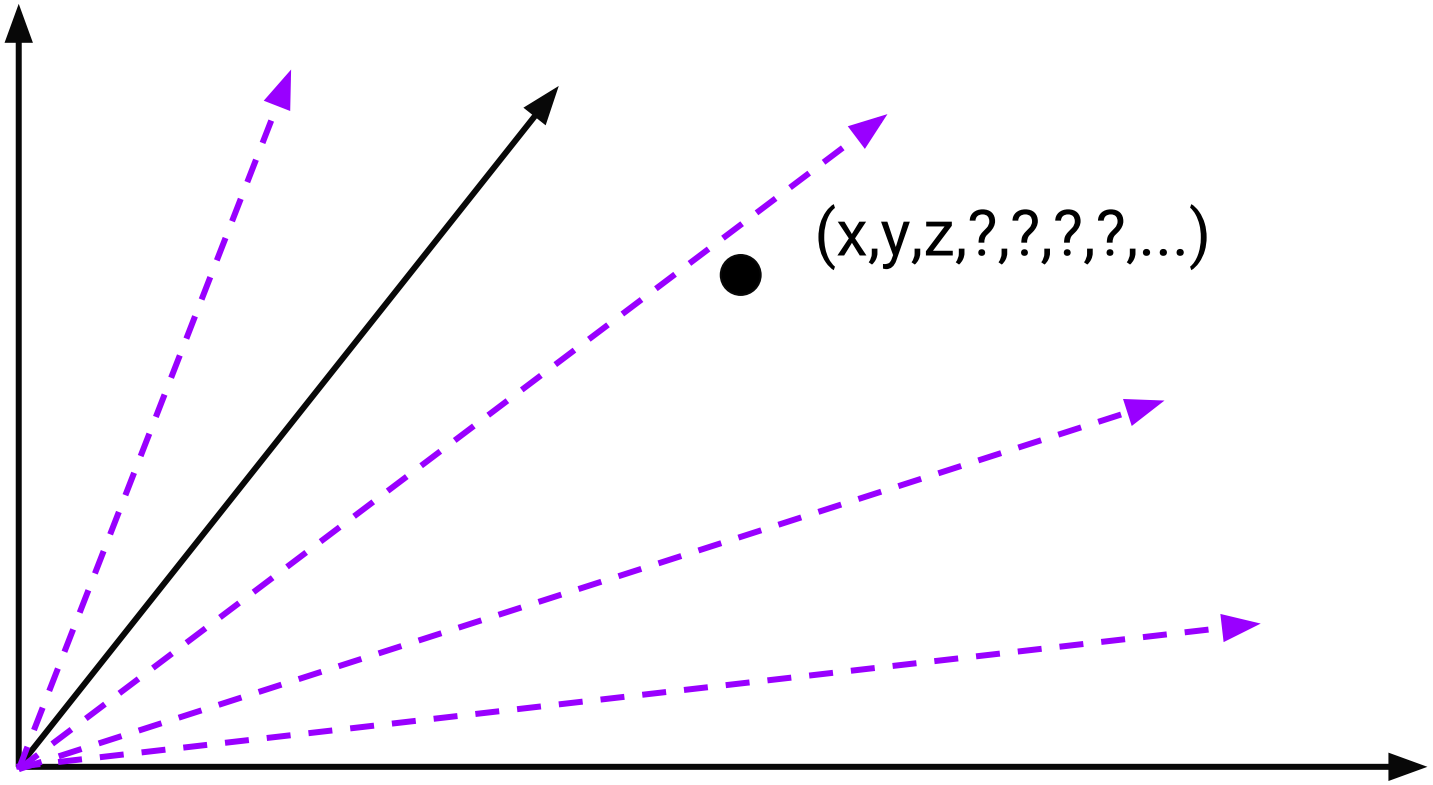
Operates over **tensors**: *n-dimensional arrays*

Using a **flow graph**: *data flow computation framework*

- Flexible, intuitive construction
- automatic differentiation
- Support for threads, queues, and asynchronous computation; distributed runtime
- Train on CPUs, GPUs, ...and coming 'soon', **TPUS**...
- Run wherever you like



https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html


(x,y,z,?,?,?,?,...)

(x,y,z,?,?,?,?,...) => tensor

Core TensorFlow data structures and concepts...

- **Graph**: A TensorFlow computation, represented as a dataflow graph.
  - collection of ops that may be executed together as a group
- **Operation**: a graph node that performs computation on tensors
- **Tensor**: a handle to one of the outputs of an Operation
  - provides a means of computing the value in a TensorFlow Session.

Core TensorFlow data structures and concepts

- **Constants**
- **Placeholders**: must be fed with data on execution
- **Variables**: a modifiable tensor that lives in TensorFlow's graph of interacting operations.
- **Session**: encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.

Operations

Category	*Examples*
Element-wise math ops	**Add**, Sub, **Mul**, Div, Exp, Log, Greater, Less…
Array ops	**Concat**, Slice, **Split**, Constant, Rank, **Shape…**
Matrix ops	**MatMul**, MatrixInverse, MatrixDeterminant…
Stateful ops	**Variable**, Assign, AssignAdd...
NN building blocks	**SoftMax**, Sigmoid, **ReLU**, **Convolution2D…**
Checkpointing ops	**Save**, **Restore**
Queue & synch ops	Enqueue, Dequeue, MutexAcquire…
Control flow ops	Merge, Switch, Enter, Leave...

(https://www.tensorflow.org/versions/r0.8/api_docs/python/nn.html, https://www.tensorflow.org/versions/r0.8/api_docs/python/train.html, ...)

Creating and running a TensorFlow graph

Create a TensorFlow graph

Follow along at: https://github.com/amygdala/tensorflow-workshop/tree/master/workshop_sections/starter_tf_graph

```
import numpy as np
import tensorflow as tf

graph = tf.Graph()
m1 = np.array([[1.,2.], [3.,4.], [5.,6.], [7., 8.]], dtype=np.float32)

with graph.as_default():
  # Input data.
  m1_input = tf.placeholder(tf.float32, shape=[4,2])
```

Create a TensorFlow graph

Follow along at: https://github.com/amygdala/tensorflow-workshop/tree/master/workshop_sections/starter_tf_graph

```
# Ops and variables pinned to the CPU because of missing GPU implementation
with tf.device('/cpu:0'):

  m2 = tf.Variable(tf.random_uniform([2,3], -1.0, 1.0))
  m3 = tf.matmul(m1_input, m2)

  # This is an identity op with the side effect of printing data when evaluating.
  m3 = tf.Print(m3, [m3], message="m3 is: ")

  # Add variable initializer.
  init = tf.initialize_all_variables()
```

Run the TensorFlow graph in a session

Follow along at: https://github.com/amygdala/tensorflow-workshop/tree/master/workshop_sections/starter_tf_graph

```
with tf.Session(graph=graph) as session:
  # We must initialize all variables before we use them.
  init.run()
  print("Initialized")

  print("m2: {}".format(m2))
  print("eval m2: {}".format(m2.eval()))

  feed_dict = {m1_input: m1}

  result = session.run([m3], feed_dict=feed_dict)
  print("\nresult: {}\n".format(result))
```

Exercise: more matrix operations

Workshop section: `starter_tf_graph`

Exercise: Modify the graph

Follow along at: https://github.com/amygdala/tensorflow-workshop/tree/master/workshop_sections/starter_tf_graph

On your own:

- Add `m3` to itself
- Store the result in `m4`
- Return the results for both `m3` and `m4`

Useful link: http://bit.ly/tf-math

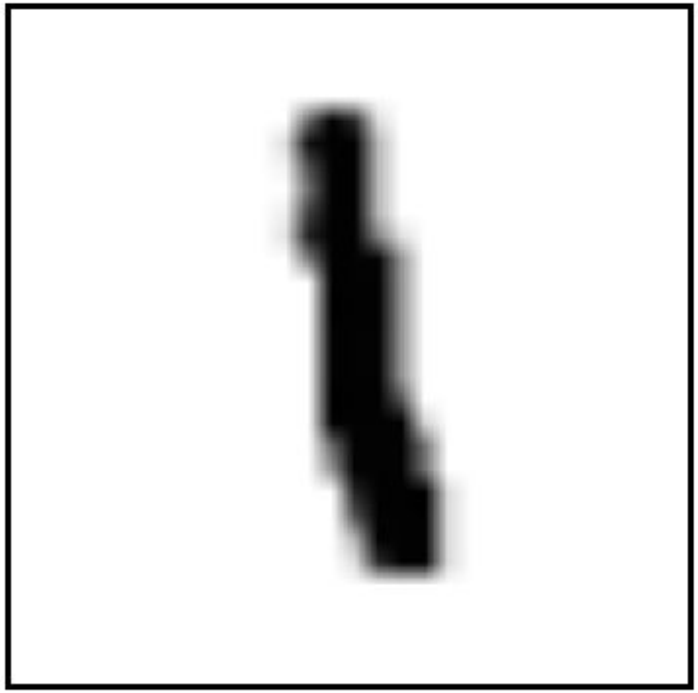
Related concepts / resources

- TensorFlow Graphs: http://bit.ly/tf-graphs
- TensorFlow Variables: http://bit.ly/tf-variables
- TensorFlow Math: http://bit.ly/tf-math

Example: building a neural net in TensorFlow

Computer Vision -- MNIST

Computer Vision -- MNIST



$$\simeq \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

http://bit.ly/tf-workshop-slides

bit.ly/tensorflow-workshop

TensorFlow - initialization

```
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

init = tf.initialize_all_variables()
```

this will become the batch size, 100

28 x 28 grayscale images

Training = computing variables W and b

TensorFlow - success metrics

flattening images

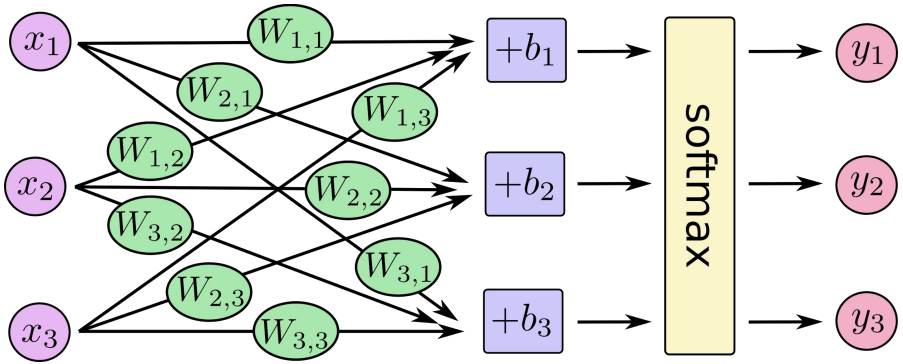
```
# model
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)
# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])
```

“one-hot” encoded

```
# loss function
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
   reduction_indices=[1]))
```

“one-hot” decoding

```
# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

TensorFlow - training

learning rate

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)
```

loss function

Because TensorFlow knows the entire graph of your computations, it can automatically use the backpropagation algorithm to efficiently determine how your variables affect the cost you ask it to minimize. Then it can apply your choice of optimization algorithm to modify the variables and reduce the cost.

TensorFlow - run!

```
sess = tf.Session()
sess.run(init)

for i in range(1000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ?
    a,c = sess.run([accuracy, cross_entropy], feed_dict=train_data)

    # success on test data ?
    test_data={X: mnist.test.images, Y_: mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy], feed=test_data)
```

running a Tensorflow computation, feeding placeholders

do this every N iterations

Common TF NN graph construction pattern:

- **inference** - Builds the part of the graph for running the network forward to make predictions.
- **loss** - Adds to the inference graph the ops required to generate loss/cost.
- **training** - add optimizer to minimize loss.

TensorFlow - full python code

initialization

```python
import tensorflow as tf


X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
init = tf.initialize_all_variables()
```

model

```python
# model
Y=tf.nn.softmax(tf.matmul(tf.reshape(X,[-1, 784]), W) + b)

# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])

# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct,tf.float32))
```

success metrics

training step

```python
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)


sess = tf.Session()
sess.run(init)

for i in range(1000):
        # load batch of images and correct answers
        batch_X, batch_Y = mnist.train.next_batch(100)
        train_data={X: batch_X, Y_: batch_Y}

        # train
        sess.run(train_step, feed_dict=train_data)

        # success ? add code to print it
        a,c = sess.run([accuracy, cross_entropy],
                       feed=train_data)

        # success on test data ?
        test_data={X:mnist.test.images, Y_:mnist.test.labels}
        a,c = sess.run([accuracy, cross_entropy],
                       feed=test_data)
```

Run

Related concepts / resources

- Softmax Function: http://bit.ly/softmax
- MNIST: http://bit.ly/mnist
- Loss Function: http://bit.ly/loss-fn
- Gradient Descent Overview: http://bit.ly/gradient-descent
- Training, Testing, & Cross Validation: http://bit.ly/ml-eval

Diving in deeper with ***word2vec***: Learning vector representations of words

What is word2vec?

- A model for learning *vector representations of words* -- **word embeddings** (feature vectors for words in supplied text).

- Vector space models address an NLP **data sparsity problem** encountered when words are discrete IDs
    - Map similar words to nearby points.

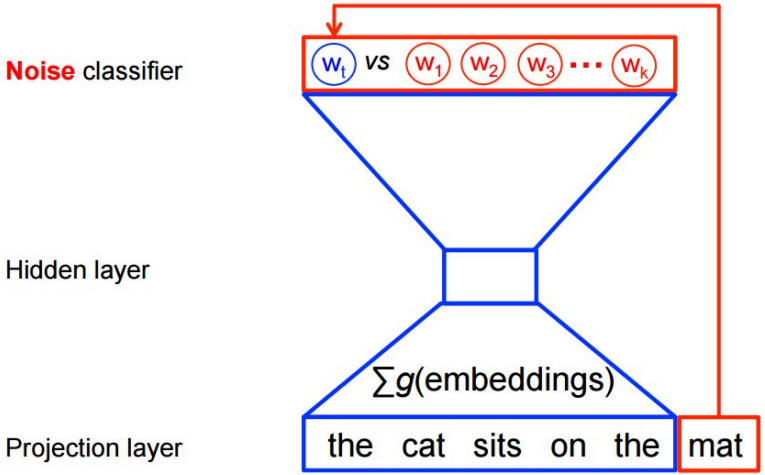
Two categories of approaches:

- count-based (e.g. LSA)
- Predictive: try to predict a word from its neighbors using learned embeddings (e.g. **word2vec** & other neural probabilistic language models)

NIPS paper: Mikolov et al.: http://bit.ly/word2vec-paper

Two flavors of word2vec

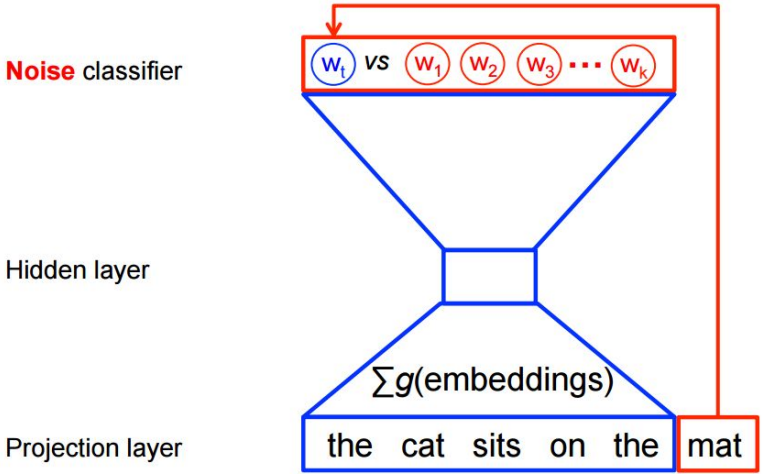
- Continuous Bag-of-Words (COBW)
  - Predicts target words from source context words
- **Skip-Gram**
  - Predicts source context words from target



https://www.tensorflow.org/versions/r0.8/images/nce-nplm.png

Making word2vec scalable

- Instead of a full probabilistic model… Use logistic regression to discriminate target words from imaginary (noise) words.
- Noise-contrastive estimation (NCE) loss
  - tf.nn.**nce_loss**()
  - Scales with number of noise words



https://www.tensorflow.org/versions/r0.8/images/nce-nplm.png

Skip-Gram model
(predict source context-words from target words)

Context/target pairs, window-size of 1 in both directions:

```
the quick brown fox jumped over the lazy dog ... →

([the, brown], quick), ([quick, fox], brown), ([brown,
jumped], fox), …
```

Skip-gram model
(predict source context-words from target words)

Context/target pairs, window-size of 1 in both directions:

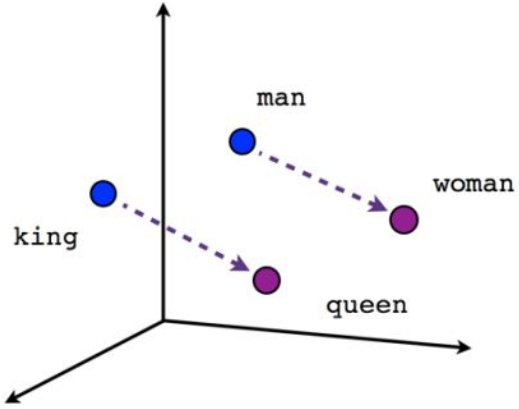
```
the quick brown fox jumped over the lazy dog ... →
```

```
([the, brown], quick), ([quick, fox], brown), ([brown,
jumped], fox), …
```

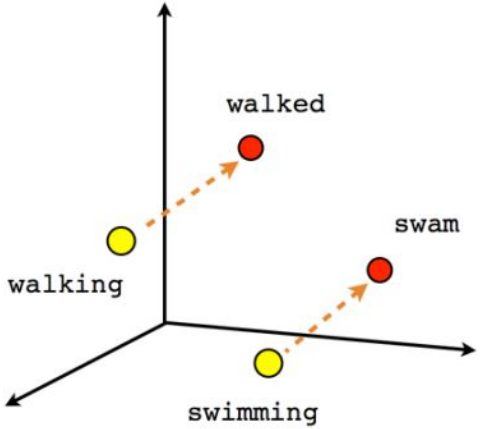
Input/output pairs:

```
(quick, the), (quick, brown), (brown, quick), (brown,
fox), …
```

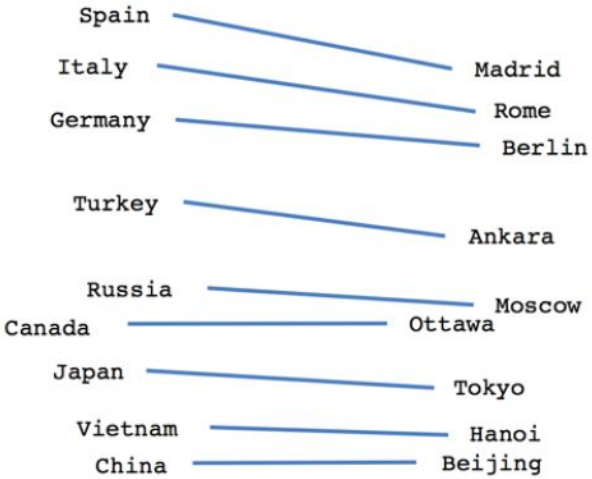
Typically optimize with stochastic gradient descent (SGD) using minibatches



Male-Female



Verb tense



Country-Capital

https://www.tensorflow.org/versions/r0.8/images/linear-relationships.png

```
model.nearby([b'cat'])

b'cat'                1.0000
b'cats'               0.6077
b'dog'                0.6030
b'pet'                0.5704
b'dogs'               0.5548
b'kitten'             0.5310
b'toxoplasma'         0.5234
b'kitty'              0.4753
b'avner'              0.4741
b'rat'                0.4641
b'pets'               0.4574
b'rabbit'             0.4501
b'animal'             0.4472
b'puppy'              0.4469
b'veterinarian'       0.4435
b'raccoon'            0.4330
b'squirrel'           0.4310
```

```
model.analogy(b'cat',
b'kitten', b'dog')
Out[1]: b'puppy'
```

Exercise: **word2vec**, and introducing **TensorBoard**

Workshop section: `intro_word2vec`

```
# Input data.
train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
valid_dataset = tf.constant(valid_examples, dtype=tf.int32)

# Ops and variables pinned to the CPU because of missing GPU implementation
with tf.device('/cpu:0'):
  # Look up embeddings for inputs.
  embeddings = tf.Variable(
      tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
  embed = tf.nn.embedding_lookup(embeddings, train_inputs)

  # Construct the variables for the NCE loss
  nce_weights = tf.Variable(
      tf.truncated_normal([vocabulary_size, embedding_size],
                          stddev=1.0 / math.sqrt(embedding_size)))
  nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
```

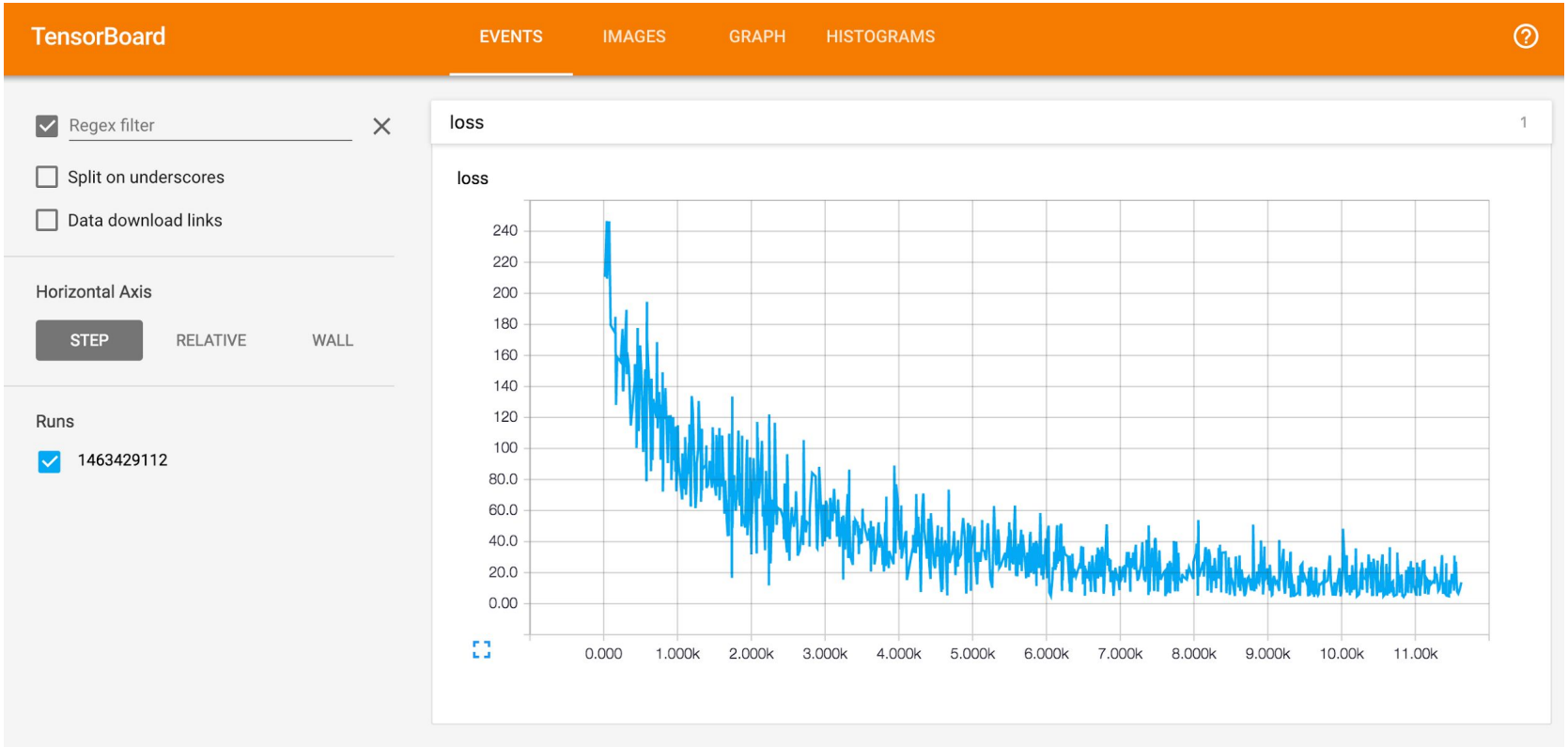
(noise-contrastive estimation loss: https://www.tensorflow.org/versions/r0.8/api_docs/python/nn.html#nce_loss)

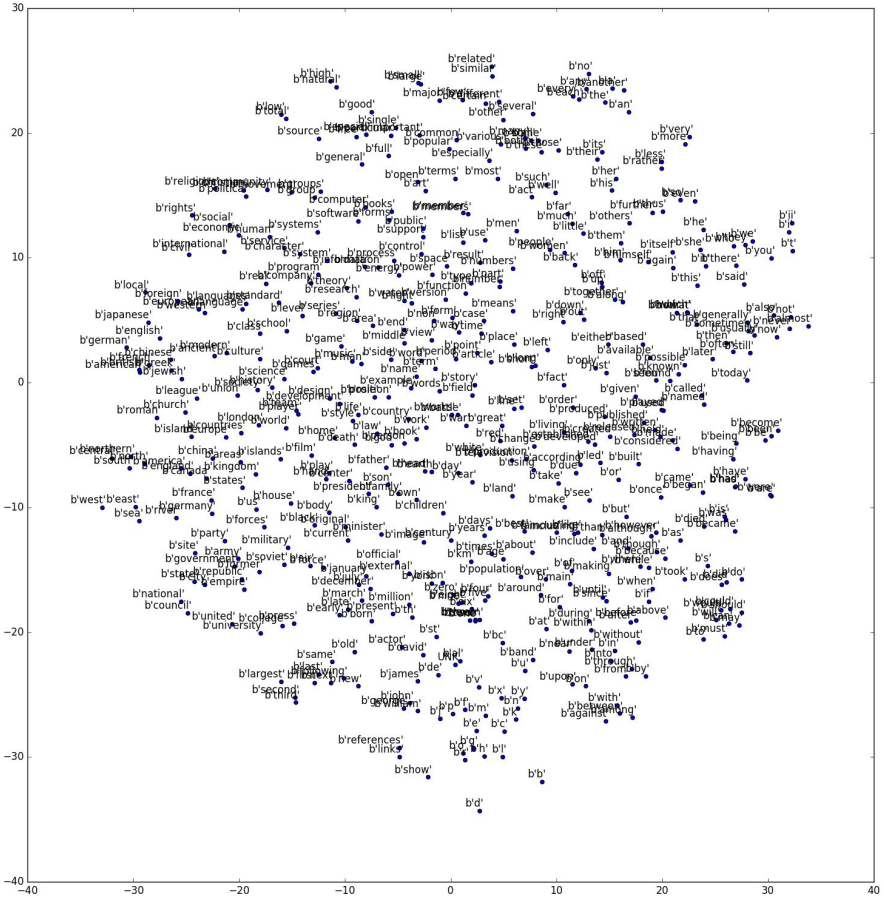
```
# Compute the average NCE loss for the batch.
# tf.nce_loss automatically draws a new sample of the negative labels each
# time we evaluate the loss.
loss = tf.reduce_mean(
    tf.nn.nce_loss(nce_weights, nce_biases, embed, train_labels,
                   num_sampled, vocabulary_size))

# Construct the SGD optimizer using a learning rate of 1.0.
optimizer = tf.train.GradientDescentOptimizer(1.0).minimize(loss)
```

```
with tf.Session(graph=graph) as session:
  ...
  for step in xrange(num_steps):
    batch_inputs, batch_labels = generate_batch(
        batch_size, num_skips, skip_window)
    feed_dict = {train_inputs : batch_inputs, train_labels : batch_labels}

    # We perform one update step by evaluating the optimizer op (including it
    # in the list of returned values for session.run()
    _, loss_val = session.run([optimizer, loss], feed_dict=feed_dict)
```


TensorBoard
EVENTS
IMAGES
GRAPH
HISTOGRAMS
Regex filter
Split on underscores
Data download links
Horizontal Axis
STEP
RELATIVE
WALL
Runs
1463429112
loss
1
loss
240
220
200
180
160
140
120
100
80.0
60.0
40.0
20.0
0.00
0.000
1.000k
2.000k
3.000k
4.000k
5.000k
6.000k
7.000k
8.000k
9.000k
10.00k
11.00k


30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40

Exercise: change **word2vec** to additionally output ‘nearby’ info for a specific word

Workshop section: `intro_word2vec`

```
Nearest to b'government':
b'governments', b'leadership', b'regime',
b'crown', b'rule', b'leaders', b'parliament',
b'elections',
```

Related concepts / resources

- Word Embeddings: http://bit.ly/word-embeddings
- word2vec Tutorial: http://bit.ly/tensorflow-word2vec
- Continuous Bag of Words vs Skip-Gram: http://bit.ly/cbow-vs-sg

Back to those word embeddings from **word2vec**…

Can we use them for analogies? Synonyms?

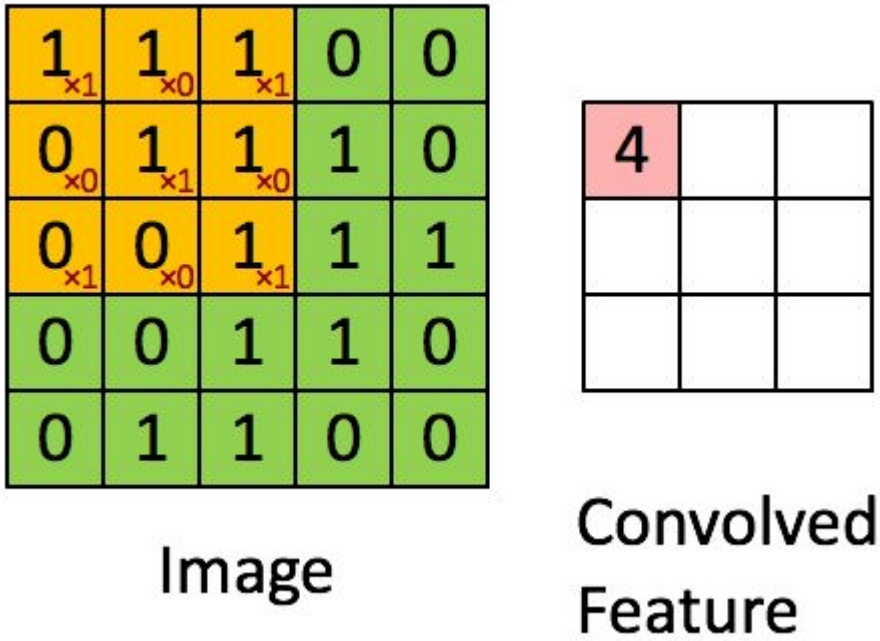
Demo: Accessing the learned word embeddings from (an optimized) **word2vec**

Workshop section: `word2vec_optimized`

and word embeddings

$\vee$

Using a Convolutional NN for Text Classification



Convolution with 3×3 Filter. Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

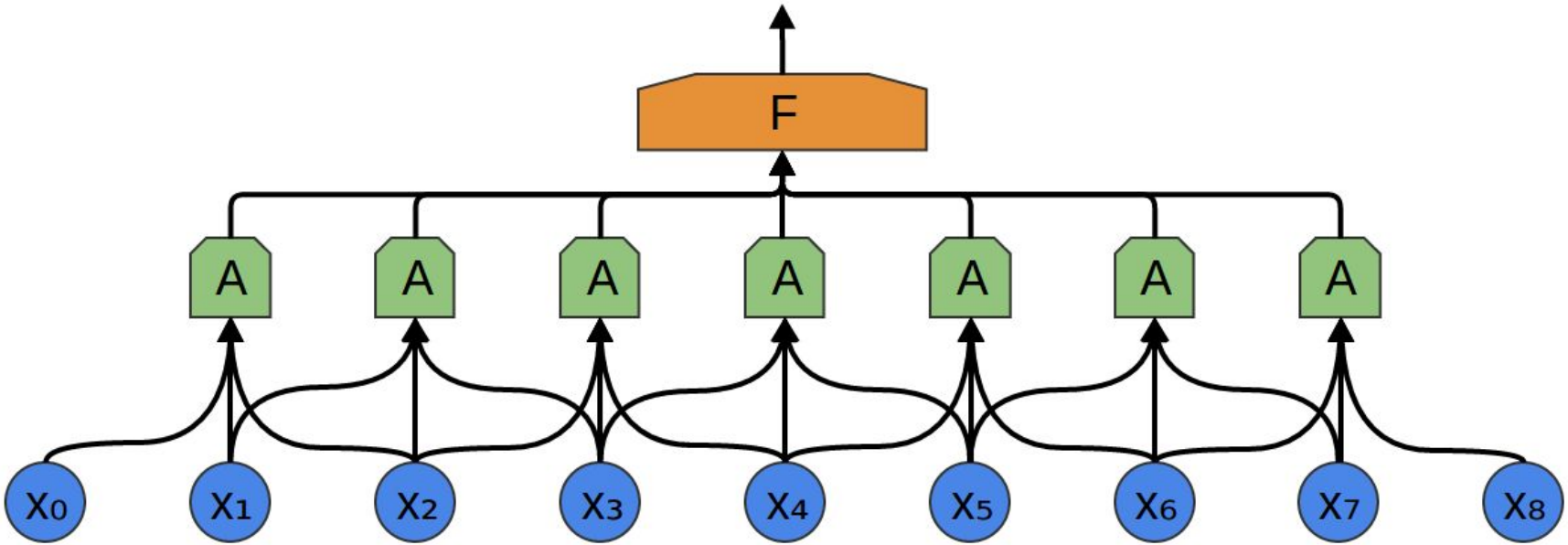


Image from: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

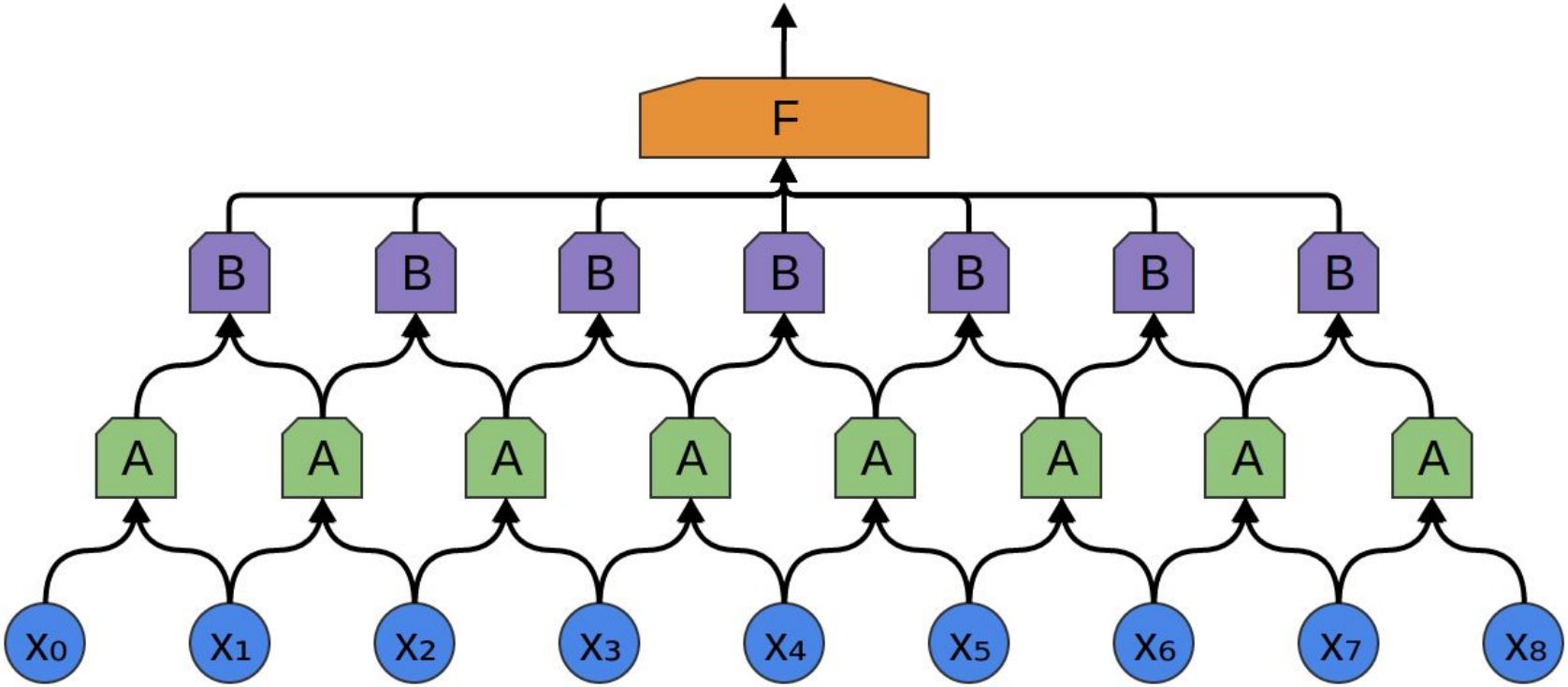
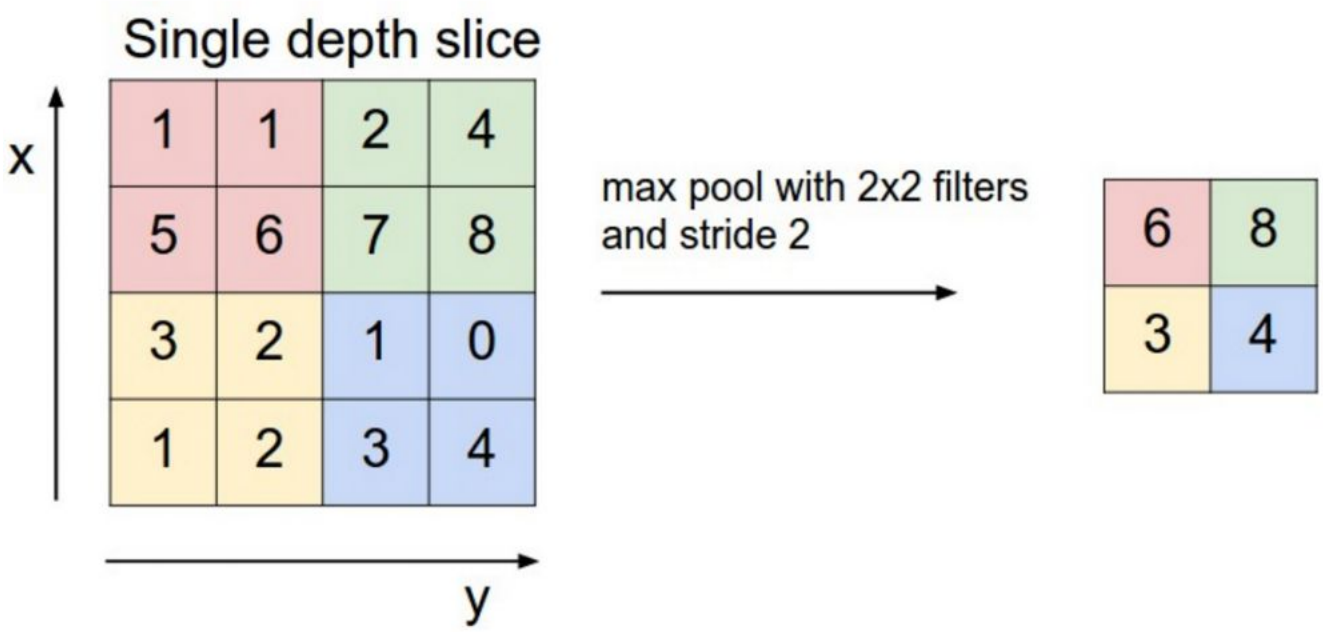


Image from: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/



Max pooling in CNN. Source: http://cs231n.github.io/convolutional-networks/#pool, via http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

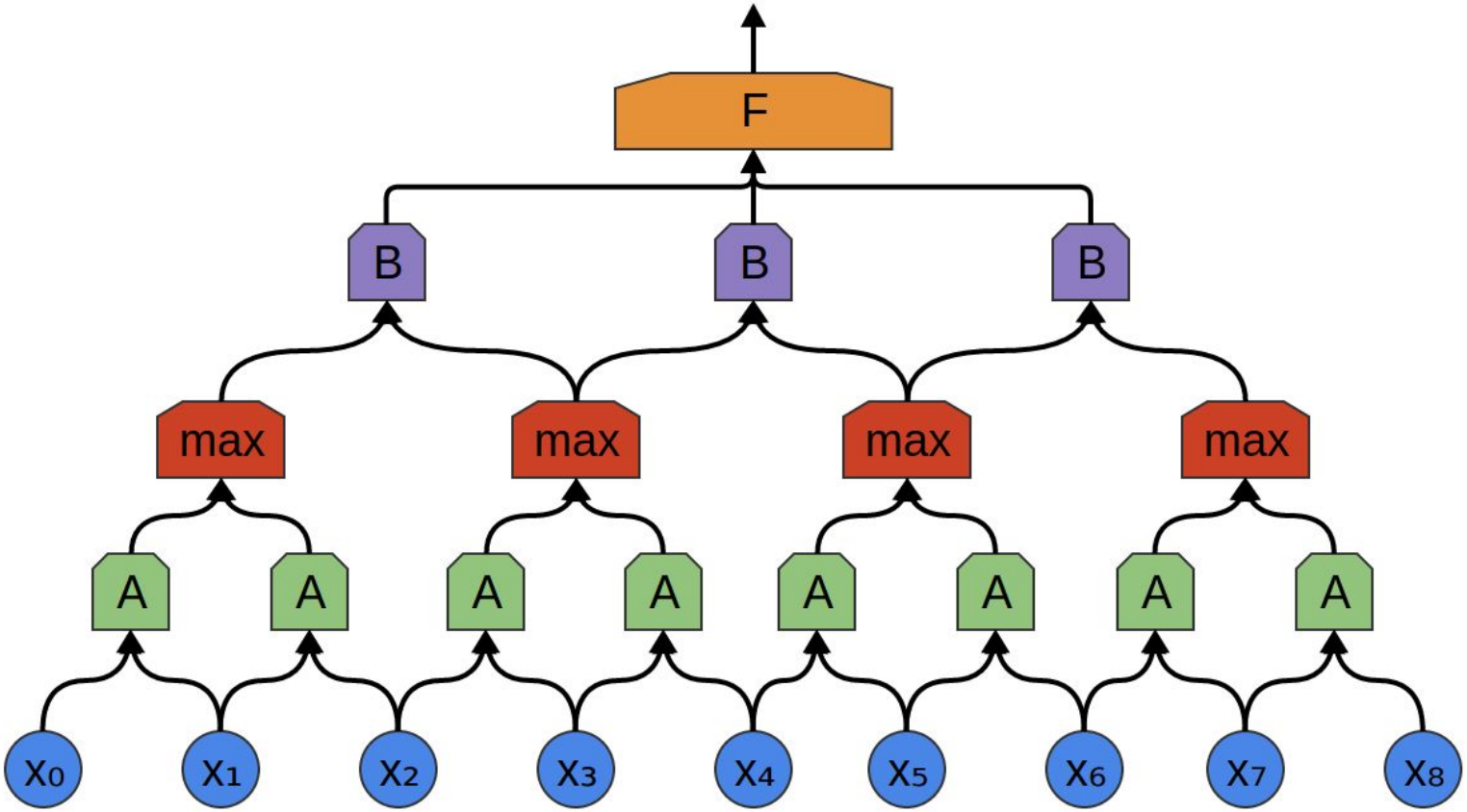


Image from: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

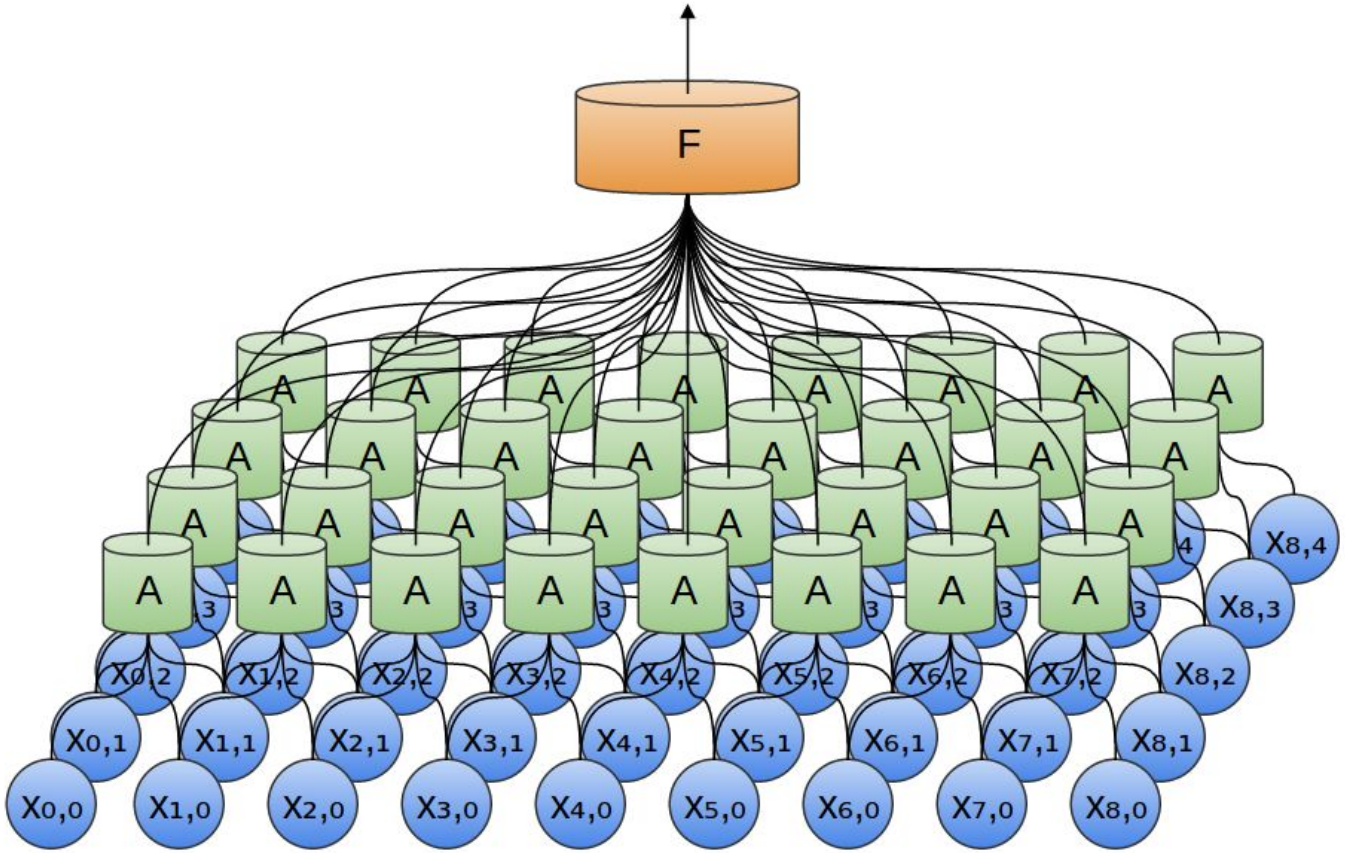


Image from: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

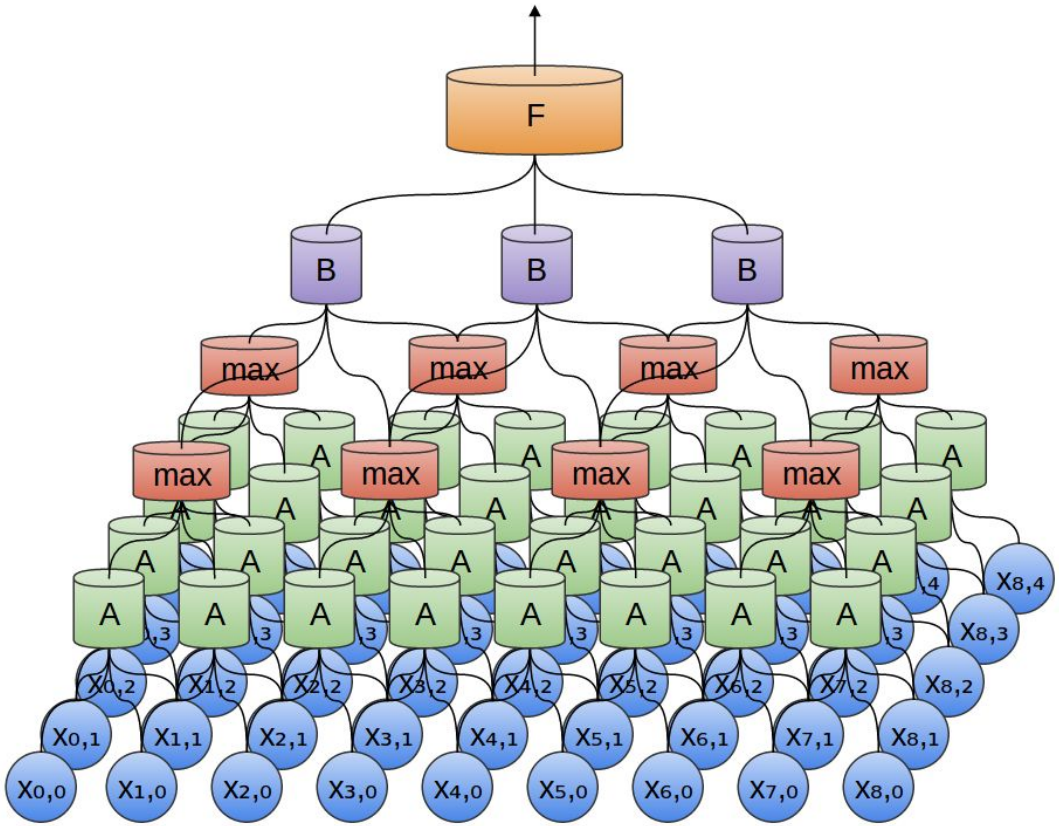
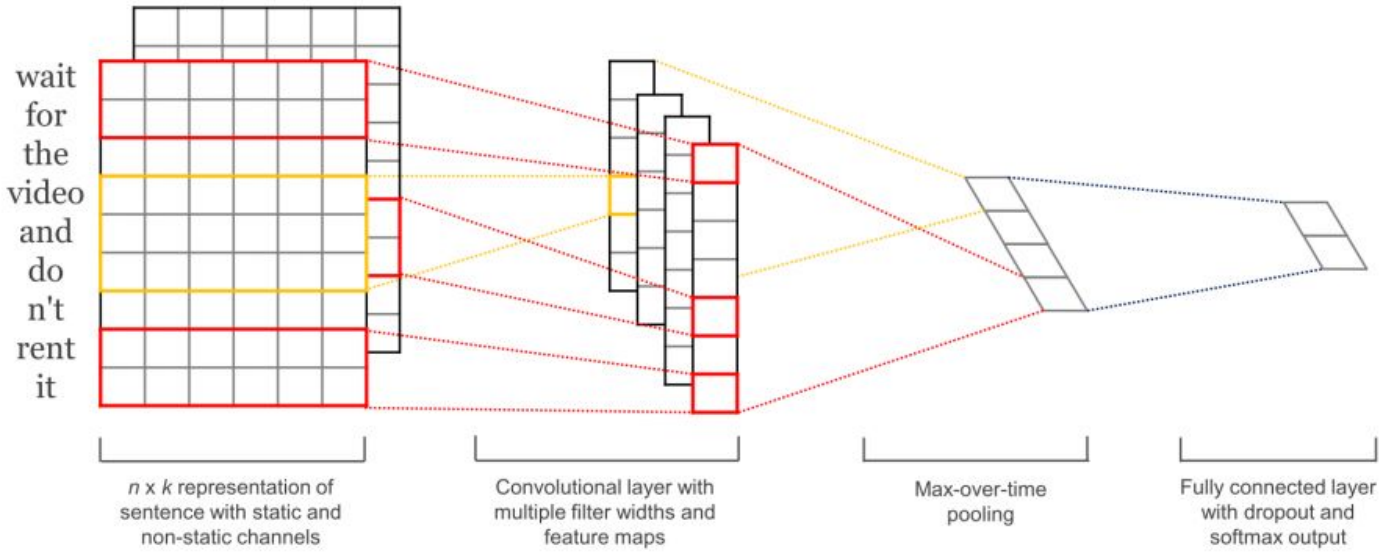


Image from: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/



From: Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. http://arxiv.org/abs/1408.5882

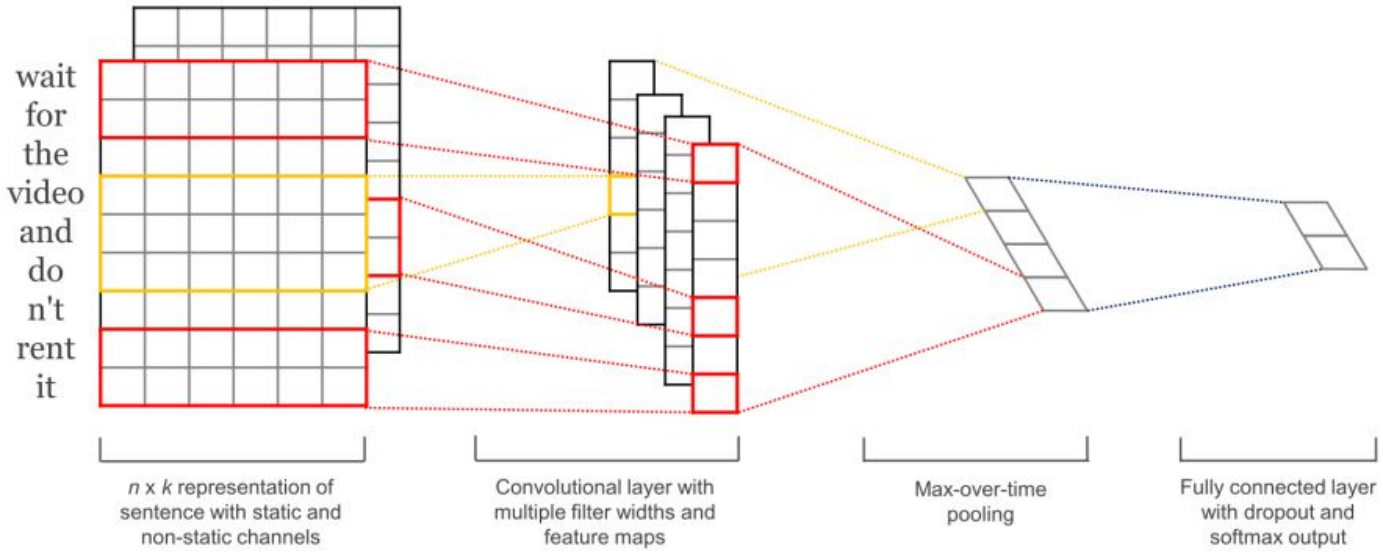
Related concepts / resources

- Convolutional Neural Networks: http://bit.ly/cnn-tutorial
- Document Classification: http://bit.ly/doc-class
- Rectifier: http://bit.ly/rectifier-ann
- MNIST: http://bit.ly/mnist

Exercise: Using a CNN for text classification (part I)

Workshop section:

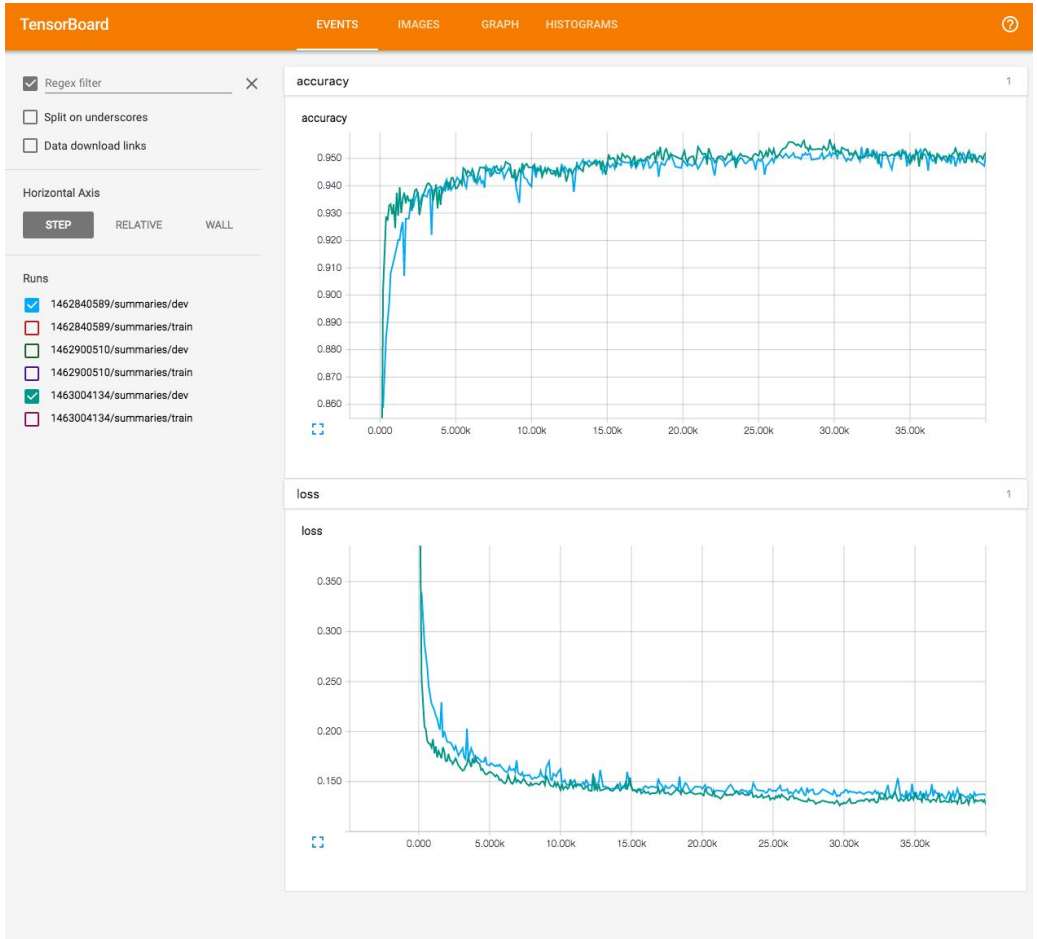
`cnn_text_classification`



From: Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. http://arxiv.org/abs/1408.5882

Exercise: Using word embeddings from *word2vec* with the text classification CNN (part 2)

Workshop section: `cnn_text_classification`



bit.ly/tensorflow-workshop

Checkpointing and reloading models

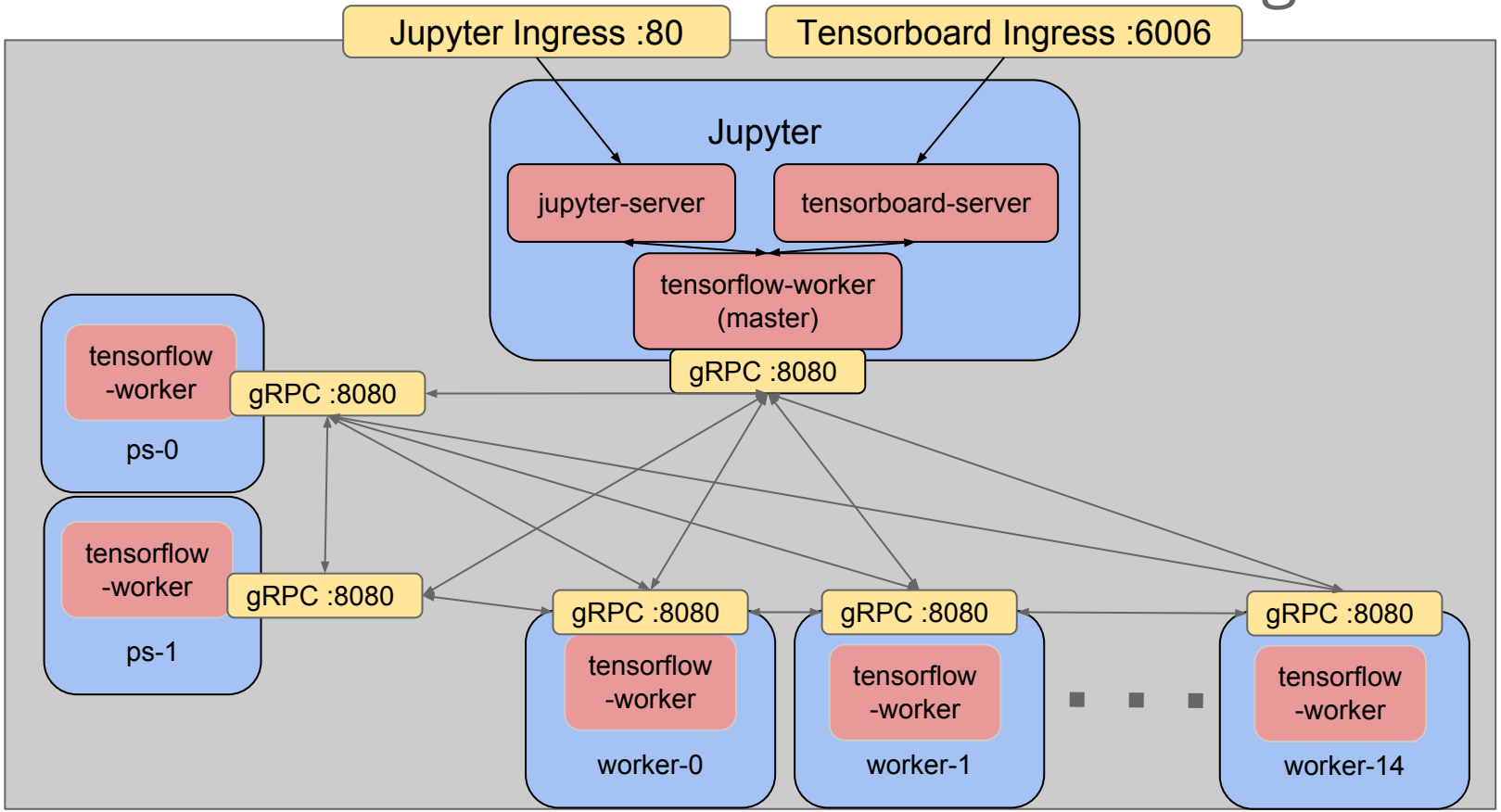
Workshop section: `cnn_text_classification`

Using the TensorFlow distributed runtime with Kubernetes

Exercise/demo: Distributed word2vec on a Kubernetes cluster

Workshop section:
`distributed_tensorflow`

Kubernetes as a Tensorflow Cluster Manager



bit.ly/tensorflow-workshop

Model Parallelism: Full Graph Replication

- Similar code runs on each worker and workers use flags to determine their role in the cluster:

```
server = tf.train.Server(cluster_def, job_name=this_job_name,
task_index=this_task_index)
if this_job_name == 'ps':
    server.join()
elif this_job_name=='worker':
    // cont’d
```

Model Parallelism: Full Graph Replication

- Copies of each variable and op are deterministically assigned to parameter servers and worker

```
with tf.device(tf.train.replica_device_setter(
    worker_device="/job:worker/task:{}".format(this_task_index),
    cluster=cluster_def)):
    // Build the model
    global_step = tf.Variable(0)
    train_op = tf.train.AdagradOptimizer(0.01).minimize(
        loss, global_step=global_step)
```

Model Parallelism: Full Graph Replication

- Workers coordinate once-per-cluster tasks using a Supervisor and train independently

```
sv = tf.train.Supervisor(
         is_chief = (this_task_index==0),
         // training, summary and initialization ops))

with sv.managed_session(server.target) as session:
     step = 0
    while not sv.should_stop() and step < 1000000:
        # Run a training step asynchronously.
         _, step = sess.run([train_op, global_step])
```

Model Parallelism: Sub-Graph Replication

- Can pin operations specifically to individual nodes in the cluster

```
with tf.Graph().as_default():
    losses = []
    for worker in loss_workers:
      with tf.device(worker):
        // Computationally expensive model section
        // e.g. loss calculation
        losses.append(loss)
```

Model Parallelism: Sub-Graph Replication

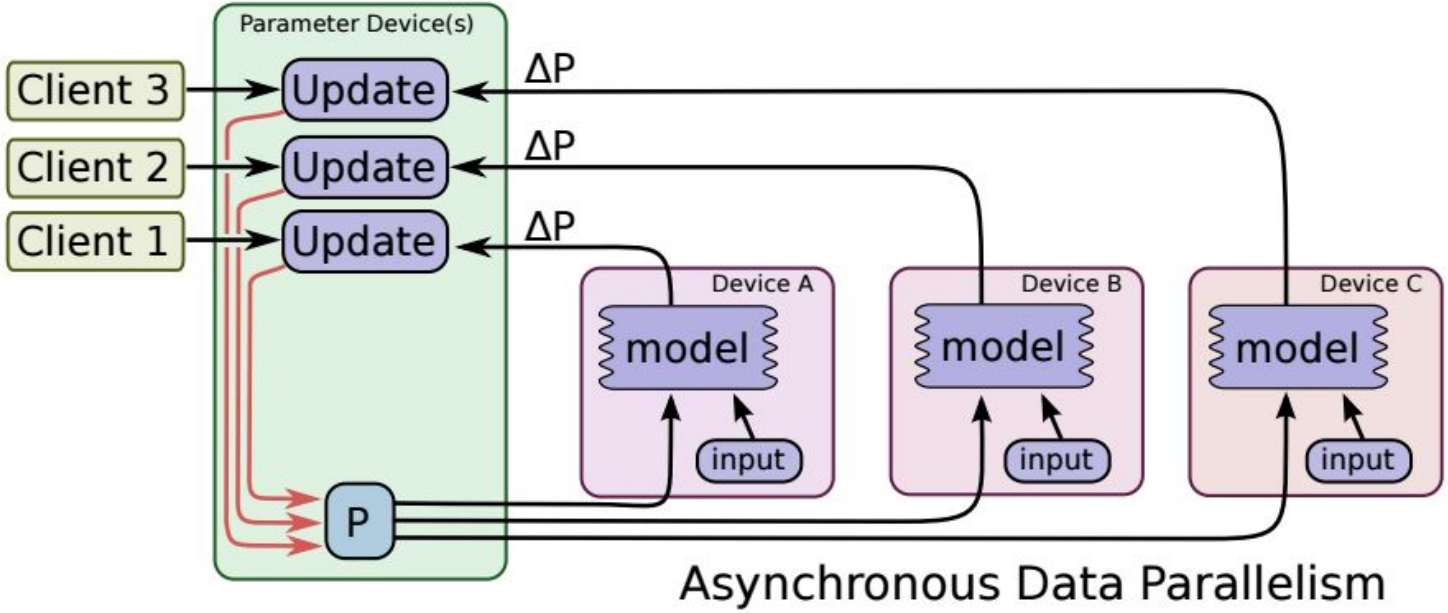
- Can use a single synchronized training step, averaging losses from multiple workers

```
with tf.device(master):
    losses_avg = tf.add_n(losses) / len(workers)
    train_op = tf.train.AdagradOptimizer(0.01).minimize(
            losses_avg, global_step=global_step)

    with tf.Session('grpc://master.address:8080') as session:
        step = 0
        while step < num_steps:
            _, step = sess.run([train_op, global_step])
```

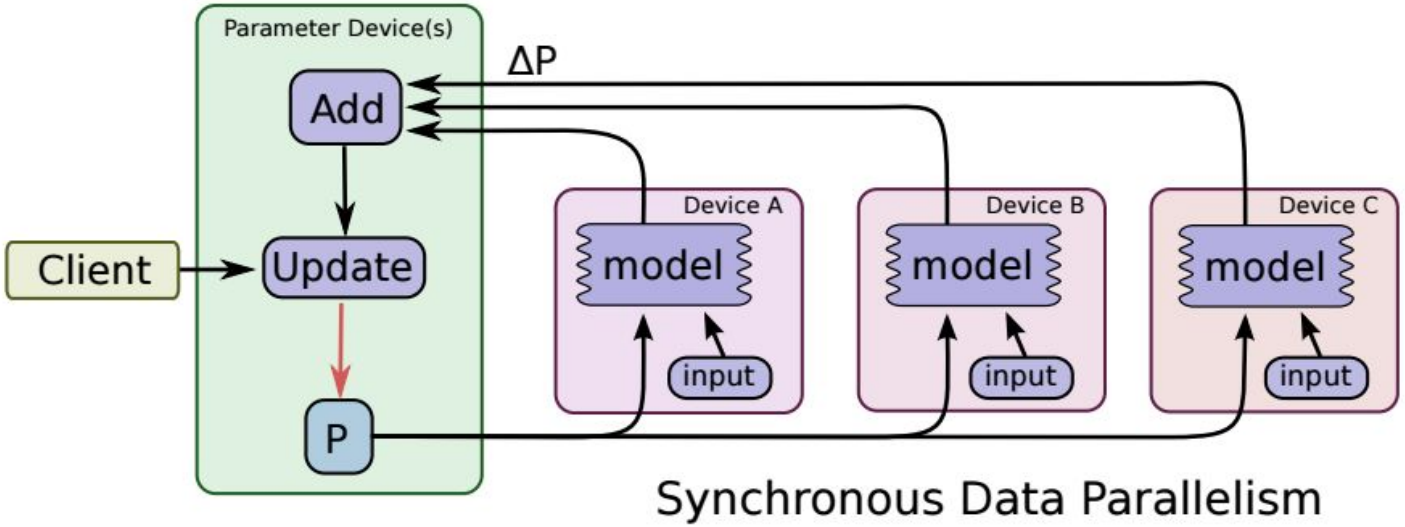
Data Parallelism: Asynchronous

```
train_op = tf.train.AdagradOptimizer(1.0, use_locking=False).minimize(
            loss, global_step=gs)
```



Data Parallelism: Synchronous

```
for worker in workers:
   with tf.device(worker):
      // expensive computation, e.g. loss
       losses.append(loss)
with tf.device(master):
   avg_loss = tf.add_n(losses) / len(workers)
   tf.train.AdagradOptimizer(1.0).minimize(avg_loss, global_step=gs)
```



Summary

Model Parallelism	
Sub-Graph	● Allows fine grained application of parallelism to slow graph components ● Larger more complex graph
Full Graph	● Code is more similar to single process models ● Not necessarily as performant (large models)

Data Parallelism	
Synchronous	● Prevents workers from “Falling behind” ● Workers progress at the speed of the slowest worker
Asynchronous	● Workers advance as fast as they can ● Can result in runs that aren’t reproducible or difficult to debug behavior (large models)

Demo

Related concepts / resources

- Distributed TensorFlow: http://bit.ly/tensorflow-k8s
- Kubernetes: http://bit.ly/k8s-for-users

Wrap up

Where to go for more

- TensorFlow whitepaper: http://bit.ly/tensorflow-wp
- Deep Learning Udacity course: http://bit.ly/udacity-tensorflow
- Deep MNIST for Experts (TensorFlow): http://bit.ly/expert-mnist
- Performing Image Recognition with TensorFlow: http://bit.ly/img-rec
- Neural Networks Demystified (video series): http://bit.ly/nn-demystified
- Gentle Guide to Machine Learning: http://bit.ly/gentle-ml
- TensorFlow tutorials: http://bit.ly/tensorflow-tutorials
- TensorFlow models: http://bit.ly/tensorflow-models

Thank you!

end