

Deep Learning for Visual Recognition

- Week 3

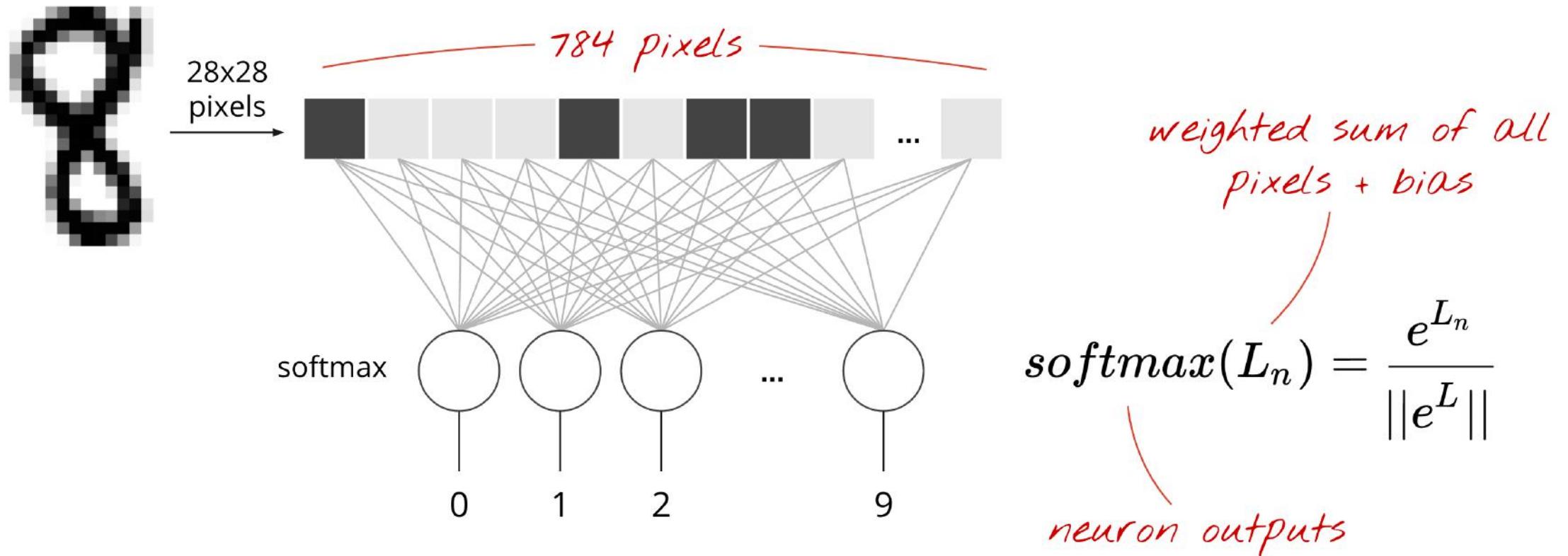
Kyu-Hwan Jung, Ph.D
kyuhwanjung@gmail.com

Class 11

Review of CNNs

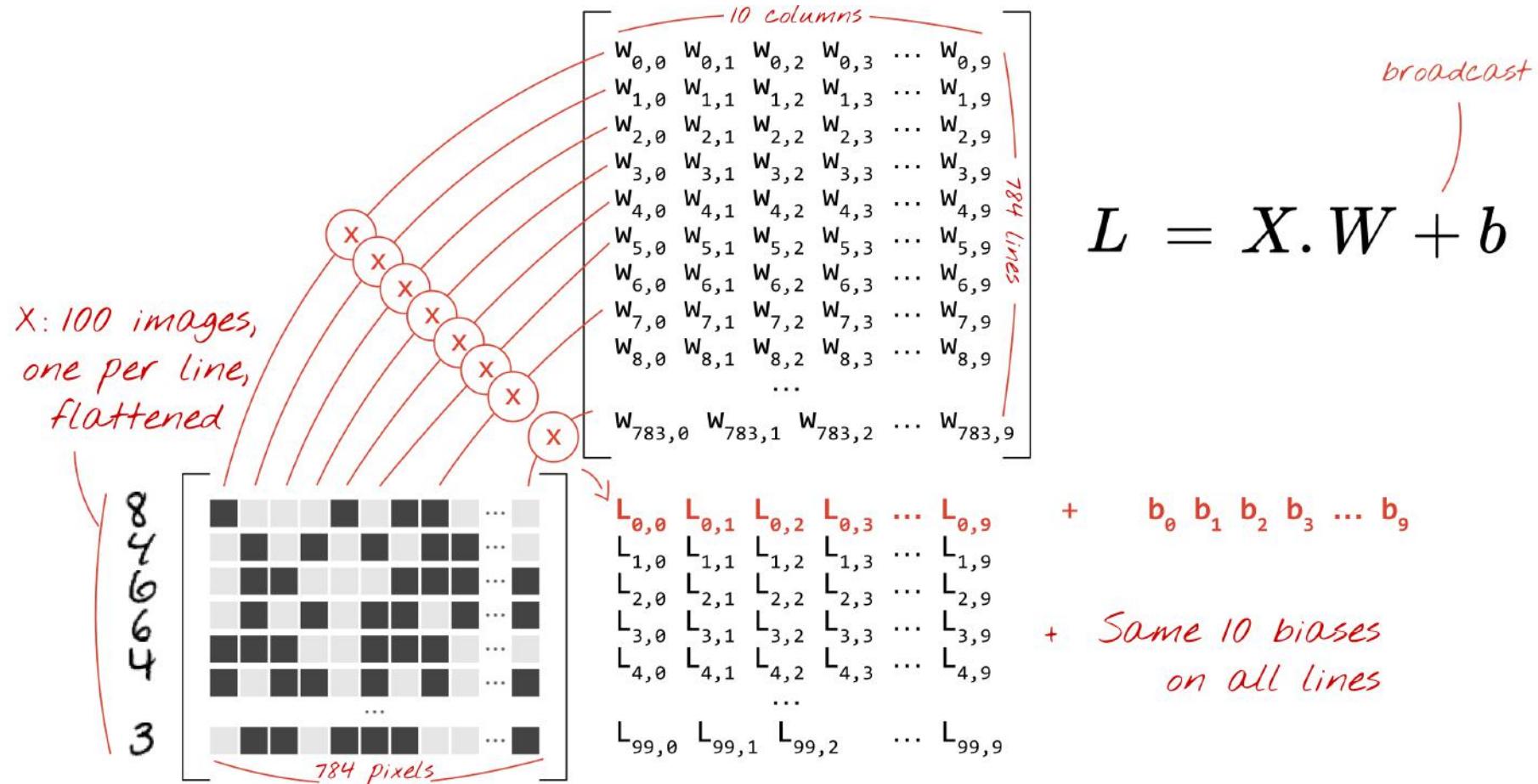
The contents of following section is borrowed from google cloud platform blog : <https://goo.gl/Qc6Qbu>

Softmax Classification



Softmax Classification

- Matrix Notation



Softmax Classification

- Batch Notation

Predictions
 $Y[100, 10]$

Images
 $X[100, 784]$

Weights
 $W[784, 10]$

Biases
 $b[10]$

$$Y = softmax(X \cdot W + b)$$

applied line by line

matrix multiply

broadcast on all lines

tensor shapes in []

tensor shapes: $X[100, 784]$ $W[784, 10]$ $b[10]$

$$Y = tf.nn.softmax(tf.matmul(X, W) + b)$$

matrix multiply

broadcast on all lines

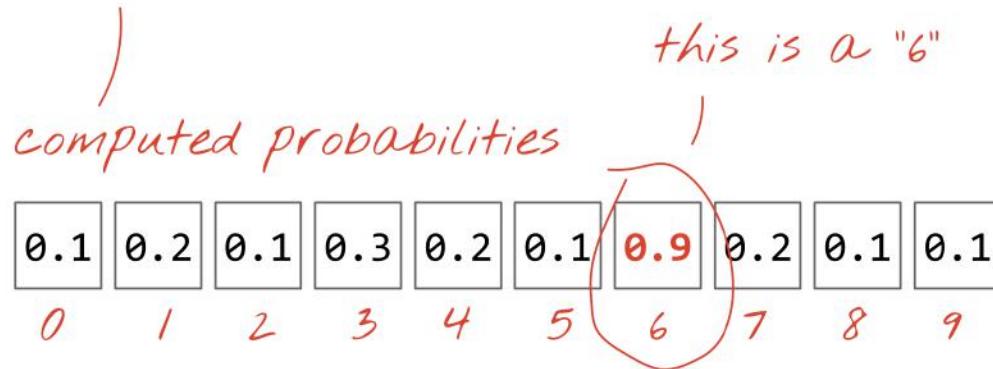
Softmax Classification

- Loss Function

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	0	0	0

actual probabilities, "one-hot" encoded

Cross entropy: $-\sum Y'_i \cdot \log(Y_i)$



Softmax Classification

- Full Python Code

```
import tensorflow as tf

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

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

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

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

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

initialisation

model

success metrics

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

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

for i in range(10000):
    # Load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data = {X: batch_X, Y_: batch_Y}

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

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

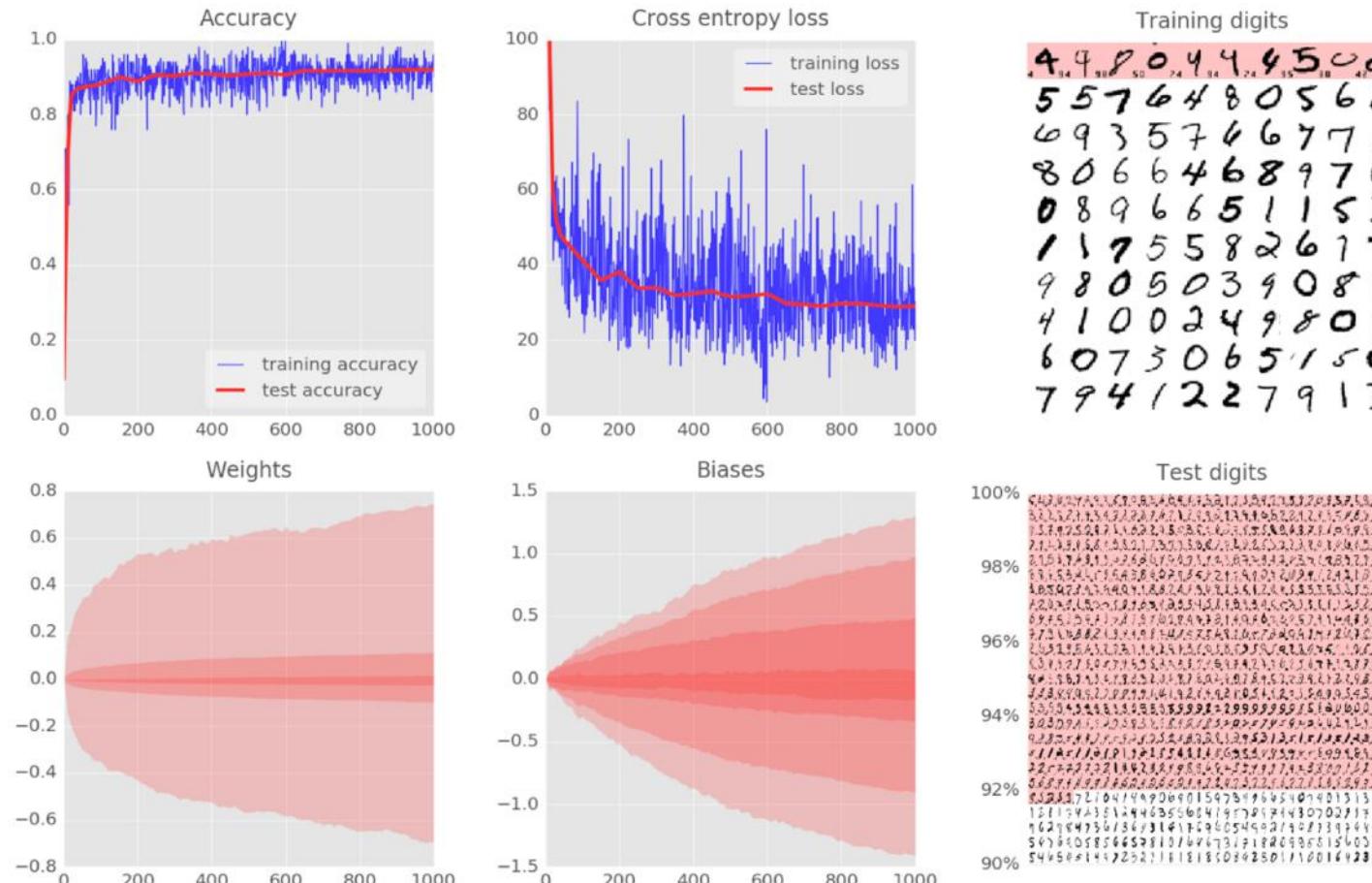
    # success on test data ?
    test_data = {X: mnist.test.images, Y_: mnist.test.labels}
    a, c = sess.run([accuracy, cross_entropy], feed=test_data)
```

training step

= Run

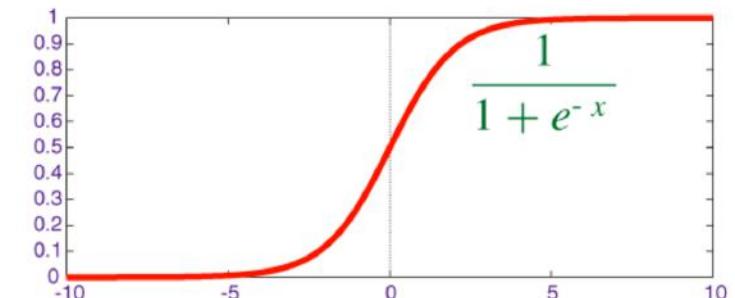
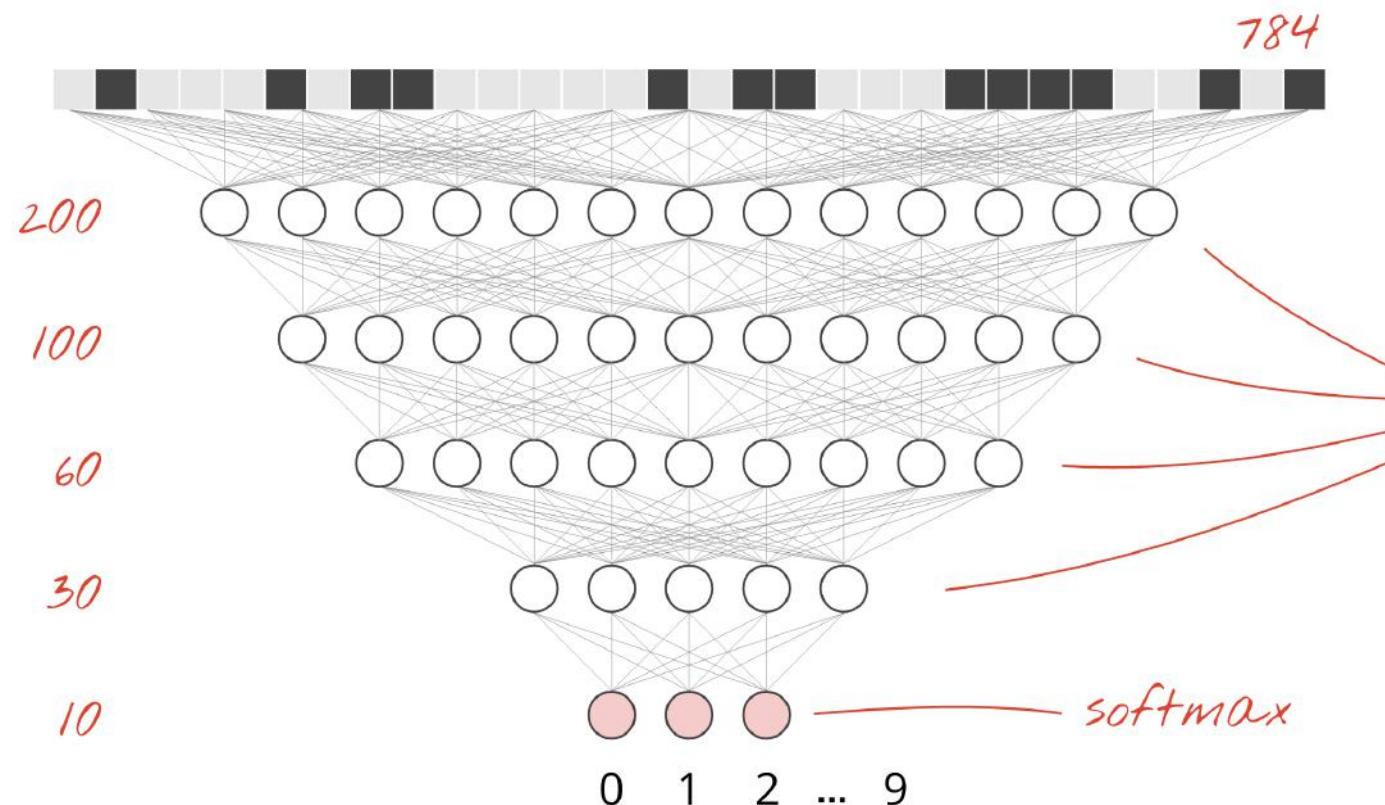
Softmax Classification

- Lets Run! : ./tensorflow-mnist/mnist_1.0_softmax.py



Multi-layered Perceptron

- Model Architecture



Multi-layered Perceptron

- TensorFlow Code

```
K = 200  
L = 100  
M = 60  
N = 30  
  
W1 = tf.Variable(tf.truncated_normal([28*28, K], stddev=0.1))  
B1 = tf.Variable(tf.zeros([K]))  
  
W2 = tf.Variable(tf.truncated_normal([K, L], stddev=0.1))  
B2 = tf.Variable(tf.zeros([L]))  
  
W3 = tf.Variable(tf.truncated_normal([L, M], stddev=0.1))  
B3 = tf.Variable(tf.zeros([M]))  
W4 = tf.Variable(tf.truncated_normal([M, N], stddev=0.1))  
B4 = tf.Variable(tf.zeros([N]))  
W5 = tf.Variable(tf.truncated_normal([N, 10], stddev=0.1))  
B5 = tf.Variable(tf.zeros([10]))
```

*weights initialised
with random values*



```
X = tf.reshape(X, [-1, 28*28])  
  
Y1 = tf.nn.sigmoid(tf.matmul(X, W1) + B1)  
Y2 = tf.nn.sigmoid(tf.matmul(Y1, W2) + B2)  
Y3 = tf.nn.sigmoid(tf.matmul(Y2, W3) + B3)  
Y4 = tf.nn.sigmoid(tf.matmul(Y3, W4) + B4)  
Y = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
```

weights and biases



Multi-layered Perceptron

▪ Truncated Normal Distribution

```
tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32,  
seed=None, name=None) {#truncated_normal}
```

Outputs random values from a truncated normal distribution.

The generated values follow a normal distribution with specified mean and standard deviation, except that values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked.

Args:

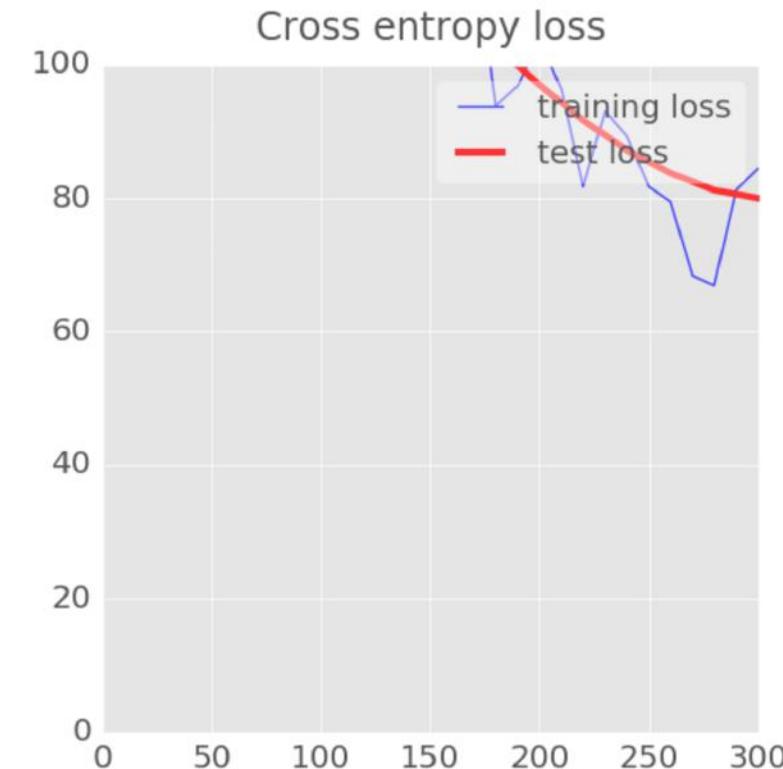
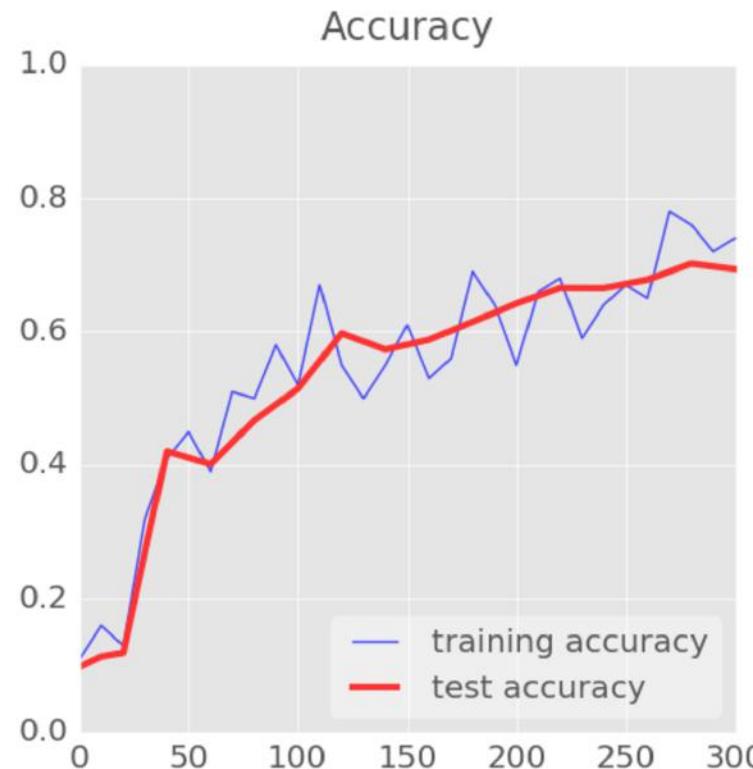
- **shape** : A 1-D integer Tensor or Python array. The shape of the output tensor.
- **mean** : A 0-D Tensor or Python value of type `dtype`. The mean of the truncated normal distribution.
- **stddev** : A 0-D Tensor or Python value of type `dtype`. The standard deviation of the truncated normal distribution.
- **dtype** : The type of the output.
- **seed** : A Python integer. Used to create a random seed for the distribution. See `set_random_seed` for behavior.
- **name** : A name for the operation (optional).

Returns:

A tensor of the specified shape filled with random truncated normal values.

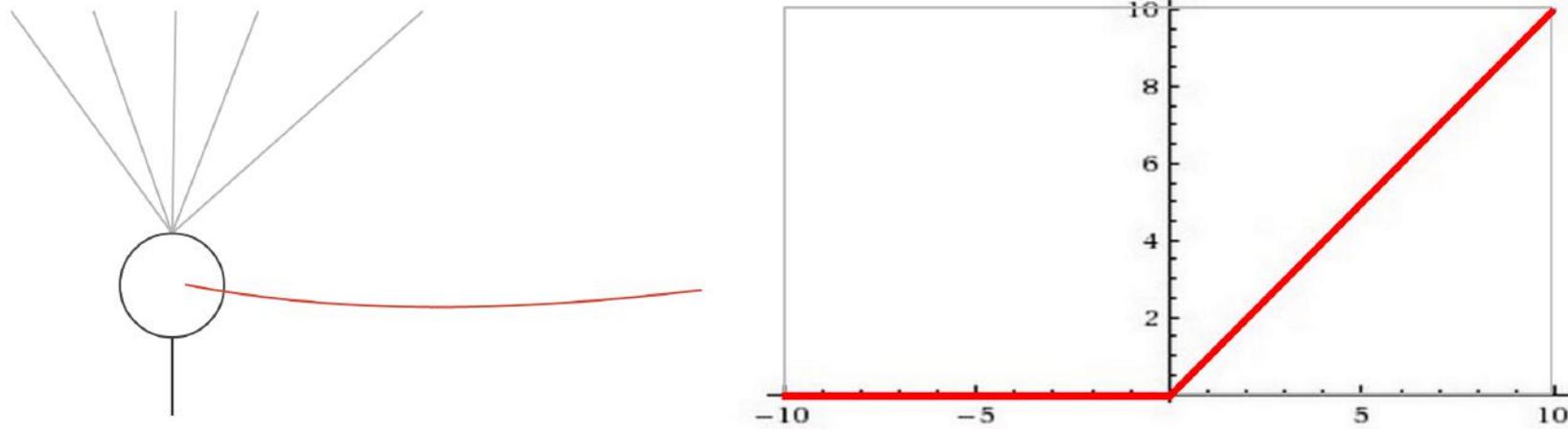
Multi-layered Perceptron

- Lets Run ! : ./tensorflow-mnist/mnist_2.0_five_layers_sigmoid.py
 - Result : Slow Start.



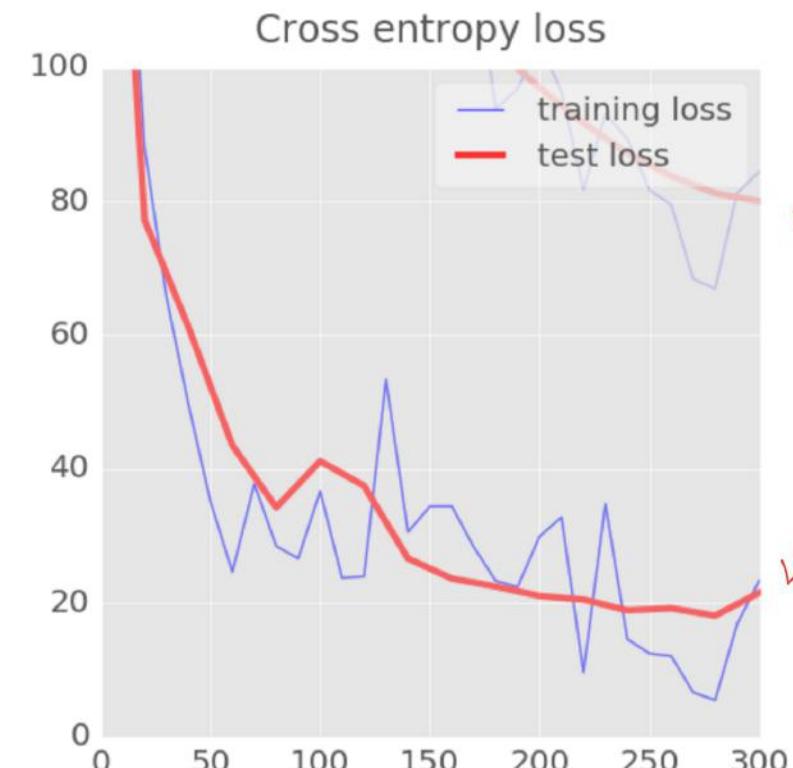
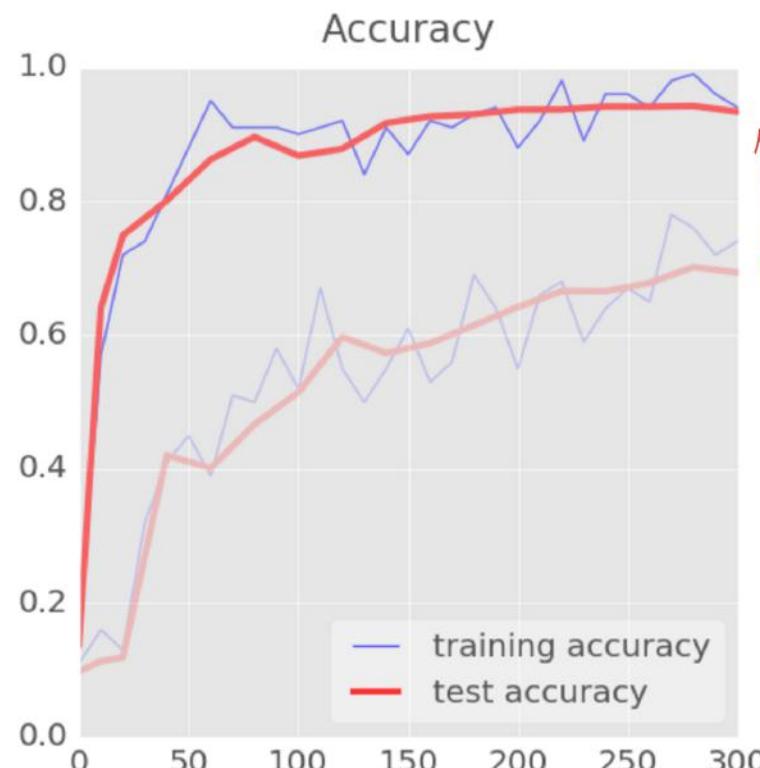
Multi-layered Perceptron

- Rectified Linear Unit


$$Y = \text{tf.nn.relu}(\text{tf.matmul}(X, W) + b)$$

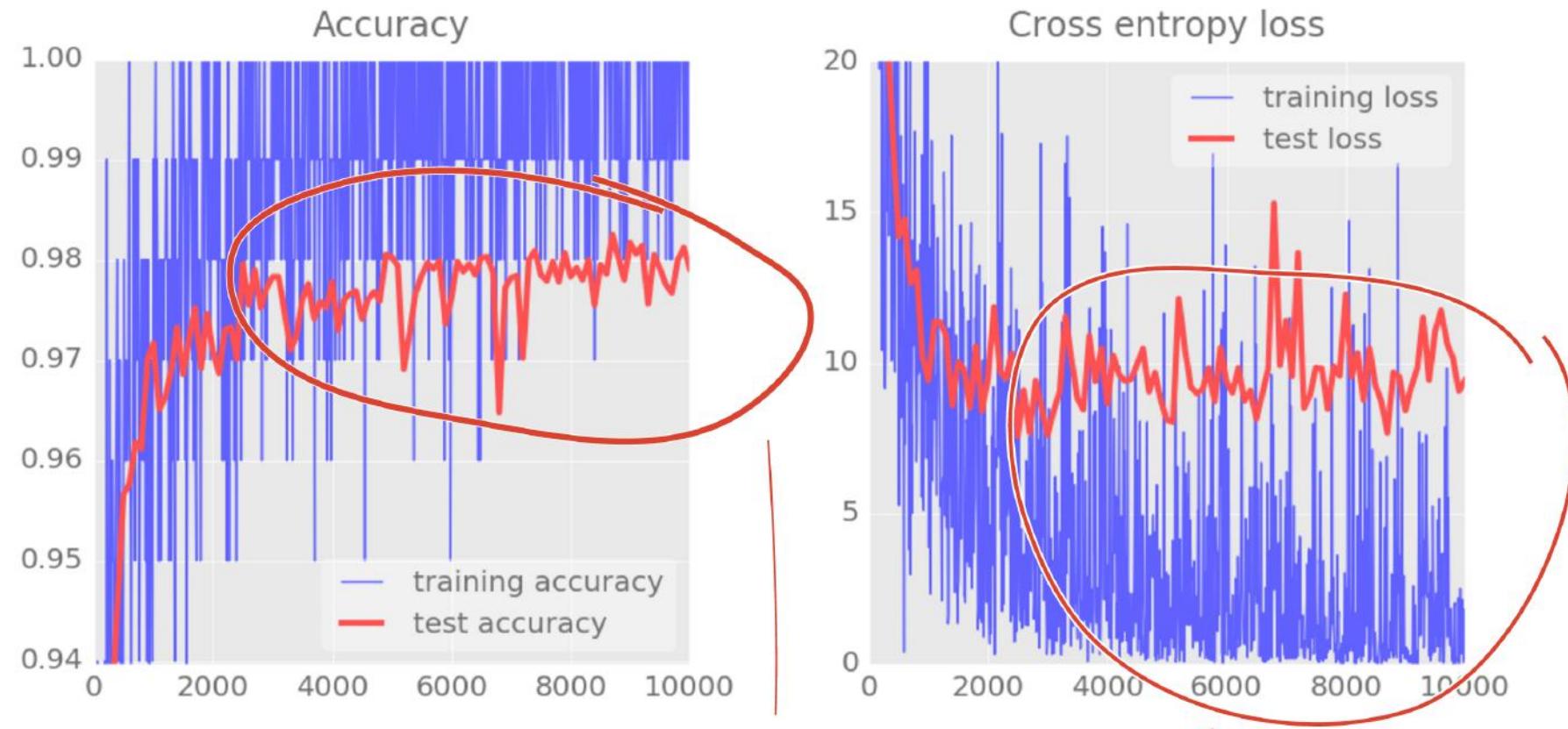
Multi-layered Perceptron

- Lets Run ! : ./tensorflow-mnist/mnist_2.1_five_layers_relu.py



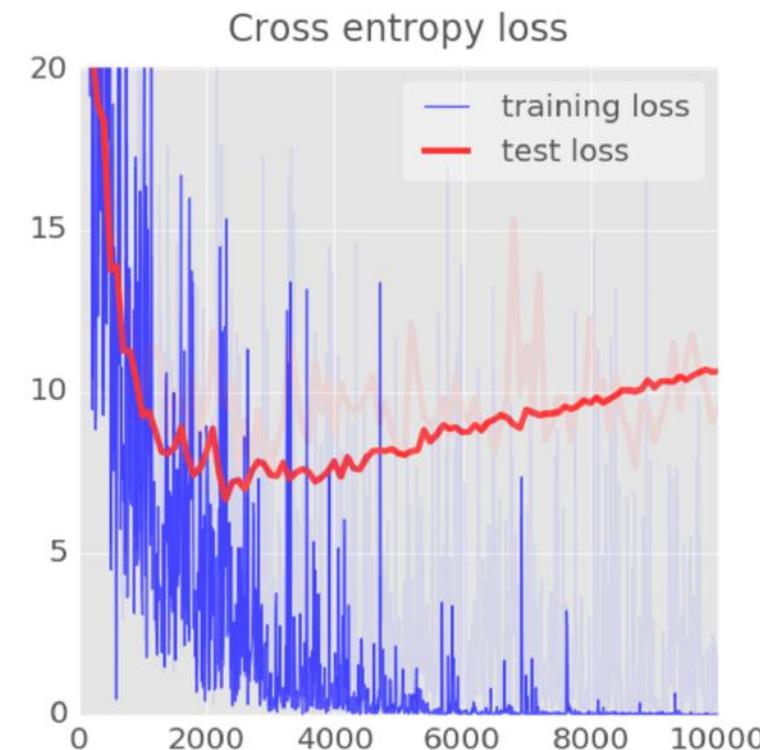
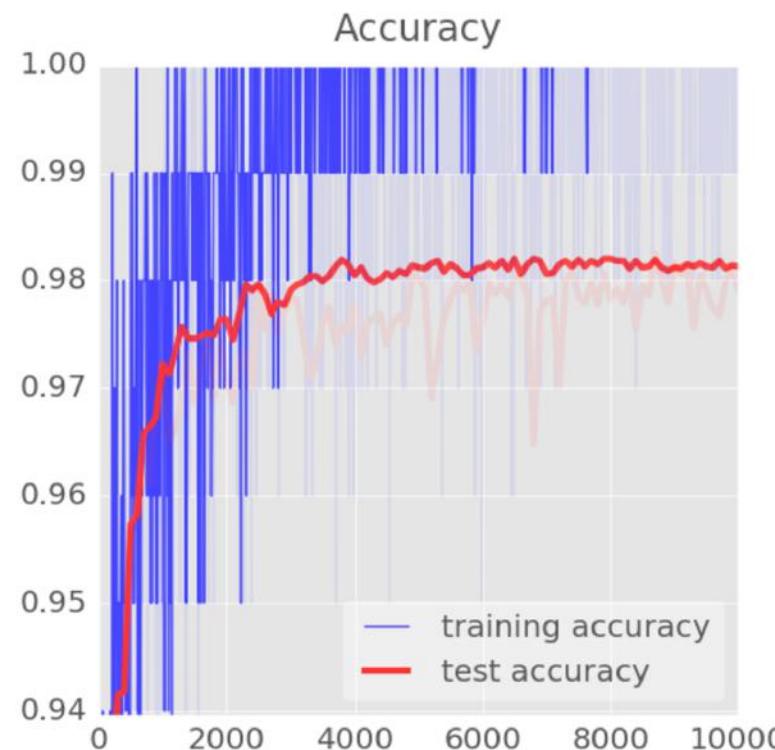
Multi-layered Perceptron

- Problem : Noisy Convergence -> Learning Rate Decay



Multi-layered Perceptron

- Let's Run ! : ./tensorflow-mnist/mnist_2.2_five_layers_relu_lrdecay.py

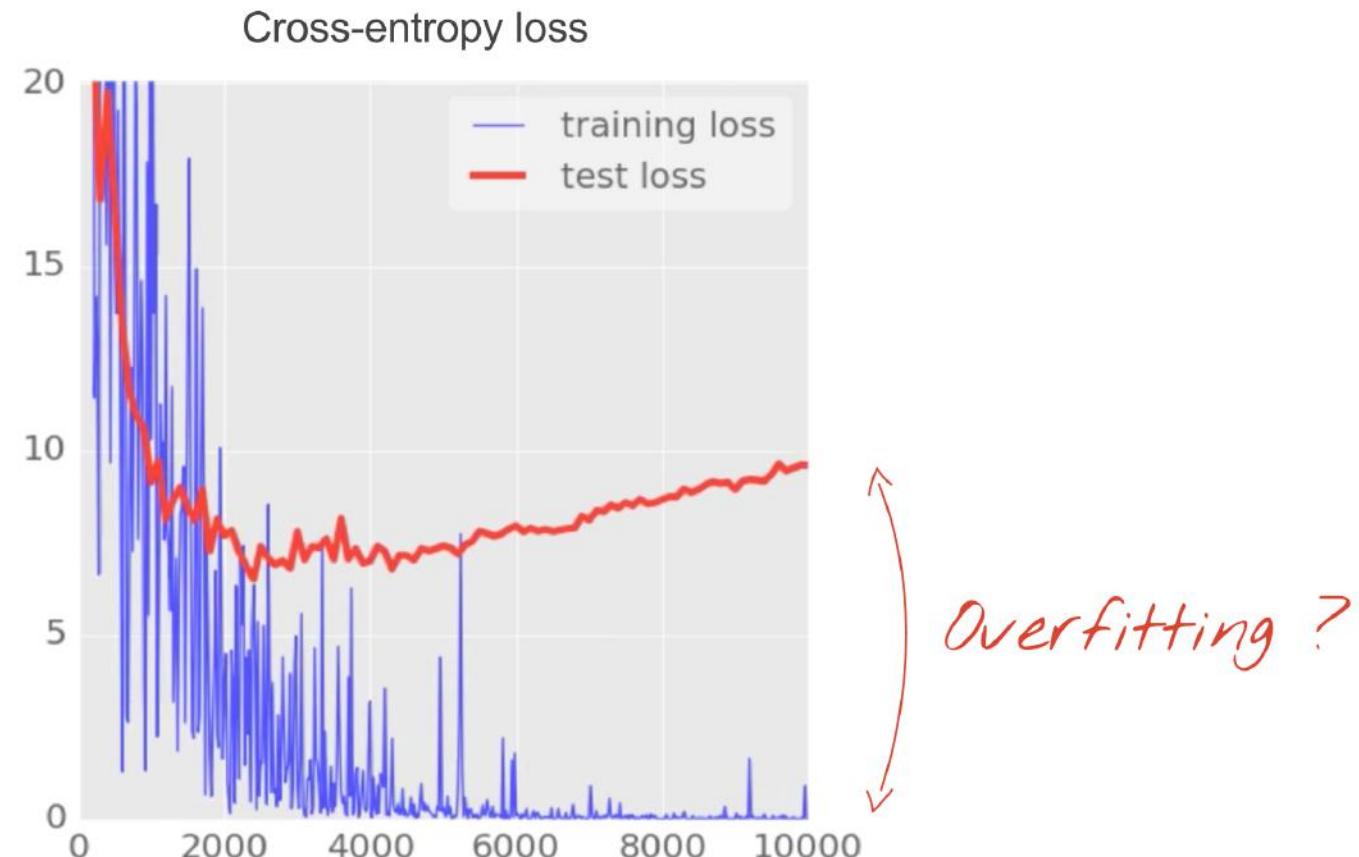


Learning rate 0.003 at start then dropping exponentially to 0.0001

```
learning_rate = min_learning_rate + (max_learning_rate - min_learning_rate) * math.exp(-i/decay_speed)
```

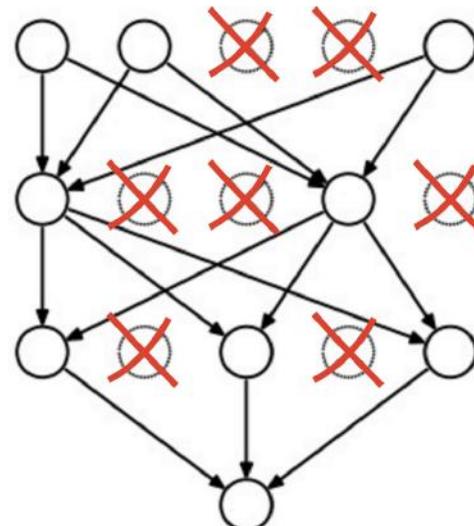
Multi-layered Perceptron

- Problem : Overfitting -> Dropout

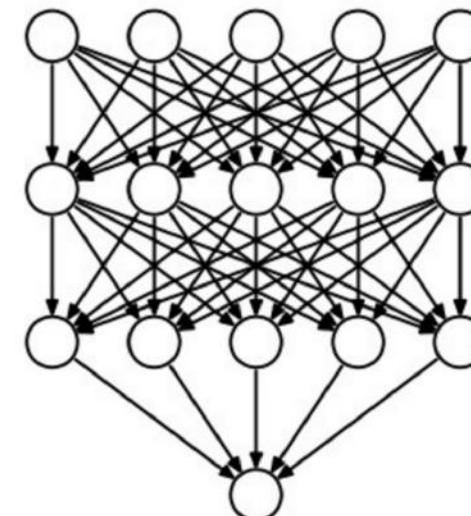


Multi-layered Perceptron

- Dropout



TRAINING
pKeep=0.75



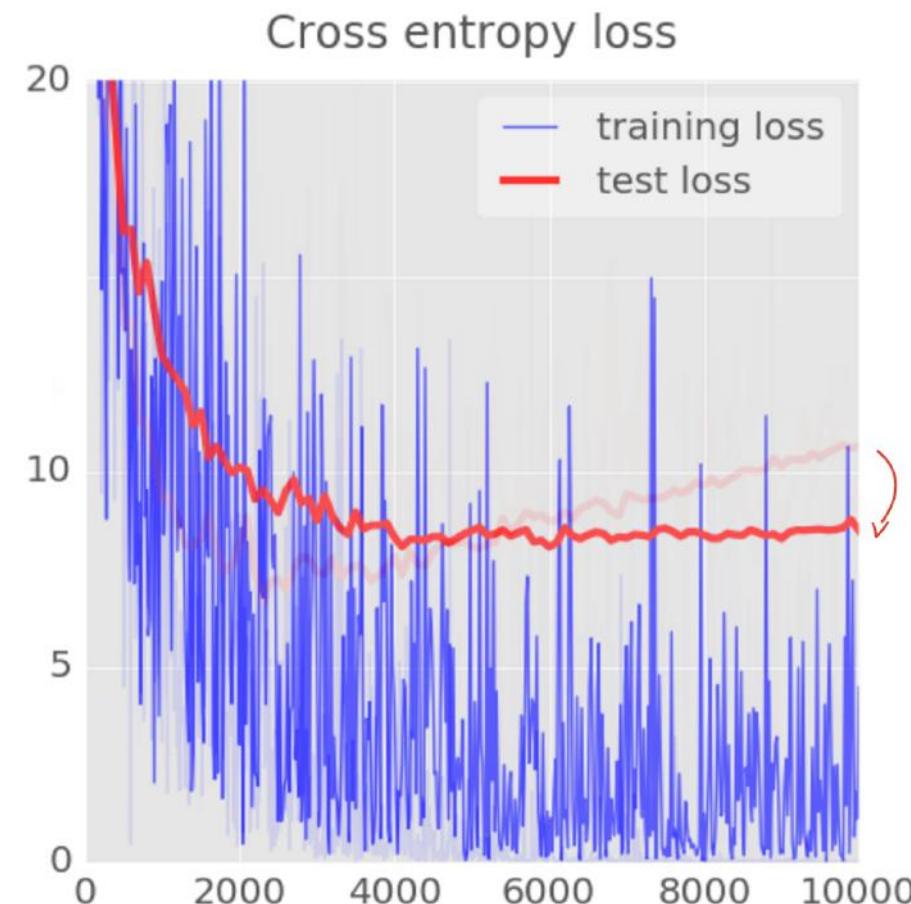
EVALUATION
pKeep=1

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

```
Yf = tf.nn.relu(tf.matmul(X, W) + B)  
Y = tf.nn.dropout(Yf, pkeep)
```

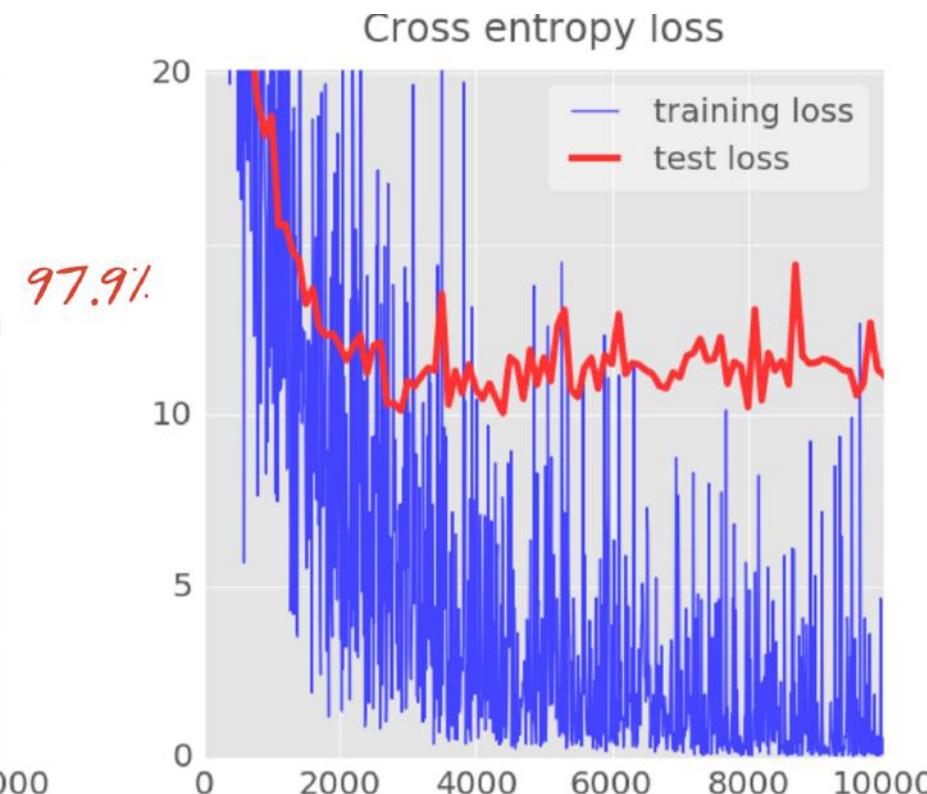
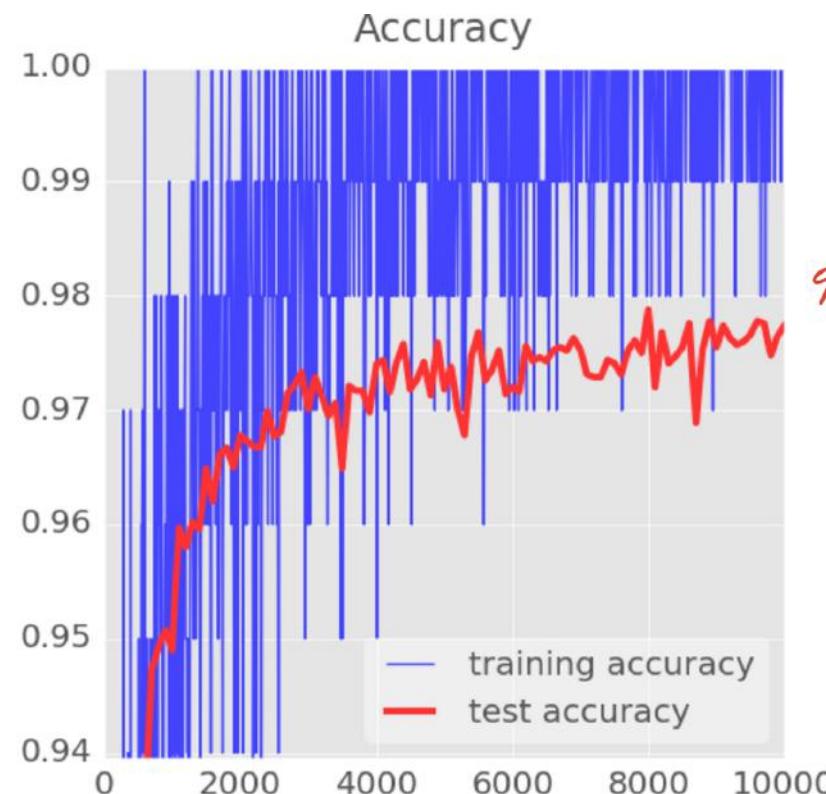
Multi-layered Perceptron

- Let's Run ! : ./tensorflow-mnist/mnist_2.3_five_layers_relu_lrdecay_dropout.py



Multi-layered Perceptron

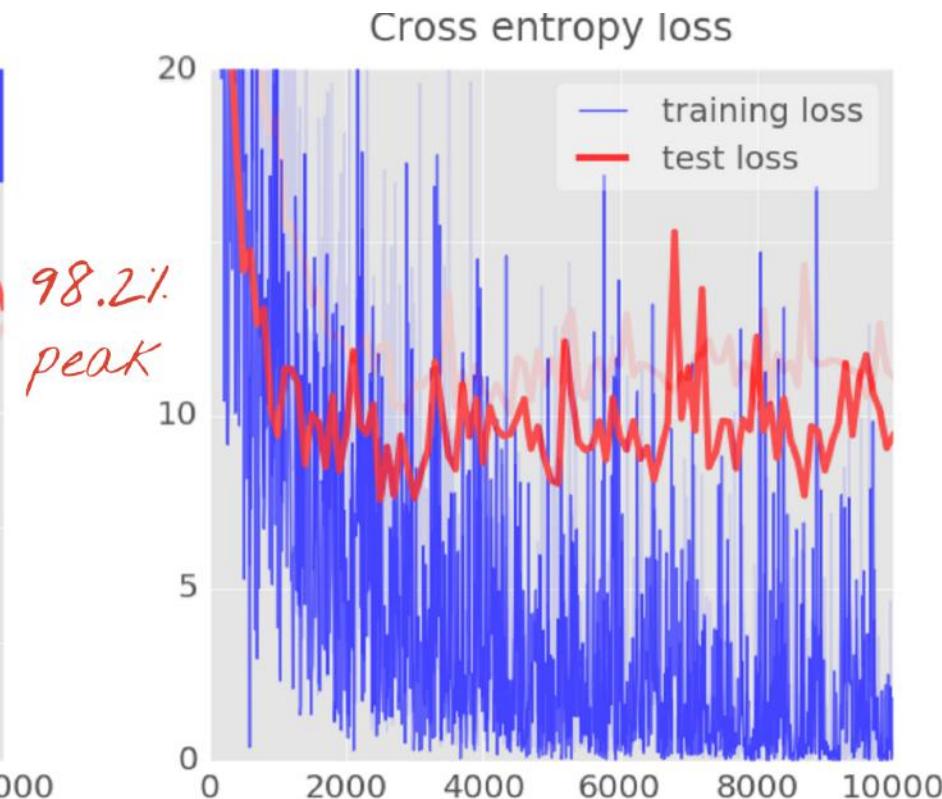
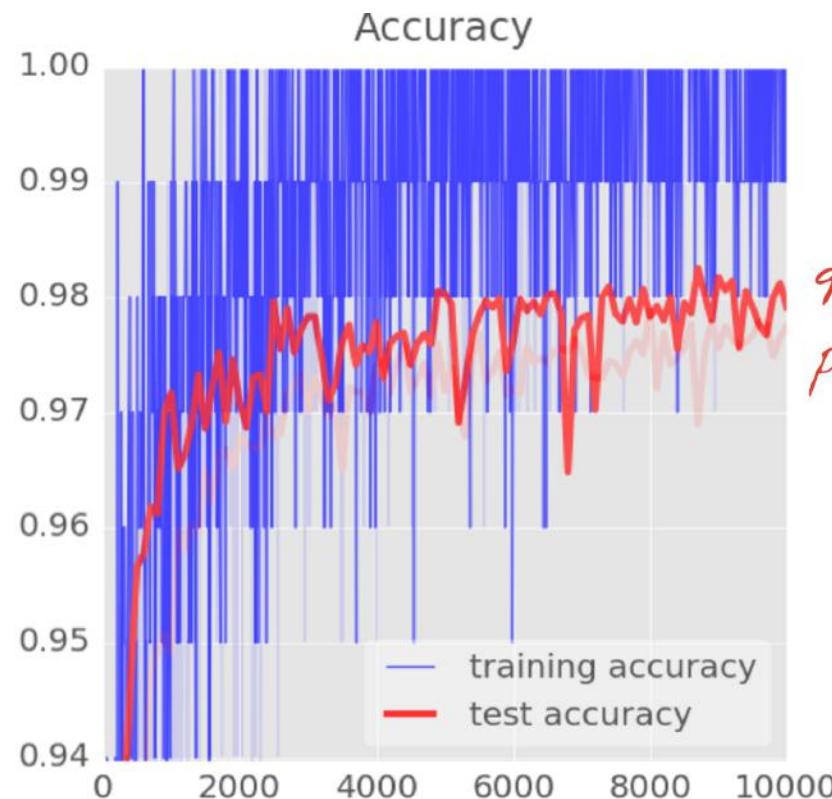
- All the Recipes



Sigmoid, learning rate = 0.003

Multi-layered Perceptron

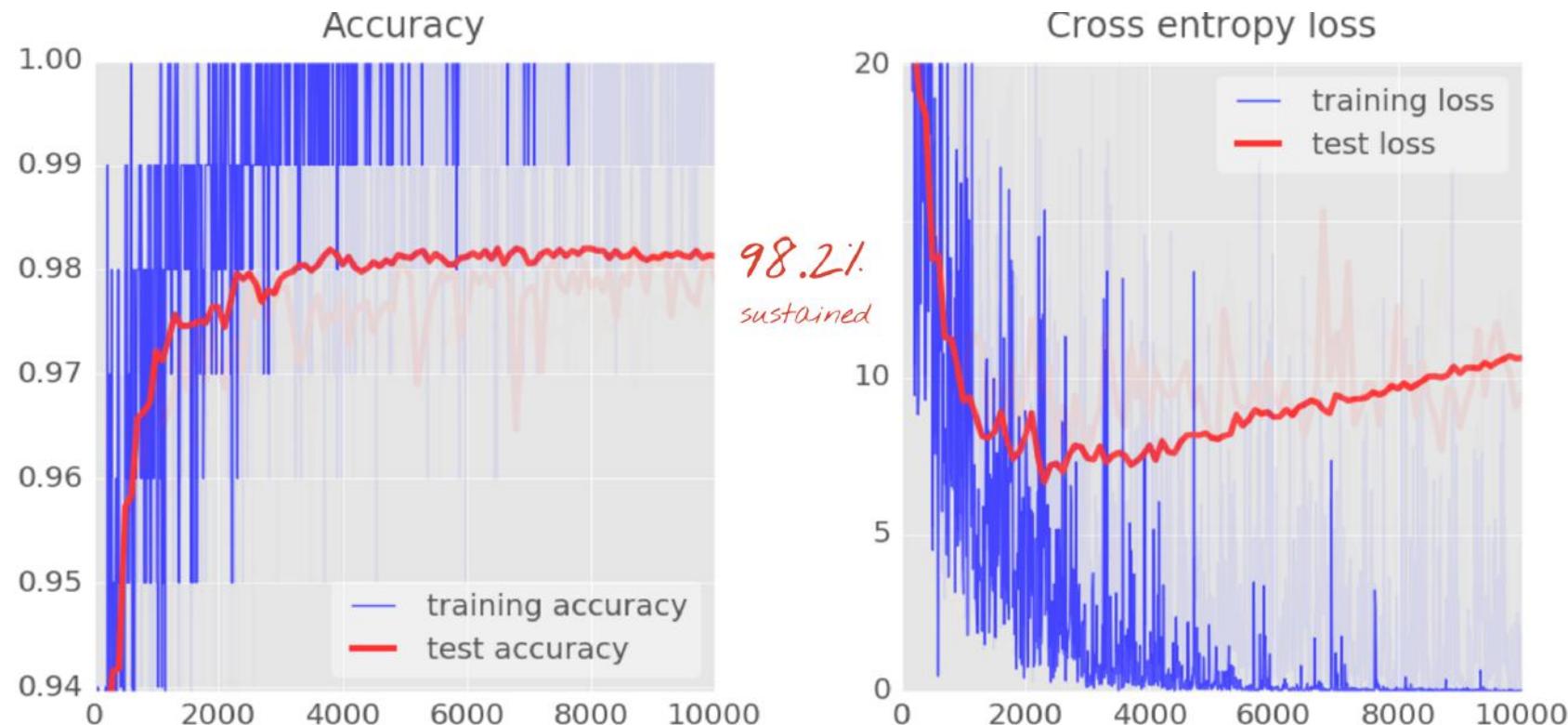
- All the Recipes



RELU, learning rate = 0.003

Multi-layered Perceptron

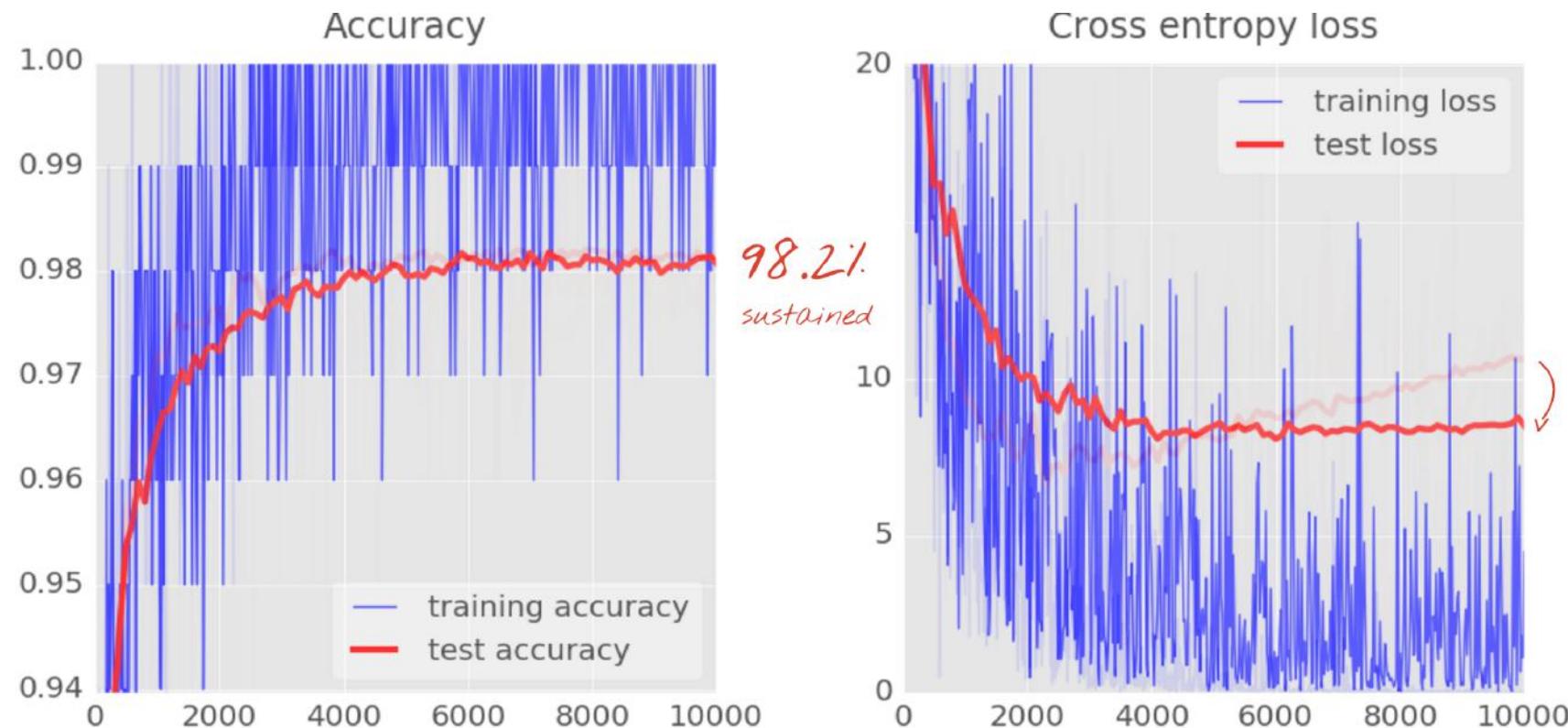
- All the Recipes



RELU, decaying learning rate 0.003 → 0.0001

Multi-layered Perceptron

- All the Recipes



RELU, decaying learning rate 0.003 → 0.0001 and dropout 0.75

Multi-layered Perceptron

- One-more Thing : `tf.get_variable()`

```
tf.get_variable(name, shape=None, dtype=None, initializer=None,  
regularizer=None, trainable=True, collections=None, caching_device=None,  
partitioner=None, validate_shape=True, custom_getter=None)
```

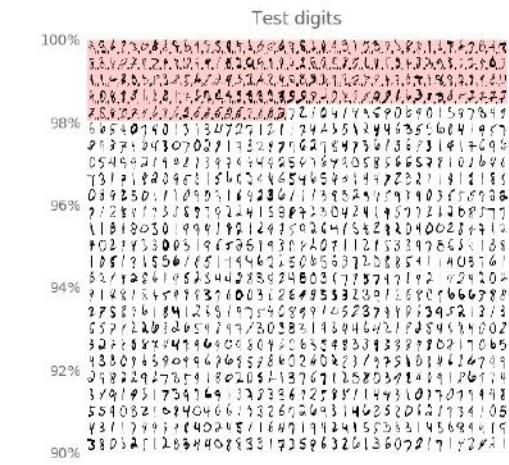
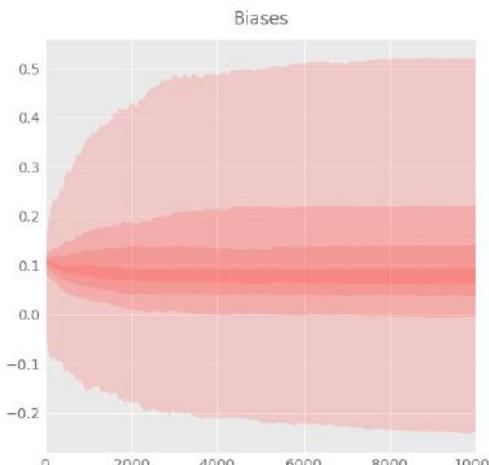
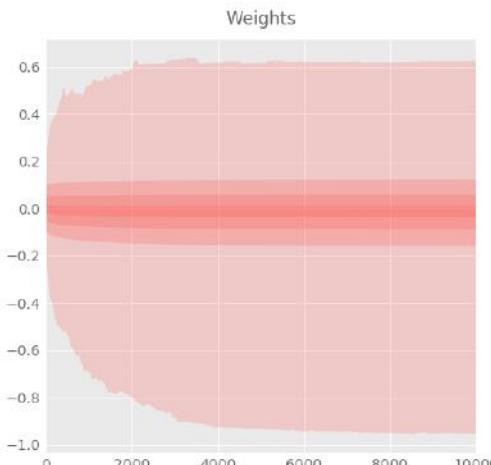
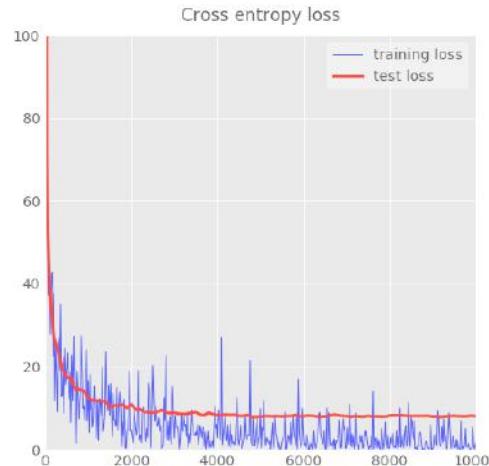
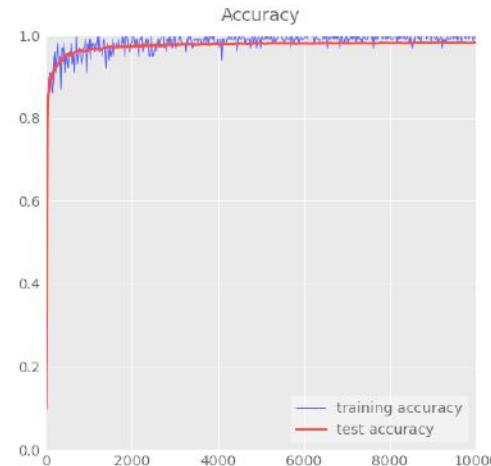
Gets an existing variable with these parameters or create a new one.

This function prefixes the name with the current variable scope and performs reuse checks. See the [Variable Scope How To](#) for an extensive description of how reusing works. Here is a basic example:

```
with tf.variable_scope("foo"):  
    v = tf.get_variable("v", [1]) # v.name == "foo/v:0"  
    w = tf.get_variable("w", [1]) # w.name == "foo/w:0"  
with tf.variable_scope("foo", reuse=True)  
    v1 = tf.get_variable("v") # The same as v above.
```

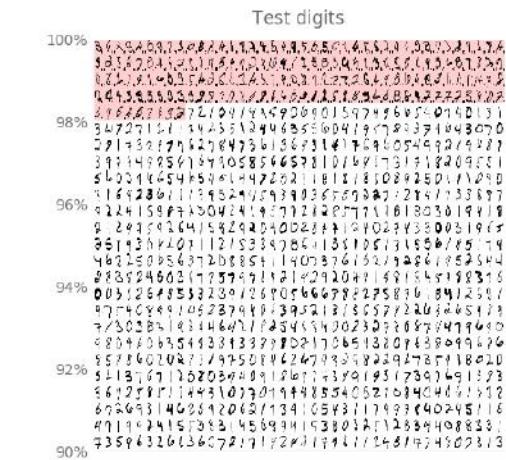
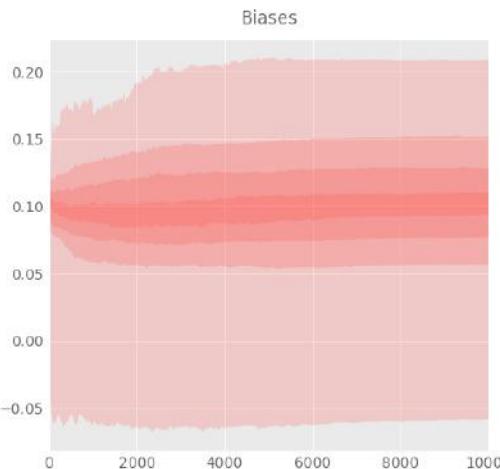
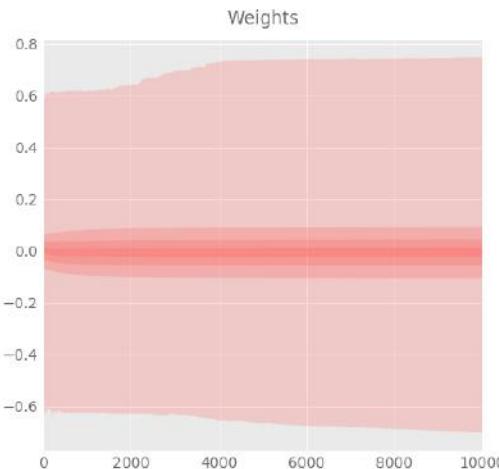
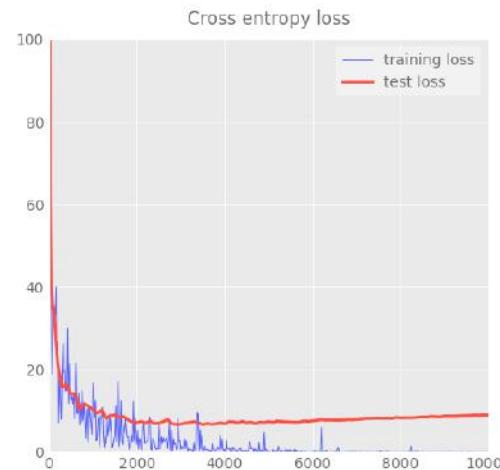
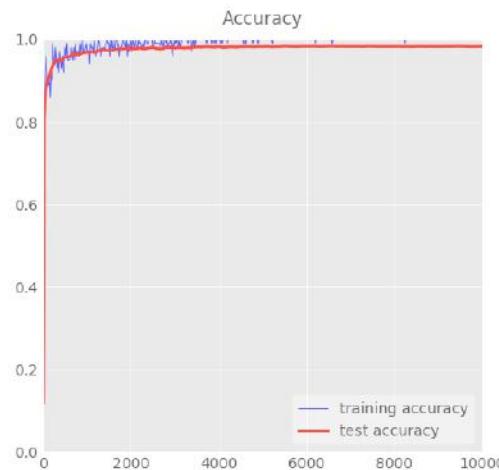
Multi-layered Perceptron

- Before : 98.25%



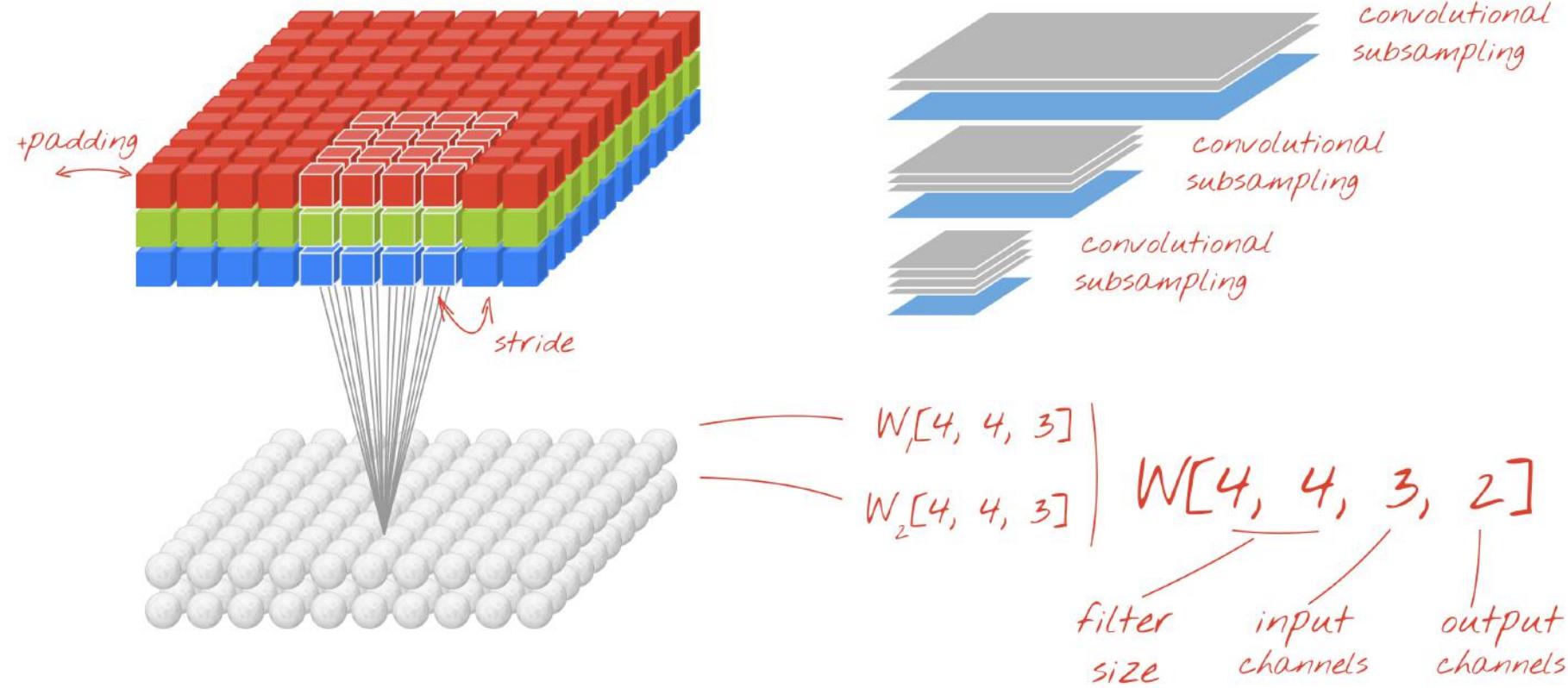
Multi-layered Perceptron

- After : 98.39%



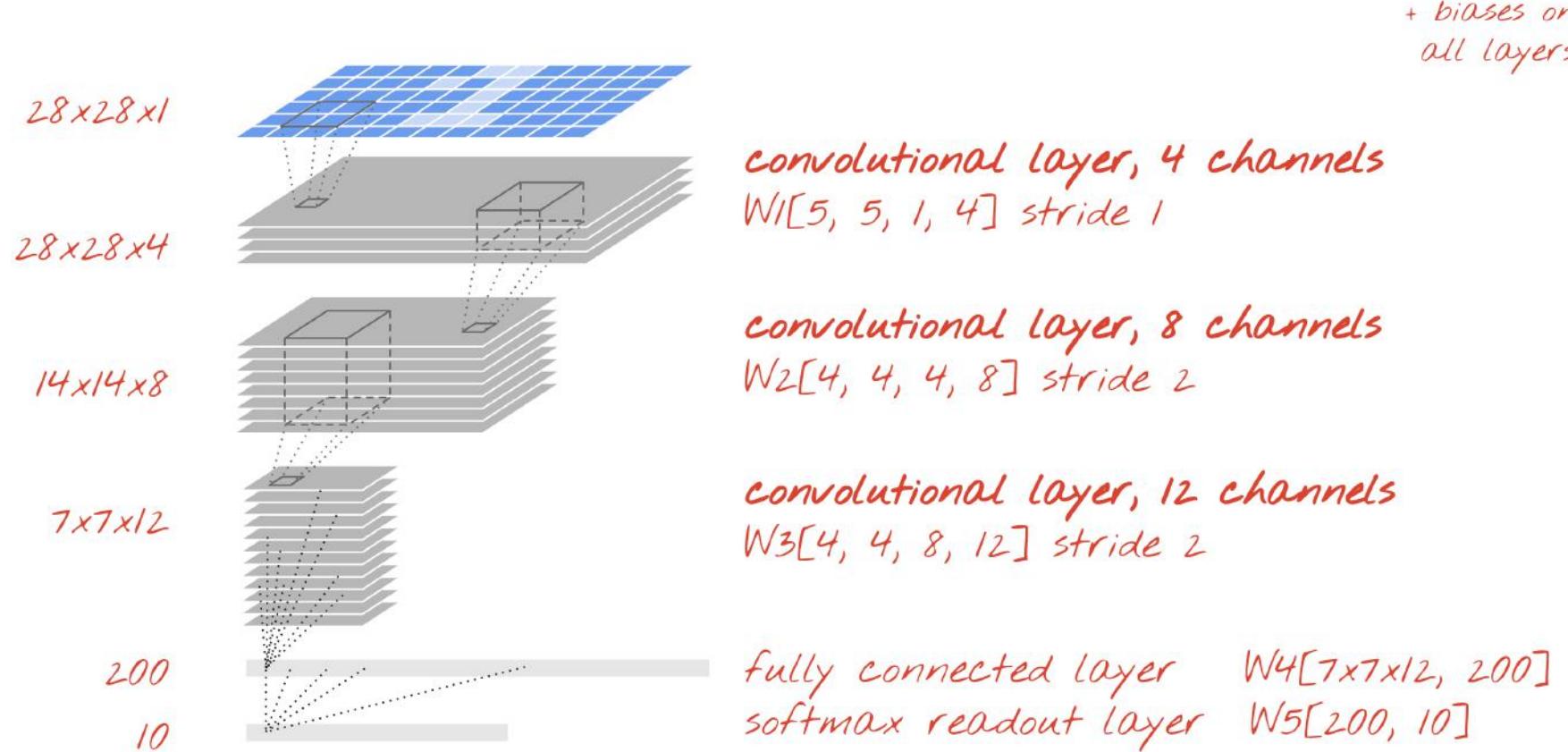
Convolutional Neural Network

- Convolution Layer



Convolutional Neural Network

- Architecture – Smaller Network



Convolutional Neural Network

▪ Code – Smaller Network

K=4
L=8
M=12

```
W1 = tf.Variable(tf.truncated_normal([5, 5, 1, K], stddev=0.1))  
B1 = tf.Variable(tf.ones([K])/10)  
W2 = tf.Variable(tf.truncated_normal([5, 5, K, L], stddev=0.1))  
B2 = tf.Variable(tf.ones([L])/10)  
W3 = tf.Variable(tf.truncated_normal([4, 4, L, M], stddev=0.1))  
B3 = tf.Variable(tf.ones([M])/10)
```

N=200

```
W4 = tf.Variable(tf.truncated_normal([7*7*M, N], stddev=0.1))  
B4 = tf.Variable(tf.ones([N])/10)  
W5 = tf.Variable(tf.truncated_normal([N, 10], stddev=0.1))  
B5 = tf.Variable(tf.zeros([10])/10)
```

filter size input channels output channels

weights initialised with random values

input image batch
X[100, 28, 28, 1]

weights

stride

bias

```
Y1 = tf.nn.relu(tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME') +  
Y2 = tf.nn.relu(tf.nn.conv2d(Y1, W2, strides=[1, 2, 2, 1], padding='SAME') +  
Y3 = tf.nn.relu(tf.nn.conv2d(Y2, W3, strides=[1, 2, 2, 1], padding='SAME') +
```

```
YY = tf.reshape(Y3, shape=[-1, 7 * 7 * M])
```

```
Y4 = tf.nn.relu(tf.matmul(YY, W4) + B4)
```

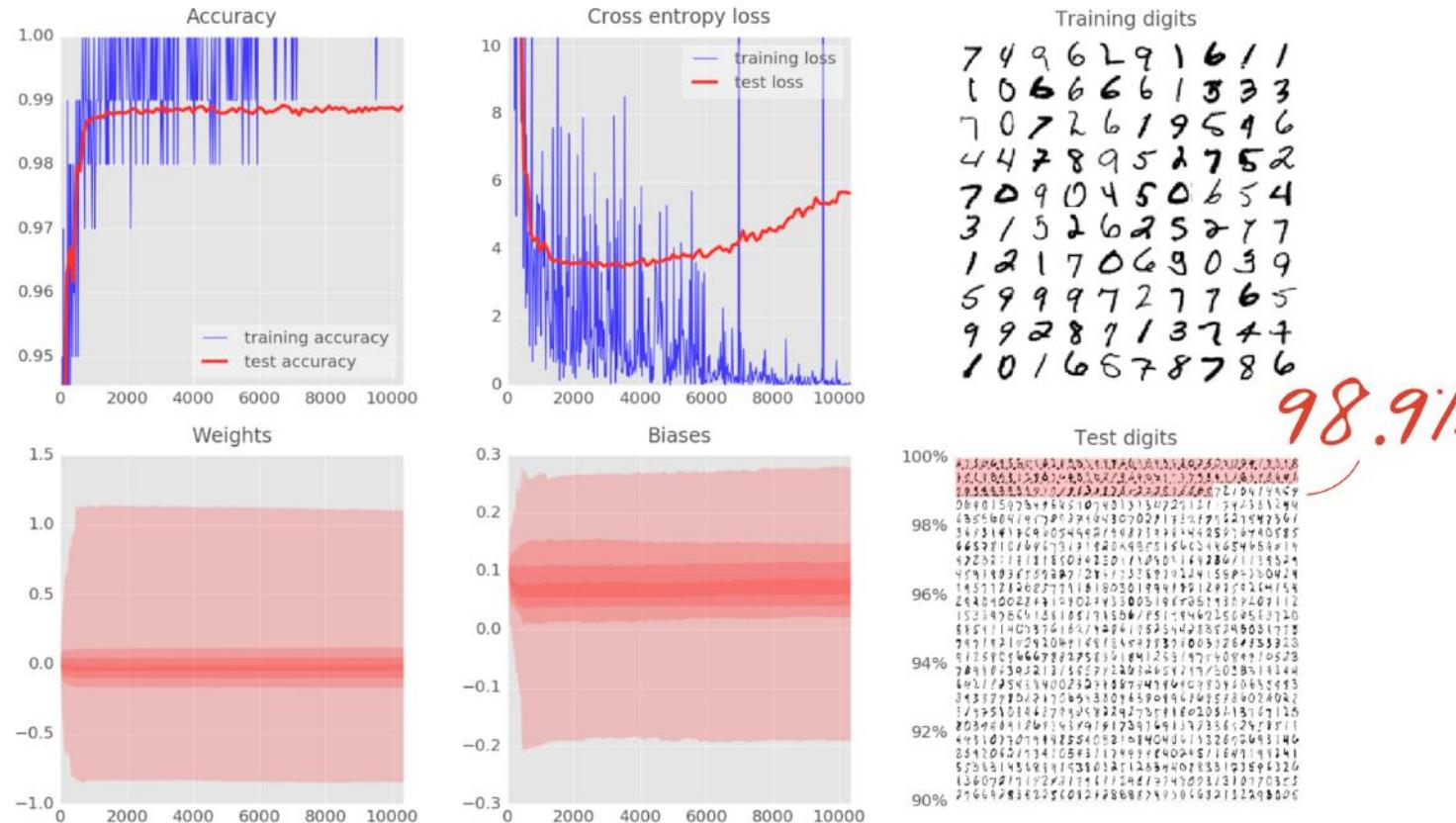
```
Y = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
```

flatten all values for
fully connected layer

Y3
YY

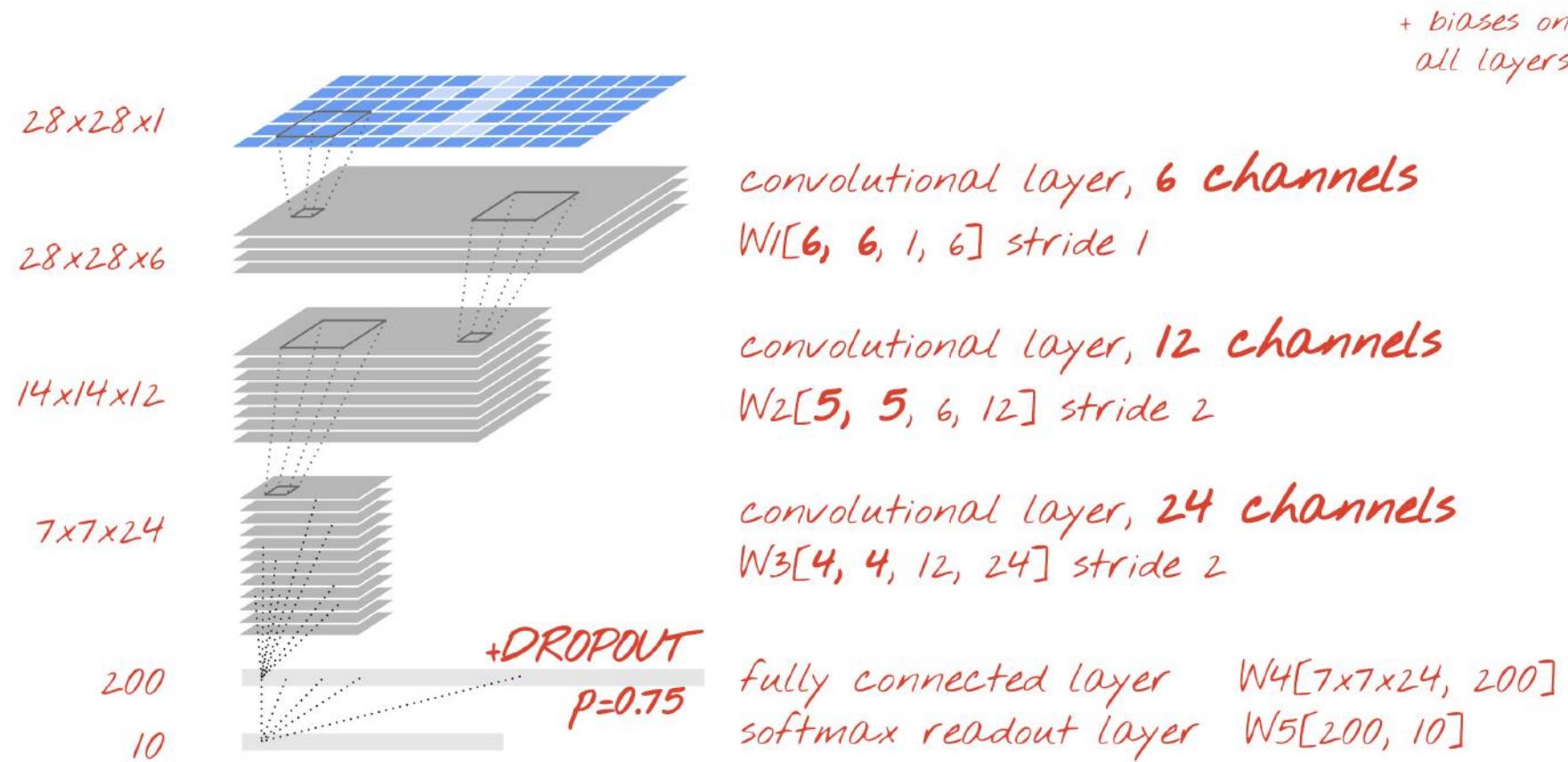
Convolutional Neural Network

- Let's Run – ./tensorflow-mnist/mnist_3.0_convolutional.py
 - Problem : Overfitting -> Bigger Network with Regularization



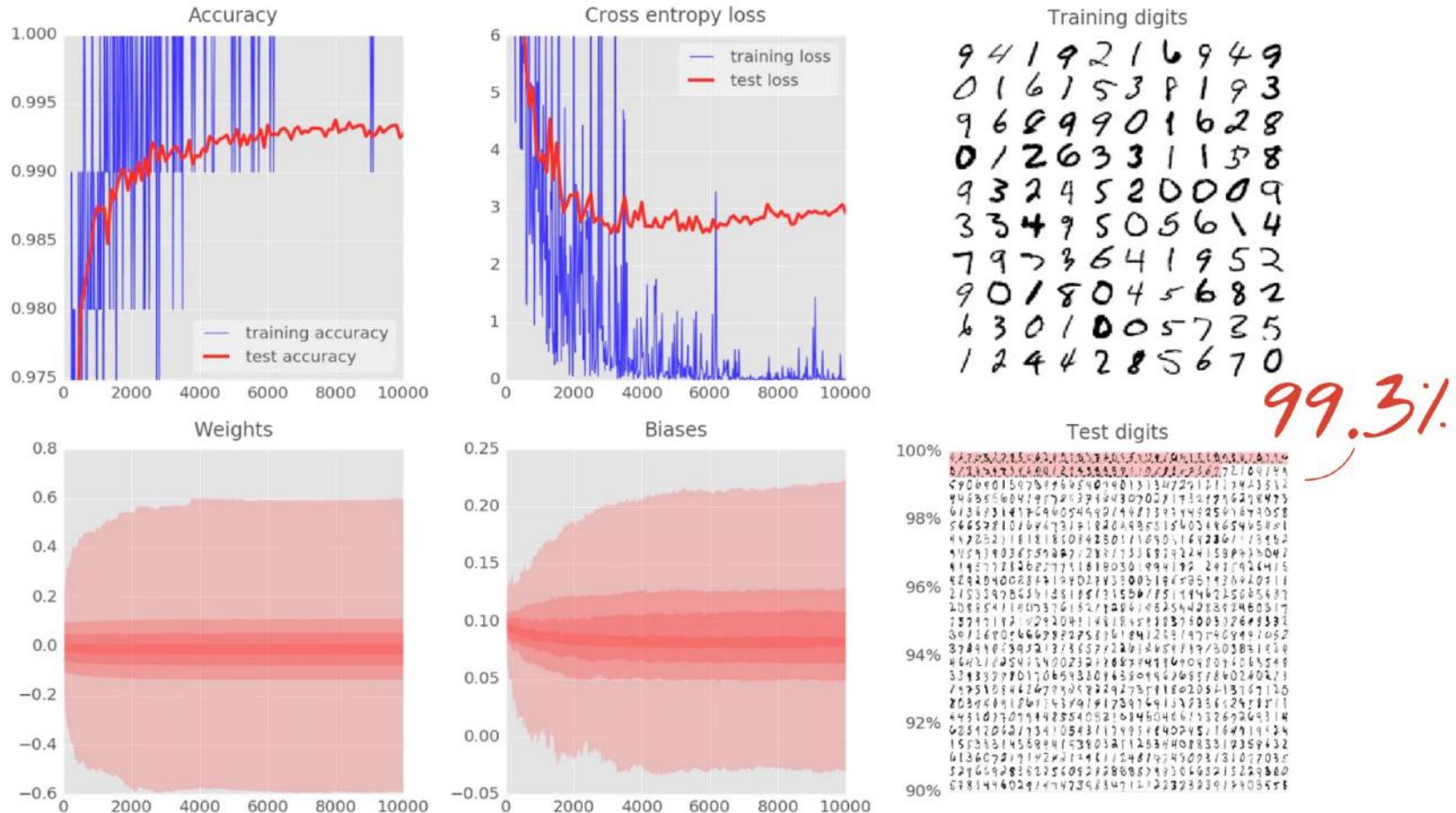
Convolutional Neural Network

- Architecture – Bigger Network



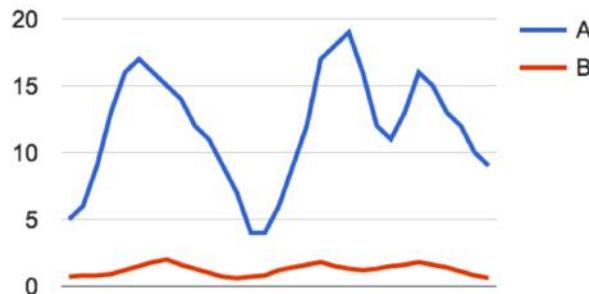
Convolutional Neural Network

- Let's Run ! : ./tensorflow-mnist/mnist_3.1_convolutional_bigger_dropout.py

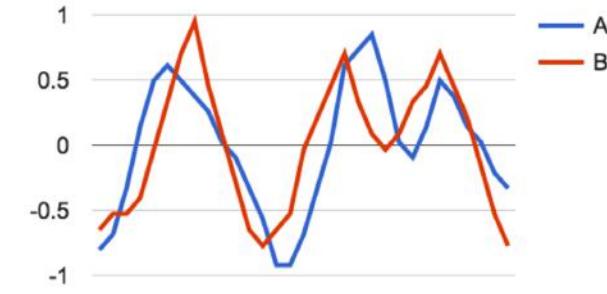
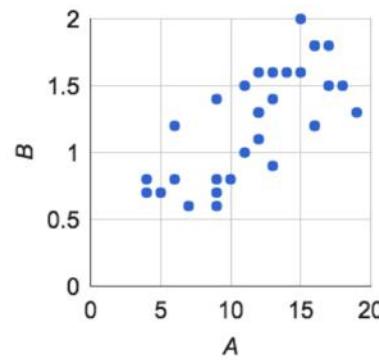


Batch Normalization

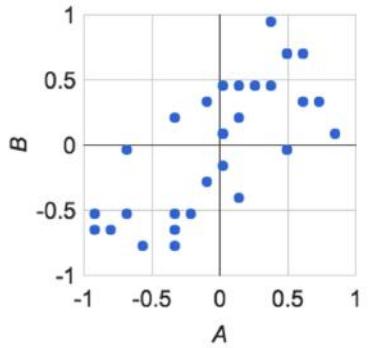
- Data Whitening



Data: large values, different scales, skewed, correlated



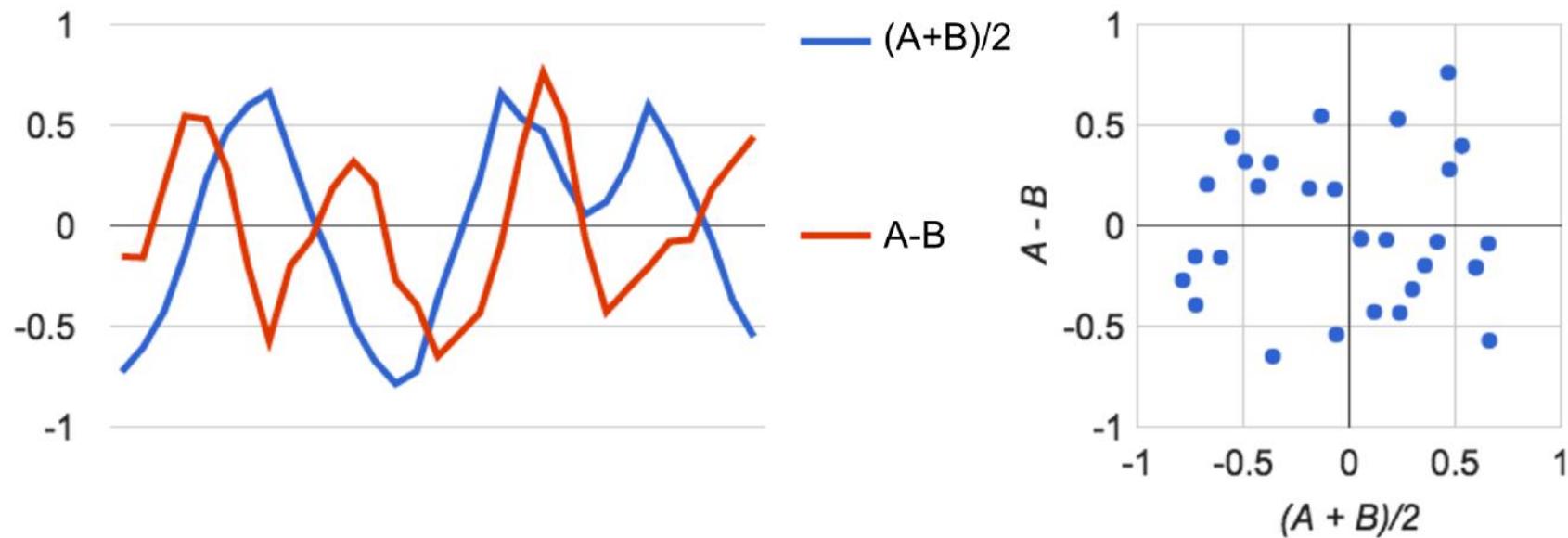
Modified data: centered around zero, rescaled...



*Subtract average
Divide by std dev*

Batch Normalization

- Data Whitening



Modified data: ... and decorrelated (that was almost a Principal Component Analysis)

Batch Normalization

- Data Whitening

$$\begin{bmatrix} \text{new } A & \text{new } B \end{bmatrix} = \begin{bmatrix} A & B \end{bmatrix} \times \begin{bmatrix} 0.05 & 0.12 \\ 0.61 & -1.23 \end{bmatrix} + \begin{bmatrix} -1.45 & 0.12 \end{bmatrix}$$

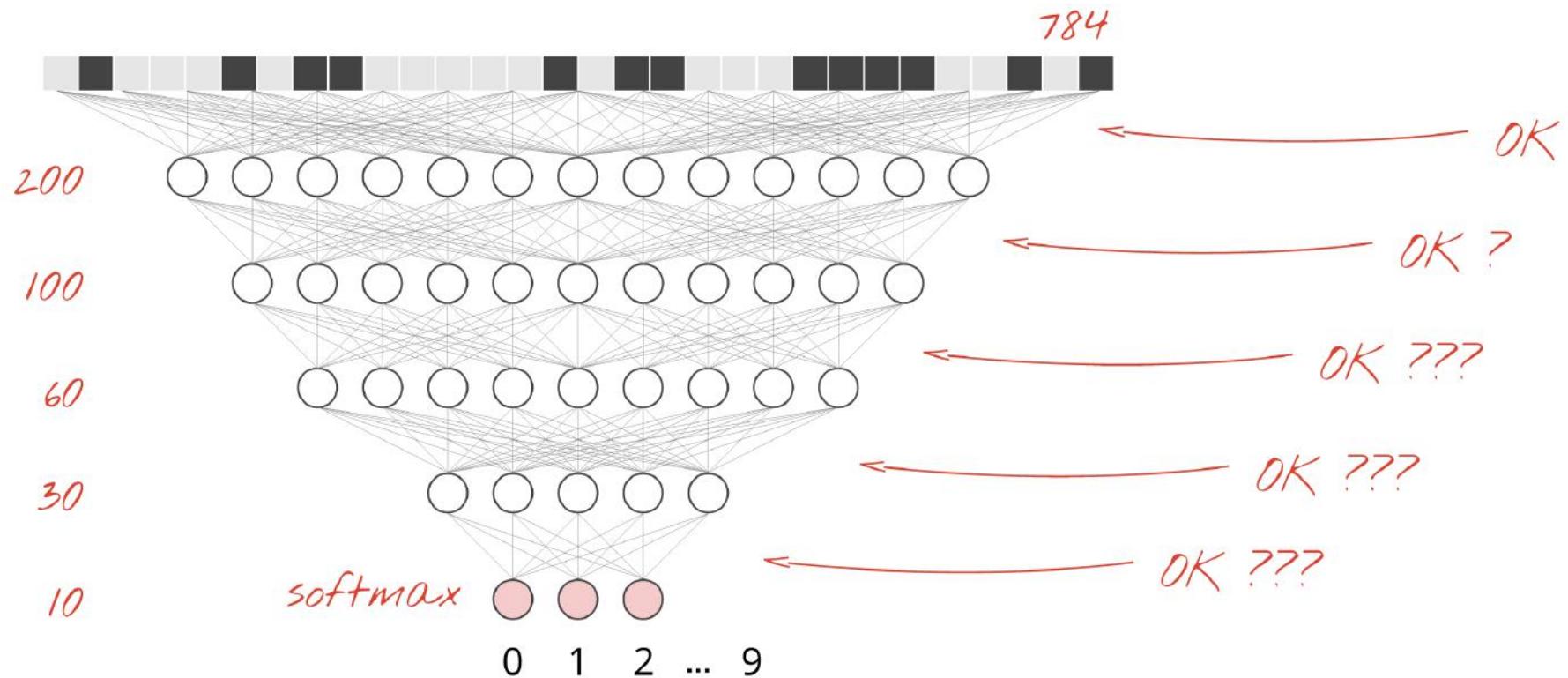
Scale & rotate shift

$W?$ $B?$

A network layer
can do this!

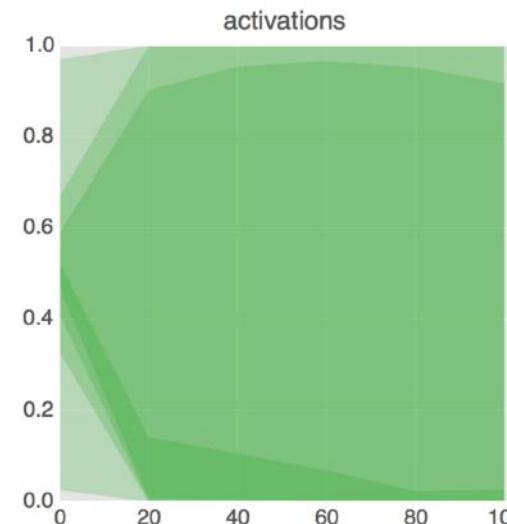
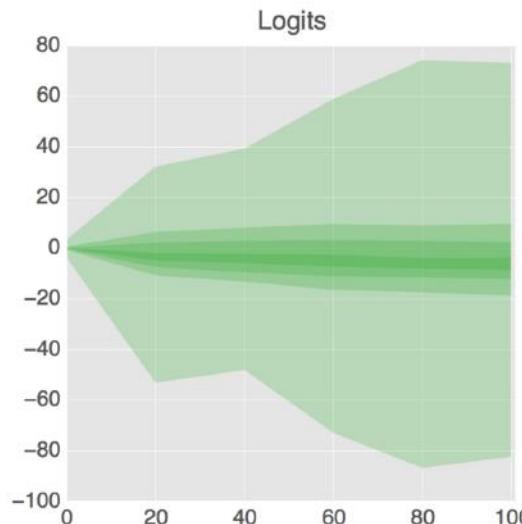
Batch Normalization

- Let's Get Back to MLP

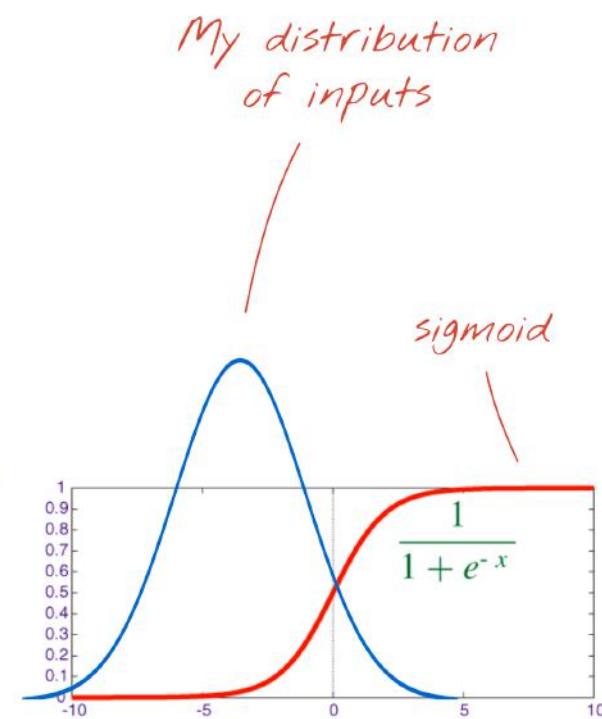


Batch Normalization

- Let's Get Back to MLP
 - Problem : Saturating Activations



boo-hoo
 σ



Batch Normalization

- Batch Normalization



Compute average and variance on mini-batch

"logit" = weighted sum + bias

Center and re-scale logits
before the activation function
(decorrelate ? no, too complex)



$$\hat{x} = \frac{x - \text{avgbatch}(x)}{\text{stdevbatch}(x) + \epsilon}$$

one of each per neuron



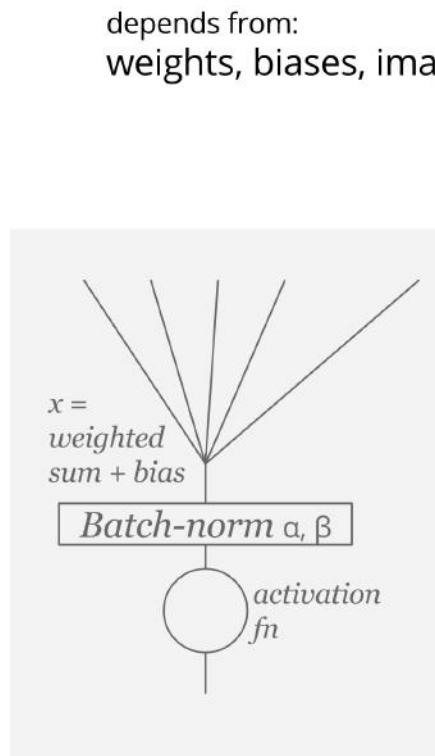
Add learnable scale and offset
for each logit so as to restore expressiveness

$$BN(x) = \alpha \hat{x} + \beta$$

Try $\alpha = \text{stdev}(x)$ and $\beta = \text{avg}(x)$ and you have $BN(x) = x$

Batch Normalization

- Batch Normalization



depends from:
weights, biases, images

$$\hat{x} = \frac{x - avg_{batch}(x)}{stdev_{batch}(x) + \epsilon}$$

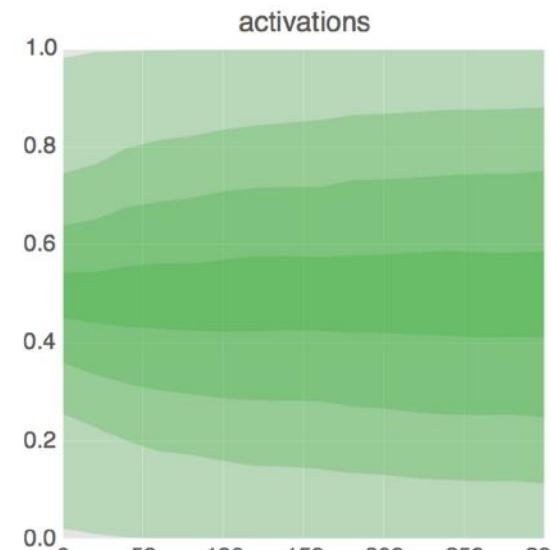
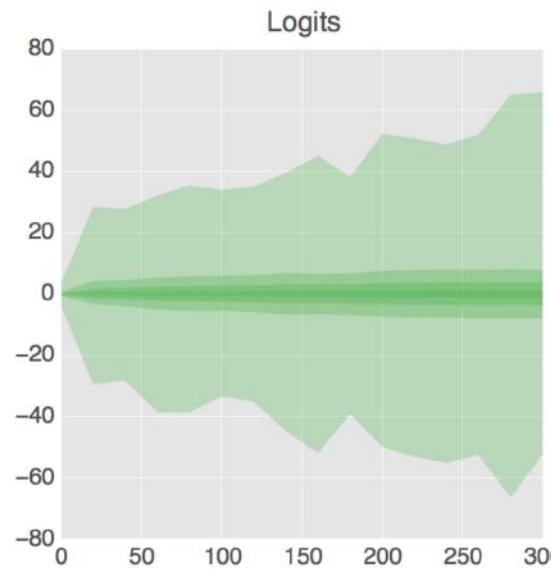
depends from:
same weights and biases, images
only one set of weights and biases in a mini-batch

$$BN(x) = \alpha \hat{x} + \beta$$

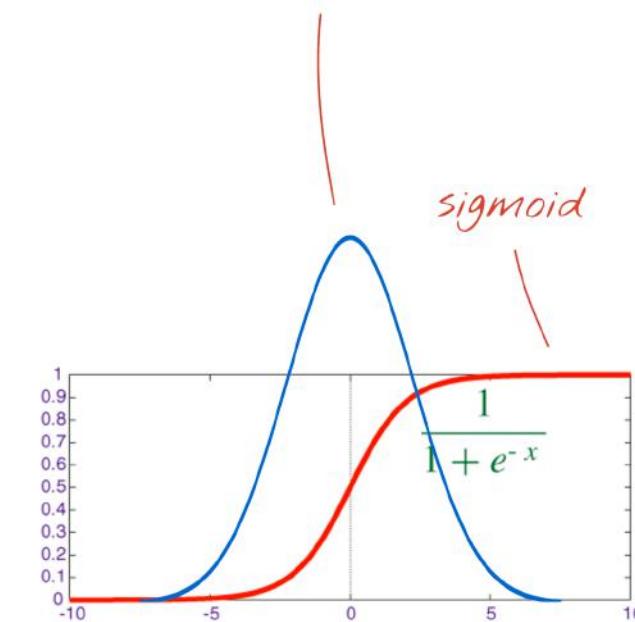
=> BN is differentiable relatively to weights, biases, α and β
It can be used as a layer in the network, gradient calculations will still work

Batch Normalization

- Let's Run ! : ./tensorflow-mnist/mnist_4.0_batchnorm_five_layers_sigmoid.py

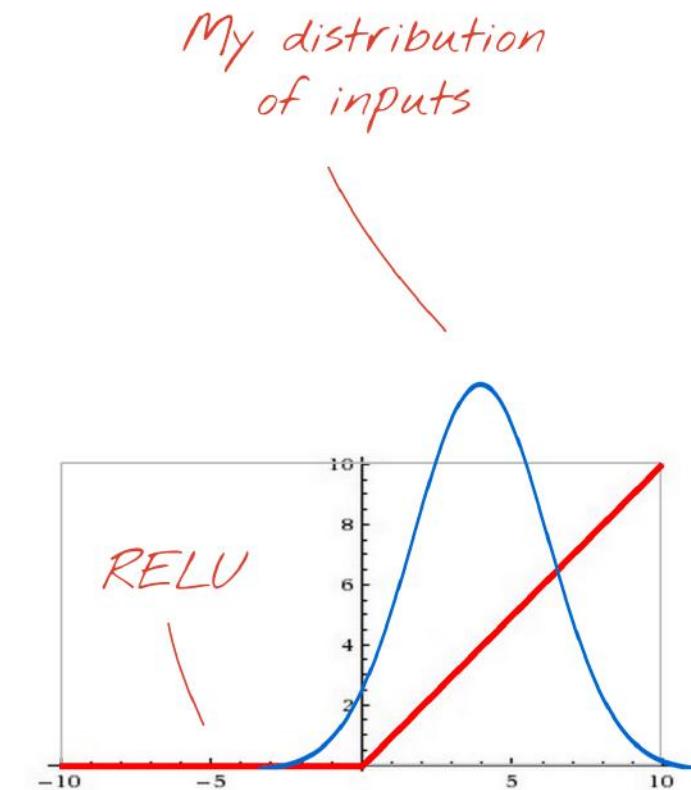
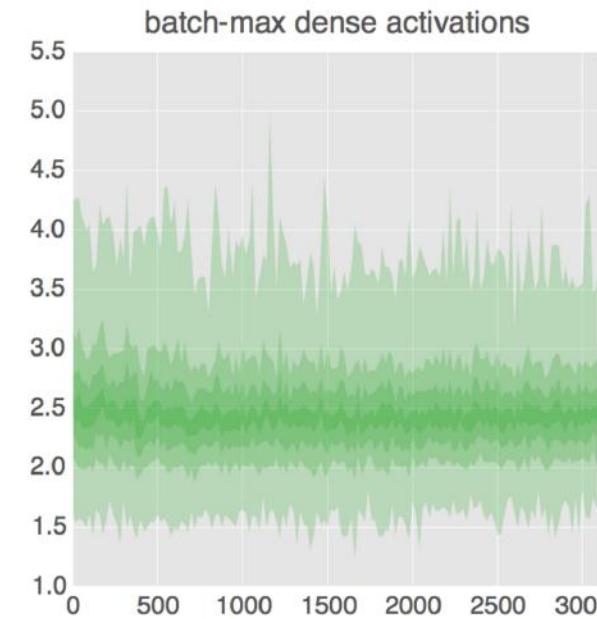
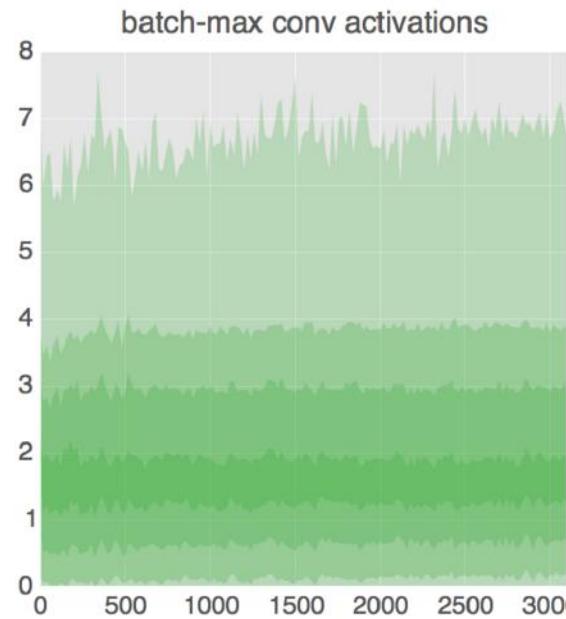


*My distribution
of inputs*



Batch Normalization

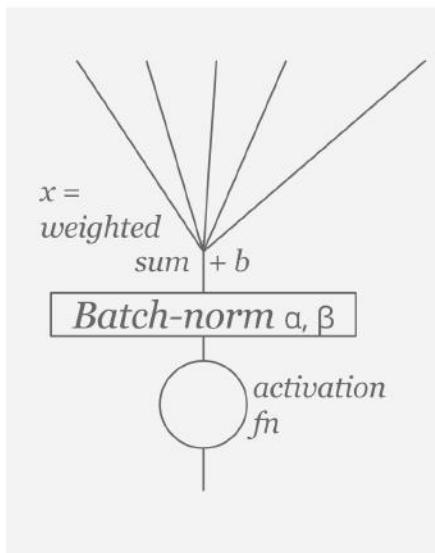
- Let's Run ! : ./tensorflow-mnist/mnist_4.1_batchnorm_five_layers_relu.py
 - Max activations for each neuron



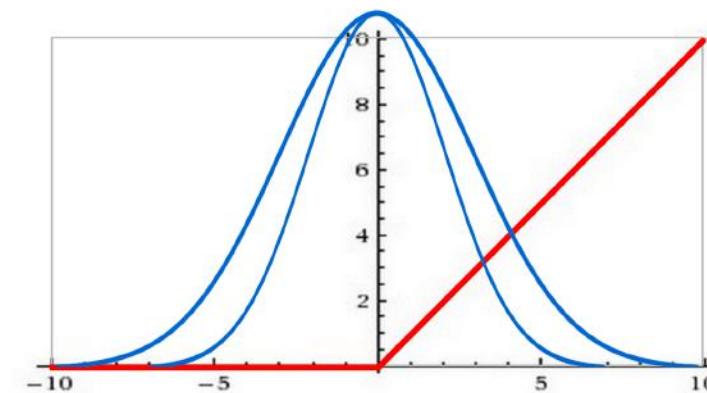
Batch Normalization

- Batch Normalization Done Right

biases :
no longer useful



Per neuron:	relu	sigmoid
without BN	bias	bias
With BN	β	α, β

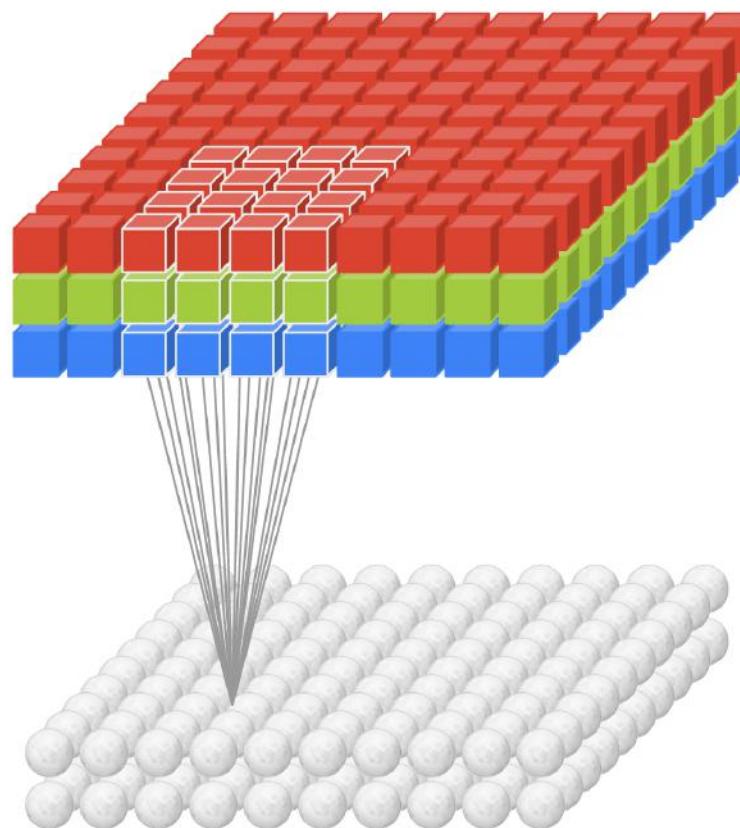


when activation fn is RELU
 α is not useful
It does not modify output distrib.

+You can go faster: use higher learning rate
+BN also regularises: lower or remove dropout

Batch Normalization

- Batch Normalization in Convolution Layers



Each neuron or patch has a value:

- per image in the batch
- per x position
- per y position

=> compute avg and stdev across all
batchsize x width x height values

$$\left. \begin{array}{l} W[4, 4, 3] \xrightarrow{b, \alpha, \beta_1} \\ W_2[4, 4, 3] \xrightarrow{b_2, \alpha_2, \beta_2} \end{array} \right\} \text{Still, one bias, scale or offset per neuron}$$

Batch Normalization

- Batch Normalization at Test Time

$$\hat{x} = \frac{x - \text{avg?}(x)}{\text{stdev?}(x) + \epsilon}$$

Stats on what?

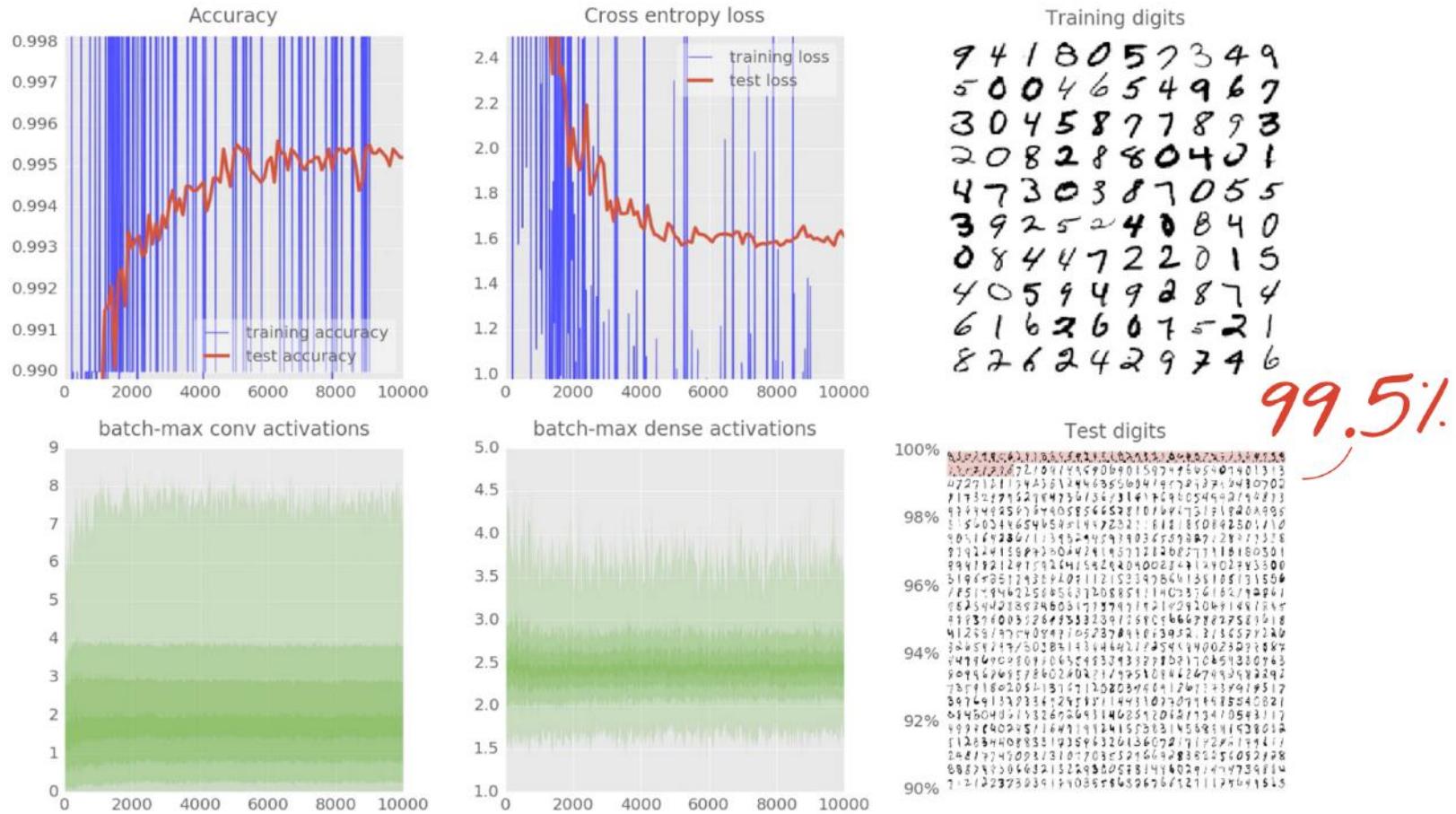
- Last batch: no
- all images: yes (but not practical)
- => Exponential moving average during training

```
def batchnorm_layer(Ylogits, is_test, Offset, Scale, iteration, convolutional=False):
    exp_moving_avg = tf.train.ExponentialMovingAverage(0.9999, iteration)
    if convolutional:      # avg across batch, width, height
        mean, variance = tf.nn.moments(Ylogits, [0, 1, 2])
    else:
        mean, variance = tf.nn.moments(Ylogits, [0])
    update_moving_averages = exp_moving_avg.apply([mean, variance])
    m = tf.cond(is_test, lambda: exp_moving_avg.average(mean), lambda: mean)
    v = tf.cond(is_test, lambda: exp_moving_avg.average(variance), lambda: variance)
    Ybn = tf.nn.batch_normalization(Ylogits, m, v, Offset, Scale, variance_epsilon=1e-5)
    return Ybn, update_moving_averages — don't forget to execute this (sess.run)
                                         ^ apply activation fn on Ybn
```

Define one offset and/or
scale per neuron

Batch Normalization

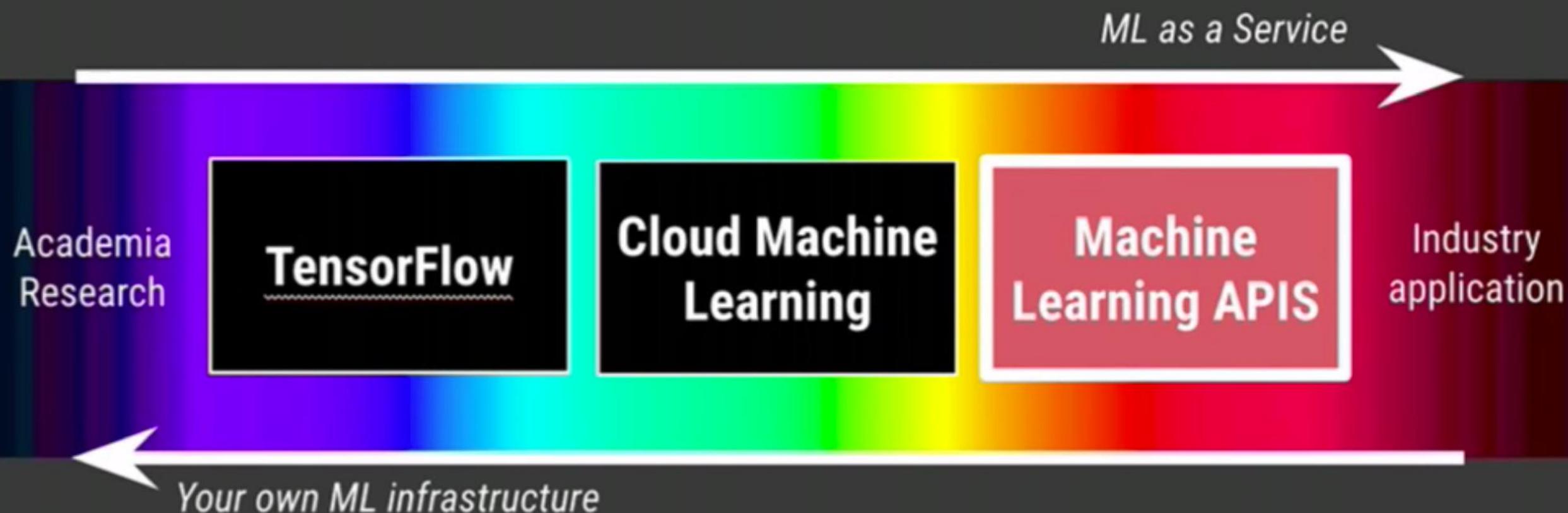
- Let's Run ! : ./tensorflow-mnist/mnist_4.2_batchnorm_convolutional.py



Class 12

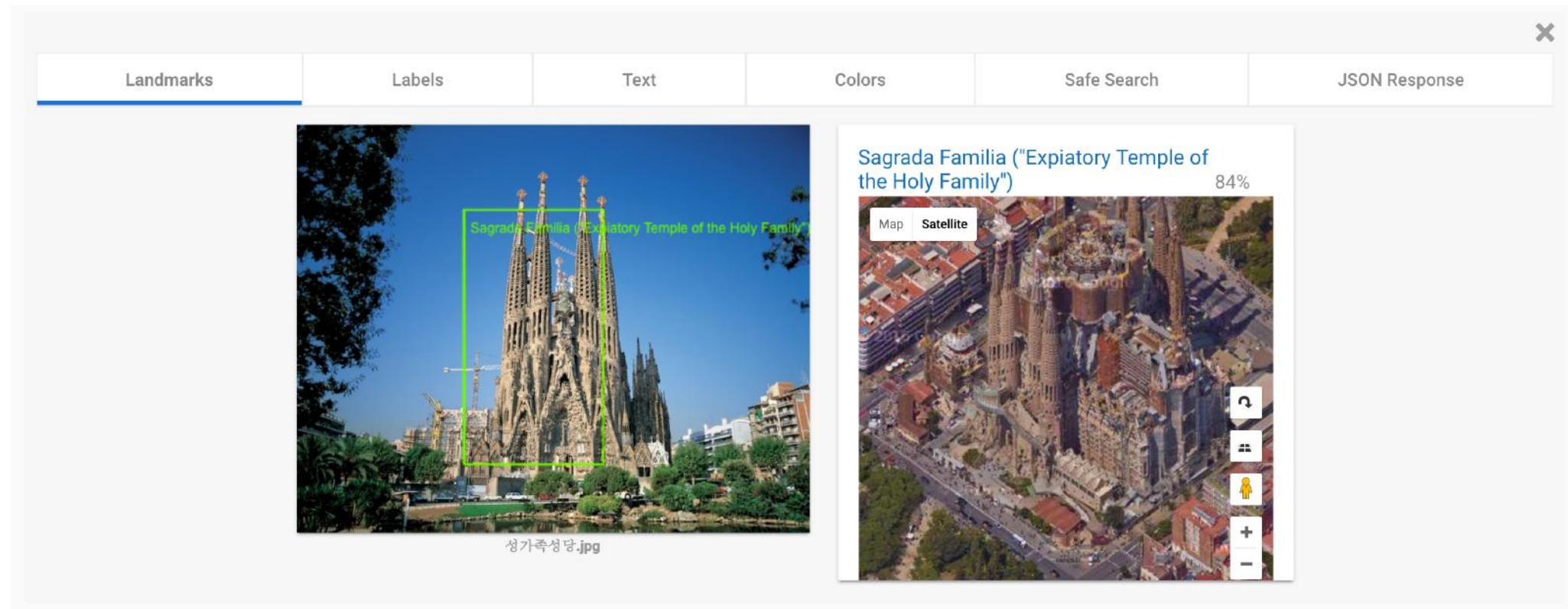
Google Cloud Platforms for Service Development

The Machine Learning Spectrum



Google Cloud Vision API

- <https://cloud.google.com/vision/>



Google Cloud Vision API

Feature Type	Description
LABEL_DETECTION	Add labels based on image content (see Detecting Labels)
TEXT_DETECTION	Perform Optical Character Recognition (OCR) on text within the image (see Detecting Text)
SAFE_SEARCH_DETECTION	Determine image safe search properties on the image (see Detecting Safe Search Properties)
FACE_DETECTION	Detect faces within the image (see Detecting Faces)
LANDMARK_DETECTION	Detect geographic landmarks within the image (see Detecting Landmarks)
LOGO_DETECTION	Detect company logos within the image (see Detecting Logos)
IMAGE_PROPERTIES	Compute a set of properties about the image (such as the image's dominant colors) (see Detecting Image Properties)

Google Cloud Vision API

Feature	1 - 1000 units/month	1001 - 1,000,000 units/month	1,000,001 to 5,000,000 units/month	5,000,001 - 20,000,000 units/month
Label Detection	Free	\$1.50	\$1.50	\$1.00
OCR	Free	\$1.50	\$1.50	\$0.60
Explicit Content Detection	Free		Now free with Label Detection*	
Facial Detection	Free	\$1.50	\$1.50	\$0.60
Landmark Detection	Free	\$1.50	\$1.50	\$0.60
Logo Detection	Free	\$1.50	\$1.50	\$0.60
Image Properties	Free	\$1.50	\$1.50	\$0.60

Google Cloud Vision API – Using Cloud SDK

- Install Google Cloud SDK
 - <https://cloud.google.com/sdk/docs/>
- Install Google Cloud Vision API
 - <https://cloud.google.com/vision/docs/reference/libraries#client-libraries-install-python>
- Hands-on
 - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part3/notebook

Google Cloud Vision API

- Using Vision API outside of Google Cloud SDK

1. Create Google Cloud Platform Project

Head over to the Google Cloud Platform Developers Console and login with your account. Ensure that you have a [Billing Account setup](#) with Google Cloud Platform.

Go to the [list of all projects](#) for your account and click on the Create Project button to create a new project. This will bring up the form as shown below:

The screenshot shows the 'New Project' dialog box. It has a title bar 'New Project'. Below it is a 'Project name' field containing 'Cloud Vision Project'. Underneath is a note: 'Your project ID will be cloud-vision-project-1249' with an 'Edit' link. A 'Billing account' dropdown menu is shown, with one item visible. At the bottom are two buttons: a blue 'Create' button and a white 'Cancel' button.

Enter a name for your project and select the Billing account that you want to apply to your project. Click on Create.

Google Cloud Vision API

- Using Vision API outside of Google Cloud SDK

2. Enable Google Cloud Vision API for Your Project

The next step is to enable the Cloud Vision API for your project. Click on the hamburger menu on top and then on API Manager as shown below:



The Cloud Vision API is not enabled for your project. Google provides several APIs and we need to find the Cloud Vision API. Enter the word "Vision" in the filter field as shown below and Cloud Vision API should come up as shown below:

Google Cloud Vision API

- Using Vision API outside of Google Cloud SDK

3. Getting Credentials

Visit the API Manager link again from the hamburger menu and click on Credentials. This will bring up a choice of what kind of Authentication mechanism that we want. For maximum flexibility, we will go with the Service Account Key, which will allow us access to the Vision API from applications running and is recommended for production use. Plus with the Application Credentials feature, authentication and authorization has been greatly simplified for the Google APIs.

Once you select the Service Account Key, you will see a screen as shown below:

The screenshot shows a dialog box titled 'Create service account key'. At the top left is a back arrow icon. Below it is a 'Service account' dropdown menu with the placeholder 'Select...'. Underneath is a 'Key type' section with two options: 'JSON' (selected) and 'P12'. A note states: 'Downloads a file that contains the public/private key pair. Store the file securely because this key can't be recovered if lost.' Below the key type are two radio buttons: 'JSON' (selected) and 'P12'. A note for P12 says: 'For backward compatibility with code using the P12 format'. At the bottom are 'Create' and 'Cancel' buttons.

Google Cloud Vision API

- Using Vision API outside of Google Cloud SDK

4. Use the Application Default Credentials

This feature makes it dead simple to integrate authentication to Google APIs in your applications. Check out the [documentation](#).

All we need to do is set an environment variable named GOOGLE_APPLICATION_CREDENTIALS and its value will be the JSON key file that we just downloaded in the previous step.

So go ahead and set the environment variable value as shown below:

On a Mac / Linux, you can do the following:

```
$ export GOOGLE_APPLICATION_CREDENTIALS=<Path To Your JSON Key File>
```

On Windows, you can do the following:

```
SET GOOGLE_APPLICATION_CREDENTIALS=<Path To Your JSON Key File>
```

Google Cloud Speech API

- <https://cloud.google.com/speech/>

Powerful Speech Recognition

Google Cloud Speech API enables developers to **convert audio to text** by applying **powerful neural network models** in an easy to use API. The API **recognizes over 80 languages and variants**, to support your global user base. You can transcribe the text of users dictating to an application, control through voice, or transcribe audio files, among many other use cases. **Recognize audio uploaded in the request**, and integrate with your audio storage on Google Cloud Storage, by using the same technology Google uses to power its own products.



Convert your voice to text right now

Click on the microphone icon to start recording

한국어 (대한민국)

날씨가 참 좋습니다

```
{
  "results": [
    {
      "alternatives": [
        {
          "transcript": "날씨가 참 좋습니다",
          "confidence": 0.96556205
        }
      ]
    }
  ]
}
```

Google Cloud Natural Language API

- <https://cloud.google.com/natural-language/>

Try the API

Google, headquartered in Mountain View, unveiled the new Android phone at the Consumer Electronic Show. Sundar Pichai said in his keynote that users love their new Android phones.

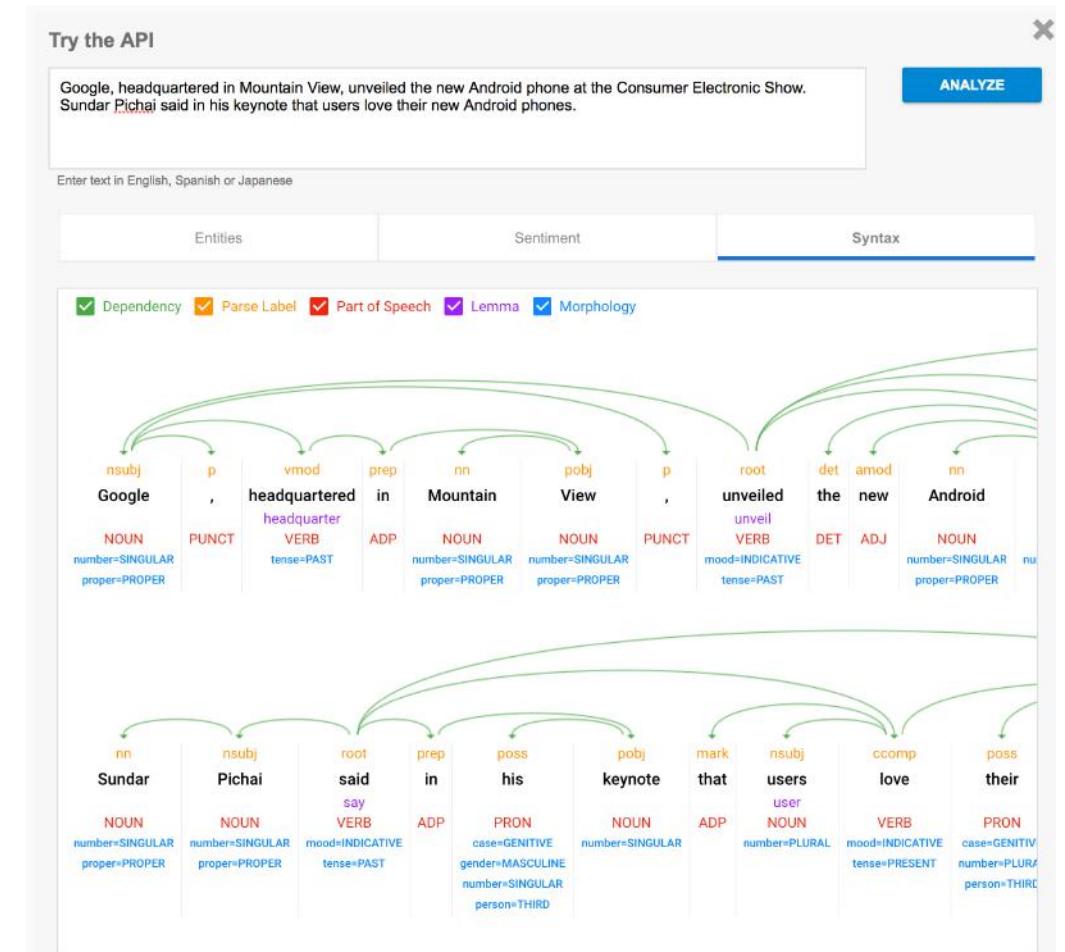
ANALYZE

Enter text in English, Spanish or Japanese

Entities Sentiment Syntax

(Google)₁, headquartered in (Mountain View)₆, unveiled the new (Android)₃ (phone)₂ at the (Consumer Electronic Show)₇. (Sundar Pichai)₅ said in his (keynote)₉ that (users)₄ love their new (Android)₃ (phones)₈.

E1 Google Wikipedia Article Salience: 0.26 [?]	ORGANIZATION	E2 phone Salience: 0.14 [?]	CONSUMER GOOD
E3 Android Wikipedia Article Salience: 0.13 [?]	CONSUMER GOOD	E4 users Salience: 0.12 [?]	PERSON
E5 Sundar Pichai Wikipedia Article Salience: 0.11 [?]	PERSON	E6 Mountain View Wikipedia Article Salience: 0.11 [?]	LOCATION
E7 Consumer Electronic Show Wikipedia Article Salience: 0.08 [?]	EVENT	E8 phones Salience: 0.03 [?]	CONSUMER GOOD
E9 keynote Salience: 0.02 [?]	OTHER		



Google Cloud Natural Language API

- <https://cloud.google.com/translate/>

TRY THE API

Source Language
한국어 (ko)

Target Language
영어 (en)

오늘 날씨가 정말 추워서 뒤질 것 같아요

Today the weather is really cold, I think dwijil

PREMIUM EDITION ⓘ
I think the weather is really cold today.

Translate Text

Detect Language

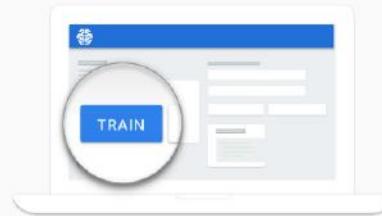
Supported Languages

Google Cloud Machine Learning

- <https://cloud.google.com/ml/>

Managed scalable machine learning

Google Cloud Machine Learning is a managed service that enables you to easily build machine learning models, that work on any type of data, of any size. Create your model with the powerful [TensorFlow](#) framework that powers many Google products, from [Google Photos](#) to [Google Cloud Speech](#). Build models of any size with our managed scalable infrastructure. Your trained model is immediately available for use with our global prediction platform that can support thousands of users and TBs of data. The service is integrated with [Google Cloud Dataflow](#) for pre-processing, allowing you to access data from [Google Cloud Storage](#), [Google BigQuery](#), and others.



Prediction at Scale

Seamlessly transition from training to prediction, using online and batch prediction services. Integration to Google global load balancing enables you to automatically scale your machine learning application, and reach users world-wide.



Build Machine Learning Models Easily

HyperTune lets you automatically tune your model training to achieve better results faster. Enable developers to easily build models using Cloud Datalab. Data Scientists can understand their data, create TensorFlow model graphs, train their models and analyze model quality.



Deep Learning Capabilities

Cloud Machine Learning supports any [TensorFlow](#) models - you can build and use models that can work on any type of data, across a whole variety of scenarios.



Fully Managed Service

Scalable and distributed training infrastructure for your largest data sets. Managed serverless infrastructure handles provisioning, scaling, and monitoring so that you can focus on building your models instead of handling clusters.



Neural Network Learning

Class 13

CNNs for Semantics Segmentation

Problem of Semantic Segmentation

- Level of Problem Difficulty

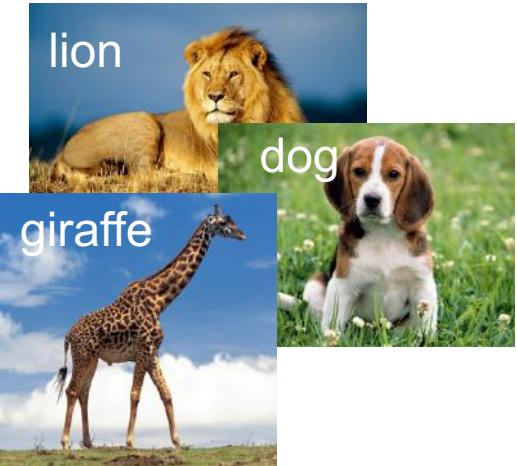
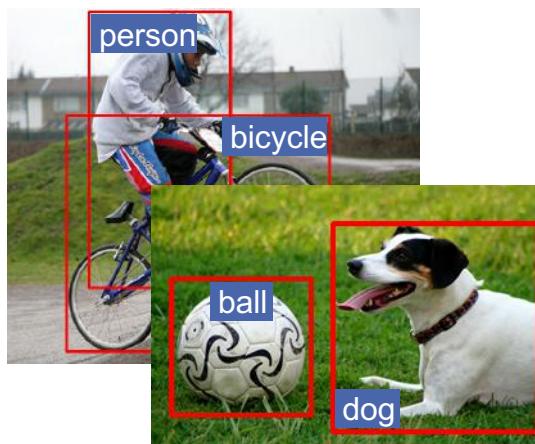


Image Classification



Object Detection

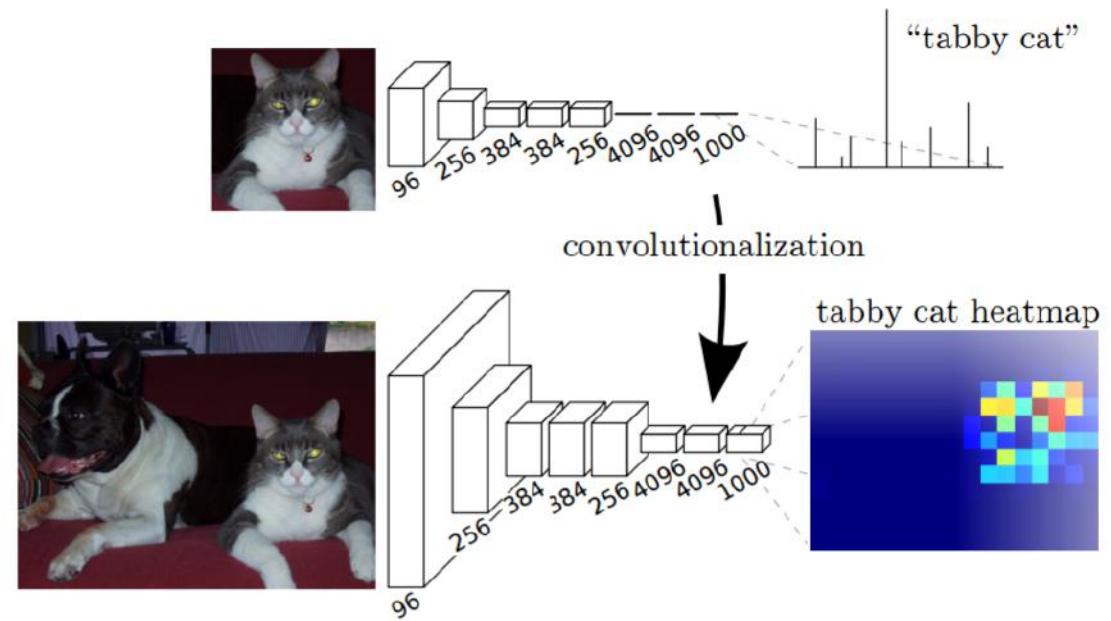
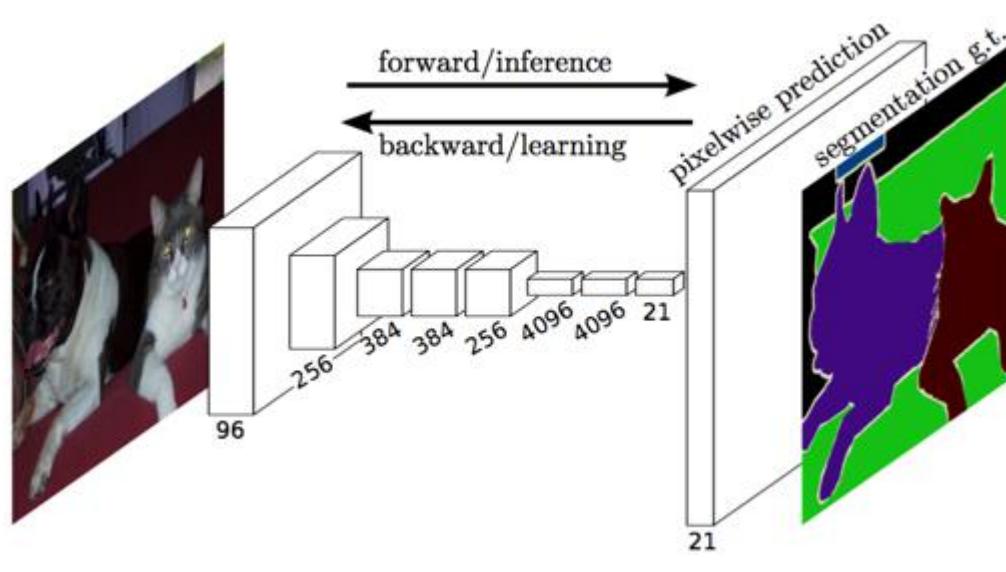


Semantic Segmentation



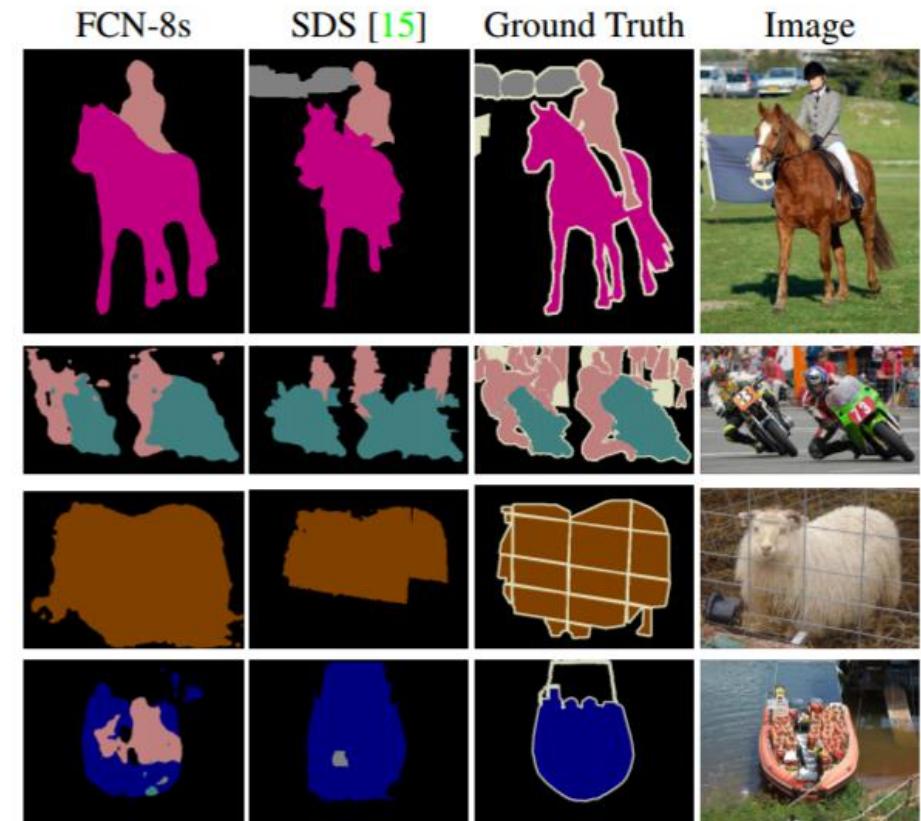
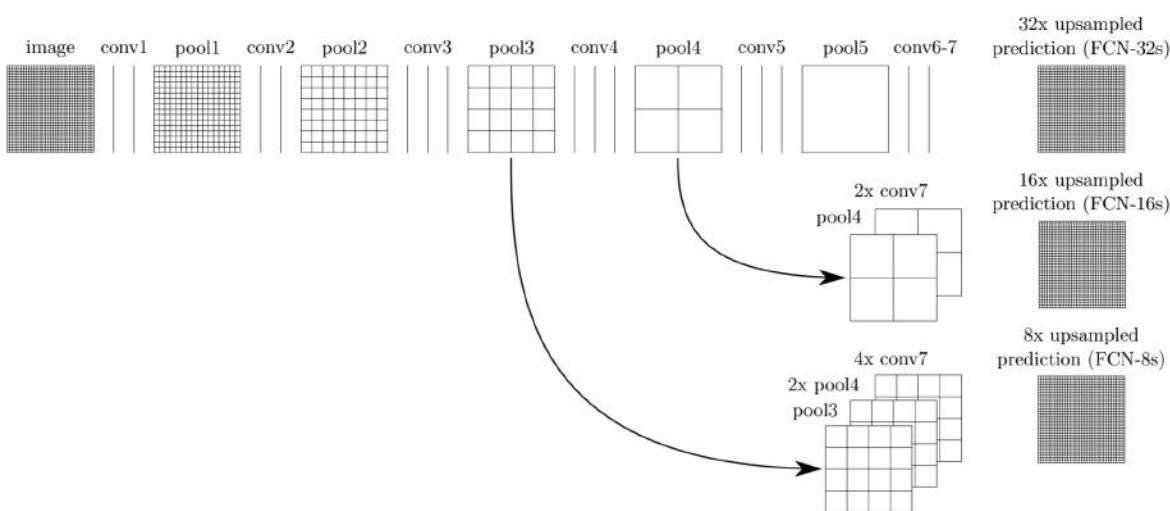
Fully Convolutional Neural Network (FCN)

- End-to-end CNN for Semantic Segmentation



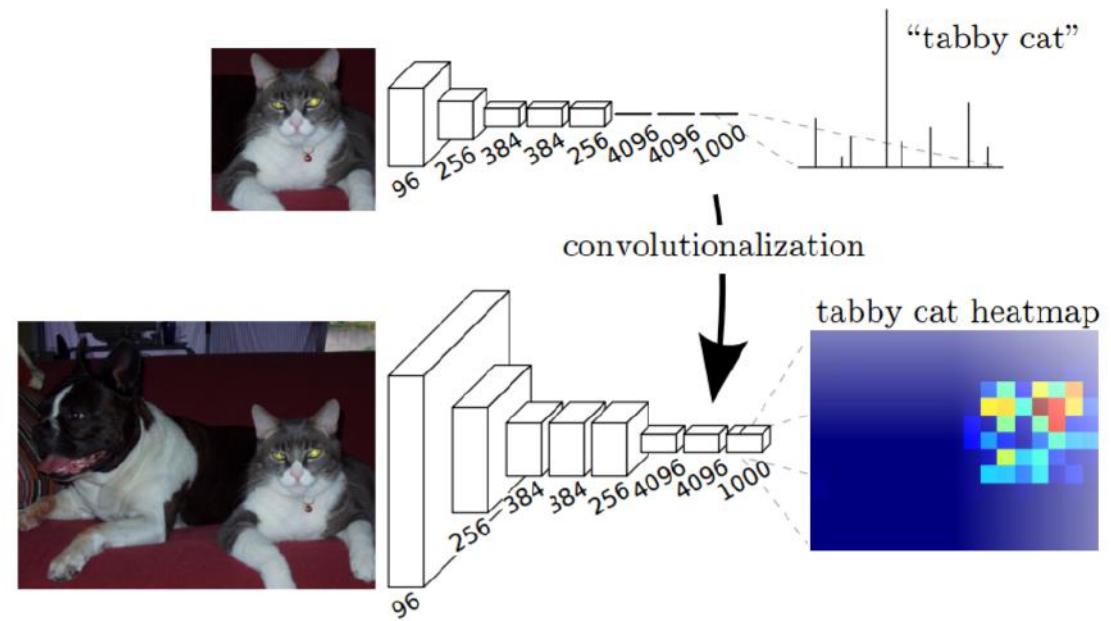
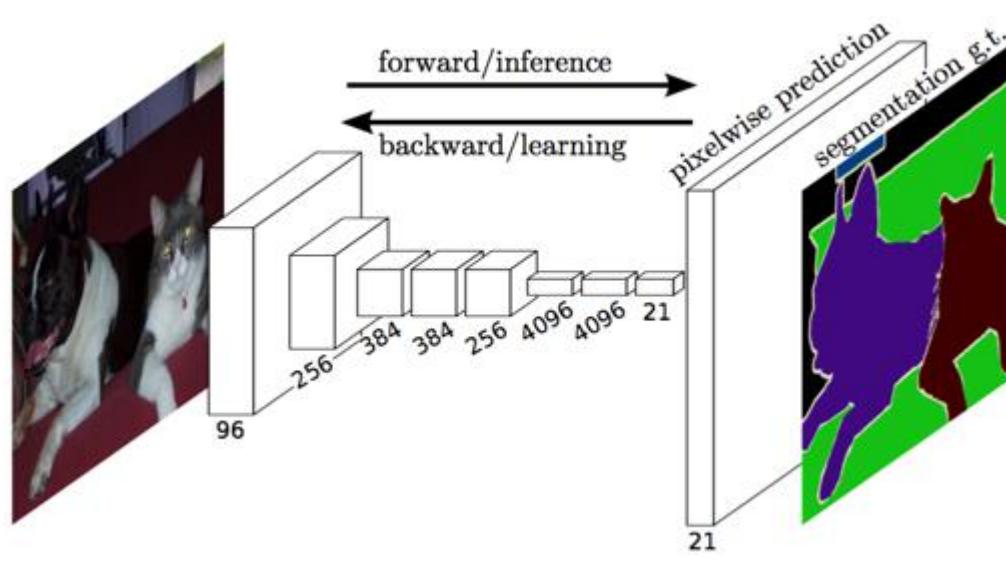
Fully Convolutional Neural Network (FCN)

- End-to-end CNN for Semantic Segmentation



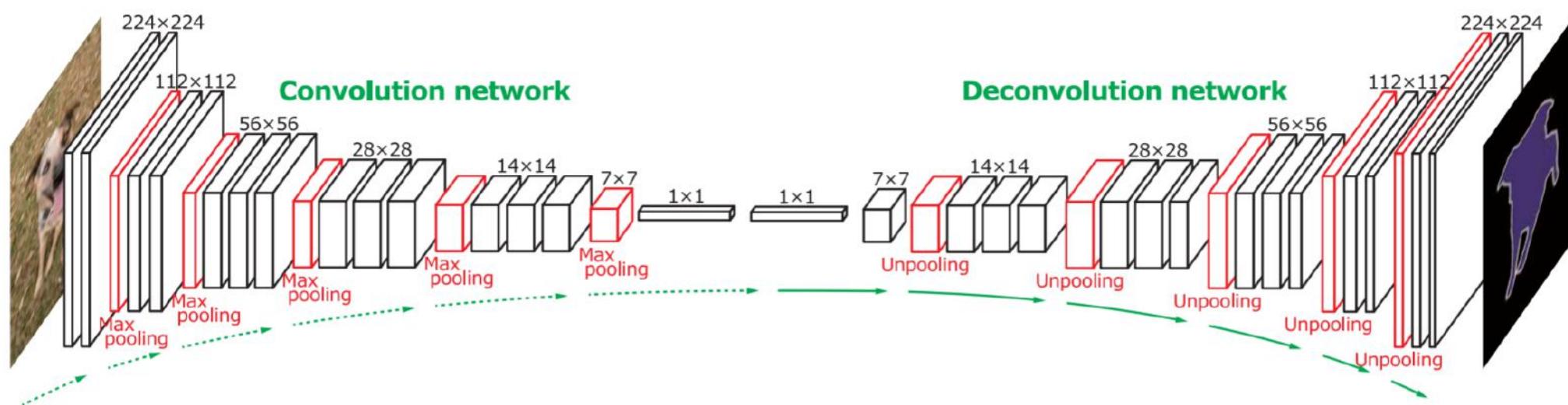
Fully Convolutional Neural Network (FCN)

- End-to-end CNN for Semantic Segmentation



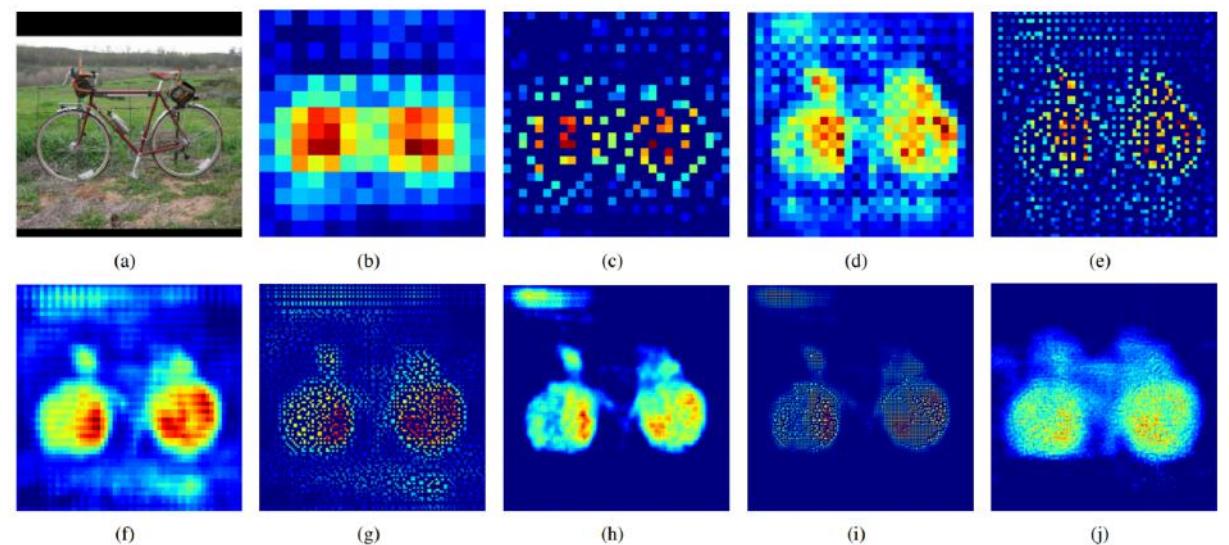
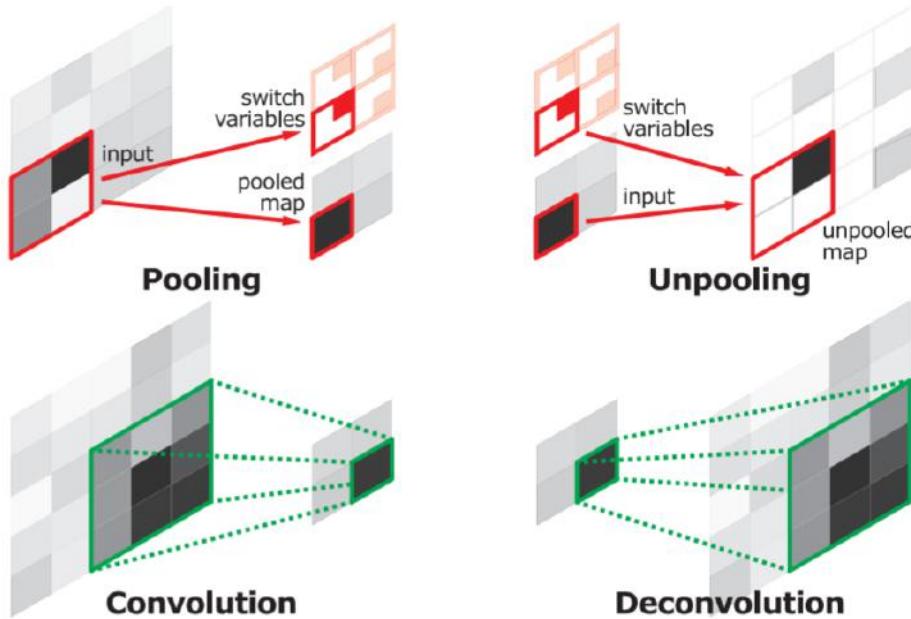
DeconvNet

- Deconvolutional Neural Network with UnPooling Operation



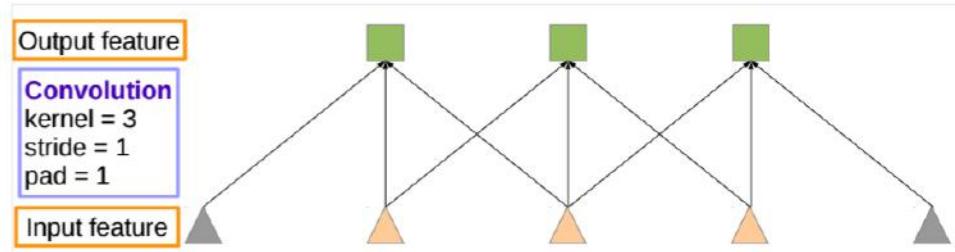
DeconvNet

- Deconvolutional Neural Network with UnPooling Operation

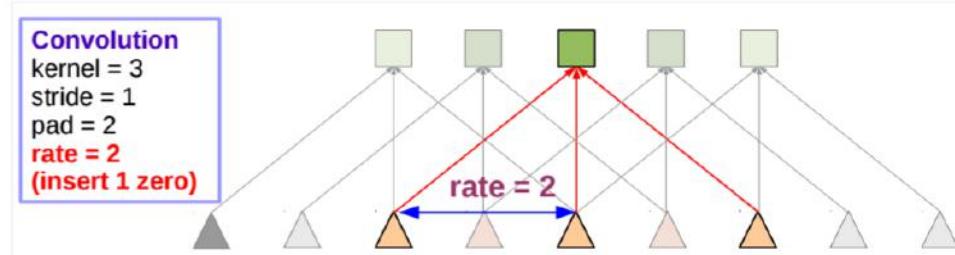


DeepLab

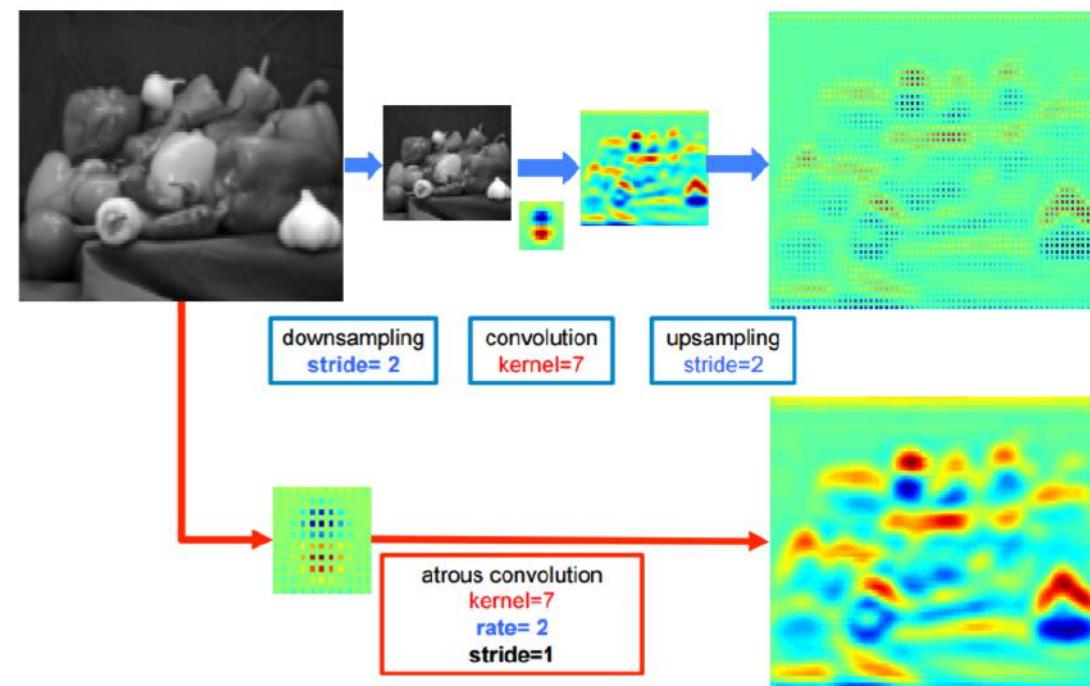
- Enlarging Field of View using Atrous Convolution



(a) Sparse feature extraction

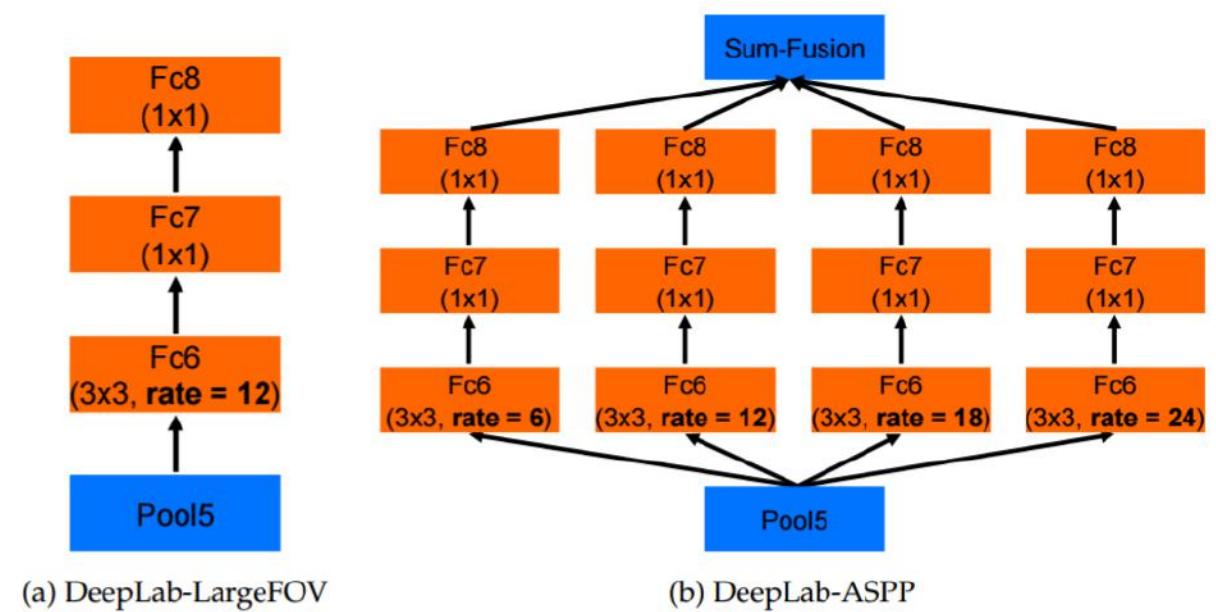
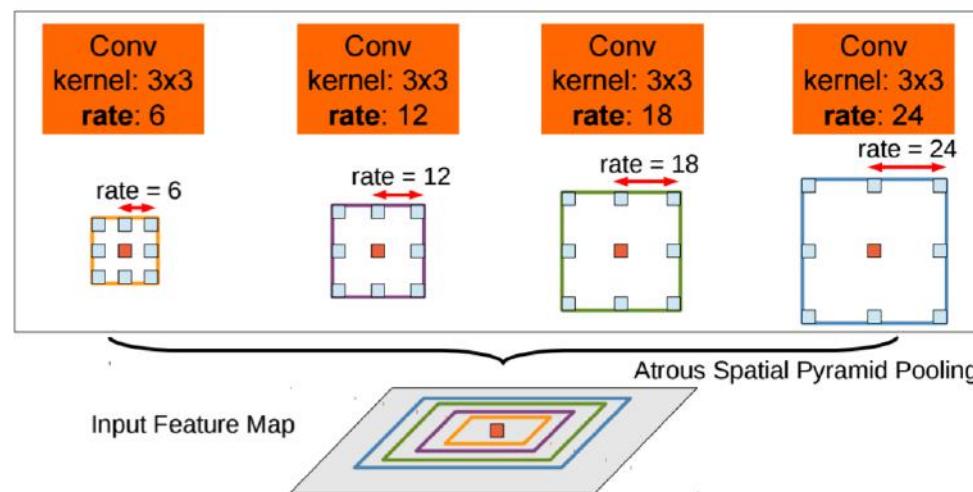


(b) Dense feature extraction



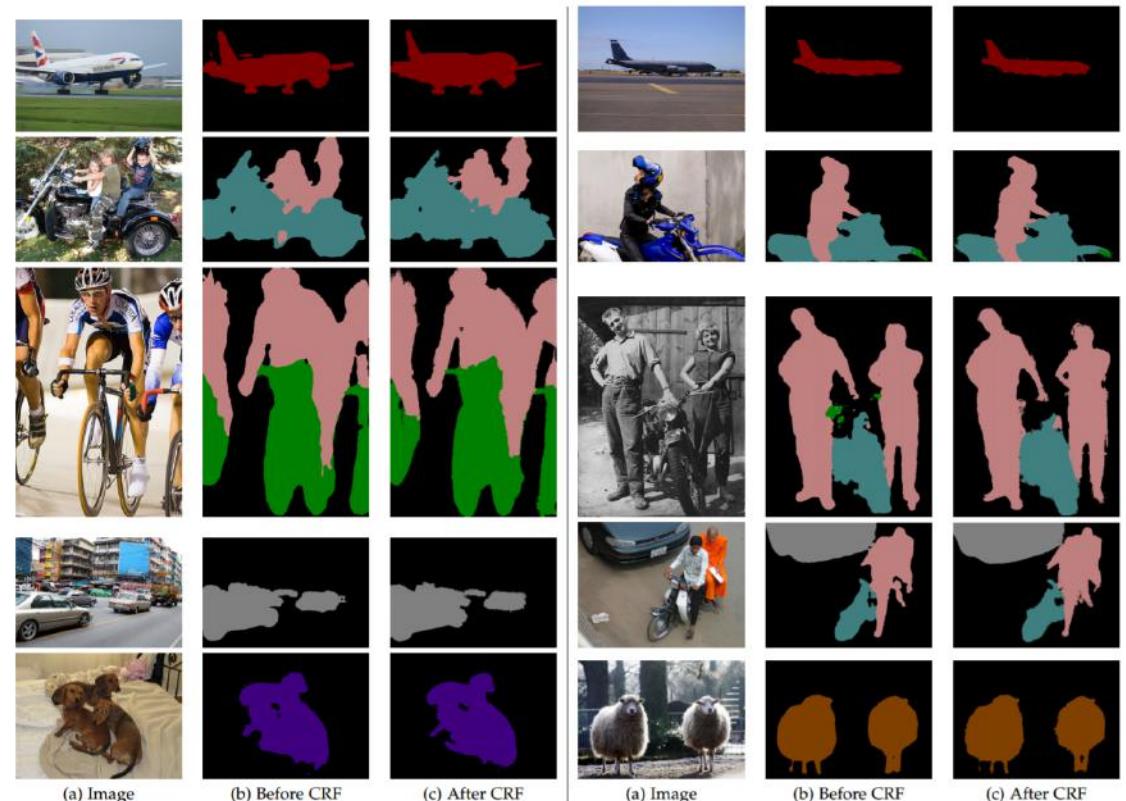
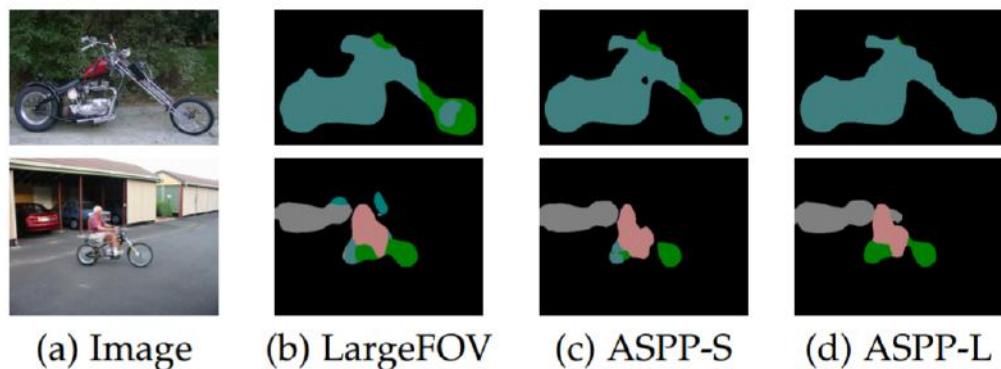
DeepLab

- Using Multiple 'Rate' for Multi-scale Object Segmentation



DeepLab

- Shows State-of-the-art Performance on VOC 2012 after CRF



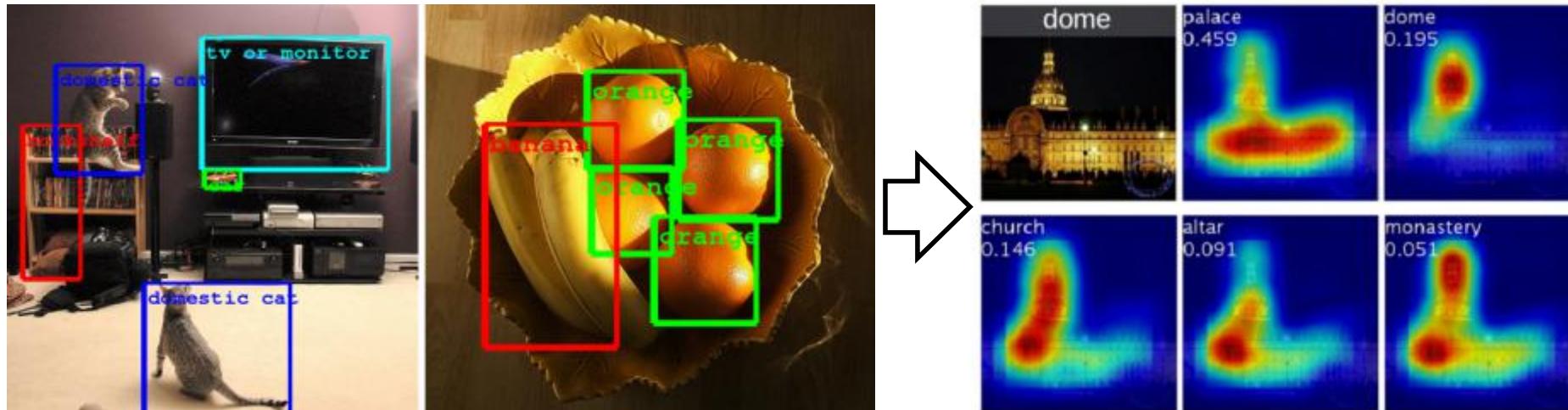
Class 14

CNNs for Weakly Supervised Learning

Weakly Supervised Learning

- Problem Definition

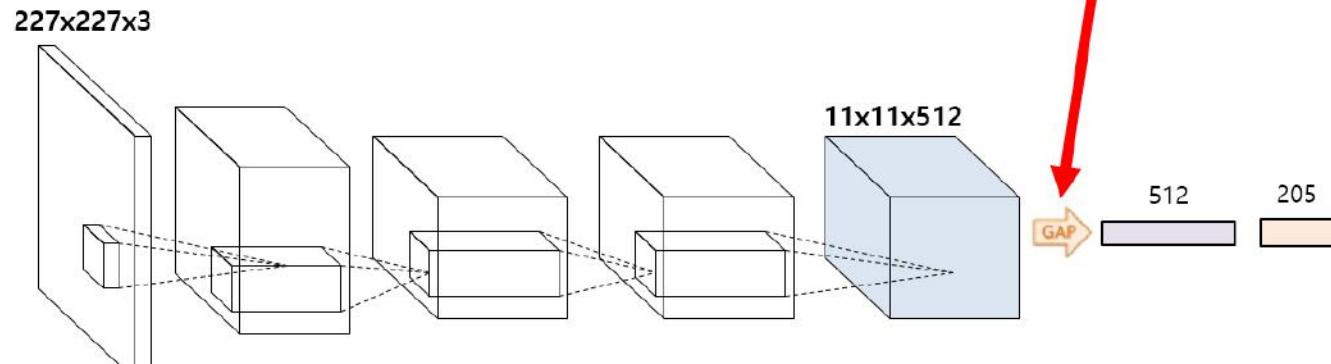
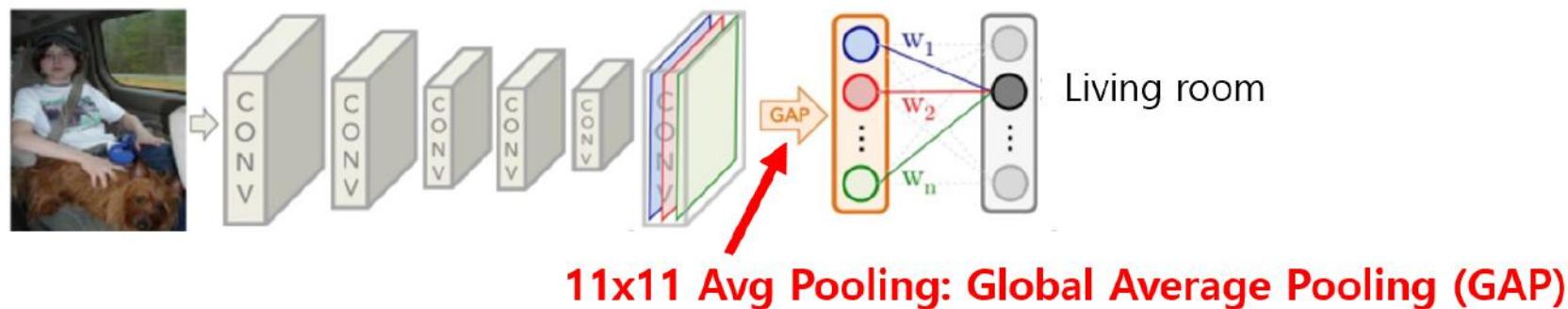
- What if we have only image-level annotations while we need to precisely locate the region of target object?



Weakly Supervised Learning

- Class Activation Map Approach

AlexNet+GAP+places205



```
('data', (10, 3, 227, 227))
('conv1', (10, 96, 55, 55))
('pool1', (10, 96, 27, 27))
('norm1', (10, 96, 27, 27))
('conv2', (10, 256, 27, 27))
('pool2', (10, 256, 13, 13))
('norm2', (10, 256, 13, 13))
('conv3', (10, 384, 13, 13))
('conv4', (10, 384, 13, 13))
('conv5', (10, 384, 13, 13))
('pool5', (10, 384, 11, 11))
('conv6', (10, 512, 11, 11))
('conv7', (10, 512, 11, 11))
('pool8_global', (10, 512, 1, 1))
('fc9', (10, 205))
('prob', (10, 205))
→ alexnetplusGAP_places205 []
```

Weakly Supervised Learning

- Class Activation Map Approach
 - Identify important image regions by projecting back the weights of output layer to convolutional feature maps.
 - CAMs can be generated for each class in single image.
 - Regions for each categories are different in given image.

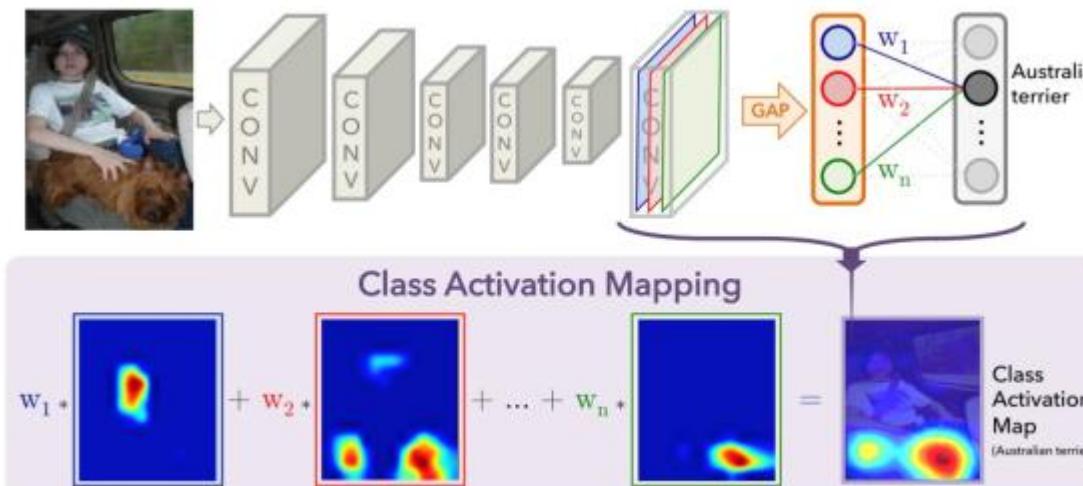


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

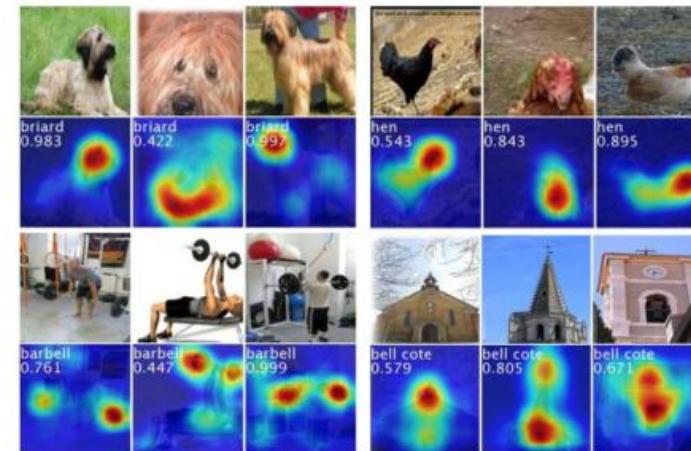


Figure 3. The CAMs of four classes from ILSVRC [20]. The maps highlight the discriminative image regions used for image classification e.g., the head of the animal for *briard* and *hen*, the plates in *barbell*, and the bell in *bell cote*.

Class 15

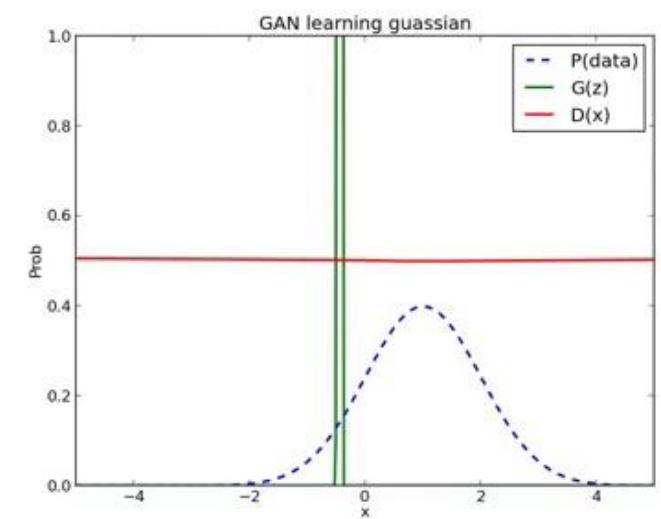
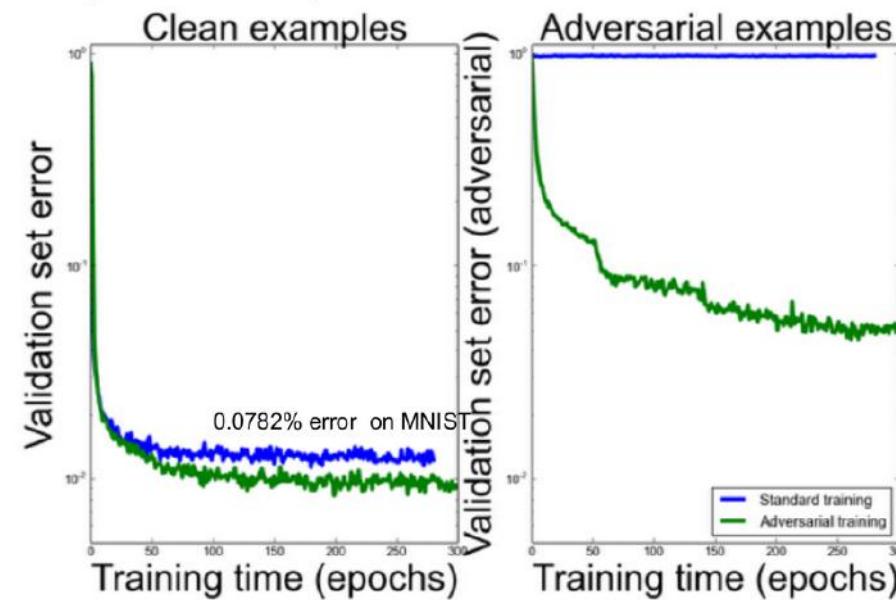
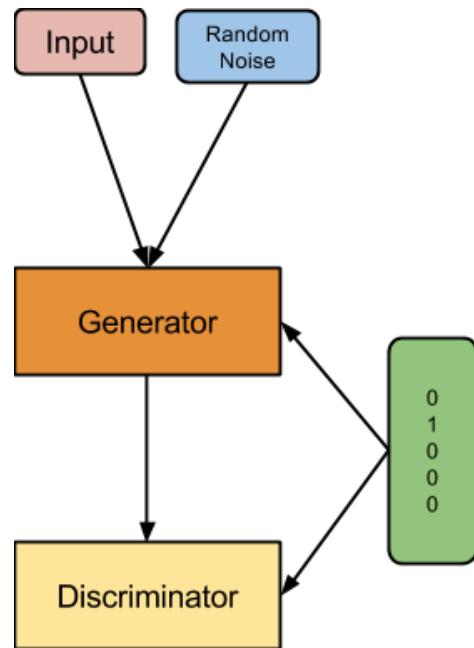
Generative Adversarial Networks

Generative Models

- Definition
 - A model for randomly generating observable data values, typically given some hidden parameters
 - “What I cannot created, I do not understand.”
- Examples
 - We can distinguish Chinese from Japanese but we cannot speak them.
 - Kids can read numbers and characters but cannot write down them.
- Deep Generative Models
 - Auto-Encoders : AE, Denoising AE, Sparse AE, Variational AE
 - Generative Adversarial Networks : GAN, DCGAN, InfoGAN,

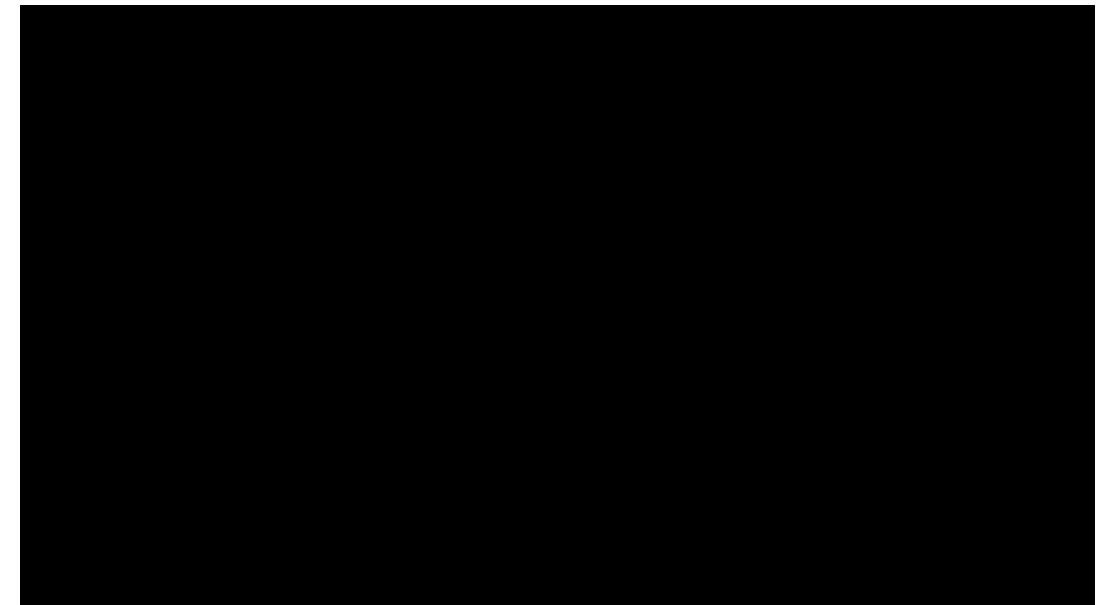
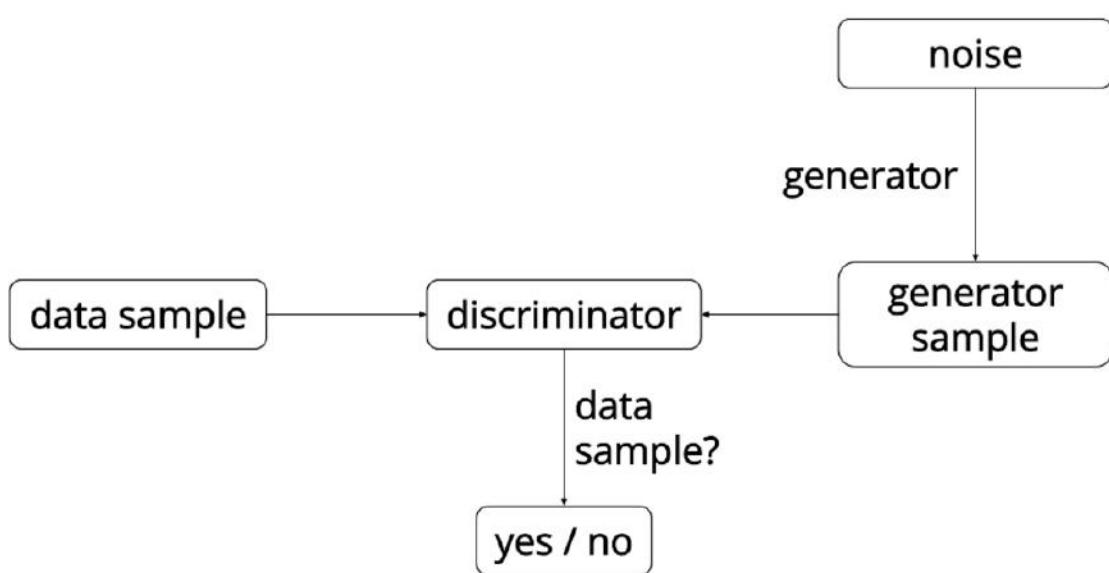
Generative Adversarial Networks (NIPS 2014)

- Generator vs Discriminator
 - Simpler generative model than MCMC-based or variational inference-based models.



Generative Adversarial Networks (NIPS 2014)

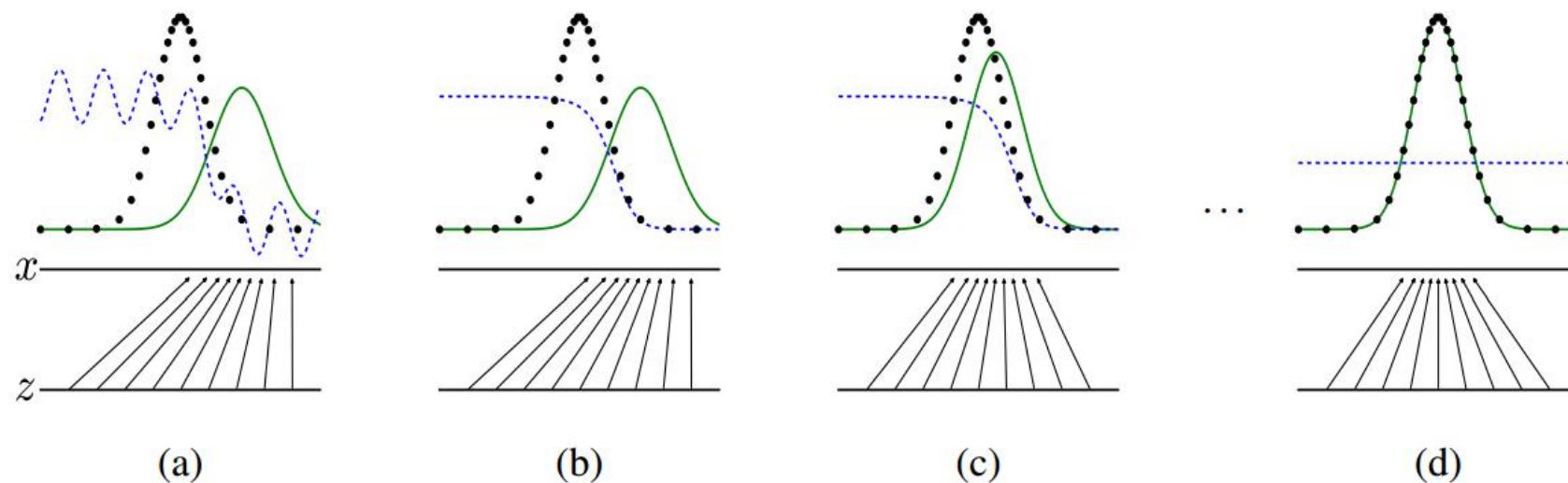
- 1D Example



Generative Adversarial Networks (NIPS 2014)

- MinMax Problem for Generator and Discriminator
 - Minimize for G, maxize for D.
 - D discriminate whether the input is from data or generative model.
 - G generate sample from

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$



Generative Adversarial Networks (NIPS 2014)

▪ The Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(x^{(i)}) + \log (1 - D(G(z^{(i)}))) \right].$$

end for

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^{(i)})))$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Training D to convergence
is expensive.

Therefore, train D k step
and train G slowly.

In the early phase D is
confident and gradient is small
Instead we can max $D(G(z))$

Generative Adversarial Networks (NIPS 2014)

- Theoretical Results
 - Optimality for D

Proposition 1. *For G fixed, the optimal discriminator D is*

$$D_G^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}$$

- Optimality for G

Theorem 1. *The global minimum of the virtual training criterion C(G) is achieved if and only if $p_g = p_{data}$. At that point, C(G) achieves the value $-\log 4$.*

- Convergence Analysis

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\mathbf{x} \sim p_{data}} [\log D_G^*(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g} [\log(1 - D_G^*(\mathbf{x}))]$$

then p_g converges to p_{data}

Generative Adversarial Networks (NIPS 2014)

- Experimental Results
 - D : MLP with maxout activation and dropout
 - G : MLP with ReLU and sigmoid activation



a)



b)



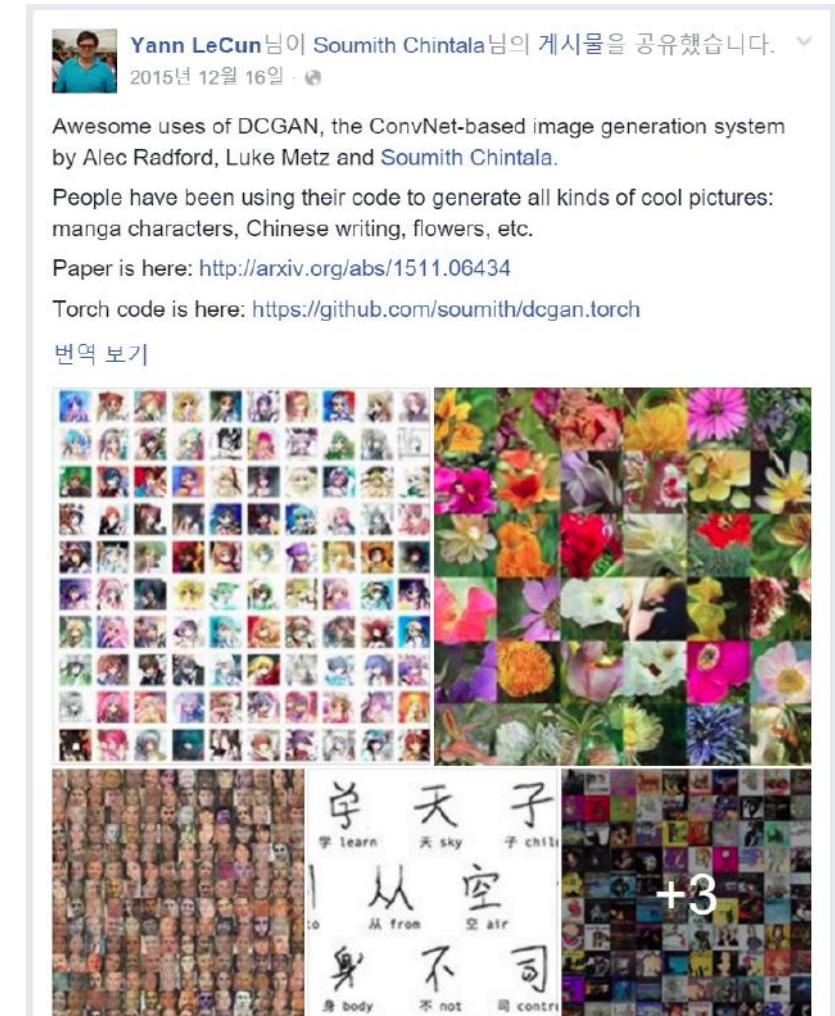
c)



d)

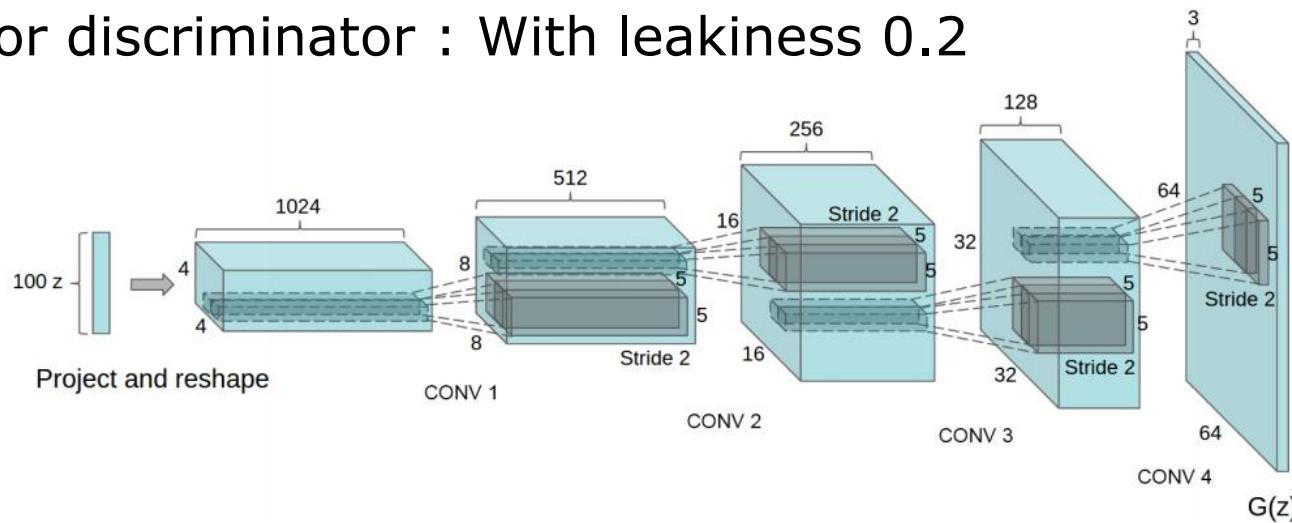
Deep Convolutional GAN (ICLR 2016)

- Bridging Gap between Supervised and Unsupervised Feature Learning
 - CNNs have mainly used for supervised tasks.
 - There are unlimited images without labels.
 - CNNs are good at dealing with Images.
 - How can we used CNNs to extract good representation of images without supervision?
- There have been some image generators
 - Yes. For example GAN and its laplacian pyramid extension, VAE, RNN, DeConvNet, ...
 - However, training network was unstable and the generated images were blurry or noisy.



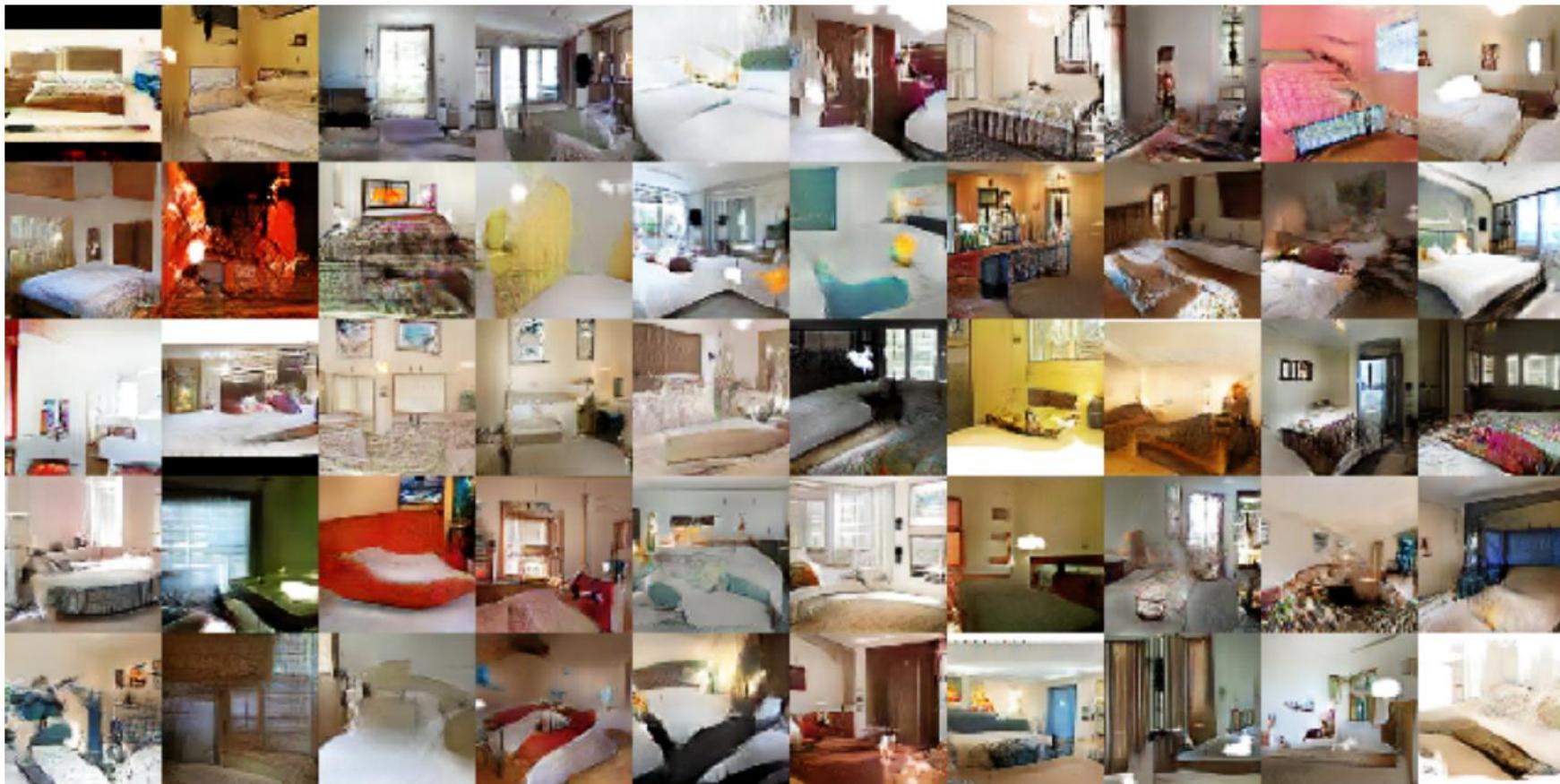
Deep Convolutional GAN (ICLR 2016)

- Recipes Making CNNs Work for GAN
 - All ConvNet : Replace pooling layer with strided convolutions.
 - No FC layers : Directly connecting last conv features to the input of generator and output of discriminator.
 - Batch normalization : Stabilizing learning process and help training deeper generator.
 - ReLU activation for generator : Except the output layer which uses Tanh
 - LeakyReLU for discriminator : With leakiness 0.2



Deep Convolutional GAN (ICLR 2016)

- Generated Samples – Large-scale Scene Understanding



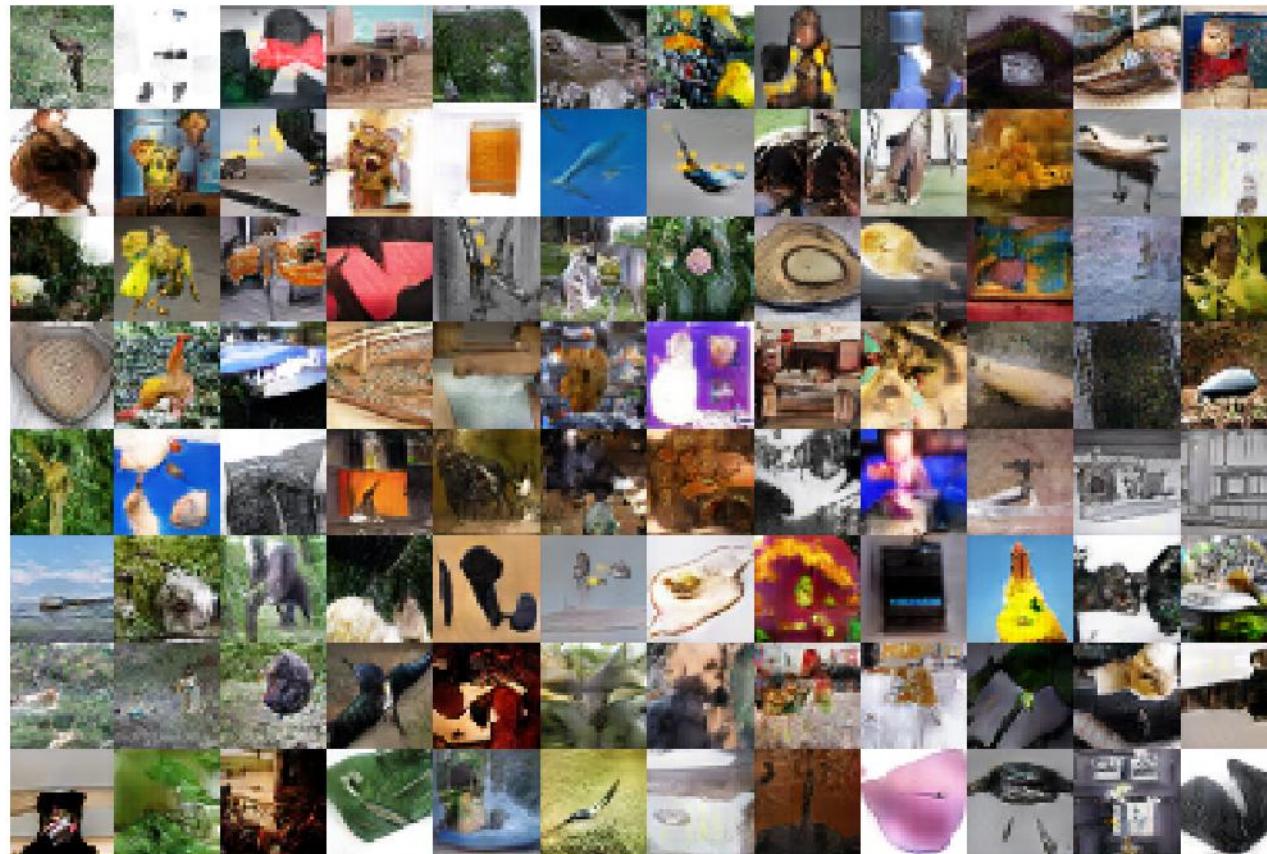
Deep Convolutional GAN (ICLR 2016)

- Generated Samples – Faces



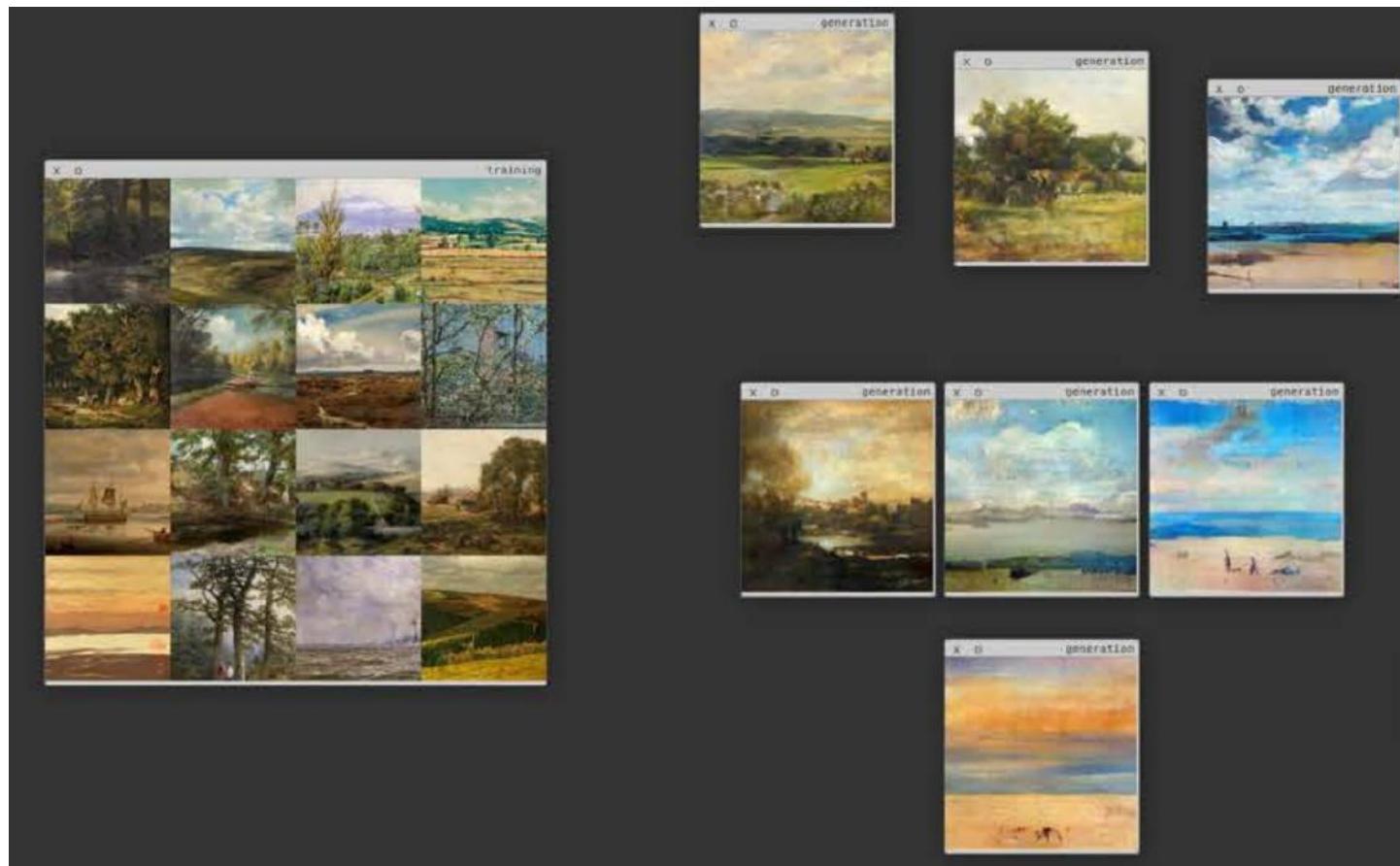
Deep Convolutional GAN (ICLR 2016)

- Generated Samples – ImageNet 1k(Trained with 32x32 center crop)



Deep Convolutional GAN (GTC 2016)

- Generated Samples – Natural Scene



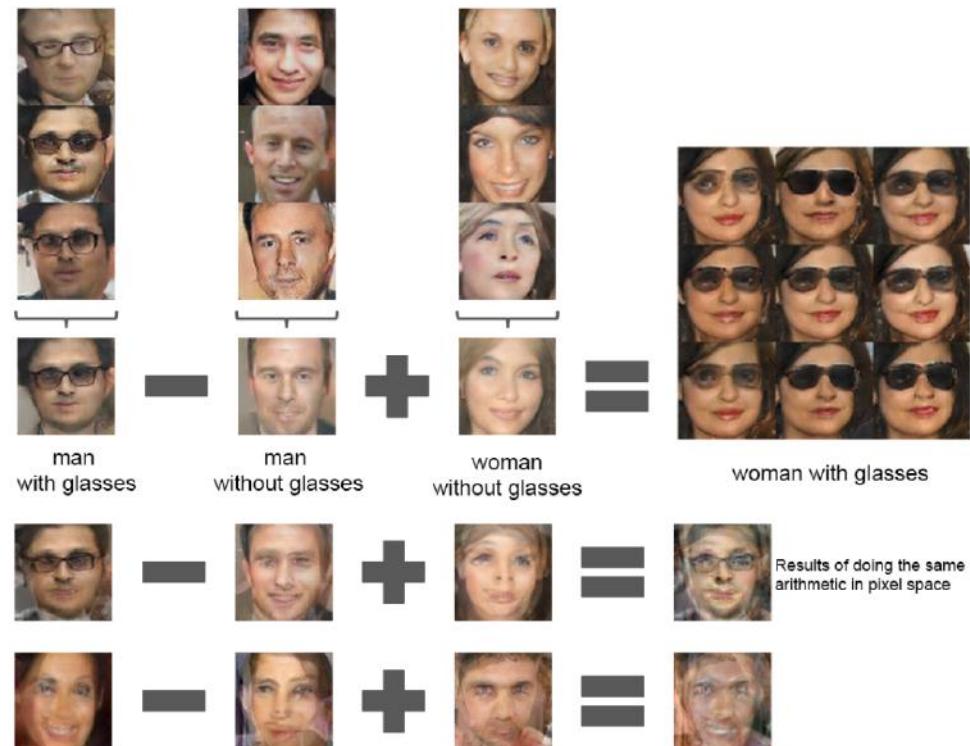
Deep Convolutional GAN (ICLR 2016)

- Walking in the Latent Space
 - If sharp transition is not observed and moving in the latent space result in semantic changes, it can be considered relevant representation has been learned.



Deep Convolutional GAN (ICLR 2016)

- Vector Arithmetic for Visual Concepts
 - Arithmetic on averaged Z representation of generator shows rich linear structure in the representation space.



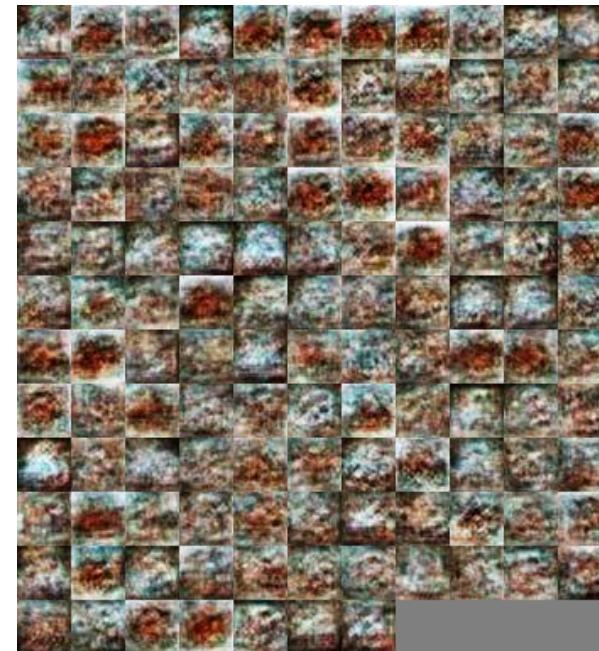
Lab

Lab

- GANs for MNIST and CIFAR-10

- https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part3/notebook

7 3 4 0 9 7 2 3 2 2
4 7 5 7 9 3 6 2 2 9
7 7 8 2 5 9 7 9 7 1
8 3 5 9 7 1 5 2 1 7
3 7 6 9 0 4 9 5 1 3
9 3 5 8 7 0 1 3 7 5
6 9 8 8 7 5 1 1 3 7
7 9 7 9 1 1 1 1 9 1
1 9 9 7 1 1 0 9 5 1
3 8 1 1 5 8 1 9 1 1



Q&A Session

Thank You!

kyuhwanjung@gmail.com