机器学习和深度学习

Machine Learning & Deep Learning

Machine Learning & Deep Learning

- 人工智能初览
- 初探机器学习
- 机器学习基础算法
- 机器学习进阶算法
- 机器学习实战项目
- 深度学习基础
- 深度学习进阶神经网络
- 深度学习核心卷积神经网络
- 深度学习网络架构
- 机器学习框架TensorFlow应用

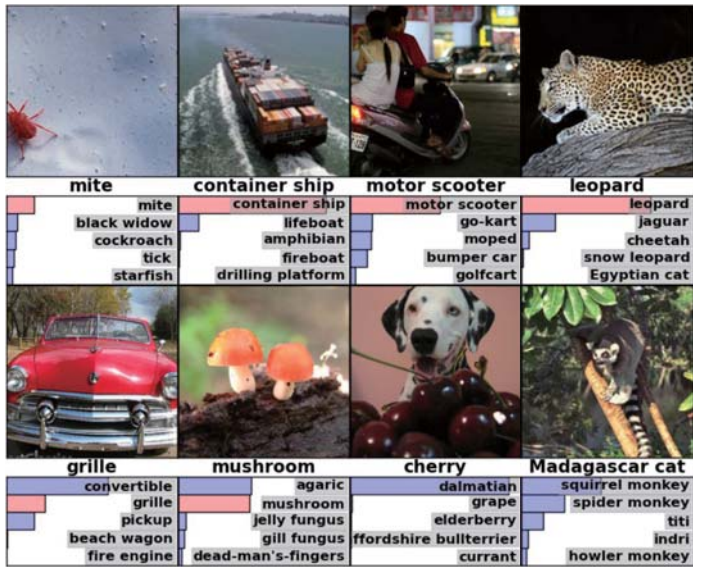
Gifts from Google and Line

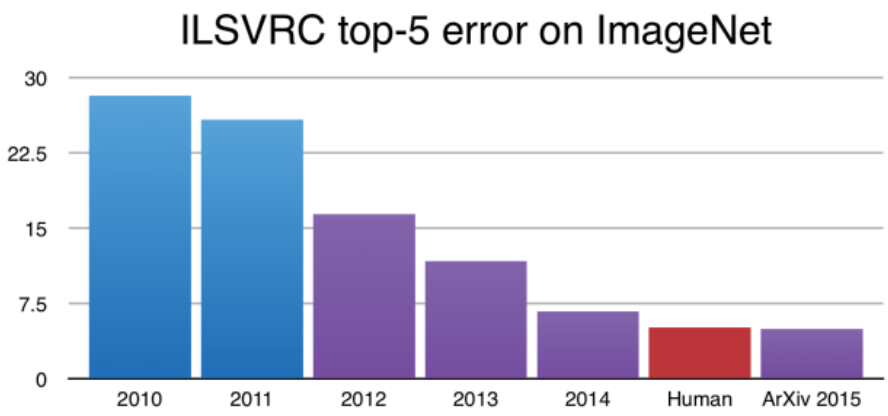



LINE Google

Shocking IT NEWS last years?
人工智能初览

Master (AI) vs. Ke Jie 9p





https://devblogs.nvidia.com/parallelforall/page/5/

Computer Vision is Hard:

Illumination Variability

References: [66]

Computer Vision is Hard:

Pose Variability and Occlusions

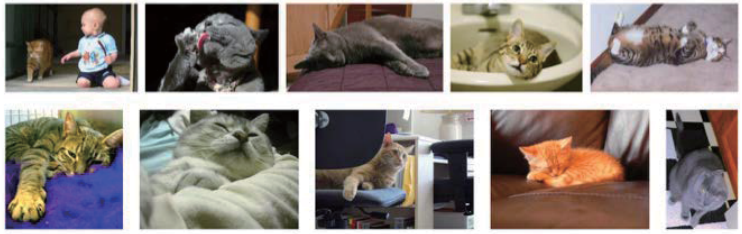


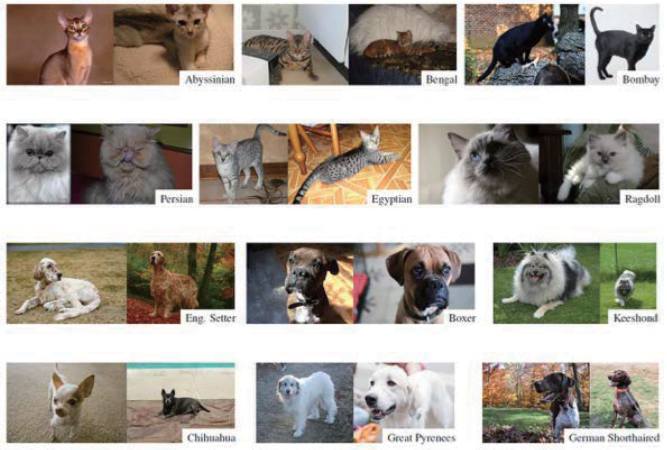
Figure 1. **The deformable and truncated cat.** Cats exhibit (al-

Parkhi et al. "The truth about cats and dogs." 2011.

References: [69]

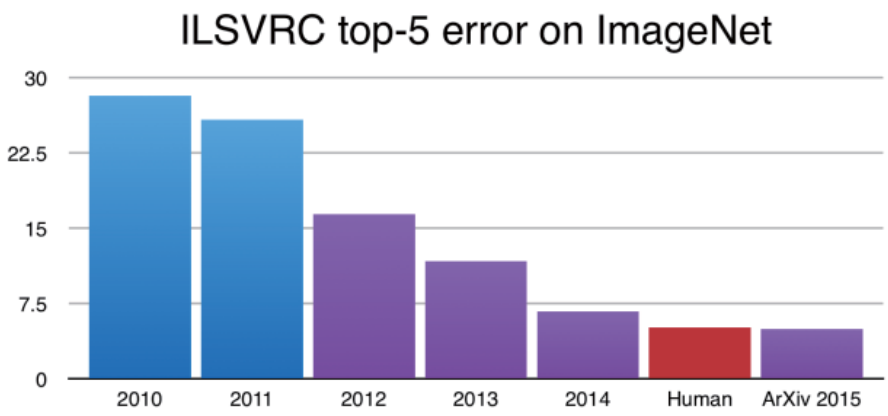
Computer Vision is Hard:

Intra-Class Variability

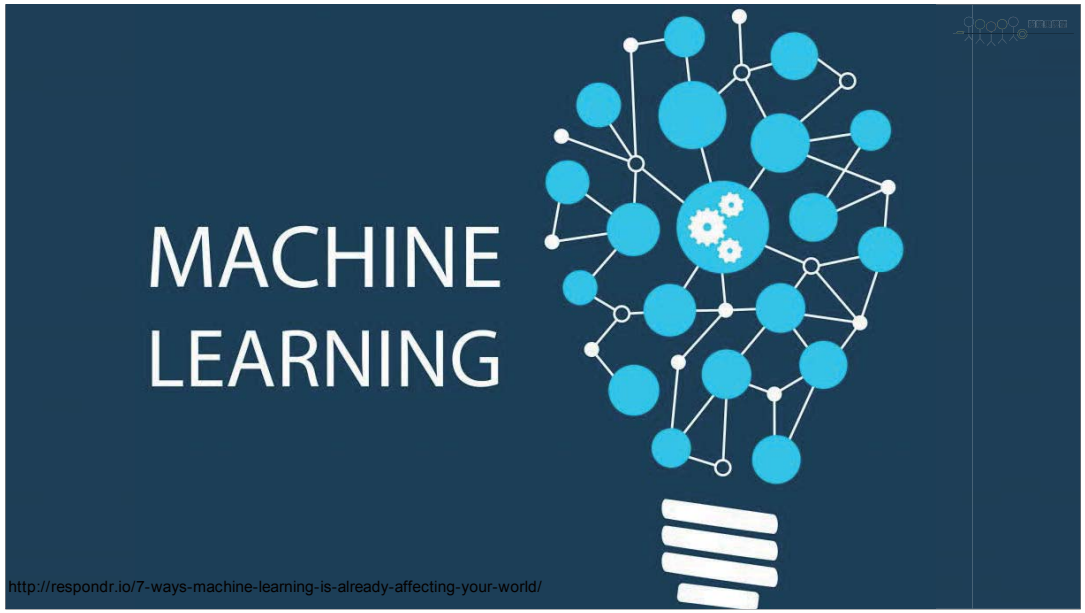


Parkhi et al. "Cats and dogs." 2012.

References: [70]



https://devblogs.nvidia.com/parallelforall/page/5/

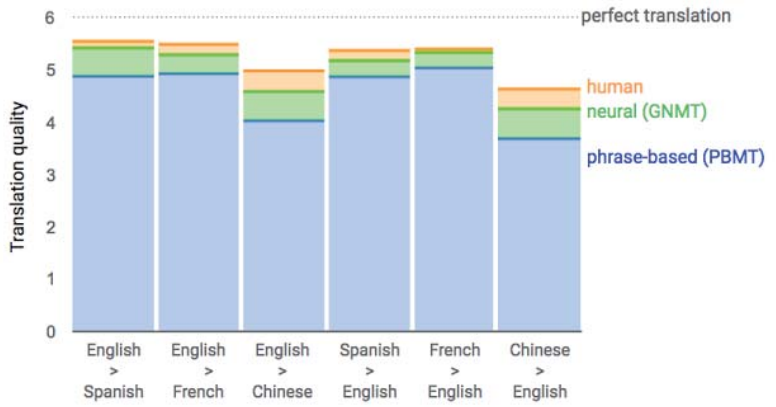


Machine Learning is Already Affecting Your World
初探机器学习

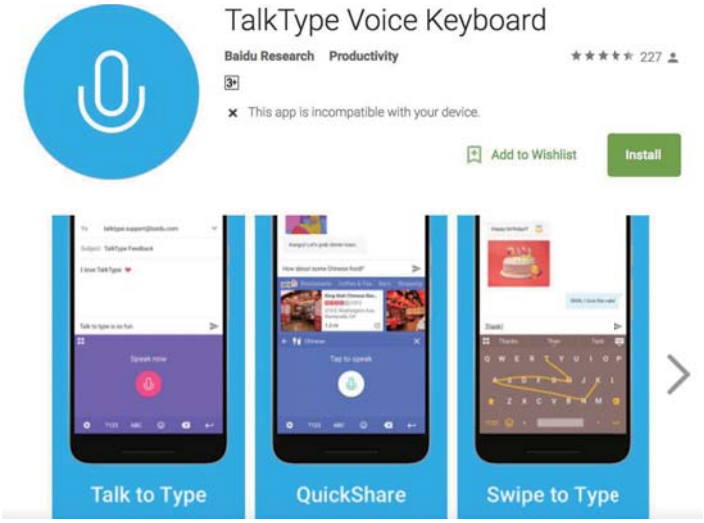
http://respondr.io/7-ways-machine-learning-is-already-affecting-your-world/

如果要建造一艘船，不要一起鼓勵人們收集
木材，
不要分配任務和工作，
而應該教他們漫長的海洋無限遠。

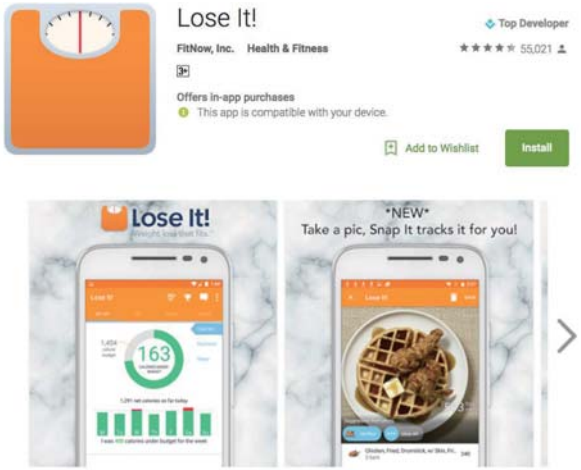
Google Neural Machine Translation (Sep 2016)



[Reference] "A Neural Network for Machine Translation, at Production Scale", https://research.googleblog.com/2016/09/a-neural-network-for-machine.html

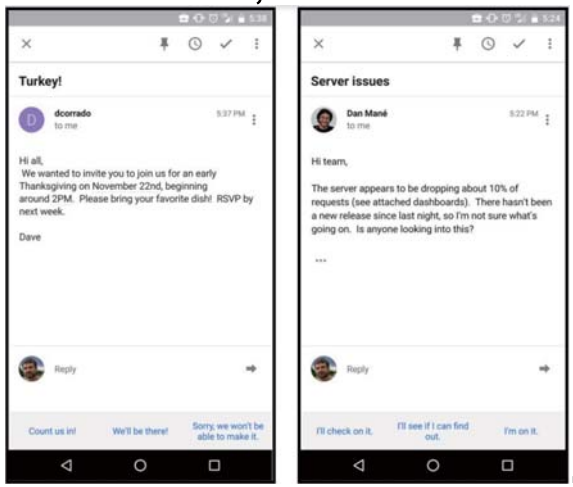


https://play.google.com/store/apps/details?id=com.baidu.research.talktype&hl=en

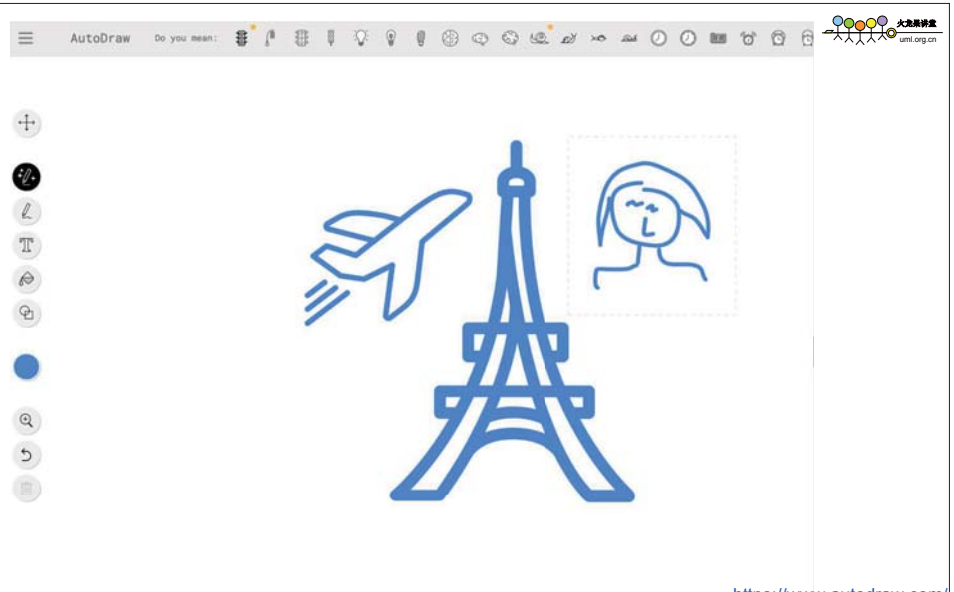


https://play.google.com/store/apps/details?id=com.fitnow.loseit&hl=en

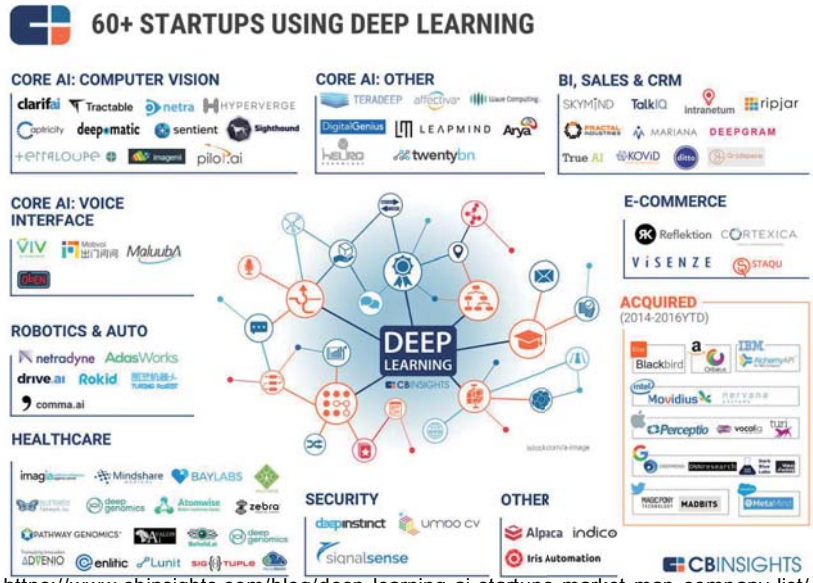
Smart Reply: Promising results
Seen 78%, Used 31.9%



http://www.kdd.org/kdd2016/papers/files/Paper_1069.pdf

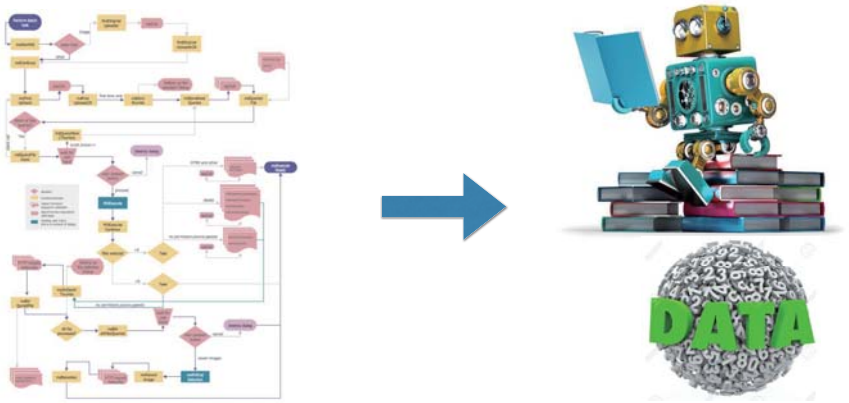


https://www.autodraw.com/



https://www.cbinsights.com/blog/deep-learning-ai-startups-market-map-company-list/

Rule based programming VS data driven learning



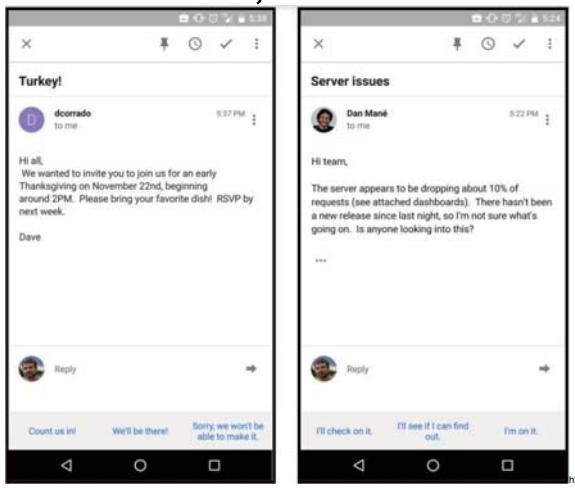
How can we develop?

Do you mean:



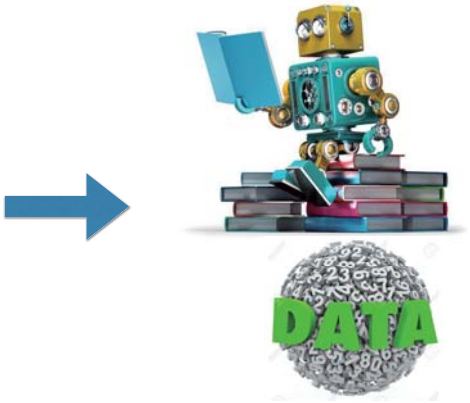
Smart Reply: Promising results

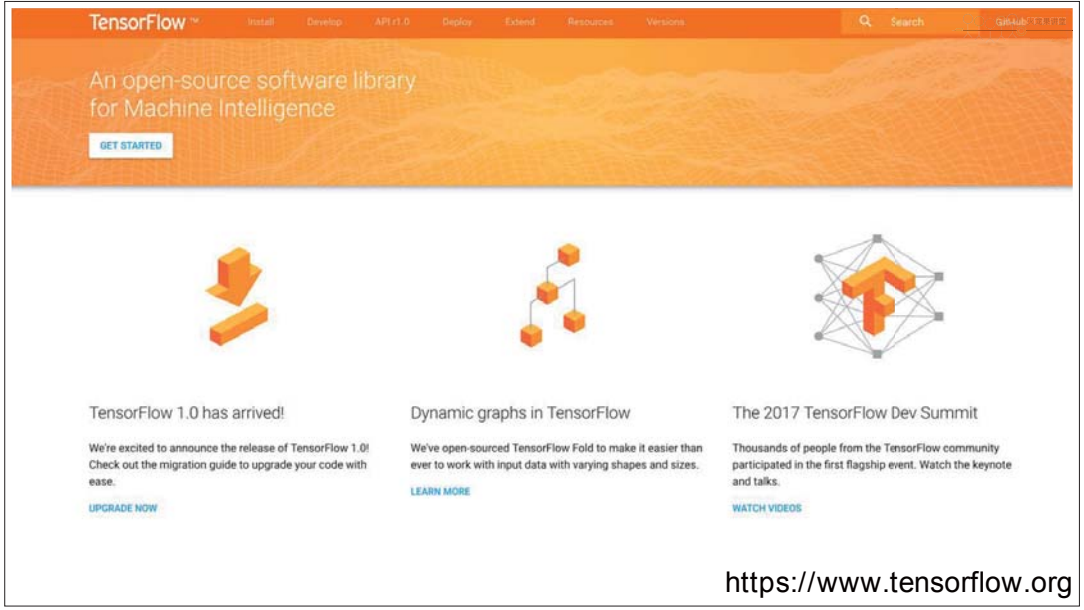
Seen 78%, Used 31.9%



http://www.kdd.org/kdd2016/papers/files/Paper_1069.pdf

Rule based programming VS data driven learning





https://www.tensorflow.org

TensorFlow

Deep learning libraries:
Accumulated GitHub metrics

Aggregate popularity (30•contrib + 10•issues + 5•forks)•1e-3	
#1: 172.29	tensorflow/tensorflow
#2: 89.78	BVLC/caffe
#3: 69.70	fchollet/keras
#4: 53.09	dmlc/mxnet
#5: 38.23	Theano/Theano
#6: 29.86	deeplearning4j/deeplearning4j
#7: 27.99	Microsoft/CNTK
#8: 17.36	torch/torch7
#9: 14.43	baidu/paddle
#10: 13.10	pfnet/chainer
#11: 12.37	NVIDIA/DIGITS
#12: 10.42	tflearn/tflearn
#13: 9.20	pytorch/pytorch

https://twitter.com/fchollet/status/830499993450450944/

Deep learning libraries: growth over past three months

new contributors from 2016-10-09 to 2017-02-10	
#1: 192	tensorflow/tensorflow
#2: 89	dmlc/mxnet
#3: 78	fchollet/keras
#4: 42	baidu/paddle
#5: 29	Microsoft/CNTK
#6: 23	pfnet/chainer
#7: 21	Theano/Theano
#8: 20	deeplearning4j/deeplearning4j
#9: 20	tflearn/tflearn
#10: 19	BVLC/caffe
#11: 9	torch/torch7
#12: 3	NVIDIA/DIGITS

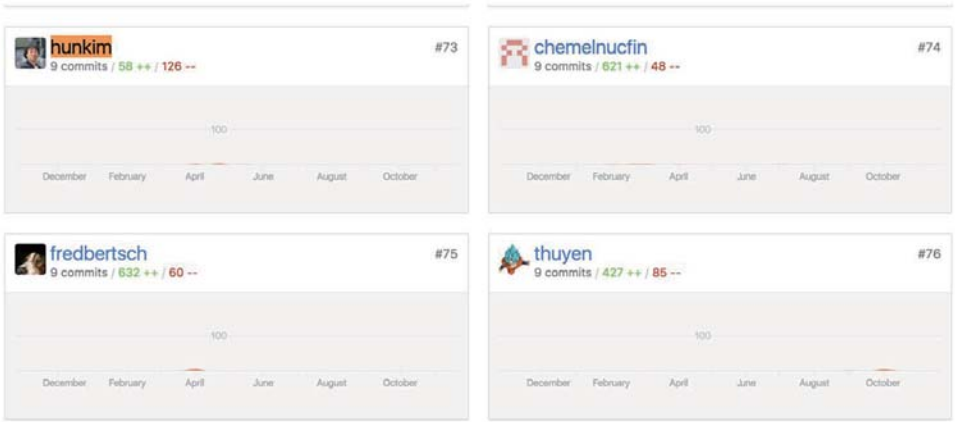
new forks from 2016-10-09 to 2017-02-10	
#1: 6525	tensorflow/tensorflow
#2: 1822	BVLC/caffe
#3: 1316	fchollet/keras
#4: 999	dmlc/mxnet
#5: 909	deeplearning4j/deeplearning4j
#6: 887	Microsoft/CNTK
#7: 324	tflearn/tflearn
#8: 321	baidu/paddle
#9: 287	Theano/Theano
#10: 257	torch/torch7
#11: 175	NVIDIA/DIGITS
#12: 142	pfnet/chainer

new issues from 2016-10-09 to 2017-02-10	
#1: 1563	tensorflow/tensorflow
#2: 979	fchollet/keras
#3: 871	dmlc/mxnet
#4: 646	baidu/paddle
#5: 486	Microsoft/CNTK
#6: 361	deeplearning4j/deeplearning4j
#7: 318	BVLC/caffe
#8: 217	NVIDIA/DIGITS
#9: 214	Theano/Theano
#10: 167	tflearn/tflearn
#11: 150	pfnet/chainer
#12: 90	torch/torch7

aggregate metrics growth from 2016-10-09 to 2017-02-10	
#1: 54.01	tensorflow/tensorflow
#2: 18.71	fchollet/keras
#3: 16.38	dmlc/mxnet
#4: 12.86	BVLC/caffe
#5: 10.17	Microsoft/CNTK
#6: 9.32	baidu/paddle
#7: 8.75	deeplearning4j/deeplearning4j
#8: 4.21	Theano/Theano
#9: 3.89	tflearn/tflearn
#10: 3.14	NVIDIA/DIGITS
#11: 2.90	pfnet/chainer
#12: 2.46	torch/torch7

François Chollet @fchollet · Feb 11
Time for an update: what does the deep learning library landscape look like, seen from GitHub? pic.twitter.com/QDZyvLrYBd

https://twitter.com/fchollet/status/830499993450450944/



TensorFlow
机器学习框架

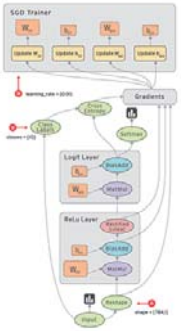
- TensorFlow™ is an open source software library for numerical computation using data flow graphs.
- Python

TensorFlow

https://www.tensorflow.org/

What is a Data Flow Graph?

- Nodes in the graph represent mathematical operations
- Edges represent the multidimensional data arrays (tensors) communicated between them.



https://www.tensorflow.org/

Installing TensorFlow

- Linux, Mac OSX, Windows
  - (sudo -H) pip install --upgrade tensorflow
  - (sudo -H) pip install --upgrade tensorflow-gpu
- From source
  - bazel ...
  - https://www.tensorflow.org/install/install_sources
- Google search/Community help

https://www.tensorflow.org/install/

Check installation and version

```
$ python3
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import tensorflow as tf
>>> tf.__version__
'1.0.0'
>>>
```

TensorFlow Hello World!

Hello TensorFlow!

In [2]:
```
# Create a constant op
# This op is added as a node to the default graph
hello = tf.constant("Hello, TensorFlow!")

# seart a TF session
sess = tf.Session()

# run the op and get result
print(sess.run(hello))
```

```
b'Hello, TensorFlow!'
```

b'String' '**b**' indicates *Bytes literals*. http://stackoverflow.com/questions/6269765/

lab-01-basics.ipynb

Computational Graph

In [4]:
```
node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
node3 = tf.add(node1, node2)
```

In [5]:
```
print("node1:", node1, "node2:", node2)
print("node3: ", node3)
```

```
node1: Tensor("Const_1:0", shape=(), dtype=float32) node2: Tensor("Const_2:0", shape=(), dtyp
e=float32)
node3:  Tensor("Add:0", shape=(), dtype=float32)
```

In [6]:
```
sess = tf.Session()
print("sess.run(node1, node2): ", sess.run([node1, node2]))
print("sess.run(node3): ", sess.run(node3))
```

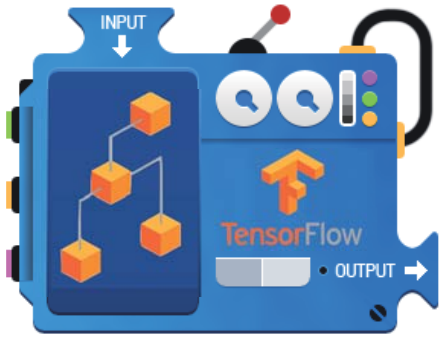
```
sess.run(node1, node2):  [3.0, 4.0]
sess.run(node3):  7.0
```

lab-01-basics.ipynb

TensorFlow Mechanics

2 feed data and run graph (operation)
sess.run (op)

1 Build graph using TensorFlow operations



3 update variables in the graph (and return values)

WWW.MATHWAREHOUSE.COM

Computational Graph

(1)Build graph (tensors) using TensorFlow operations

```
In [4]: node1 = tf.constant(3.0, tf.float32)
        node2 = tf.constant(4.0) # also tf.float32 implicitly
        node3 = tf.add(node1, node2)
```

(2) feed data and run graph (operation)
sess.run (op)

(3) update variables in the graph
(and return values)

```
In [6]: sess = tf.Session()
        print("sess.run(node1, node2): ", sess.run([node1, node2]))
        print("sess.run(node3): ", sess.run(node3))
```

```
sess.run(node1, node2):  [3.0, 4.0]
sess.run(node3):  7.0
```

lab-01-basics.ipynb

Placeholder

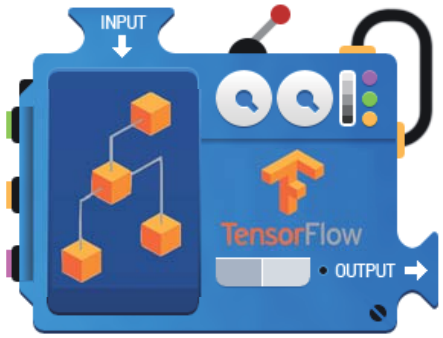
```
In [7]: a = tf.placeholder(tf.float32)
        b = tf.placeholder(tf.float32)
        adder_node = a + b  # + provides a shortcut for tf.add(a, b)

        print(sess.run(adder_node, feed_dict={a: 3, b: 4.5}))
        print(sess.run(adder_node, feed_dict={a: [1,3], b: [2, 4]}))
```

```
7.5
[ 3.  7.]
```

lab-01-basics.ipynb

TensorFlow Mechanics

2 feed data and run graph (operation)
sess.run (op, feed_dict={x: x_data})



1 Build graph using TensorFlow operations

3 update variables in the graph (and return values)

WWW.MATHWAREHOUSE.COM

Everything is **Tensor**

Tensors

```
In [3]: 3 # a rank 0 tensor; this is a scalar with shape []
        [1. ,2., 3.] # a rank 1 tensor; this is a vector with shape [3]
        [[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
        [[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
Out[3]: [[[1.0, 2.0, 3.0]], [[7.0, 8.0, 9.0]]]
```

```
t = tf.Constant([1., 2., 3.])
```

Tensor Ranks, Shapes, and Types

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Rank	Math entity	Python example
0	Scalar (magnitude only)	`s = 483`
1	Vector (magnitude and direction)	`v = [1.1, 2.2, 3.3]`
2	Matrix (table of numbers)	`m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	3-Tensor (cube of numbers)	`t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]`
n	n-Tensor (you get the idea)	`....`

https://www.tensorflow.org/programmers_guide/dims_types

Tensor Ranks, Shapes, and Types

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

https://www.tensorflow.org/programmers_guide/dims_types

Tensor Ranks, Shapes, and Types

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

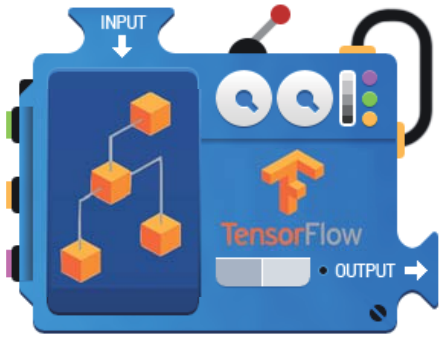
Data type	Python type	Description
`DT_FLOAT`	`tf.float32`	32 bits floating point.
`DT_DOUBLE`	`tf.float64`	64 bits floating point.
`DT_INT8`	`tf.int8`	8 bits signed integer.
`DT_INT16`	`tf.int16`	16 bits signed integer.
`DT_INT32`	`tf.int32`	32 bits signed integer.
`DT_INT64`	`tf.int64`	64 bits signed integer.

...

https://www.quora.com/When-should-I-use-tf-float32-vs-tf-float64-in-TensorFlow

TensorFlow Mechanics

2 feed data and run graph (operation)
sess.run (op, feed_dict={x: x_data})



1 Build graph using TensorFlow operations

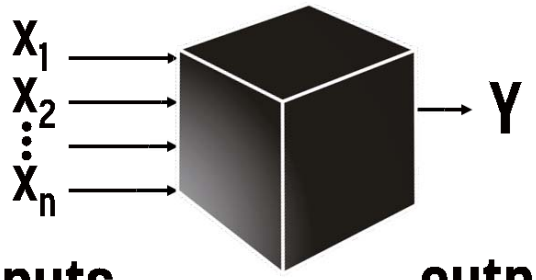
3 update variables in the graph (and return values)

WWW.MATHWAREHOUSE.COM

Machine Learning Basics
机器学习&深度学习
基础与进阶算法

- Linear Regression
- Logistic Regression (Binary classification)
- Softmax Classification
- Neural Networks

Regression



Predicting exam score: regression

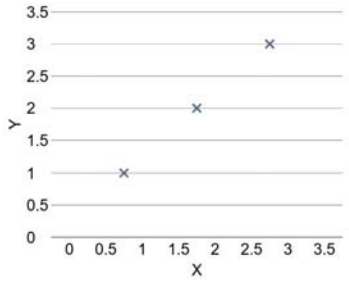
x (hours)	y (score)
10	90
9	80
3	50
2	30

Regression (data)

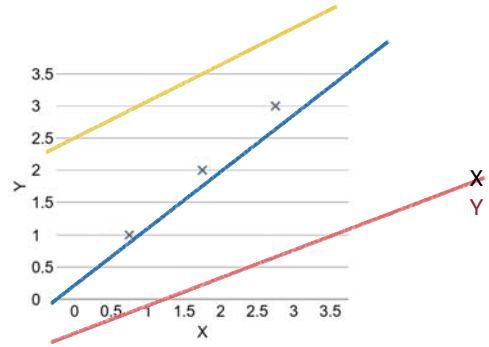
x	y
1	1
2	2
3	3

Regression (presentation)

x	Y
1	1
2	2
3	3



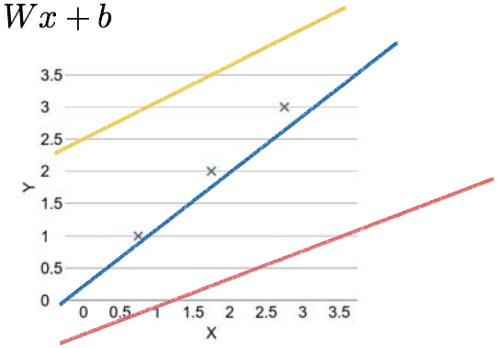
(Linear) Hypothesis



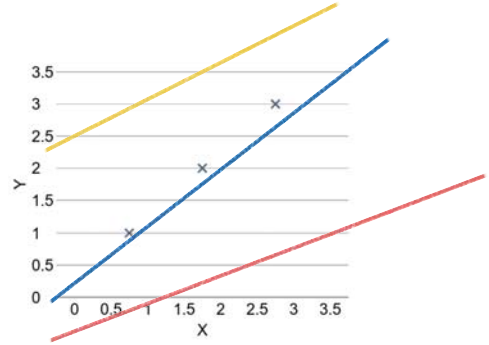
X = 2.5
Y = ?

(Linear) Hypothesis

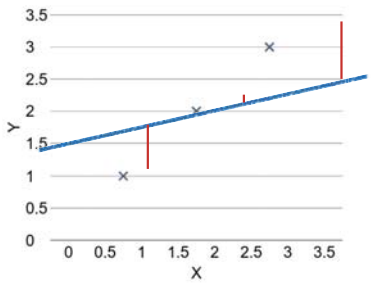
$$H(x) = Wx + b$$



Which hypothesis is better?



Which hypothesis is better?

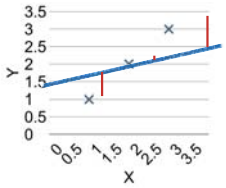


Cost function

- How fit the line to our (training) data

$$H(x) - y$$

$H(x) = Wx + b$

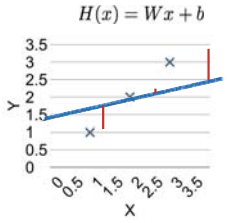


Cost function

- How fit the line to our (training) data

$$\frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(2)}) - y^{(2)})^2 + (H(x^{(3)}) - y^{(3)})^2}{3}$$

$H(x) = Wx + b$



$$cost = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

Cost function

$$cost = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

$$H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

Simplified hypothesis in TF

$$H(x) = Wx$$

```
y_model = tf.mul(X, w)
```

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

```
cost = tf.square(Y - y_model)
```

Goal: Minimize cost

$$\underset{W,b}{\text{minimize}}\, cost(W, b)$$

What *cost(W)* looks like?

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

- *W*=1, *cost(W)*=?

What *cost(W)* looks like?

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

- W=1, cost(W)=0

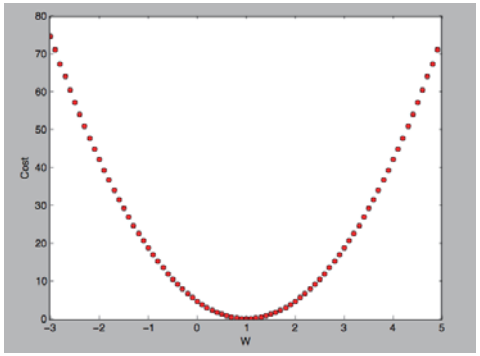
  $\frac{1}{3}((1*1-1)^2 + (1*2-2)^2 + (1*3-3)^2)$
- W=0, cost(W)=4.67

  $\frac{1}{3}((0*1-1)^2 + (0*2-2)^2 + (0*3-3)^2)$
- W=2, cost(W)=?

What *cost(W)* looks like?

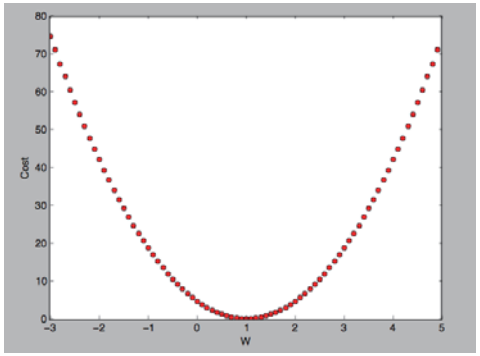
$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

- W=1, cost(W)=0
- W=0, cost(W)=4.67
- W=2, cost(W)=4.67



How to minimize cost?

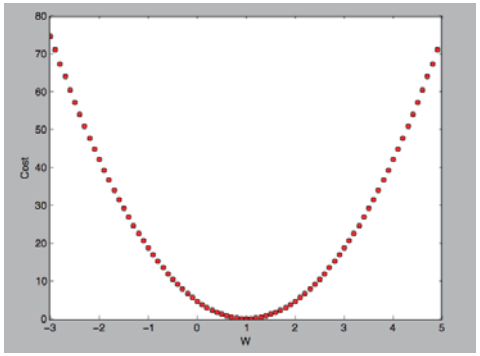
$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$



Formal definition

$$cost(W) = \frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$



Gradient descent algorithm in TF

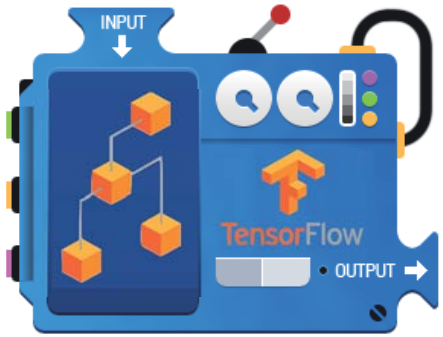
$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

$$W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})x^{(i)}$$

```
cost = tf.square(Y - y_model) # use square error for cost function

train_op = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
```

TensorFlow Mechanics

2 feed data and run graph (operation)
sess.run (op, feed_dict={x: x_data})



1 Build graph using TensorFlow operations

3 update variables in the graph (and return values)

WWW.MATHWAREHOUSE.COM

1 Build graph using TF operations

$$H(x) = Wx + b$$

```
# X and Y data
x_train = [1, 2, 3]
y_train = [1, 2, 3]

W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
# Our hypothesis XW+b
hypothesis = x_train * W + b
```

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

```
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - y_train))
```

1 Build graph using TF operations

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

```
t = [1., 2. ,3. ,4.]
tf.reduce_mean(t) ==> 2.5
```

```
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - y_train))
```

GradientDescent

```
# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)
```

https://www.tensorflow.org/api_docs/python/tf/reduce_mean

2 3 Run/update graph and get results

```
# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

# Fit the line
for step in range(2001):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(cost), sess.run(W), sess.run(b))
```

Full code (less than 20 lines)

```
import tensorflow as tf

# X and Y data
x_train = [1, 2, 3]
y_train = [1, 2, 3]

W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Our hypothesis XW+b
hypothesis = x_train * W + b

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - y_train))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

# Fit the line
for step in range(2001):
   sess.run(train)
   if step % 20 == 0:
       print(step, sess.run(cost), sess.run(W), sess.run(b))
```

```
'''
0 2.82329 [ 2.12867713] [-0.85235667]
20 0.190351 [ 1.53392804] [-1.05059612]
40 0.151357 [ 1.45725465] [-1.02391243]
...

1920 1.77484e-05 [ 1.00489295] [-0.01112291]
1940 1.61197e-05 [ 1.00466311] [-0.01060018]
1960 1.46397e-05 [ 1.004444] [-0.01010205]
1980 1.32962e-05 [ 1.00423515] [-0.00962736]
2000 1.20761e-05 [ 1.00403607] [-0.00917497]
'''
```

Placeholders

```
In [7]: a = tf.placeholder(tf.float32)
        b = tf.placeholder(tf.float32)
        adder_node = a + b  # + provides a shortcut for tf.add(a, b)

        print(sess.run(adder_node, feed_dict={a: 3, b: 4.5}))
        print(sess.run(adder_node, feed_dict={a: [1,3], b: [2, 4]}))
```

```
7.5
[ 3.  7.]
```

lab-01-basics.ipynb

Placeholders

```
# X and Y data
x_train = [1, 2, 3]
y_train = [1, 2, 3]

# Now we can use X and Y in place of x_data and y_data
# # placeholders for a tensor that will be always fed using feed_dict
# See http://stackoverflow.com/questions/36693740/
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
...
# Fit the line
for step in range(2001):
   cost_val, W_val, b_val, _ = \
       sess.run([cost, W, b, train],
                feed_dict={X: [1, 2, 3], Y: [1, 2, 3]})
   if step % 20 == 0:
       print(step, cost_val, W_val, b_val)
```

Full code with placeholders

```
import tensorflow as tf
W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

X = tf.placeholder(tf.float32, shape=[None])
Y = tf.placeholder(tf.float32, shape=[None])

# Our hypothesis XW+b
hypothesis = X * W + b
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

# Fit the line
for step in range(2001):
   cost_val, W_val, b_val, _ = sess.run([cost, W, b, train],
                feed_dict={X: [1, 2, 3], Y: [1, 2, 3]})
   if step % 20 == 0:
       print(step, cost_val, W_val, b_val)
```

```
...
1980 1.32962e-05 [ 1.00423515] [-0.00962736]
2000 1.20761e-05 [ 1.00403607] [-0.00917497]


# Testing our model
print(sess.run(hypothesis, feed_dict={X: [5]}))
print(sess.run(hypothesis, feed_dict={X: [2.5]}))
print(sess.run(hypothesis,
               feed_dict={X: [1.5, 3.5]}))


[ 5.0110054]
[ 2.50091505]
[1.49687922 3.50495124]
```

Full code with placeholders

```
import tensorflow as tf
W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
X = tf.placeholder(tf.float32, shape=[None])
Y = tf.placeholder(tf.float32, shape=[None])

# Our hypothesis XW+b
hypothesis = X * W + b
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

# Fit the line with new training data
for step in range(2001):
    cost_val, W_val, b_val, _ = sess.run([cost, W, b, train],
         feed_dict={X: [1, 2, 3, 4, 5],
                    Y: [2.1, 3.1, 4.1, 5.1, 6.1]})
    if step % 20 == 0:
        print(step, cost_val, W_val, b_val)
```

```
1960 3.32396e-07 [ 1.00037301] [ 1.09865296]
1980 2.90429e-07 [ 1.00034881] [ 1.09874094]
2000 2.5373e-07 [ 1.00032604] [ 1.09882331]

# Testing our model
print(sess.run(hypothesis, feed_dict={X: [5]}))
print(sess.run(hypothesis, feed_dict={X: [2.5]}))
print(sess.run(hypothesis,
               feed_dict={X: [1.5, 3.5]}))


[ 6.10045338]
[ 3.59963846]
[ 2.59931231  4.59996414]
```

TensorFlow Mechanics

2 feed data and run graph (operation)
sess.run (op, feed_dict={x: x_data})

```
feed_dict={X: [1, 2, 3, 4, 5],
 Y: [2.1, 3.1, 4.1, 5.1, 6.1]})
```

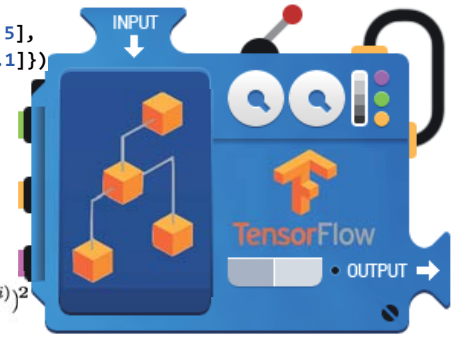
1 Build graph using TensorFlow operations

$$H(x) = Wx + b$$

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

…

```
train =
optimizer.minimize(cost)
```



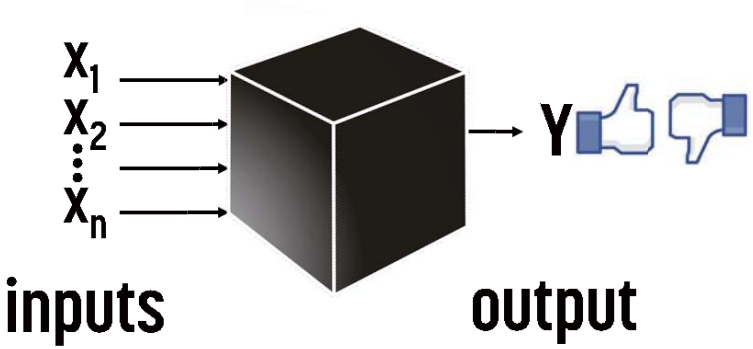
3 update variables in the graph (and return values)

WWW.MATHWAREHOUSE.COM

Machine Learning Basics

- Linear Regression
- Logistic Regression (Binary classification)
- Softmax Classification
- Neural Networks

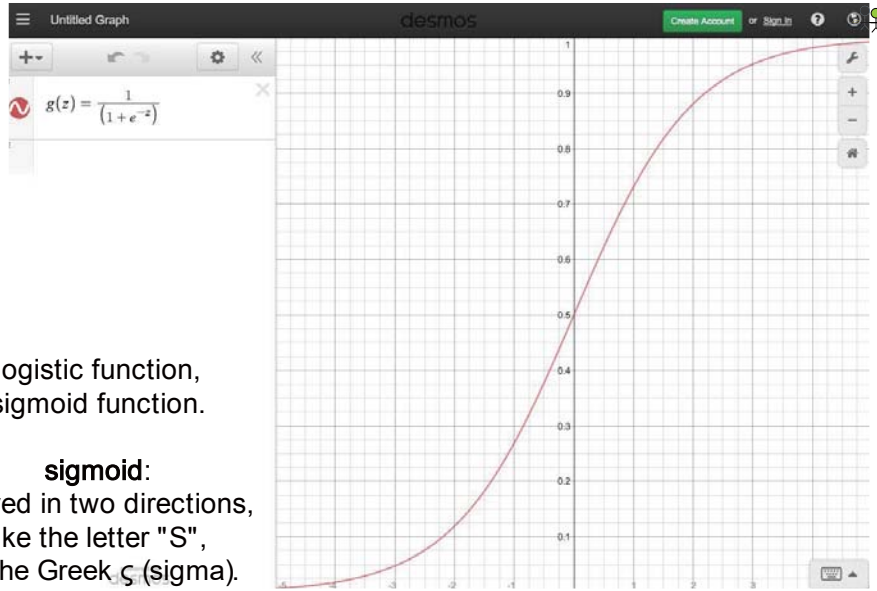
Classification



Linear Regression Hypothesis

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

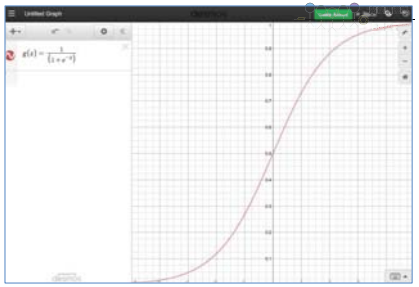


logistic function,
sigmoid function.

sigmoid:
Curved in two directions,
like the letter "S",
or the Greek ς (sigma).

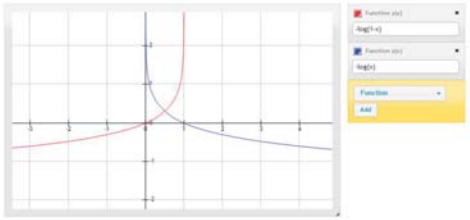
Logistic Regression

$$H(X) = sigmoid(XW) = \frac{1}{1+e^{-XW}}$$



$$cost(W) = -\frac{1}{m}\sum \underbrace{ylog(H(x))}_{\text{Case for y=1}} + \underbrace{(1-y)(log(1-H(x))}_{\text{Case for y=0}}$$

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$



Logistic Classification in TF

$$H(X) = sigmoid(XW) = \frac{1}{1+e^{-XW}}$$

```
hypothesis =
tf.sigmoid(tf.matmul(X, W) + b)
```

$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1-y)(log(1-H(x))$$

```
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))
```

Training Data

```
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
```

lab-05-1-logistic_regression.py

```
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')


# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W) + b))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
```

$$H(X) = \frac{1}{1+e^{-W^T X}}$$

$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1-y)(log(1-H(x))$$

```
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))


train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

```
# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

lab-05-1-logistic_regression.py

Train the model

```
# Launch graph
with tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())

   for step in range(10001):
       cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
       if step % 200 == 0:
           print(step, cost_val)

   # Accuracy report
   h, c, a = sess.run([hypothesis, predicted, accuracy],
                      feed_dict={X: x_data, Y: y_data})
   print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)
```

lab-05-1-logistic_regression.py

```
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]
# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())

   for step in range(10001):
       cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
       if step % 200 == 0:
           print(step, cost_val)

   # Accuracy report
   h, c, a = sess.run([hypothesis, predicted, accuracy],
                      feed_dict={X: x_data, Y: y_data})
   print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)
```

lab-05-1-logistic_regression.py

```
# step, cost
0 1.73078
200 0.571512
400 0.507414
...
9600 0.154132
9800 0.151778
10000 0.149496


Hypothesis:
[[ 0.03074029]
 [ 0.15884677]
 [ 0.30486736]
 [ 0.78138196]
 [ 0.93957496]
 [ 0.98016882]]

Correct (Y):
[[ 0.]
 [ 0.]
 [ 0.]
 [ 1.]
 [ 1.]
 [ 1.]]

Accuracy:  1.0
```

Classifying diabetes
机器学习实战项目

-0.411765	0.165829	0.213115	0	0	-0.23696	-0.894962	-0.7	1
-0.647059	-0.21608	-0.180328	-0.353535	-0.791962	-0.0760059	-0.854825	-0.833333	0
0.176471	0.155779	0	0	0	0.052161	-0.952178	-0.733333	1
-0.764706	0.979899	0.147541	-0.0909091	0.283688	-0.0909091	-0.931682	0.0666667	0
-0.0588235	0.256281	0.57377	0	0	0	-0.868488	0.1	0
-0.529412	0.105528	0.508197	0	0	0.120715	-0.903501	-0.7	1
0.176471	0.688442	0.213115	0	0	0.132638	-0.608027	-0.566667	0
0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1

```
xy = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

lab-05-2-logistic_regression_diabetes.py

```
xy = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 8])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([8, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    feed = {X: x_data, Y: y_data}
    for step in range(10001):
        sess.run(train, feed_dict=feed)
        if step % 200 == 0:
            print(step, sess.run(cost, feed_dict=feed))

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict=feed)
    print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)
```

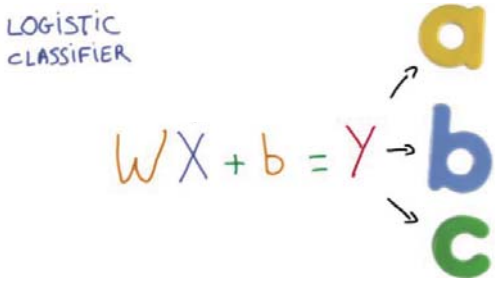
lab-05-2-logistic_regression_diabetes.py

```
0 0.82794
200 0.755181
400 0.726355
600 0.705179
800 0.686631
...
9600 0.492056
9800 0.491396
10000 0.490767


 [ 0.7461012 ]
 [ 0.79919308]
 [ 0.72995949]
 [ 0.88297188]]

[ 1.]
[ 1.]
[ 1.]]
Accuracy:
0.762846
```

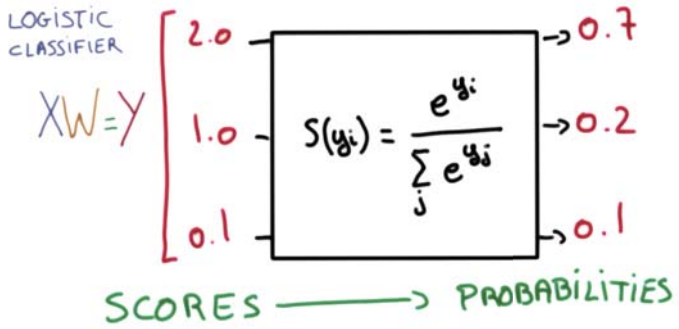
Machine Learning Basics

- Linear Regression
- Logistic Regression (Binary classification)
- Softmax Classification
- Neural Networks

Multiple labels (a, b, c)



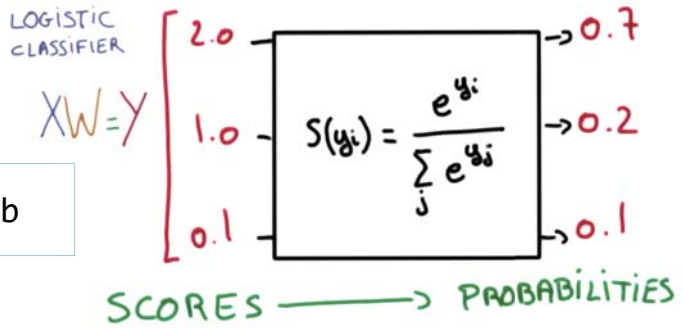
Softmax function



https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

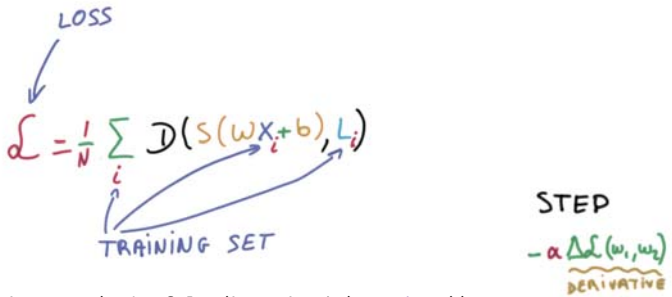
```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

```
tf.matmul(X,W)+b
```



https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

Cost function: cross entropy




```
# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

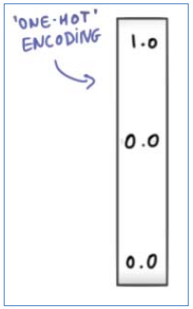
https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

Cost function: cross entropy

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)

# Cross entropy cost/loss
cost = tf.reduce_mean
       (-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).
           minimize(cost)
```

https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

```
x_data = [[1, 2, 1, 1], [2, 1, 3, 2], [3, 1, 3, 4], [4, 1, 5, 5], [1, 7, 5, 5],
                                                   [1, 2, 5, 6], [1, 6, 6, 6], [1, 7, 7, 7]]
y_data = [[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 1, 0], [0, 1, 0], [1, 0, 0], [1, 0, 0]]

X = tf.placeholder("float", [None, 4])
Y = tf.placeholder("float", [None, 3])
nb_classes = 3

W = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Launch graph
with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())

   for step in range(2001):
       sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
       if step % 200 == 0:
           print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}))
```

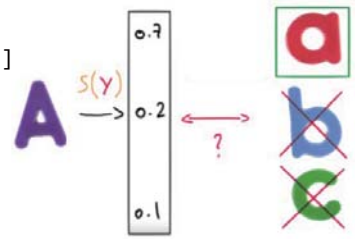


lab-06-1-softmax_classifier.py

Test & one-hot encoding

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)

# Testing & One-hot encoding
a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]
print(a, sess.run(tf.arg_max(a, 1)))
```

```
[[ 1.38904958e-03  9.98601854e-01  9.06129117e-06]] [1]
```



lab-06-1-softmax_classifier.py

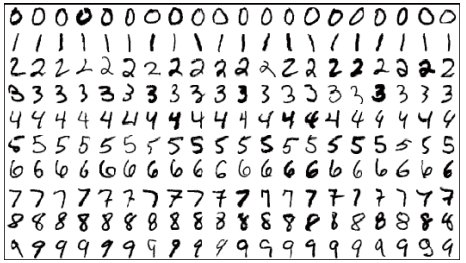
Test & one-hot encoding

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)

all = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9],
                                          [1, 3, 4, 3],
                                          [1, 1, 0, 1]]})
print(all, sess.run(tf.arg_max(all, 1)))
```

```
[[ 1.38904958e-03  9.98601854e-01  9.06129117e-06]
 [ 9.31192040e-01  6.29020557e-02  5.90589503e-03]
 [ 1.27327668e-08  3.34112905e-04  9.99665856e-01]]

[1 0 2]
```

lab-06-1-softmax_classifier.py

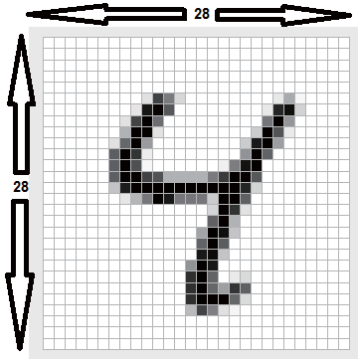
MNIST Dataset 机器学习实战项目



```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

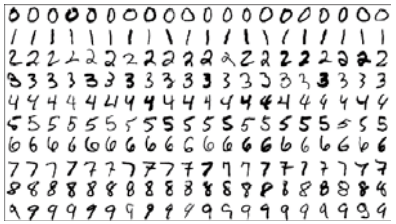
http://yann.lecun.com/exdb/mnist/

28x28x1 image




```
# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])
```

http://derindelimavi.blogspot.hk/2015/04/mnist-el-yazs-rakam-veri-seti.html



MNIST Dataset

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
…
batch_xs, batch_ys = mnist.train.next_batch(100)
…
print("Accuracy: ", accuracy.eval(session=sess,
                                  feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

lab-07-4-mnist_introduction.py

Reading data and set variables

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))
```

lab-07-4-mnist_introduction.py

Softmax!

```
# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

lab-07-4-mnist_introduction.py

Training epoch/batch

```
# parameters
training_epochs = 15
batch_size = 100

with tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())
   # Training cycle
   for epoch in range(training_epochs):
       avg_cost = 0
       total_batch = int(mnist.train.num_examples / batch_size)

       for i in range(total_batch):
           batch_xs, batch_ys = mnist.train.next_batch(batch_size)
           c, _ = sess.run([cost, optimizer], feed_dict={X: batch_xs, Y: batch_ys})
           avg_cost += c / total_batch

       print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
```

lab-07-4-mnist_introduction.py

Training epoch/batch

In the neural network terminology:

one **epoch** = one forward pass and one backward pass of *all* the training examples

batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.

number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

http://stackoverflow.com/questions/4752626/epoch-vs-iteration-when-training-neural-networks

Training epoch/batch

```
# parameters
training_epochs = 15
batch_size = 100

with tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())
   # Training cycle
   for epoch in range(training_epochs):
       avg_cost = 0
       total_batch = int(mnist.train.num_examples / batch_size)

       for i in range(total_batch):
           batch_xs, batch_ys = mnist.train.next_batch(batch_size)
           c, _ = sess.run([cost, optimizer], feed_dict={X: batch_xs, Y: batch_ys})
           avg_cost += c / total_batch

       print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
```

lab-07-4-mnist_introduction.py

Report results on test dataset

```
# Test the model using test sets
print("Accuracy: ", accuracy.eval(session=sess,
      feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

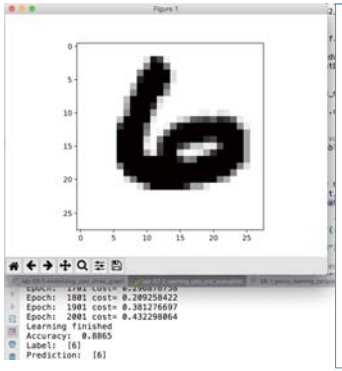
lab-07-4-mnist_introduction.py

```
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost

is_correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
training_epochs = 15
batch_size = 100

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0
        total_batch = int(mnist.train.num_examples / batch_size)

        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            c, _ = sess.run([cost, optimizer],
                            feed_dict={X: batch_xs, Y: batch_ys})
            avg_cost += c / total_batch

        print('Epoch:', '%04d' % (epoch + 1),
                            'cost =', '{:.9f}'.format(avg_cost))
```

lab-07-4-mnist_introduction.py

```
Epoch: 0001 cost = 2.868104637
Epoch: 0002 cost = 1.134684615
Epoch: 0003 cost = 0.908220728
Epoch: 0004 cost = 0.794199896
Epoch: 0005 cost = 0.721815854
Epoch: 0006 cost = 0.670184430
Epoch: 0007 cost = 0.630576546
Epoch: 0008 cost = 0.598888191
Epoch: 0009 cost = 0.573027079
Epoch: 0010 cost = 0.550497213
Epoch: 0011 cost = 0.532001859
Epoch: 0012 cost = 0.515517795
Epoch: 0013 cost = 0.501175288
Epoch: 0014 cost =
```

Sample image show and prediction



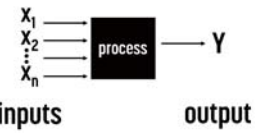
```
import matplotlib.pyplot as plt
import random

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label:", sess.run(tf.argmax(mnist.test.labels[r:r+1], 1)))
print("Prediction:", sess.run(tf.argmax(hypothesis, 1),
                              feed_dict={X: mnist.test.images[r:r + 1]}))

plt.imshow(mnist.test.images[r:r + 1].
          reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

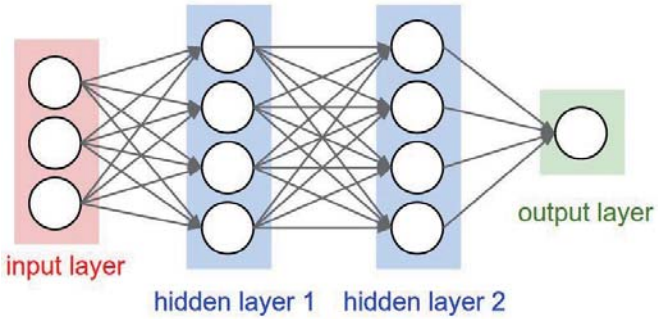
lab-07-4-mnist_introduction.py

Machine Learning Basics

- Linear Regression
- Logistic Regression (Binary classification)
- Softmax Classification
- Neural Networks

Neural Net
深度学习进阶
神经网络





Softmax classifier for MNIST

```
# weights & bias for nn layers
W = tf.Variable(tf.random_normal([784, 10]))
b = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(X, W) + b
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

```
cost = tf.reduce_mean(-
tf.reduce_sum(Y *
tf.log(hypothesis), axis=1))
```

```
Epoch: 0001 cost = 5.888845987
Epoch: 0002 cost = 1.860620173
Epoch: 0003 cost = 1.159035648
Epoch: 0004 cost = 0.892340870
Epoch: 0005 cost = 0.751155428
Epoch: 0006 cost = 0.662484806
Epoch: 0007 cost = 0.601544010
Epoch: 0008 cost = 0.556526115
Epoch: 0009 cost = 0.521186961
Epoch: 0010 cost = 0.493068354
Epoch: 0011 cost = 0.469686249
Epoch: 0012 cost = 0.449967254
Epoch: 0013 cost = 0.433519321
Epoch: 0014 cost = 0.419000337
Epoch: 0015 cost = 0.406490815
Learning Finished!
Accuracy: 0.9035
```

lab-10-1-mnist_softmax.py

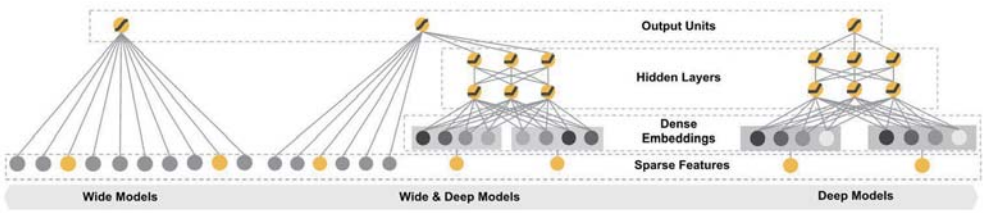
NN for MNIST

```
# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
   logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
Epoch: 0001 cost = 141.207671860
Epoch: 0002 cost = 38.788445864
Epoch: 0003 cost = 23.977515479
Epoch: 0004 cost = 16.315132428
Epoch: 0005 cost = 11.702554882
Epoch: 0006 cost = 8.573139748
Epoch: 0007 cost = 6.370995680
Epoch: 0008 cost = 4.537178684
Epoch: 0009 cost = 3.216900532
Epoch: 0010 cost = 2.329708954
Epoch: 0011 cost = 1.715552875
Epoch: 0012 cost = 1.189857912
Epoch: 0013 cost = 0.820965160
Epoch: 0014 cost = 0.624131458
Epoch: 0015 cost = 0.454633765
Learning Finished!
Accuracy: 0.9455
```

lab-10-2-mnist_nn.py

Wide & Deep



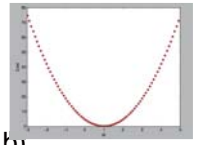
https://www.tensorflow.org/versions/r0.11/tutorials/wide_and_deep/index.html

Homework

- Tensor manipulation
  - https://docs.google.com/presentation/d/1gQ7XxrhylKr5Kk5pG15yvX3yOIn_hk2-H6jrQeXqKmU/
- Titanic with NN 改进泰坦尼克获救预测
  - k0-01-titanic

- CNN
- RNN
- Example code: DeepLearningZeroToAll

How to minimize cost?



- Start with initial guesses
  - Start at 0,0 (or any other value)
  - Keeping changing W and b a little bit to try and reduce cost(W, b)
- Each time you change the parameters, you select the gradient which reduces cost(W, b) the most possible
- Repeat
- Do so until you converge to a local minimum
- Has an interesting property
  - Where you start can determine which minimum you end up

http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html

Formal definition

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

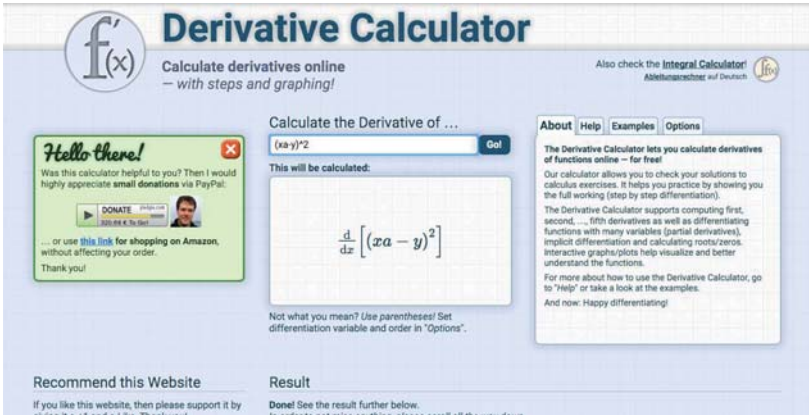
$$cost(W) = \frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

Formal definition

$$W := W - \alpha \frac{\partial}{\partial W} \frac{1}{2m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^{m} 2(Wx^{(i)} - y^{(i)}) x^{(i)}$$

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)}) x^{(i)}$$



Gradient descent algorithm

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)}) x^{(i)}$$

CNN

深度学习核心卷积神经网络

Gifts from Google and Line

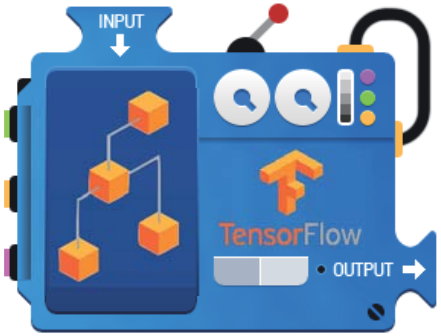



LINE Google

TensorFlow Mechanics

2 feed data and run graph (operation)
sess.run (op, feed_dict={x: x_data})

1 Build graph using TensorFlow operations



3 update variables in the graph (and return values)

WWW.MATHWAREHOUSE.COM

Machine Learning Basics

- Linear Regression
- Logistic Regression (Binary classification)
- Softmax Classification
- Neural Networks

Linear Regression

1 $H(x) = Wx + b$

```
W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
# Our hypothesis XW+b
hypothesis = X * W + b
```

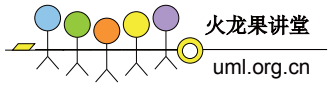
2 $cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$

```
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - y_train))
```

3 GradientDescent $W := W - \alpha\frac{\partial}{\partial W}cost(W)$

```
# Minimize
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

Logistic Regression



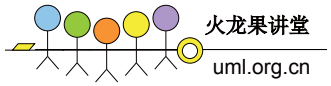
1. Model? (Hypothesis?)

2. Cost?

3. GradientDescent $W := W - \alpha \frac{\partial}{\partial W} cost(W)$

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

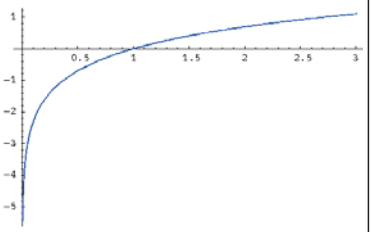
Logistic Regression



1. Model? (Hypothesis?)

$$H(X) = sigmoid(XW) = \frac{1}{1+e^{-XW}}$$

```
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
```



2. Cost?

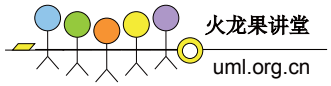
$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1-y)(log(1-H(x))$$

```
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))
```

3. GradientDescent $W := W - \alpha \frac{\partial}{\partial W} cost(W)$

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

Softmax Classifier



1. Model? (Hypothesis?)

```
logits = tf.matmul(X,W)+b
hypothesis = tf.nn.softmax(logits)
```

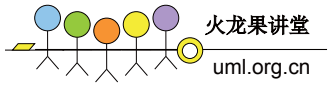
2. Cost?

```
# Cross entropy cost/loss
cost = tf.reduce_mean (-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

3. GradientDescent $W := W - \alpha \frac{\partial}{\partial W} cost(W)$

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

Softmax Classifier



1. Model? (Hypothesis?)

```
logits = tf.matmul(X,W)+b
hypothesis = tf.nn.softmax(logits)
```

2. Cost?

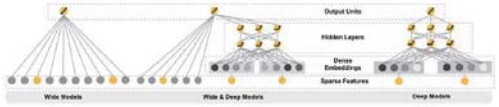
```
# Cross entropy cost/loss
cost = tf.reduce_mean (-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

cost = tf.reduce_mean(
     tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=Y))
```

3. GradientDescent $W := W - \alpha \frac{\partial}{\partial W} cost(W)$

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

(Deep) Neural Nets


```
# input placeholders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
   logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```



NN tips

- Initilizing weights
- Activation functions
- Regularization
- Optimizers

Random?


```
W = tf.Variable(tf.random_normal([1]), name='weight')
```

How to do Xavier initialization on TensorFlow

32

I'm porting my Caffe network over to TensorFlow but it doesn't seem to have xavier initialization. I'm using `truncated_normal` but this seems to be making it a lot harder to train.

python tensorflow

share edit flag

edited Dec 19 '15 at 22:25
Hooked
32.7k • 13 • 108 • 169

asked Nov 10 '15 at 22:07
Aleph7
2,661 • 1 • 14 • 24

10

56

Since version 0.8 there is a Xavier initializer, see here for the docs.

You can use something like this:

```
W = tf.get_variable("W", shape=[784, 256],
           initializer=tf.contrib.layers.xavier_initializer())
```

share edit delete flag

edited Dec 1 '16 at 16:32
fabian789
5,351 • 3 • 32 • 75

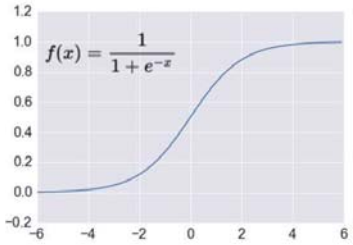
answered Apr 22 '16 at 4:23
Sung Kim
2,451 • 1 • 16 • 24

http://stackoverflow.com/questions/33640581/

NN tips

- Initilizing weights
- Activation functions
- Regularization
- Optimizers

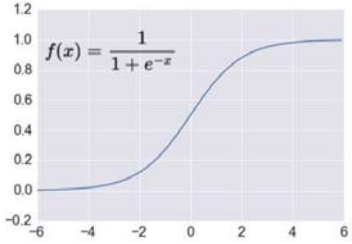
Activation functions



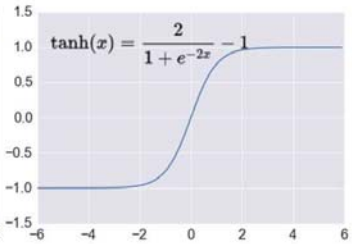
http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

Activation functions

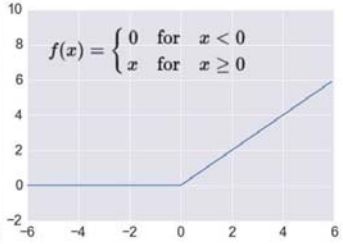
tf.sigmoid



tf.tanh



tf.nn.relu

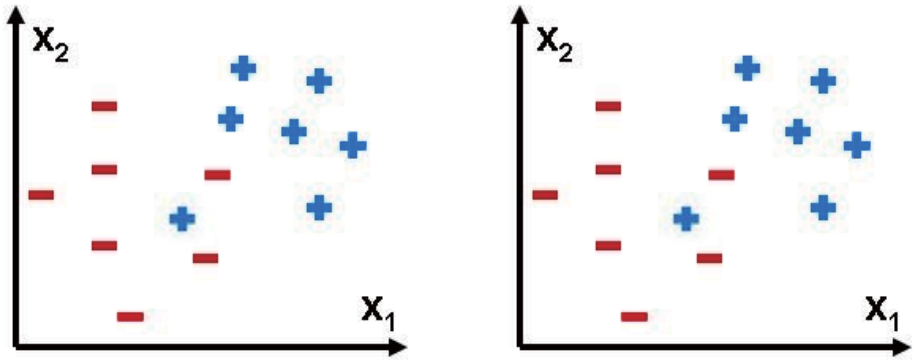


http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

NN tips

- Initilizing weights
- Activation functions
- Regularization
- Optimizers

Overfitting



Am I overfitting?

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

Solutions for overfitting

- More training data!
- Reduce the number of features
- **Regularization**

Regularization

- Let's not have too big numbers in the weight

Regularization

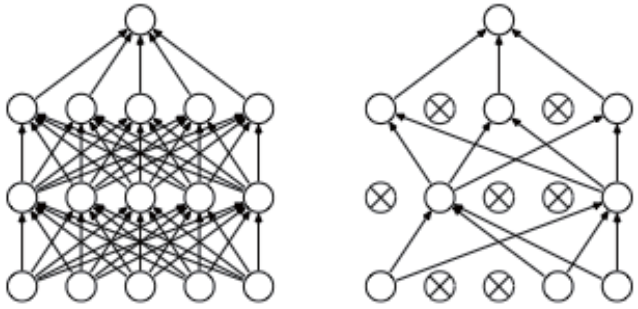
- Let's not have too big numbers in the weight

cos
t $+\lambda \sum W^2$

regularization strength

```
l2reg = 0.001 * tf.reduce_sum(tf.square(W))
```

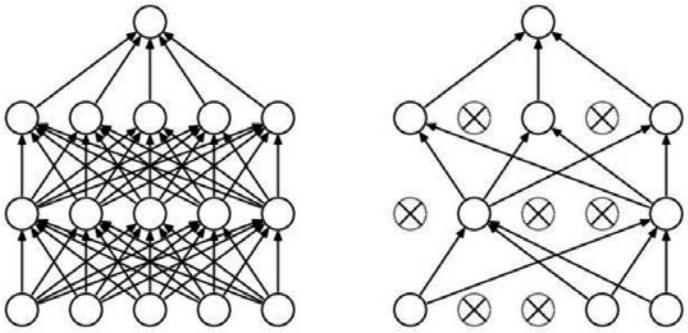

Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]



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

Regularization: **Dropout**

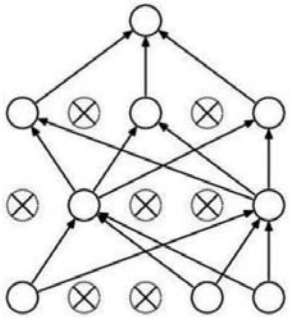
"randomly set some neurons to zero in the forward pass"



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

[Srivastava et al., 2014]

Waaaait a second…
How could this possibly be a good idea?



Forces the network to have a redundant representation.

TensorFlow implementation


```
keep_prob = tf.placeholder("float")

L1 = …

L1_d = tf.nn.dropout(L1, keep_prob=keep_prob)
```

TRAIN:

```
sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys, keep_prob: 0.7})
```

EVALUATION:

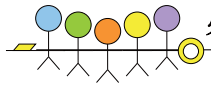
```
print "Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1})
```

Dropout for MNIST


```
# dropout (keep_prob) rate  0.7 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

W1 = tf.get_variable("W1", shape=[784, 512])
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

W2 = tf.get_variable("W2", shape=[512, 512])
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
…
# train my model
for epoch in range(training_epochs):
    ...
   for i in range(total_batch):
       batch_xs, batch_ys = mnist.train.next_batch(batch_size)
       feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
       c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
       avg_cost += c / total_batch

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
     X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))
```

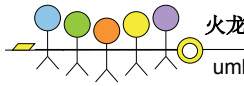
lab-10-5-mnist_nn_dropout.py

NN tips



- Initilizing weights
- Activation functions
- Regularization
- Optimizers

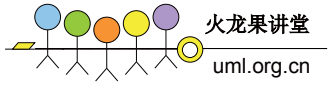
Optimizers




```
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

https://www.tensorflow.org/api_guides/python/train

Optimizers




```
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

tf.train.AdadeltaOptimizer

tf.train.AdagradOptimizer

tf.train.AdagradDAOptimizer

tf.train.MomentumOptimizer

tf.train.AdamOptimizer

tf.train.FtrlOptimizer

tf.train.ProximalGradientDescentOptimizer

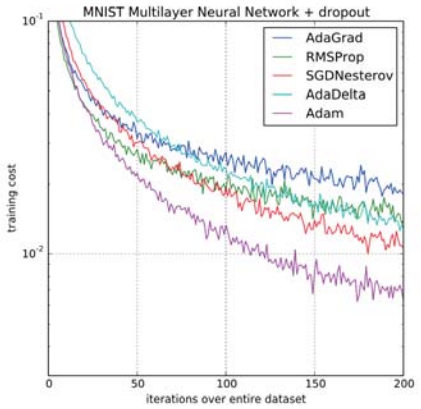
tf.train.ProximalAdagradOptimizer

tf.train.RMSPropOptimizer

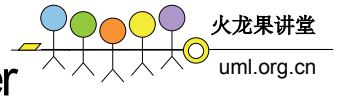
https://www.tensorflow.org/api_guides/python/train

ADAM: a method for stochastic optimization

[Kingma et al. 2015]



Use Adam Optimizer



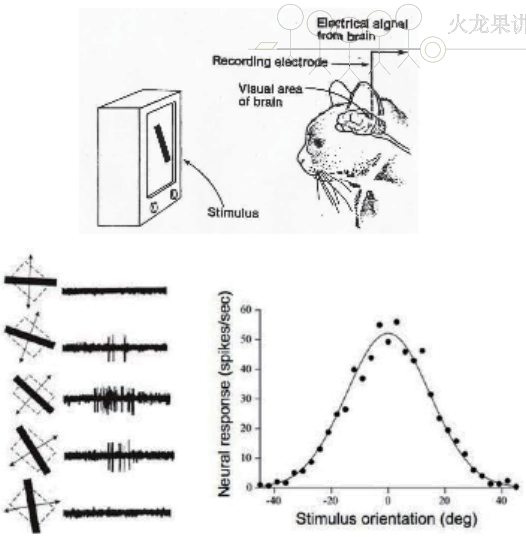

```
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                                                logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

Convolutional Neural Networks

http://cs231n.stanford.edu/

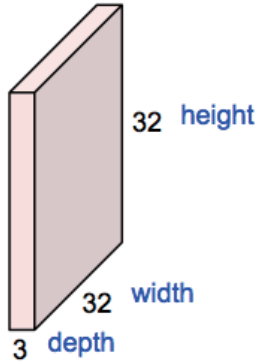
A bit of history:

Hubel & Wiesel,
1959
RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX
1962
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX
1968...



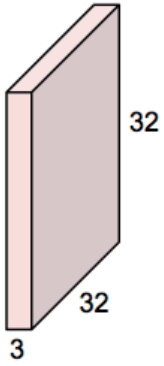
Convolution Layer

32x32x3 image

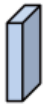


Convolution Layer

32x32x3 image



5x5x3 filter

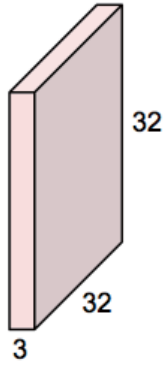


Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

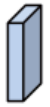
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

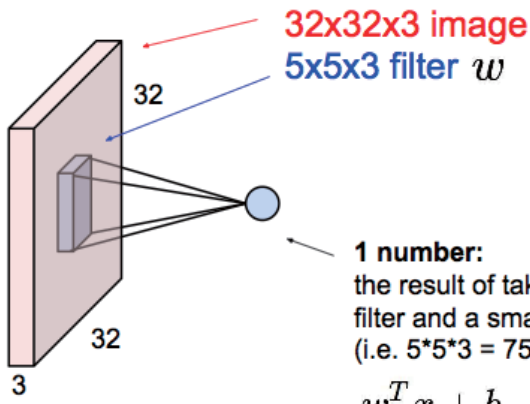


5x5x3 filter



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

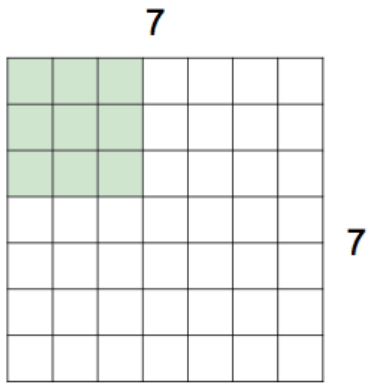
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

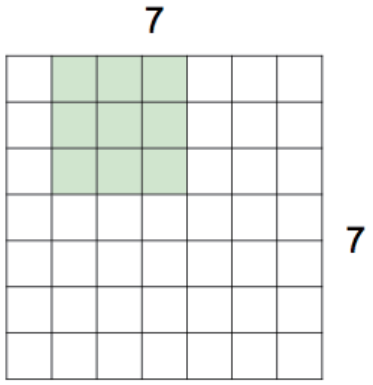
$$w^T x + b$$

A closer look at spatial dimensions:



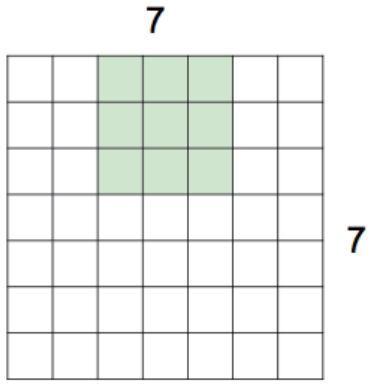
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



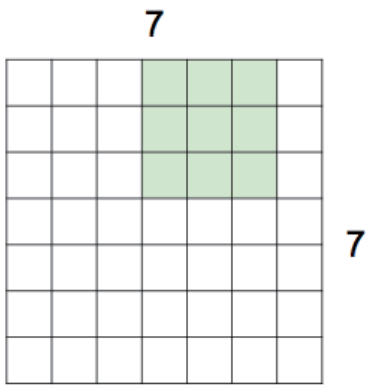
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



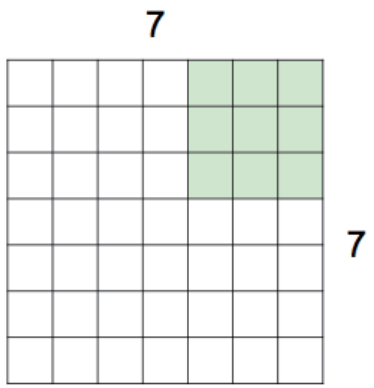
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

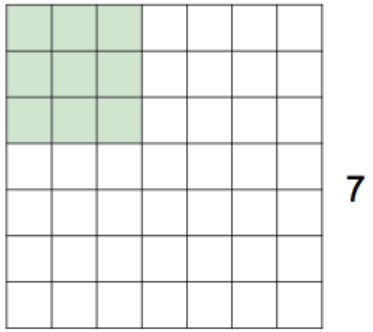
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

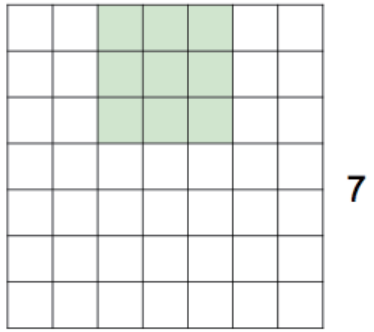
=> 5x5 output

A closer look at spatial dimensions:



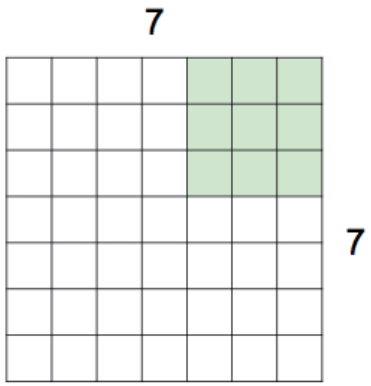
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



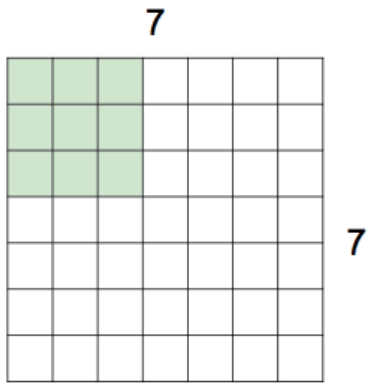
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



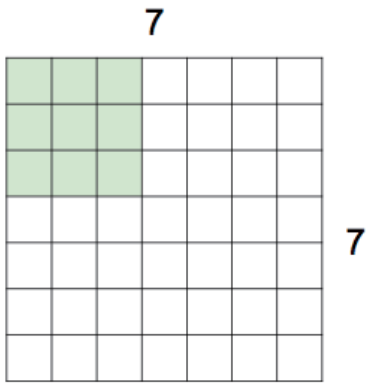
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

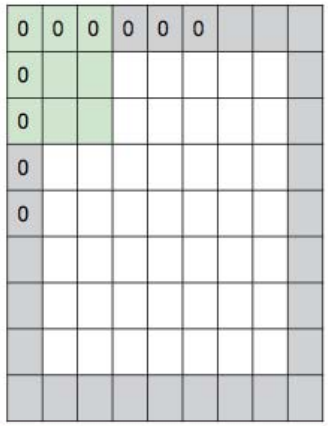
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

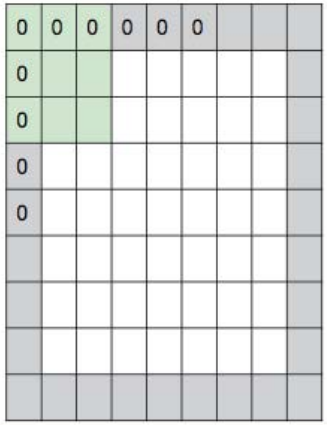
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

In practice: Common to zero pad the border

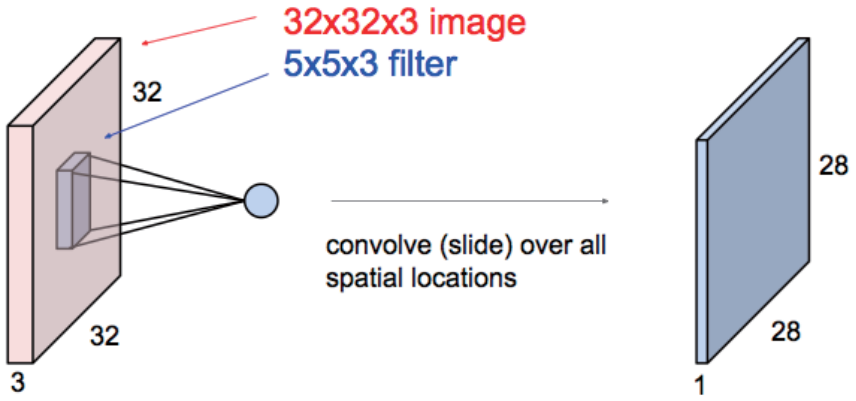


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

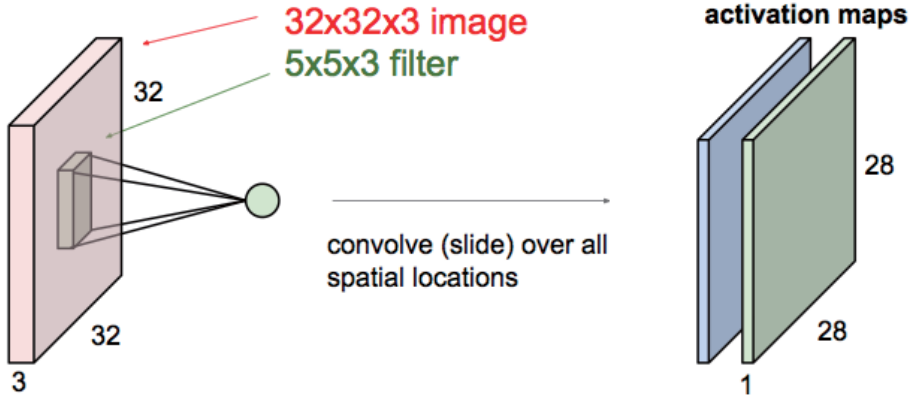
Convolution Layer

activation map

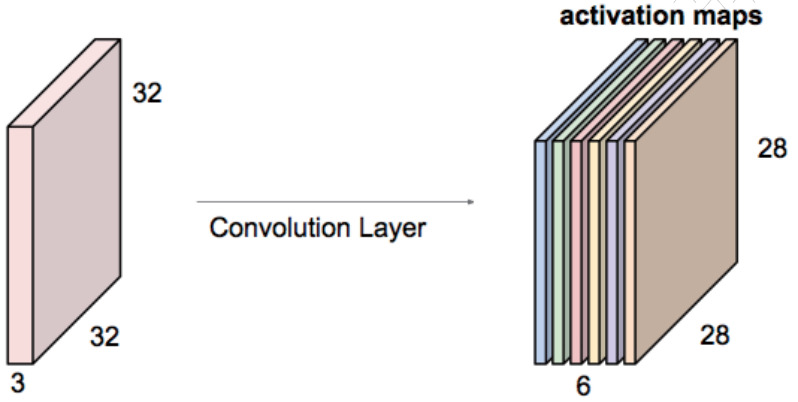


Convolution Layer

consider a second, green filter

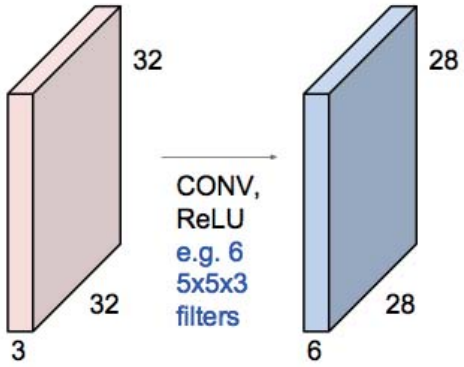


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

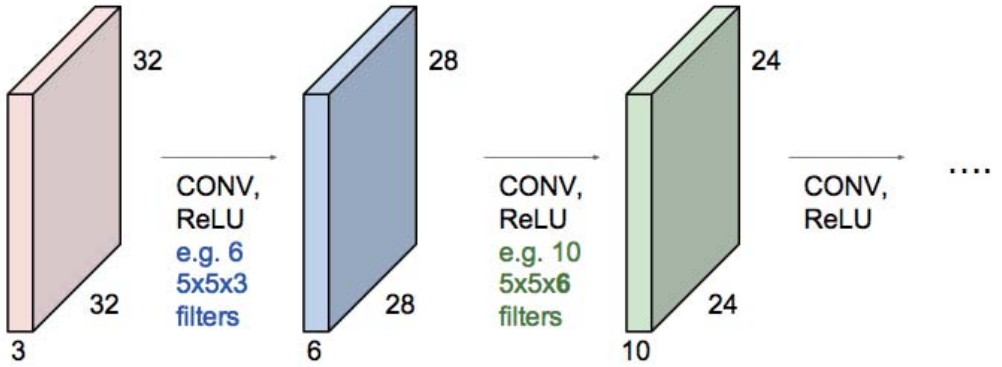


We stack these up to get a "new image" of size 28x28x6!

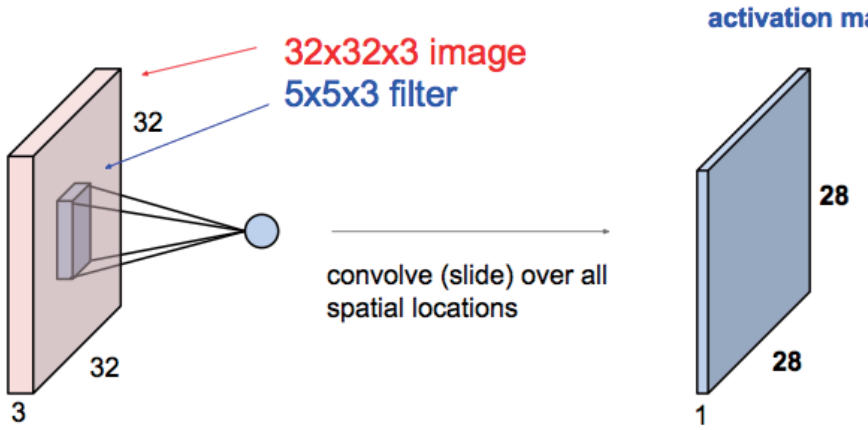
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



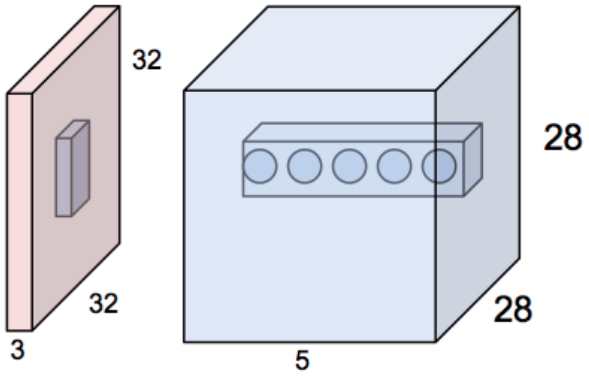
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

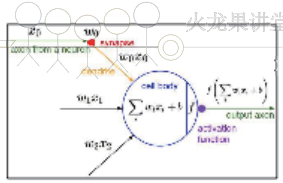


A closer look at spatial dimensions:



The brain/neuron view of CONV Layer



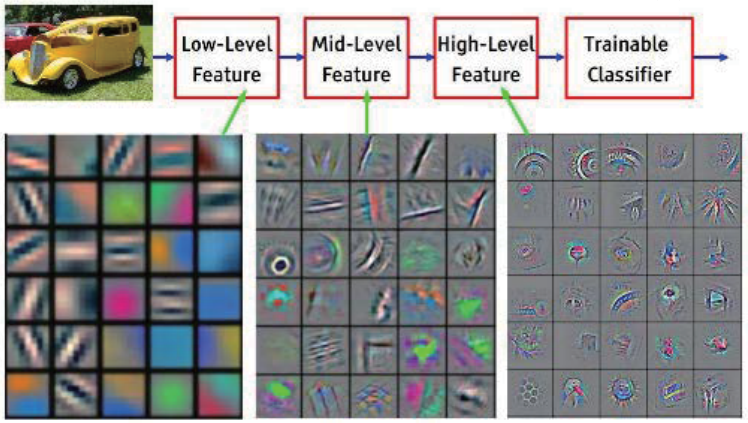


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

There will be 5 different
neurons all looking at the same
region in the input volume

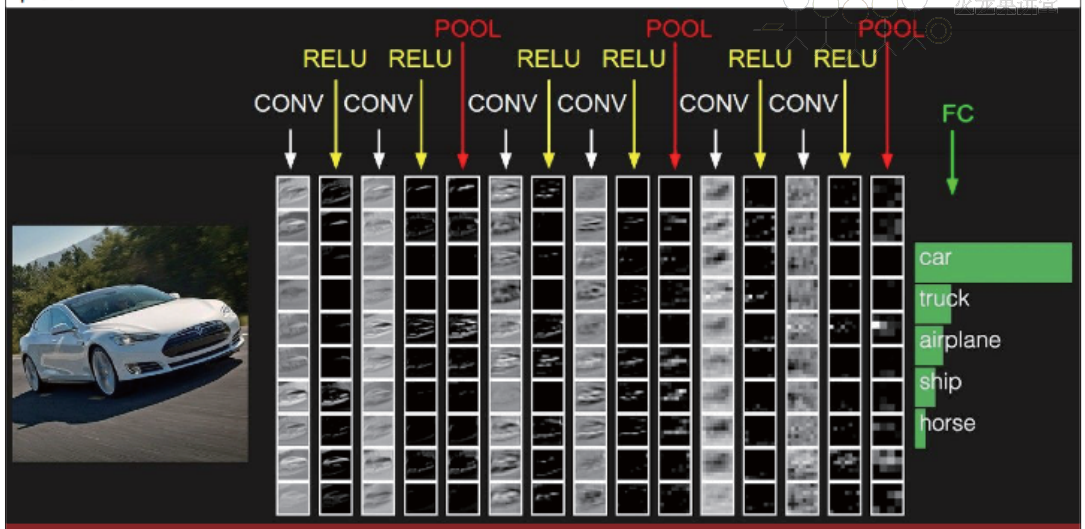
Preview

[From recent Yann LeCun slides]



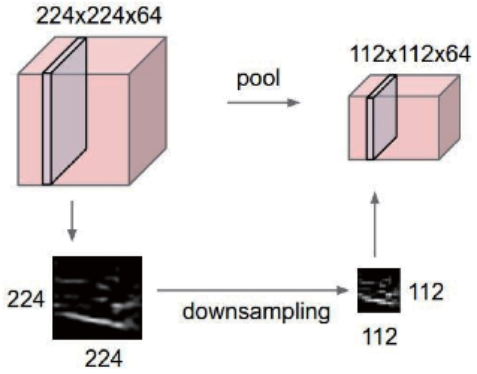
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

preview:

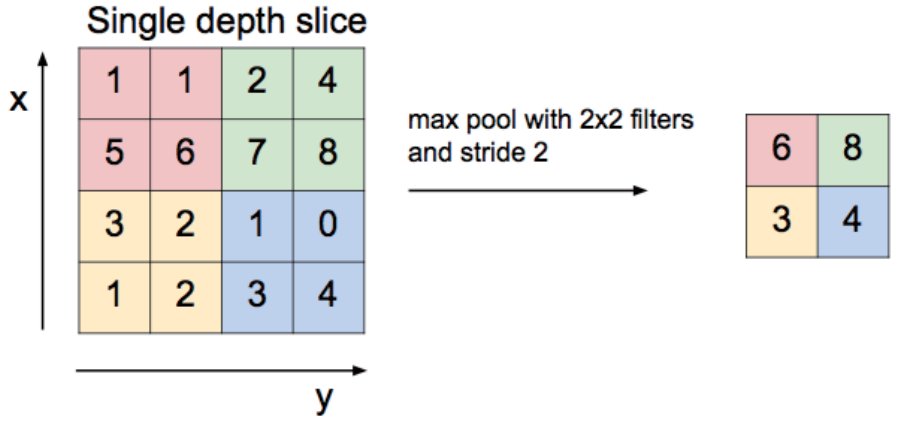


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

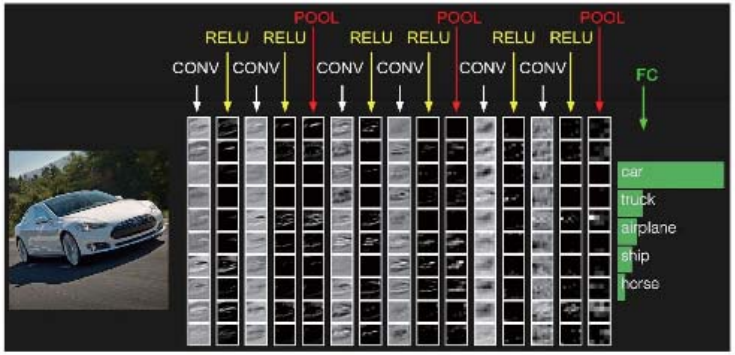


MAX POOLING



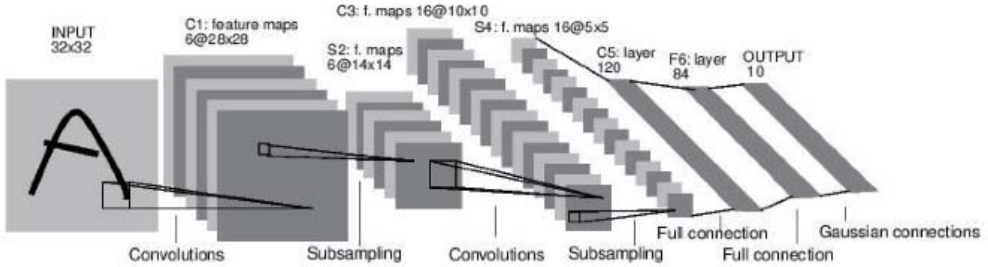
Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Case Study: LeNet-5

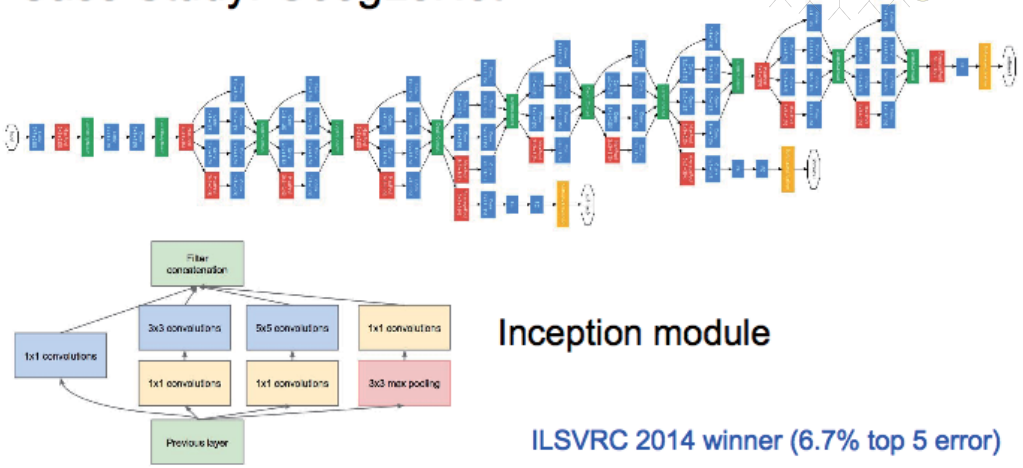
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Convolutional Neural Networks for Sentence Classification

[Yoon Kim, 2014]

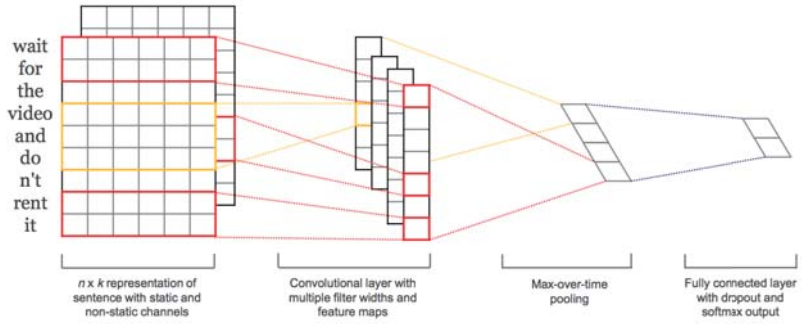
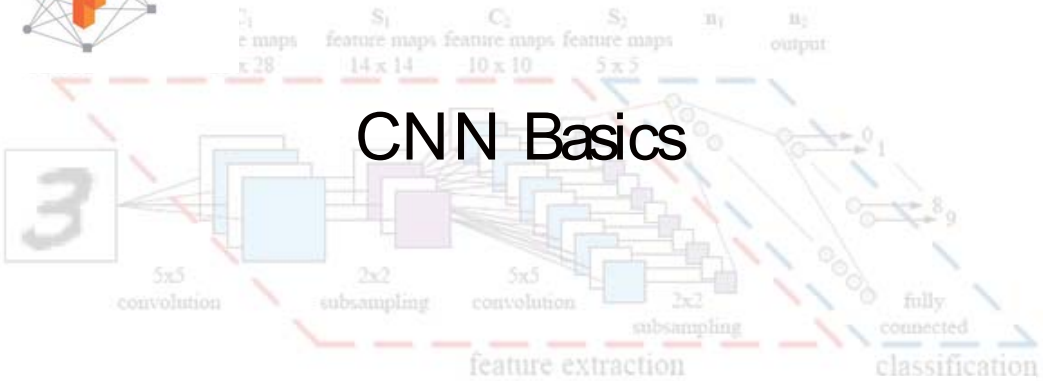


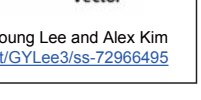
Figure 1: Model architecture with two channels for an example sentence.

With TF 1.0!

火龙果讲堂
uml.org.cn

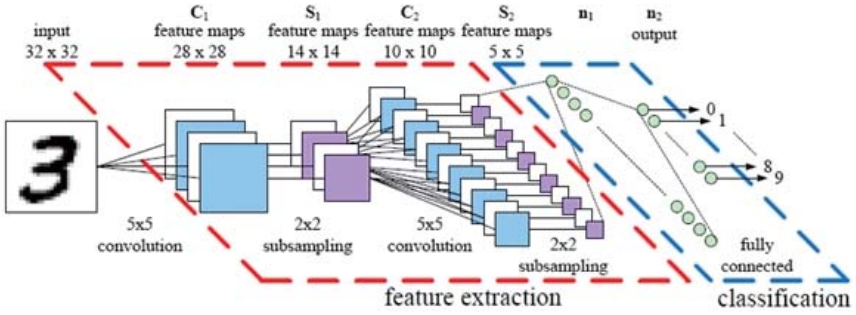
CNN Basics





火龙果讲堂
uml.org.cn

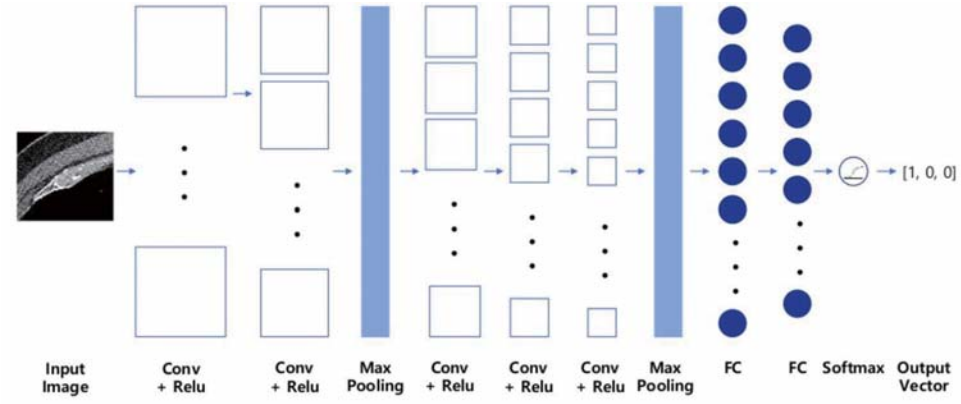
CNN



http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

火龙果讲堂
uml.org.cn

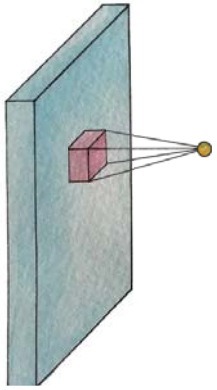
CNN for CT images

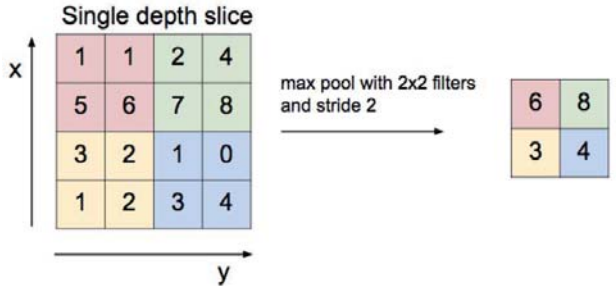


Asan Medical Center & Microsoft Medical Bigdata Contest Winner by GeunYoung Lee and Alex Kim
https://www.slideshare.net/GYLee3/ss-72966495

火龙果讲堂
uml.org.cn

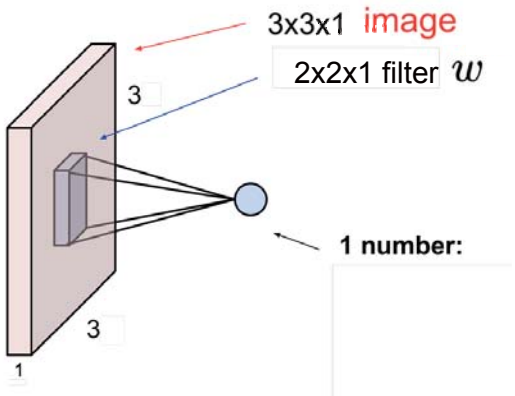
Convolution layer and max pooling

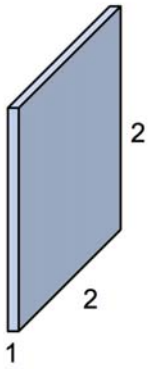




Simple convolution layer

Stride: 1x1

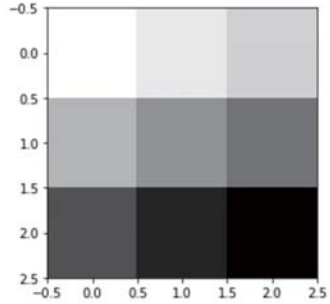





```
In [2]: sess = tf.InteractiveSession()
        image = np.array([[[[1],[2],[3]],
                           [[4],[5],[6]],
                           [[7],[8],[9]]]], dtype=np.float32)
        print(image.shape)
        plt.imshow(image.reshape(3,3), cmap='Greys')
```

```
(1, 3, 3, 1)
```

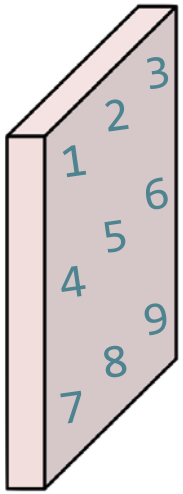
```
Out[2]: <matplotlib.image.AxesImage at 0x10db67dd8>
```

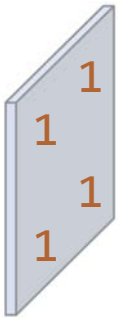


Toy image

Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID






```
[[[[1.]],[[1.]]],
 [[[1.]],[[1.]]]]
shape=(2,2,1,1)
```

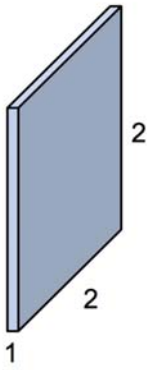
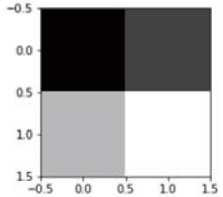
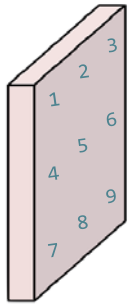


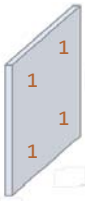
Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID

```
# print("imag:\n", image)
print("image.shape", image.shape)
weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='VALID')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(2,2))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(2,2), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 2, 2, 1)
[[ 12.  16.]
 [ 24.  28.]]
```







Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: SAME

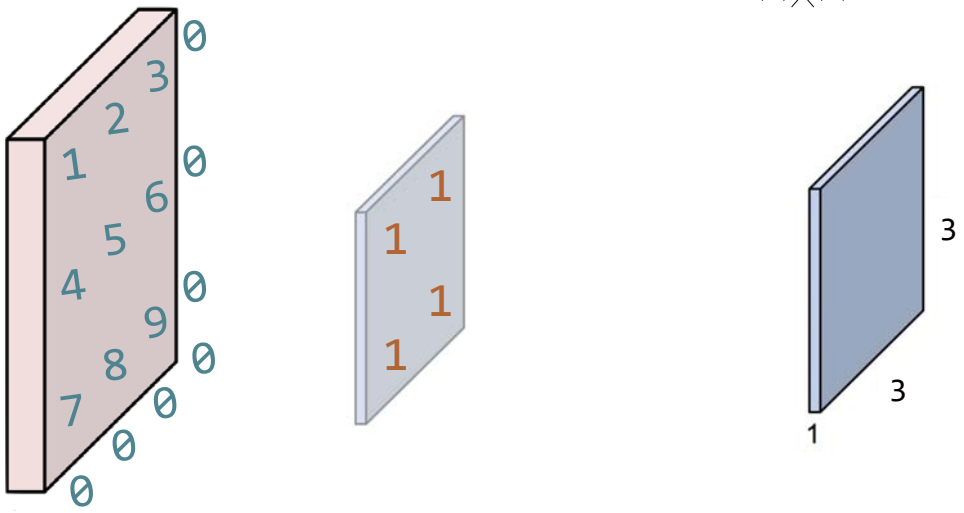
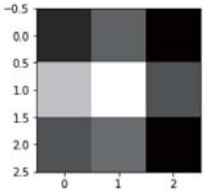
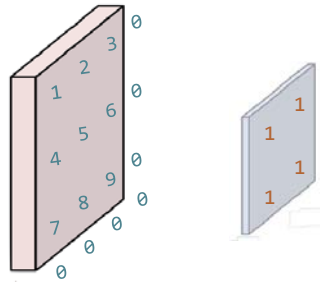


Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: SAME


```
# print("imag:\n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 3, 3, 1)
[[ 12.  16.   9.]
 [ 24.  28.  15.]
 [ 15.  17.   9.]]
```

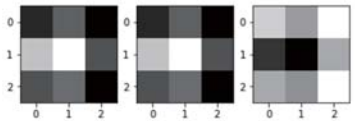




3 filters (2,2,1,3)


```
# print("imag:\n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.,10.,-1.]],[[1.,10.,-1.]]],
                      [[[1.,10.,-1.]],[[1.,10.,-1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,3,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 3)
conv2d_img.shape (1, 3, 3, 3)
[[ 12.  16.   9.]
 [ 24.  28.  15.]
 [ 15.  17.   9.]]
[[ 120.  160.   90.]
 [ 240.  280.  150.]
 [ 150.  170.   90.]]
[[-12. -16.  -9.]
 [-24. -28. -15.]
 [-15. -17.  -9.]]
```



Max Pooling

4	3
2	1

```
In [19]: image = np.array([[[[4],[3]],
                           [[2],[1]]]], dtype=np.float32)
         pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
                               strides=[1, 1, 1, 1], padding='SAME')
         print(pool.shape)
         print(pool.eval())
```

```
(1, 2, 2, 1)
[[[[ 4.]
   [ 3.]]

  [[ 2.]
   [ 1.]]]]
```

SAME: Zero paddings

4	3	0
2	1	0
0	0	0

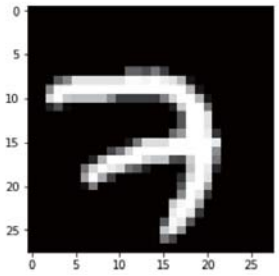
4	3	0
2	1	0
0	0	0

4	3	0
2	1	0
0	0	0

4	3	0
2	1	0
0	0	0

lab-11-0-cnn_basics.ipynb

```
In [6]: from tensorflow.examples.tutorials.mnist import input_data
        mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
        # Check out https://www.tensorflow.org/get_started/mnist/beginners for
        # more information about the mnist dataset

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 [7]: img = mnist.train.images[0].reshape(28,28)
        plt.imshow(img, cmap='gray')

Out[7]: <matplotlib.image.AxesImage at 0x115029ac8>
```

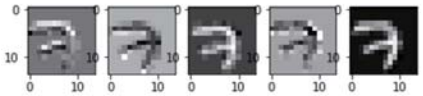


MNIST image loading

lab-11-0-cnn_basics.ipynb

MNIST Convolution layer

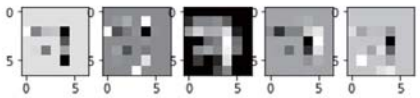
```
In [8]: sess = tf.InteractiveSession()

        img = img.reshape(-1,28,28,1)
        W1 = tf.Variable(tf.random_normal([3, 3, 1, 5], stddev=0.01))
        conv2d = tf.nn.conv2d(img, W1, strides=[1, 2, 2, 1], padding='SAME')
        print(conv2d)
        sess.run(tf.global_variables_initializer())
        conv2d_img = conv2d.eval()
        conv2d_img = np.swapaxes(conv2d_img, 0, 3)
        for i, one_img in enumerate(conv2d_img):
            plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(14,14), cmap='gray')

Tensor("Conv2D_1:0", shape=(1, 14, 14, 5), dtype=float32)
```



lab-11-0-cnn_basics.ipynb

MNIST Max pooling

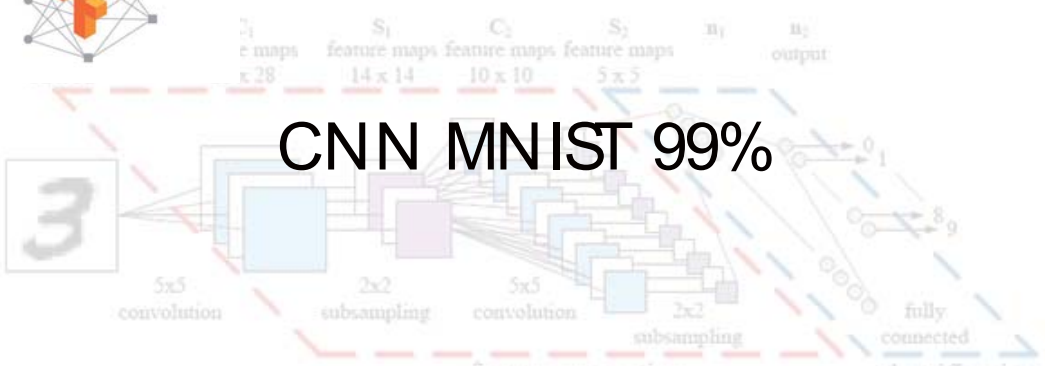
```
In [9]: pool = tf.nn.max_pool(conv2d, ksize=[1, 2, 2, 1], strides=[
                              1, 2, 2, 1], padding='SAME')
        print(pool)
        sess.run(tf.global_variables_initializer())
        pool_img = pool.eval()
        pool_img = np.swapaxes(pool_img, 0, 3)
        for i, one_img in enumerate(pool_img):
            plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(7, 7), cmap='gray')

Tensor("MaxPool_2:0", shape=(1, 7, 7, 5), dtype=float32)
```



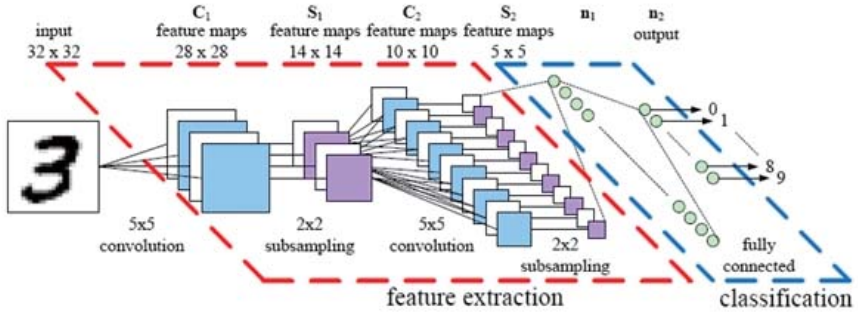
lab-11-0-cnn_basics.ipynb

With TF 1.0!

CNN MNIST 99%

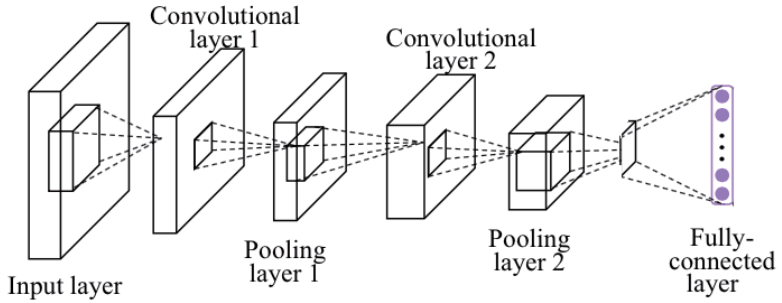


CNN

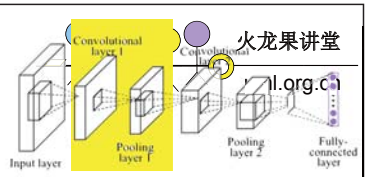


http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

Simple CNN



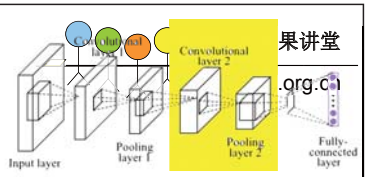
Conv layer 1



```
# input placeholders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 10])

# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv      -> (?, 28, 28, 32)
#    Pool      -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
          strides=[1, 2, 2, 1], padding='SAME')
'''
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
'''
```

lab-11-1-mnist_cnn.py

Conv layer 2

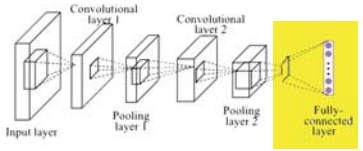


```
'''
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
'''

# L2 ImgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.reshape(L2, [-1, 7 * 7 * 64])
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)
```

lab-11-1-mnist_cnn.py

Fully Connected (FC, Dense) layer


```
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)
'''
L2 = tf.reshape(L2, [-1, 7 * 7 * 64])

# Final FC 7x7x64 inputs -> 10 outputs
W3 = tf.get_variable("W3", shape=[7 * 7 * 64, 10],
initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis,
labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```



lab-11-1-mnist_cnn.py

Training and Evaluation


```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
print('Learning stared. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _, = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y:
mnist.test.labels}))
```

lab-11-1-mnist_cnn.py

Training and Evaluation


```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
print('Learning stared. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _, = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y:
mnist.test.labels}))
```

```
Epoch: 0001 cost = 0.340291267
Epoch: 0002 cost = 0.090731326
Epoch: 0003 cost = 0.064477619
Epoch: 0004 cost = 0.050683064
...
Epoch: 0011 cost = 0.017758641
Epoch: 0012 cost = 0.014156652
Epoch: 0013 cost = 0.012397016
Epoch: 0014 cost = 0.010693789
Epoch: 0015 cost = 0.009469977
Learning Finished!
Accuracy: 0.9885
```

lab-11-1-mnist_cnn.py

Deep CNN

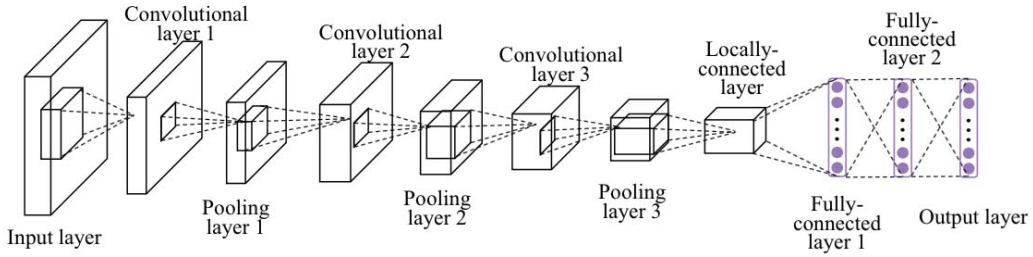


Image credit: http://personal.ie.cuhk.edu.hk/~ccloy/project_target_code/index.html

```
# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv      -> (?, 28, 28, 32)
#    Pool      -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
'''Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
   Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
   Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
   Tensor("dropout/mul:0", shape=(?, 14, 14, 32), dtype=float32)'''

# L2 ImgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
'''Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
   Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
   Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
   Tensor("dropout_1/mul:0", shape=(?, 7, 7, 64), dtype=float32)'''
```

```
# L3 ImgIn shape=(?, 7, 7, 64)
W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
#    Conv      ->(?, 7, 7, 128)
#    Pool      ->(?, 4, 4, 128)
#    Reshape   ->(?, 4 * 4 * 128) # Flatten them for FC
L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
L3 = tf.nn.relu(L3)
L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')
L3 = tf.nn.dropout(L3, keep_prob=keep_prob)
L3 = tf.reshape(L3, [-1, 128 * 4 * 4])
'''Tensor("Conv2D_2:0", shape=(?, 7, 7, 128), dtype=float32)
   Tensor("Relu_2:0", shape=(?, 7, 7, 128), dtype=float32)
   Tensor("MaxPool_2:0", shape=(?, 4, 4, 128), dtype=float32)
   Tensor("dropout_2/mul:0", shape=(?, 4, 4, 128), dtype=float32)
   Tensor("Reshape_1:0", shape=(?, 2048), dtype=float32)'''
# L4 FC 4x4x128 inputs -> 625 outputs
W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([625]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)
'''Tensor("Relu_3:0", shape=(?, 625), dtype=float32)
   Tensor("dropout_3/mul:0", shape=(?, 625), dtype=float32)'''
# L5 Final FC 625 inputs -> 10 outputs
W5 = tf.get_variable("W5", shape=[625, 10],
initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5
'''Tensor("add_1:0", shape=(?, 10), dtype=float32)'''
```

lab-11-2-mnist_deep_cnn.py


```
# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv      -> (?, 28, 28, 32)
#    Pool      -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
'''Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
   Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
   Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
   Tensor("dropout/mul:0", shape=(?, 14, 14, 32), dtype=float32)'''
...
...
# L4 FC 4x4x128 inputs -> 625 outputs
W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([625]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)
'''Tensor("Relu_3:0", shape=(?, 625), dtype=float32)
   Tensor("dropout_3/mul:0", shape=(?, 625), dtype=float32)'''
# L5 Final FC 625 inputs -> 10 outputs
W5 = tf.get_variable("W5", shape=[625, 10],
initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5
'''Tensor("add_1:0", shape=(?, 10), dtype=float32)'''
```

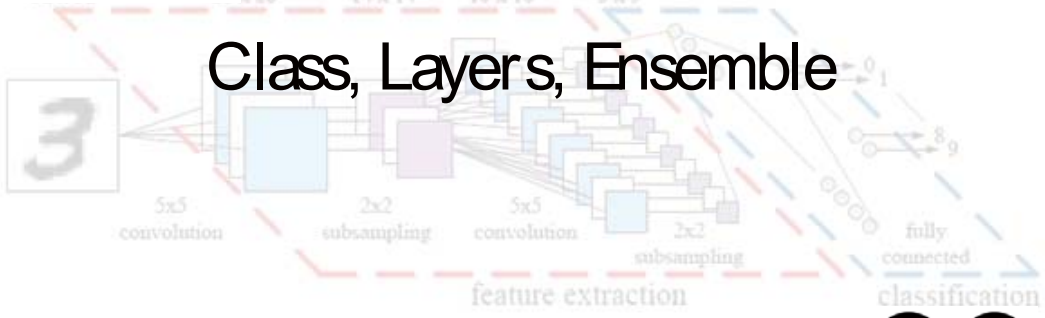

lab-11-2-mnist_deep_cnn.py

```
# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1),
        tf.argmax(Y, 1))
accuracy =
tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
print('Accuracy:', sess.run(accuracy,
feed_dict={X: mnist.test.images,
           Y: mnist.test.labels, keep_prob: 1}))
```

```
Epoch: 0013 cost = 0.027188021
Epoch: 0014 cost = 0.023604777
Epoch: 0015 cost = 0.024607201
Learning Finished!
Accuracy: 0.9938
```

With TF 1.0!

Class, Layers, Ensemble




```
# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv      -> (?, 28, 28, 32)
#    Pool      -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
'''Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
   Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
   Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
   Tensor("dropout/mul:0", shape=(?, 14, 14, 32), dtype=float32)'''
...
...
# L4 FC 4x4x128 inputs -> 625 outputs
W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([625]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)
'''Tensor("Relu_3:0", shape=(?, 625), dtype=float32)
   Tensor("dropout_3/mul:0", shape=(?, 625), dtype=float32)'''
# L5 Final FC 625 inputs -> 10 outputs
W5 = tf.get_variable("W5", shape=[625, 10],
initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5
'''Tensor("add_1:0", shape=(?, 10), dtype=float32)'''
```


lab-11-2-mnist_deep_cnn.py


```
# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1),
        tf.argmax(Y, 1))
accuracy =
tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
print('Accuracy:', sess.run(accuracy,
feed_dict={X: mnist.test.images,
           Y: mnist.test.labels, keep_prob: 1}))
```

```
Epoch: 0013 cost = 0.027188021
Epoch: 0014 cost = 0.023604777
Epoch: 0015 cost = 0.024607201
Learning Finished!
Accuracy: 0.9938
```

Python Class


```
class Model:

    def __init__(self, sess, name):
        self.sess = sess
        self.name = name
        self._build_net()

    def _build_net(self):
        with tf.variable_scope(self.name):
            # input place holders
            self.X = tf.placeholder(tf.float32, [None, 784])
            # img 28x28x1 (black/white)
            X_img = tf.reshape(self.X, [-1, 28, 28, 1])
            self.Y = tf.placeholder(tf.float32, [None, 10])

            # L1 ImgIn shape=(?, 28, 28, 1)
            W1 = tf.Variable(tf.random_normal([3, 3, 1, 32],
                                              stddev=0.01))
            ...
def predict(self, x_test, keep_prop=1.0):
    return self.sess.run(self.logits,
         feed_dict={self.X: x_test, self.keep_prob: keep_prop})

def get_accuracy(self, x_test, y_test, keep_prop=1.0):
    return self.sess.run(self.accuracy,
     feed_dict={self.X: x_test, self.Y: y_test, self.keep_prob: keep_prop})

def train(self, x_data, y_data, keep_prop=0.7):
    return self.sess.run([self.cost, self.optimizer], feed_dict={
        self.X: x_data, self.Y: y_data, self.keep_prob: keep_prop})
```

```
# initialize
sess = tf.Session()
m1 = Model(sess, "m1")

sess.run(tf.global_variables_initializer())

print('Learning Started!')

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        c, _ = m1.train(batch_xs, batch_ys)
        avg_cost += c / total_batc
```

lab-11-3-mnist_cnn_class.py

tf.layers

`average_pooling1d(...)` : Average Pooling layer for 1D inputs.

`average_pooling2d(...)` : Average pooling layer for 2D inputs (e.g. images).

`average_pooling3d(...)` : Average pooling layer for 3D inputs (e.g. volumes).

`batch_normalization(...)` : Functional interface for the batch normalization layer.

`conv1d(...)` : Functional interface for 1D convolution layer (e.g. temporal convolution).

`conv2d(...)` : Functional interface for the 2D convolution layer.

`conv2d_transpose(...)` : Transposed convolution layer (sometimes called Deconvolution).

`conv3d(...)` : Functional interface for the 3D convolution layer.

`dense(...)` : Functional interface for the densely-connected layer.

`dropout(...)` : Applies Dropout to the input.

`max_pooling1d(...)` : Max Pooling layer for 1D inputs.

`max_pooling2d(...)` : Max pooling layer for 2D inputs (e.g. images).

`max_pooling3d(...)` : Max pooling layer for 3D inputs (e.g. volumes).

`separable_conv2d(...)` : Functional interface for the depthwise separable 2D convolution layer.

https://www.tensorflow.org/api_docs/python/tf/layers

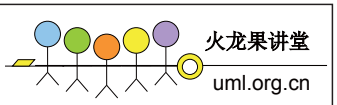
tf.layers

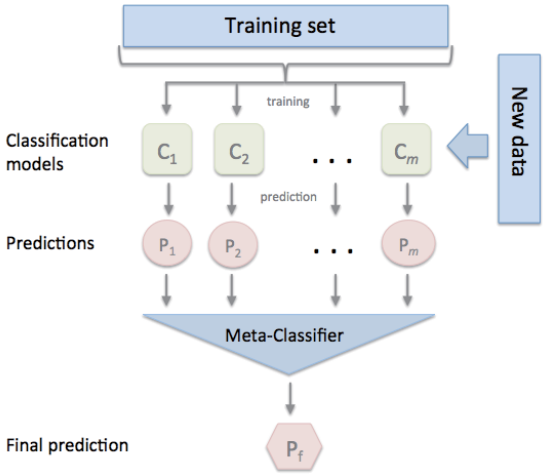

```
        # L1 ImgIn shape=(?, 28, 28, 1)
        W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
        L1 = tf.nn.relu(L1)
        L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
        L1 = tf.nn.dropout(L1, keep_prob=self.keep_prob)
        …
        # L2 ImgIn shape=(?, 14, 14, 32)
        W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
```

```
# Convolutional Layer #1
conv1 = tf.layers.conv2d(inputs=X_img,filters=32,kernel_size=[3,3],padding="SAME",activation=tf.nn.relu)
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], padding="SAME", strides=2)
dropout1 = tf.layers.dropout(inputs=pool1,rate=0.7, training=self.training)

# Convolutional Layer #2
conv2 = tf.layers.conv2d(inputs=dropout1,filters=64,kernel_size=[3,3],padding="SAME",activation=tf.nn.relu)
…
flat = tf.reshape(dropout3, [-1, 128 * 4 * 4])
dense4 = tf.layers.dense(inputs=flat, units=625, activation=tf.nn.relu)
dropout4 = tf.layers.dropout(inputs=dense4, rate=0.5, training=self.training)
...
```

lab-11-4-mnist_cnn_layers.py

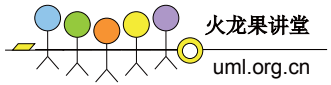
Ensemble





http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

Ensemble training

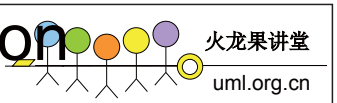



```
class Model:

  def __init__(self, sess, name):
    self.sess = sess
    self.name = name
    self._build_net()

  def _build_net(self):
    with tf.variable_scope(self.name):

  ...
```

```
models = []
num_models = 7
for m in range(num_models):
   models.append(Model(sess, "model" + str(m)))

sess.run(tf.global_variables_initializer())
print('Learning Started!')

# train my model
for epoch in range(training_epochs):
   avg_cost_list = np.zeros(len(models))
   total_batch = int(mnist.train.num_examples / batch_size)
   for i in range(total_batch):
       batch_xs, batch_ys =mnist.train.next_batch(batch_size)

       # train each model
       for m_idx, m in enumerate(models):
           c, _ = m.train(batch_xs, batch_ys)
           avg_cost_list[m_idx] += c / total_batch

   print('Epoch:','%04d'%(epoch + 1),'cost =', avg_cost_list)

print('Learning Finished!')
```

lab-11-5-mnist_cnn_ensemble_layers.py

Ensemble prediction

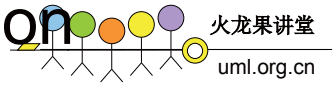


C_1

C_2

C_m

Ensemble prediction



	0	1	2	3	4	5	6	7	8	9
C_1	0.1	0.01	0.02	0.8	...	...	...	...	...	...
C_2	0.01	0.5	0.02	0.4	...	...	...	...	...	...
.										
.										
.										
C_m	0.01	0.01	0.1	0.7	...	...	...	...	...	...
Sum	0.12	0.52	0.14	**1.9**	...	...	...	...	...	...

argmax

Ensemble prediction

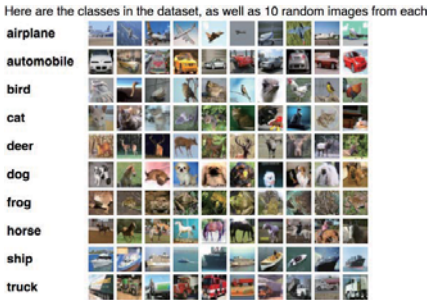

```
# Test model and check accuracy
test_size = len(mnist.test.labels)
predictions = np.zeros(test_size * 10).reshape(test_size, 10)


for m_idx, m in enumerate(models):
   print(m_idx, 'Accuracy:', m.get_accuracy(mnist.test.images, mnist.test.labels))
   p = m.predict(mnist.test.images)
   predictions += p

ensemble_correct_prediction = tf.equal(
    tf.argmax(predictions, 1), tf.argmax(mnist.test.labels, 1))
ensemble_accuracy = tf.reduce_mean(
   tf.cast(ensemble_correct_prediction, tf.float32))
print('Ensemble accuracy:', sess.run(ensemble_accuracy))
```

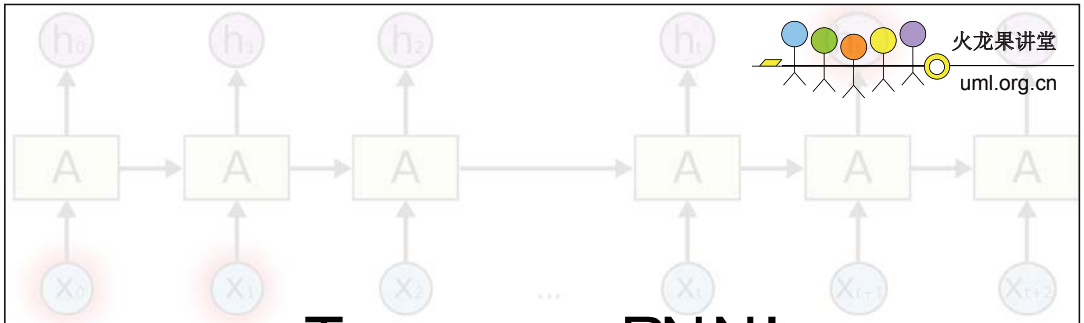
```
0 Accuracy: 0.9933
1 Accuracy: 0.9946
2 Accuracy: 0.9934
3 Accuracy: 0.9935
4 Accuracy: 0.9935
5 Accuracy: 0.9949
6 Accuracy: 0.9941

Ensemble accuracy: 0.9952
```

lab-11-5-mnist_cnn_ensemble_layers.py

Exercise

- Deep & Wide?
- CIFAR 10
- ImageNet





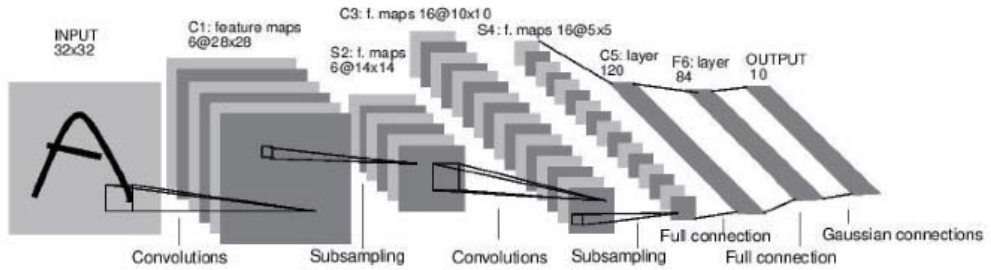
Tomorrow RNN!

- Time series prediction (stock market)
- Character/sentence analysis
- Neural translation (sequence to sequence)

Backup slides

Case Study: LeNet-5

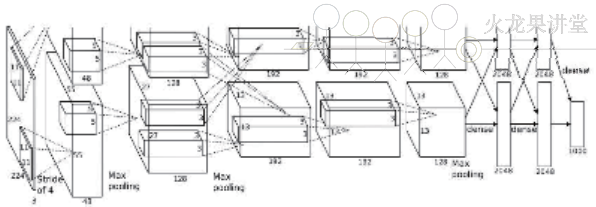
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

[Krizhevsky et al. 2012]



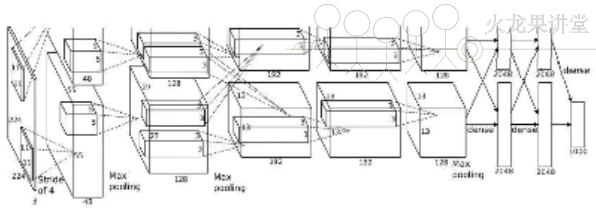
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

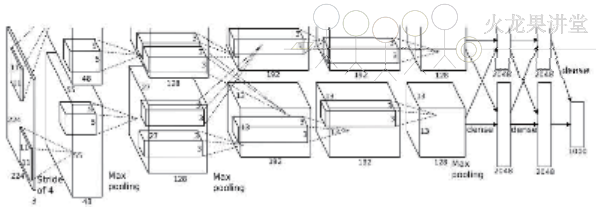


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet

[Krizhevsky et al. 2012]

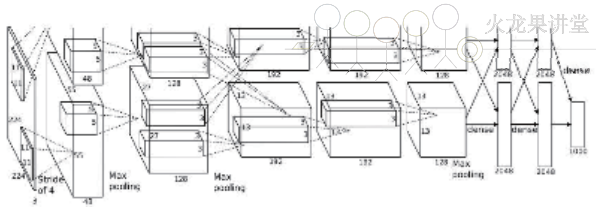


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet

[Krizhevsky et al. 2012]

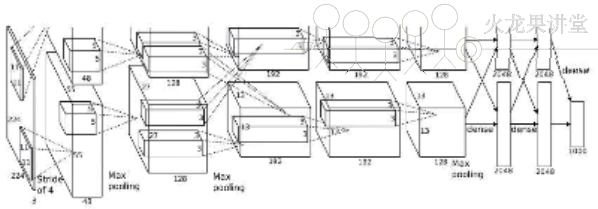


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Case Study: AlexNet

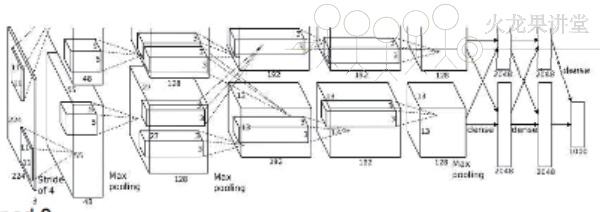
[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Case Study: AlexNet

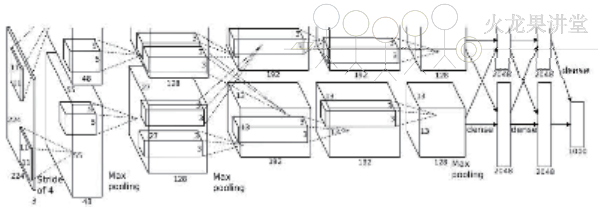
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Case Study: AlexNet

[Krizhevsky et al. 2012]

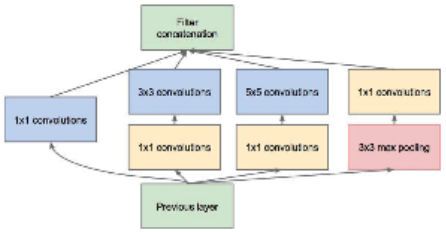


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study: GoogLeNet

[Szegedy et al., 2014]

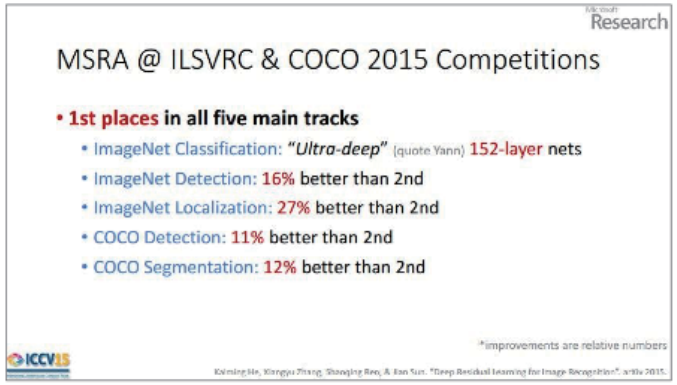


Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Case Study: ResNet [He et al., 2015]

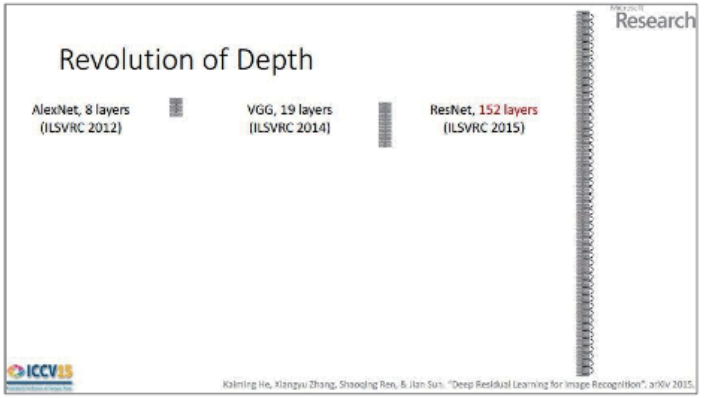
ILSVRC 2015 winner (3.6% top 5 error)



Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

Case Study: ResNet [He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



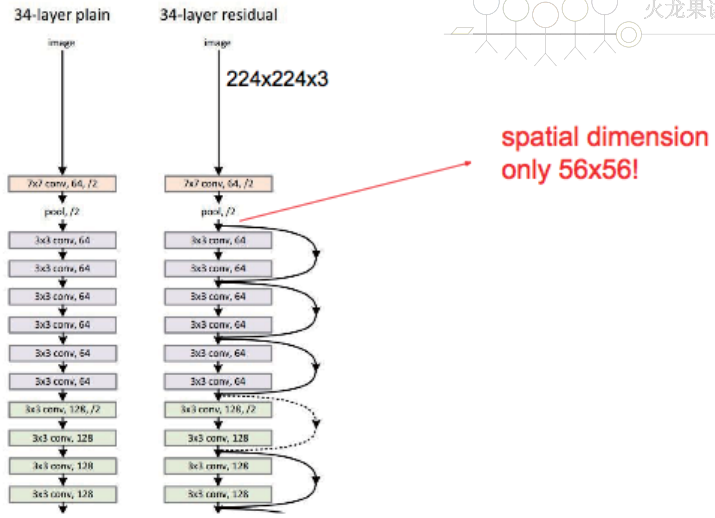
2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

(slide from Kaiming He's recent presentation)

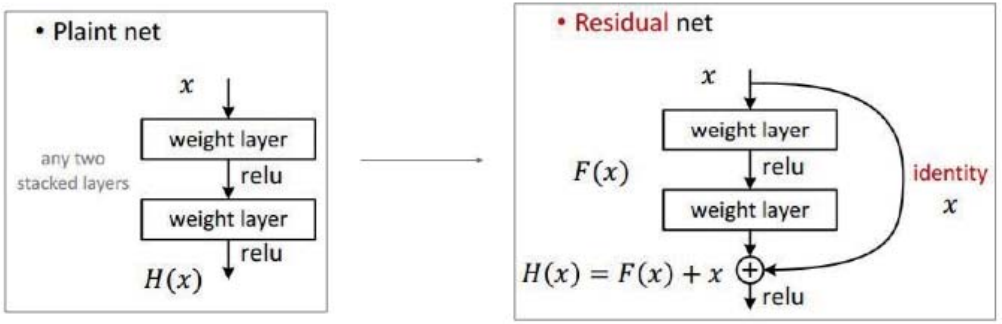
Case Study: ResNet

[He et al., 2015]



spatial dimension only 56x56!

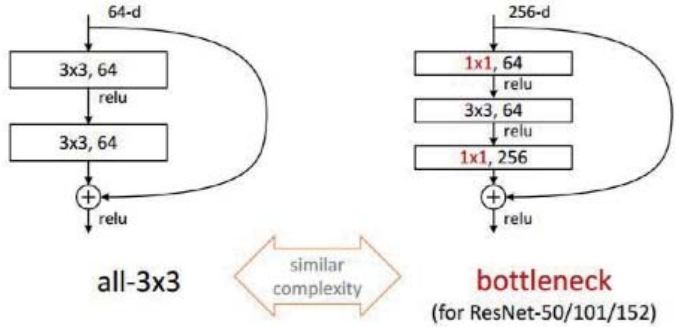
Case Study: ResNet [He et al., 2015]



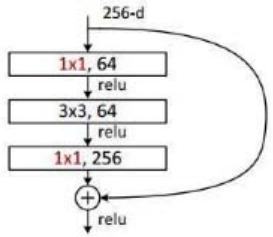
Case Study: ResNet [He et al., 2015]

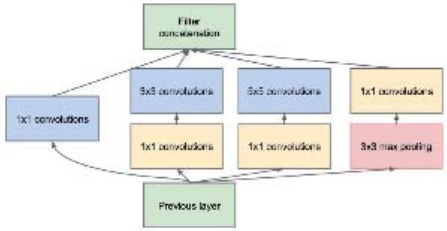
- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Case Study: ResNet [He et al., 2015]

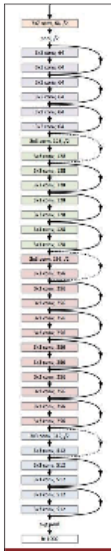


Case Study: ResNet [He et al., 2015]





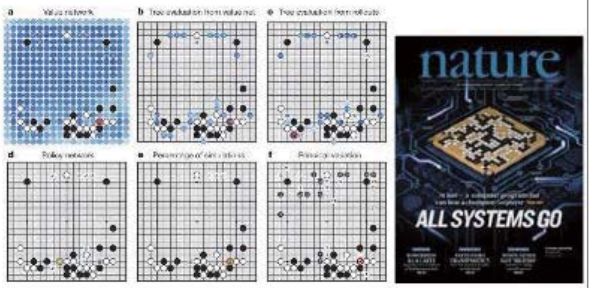
(this trick is also used in GoogLeNet)



Case Study: ResNet [He et al., 2015]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Case Study Bonus: DeepMind's AlphaGo



The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

RNN

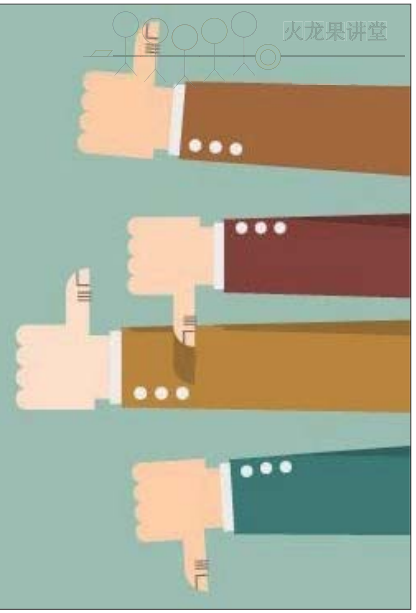
循环神经网络
深度学习网络架构

火龙果讲堂
uml.org.cn

Gifts from Google and Line

LINE Google

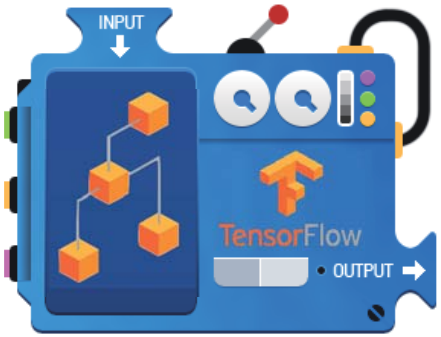
Feedback



TensorFlow Mechanics

2 feed data and run graph (operation)
sess.run (op, feed_dict={x: x_data})

1 Build graph using TensorFlow operations



3 update variables in the graph (and return values)

WWW.MATHWAREHOUSE.COM

Machine Learning Basics

- Linear Regression
- Logistic Regression (Binary classification)
- Softmax Classification
- Neural Networks
- CNN

Linear Regression

1 $H(x) = Wx + b$

```
W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
# Our hypothesis XW+b
hypothesis = X * W + b
```

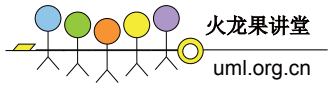
2 $cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$

```
# cost/Loss function
cost = tf.reduce_mean(tf.square(hypothesis - y_train))
```

3 GradientDescent $W := W - \alpha\frac{\partial}{\partial W}cost(W)$

```
# Minimize
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

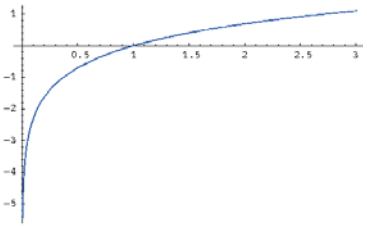
Logistic Regression



1. Model? (Hypothesis?)

$$H(X) = sigmoid(XW) = \frac{1}{1+e^{-XW}}$$

```
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
```



2. Cost?

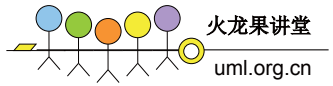
$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1-y)(log(1-H(x))$$

```
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))
```

3. GradientDescent $W := W - \alpha \frac{\partial}{\partial W} cost(W)$

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

Softmax Classifier



1. Model? (Hypothesis?)

```
logits = tf.matmul(X,W)+b
hypothesis = tf.nn.softmax(logits)
```

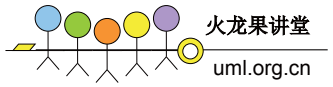
2. Cost?

```
# Cross entropy cost/loss
cost = tf.reduce_mean (-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

3. GradientDescent $W := W - \alpha \frac{\partial}{\partial W} cost(W)$

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

Softmax Classifier



1. Model? (Hypothesis?)

```
logits = tf.matmul(X,W)+b
hypothesis = tf.nn.softmax(logits)
```

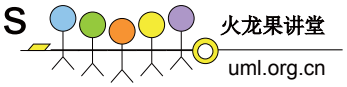
2. Cost?

```
# Cross entropy cost/loss
cost = tf.reduce_mean (-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

cost = tf.reduce_mean(
     tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=Y))
```

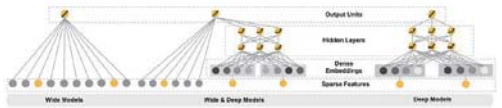
3. GradientDescent $W := W - \alpha \frac{\partial}{\partial W} cost(W)$

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

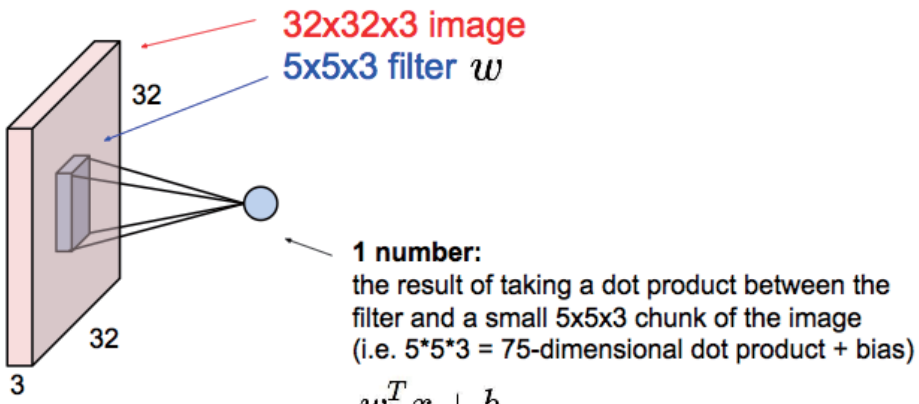
(Deep) Neural Nets



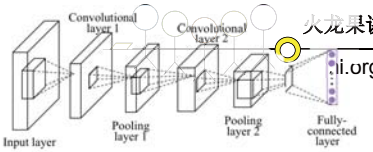

```
# input placeholders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
   logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```



Convolution Layer



Simple CNN




```
# img 28x28x1 (black/white), Input Layer
X_img = tf.reshape(self.X, [-1, 28, 28, 1])
self.Y = tf.placeholder(tf.float32, [None, 10])

# Convolutional Layer and pooling layer #1
conv1 = tf.layers.conv2d(inputs=X_img, filters=32, kernel_size=[3, 3], padding="SAME", activation=tf.nn.relu)
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], padding="SAME", strides=2)
dropout1 = tf.layers.dropout(inputs=pool1, rate=0.7, training=self.training)

# Convolutional Layer and Pooling Layer #2
conv2 = tf.layers.conv2d(inputs=dropout1, filters=64, kernel_size=[3, 3], padding="SAME", activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], padding="SAME", strides=2)
dropout2 = tf.layers.dropout(inputs=pool2, rate=0.7, training=self.training)

# Dense Layer with Relu
flat = tf.reshape(dropout3, [-1, 64 * 7 * 7])
dense4 = tf.layers.dense(inputs=flat, units=625, activation=tf.nn.relu)
dropout4 = tf.layers.dropout(inputs=dense4, rate=0.5, training=self.training)

# Logits (no activation) Layer: L5 Final FC 625 inputs -> 10 outputs
self.logits = tf.layers.dense(inputs=dropout4, units=10)

# define cost/loss & optimizer
self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=self.logits, labels=self.Y))
self.optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(self.cost)
```

NN tips

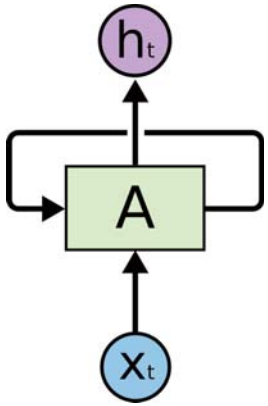
- Initilizing weights
- Activation functions
- Regularization
- Optimizers

Sequence data

- Hard to understand the context from single word
- We understand based on the previous words + this word. (time series)
- NN/CNN cannot do this

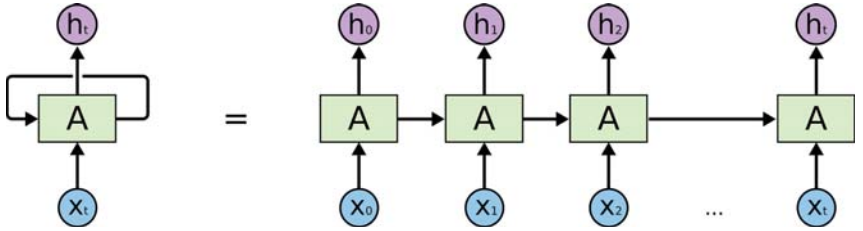
Sequence data

- Hard to understand the context from single word
- We understand based on the previous words + this word. (time series)
- NN/CNN cannot do this



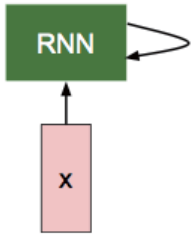
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

火龙果讲堂
uml.org.cn

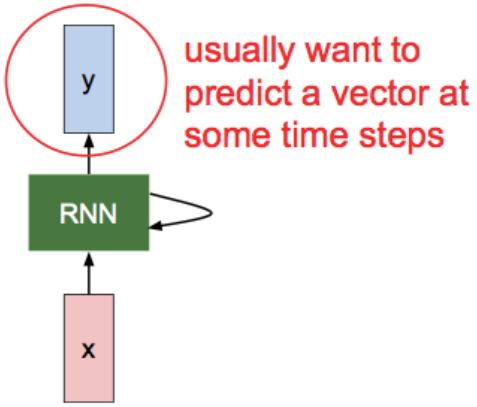


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Network



Recurrent Neural Network

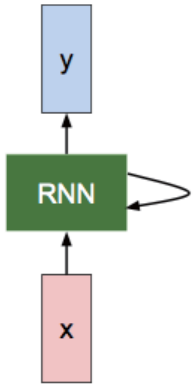


Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state — some function with parameters W — old state — input vector at some time step

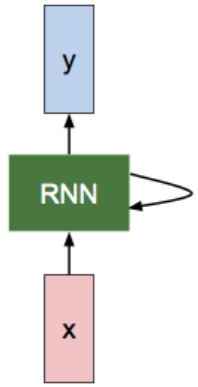


Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

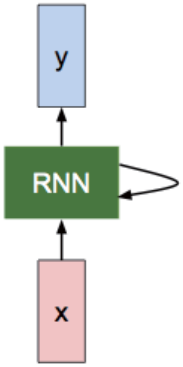
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.



(Vanilla) Recurrent Neural Network

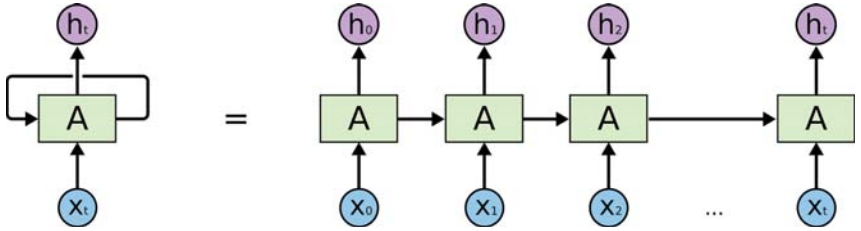
The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

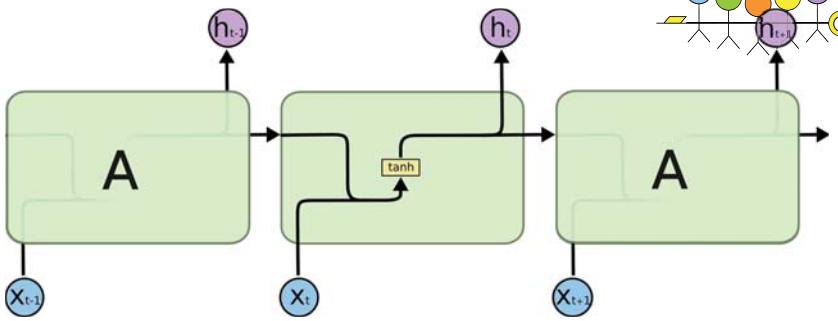
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$



Notice: the same function and the same set of parameters are used at every time step.

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



Given list of word **vectors:** $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$

At a single time step:

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

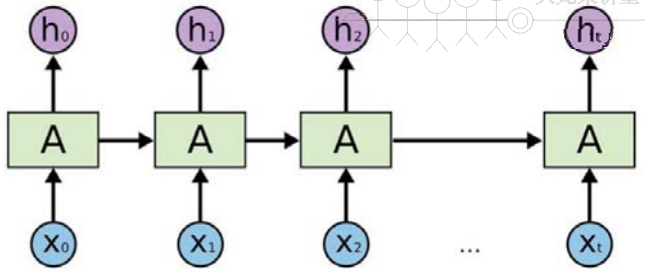
$$\hat{y}_t = \text{softmax}\left(W^{(S)} h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$

Character-level language model example

Vocabulary:
[h,e,l,o]

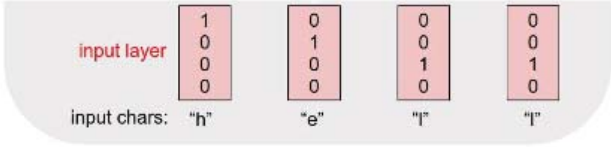
Example training sequence:
"hello"



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"

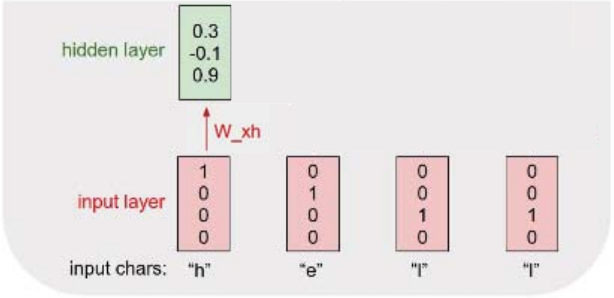


Character-level language model example

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"

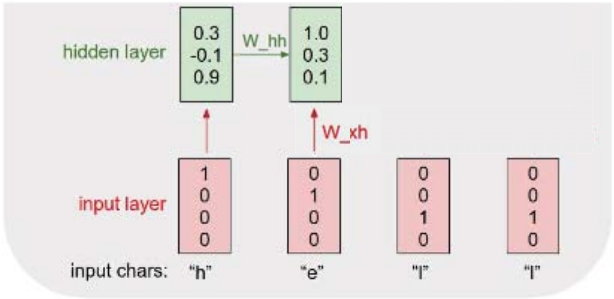


Character-level language model example

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"

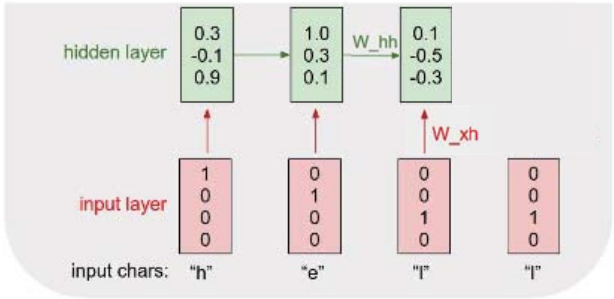


Character-level language model example

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"

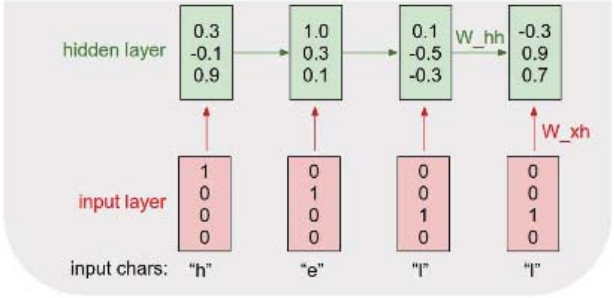


Character-level language model example

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"

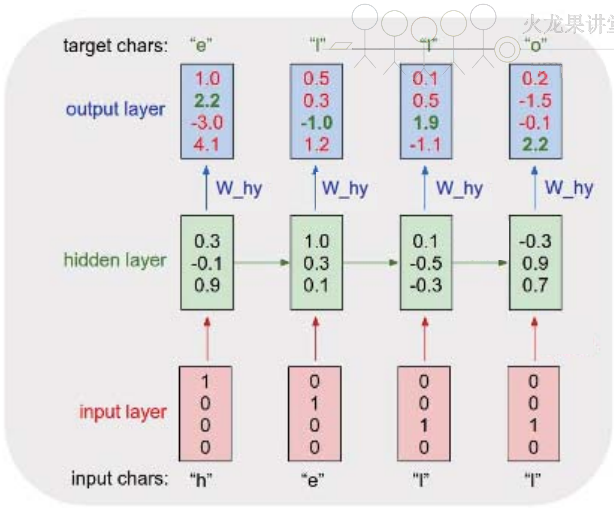


Character-level language model example

$$y_t = W_{hy}h_t$$

Vocabulary:
[h,e,l,o]

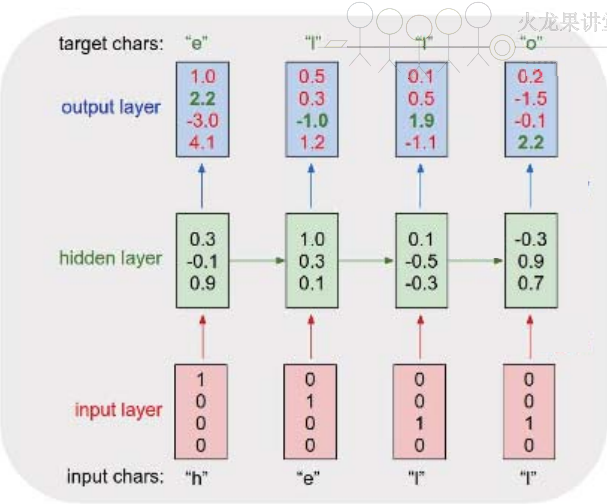
Example training sequence:
"hello"



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"



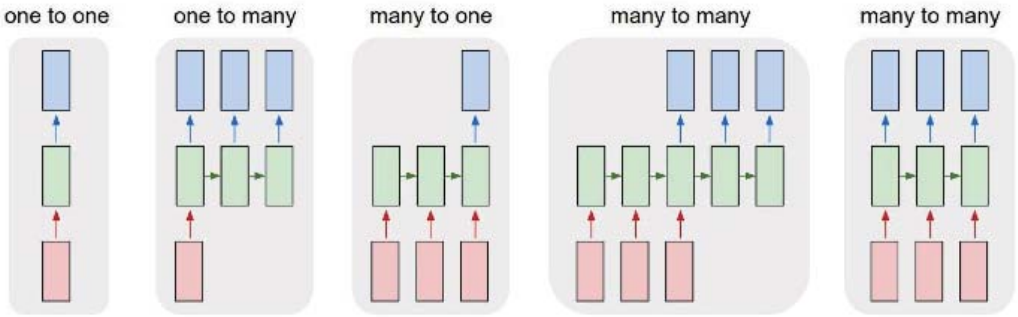
RNN applications

https://github.com/TensorFlowKR/awesome_tensorflow_implementations

- Language Modeling
- Speech Recognition
- Machine Translation
- Conversation Modeling/Question Answering
- Image/Video Captioning
- Image/Music/Dance Generation

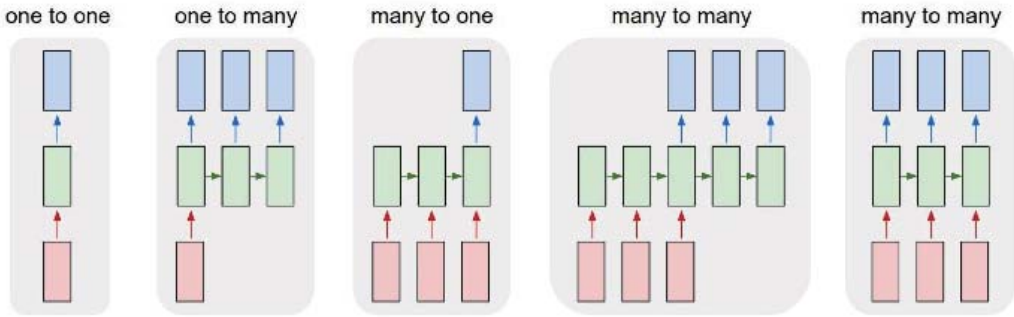
https://kjw0612.github.io/awesome-rnn/

Recurrent Networks offer a lot of flexibility:



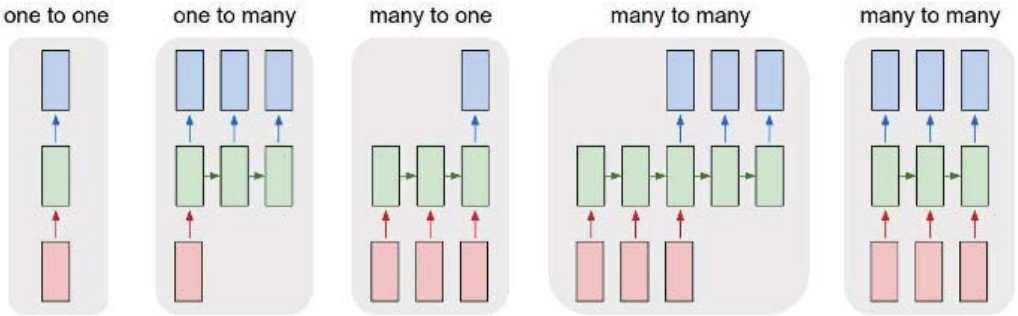
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:



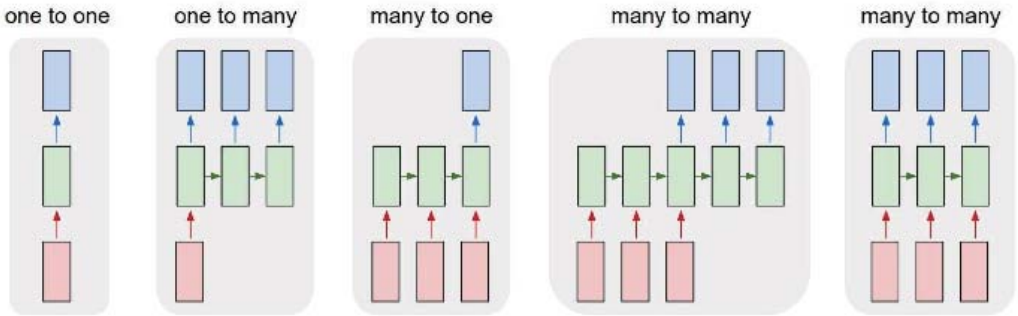
e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:



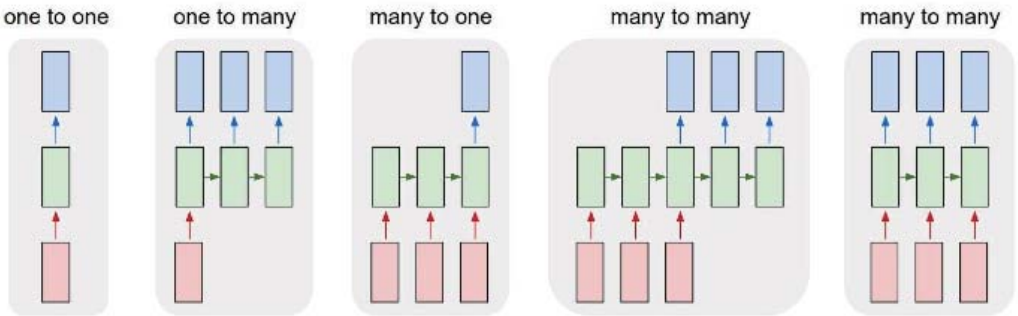
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:



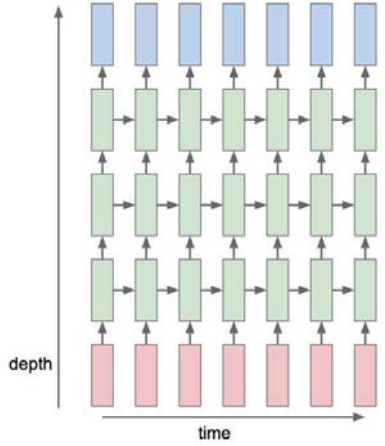
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:



e.g. **Video classification on frame level**

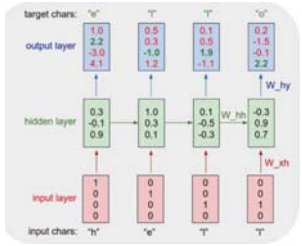
Multi-Layer RNN



Training RNNs is challenging

- Several advanced models
  - Long Short Term Memory (LSTM)
  - GRU by Cho et al. 2014

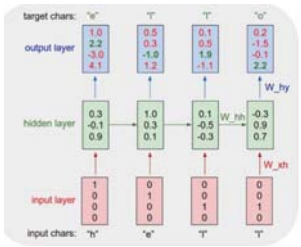
char-rnn



Shakespeare

It looks like we can learn to spell English words. But how about if there is more structure and style in the data? To examine this I downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file. We can now afford to train a larger network, in this case lets try a 3-layer RNN with 512 hidden nodes on each layer. After we train the network for a few hours we obtain samples such as:

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Linux Source Code

I wanted to push structured data to its limit, so for the final challenge I decided to use code. In particular, I took all the source and header files found in the Linux repo on Github, concatenated all of them in a single giant file (474MB of C code) (I was originally going to train only on the kernel but that by itself is only ~16MB). Then I trained several as-large-as-fits-on-my-GPU 3-layer LSTMs over a period of a few days. These models have about 10 million parameters, which is still on the lower end for RNN models. The results are superfun:

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbkregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATE << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```


Google Translate

Multilingual Machine Translation with RNN

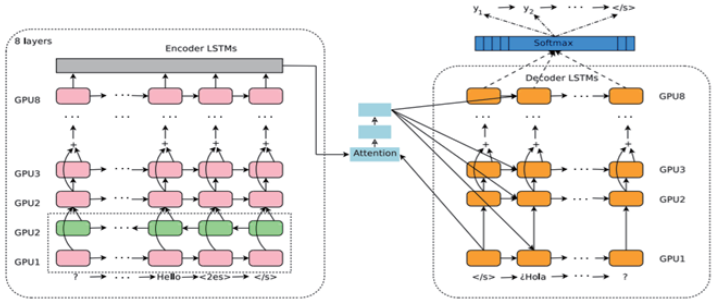


Figure 1: The model architecture of the Multilingual GNMT system. In addition to what is described in [24], our input has an artificial token to indicate the required target language. In this example, the token "<2es>" indicates that the target sentence is in Spanish, and the source sentence is reversed as a processing step. For most of our experiments we also used direct connections between the encoder and decoder although we later found out that the effect of these connections is negligible (however, once you train with those they have to be present for inference as well). The rest of the model architecture is the same as in [24].

Source: https://arxiv.org/pdf/1611.04558.pdf

Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation

Question Answering with RNN

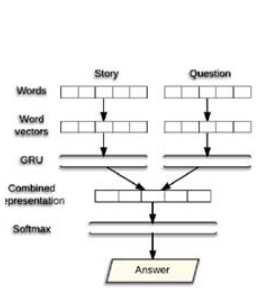


Figure 1: GRU baseline

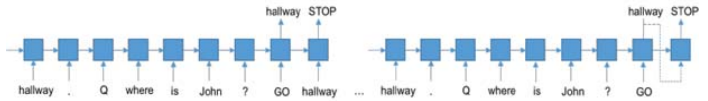


Figure 2: Sequence-to-sequence baseline: training (left); validation / testing (right)

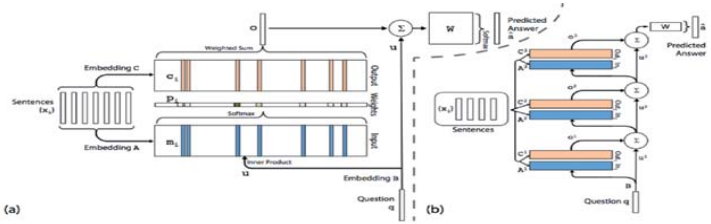


Figure 3: End to end memory network from [11]

Source: https://cs224d.stanford.edu/reports/StrohMathur.pdf

Question Answering Using Deep Learning

Handwriting generation

https://www.cs.toronto.edu/~graves/handwriting.html

Sketch-RNN

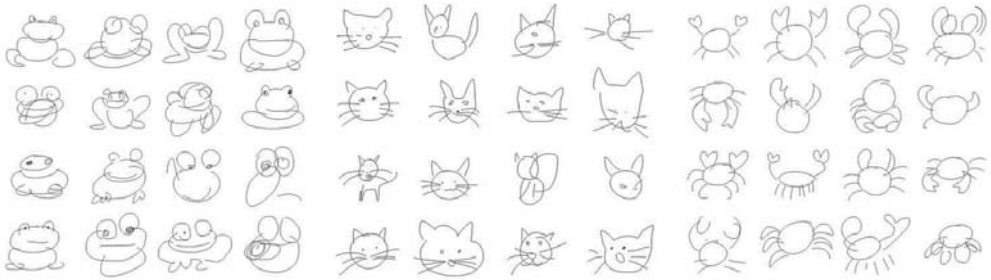


Figure 20: Unconditional generated sketches of frogs, cats, and crabs at $\tau = 0.8$.

https://arxiv.org/pdf/1704.03477v1.pdf

Speech recognition with RNN

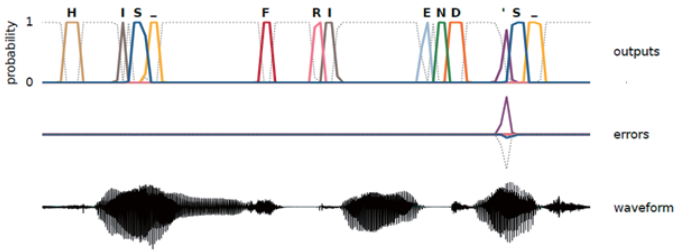


Figure 4. **Network outputs.** The figure shows the frame-level character probabilities emitted by the CTC layer (different colour for each character, dotted grey line for 'blanks'), along with the corresponding training errors, while processing an utterance. The target transcription was 'HIS_FRIENDS_', where the underscores are end-of-word markers. The network was trained with WER loss, which tends to give very sharp output decisions, and hence sparse error signals (if an output probability is 1, nothing else can be sampled, so the gradient is 0 even if the output is wrong). In this case the only gradient comes from the extraneous apostrophe before the 'S'. Note that the characters in common sequences such as 'IS', 'RI' and 'END' are emitted very close together, suggesting that the network learns them as single sounds.

Source: http://proceedings.mlr.press/v32/graves14.pdf
Towards End-to-End Speech Recognition with Recurrent Neural Networks

Generate Molecular structures with RNN

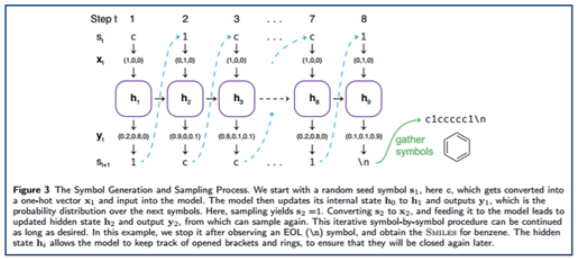


Figure 3 The Symbol Generation and Sampling Process. We start with a random seed symbol s_1, here c, which gets converted into a one-hot vector x_1 and input into the model. The model then updates its internal state h_0 to h_1 and outputs y_1, which is the probability distribution over the next symbols. Here, sampling yields s_2 =1. Converting s_2 to x_2, and feeding it to the model leads to updated hidden state h_2 and output y_2, from which can sample again. This iterative symbol-by-symbol procedure can be continued as long as desired. In this example, we stop it after observing an EOL (\n) symbol, and obtain the SMILES for benzene. The hidden state h_i allows the model to keep track of opened brackets and rings, to ensure that they will be closed again later.

Table 1 Molecules sampled during training.

Batch	Generated Example	valid
0	0c.BK5i%ur+7oAFc7L3T=F8B5e=n)CS6RCTAR((OVCp1CApb)	no
1000	OF=CCC2OCCCC)C2)C1CNC2CCCCCCCCCCCCCCCCCCCCCC	no
2000	O=C(N)C(=O)N(c1occc1OC)c2ccccc2OC	yes
3000	O=C1C=2N(c3cc(ccc3OC2CCC1)CCCc4cn(c5c(Cl)cccc54)C)C	yes

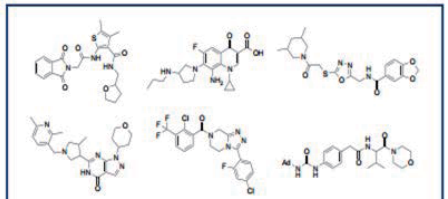


Figure 4 A few randomly selected, generated molecules. Ad = Adamantyl

Source: https://arxiv.org/pdf/1701.01329.pdf
Generating Focussed Molecule Libraries for Drug Discovery with Recurrent Neural Networks

Predict Stock prices with RNN

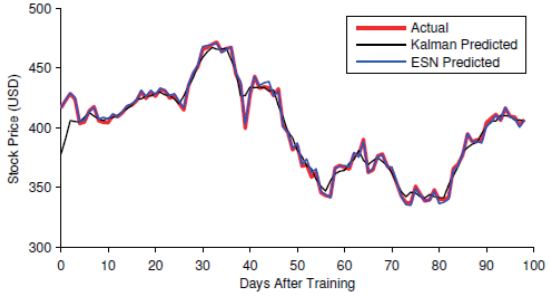


Figure 5: Google stock price prediction for ESN and Kalman filter. ESN predicts rapid changes in stock price wheras Kalman filter tends to smooth rapid changes. Test error, $\frac{(y-y_{target})^2}{\sigma^2_{y^2_{target}}}$, for the ESN is 0.0027. In contrast, test error for the Kalman filter is 0.2135.

Source: http://cs229.stanford.edu/proj2012/BernalFokPidaparthi-FinancialMarketTimeSeriesPredictionwithRecurrentNeural.pdf
Financial Market Time Series Prediction with Recurrent Neural Networks

Weather prediction with Graphical RNN

Algorithm 1 Graphical RNN model: forward pass

Require: $V, \mathcal{C}, X, Y, E, F, \mathcal{L}, \Theta, inroll = I$

$\forall u \in V, h_{u,0,I} = \vec{0} \in \mathbb{R}^d$
$loss = 0$
for $t = [1, 2, 3, \cdots T]$ **do**
  for all $u \in V$ **do**
    $h_{u,t,0} = h_{u,t-1,I}$
  for $i = [1, 2, 3, \cdots I]$ **do**
    for all $u \in V$ **do**
      for all $k \in [K]$ **do**
        $s_{u,k,t,i} = f_k(\{h_{v,t,i-1} | v \in N_{u,k}\})$
      $s_{u,t,i} = concat\left((s_{u,k,t,i})_{k=1}^{K}\right)$
      $x_{u,t,i} = concat\left(X_{g(u)}(u,t), s_{u,t,i}\right)$
    for all $u \in V$ **do**
      $(\hat{y}_{u,t,i}, h_{u,t,i}) = RNN_{\theta_{g(u)}}(x_{u,t,i}, h_{u,t,i-1})$
  for all $u \in V$ **do**
    $loss = loss + \mathcal{L}_{g(u)}(Y_{g(u)}(u,t), \hat{y}_{u,t,I})$

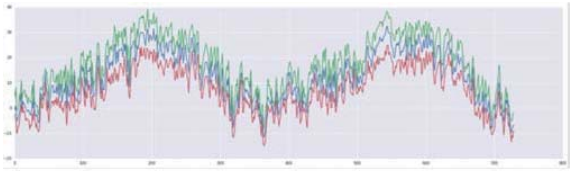


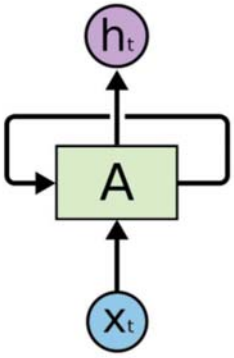
Figure 2: Sample input temperatures

Model type	Details	Train L2	Test L2
Simple	Steady state prediction	100.00	100.00
	Linear	97.22	97.34
RNN	LSTM, tanh	90.48	92.75
	iRNN, ReLU	87.62	91.01
gRNN	iRNN, ReLU, summation summary	84.41	89.38
	iRNN, ReLU, average summary	81.80	87.33

Table 2: Results on weather prediction task (% of steady state baseline).

Source: https://arxiv.org/pdf/1612.05054.pdf
Graphical RNN Models

RNN in TensorFlow

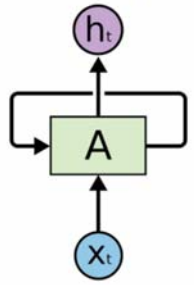



```
cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)

...
outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
```

lab-12-0-rnn_basics.ipynb

One node: 4 (*input-dim*) in 2 (*hidden_size*)

```
[[[x,x]]]
shape=(1,1,2)
```



hidden_size=2

```
[[[1,0,0,0]]]
shape=(1,1,4)
```

```
# One hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

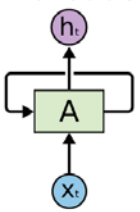
One node: 4 (*input-dim*) in 2 (*hidden_size*)

```
# One cell RNN input_dim (4) -> output_dim (2)
hidden_size = 2
cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)

x_data = np.array([[[1,0,0,0]]], dtype=np.float32)
outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)

sess.run(tf.global_variables_initializer())
pp.pprint(outputs.eval())
```

```
array([[[-0.42409304,  0.64651132]]])
```

hidden_size=2

```
[[[x,x]]]
shape=(1,1,2)
```



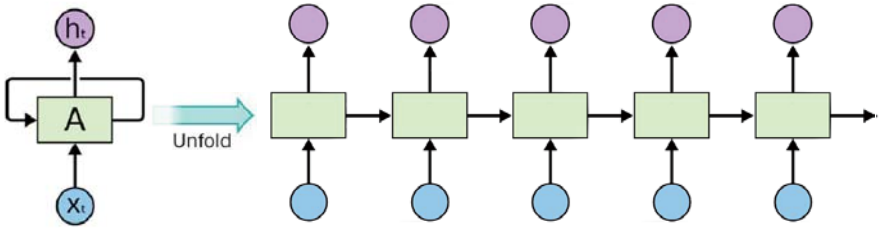

```
[[[1,0,0,0]]]
shape=(1,1,4)
```

lab-12-0-rnn_basics.ipynb

Unfolding to n sequences

Hidden_size=2
sequence_length=5

```
shape=(1,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]]]
```




```
shape=(1,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]
                      h         e         l         l         o
```

lab-12-0-rnn_basics.ipynb

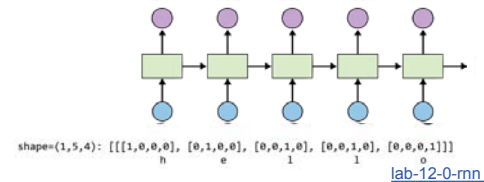
Unfolding to n sequences


```
# One cell RNN input_dim (4) -> output_dim (2). sequence: 5
    hidden_size = 2
    cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)
    x_data = np.array([[h, e, l, l, o]], dtype=np.float32)
    print(x_data.shape)
    pp.pprint(x_data)
    outputs, states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
    sess.run(tf.global_variables_initializer())
    pp.pprint(outputs.eval())
```

```
# One hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

Hidden_size=2
sequence_length=5

shape=(1,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]]]



shape=(1,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]
h e l l o

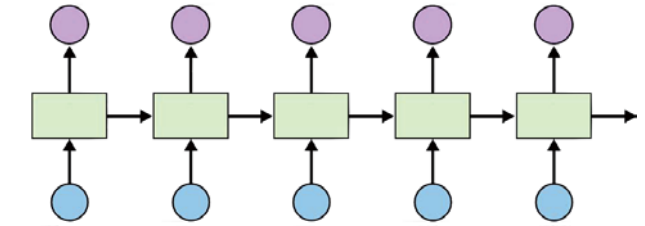
X_data = array
([[[1., 0., 0., 0.],
[0., 1., 0., 0.],
[0., 0., 1., 0.],
[0., 0., 1., 0.],
[0., 0., 0., 1.]]], dtype=float32)
Outputs = array
([[[0.19709368, 0.24918222],
[-0.11721198, 0.1784237],
[-0.35297349, -0.66278851],
[-0.70915914, -0.58334434],
[-0.38886023, 0.47304463]]], dtype=float32)

lab-12-0-rnn_basics.ipynb

Batching input

Hidden_size=2
sequence_length=5
batch_size=3

shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],
[[x,x], [x,x], [x,x], [x,x], [x,x]],
[[x,x], [x,x], [x,x], [x,x], [x,x]]]



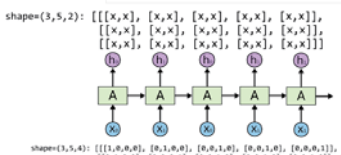
shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello
[[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]] # eolll
[[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # lleel

Batching input


```
# One cell RNN input_dim (4) -> output_dim (2). sequence: 5, batch 3
# 3 batches 'hello', 'eolll', 'lleel'
x_data = np.array([[h, e, l, l, o],
                   [e, o, l, l, l],
                   [l, l, e, e, l]], dtype=np.float32)
pp.pprint(x_data)

cell = rnn.BasicLSTMCell(num_units=2, state_is_tuple=True)
outputs, _states = tf.nn.dynamic_rnn(cell, x_data,
                                     dtype=tf.float32)
sess.run(tf.global_variables_initializer())
pp.pprint(outputs.eval())
```

```
array([[[ 1.,  0.,  0.,  0.],
        [ 0.,  1.,  0.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  0.,  1.]],

       [[ 0.,  1.,  0.,  0.],
        [ 0.,  0.,  0.,  1.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.]],

       [[ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  1.,  0.,  0.],
        [ 0.,  1.,  0.,  0.],
        [ 0.,  0.,  1.,  0.]]],
```

shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],
[[x,x], [x,x], [x,x], [x,x], [x,x]],
[[x,x], [x,x], [x,x], [x,x], [x,x]]]



shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello
[[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]] # eolll
[[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # lleel

Hidden_size=2
sequence_length=5
batch_size=3

lab-12-0-rnn_basics.ipynb

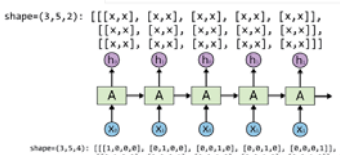
Batching input


```
# One cell RNN input_dim (4) -> output_dim (2). sequence: 5, batch 3
# 3 batches 'hello', 'eolll', 'lleel'
x_data = np.array([[h, e, l, l, o],
                   [e, o, l, l, l],
                   [l, l, e, e, l]], dtype=np.float32)
pp.pprint(x_data)

cell = rnn.BasicLSTMCell(num_units=2, state_is_tuple=True)
outputs, _states = tf.nn.dynamic_rnn(cell, x_data,
                                     dtype=tf.float32)
sess.run(tf.global_variables_initializer())
pp.pprint(outputs.eval())
```

```
array([[[ 1.,  0.,  0.,  0.],
        [ 0.,  1.,  0.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  0.,  1.]],

       [[ 0.,  1.,  0.,  0.],
        [ 0.,  0.,  0.,  1.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.]],

       [[ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  1.,  0.,  0.],
        [ 0.,  1.,  0.,  0.],
        [ 0.,  0.,  1.,  0.]]],
```

```
array([[[-0.0173022 , -0.12929453],
        [-0.14995177, -0.23189341],
        [ 0.03294011,  0.01962204],
        [ 0.12852104,  0.12375218],
        [ 0.13597946,  0.31746736]],

       [[-0.15243632, -0.14177315],
        [ 0.04586344,  0.12249056],
        [ 0.14292534,  0.15872268],
        [ 0.18998367,  0.21004884],
        [ 0.21788891,  0.24151592]],

       [[ 0.10713603,  0.11001928],
        [ 0.17076059,  0.1799853 ],
        [-0.03531617,  0.08993293],
        [-0.1881337 , -0.08296411],
        [-0.00404597,  0.07156041]]],
```

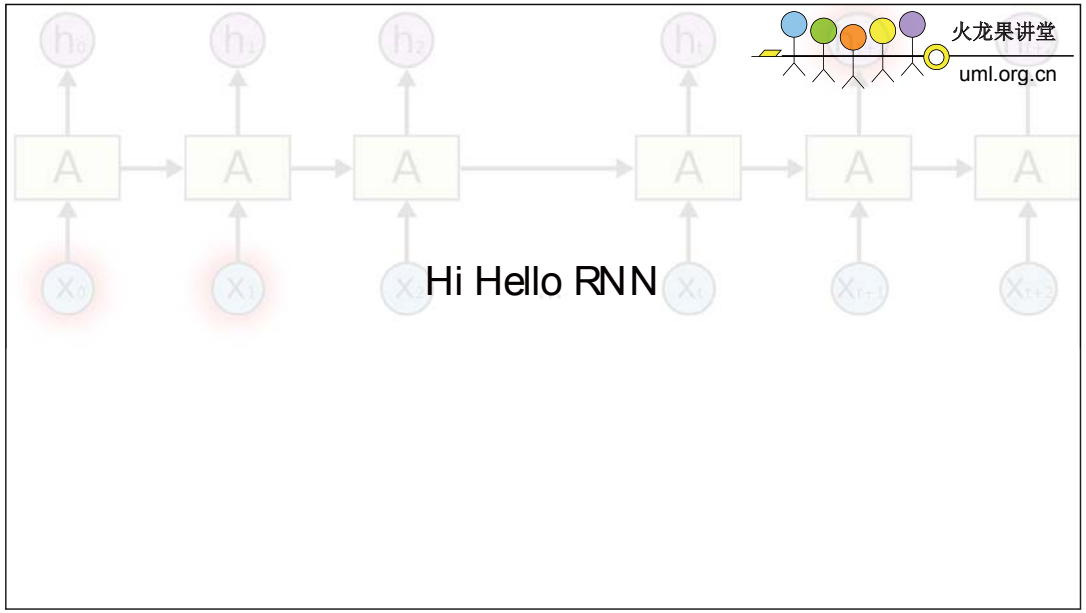
shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],
[[x,x], [x,x], [x,x], [x,x], [x,x]],
[[x,x], [x,x], [x,x], [x,x], [x,x]]]

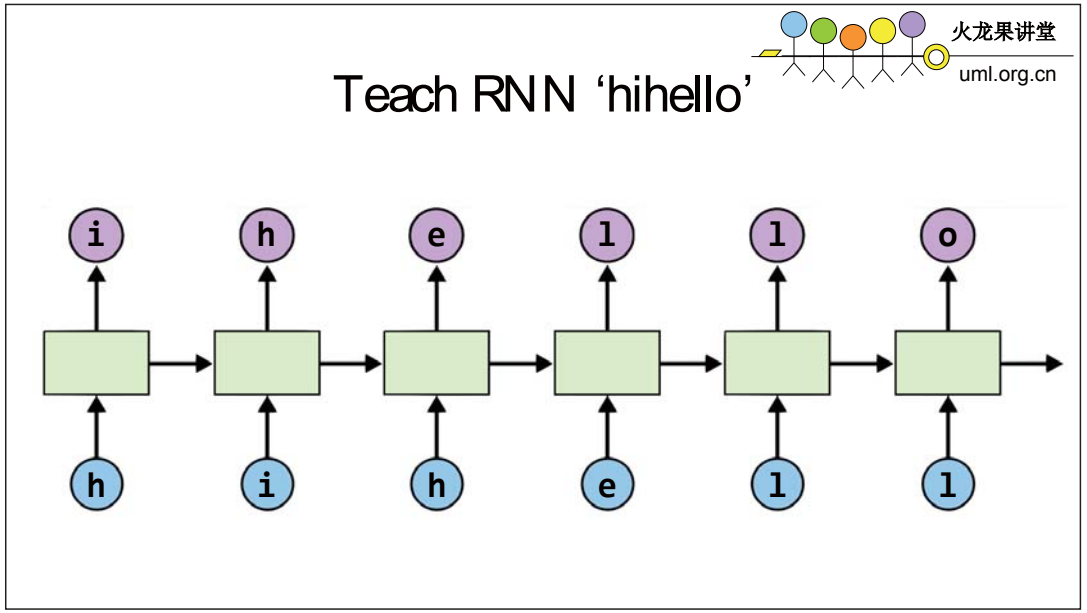


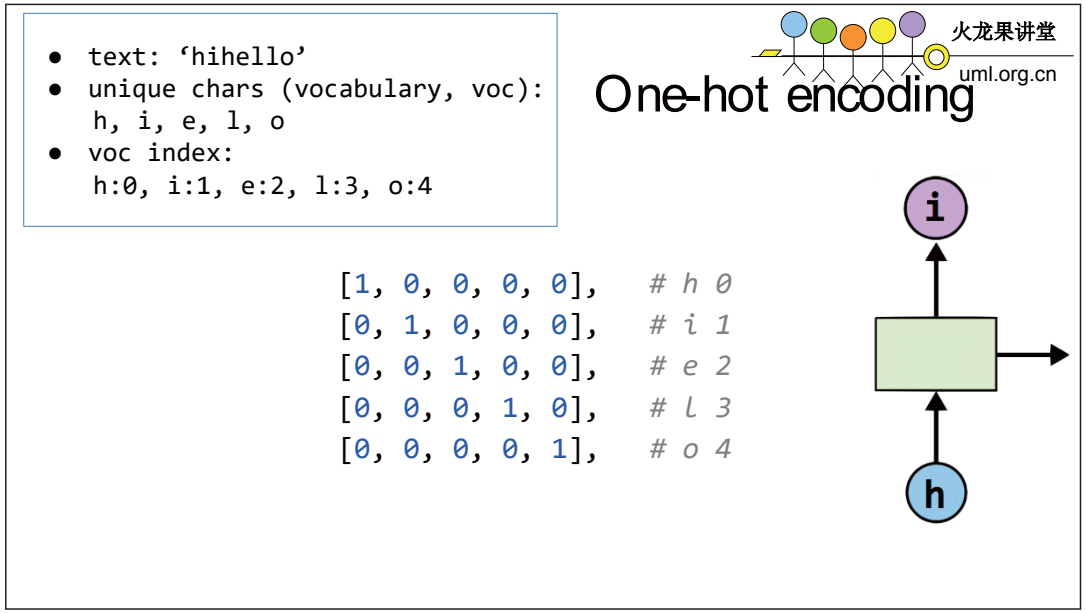
shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello
[[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]] # eolll
[[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # lleel

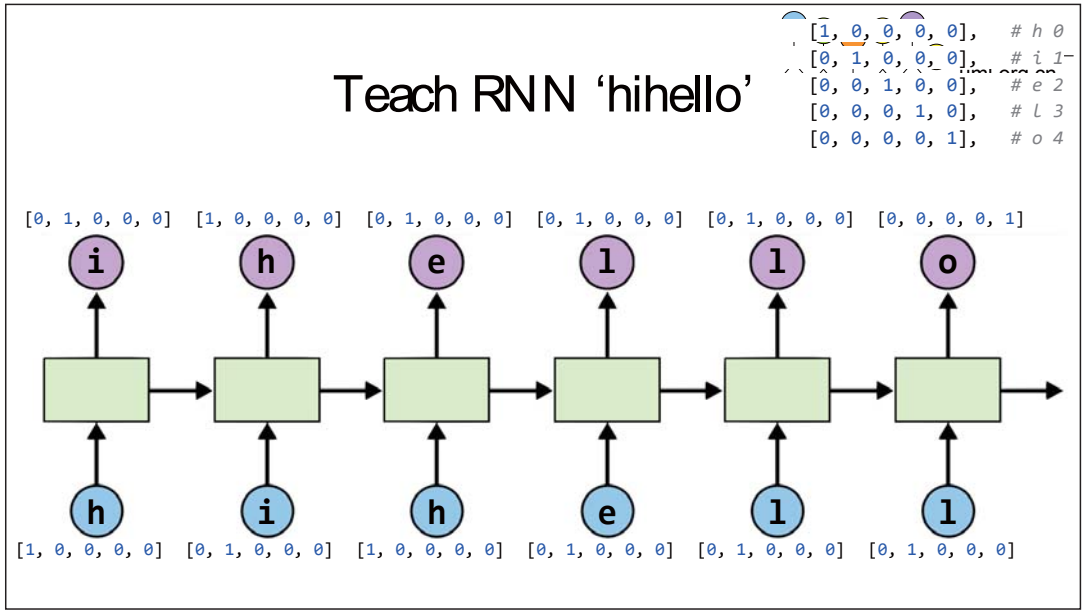
Hidden_size=2
sequence_length=5
batch_size=3

lab-12-0-rnn_basics.ipynb


火龙果讲堂
uml.org.cn
Hi Hello RNN

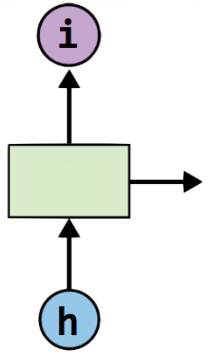

火龙果讲堂
uml.org.cn
Teach RNN ‘hihello’
i
h
e
l
l
o
h
i
h
e
l
l


text: ‘hihello’
unique chars (vocabulary, voc):
h, i, e, l, o
voc index:
h:0, i:1, e:2, l:3, o:4
火龙果讲堂
uml.org.cn
One-hot encoding
[1, 0, 0, 0, 0], # h 0
[0, 1, 0, 0, 0], # i 1
[0, 0, 1, 0, 0], # e 2
[0, 0, 0, 1, 0], # l 3
[0, 0, 0, 0, 1], # o 4
i
h


Teach RNN ‘hihello’
[1, 0, 0, 0, 0], # h 0
[0, 1, 0, 0, 0], # i 1
[0, 0, 1, 0, 0], # e 2
[0, 0, 0, 1, 0], # l 3
[0, 0, 0, 0, 1], # o 4
[0, 1, 0, 0, 0] [1, 0, 0, 0, 0] [0, 1, 0, 0, 0] [0, 1, 0, 0, 0] [0, 1, 0, 0, 0] [0, 0, 0, 0, 1]
i
h
e
l
l
o
h
i
h
e
l
l
[1, 0, 0, 0, 0] [0, 1, 0, 0, 0] [1, 0, 0, 0, 0] [0, 1, 0, 0, 0] [0, 1, 0, 0, 0] [0, 1, 0, 0, 0]

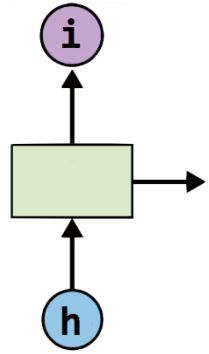
Creating rnn cell


```
# RNN model
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
```



Creating rnn cell

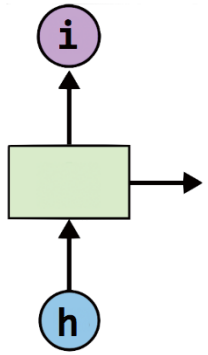

```
# RNN model
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)

rnn_cell = rnn_cell. BasicLSTMCell(rnn_size)
rnn_cell = rnn_cell. GRUCell(rnn_size)
```



Execute RNN

hidden_rnn_size

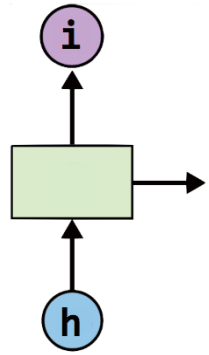
```
# RNN model
rnn_cell = rnn_cell.BasicRNNCell(rnn_siz 1

outputs, _states = tf.nn.dynamic_rnn(
                                    rnn_cell,
                    x,
                    2
        initial_state=initial_state,

        dtype=tf.float32)
```



- text: ‘hihello’
- unique chars (vocabulary, voc):
  h, i, e, l, o
- voc index:
  h:0, i:1, e:2, l:3, o:4

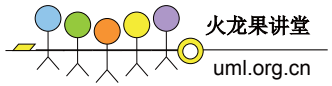
RNN parameters


```
hidden_size = 5       # output from the LSTM
input_dim = 5         # one-hot size
batch_size = 1        # one sentence
sequence_length = 6   # |ihello| == 6
```



lab-12-1-hello-rnn.py

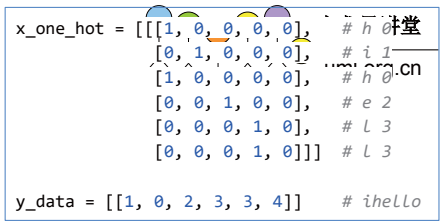
Data creation




```
idx2char = ['h', 'i', 'e', 'l', 'o'] # h=0, i=1, e=2, l=3, o=4
x_data = [[0, 1, 0, 2, 3, 3]]        # hihell
x_one_hot = [[[1, 0, 0, 0, 0],       # h 0
              [0, 1, 0, 0, 0],       # i 1
              [1, 0, 0, 0, 0],       # h 0
              [0, 0, 1, 0, 0],       # e 2
              [0, 0, 0, 1, 0],       # l 3
              [0, 0, 0, 1, 0]]]      # l 3

y_data = [[1, 0, 2, 3, 3, 4]]        # ihello
X = tf.placeholder(tf.float32,
        [None, sequence_length, input_dim]) # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label
```

lab-12-1-hello-rnn.py

Feed to RNN

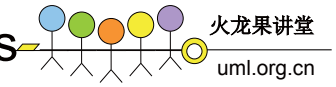



```
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 0],   # l 3
              [0, 0, 0, 1, 0]]]  # l 3

y_data = [[1, 0, 2, 3, 3, 4]]    # ihello
```

```
X = tf.placeholder(
   tf.float32, [None, sequence_length, hidden_size])  # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label

cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size,
state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
   cell, X, initial_state=initial_state, dtype=tf.float32)
```

lab-12-1-hello-rnn.py

Cost: sequence_loss

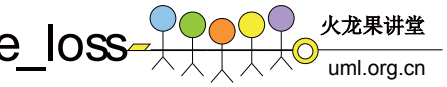



```
# [batch_size, sequence_length]
y_data = tf.constant([[1, 1, 1]])

# [batch_size, sequence_length, emb_dim ]
prediction = tf.constant([[[0, 1], [1, 0], [0, 1]]], dtype=tf.float32)

# [batch_size * sequence_length]
weights = tf.constant([[1, 1, 1]], dtype=tf.float32)

sequence_loss = tf.contrib.seq2seq.sequence_loss(prediction, y_data, weights)
sess.run(tf.global_variables_initializer())
print("Loss: ", sequence_loss.eval())
```

```
Loss:  0.646595
```

lab-12-0-rnn_basics.ipynb

Cost: sequence_loss

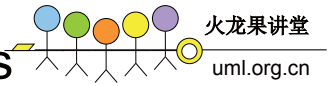



```
# [batch_size, sequence_length]
y_data = tf.constant([[1, 1, 1]])

# [batch_size, sequence_length, emb_dim ]
prediction1 = tf.constant([[[0, 1], [0, 1], [0, 1]]], dtype=tf.float32)
prediction2 = tf.constant([[[1, 0], [1, 0], [1, 0]]], dtype=tf.float32)
prediction3 = tf.constant([[[0, 1], [1, 0], [0, 1]]], dtype=tf.float32)

# [batch_size * sequence_length]
weights = tf.constant([[1, 1, 1]], dtype=tf.float32)

sequence_loss1 = tf.contrib.seq2seq.sequence_loss(prediction1, y_data, weights)
sequence_loss2 = tf.contrib.seq2seq.sequence_loss(prediction2, y_data, weights)
sequence_loss3 = tf.contrib.seq2seq.sequence_loss(prediction3, y_data, weights)

sess.run(tf.global_variables_initializer())
print("Loss1: ", sequence_loss1.eval(),
      "Loss2: ", sequence_loss2.eval(),
      "Loss3: ", sequence_loss3.eval())
```

```
Loss1:  0.313262 Loss2:  1.31326 Loss3:  0.646595
```

lab-12-0-rnn_basics.ipynb

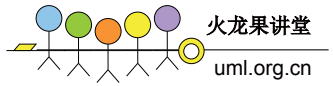
Cost: sequence_loss




```
outputs, _states = tf.nn.dynamic_rnn(
   cell, X, initial_state=initial_state, dtype=tf.float32)
weights = tf.ones([batch_size, sequence_length])

sequence_loss = tf.contrib.seq2seq.sequence_loss(
                        logits=outputs, targets=Y, weights=weights)
loss = tf.reduce_mean(sequence_loss)
train = tf.train.AdamOptimizer(learning_rate=0.1).minimize(loss)
```

lab-12-1-hello-rnn.py

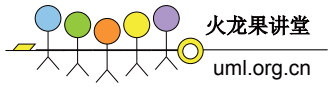
Training




```
prediction = tf.argmax(outputs, axis=2)

with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())
   for i in range(2000):
       l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
       result = sess.run(prediction, feed_dict={X: x_one_hot})
       print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

       # print char using dic
       result_str = [idx2char[c] for c in np.squeeze(result)]
       print("\tPrediction str: ", ''.join(result_str))
```

lab-12-1-hello-rnn.py

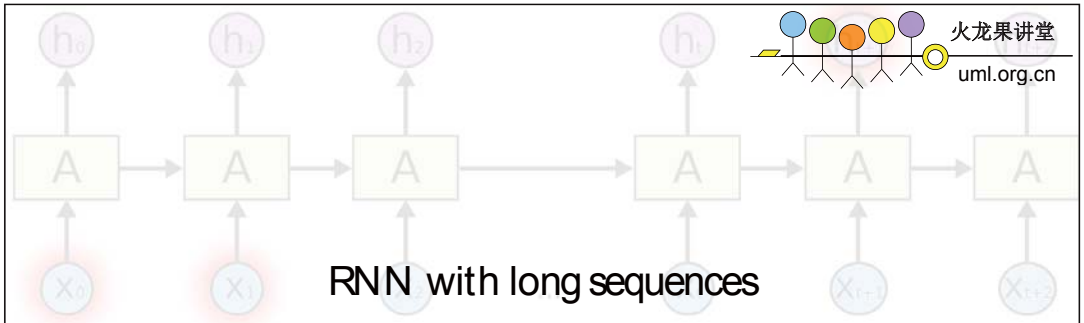
Results



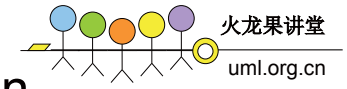

```
prediction = tf.argmax(outputs, axis=2)

with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())
   for i in range(2000):
       l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
       result = sess.run(prediction, feed_dict={X: x_one_hot})
       print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

       # print char using dic
       result_str = [idx2char[c] for c in np.squeeze(result)]
       print("\tPrediction str: ", ''.join(result_str))
```

```
0 loss: 1.55474 prediction:  [[3 3 3 3 4 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  lllloo
1 loss: 1.55081 prediction:  [[3 3 3 3 4 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  lllloo
2 loss: 1.54704 prediction:  [[3 3 3 3 4 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  lllloo
3 loss: 1.54342 prediction:  [[3 3 3 3 4 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  lllloo
...
1998 loss: 0.75305 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  ihello
1999 loss: 0.752973 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:
ihello
```

lab-12-1-hello-rnn.py



Manual data creation




```
idx2char = ['h', 'i', 'e', 'l', 'o']
x_data = [[0, 1, 0, 2, 3, 3]]    # hihell
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 0],   # l 3
              [0, 0, 0, 1, 0]]]  # l 3

y_data = [[1, 0, 2, 3, 3, 4]]    # ihello
```

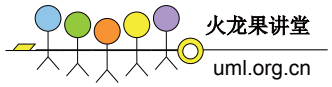
lab-12-1-hello-rnn.py

Better data creation


```
sample = " if you want you"
idx2char = list(set(sample))  # index -> char
char2idx = {c: i for i, c in enumerate(idx2char)}  # char -> idx

sample_idx = [char2idx[c] for c in sample]  # char to index
x_data = [sample_idx[:-1]]  # X data sample (0 ~ n-1) hello: hell
y_data = [sample_idx[1:]]   # Y label sample (1 ~ n)  hello: ello

X = tf.placeholder(tf.int32, [None, sequence_length])  # X data
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label

X_one_hot = tf.one_hot(X, num_classes)  # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0
```

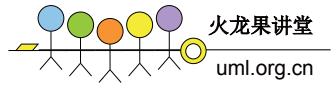
lab-12-2-char-seq-rnn.py

Hyper parameters




```
sample = " if you want you"
idx2char = list(set(sample))  # index -> char
char2idx = {c: i for i, c in enumerate(idx2char)}  # char -> idx

# hyper parameters
dic_size = len(char2idx)  # RNN input size (one hot size)
rnn_hidden_size = len(char2idx)  # RNN output size
num_classes = len(char2idx)  # final output size (RNN or softmax, etc.)
batch_size = 1  # one sample data, one batch
sequence_length = len(sample) - 1  # number of lstm unfolding (unit #)
```

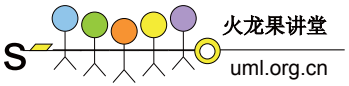
LSTM and Loss




```
X = tf.placeholder(tf.int32, [None, sequence_length])  # X data
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label

X_one_hot = tf.one_hot(X, num_classes)  # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0

cell = tf.contrib.rnn.BasicLSTMCell(num_units=rnn_hidden_size, state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
   cell, X_one_hot, initial_state=initial_state, dtype=tf.float32)

weights = tf.ones([batch_size, sequence_length])
sequence_loss = tf.contrib.seq2seq.sequence_loss(logits=outputs, targets=Y,weights=weights)
loss = tf.reduce_mean(sequence_loss)
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss)

prediction = tf.argmax(outputs, axis=2)
```

lab-12-2-char-seq-rnn.py

Training and Results




```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(3000):
        l, _ = sess.run([loss, train], feed_dict={X: x_data, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_data})
        # print char using dic
        result_str = [idx2char[c] for c in np.squeeze(result)]
        print(i, "loss:", l, "Prediction:", ''.join(result_str))
```

```
0 loss: 2.29895 Prediction: nnuffuunnuuuyuy
1 loss: 2.29675 Prediction: nnuffuunnuuuyuy
...

1418 loss: 1.37351 Prediction: if you want you
1419 loss: 1.37331 Prediction: if you want you
```

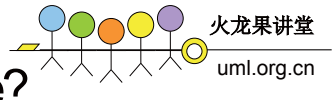
lab-12-2-char-seq-rnn.py

Really long sentence?


```
sentence = ("if you want to build a ship, don't drum up people together to "
           "collect wood and don't assign them tasks and work, but rather "
           "teach them to long for the endless immensity of the sea.")
```

lab-12-4-rnn_long_char.py

Really long sentence?




```
sentence = ("if you want to build a ship, don't drum up people together to "
           "collect wood and don't assign them tasks and work, but rather "
           "teach them to long for the endless immensity of the sea.")
```

```
# training dataset
0 if you wan -> f you want
1 f you want ->  you want
2  you want  -> you want t
3 you want t -> ou want to

168  of the se -> of the sea
169 of the sea -> f the sea.
```

lab-12-4-rnn_long_char.py

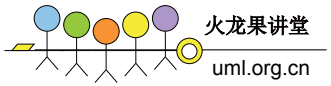
Making dataset


```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}

dataX = []
dataY = []
for i in range(0, len(sentence) - seq_length):
    x_str = sentence[i:i + seq_length]
    y_str = sentence[i + 1: i + seq_length + 1]
    print(i, x_str, '->', y_str)

    x = [char_dic[c] for c in x_str]  # x str to index
    y = [char_dic[c] for c in y_str]  # y str to index

    dataX.append(x)
    dataY.append(y)
```

```
# training dataset
0 if you wan -> f you want
1 f you want ->  you want
2  you want  -> you want t
3 you want t -> ou want to

168  of the se -> of the sea
169 of the sea -> f the sea.
```

lab-12-4-rnn_long_char.py

RNN parameters



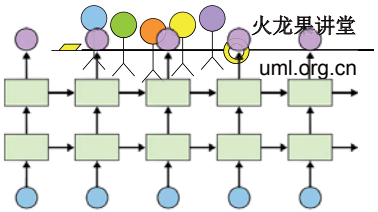

```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}

data_dim = len(char_set)
hidden_size = len(char_set)
num_classes = len(char_set)
seq_length = 10  # Any arbitrary number

batch_size = len(dataX)
```

 # training dataset
0 if you wan -> f you want
1 f you want -> you want
2 you want -> you want t
3 you want t -> ou want to

168 of the se -> of the sea
169 of the sea -> f the sea.

lab-12-4-rnn_long_char.py

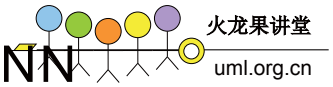
Stacked RNN




```
X = tf.placeholder(tf.int32, [None, seq_length])
Y = tf.placeholder(tf.int32, [None, seq_length])

# One-hot encoding
X_one_hot = tf.one_hot(X, num_classes)
print(X_one_hot)  # check out the shape

# Make a lstm cell with hidden_size (each unit output vector size)
cell = rnn.BasicLSTMCell(hidden_size, state_is_tuple=True)
cell = rnn.MultiRNNCell([cell] * 2, state_is_tuple=True)

# outputs: unfolding size x hidden size, state = hidden size
outputs, _states = tf.nn.dynamic_rnn(cell, X_one_hot, dtype=tf.float32)
```

lab-12-4-rnn_long_char.py

Softmax (FC) in Deep CNN



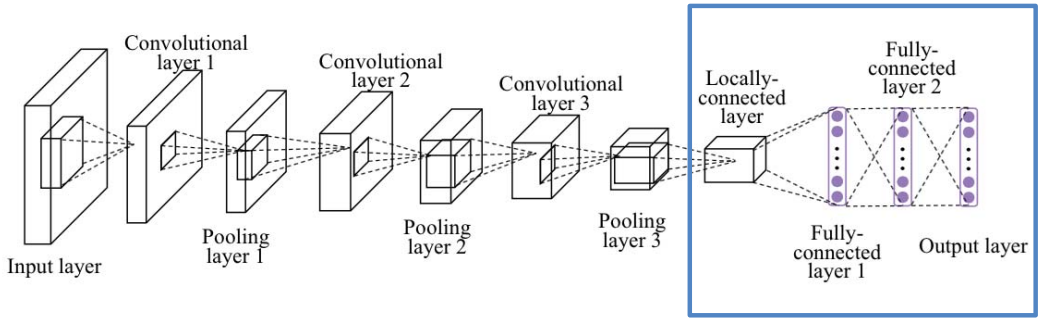


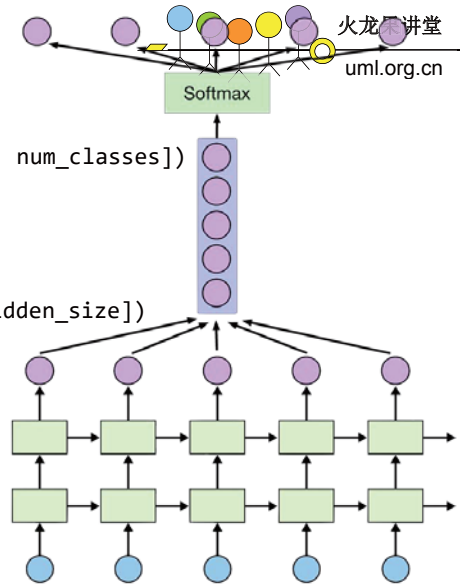
Image credit: http://personal.ie.cuhk.edu.hk/~ccloy/project_target_code/index.html

Softmax


```
outputs, _ = tf.nn.dynamic_rnn(cell, X_one_hot)
```


lab-12-4-rnn_long_char.py

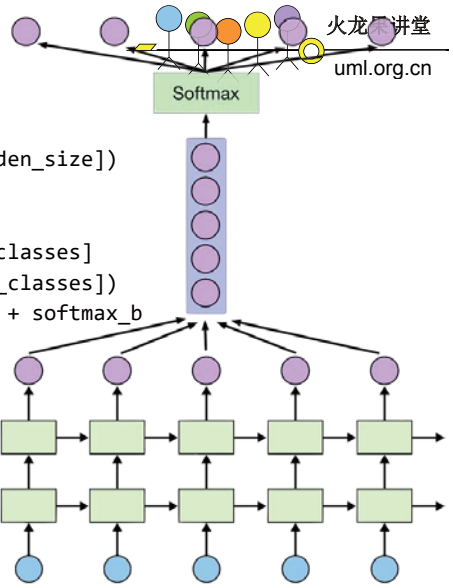
Softmax




```
outputs = tf.reshape(outputs,
     [batch_size, seq_length, num_classes])

X_for_softmax = tf.reshape(outputs,
                           [-1, hidden_size])

outputs, _ = tf.nn.dynamic_rnn(cell, X_one_hot)
```

lab-12-4-rnn_long_char.py

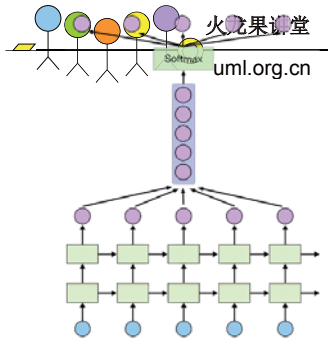
Softmax




```
# (optional) softmax layer
X_for_softmax = tf.reshape(outputs, [-1, hidden_size])

softmax_w = tf.get_variable("softmax_w",
                            [hidden_size, num_classes]
softmax_b = tf.get_variable("softmax_b",[num_classes])
outputs = tf.matmul(X_for_softmax,softmax_w) + softmax_b

outputs = tf.reshape(outputs,
      [batch_size, seq_length, num_classes])
```

lab-12-4-rnn_long_char.py

Loss



```
# reshape out for sequence_loss
outputs = tf.reshape(outputs,
          [batch_size, seq_length, num_classes])
# All weights are 1 (equal weights)
weights = tf.ones([batch_size, seq_length])

sequence_loss = tf.contrib.seq2seq.sequence_loss(
   logits=outputs, targets=Y, weights=weights)
mean_loss = tf.reduce_mean(sequence_loss)

train_op =
       tf.train.AdamOptimizer(learning_rate=0.1).minimize(mean_loss)
```

lab-12-4-rnn_long_char.py

Training and print results

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

for i in range(500):
   _, l, results = sess.run(
              [train_op, mean_loss, outputs],
              feed_dict={X: dataX, Y: dataY})

   for j, result in enumerate(results):
       index = np.argmax(result, axis=1)
       print(i, j, ''.join([char_set[t] for t in index]), l)
```

```
0 167 ttttttttt 3.23111
0 168 ttttttttt 3.23111
0 169 ttttttttt 3.23111

499 167 oof the se 0.229306
499 168 tf the sea 0.229306
499 169 n the sea. 0.229306
```

lab-12-4-rnn_long_char.py

Training and print results

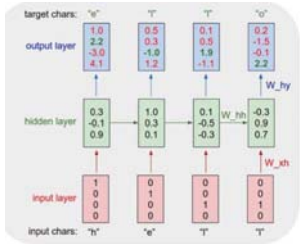
```
# Let's print the last char of each result to check it works
results = sess.run(outputs, feed_dict={X: dataX})
for j, result in enumerate(results):
    index = np.argmax(result, axis=1)
    if j is 0:  # print all for the first result to make a sentence
        print(''.join([char_set[t] for t in index]), end='')
    else:
        print(char_set[index[-1]], end='')
```

g you want to build a ship, don't drum up people together to collect wood and don't assign them tasks and work, but rather teach them to long for the endless immensity of the sea.

lab-12-4-rnn_long_char.py

火龙果讲堂
uml.org.cn

char-rnn

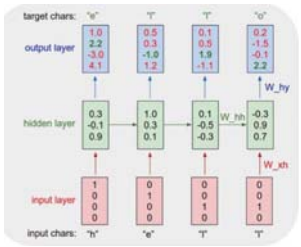


Shakespeare

It looks like we can learn to spell English words. But how about if there is more structure and style in the data? To examine this I downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file. We can now afford to train a larger network, in this case lets try a 3-layer RNN with 512 hidden nodes on each layer. After we train the network for a few hours we obtain samples such as:

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

火龙果讲堂



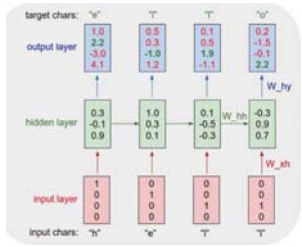
Linux Source Code

I wanted to push structured data to its limit, so for the final challenge I decided to use code. In particular, I took all the source and header files found in the Linux repo on Github, concatenated all of them in a single giant file (474MB of C code) (I was originally going to train only on the kernel but that by itself is only ~16MB). Then I trained several as-large-as-fits-on-my-GPU 3-layer LSTMs over a period of a few days. These models have about 10 million parameters, which is still on the lower end for RNN models. The results are superfun:

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbkregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATE << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

火龙果讲堂
uml.org.cn

char/word rnn (char/word level n to n model)



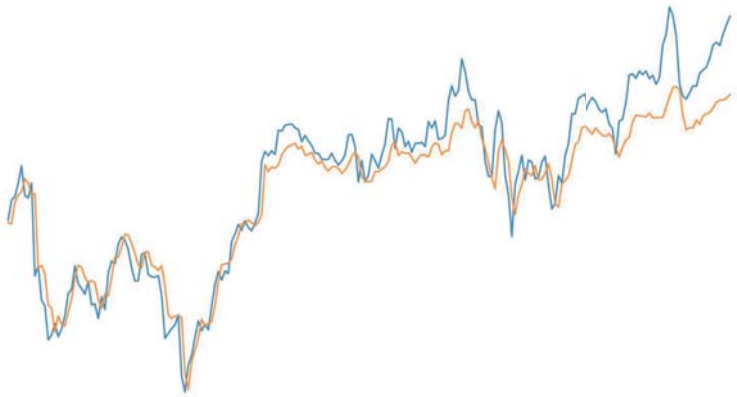
char-rnn-tensorflow

word-rnn-tensorflow

RNN with time series data (stock)

火龙果讲堂
uml.org.cn

Time series data



火龙果讲堂
uml.org.cn

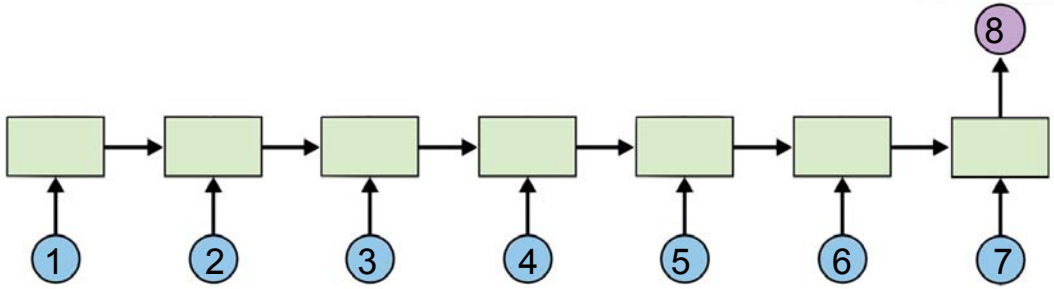
Time series data

Open	High	Low	Volume	Close
828.659973	833.450012	828.349976	1247700	**831.659973**
823.02002	828.070007	821.655029	1597800	**828.070007**
819.929993	824.400024	818.97998	1281700	**824.159973**
819.359985	823	818.469971	1304000	**818.97998**
819	823	816	1053600	**820.450012**
816	820.958984	815.48999	1198100	**819.23999**
811.700012	815.25	809.780029	1129100	**813.669983**
809.51001	810.659973	804.539978	989700	**809.559998**
807	811.840027	803.190002	1155300	**808.380005**

`'data-02-stock_daily.csv'`

火龙果讲堂
uml.org.cn

Many to one



lab-12-5-rnn_stock_prediction.py

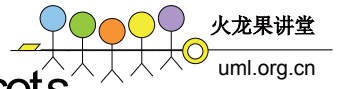
Reading data


```
timesteps = seq_length = 7
data_dim = 5
output_dim = 1
# Open,High,Low,Close,Volume
xy = np.loadtxt('data-02-stock_daily.csv', delimiter=',')
xy = xy[::-1]  # reverse order (chronically ordered)
xy = MinMaxScaler(xy)
x = xy
y = xy[:, [-1]]  # Close as label

dataX = []
dataY = []
for i in range(0, len(y) - seq_length):
    _x = x[i:i + seq_length]
    _y = y[i + seq_length]  # Next close price
    print(_x, "->", _y)
    dataX.append(_x)
    dataY.append(_y)
```

```
[ 0.18667876  0.20948057  0.20878184  0.
  0.21744815]

[ 0.30697388  0.31463414  0.21899367
  0.01247647  0.21698189]

[ 0.21914211  0.26390721  0.2246864
  0.45632338  0.22496747]

[ 0.23312993  0.23641916  0.16268272
  0.57017119  0.14744274]

[ 0.13431201  0.15175877  0.11617252
  0.39380658  0.13289962]

[ 0.13973232  0.17060429  0.15860382
  0.28173344  0.18171679]

[ 0.18933069  0.20057799  0.19187983
  0.29783096  0.2086465 ]]

-> [ 0.14106001]
```

lab-12-5-rnn_stock_prediction.py

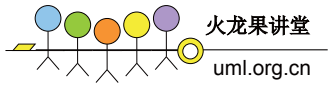
Training and test datasets




```
# split to train and testing
train_size = int(len(dataY) * 0.7)
test_size = len(dataY) - train_size
trainX, testX = np.array(dataX[0:train_size]),
                np.array(dataX[train_size:len(dataX)])
trainY, testY = np.array(dataY[0:train_size]),
                np.array(dataY[train_size:len(dataY)])

# input placeholders
X = tf.placeholder(tf.float32, [None, seq_length, data_dim])
Y = tf.placeholder(tf.float32, [None, 1])
```

lab-12-5-rnn_stock_prediction.py

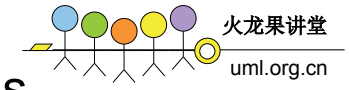
LSTM and Loss



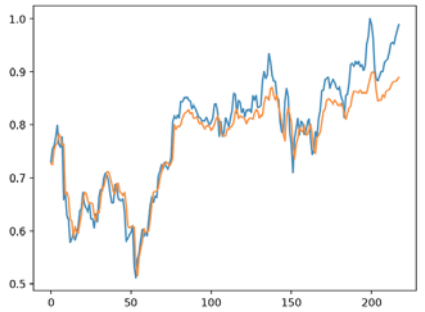

```
# input placeholders
X = tf.placeholder(tf.float32, [None, seq_length, data_dim])
Y = tf.placeholder(tf.float32, [None, 1])

cell = tf.contrib.rnn.BasicLSTMCell(num_units=output_dim, state_is_tuple=True)
outputs, _states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)
Y_pred = outputs[:, -1]  # We use the last cell's output

# cost/loss
loss = tf.reduce_sum(tf.square(Y_pred - Y))  # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

lab-12-5-rnn_stock_prediction.py

Training and Results




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

for i in range(1000):
    _, l = sess.run([train, loss],
            feed_dict={X: trainX, Y: trainY})
    print(i, l)

testPredict = sess.run(Y_pred, feed_dict={X: testX})

import matplotlib.pyplot as plt
plt.plot(testY)
plt.plot(testPredict)
plt.show()
```



lab-12-5-rnn_stock_prediction.py

Exercise

- Implement stock prediction using linear regression only
- Improve stock prediction results using more features such as keywords and/or sentiments in top news

- Implement sequence to sequence mode (translate a language to another)
  - Keras/klab-12-5-seq2seq.py

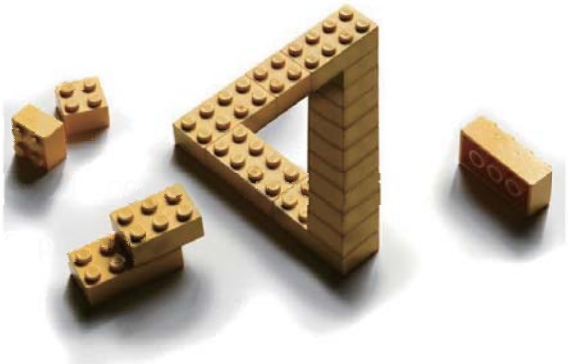
火龙果讲堂
uml.org.cn

Deep Learning

- DNN
- CNN
- RNN

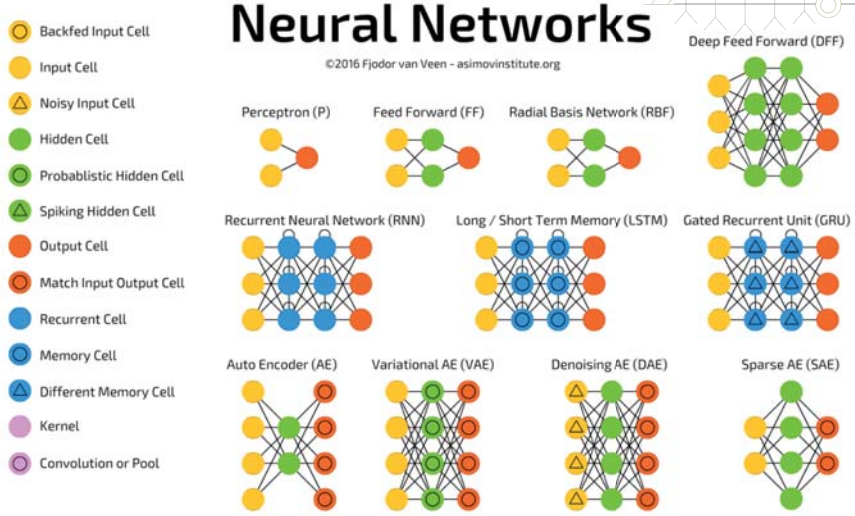
火龙果讲堂

'The only limit is your imagination'

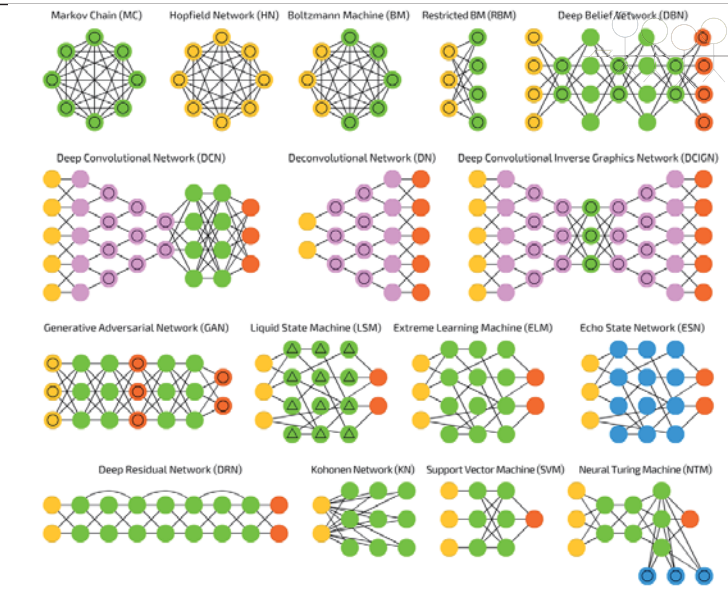


http://itchyi.squarespace.com/thelatest/2012/5/17/the-only-limit-is-your-imagination.html

火龙果讲堂
org.cn



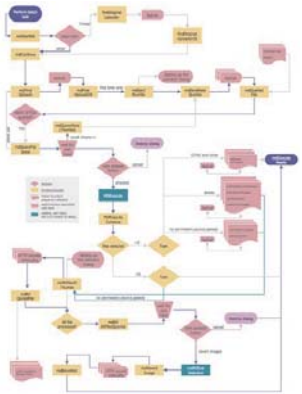
http://www.asimovinstitute.org/neural-network-zoo/



http://www.asimovinstitute.org/neural-network-zoo/

火龙果讲堂
uml.org.cn

Rule based programming VS data driven learning



火龙果讲堂
uml.org.cn

Backup Slides

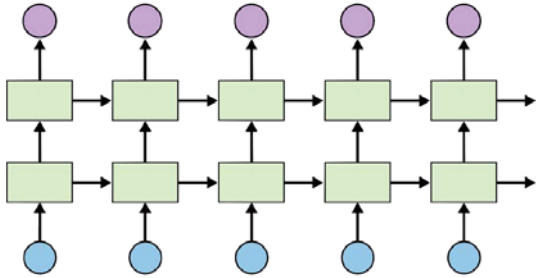
火龙果讲堂
uml.org.cn

With TF 1.0!

Dynamic RNN

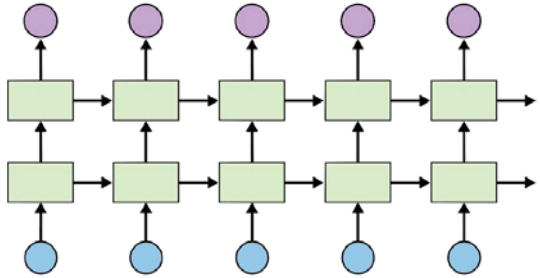
火龙果讲堂
uml.org.cn

Different sequence length



h	e	l	l	o
h	i			
w	h	y		
...				

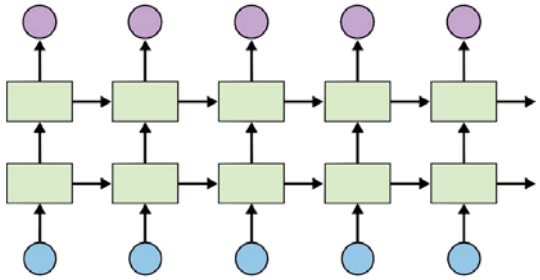
Different sequence length



h	e	l	l	o
h	i	<pad>	<pad>	<pad>
w	h	y	<pad>	<pad>
...				

Different sequence length

```
sequence_length=[5,2,3]
```



h	e	l	l	o
h	i			
w	h	y		
...				

Dynamic RNN

```
# 3 batches 'hello', 'eolll', 'lleel'
x_data = np.array([[[...]]], dtype=np.float32)

hidden_size = 2
cell = rnn.BasicLSTMCell(num_units=hidden_size,
                                 state_is_tuple=True)
outputs, _states = tf.nn.dynamic_rnn(
        cell, x_data, sequence_length=[5,3,4],
        dtype=tf.float32)
sess.run(tf.global_variables_initializer())
print(outputs.eval())
```

```
array([[[-0.17904168, -0.08053244],
        [-0.01294809,  0.01660814],
        [-0.05754048, -0.1368292 ],
        [-0.08655578, -0.20553185],
        [ 0.07297077, -0.21743253]],

       [[ 0.10272847,  0.06519825],
        [ 0.20188759, -0.05027055],
        [ 0.09514933, -0.16452041],
        [ 0.        ,  0.        ],
        [ 0.        ,  0.        ]],

       [[-0.04893036, -0.14655617],
        [-0.07947272, -0.20996611],
        [ 0.06466491, -0.02576563],
        [ 0.15087658,  0.05166111],
        [ 0.        ,  0.        ]]],
```

lab-12-0-rnn_basics.ipynb