2017 UNITEC-NTHU Summer School
on the Frontier of Information Technology

Deep Learning Lab (Prof. Min Sun)
-- TensorFlow Tutorial

Speaker: Tz-Ying (Gina) Wu

Outline

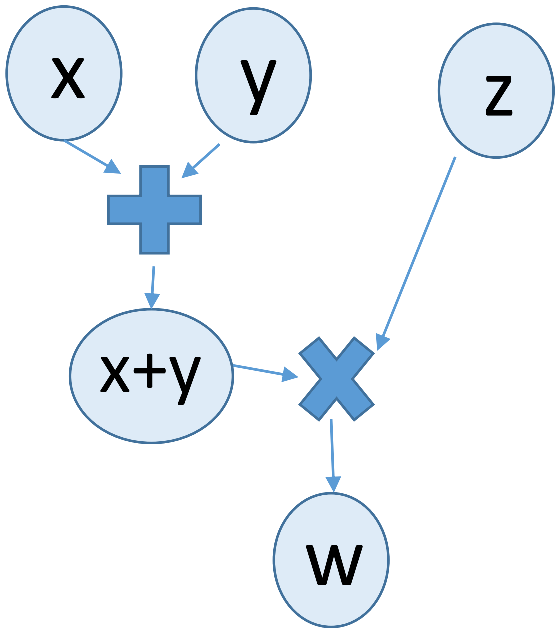
- Introduction to TensorFlow

- TensorFlow sample codes
  - Linear regression
  - Neural Network (classification)

- Exercise
  - CNN
  - RNN

Introduction to TensorFlow

What is TensorFlow?



TensorFlow

data graph

multidimensional data array

computation using data flow graphs

What is TensorFlow?

- TensorFlow is a ***deep learning*** library open-sourced by Google in 2015

- provides primitives for defining functions on tensors and automatically computing their derivatives

  You don't need to write backpropagation by yourself

- Support CPU-only, GPU usage

To write a TensorFlow program, we need to …

- Build a graph (define your model)
- Create a session
- Run the session
  - Initialize variables (if there are variables in the graph)
  - Feed the data
  - Run the graph

To write a TensorFlow program, we need to …

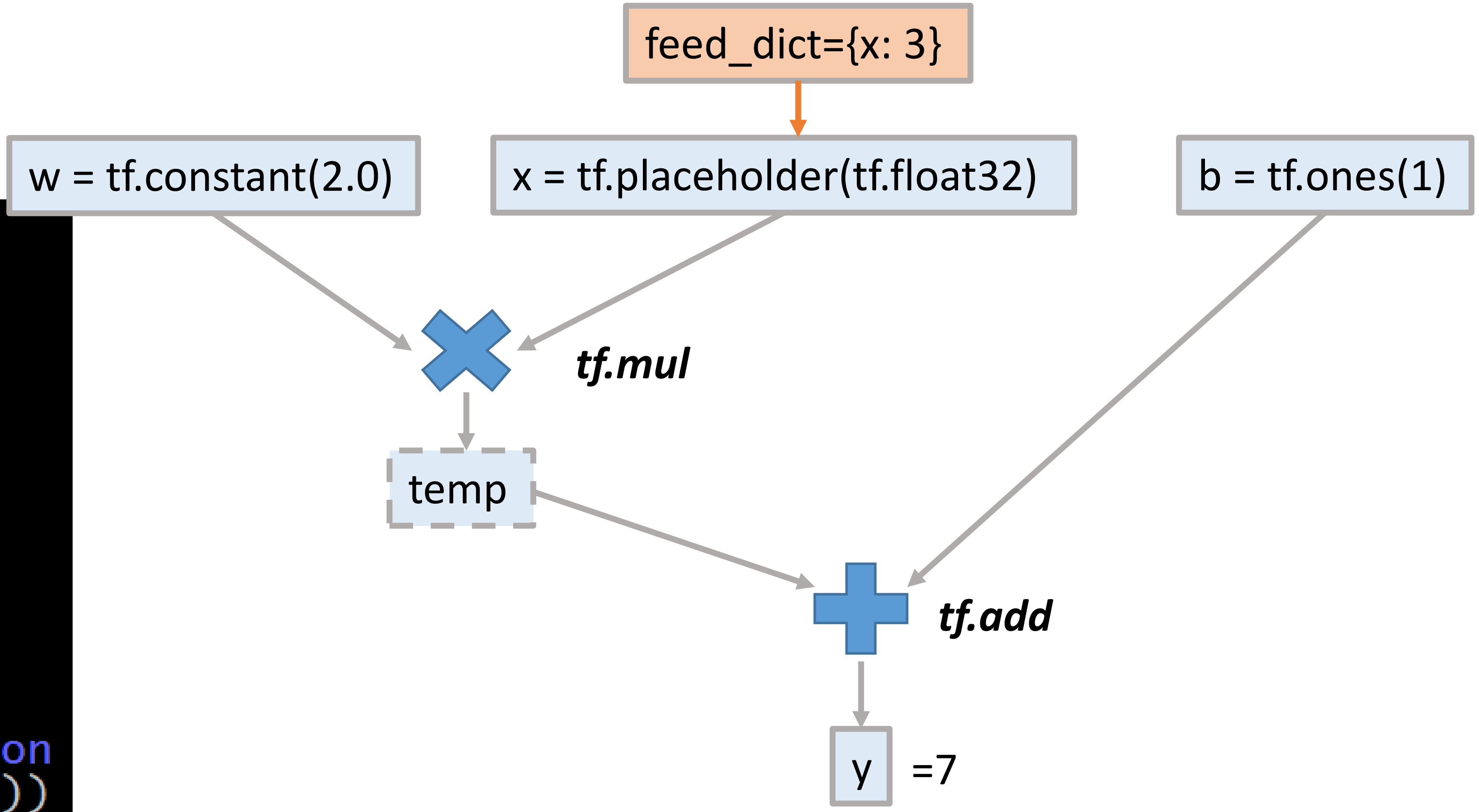
- Build a graph (define your model)
- Create a session
- Run the session
  - Initialize variables (if there are variables in the graph)
  - Feed the data
  - Run the graph

Build a graph (define your model)

- All the computations in TensorFlow graph are tensor operations

E.g. $\boldsymbol{y} = \boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}$

```
import tensorflow as tf

# build the graph
w = tf.constant(2.0)
x = tf.placeholder(tf.float32)
b = tf.ones(1)

# y = w*x+b
y = tf.add(tf.mul(w, x), b)

# create a session
sess = tf.Session()
# feed the data, and run the session
print(sess.run(y, feed_dict={x: 3}))
```



Build a graph (define your model)

- Tensors can be declared by various ways, e.g.,
  - *tf.zeros*((2,2)), *tf.ones*((1, 2, 3)) $\longleftrightarrow$ np.zeros((2, 2), np.ones((1, 2, 3))
  - *tf.constant*([2, 3]) $\longleftrightarrow$ np.array([2, 3])
  - *tf.Variable*(*tf.zeros*((2,2)), name="weights") $\longleftrightarrow$ np.array([[0, 0],[0, 0]])
  - *tf.placeholder*(*tf.float32*, shape=(10, 1))

  etc.

- Variables should be initialized before running
- ***tf.placeholder()*** is to reserve the place for input data

To write a TensorFlow program, we need to …

- Build a graph (define your model)
- Create a session
- Run the session
  - Initialize variables (if there are variables in the graph)
  - Feed the data
  - Run the graph

Create a session

- “A **Session** object encapsulates the environment in which **Operation** objects are executed, and **Tensor** objects are evaluated.” - TensorFlow Docs

- Use ***tf.Session()*** or ***tf.InteractiveSession()*** to create a session

```
sess = tf.Session()
…
sess.close()
```

or

```
with tf.Session() as sess:
    …
```

To write a TensorFlow program, we need to …

- Build a graph (define your model)
- Create a session
- Run the session
  - Initialize variables (if there are variables in the graph)
  - Feed the data
  - Run the graph

Initialize variables

- “The **Variable()** constructor requires an initial value for the variable, which can be a **Tensor** of any type and shape. ” - TensorFlow Docs

- Declaration: ***tf.Variable(<initial-value>, name=<optional-name>)***
  - ***<initial-value>*** can be a fixed-value tensor or be random initialized from a distribution
  - E.g. *tf.Variable*(*tf.zeros*((2,2)), name="weights")
  - E.g. *tf.Variable(tf.random_uniform([100, 2], -1.0, 1.0))*

- Initialization: ***sess.run(tf.initialize_all_variables())***
- [Optional: restore parameters from a TensorFlow model (use Saver)]

Feed the data

- Tensorflow provide ***feed_dict*** as the bridge between **numpy array** and **tensor**
- Usage: *feed_dict={<placeholder_name>: <numpy_array>}*
- e.g.

declaring placeholder when building the graph

The shape of the placeholder and the data fed in must be same!!!

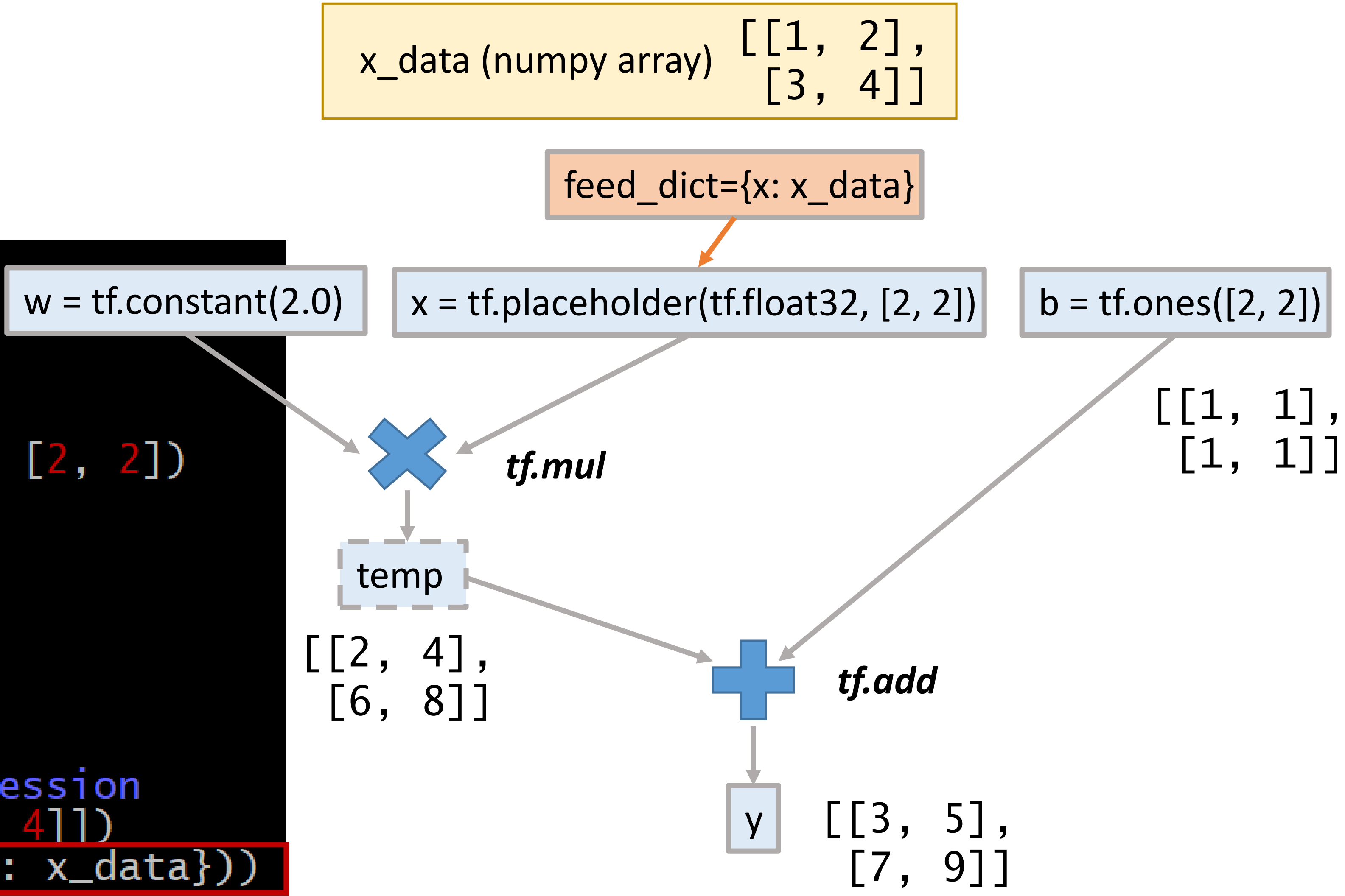
feed the data into the placeholder when running the session

```
import tensorflow as tf
import numpy as np

# build the graph
w = tf.constant(2.0)
x = tf.placeholder(tf.float32, [2, 2])
b = tf.ones([2, 2])

# y = w*x+b
y = tf.add(tf.mul(w, x), b)

# create a session
sess = tf.Session()
# feed the data, and run the session
x_data = np.array([[1, 2], [3, 4]])
print(sess.run(y, feed_dict={x: x_data}))
```

Feed the data

E.g. $\boldsymbol{y} = \boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}$

```python
import tensorflow as tf
import numpy as np

# build the graph
w = tf.constant(2.0)
x = tf.placeholder(tf.float32, [2, 2])
b = tf.ones([2, 2])

# y = w*x+b
y = tf.add(tf.mul(w, x), b)

# create a session
sess = tf.Session()
# feed the data, and run the session
x_data = np.array([[1, 2], [3, 4]])
print(sess.run(y, feed_dict={x: x_data}))
```



How to get the result?

Run the graph

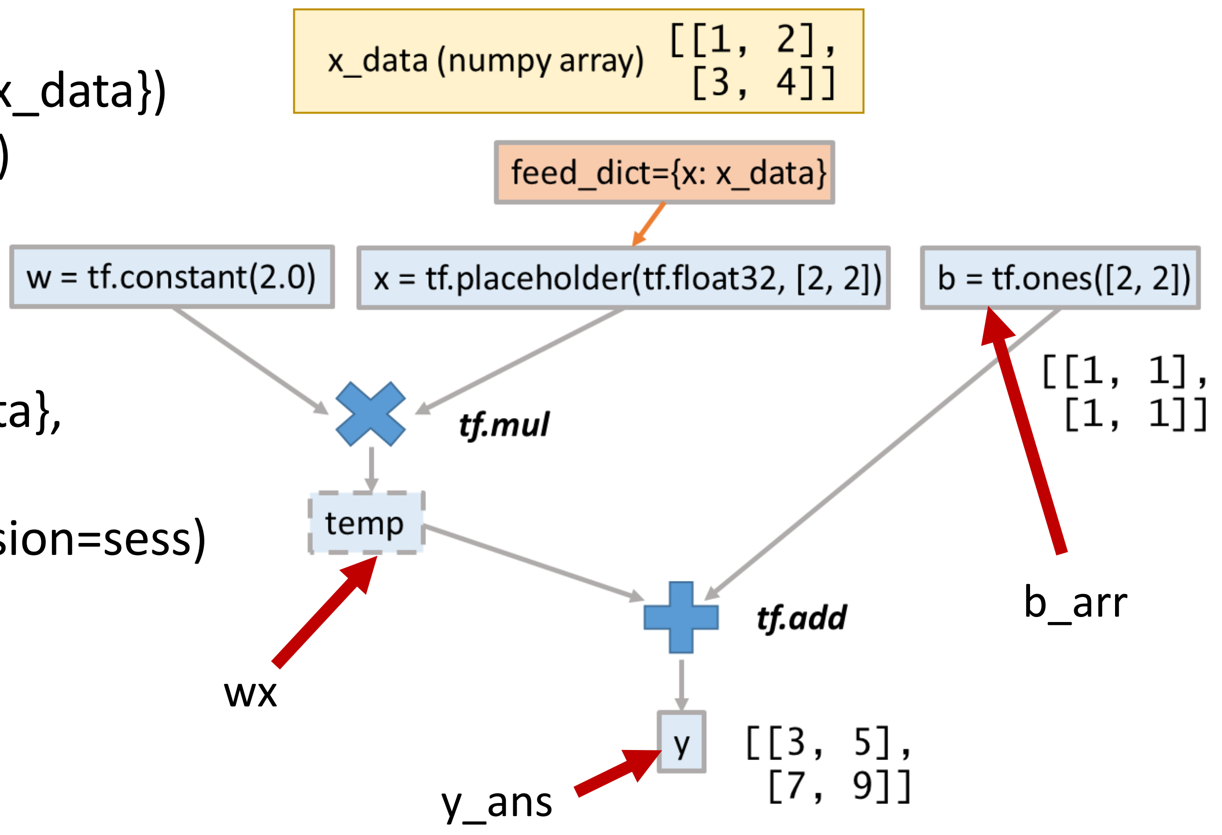
Usage:

- ***sess.run([<nodes>], <feed_dict>)***
  - E.g. b_arr = sess.run(b)
  - E.g. wx = sess.run(tf.mul(w, x), feed_dict={x: x_data})
  - E.g. y_ans = sess.run(y, feed_dict={x: x_data})

- ***<tensor>.eval(session=sess)***
  - E.g. b_arr = b.eval(session=sess)
  - E.g. wx = tf.mul(w, x).eval(feed_dict={x: x_data}, session=sess)
  - E.g. y_ans = y.eval(feed_dict={x: x_data}, session=sess)

Only run the graph before the node you designate!

E.g. $y = w \cdot x + b$



References

- Stanford CS224d: TensorFlow Tutorial
- Stanford CS231n: Deep Learning Software
- TensorFlow docs

Environment Setup

Environment Setup

 # install miniconda (python2.7)

 # if you don’t have wget, you can directly go to the website to download the script

```
wget https://repo.continuum.io/miniconda/Miniconda2-latest-MacOSX-x86_64.sh

bash Miniconda2-latest-MacOSX-x86_64.sh
```

 # append the following line to ~/.bashrc if it is not done automatically

```
export PATH="path/to/anaconda2/bin":$PATH
```

 # check conda installation

```
conda list
```

Environment Setup

 # create new environment

conda create -n tensorflow

 # activate the environment

source activate tensorflow

 # install required package in the environment

pip install opencv

pip install matplotlib

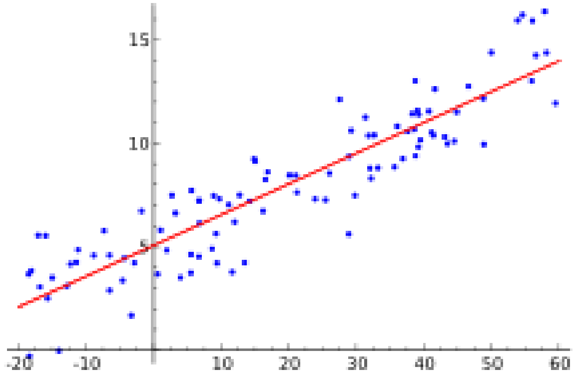
pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.2.0-py2-none-any.whl

pip install jupyter notebook

TensorFlow Sample codes

Linear Regression

E.g. $\boldsymbol{y} = \boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}$



[[2, 6],
[1, 2],
[4, 5],
[6, 8]]

[[43],
[20],
[44],
[65]]

feed_dict={x: data, y: label}

x = tf.placeholder(tf.float32, [batch_size, data_dim])

y = tf.placeholder(tf.float32, [batch_size, data_dim])

W = tf.Variable(tf.random_uniform([data_dim, 1], -1, 1))

b = tf.Variable(tf.random_uniform([1], -1, 1))

Variables are optimized during training

When testing, you only run to ***y_pred*** or ***loss***

y_pred = tf.add(tf.matmul(x, W), b)

loss = tf.reduce_mean(tf.square(y-y_pred))

$$J(W, b) = \frac{1}{N} \sum_{i=1}^{N} (y - y\_pred)^2$$

Testing

Training

During training, you need to run the **optimizer**

opt = tf.train.AdamOptimizer(learning_rate=1).minimize(loss)

In [2]:
```python
import tensorflow as tf
import numpy as np
```

Linear Regression

$y = W \cdot x + b$

Given some data points and their labels, we can learn the parameters (W and b) of the model by reducing the loss.

The answer of this model's parameters are:
W_ans = [[3, 5]]
b_ans = [7]

In [3]:
```python
# data & label
data = np.array([[2, 6], [1, 2], [4, 5], [6, 8]])
label = np.array([[43], [20], [44], [65]])
```

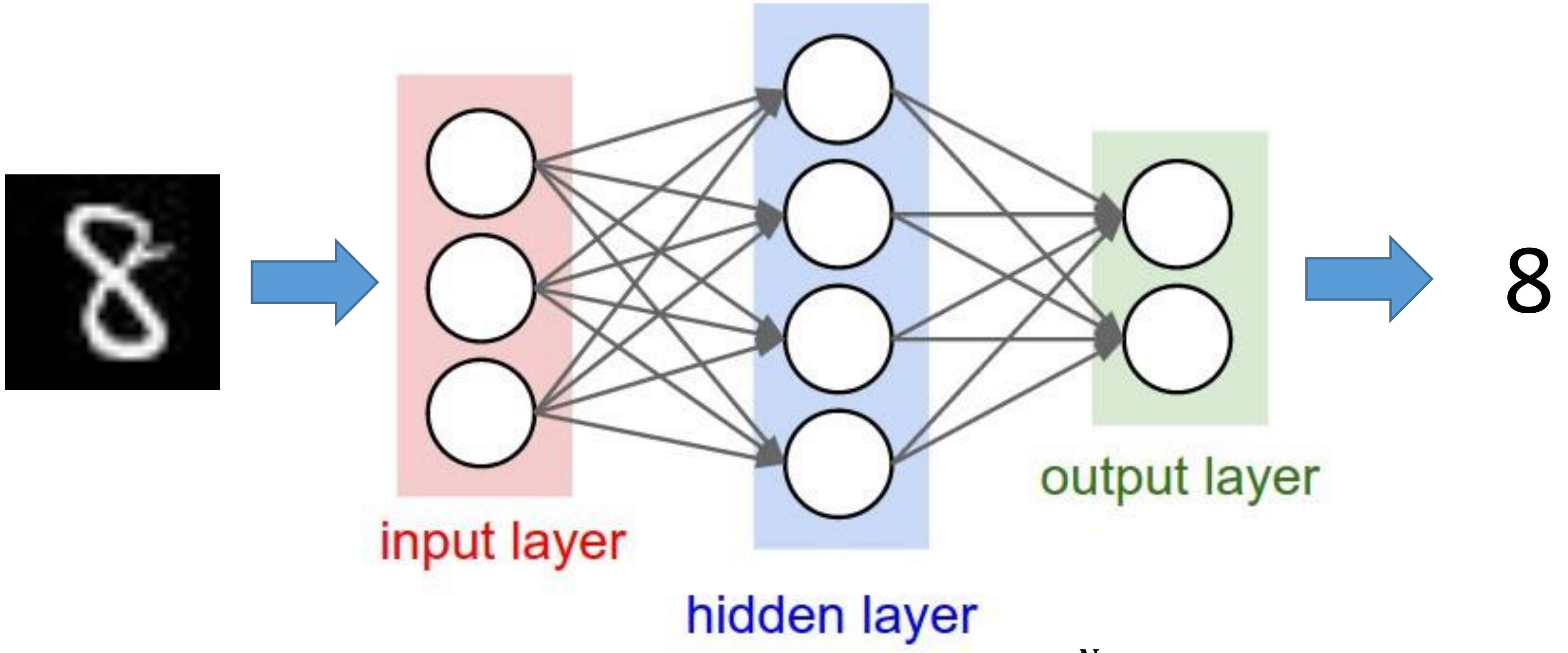
Build the graph (define your model)

In [4]:
```python
[batch_size, data_dim] = data.shape
# reserve place for x and y by placeholder
x = tf.placeholder(tf.float32, [batch_size, data_dim])
y = tf.placeholder(tf.float32, [batch_size, 1])

# W and b are random initialized
W = tf.Variable(tf.random_uniform([data_dim, 1], -1, 1))
b = tf.Variable(tf.random_uniform([1], -1, 1))

# y = w*x+b
y_pred = tf.add(tf.matmul(x, W), b)

# compute the loss
loss = tf.reduce_mean(tf.square(y-y_pred))

# declare an optimizer
opt = tf.train.AdamOptimizer(learning_rate=1).minimize(loss)
```

Training and Testing

In [37]:
```python
# create a session
sess = tf.Session()
# initialize variables
```

Mnist classification using Neural Network



$$cross - entropy\ loss = -\sum_{i=1}^{N} y_i * \log(\hat{y}_i)$$

y is one hot encoding of the correct class

Neural Network classification in MNIST digits dataset

In [2]:
```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
```

In [3]:
```
# Define training parameter
max_iter = 10000
batch_size = 100

# get MNIST data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

```
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting MNIST_data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

Function for Neural Network

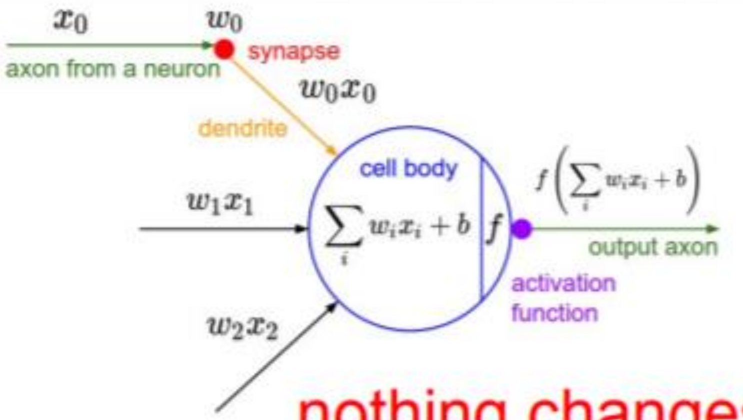
In [4]:
```
def add_layer(inputs, in_size, out_size, activation_function=None):
    # add one more layer and return the output of this layer
    Weights = tf.Variable(tf.random_normal([in_size, out_size]))
    biases = tf.Variable(tf.zeros([1, out_size]) + 0.1)
    Wx_plus_b = tf.add(tf.matmul(inputs, Weights),biases)
    # apply dropout and the given activation function
    if activation_function is None:
        outputs = tf.nn.dropout(Wx_plus_b, keep_prob)
    else:
        outputs = tf.nn.dropout(activation_function(Wx_plus_b), keep_prob)
    return outputs
```

In [5]:
```
def compute_accuracy(x, y):
    y_pre = sess.run(prediction, feed_dict={xs: x, keep_prob: 1.})
    correct_prediction = tf.equal(tf.argmax(y_pre, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    result = sess.run(accuracy, feed_dict={xs: x, ys: y, keep_prob: 1.})
    return result
```

CNN sample codes

Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



nothing changes

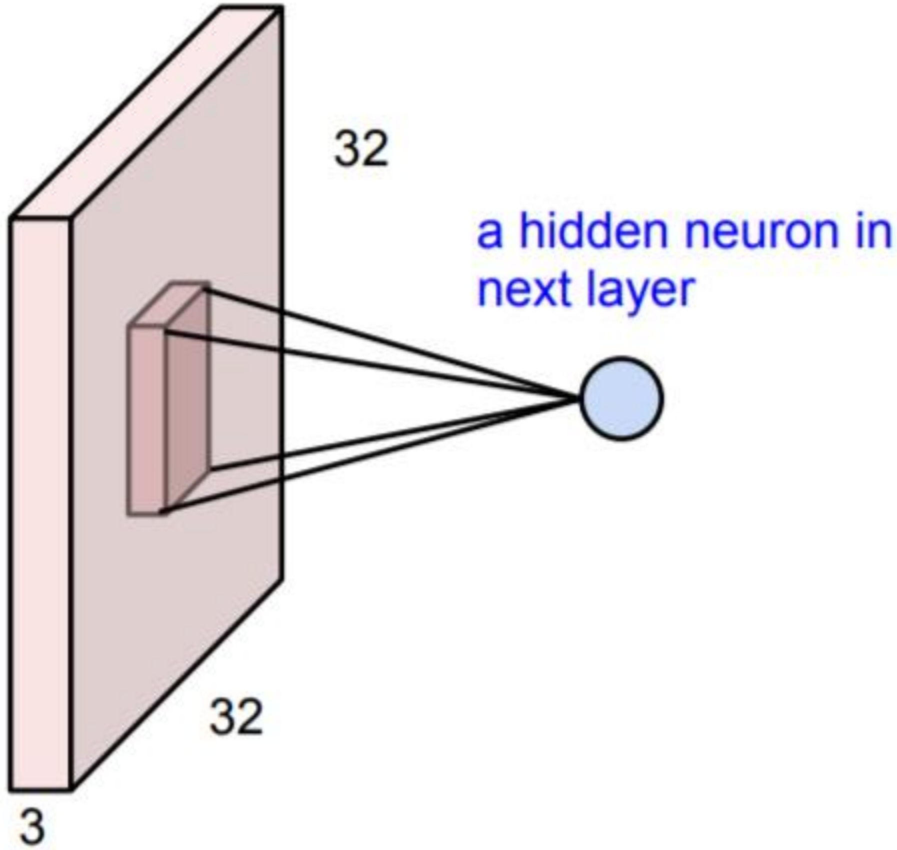


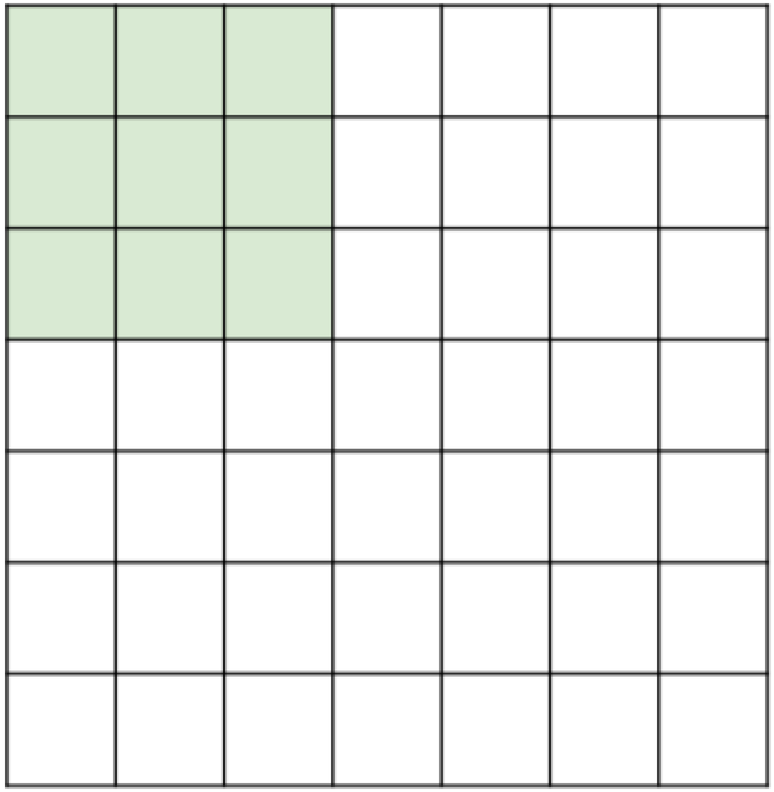
image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

note that connectivity is:

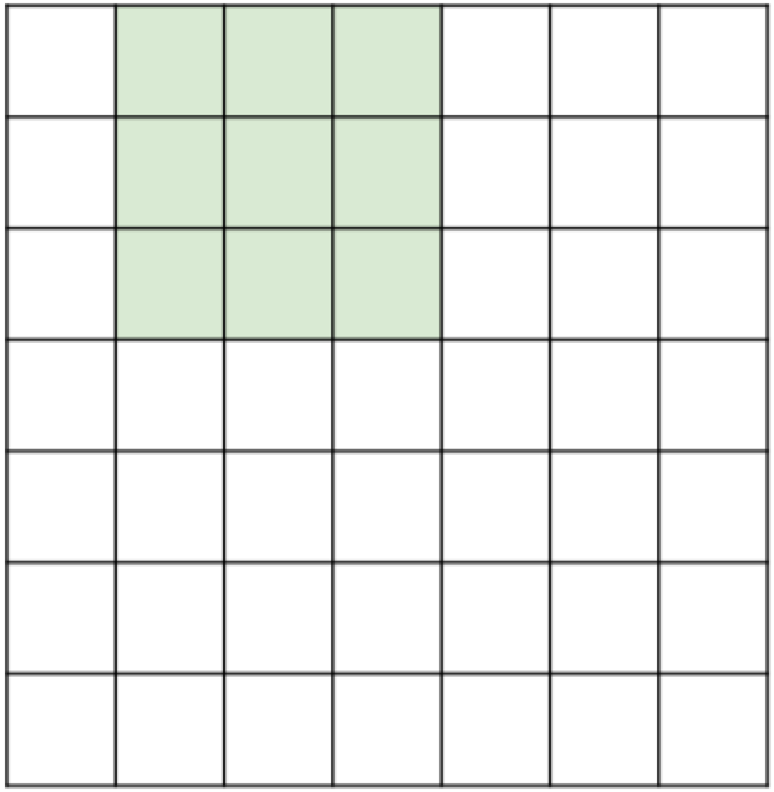
- <u>local in space</u> (5x5 inside 32x32)
- <u>but full in depth</u> (all 3 depth channels)

Replicate this column of hidden neurons across space, with some **stride.**



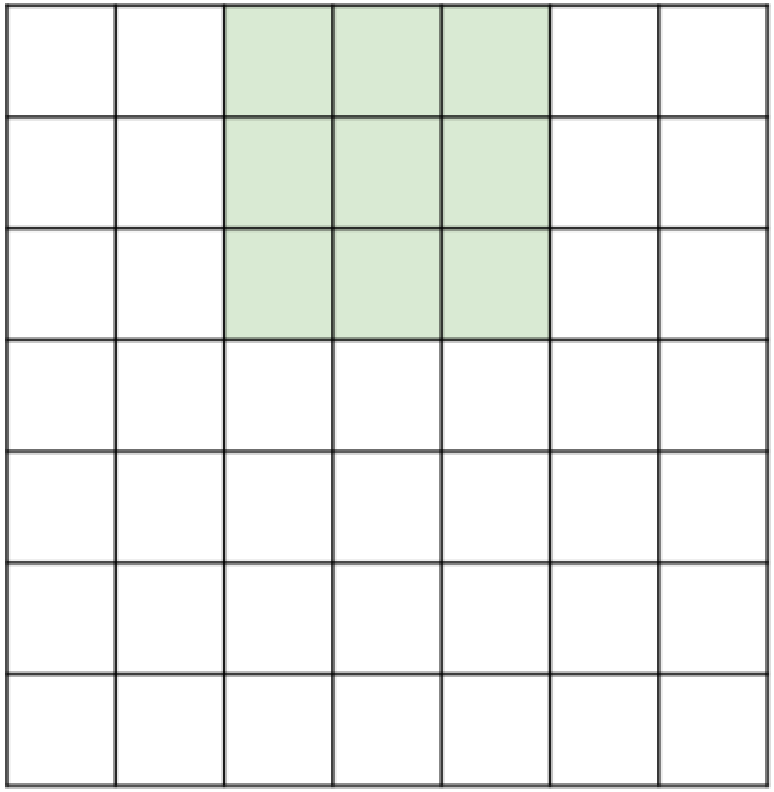
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



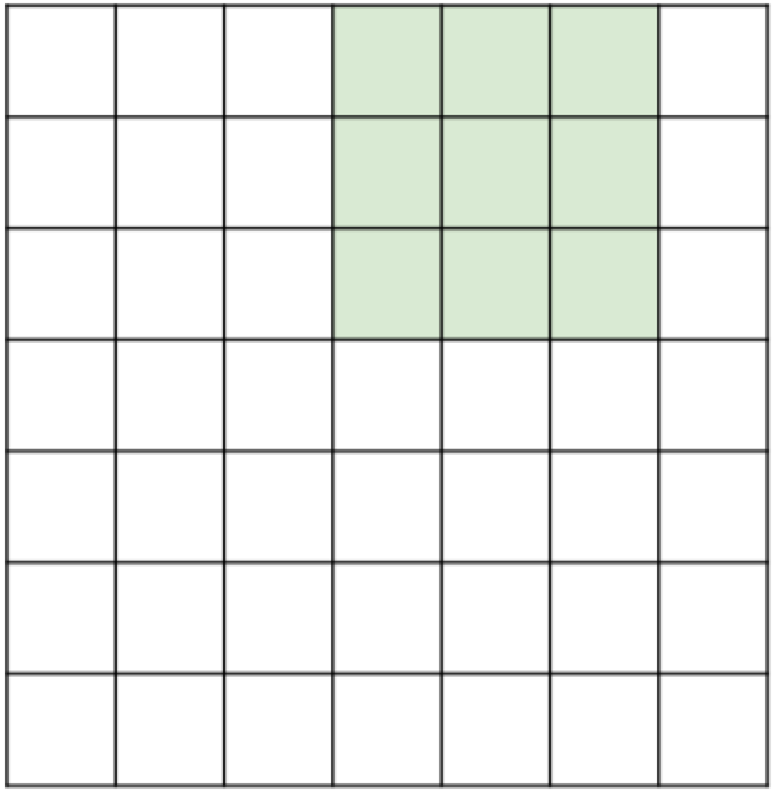
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



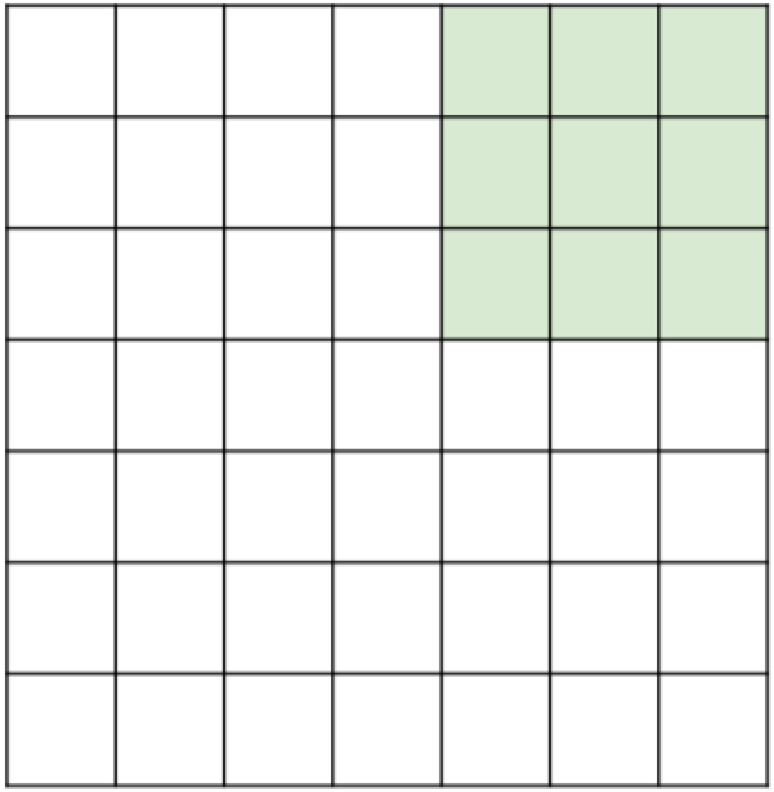
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



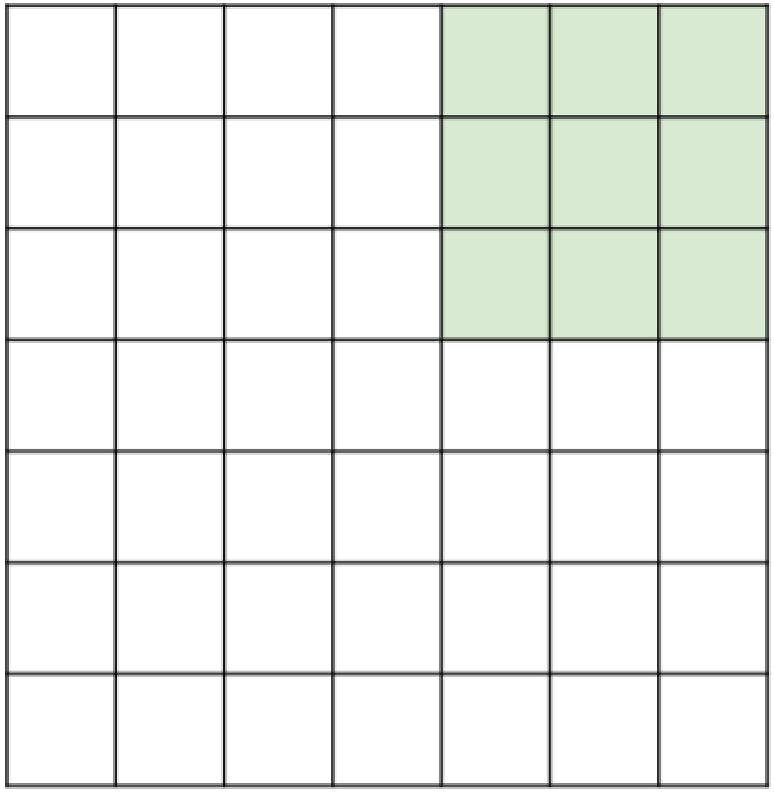
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



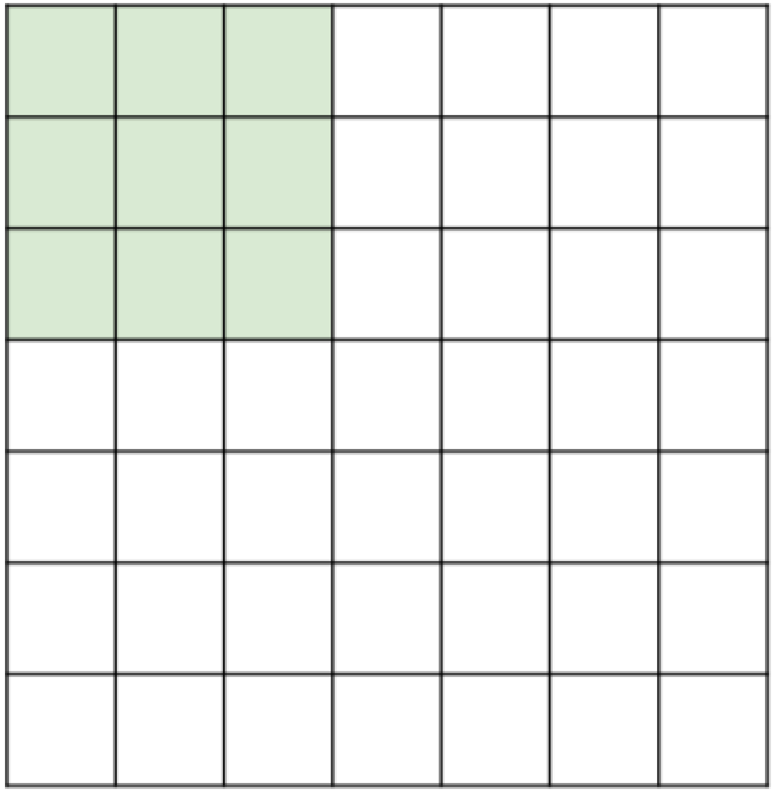
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

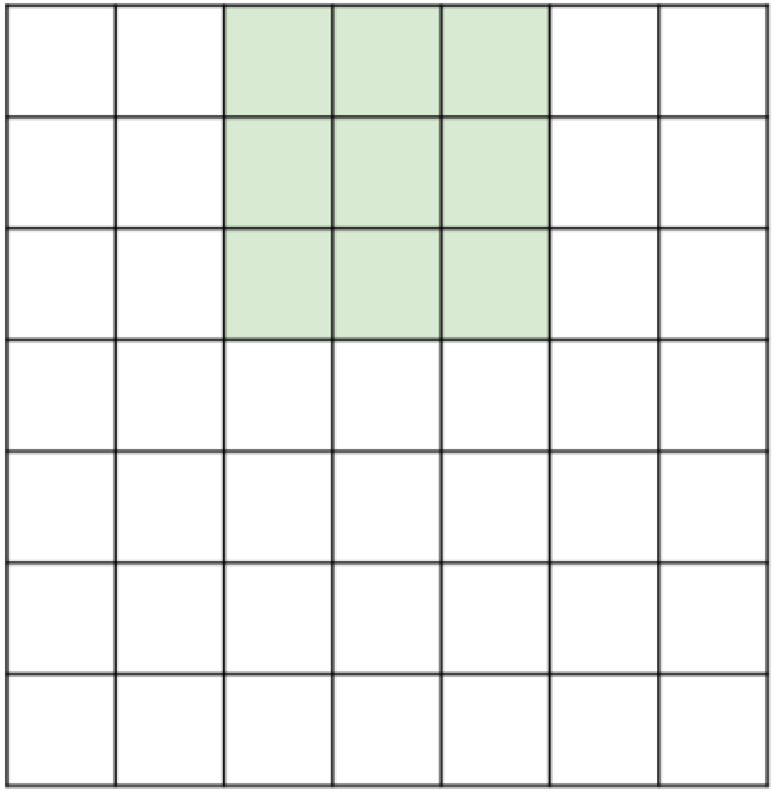
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> 5x5 output

what about stride 2?

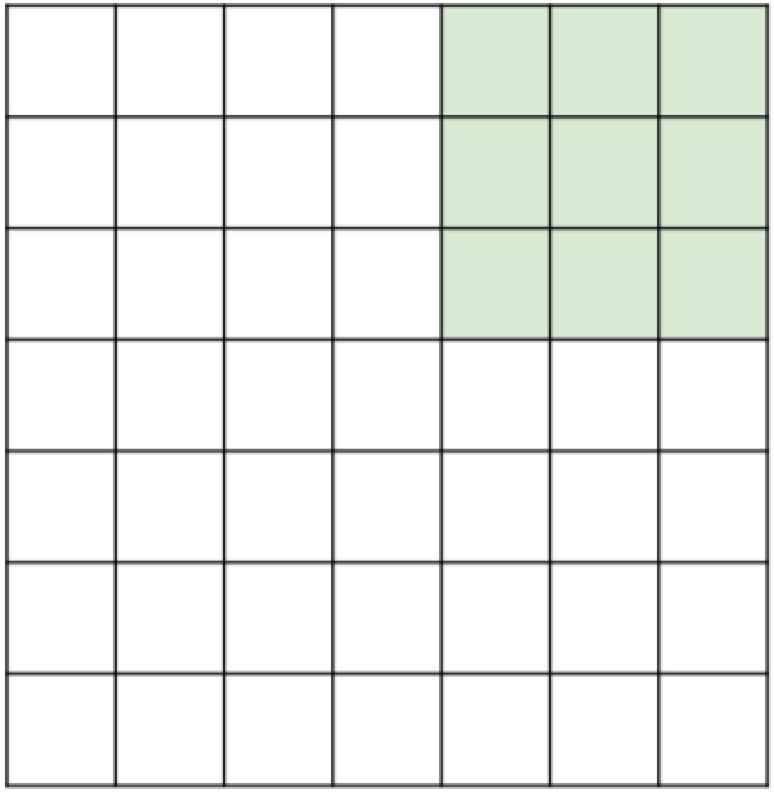
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> 5x5 output

what about stride 2?

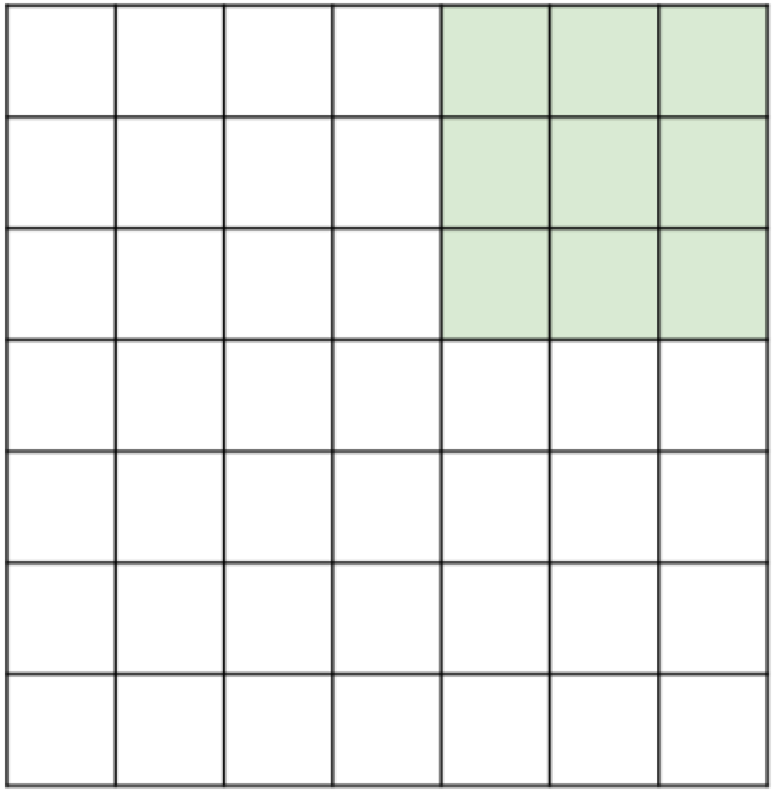
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

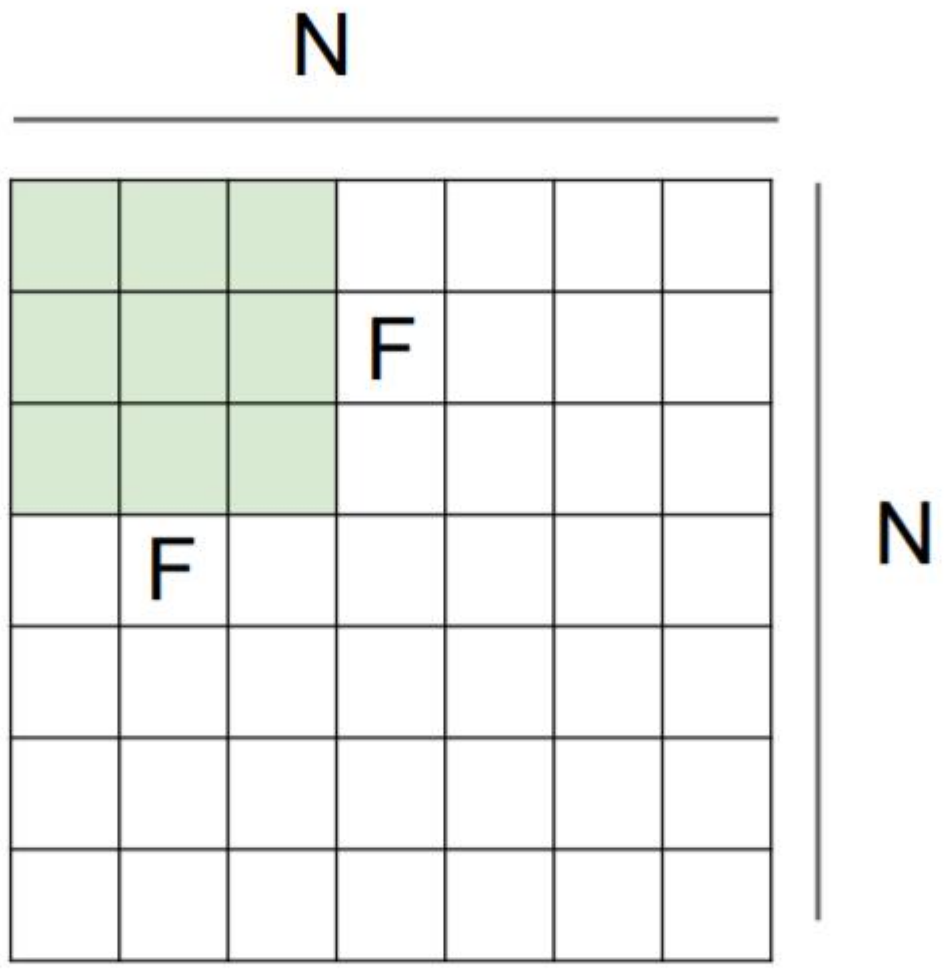
what about stride 2?

Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

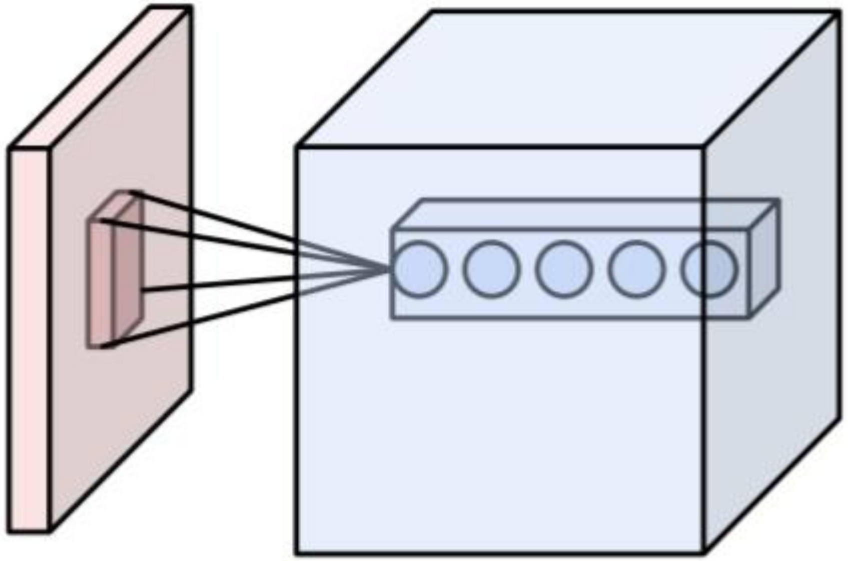
what about stride 2?
=> **3x3 output**



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = … :\

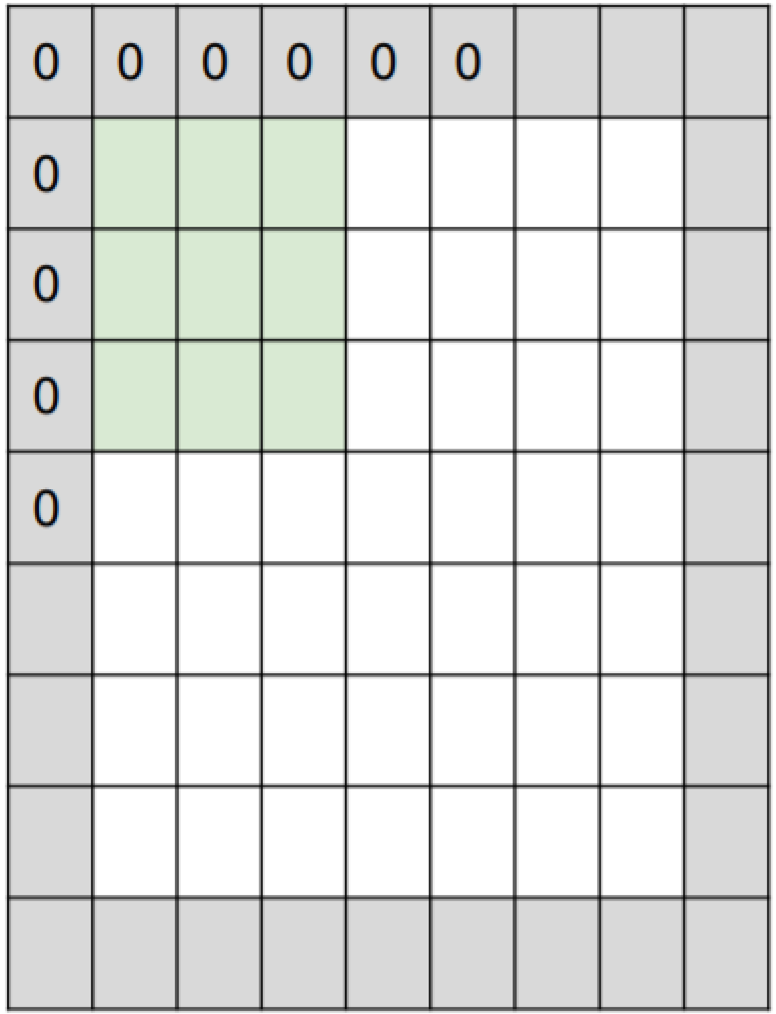
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: (32 - 5) / 1 + 1 = 28, so: **28x28x5**
How many weights for each of the 28x28x5 neurons? **5x5x3 = 75**

In practice: Common to zero pad the border



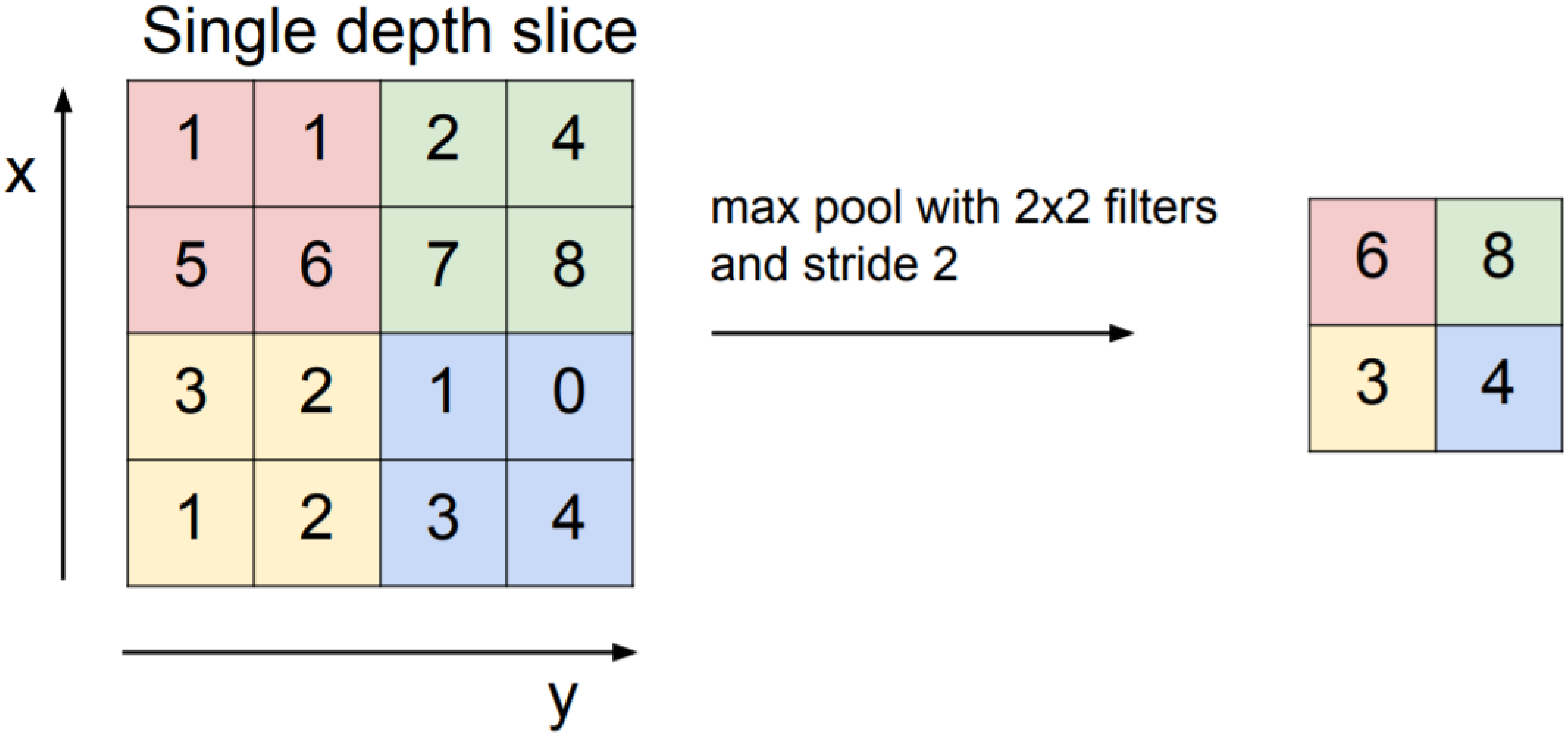
(in each channel)

e.g. input 7x7
neuron with receptive field 3x3, stride 1
pad with 1 pixel border => what is the output?

7x7 => preserved size!

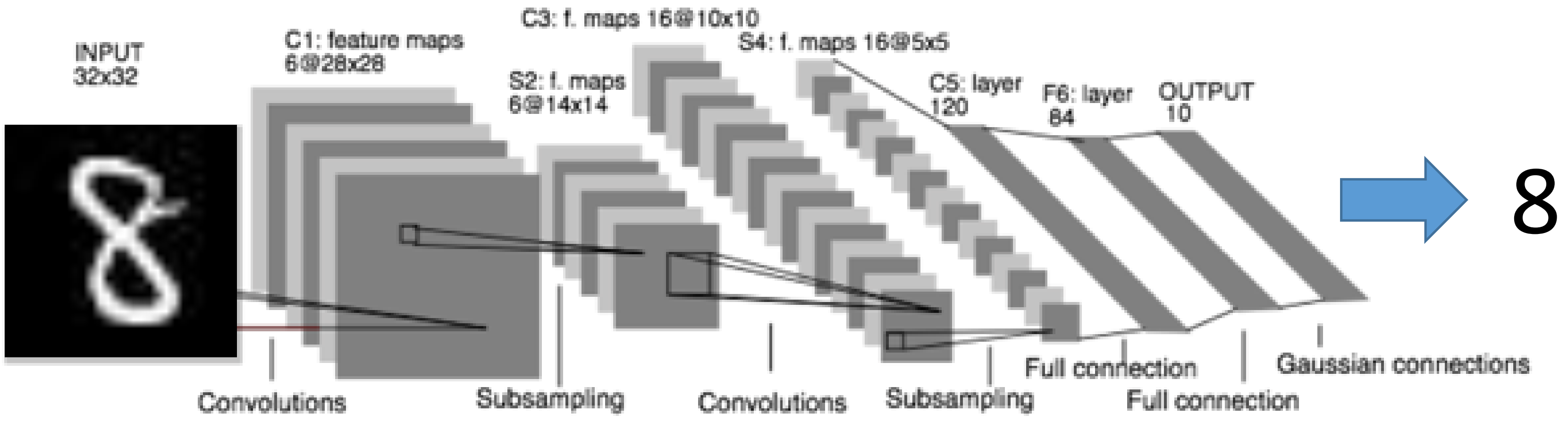
in general, common to see stride 1, size F, and
zero-padding with (F-1)/2.
(Will preserve input size spatially)

MAX POOLING


Single depth slice
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2
6 8
3 4

Mnist classification using LeNet

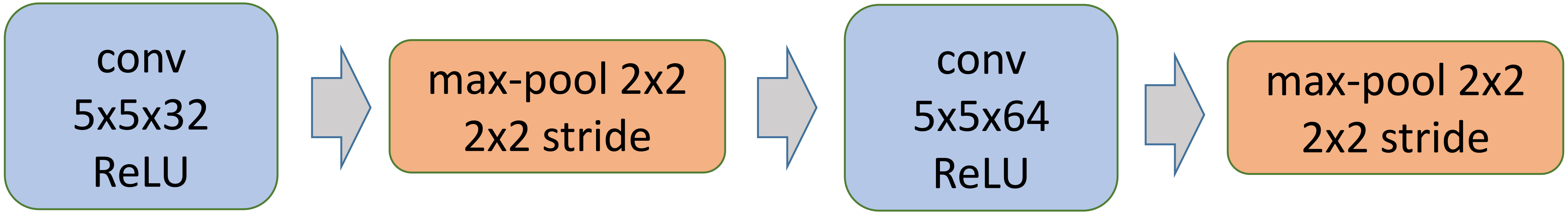
Convolutional Networks: 1989



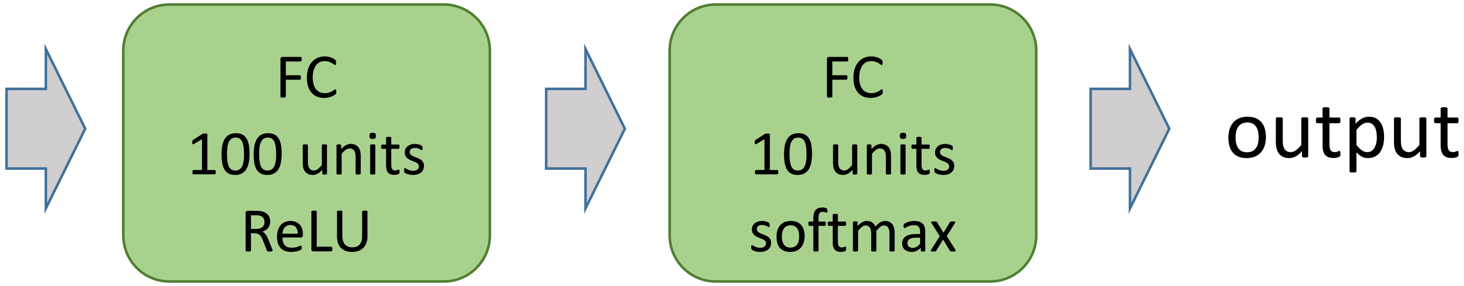
LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [LeNet]

CNN model for mnist

PART I

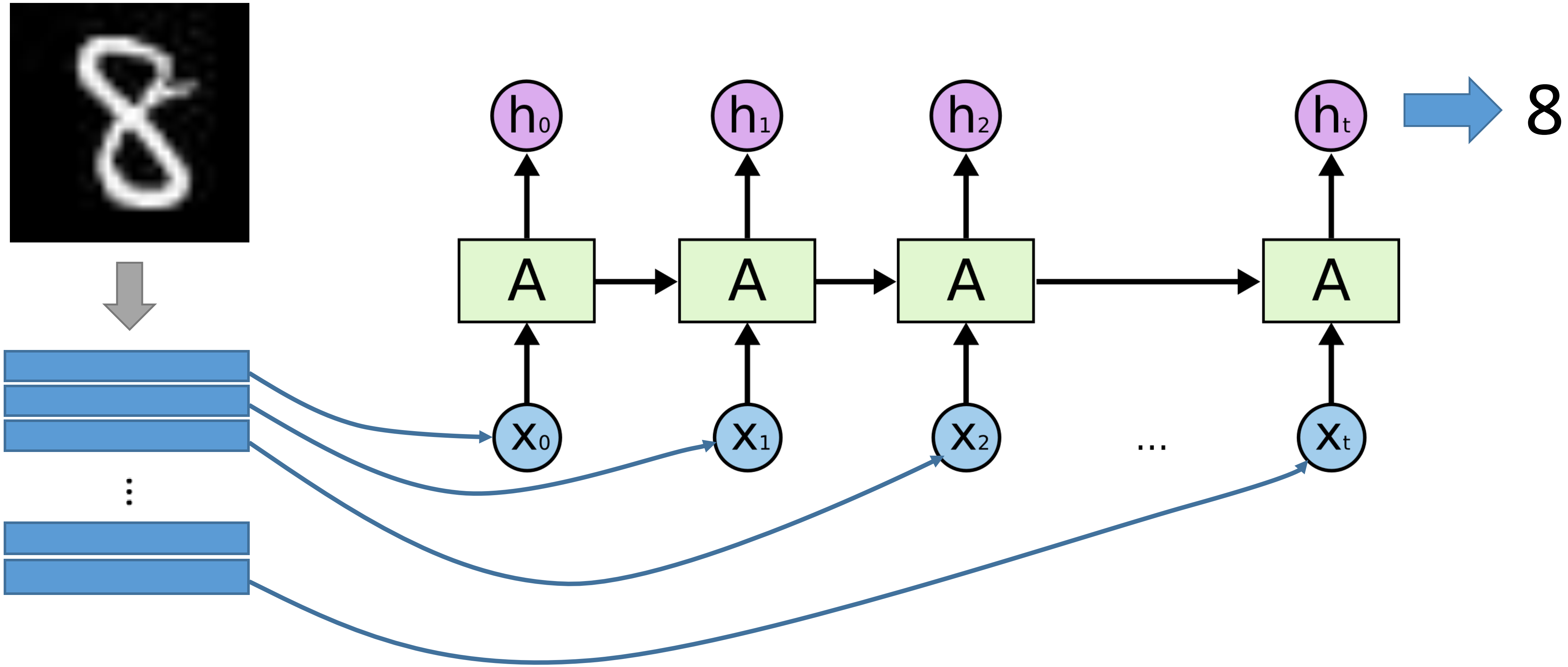


PART II



RNN sample codes

Mnist classification using RNN



References

- Stanford CS231: Convolutional Neural Networks for Visual Recognition