

Introduction to

 **TensorFlow™**

...

Aurélien Géron

TECHNOLOGIES ET ARCHITECTURES INTERNET

Corba, COM, XML, J2EE,
.NET, Web Services



01

Picardie
Réseaux & Télécom
David Doussot • A. Prost

DUNOD

WIFI PROFESSIONNEL

La norme 802.11,
le déploiement, la sécurité



Aurélien Géron
Préface de Marc Taïeb

3^e édition

DUNOD

POUR MIEUX DÉVELOPPER AVEC C++

Design patterns, STL, RTTI
et smart pointers



Aurélien Géron
Fatmé Tawbi
Préface de Gilles Clavell

DUNOD

O'REILLY®

Hands-on Machine Learning with Scikit-Learn & TensorFlow

CONCEPTS, TOOLS, AND TECHNIQUES
TO BUILD INTELLIGENT SYSTEMS



Aurélien Géron

wheelchair basketball: 0.829
basketball: 0.114
streetball: 0.020



Large-scale Video Classification with Convolutional Neural Networks, CVPR 2014
https://youtu.be/qrzQ_AB1DZk

Agenda – Today

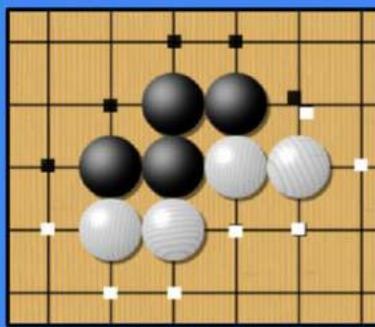
- A Quick Look at the Landscape
- TensorFlow Basics
- Linear Regression with TensorFlow
- Using Autodiff and Optimizers
- Feeding Data to a Model
- Saving and Restoring a Model
- TensorBoard (optional homework)

Agenda – Tomorrow

- Organizing Your Code
- Artificial Neural Networks
- Techniques For Training Deep Nets
- If time allows:
 - Convolutional Neural Nets for Image Classification

A Quick Look at the Landscape

The Machine Learning Revolution



Growing Use of Deep Learning at Google

Number of directories containing model description files

1500

1000

500

2012

2013

2014

2015

Across many areas

- AlphaGo
- Apps
- Maps
- Photos
- Gmail
- Speech
- Android
- YouTube
- Translation
- Robotics Research
- Image Understanding
- Natural Language Understanding
- Drug Discovery

Source: Hadoop Summit presentation by Kaz Sato, Google (goo.gl/95yfW4)

Deep Learning Frameworks

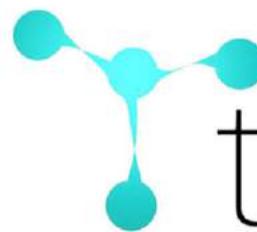
Caffe

dmlc

mxnet

 TensorFlow™

DEEPMLEARNING4J

 torch

theano

amazon
DSSTNE

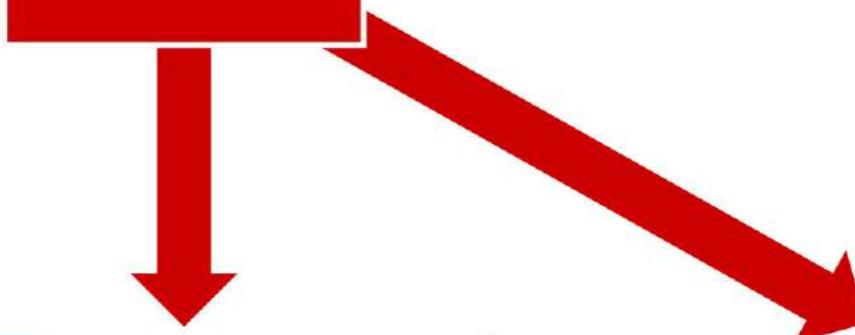
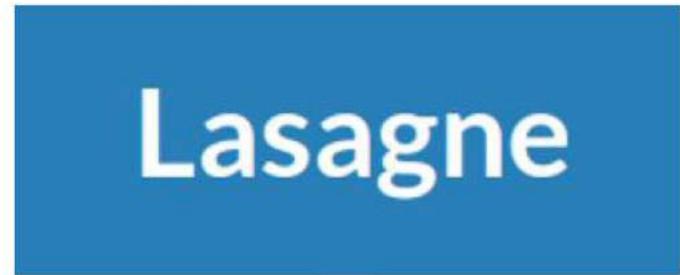
 Microsoft
CNTK

Frameworks on Top of Frameworks

Keras



Lasagne

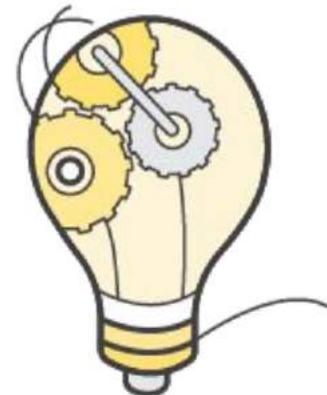


Cloud Platforms for Machine Learning

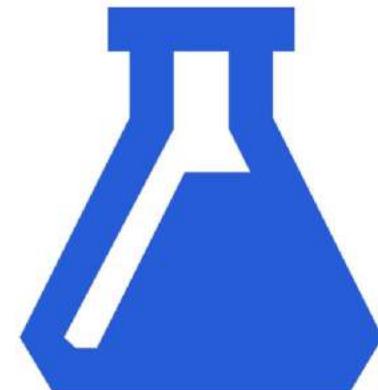
Google



amazon



Microsoft



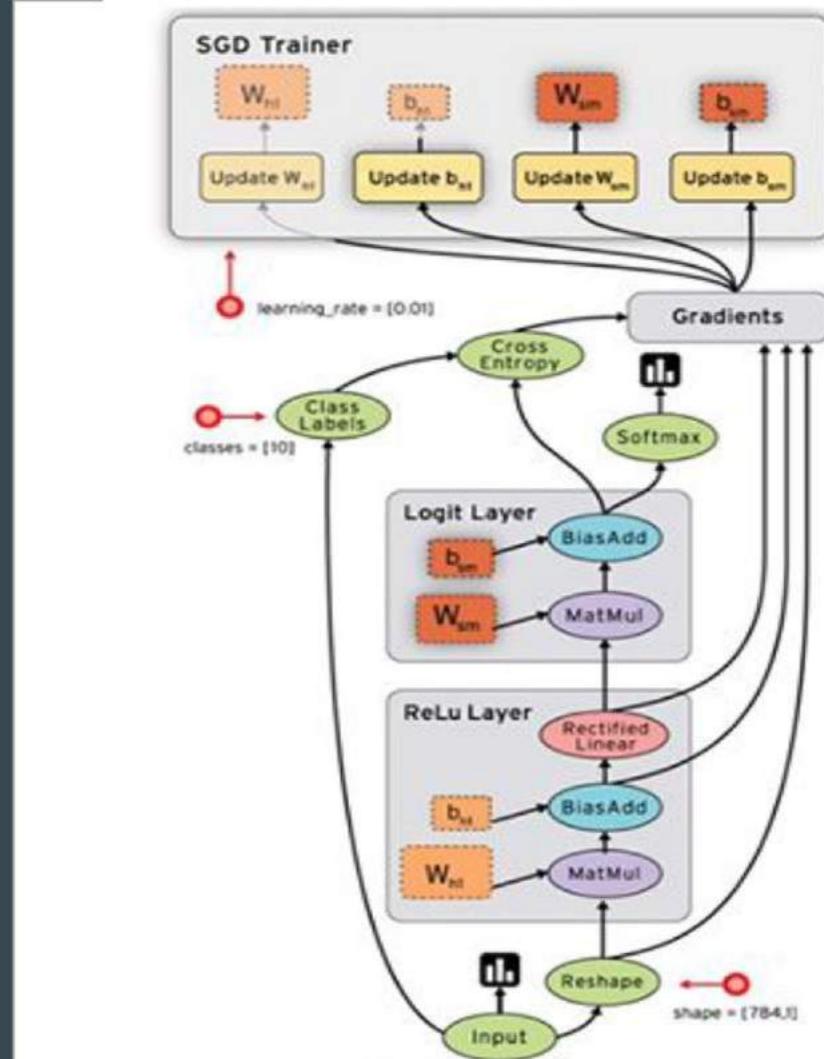
Possibilities



Why TensorFlow?

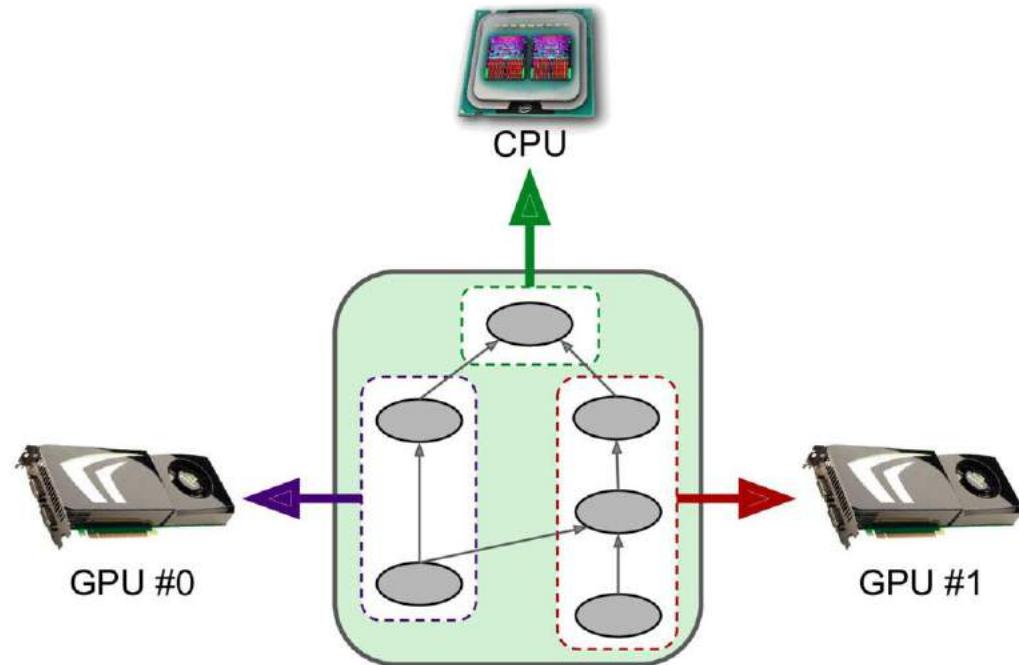
Why Choose TensorFlow?

Flexibility



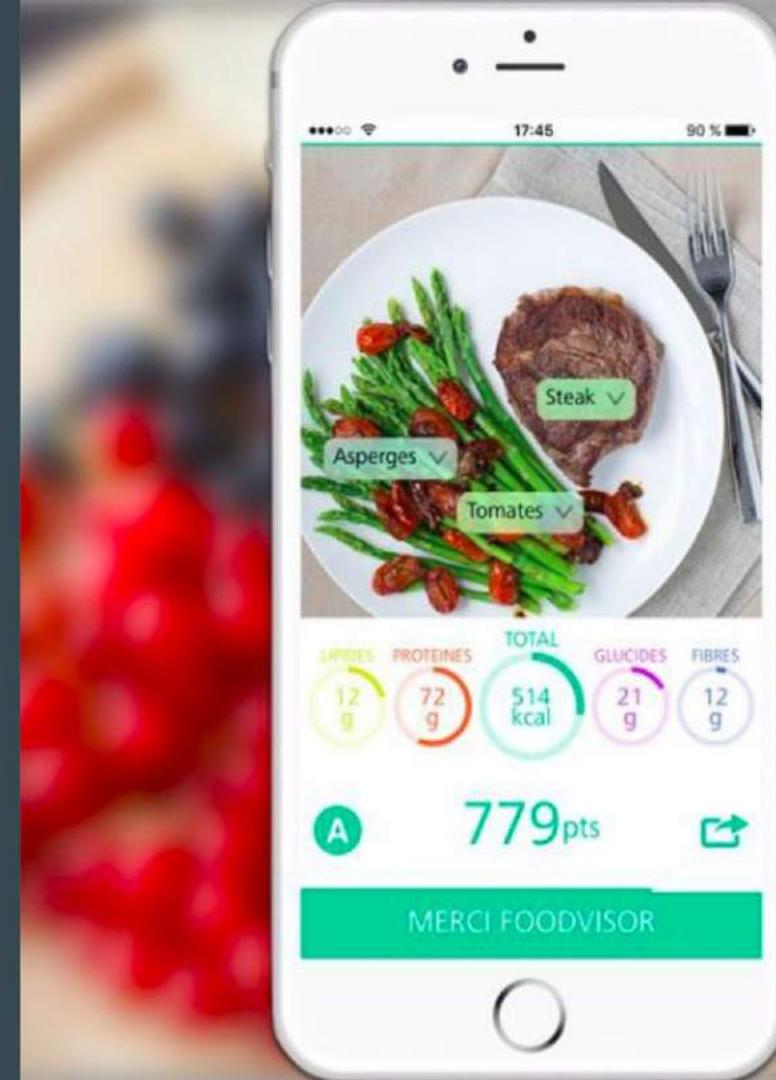
Why Choose TensorFlow?

Scalability



Why Choose TensorFlow?

Portability



Why Choose TensorFlow?

Popularity



Star

51,800

Fork

24,320

TensorFlow Basics

TensorFlow Basics > Import TensorFlow

```
>>> import tensorflow as tf  
>>> tf.__version__  
'1.7.0'
```

Construction phase

```
>>> a = tf.constant(3)
>>> b = tf.constant(5)
>>> s = a + b
```

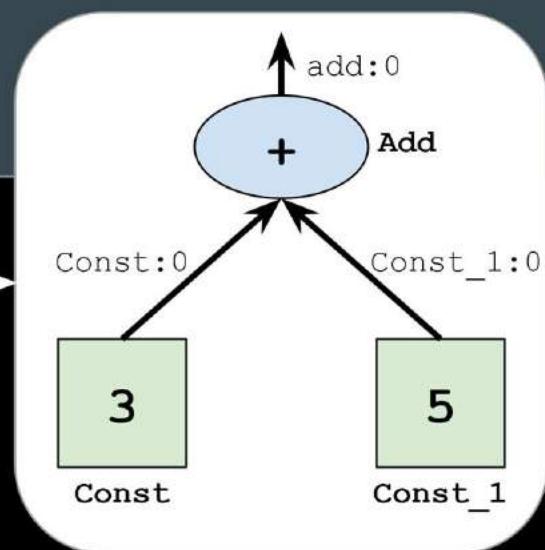
Construction phase

```
>>> a = tf.constant(3)
>>> b = tf.constant(5)
>>> s = a + b
>>> a
<tf.Tensor 'Const:0' shape=() dtype=int32>
>>> b
<tf.Tensor 'Const_1:0' shape=() dtype=int32>
>>> s
<tf.Tensor 'add:0' shape=() dtype=int32>
```

TensorFlow Basics > **Construction phase**

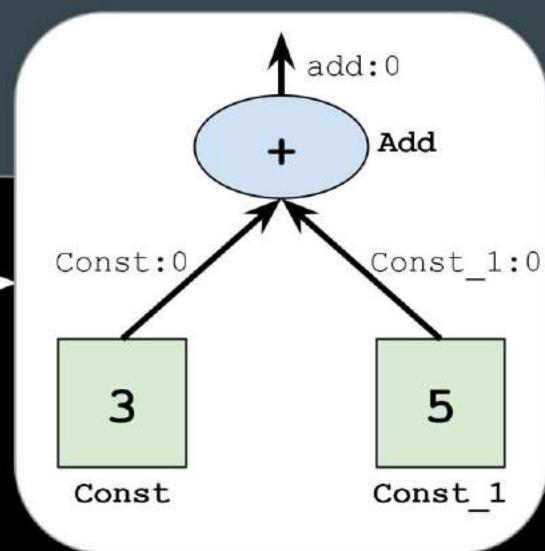
```
>>> a = tf.constant(3)
>>> b = tf.constant(5)
>>> s = a + b
```

```
>>> a
<tf.Tensor 'Const:0' shape=() dtype=int32>
>>> b
<tf.Tensor 'Const_1:0' shape=() dtype=int32>
>>> s
<tf.Tensor 'add:0' shape=() dtype=int32>
```



TensorFlow Basics > **Construction phase**

```
>>> a = tf.constant(3)
>>> b = tf.constant(5)
>>> s = a + b
>>> a
<tf.Tensor 'Const:0' shape=() dtype=int32>
>>> b
<tf.Tensor 'Const_1:0' shape=() dtype=int32>
>>> s
<tf.Tensor 'add:0' shape=() dtype=int32>
>>> tf.get_default_graph()
<tensorflow.python.framework.ops.Graph object at 0x7f7c251a03c8>
```



Construction phase

```
>>> graph = tf.Graph()  
>>> with graph.as_default():  
...     a = tf.constant(3)  
...     b = tf.constant(5)  
...     s = a + b  
...
```

Execution phase

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     a = tf.constant(3)
...     b = tf.constant(5)
...     s = a + b
...
>>> with tf.Session(graph=graph) as sess:
...     result = s.eval()
...
>>> result
```

Execution phase

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     a = tf.constant(3)
...     b = tf.constant(5)
...     s = a + b
...
>>> with tf.Session(graph=graph) as sess:
...     result = s.eval()
...
>>> result
```

Execution phase

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     a = tf.constant(3)
...     b = tf.constant(5)
...     s = a + b
...
>>> with tf.Session(graph=graph) as sess:
...     result = sess.run(s)
...
>>> result
```

Execution phase

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     a = tf.constant(3)
...     b = tf.constant(5)
...     s = a + b
...
>>> with tf.Session(graph=graph) as sess:
...     result = sess.run([a,b,s])
...
>>> result
[3, 5, 8]
```

10 minutes

Exercise 1



TensorFlow Basics >

Variables

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.Variable(100)
...     c = tf.constant(5)
...     increment_op = tf.assign(x, x + c)
...
...
```

TensorFlow Basics >

Variables

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.Variable(100)
...     c = tf.constant(5)
...     increment_op = tf.assign(x, x + c)
...
...
>>> with tf.Session(graph=graph) as sess:
...     x.initializer.run()
```

TensorFlow Basics >

Variables

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.Variable(100)
...     c = tf.constant(5)
...     increment_op = tf.assign(x, x + c)
...
...
>>> with tf.Session(graph=graph) as sess:
...     x.initializer.run()
...     print(x.eval())      # 100
```

Variables

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.Variable(100)
...     c = tf.constant(5)
...     increment_op = tf.assign(x, x + c)
...
...
>>> with tf.Session(graph=graph) as sess:
...     x.initializer.run()
...     print(x.eval())      # 100
...     for iteration in range(10):
...         increment_op.eval()
...     print(x.eval())      # 150
```

Variables

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.Variable(100)
...     c = tf.constant(5)
...     increment_op = tf.assign(x, x + c)
...
...
>>> with tf.Session(graph=graph) as sess:
...     x.initializer.run()
...     print(x.eval())      # 100
...     for iteration in range(10):
...         increment_op.eval()
...     print(x.eval())      # 150
```

Variable initializer

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.Variable(100)
...     c = tf.constant(5)
...     increment_op = tf.assign(x, x + c)
...     init = tf.global_variables_initializer()

>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     print(x.eval())      # 100
...     for iteration in range(10):
...         increment_op.eval()
...     print(x.eval())      # 150
```

Variable state

```
>>> session1 = tf.Session(graph=graph)
>>> session2 = tf.Session(graph=graph)
>>> x.initializer.run(session=session1)
>>> x.initializer.run(session=session2)
```

Variable state

```
>>> session1 = tf.Session(graph=graph)
>>> session2 = tf.Session(graph=graph)
>>> x.initializer.run(session=session1)
>>> x.initializer.run(session=session2)
>>> increment_op.eval(session=session1)
105
>>> x.eval(session=session1)
105
>>> x.eval(session=session2)
100
```

Variable state

```
>>> session1 = tf.Session(graph=graph)
>>> session2 = tf.Session(graph=graph)
>>> x.initializer.run(session=session1)
>>> x.initializer.run(session=session2)
>>> increment_op.eval(session=session1)
105
>>> x.eval(session=session1)
105
>>> x.eval(session=session2)
100
>>> session1.close()
>>> session2.close()
```

12 minutes

Exercise 2



TensorFlow Basics >

Collections

```
>>> graph.get_collection(tf.GraphKeys.GLOBAL_VARIABLES)
[<tensorflow.python.ops.variables.Variable at 0x7f6ba451a5c0>]
```

TensorFlow Basics >

Collections

```
>>> graph.get_collection(tf.GraphKeys.GLOBAL_VARIABLES)
[<tensorflow.python.ops.variables.Variable at 0x7f6ba451a5c0>]
>>> tf.GraphKeys.GLOBAL_VARIABLES
'variables'
```

TensorFlow Basics >

Collections

```
>>> graph.get_collection(tf.GraphKeys.GLOBAL_VARIABLES)
[<tensorflow.python.ops.variables.Variable at 0x7f6ba451a5c0>]
>>> tf.GraphKeys.GLOBAL_VARIABLES
'variables'
```

Other standard collections:
LOCAL_VARIABLES, MODEL_VARIABLES,
TRAINABLE_VARIABLES, SUMMARIES,
QUEUE_RUNNERS, MOVING_AVERAGE_VARIABLES,
REGULARIZATION_LOSSES, WEIGHTS,
BIASES, ACTIVATIONS

TensorFlow Basics >

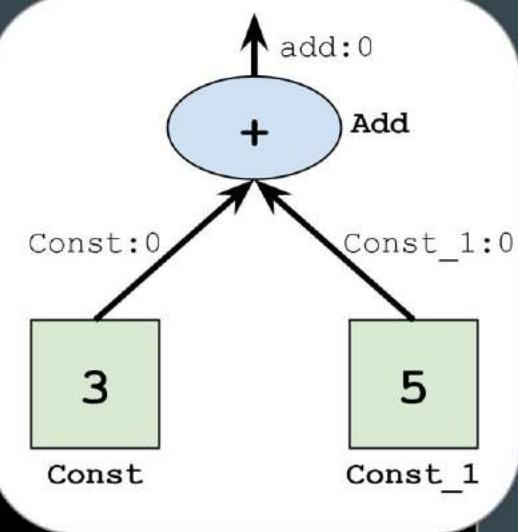
Collections

```
>>> graph.get_collection(tf.GraphKeys.GLOBAL_VARIABLES)
[<tensorflow.python.ops.variables.Variable at 0x7f6ba451a5c0>]
>>> tf.GraphKeys.GLOBAL_VARIABLES
'variables'

>>> graph.add_to_collection("my_collection", c)
>>> graph.get_collection("my_collection")
[<tf.Tensor 'Const:0' shape=() dtype=int32>]
```

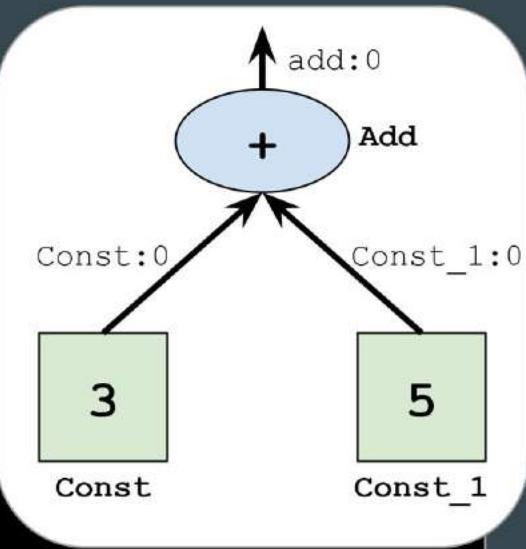
TensorFlow Basics > Navigating the Graph

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     a = tf.constant(3)
...     b = tf.constant(5)
...     s = a + b
...
>>> graph.get_operations()
[<tf.Operation 'Const' type=Const>,
 <tf.Operation 'Const_1' type=Const>,
 <tf.Operation 'add' type=Add>]
```



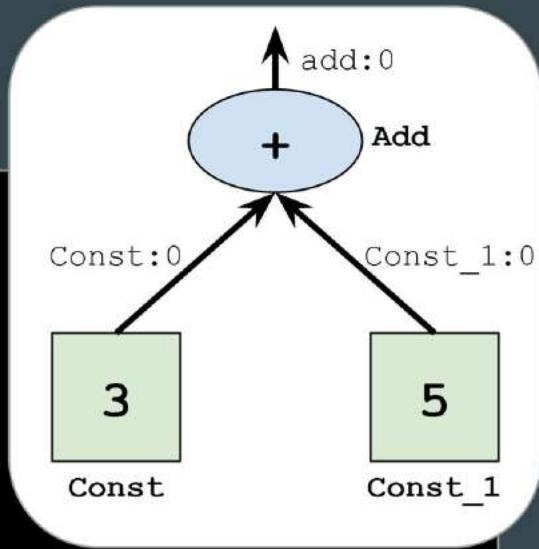
TensorFlow Basics > Navigating the Graph

```
>>> graph.get_operation_by_name("add") is s.op  
True  
>>> graph.get_tensor_by_name("add:0") is s  
True
```



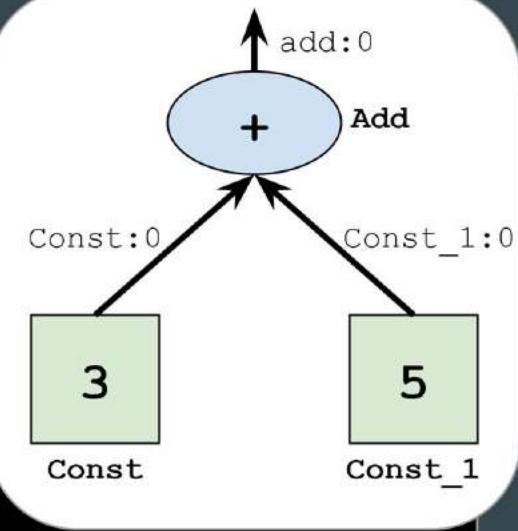
TensorFlow Basics > Navigating the Graph

```
>>> list(s.op.inputs)
[<tf.Tensor 'Const:0' shape=() dtype=int32>,
 <tf.Tensor 'Const_1:0' shape=() dtype=int32>]
>>> list(s.op.outputs)
[<tf.Tensor 'add:0' shape=() dtype=int32>]
```



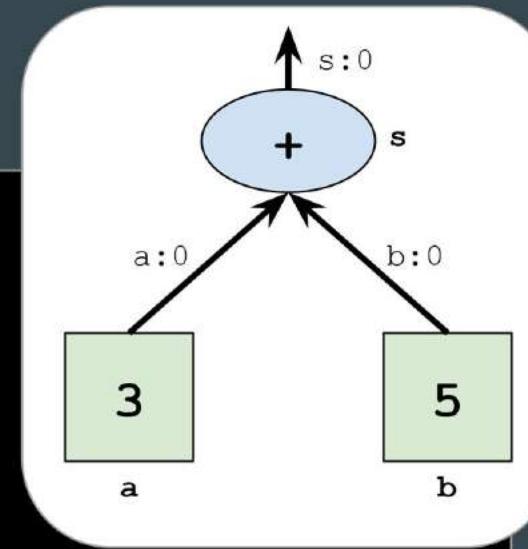
TensorFlow Basics > Naming operations

```
>>> graph = tf.Graph()  
>>> with graph.as_default():  
...     a = tf.constant(3)  
...     b = tf.constant(5)  
...     s = a + b  
...
```



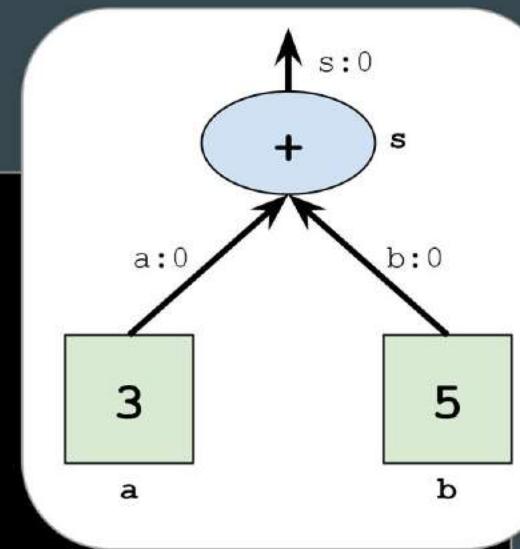
TensorFlow Basics > Naming operations

```
>>> graph = tf.Graph()  
>>> with graph.as_default():  
...     a = tf.constant(3, name='a')  
...     b = tf.constant(5, name='b')  
...     s = tf.add(a, b, name='s')  
...
```



TensorFlow Basics > Naming operations

```
>>> graph = tf.Graph()  
>>> with graph.as_default():  
...     a = tf.constant(3, name='a')  
...     b = tf.constant(5, name='b')  
...     s = tf.add(a, b, name='s')  
...  
>>> graph.get_operations()  
[<tf.Operation 'a' type=Const>,  
<tf.Operation 'b' type=Const>,  
<tf.Operation 's' type=Add>]
```



10 minutes
(optional)

Exercise 3



Cheat Sheet

- Quick installation: `pip install tensorflow`
- Construction phase, graphs: `graph = tf.Graph(); with graph.as_default(): ...`
- Execution phase, sessions: `with tf.Session(graph=graph) as sess: ...`
- Evaluating operations: `r = s.eval(); r1, r2 = sess.run([s1, s2])`
- Constants: `a = tf.constant([[1.0, 2.0], [3.0, 4.0]])`
- Variables:
 - Initializer: `init = tf.global_variables_initializer()`
 - Assignment: `increment_op = tf.assign(x, x + c)`
- Collections: `tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLE)`
- Navigating the graph: `ops = graph.get_operations()`
- Naming operations: `s = tf.add(a, b, name='s')`

Linear Regression with TensorFlow

Example: Does Money Make People Happier?

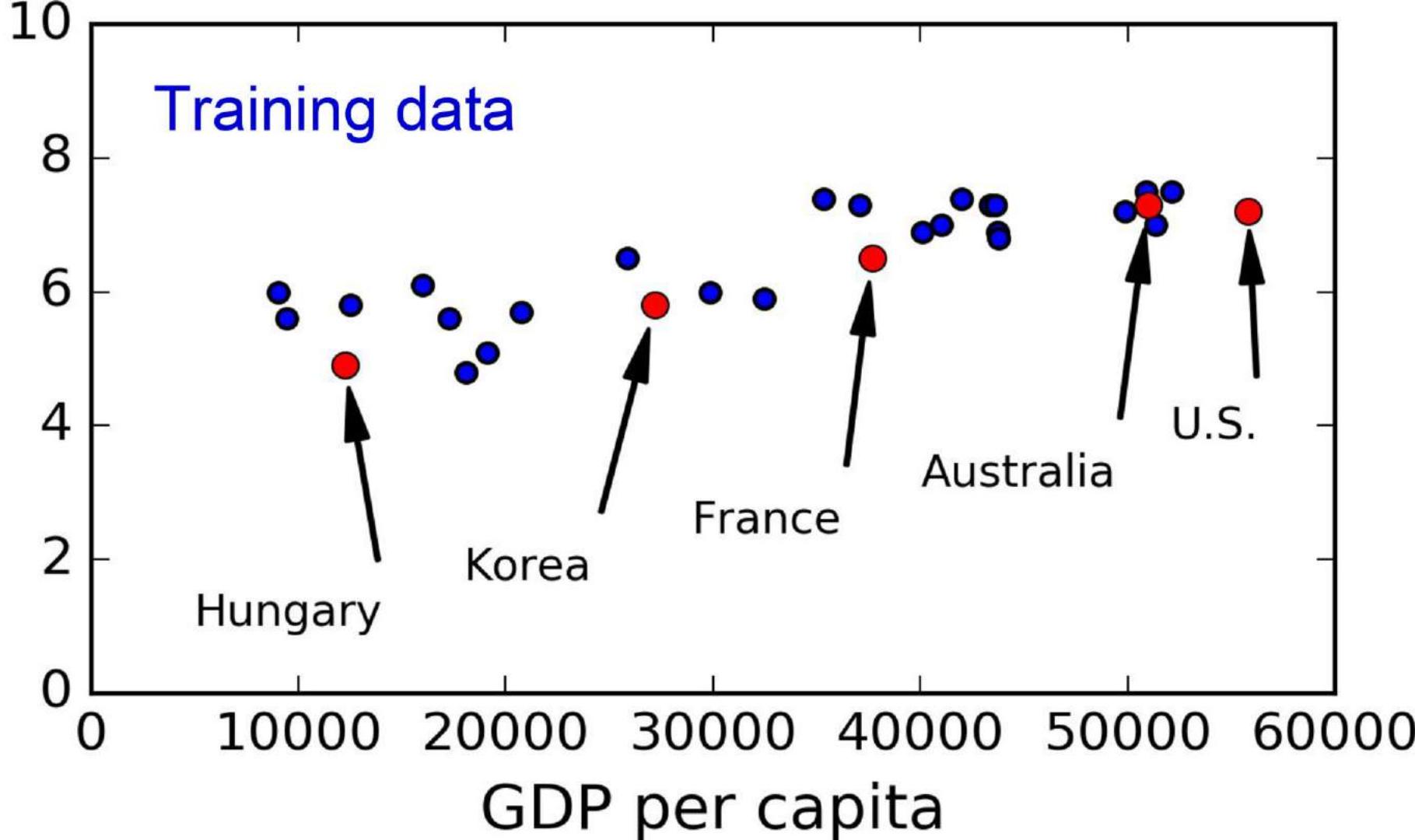
What is the mean life satisfaction in Cyprus? (from 0 to 10)

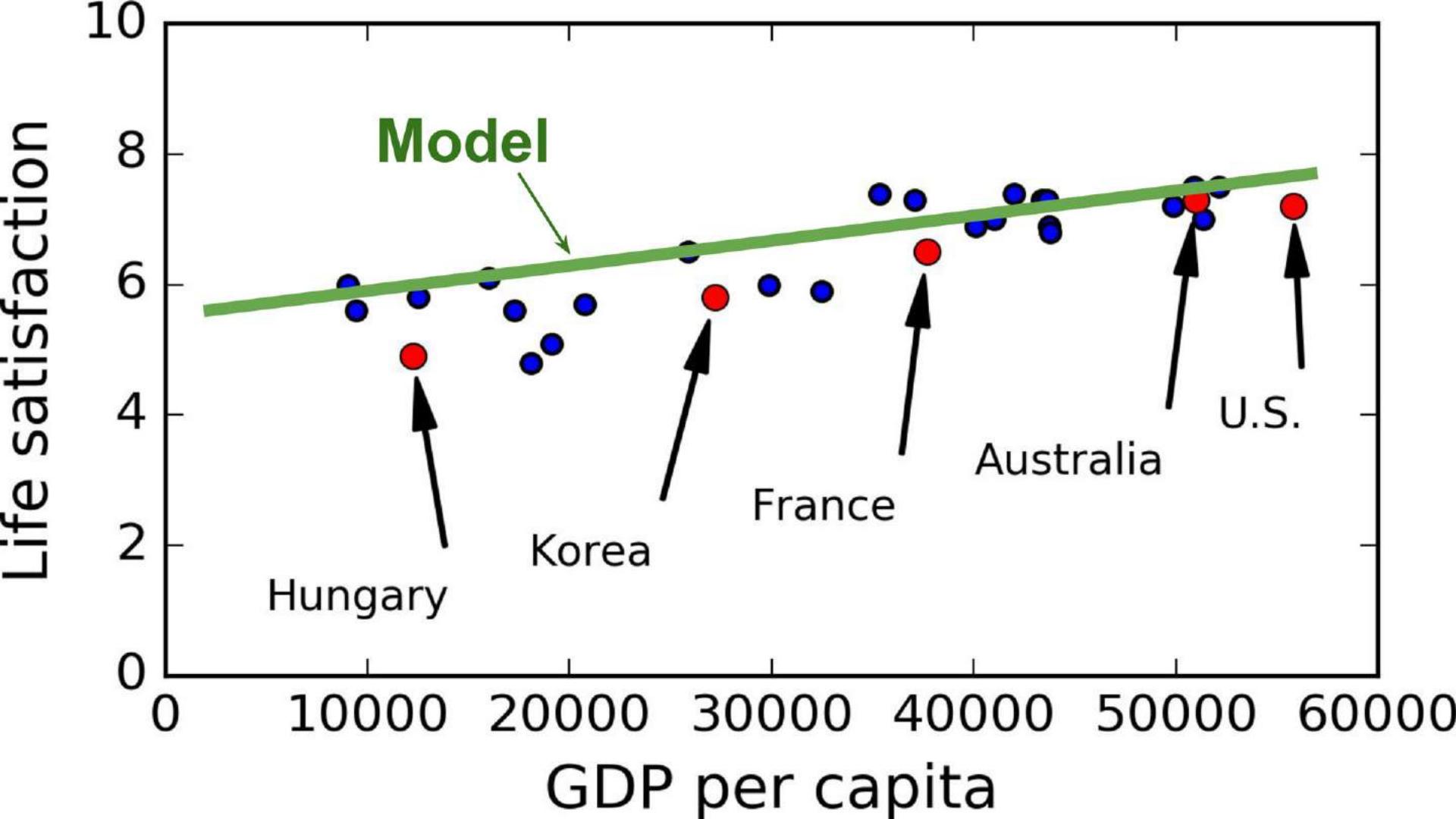
Example: Does Money Make People Happier?

What is the mean life satisfaction in Cyprus? (from 0 to 10)

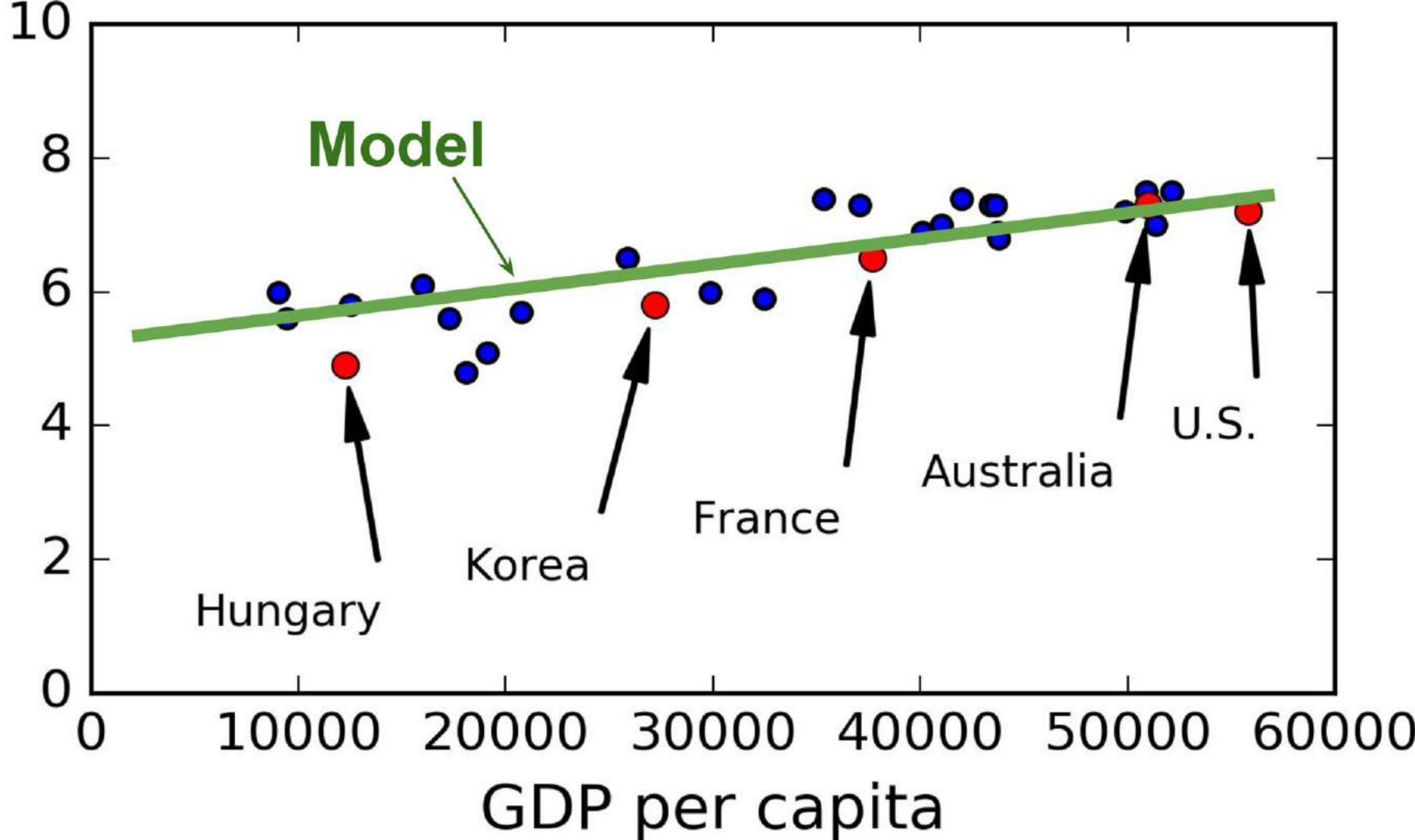
- GDP per capita is \$22,000

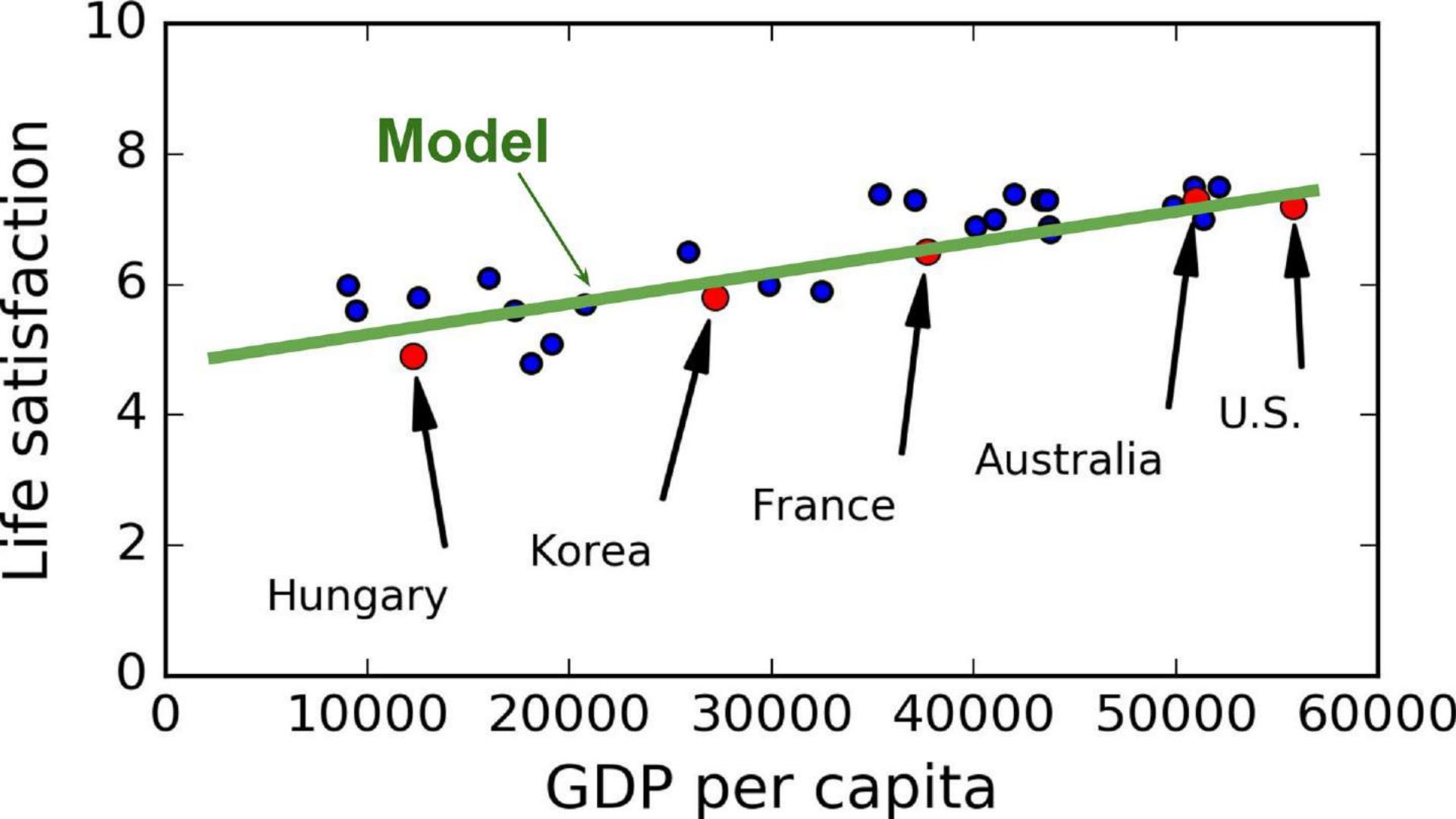
Life satisfaction

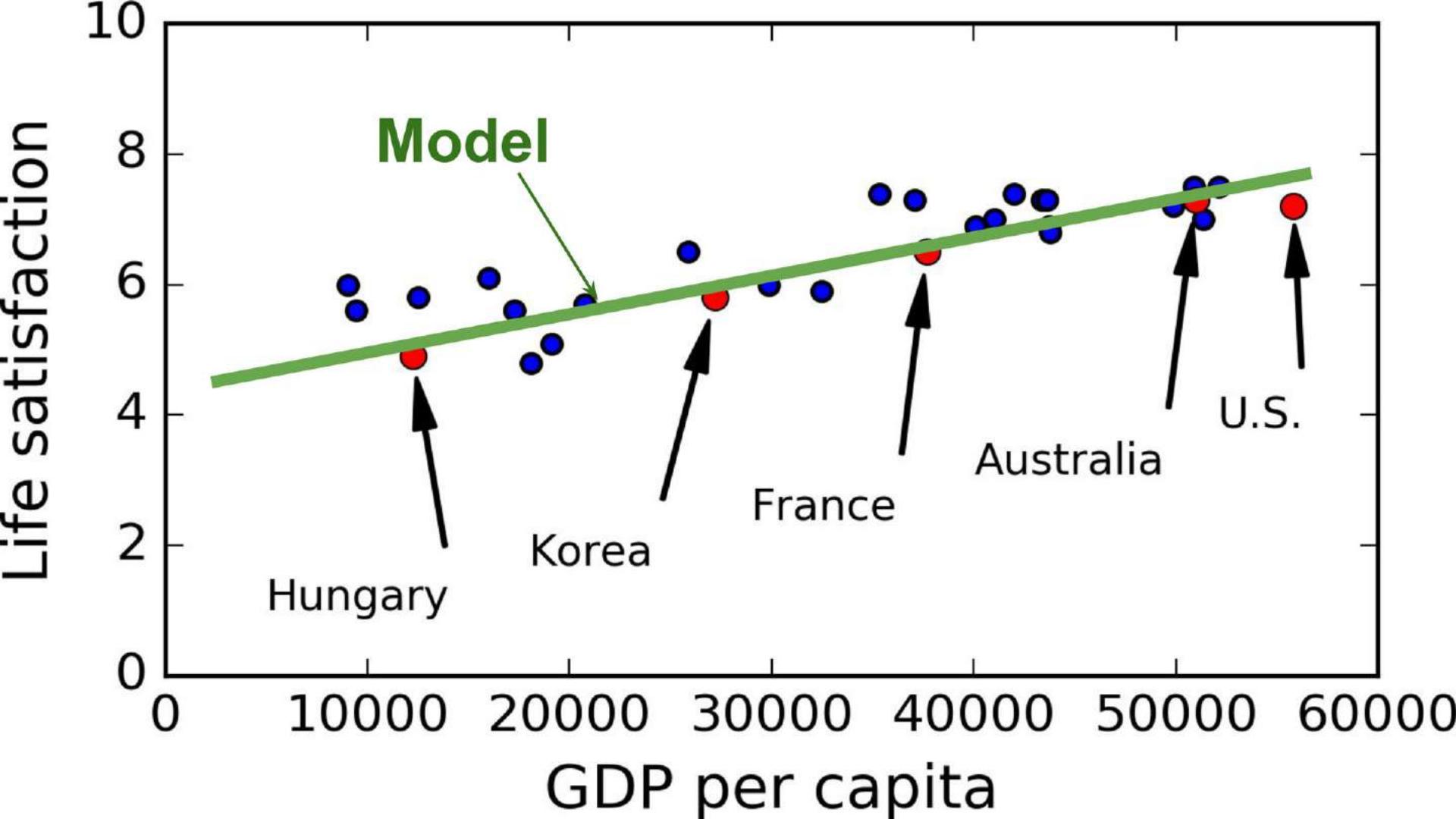


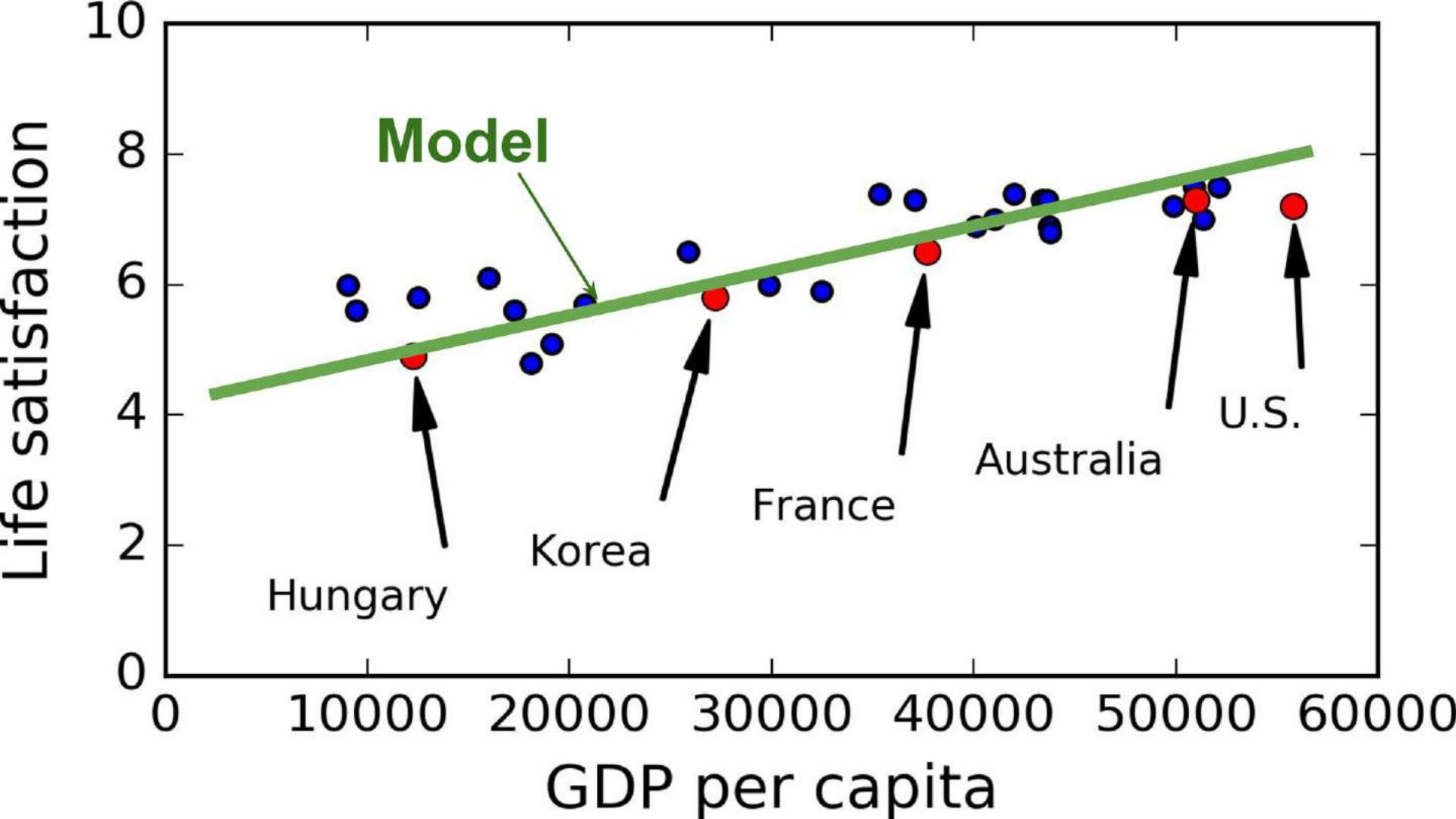


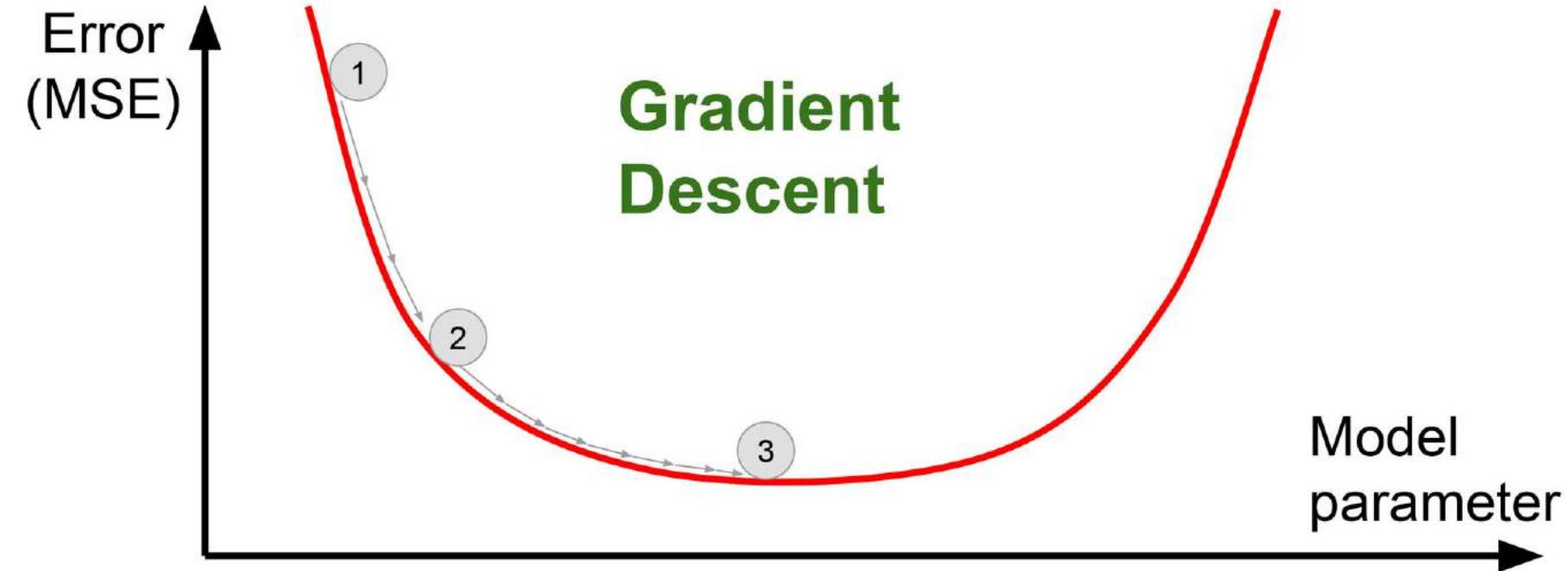
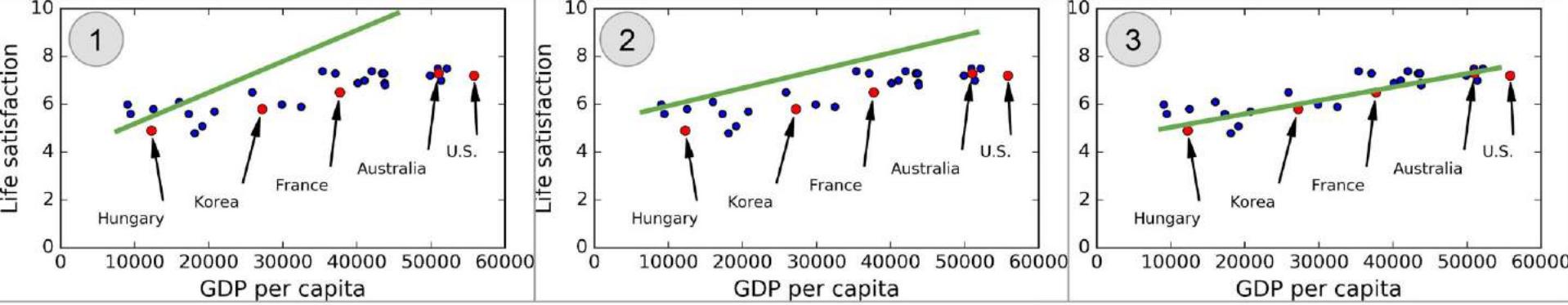
Life satisfaction



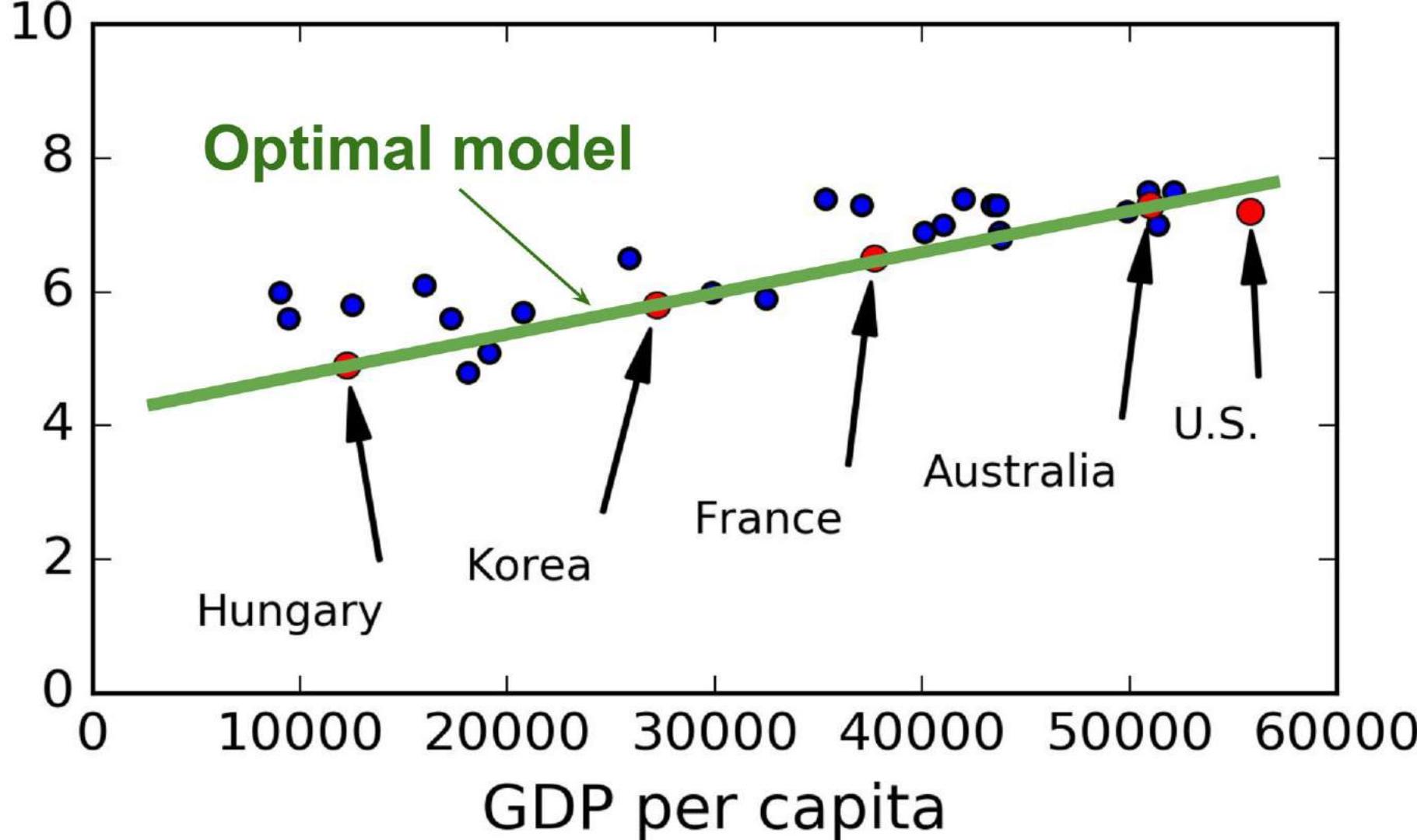


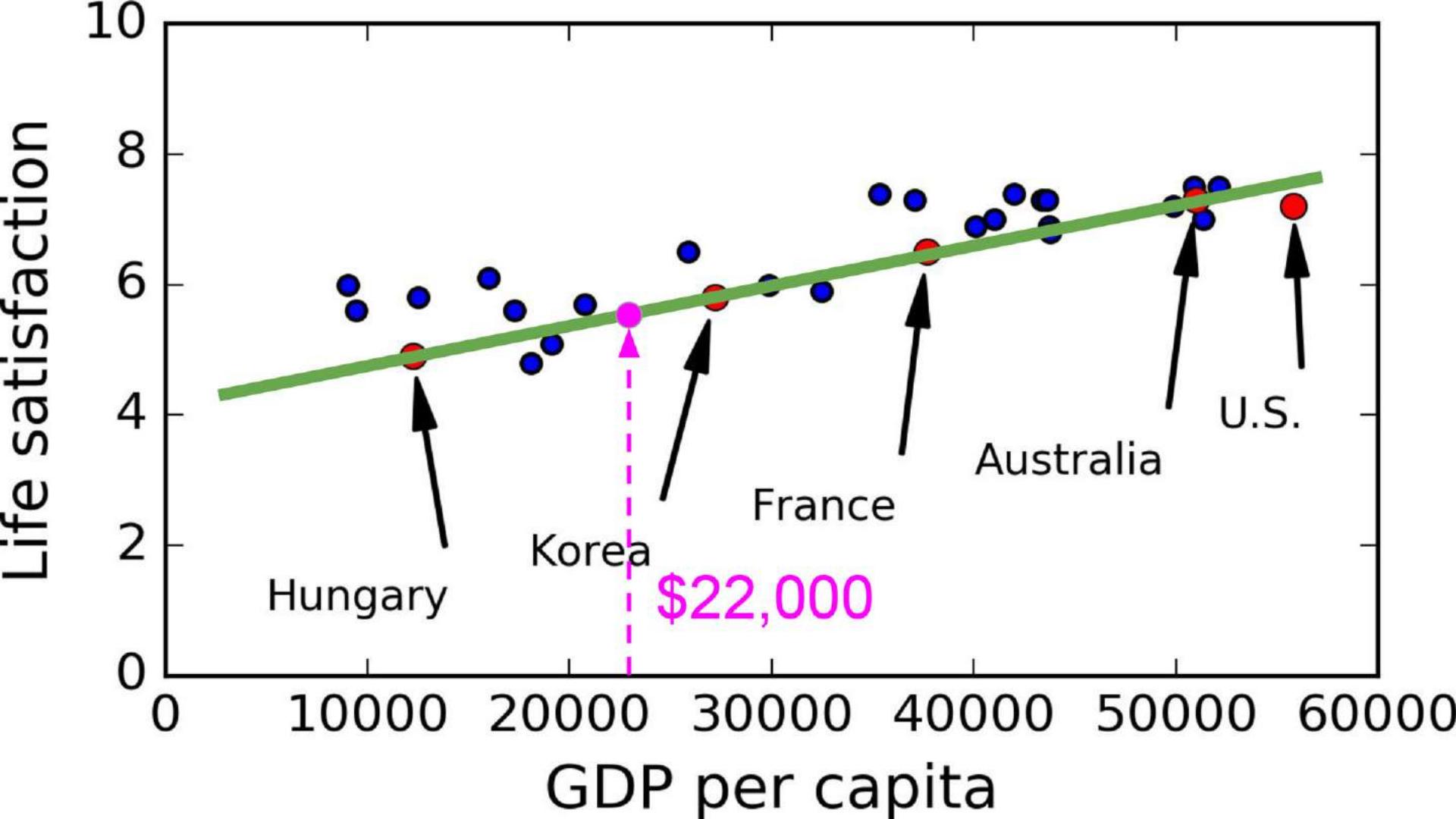




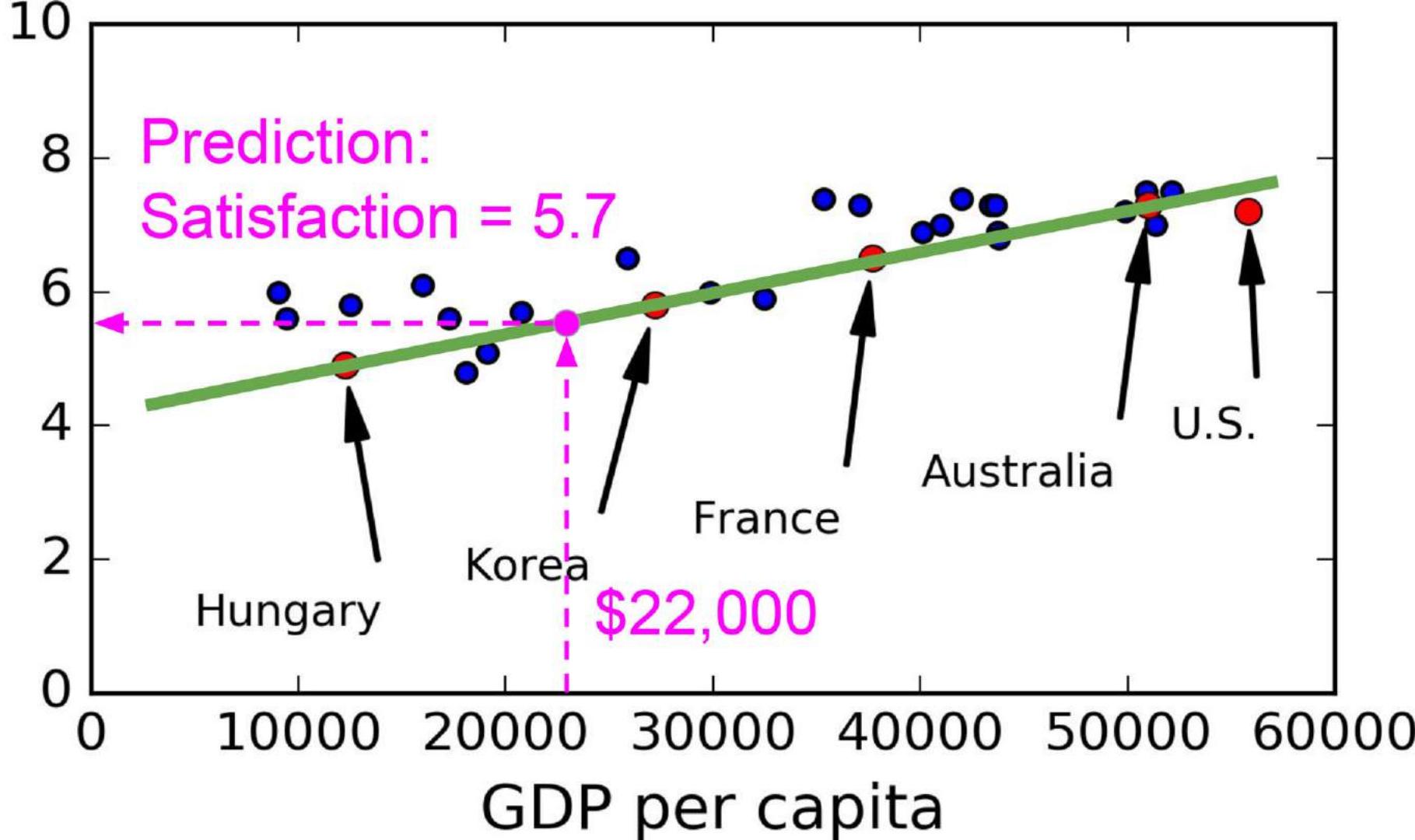


Life satisfaction





Life satisfaction

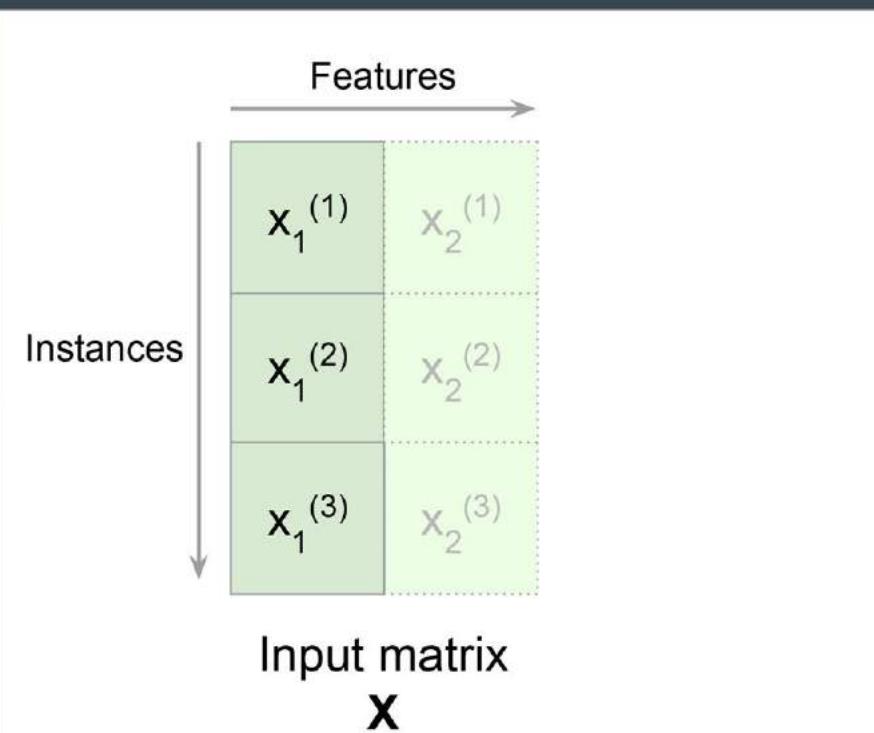


Linear Regression with TF > Loading the Data

```
>>> import numpy as np  
>>> data = np.loadtxt("data/life_satisfaction.csv",  
...                      dtype=np.float32,  
...                      delimiter=",",  
...                      skiprows=1,  
...                      usecols=[1, 2])  
...  
>>> x_train = data[:, 0:1] / 10000 # feature scaling  
>>> y_train = data[:, 1:2]
```

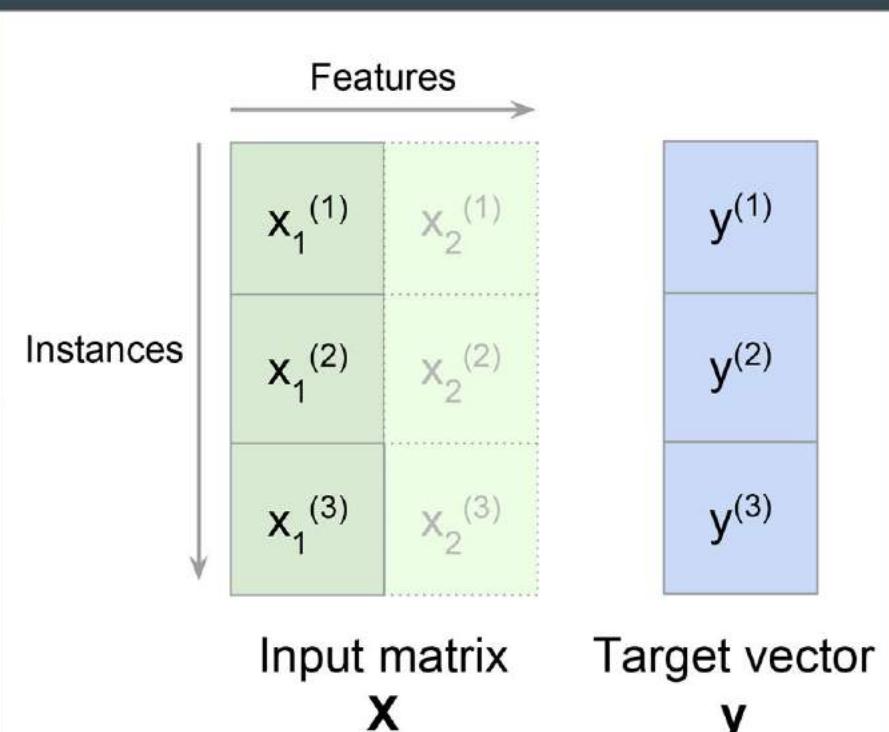
Linear Regression with TF > Loading the Data

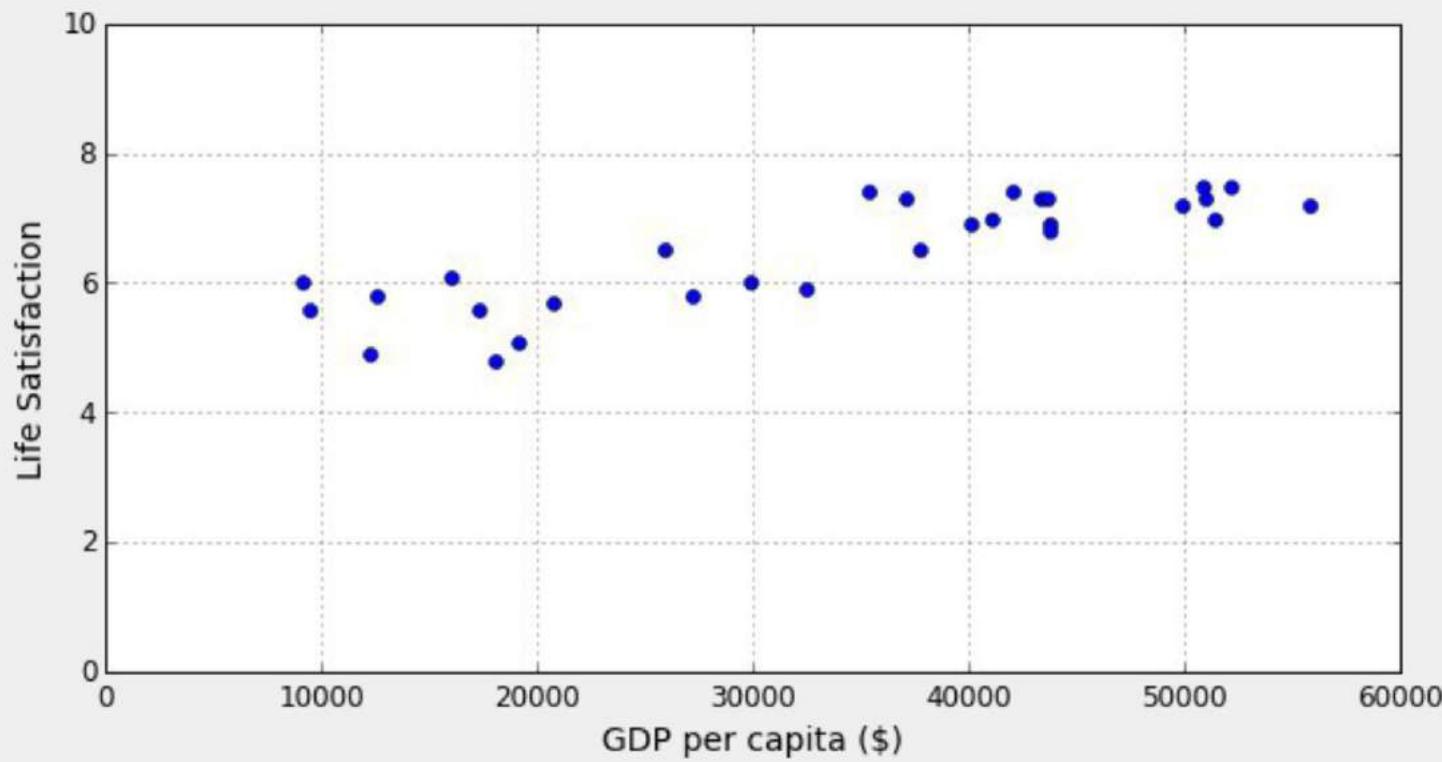
```
>>> import numpy as np  
>>> data = np.loadtxt("data/life_s  
...  
...  
...  
...  
...  
...  
>>> x_train = data[:, 0:1] / 10000  
>>> y_train = data[:, 1:2]
```



Linear Regression with TF > Loading the Data

```
>>> import numpy as np  
>>> data = np.loadtxt("data/life_s  
...  
...  
...  
...  
...  
...  
>>> x_train = data[:, 0:1] / 10000  
>>> y_train = data[:, 1:2]
```





25 minutes

Exercise 4



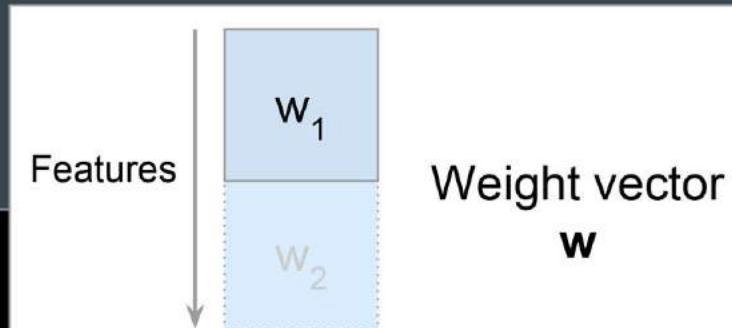
Building the Model

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     X = tf.constant(X_train, name="X")
...     y = tf.constant(y_train, name="y")
...     b = tf.Variable(0.0, name="b")
...     w = tf.Variable(tf.zeros([1, 1]), name="w")
...     y_pred = tf.add(tf.matmul(X, w), b, name="y_pred")
...     init = tf.global_variables_initializer()
```

Linear Regression with TF

Building the Model

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     X = tf.constant(X_train, name="X")
...     y = tf.constant(y_train, name="y")
...     b = tf.Variable(0.0, name="b")
...     w = tf.Variable(tf.zeros([1, 1]), name="w")
...     y_pred = tf.add(tf.matmul(X, w), b, name="y_pred")
...     init = tf.global_variables_initializer()
```



Building the Model

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     X = tf.constant(X_train, name="X")
...     y = tf.constant(y_train, name="y")
...     b = tf.Variable(0.0, name="b")
...     w = tf.Variable(tf.zeros([1, 1]), name="w")
...     y_pred = tf.add(tf.matmul(X, w), b, name="y pred")
...     init = tf.global_variables_initializer()
```

$$\hat{y} = X \cdot w + b$$

Linear Regression with TF > Building the

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.constant(x_t)
...     y = tf.constant(y_t)
...     b = tf.Variable(0.0)
...     w = tf.Variable(tf.
...     y_pred = tf.add(tf.
...     init = tf.global_variables_initializer()
```

Instances

$$\hat{y}^{(1)} = w_1 x_1^{(1)} + w_2 x_2^{(1)} + \dots + b$$

$$\hat{y}^{(2)} = w_1 x_1^{(2)} + w_2 x_2^{(2)} + \dots + b$$

$$\hat{y}^{(3)} = w_1 x_1^{(3)} + w_2 x_2^{(3)} + \dots + b$$

Prediction vector
 \hat{y}

$$\hat{y} = X \cdot w + b$$

Linear Regression with TF

Computing the Error

```
>>> with graph.as_default():
...     error = y_pred - y
...     square_error = tf.square(error)
...     mse = tf.reduce_mean(square_error, name="mse")
```

$$\text{MSE}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

Linear Regression with TF >

Computing the Gradients

```
>>> learning_rate = 0.01
>>> with graph.as_default():
...     m = len(X_train)
...     gradients_w = 2/m * tf.matmul(tf.transpose(X), error)
```

$$\nabla_{\mathbf{w}} \text{MSE}(\mathbf{w}, b) = \begin{pmatrix} \frac{\partial}{\partial w_0} \text{MSE}(\mathbf{w}, b) \\ \frac{\partial}{\partial w_1} \text{MSE}(\mathbf{w}, b) \\ \vdots \\ \frac{\partial}{\partial w_n} \text{MSE}(\mathbf{w}, b) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})$$

Linear Regression with TF

Computing the Gradients

```
>>> learning_rate = 0.01
>>> with graph.as_default():
...     m = len(X_train)
...     gradients_w = 2/m * tf.matmul(tf.transpose(X), error)
...     gradients_b = 2 * tf.reduce_mean(error)
```

$$\frac{\partial}{\partial b} \text{MSE}(\mathbf{w}, b) = \frac{2}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})$$

Linear Regression with TF

Gradient Descent Step

```
>>> learning_rate = 0.01
>>> with graph.as_default():
...     m = len(X_train)
...     gradients_w = 2/m * tf.matmul(tf.transpose(X), error)
...     gradients_b = 2 * tf.reduce_mean(error)
...     tweak_w_op = tf.assign(w, w - learning_rate * gradients_w)
...     tweak_b_op = tf.assign(b, b - learning_rate * gradients_b)
```

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{MSE}(\mathbf{w}, \mathbf{b})$$

$$\mathbf{b} \leftarrow \mathbf{b} - \eta \frac{\partial}{\partial b} \text{MSE}(\mathbf{w}, \mathbf{b})$$

Gradient Descent Step

```
>>> learning_rate = 0.01
>>> with graph.as_default():
...     m = len(X_train)
...     gradients_w = 2/m * tf.matmul(tf.transpose(X), error)
...     gradients_b = 2 * tf.reduce_mean(error)
...     tweak_w_op = tf.assign(w, w - learning_rate * gradients_w)
...     tweak_b_op = tf.assign(b, b - learning_rate * gradients_b)
...     training_op = tf.group(tweak_w_op, tweak_b_op)
```

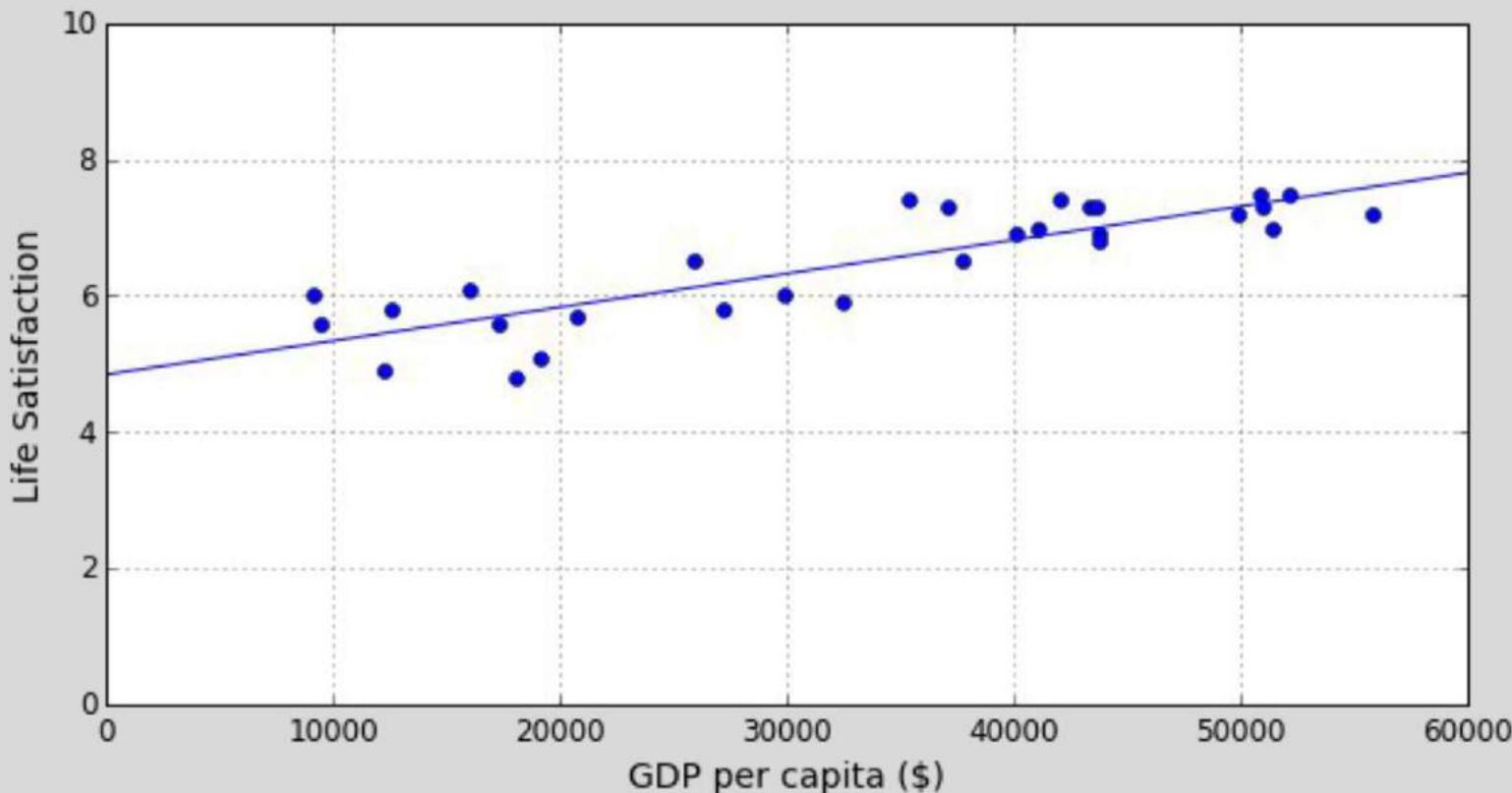
Linear Regression with TF > **It's Training Time!**

```
>>> n_iterations = 2000
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     for iteration in range(n_iterations):
...         if iteration % 100 == 0:
...             print("Iteration {:5}, MSE: {:.4f}"
...                   .format(iteration, mse.eval()))
...             training_op.run()
...     w_val, b_val = sess.run([w, b])
...
Iteration      0, MSE: 7.1302
Iteration    100, MSE: 1.4402
[...]
```

Linear Regression with TF >

Making Predictions

```
>>> cyprus_gdp_per_capita = 22000  
>>> w_val[0][0] * cyprus_gdp_per_capita / 10000 + b_val  
5.9329453289508818
```



Using Autodiff and Optimizers

Computing Gradients Manually

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     [...]
...     m = len(X_train)
...     error = y_pred - y
...     gradients_w = 2/m * tf.matmul(tf.transpose(X), error)
...     gradients_b = 2 * tf.reduce_mean(error)
...     [...]
```

Computing Gradients Using Autodiff

```
>>> graph = tf.Graph()
>>> with graph.as_default():
... [...]
...     gradients_w, gradients_b = tf.gradients(mse, [w, b])
... [...]
```

Computing Gradients Using Autodiff

```
>>> graph = tf.Graph()  
>>> with graph.as_default():  
... [...]  
...     gradients_w, gradients_b = tf.gradients(mse, [w, b])  
... [...]
```



Performing Gradient Descent Manually

```
>>> graph = tf.Graph()
>>> with graph.as_default():
... [...]
...     gradients_w, gradients_b = tf.gradients(mse, [w, b])
...     tweak_w_op = tf.assign(w, w - learning_rate*gradients_w)
...     tweak_b_op = tf.assign(b, b - learning_rate*gradients_b)
...     training_op = tf.group(tweak_w_op, tweak_b_op)
... [...]
```

Performing Gradient Descent Using An Optimizer

```
>>> graph = tf.Graph()
>>> with graph.as_default():
... [...]
...     optimizer = tf.train.GradientDescentOptimizer(learning_rate)
...     training_op = optimizer.minimize(mse)
... [...]
```

Performing Gradient Descent Using An Optimizer

```
>>> graph = tf.Graph()  
>>> with graph.as_default():  
... [...]  
...     optimizer = tf.train.GradientDescentOptimizer(learning_rate)  
...     training_op = optimizer.minimize(mse)  
... [...]
```

A cartoon illustration of a wizard with a long white beard, wearing a purple robe and a blue turban. He is holding a wooden staff with a glowing blue orb at the top. A large, white, five-pointed starburst shape is positioned behind him, containing the text "More magic!"

More magic!

Faster Optimizers

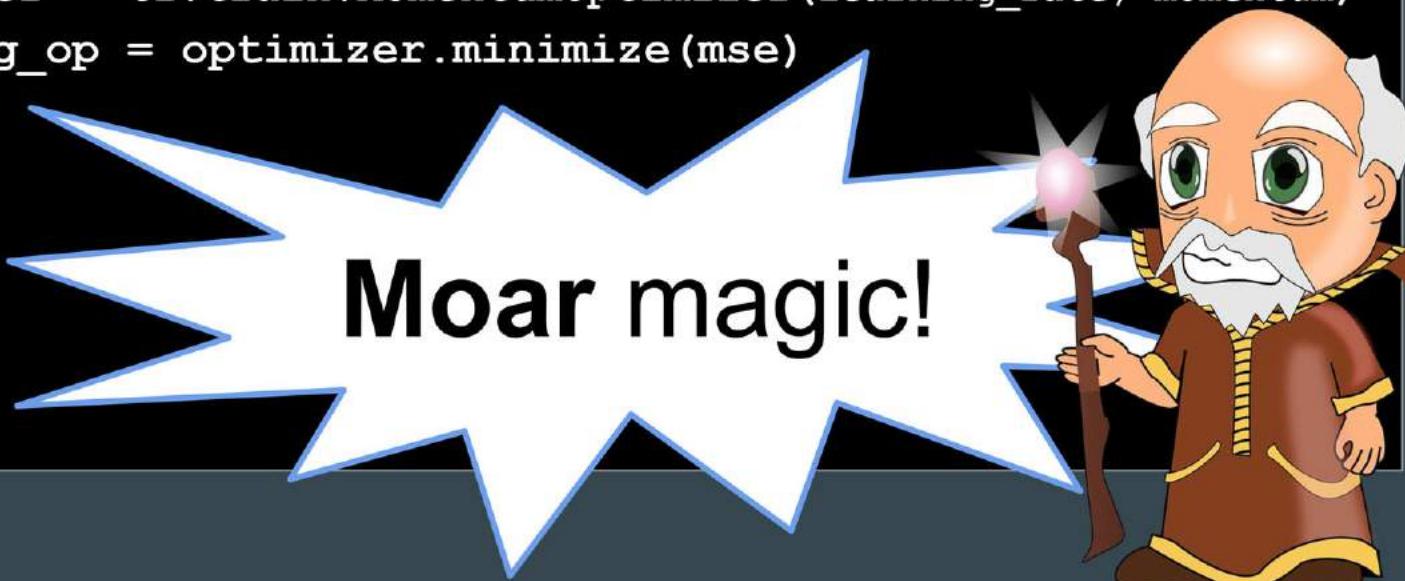
```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     [...]
...     optimizer = tf.train.MomentumOptimizer(learning_rate, momentum)
...     training_op = optimizer.minimize(mse)
...     [...]
```

Autodiff & Optimizers >

Faster Optimizers

```
>>> graph = tf.Graph()  
>>> with graph.as_default():  
... [...]  
...     optimizer = tf.train.MomentumOptimizer(learning_rate, momentum)  
...     training_op = optimizer.minimize(mse)  
... [...]
```

Moar magic!



Autodiff & Optimizers >

Trainable Variables

```
>>> coll = graph.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES)
>>> [var.op.name for var in coll]
[ 'b', 'w' ]
```

Autodiff & Optimizers >

Trainable Variables

```
>>> coll = graph.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES)  
>>> [var.op.name for var in coll]  
[ 'b', 'w' ]
```



By default, variables are trainable!

Example of a non-trainable variable: Global Step

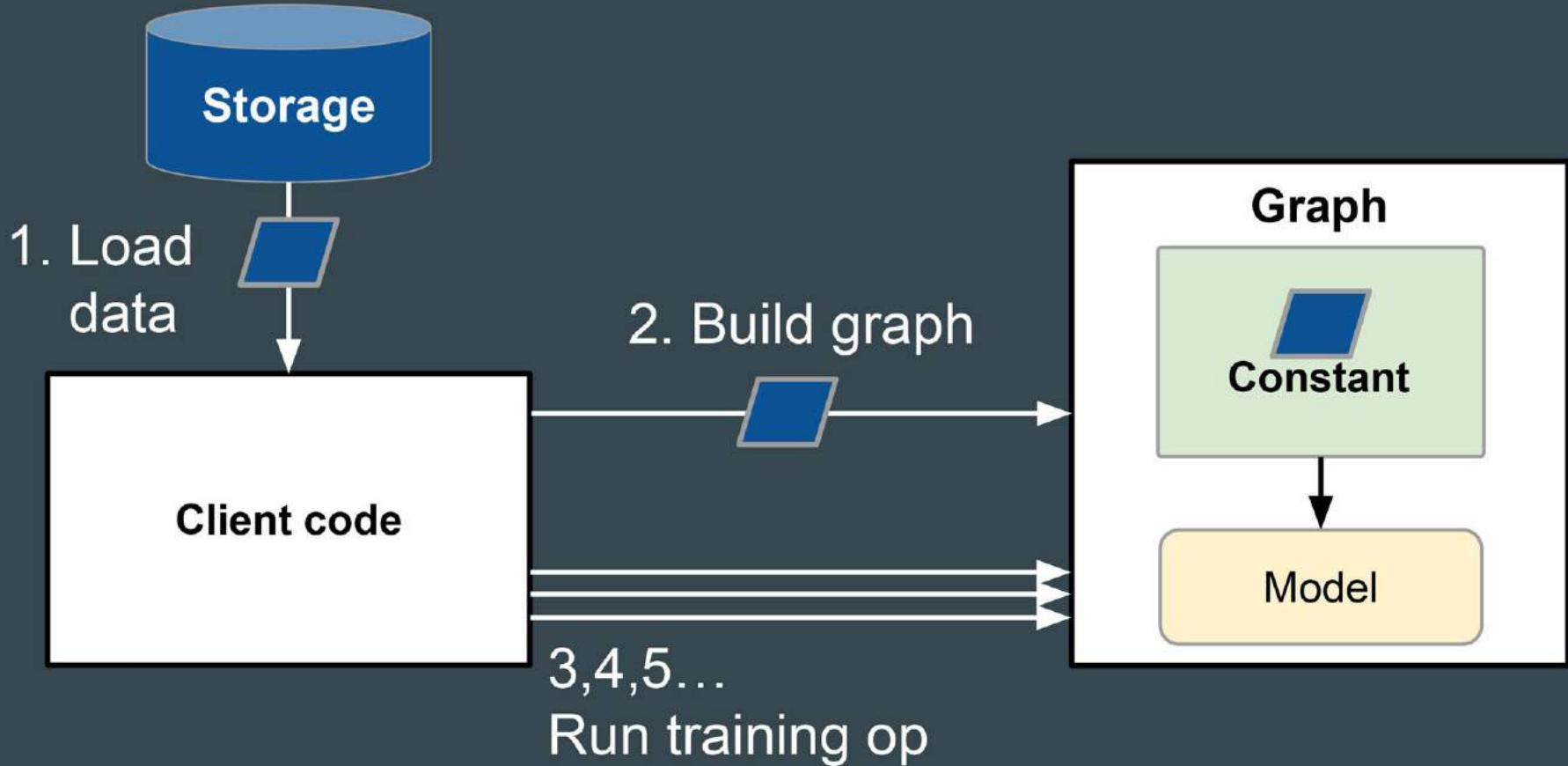
```
>>> coll = graph.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES)
>>> [var.op.name for var in coll]
['b', 'w']
>>> global_step = tf.Variable(0, trainable=False, name='global_step')
```

Example of a non-trainable variable: Global Step

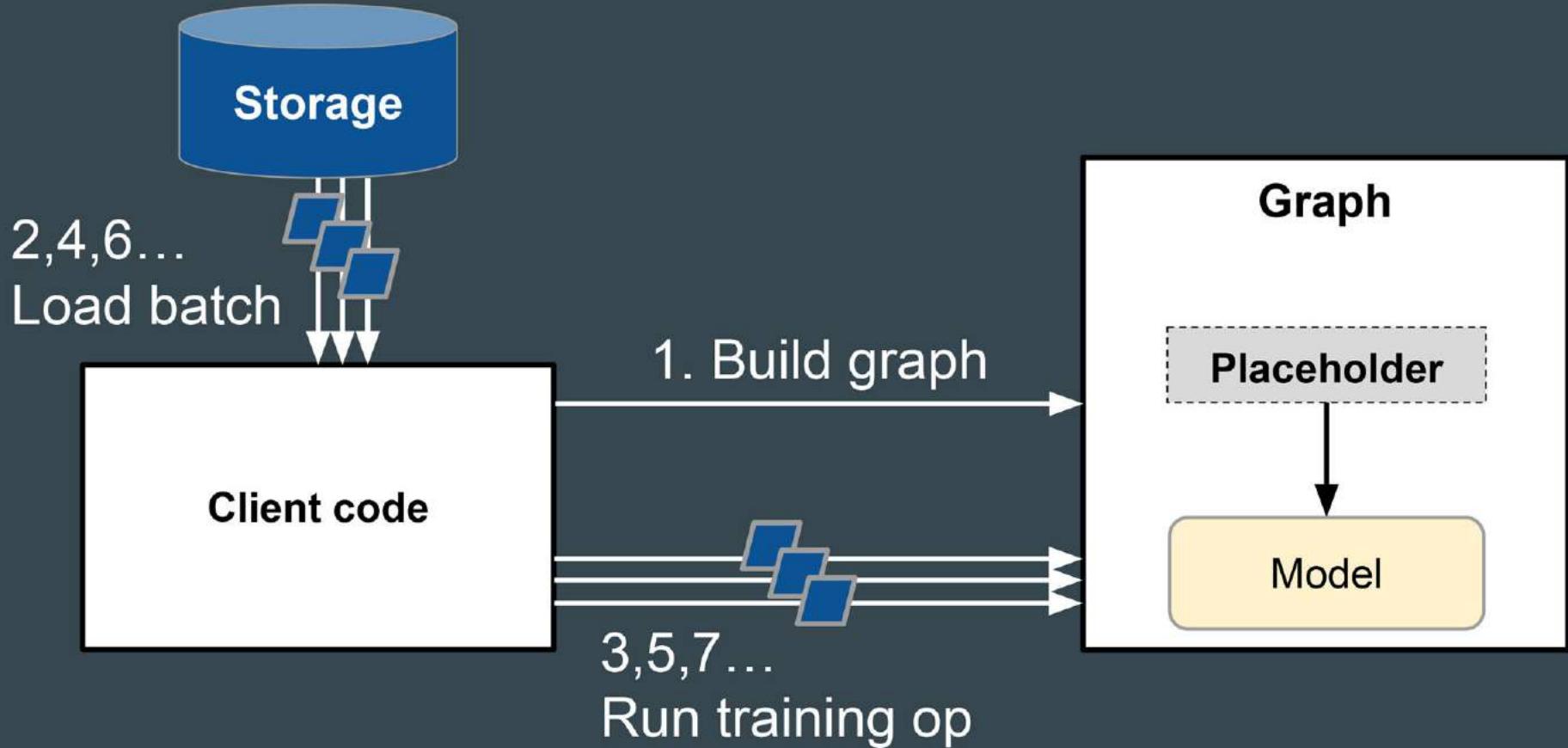
```
>>> coll = graph.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES)
>>> [var.op.name for var in coll]
['b', 'w']
>>> global_step = tf.Variable(0, trainable=False, name='global_step')
>>> training_op = optimizer.minimize(mse, global_step=global_step)
```

Feeding Data to a Model

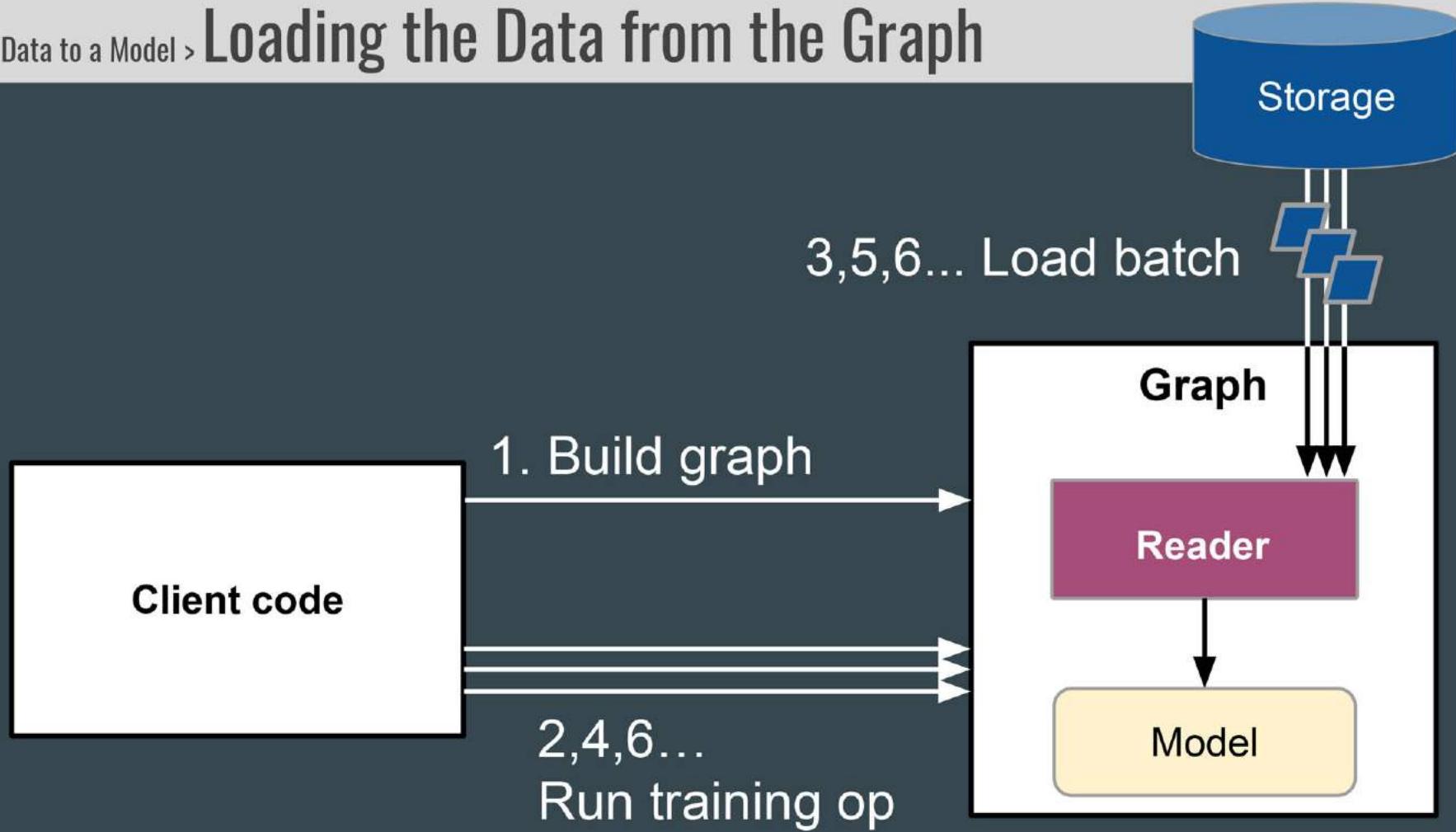
Preloading the Data



Feeding the Data at Runtime



Loading the Data from the Graph



Building the Model with Preloaded Data

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.constant(X_train, name="X")
...     y = tf.constant(y_train, name="y")
...     b = tf.Variable(0.0)
...     w = tf.Variable(tf.zeros([1, 1]))
...     y_pred = tf.add(tf.matmul(X, w), b)
...     init = tf.global_variables_initializer()
```

$$\hat{y} = X \cdot w + b$$

Building the Model with Placeholders

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.placeholder(tf.float32, shape=[None, 1], name="X")
...     y = tf.placeholder(tf.float32, shape=[None, 1], name="y")
...     b = tf.Variable(0.0)
...     w = tf.Variable(tf.zeros([1, 1]))
...     y_pred = tf.add(tf.matmul(X, w), b)
...     init = tf.global_variables_initializer()
```

$$\hat{y} = X \cdot w + b$$

Feeding Data to a Model >

It's Training Time with Preloaded Data!

```
>>> n_iterations = 2000
>>>
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     for iteration in range(n_iterations):
...         training_op.run()
```

Feeding Data to a Model >

It's Training Time with Placeholders!

```
>>> n_iterations = 2000
>>>
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     for iteration in range(n_iterations):
...         training_op.run(feed_dict={X: X_train, y: y_train})
```

Feeding Data to a Model >

It's Training + Test Time with Placeholders!

```
>>> n_iterations = 2000
>>> X_test = np.array([[22000]], dtype=np.float32) / 10000
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     for iteration in range(n_iterations):
...         training_op.run(feed_dict={X: X_train, y: y_train})
...
...
```

Feeding Data to a Model >

It's Training + Test Time with Placeholders!

```
>>> n_iterations = 2000
>>> x_test = np.array([[22000]], dtype=np.float32) / 10000
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     for iteration in range(n_iterations):
...         training_op.run(feed_dict={X: X_train, y: y_train})
...     y_pred_val = y_pred.eval(feed_dict={X: x_test})
...
>>> y_pred_val
array([[ 5.93294525]], dtype=float32)
```

12 minutes
(optional)

Exercise 5



Data API

Data API >

Datasets and Iterators

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     dataset = tf.data.Dataset.from_tensor_slices(np.arange(10))
...     dataset = dataset.repeat(3).batch(7)
...     iterator = dataset.make_one_shot_iterator()
...     next_element = iterator.get_next()
```

Data API > Execution Phase

```
>>> with tf.Session(graph=graph) as sess:  
...     try:  
...         while True:  
...             print(next_element.eval())  
...     except tf.errors.OutOfRangeError:  
...         pass  
...  
[0 1 2 3 4 5 6]  
[7 8 9 0 1 2 3]  
[4 5 6 7 8 9 0]  
[1 2 3 4 5 6 7]  
[8 9]
```

Data API > Execution Phase

```
>>> with tf.Session(graph=graph) as sess:  
...     try:  
...         while True:  
...             print(sess.run([next_element, next_element]))  
...     except tf.errors.OutOfRangeError:  
...         pass  
...  
[array([0, 1, 2, 3, 4, 5, 6]), array([0, 1, 2, 3, 4, 5, 6])]  
[array([7, 8, 9, 0, 1, 2, 3]), array([7, 8, 9, 0, 1, 2, 3])]  
[array([4, 5, 6, 7, 8, 9, 0]), array([4, 5, 6, 7, 8, 9, 0])]  
[array([1, 2, 3, 4, 5, 6, 7]), array([1, 2, 3, 4, 5, 6, 7])]  
[array([8, 9]), array([8, 9])]
```

```
>>> with graph.as_default():
...     dataset = tf.data.Dataset.from_tensor_slices(np.arange(10))
...     dataset = dataset.repeat(3).batch(7)
...     dataset = dataset.interleave(
...         lambda v: tf.data.Dataset.from_tensor_slices(v),
...         cycle_length=3,
...         block_length=2)
...     iterator = dataset.make_one_shot_iterator()
...     next_element = iterator.get_next()
...
...
```

Data API > Interleave

```
>>> with tf.Session(graph=graph) as sess:  
...     try:  
...         while True:  
...             print(next_element.eval(), end="")  
...     except tf.errors.OutOfRangeError:  
...         pass  
... 
```

017845239067451289630128934567

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

Data API > Interleave

```
>>> with tf.Session(graph=graph) as sess:  
...     try:  
...         while True:  
...             print(next_element.eval(), end="")  
...     except tf.errors.OutOfRangeError:  
...         pass  
... 
```

017845239067451289630128934567

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

Data API > **Interleave**

```
>>> with tf.Session(graph=graph) as sess:  
...     try:  
...         while True:  
...             print(next_element.eval(), end="")  
...     except tf.errors.OutOfRangeError:  
...         pass  
... 
```

017845239067451289630128934567

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

Data API > Interleave

```
>>> with tf.Session(graph=graph) as sess:  
...     try:  
...         while True:  
...             print(next_element.eval(), end="")  
...     except tf.errors.OutOfRangeError:  
...         pass  
... 
```

```
017845239067451289630128934567
```

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

Data API >

Using Readers

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     dataset = tf.data.Dataset.list_files(filenames)
...     dataset = dataset.repeat(n_epochs)
...     dataset = dataset.interleave(
...         lambda fn: tf.data.TextLineDataset(fn).skip(1),
...                     cycle_length=n_readers)
...     dataset = dataset.map(
...         parser_fn, num_parallel_calls=n_map_threads)
...     dataset = dataset.batch(batch_size).prefetch(1)
...     iterator = dataset.make_one_shot_iterator()
...     country, gdp_per_capita, life_sat = iterator.get_next()
```

Data API > Parsing a CSV Line

```
>>> def parser_fn(record):
...     defs = [[''], [0.0], [0.0]]
...     return tf.decode_csv(record, record_defaults=defs)
```

Saving and Restoring a Model

Saving & Restoring a Model >

Saving the Model

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     X = tf.placeholder(tf.float32, shape=[None, 1], name="X")
...     y = tf.placeholder(tf.float32, shape=[None, 1], name="y")
...     b = tf.Variable(0.0)
...     w = tf.Variable(tf.zeros([1, 1]))
...     y_pred = tf.add(tf.matmul(X, w), b)
...     init = tf.global_variables_initializer()
...     saver = tf.train.Saver()
```

Saving & Restoring a Model >

Saving the Model

```
>>> n_iterations = 2000
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     for iteration in range(n_iterations):
...         feed_dict = {X: X_train, y: y_train}
...         training_op.run(feed_dict)
...         if iteration % 500 == 0:
...             saver.save(sess, "/tmp/my_ls_model.ckpt")
... 
```

Saving & Restoring a Model >

Saving the Model

```
>>> n_iterations = 2000
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     for iteration in range(n_iterations):
...         feed_dict = {X: X_train, y: y_train}
...         training_op.run(feed_dict)
...         if iteration % 500 == 0:
...             saver.save(sess, "/tmp/my_ls_model.ckpt")
saver.save(sess, "./my_ls_model")
... 
```

Saving & Restoring a Model >

Restoring the Model

```
>>> x_test = np.array([[22000]], dtype=np.float32) / 10000
>>> graph = tf.Graph()
>>> with graph.as_default():
...     X = tf.placeholder(tf.float32, shape=[None, 1], name="X")
...     [...]
>>> with tf.Session(graph=graph) as sess:
...     saver.restore(sess, "./my_ls_model")
...     y_pred_val = y_pred.eval(feed_dict={X: x_test})
...
>>> y_pred_val
array([[ 5.93294525]], dtype=float32)
```

Saving & Restoring a Model >

Restoring the Whole Graph

```
>>> x_test = np.array([[22000]], dtype=np.float32) / 10000
>>> graph = tf.Graph()
>>> with graph.as_default():
...     saver = tf.train.import_meta_graph("./my_ls_model.meta")
```

Saving & Restoring a Model >

Restoring the Whole Graph

```
>>> x_test = np.array([[22000]], dtype=np.float32) / 10000
>>> graph = tf.Graph()
>>> with graph.as_default():
...     saver = tf.train.import_meta_graph("./my_ls_model.meta")
...
>>> with tf.Session(graph=graph) as sess:
...     saver.restore(sess, "./my_ls_model")
```

Restoring the Whole Graph

```
>>> x_test = np.array([[22000]], dtype=np.float32) / 10000
>>> graph = tf.Graph()
>>> with graph.as_default():
...     saver = tf.train.import_meta_graph("./my_ls_model.meta")
...
>>> with tf.Session(graph=graph) as sess:
...     saver.restore(sess, "./my_ls_model")
...     y_pred = graph.get_tensor_by_name("y_pred:0")
```

Saving & Restoring a Model >

Restoring the Whole Graph

```
>>> X_test = np.array([[22000]], dtype=np.float32) / 10000
>>> graph = tf.Graph()
>>> with graph.as_default():
...     saver = tf.train.import_meta_graph("./my_ls_model.meta")
...
>>> with tf.Session(graph=graph) as sess:
...     saver.restore(sess, "./my_ls_model")
...     y_pred = graph.get_tensor_by_name("y_pred:0")
...     y_pred_val = y_pred.eval(feed_dict={"X:0": X_test})
```

Saving & Restoring a Model >

Restoring the Whole Graph

```
>>> X_test = np.array([[22000]], dtype=np.float32) / 10000
>>> graph = tf.Graph()
>>> with graph.as_default():
...     saver = tf.train.import_meta_graph("./my_ls_model.meta")
...
>>> with tf.Session(graph=graph) as sess:
...     saver.restore(sess, "./my_ls_model")
...     y_pred = graph.get_tensor_by_name("y_pred:0")
...     y_pred_val = y_pred.eval(feed_dict={"X:0": X_test})
...
>>> y_pred_val
array([[ 5.93294525]], dtype=float32)
```

TensorBoard

TensorBoard > Starting the Server

```
~$ cd tensorflow-safari-course
~/tensorflow-safari-course$ mkdir tf_logs
~/tensorflow-safari-course$ source env/bin/activate
(env) ~/tensorflow-safari-course$ tensorboard --logdir=tf_logs
Starting TensorBoard b'41' on port 6006
(You can navigate to http://127.0.1.1:6006)
```

TensorBoard > Starting the Server

```
~$ cd tensorflow-safari-course  
~/tensorflow-safari-course$ mkdir tf_logs  
~/tensorflow-safari-course$ source env/bin/activate  
(env) ~/tensorflow-safari-course$ tensorboard --logdir=tf_logs  
Starting TensorBoard b'41' on port 6006  
(You can navigate to http://127.0.1.1:6006)
```



This is goog(le) upside down. ;)

TensorBoard > Logging the Graph

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     [...]
...
>>> log_dir = "tf_logs/run_1/"
>>> summary_writer = tf.summary.FileWriter(logdir, graph)
```

TensorBoard > Logging the Graph

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     [...]
...
>>> from datetime import datetime
>>> def logdir():
...     root_logdir = "tf_logs"
...     now = datetime.utcnow().strftime("%Y%m%d%H%M%S")
...     return "{}/run_{}".format(root_logdir, now)
...
>>> summary_writer = tf.summary.FileWriter(logdir(), graph)
```

TensorBoard >

Output of Multiple Runs

```
(env) ~/tensorflow-safari-course$ tree tf_logs
tf_logs
└── run_20170402193727
    └── events.out.tfevents.1491161847.hoppy
└── run_20170402193732
    └── events.out.tfevents.1491161852.hoppy
```



Fit to screen



Download PNG

Run
(2)

Session

runs (0)

Upload

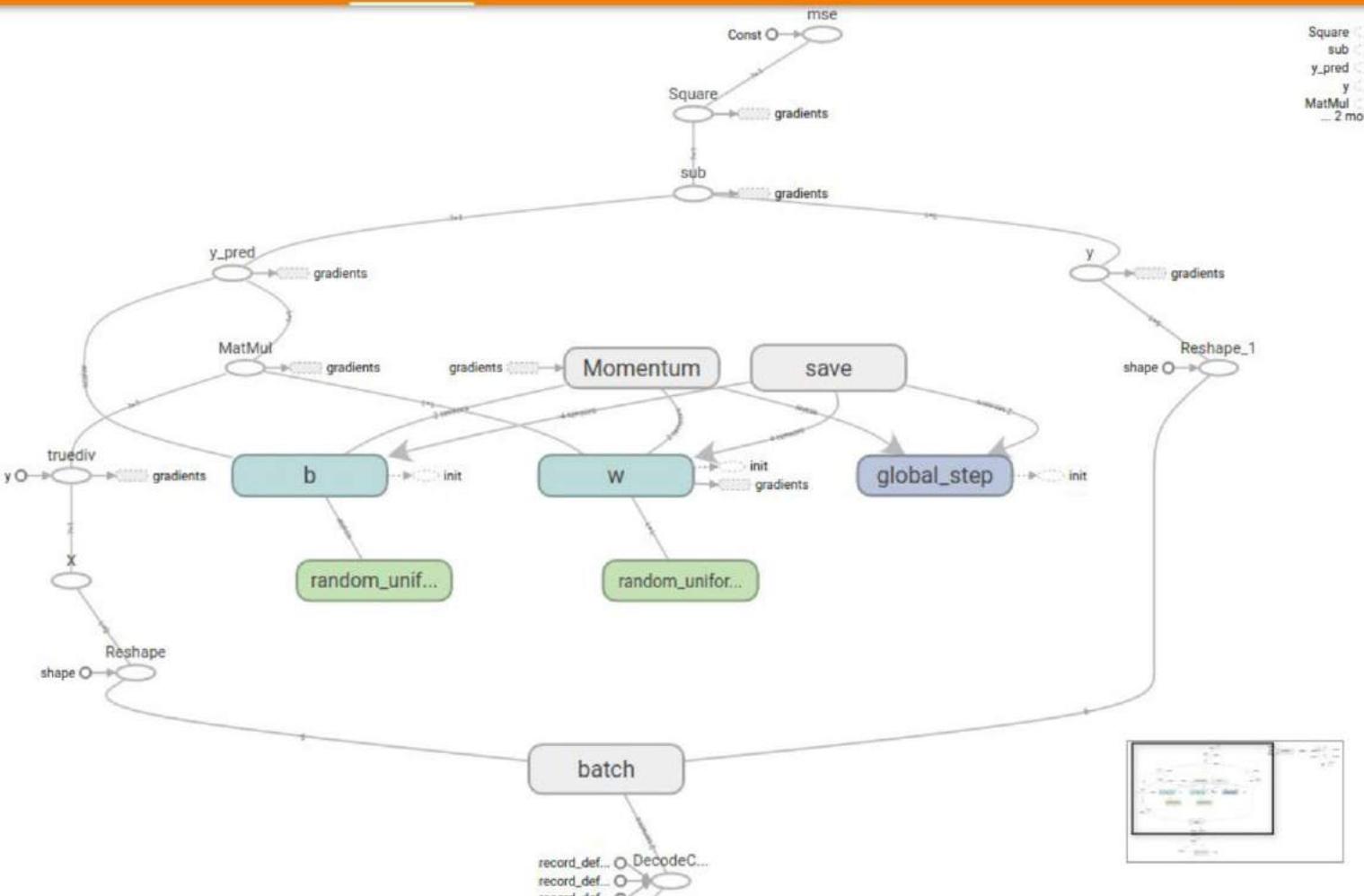
Choose File

Trace inputs Color Structure Device

colors same substructure

unique substructure

Graph (* = expandable)

 Namespace* OpNode Unconnected series* Connected series* Constant Summary Dataflow edge Control dependency edge Reference edge

Name scopes

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     with tf.name_scope("linear_model"):
...         x = tf.placeholder_with_default(..., [...])
...         with tf.name_scope("train"):
...             mse = tf.reduce_mean(tf.square(y_pred - y), ...)
...             [...]
```

Name scopes

```
>>> X.name
'linear_model/X:0'
>>> y.name
'linear_model/y:0'
>>> b.name
'linear_model/b:0'
>>> mse.name
'train/mse:0'
```



Fit to screen



Download PNG

Run
(2)

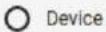
Session runs (0)



Choose File



Trace inputs

Color Structure

colors same substructure



Graph (* = expandable)



Namespace*



OpNode



Unconnected series*



Connected series*



Constant



Summary



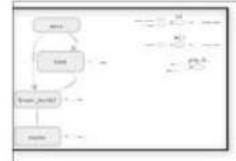
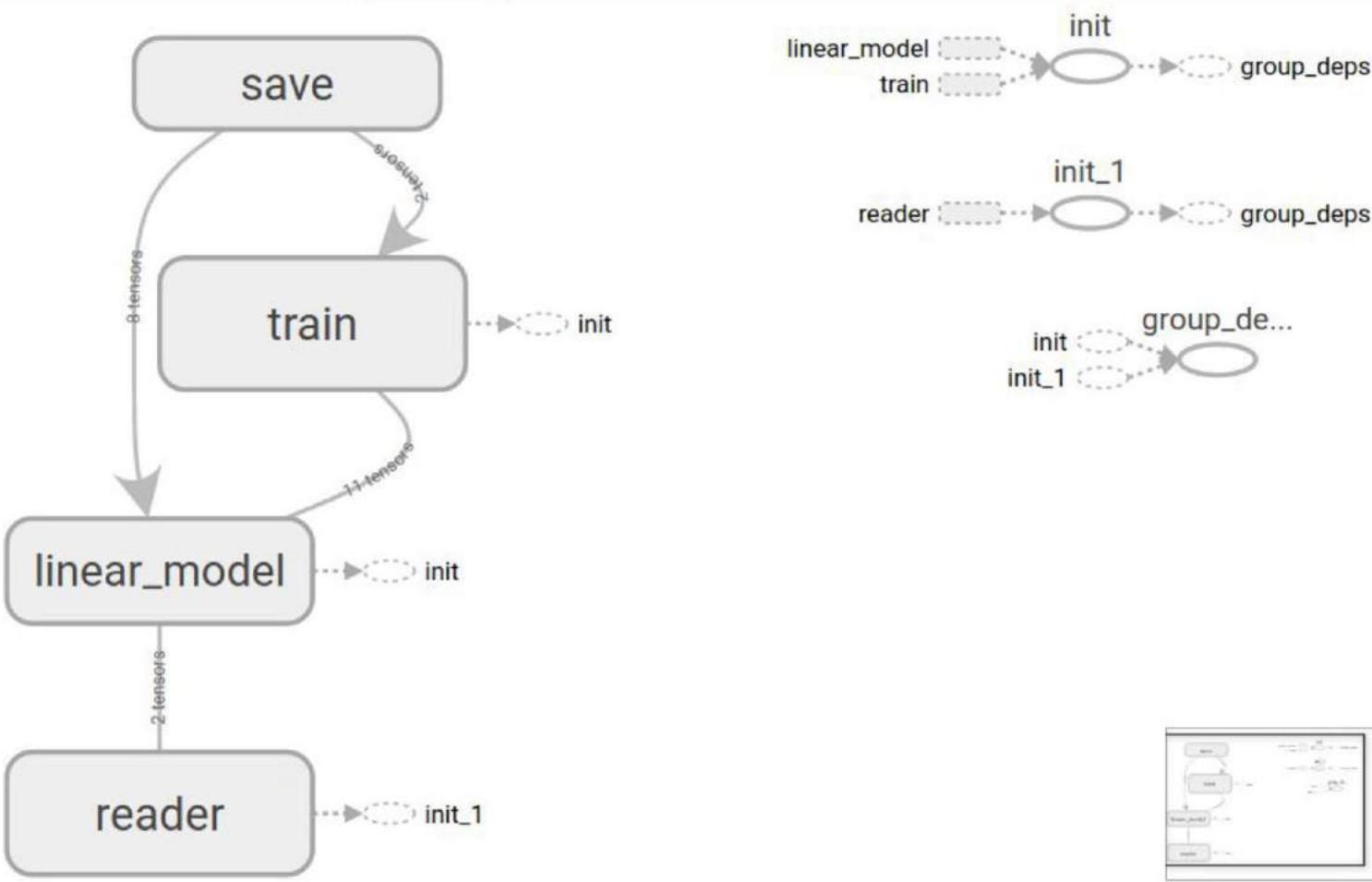
Dataflow edge



Control dependency edge



Reference edge





Fit to screen



Download PNG

Run
(2)Session
runs (0)

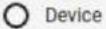
Choose File



Trace inputs



Structure



Device



colors same substructure



unique substructure



Graph (* = expandable)



Namespace*



OpNode



Unconnected series*



Connected series*



Constant



Summary



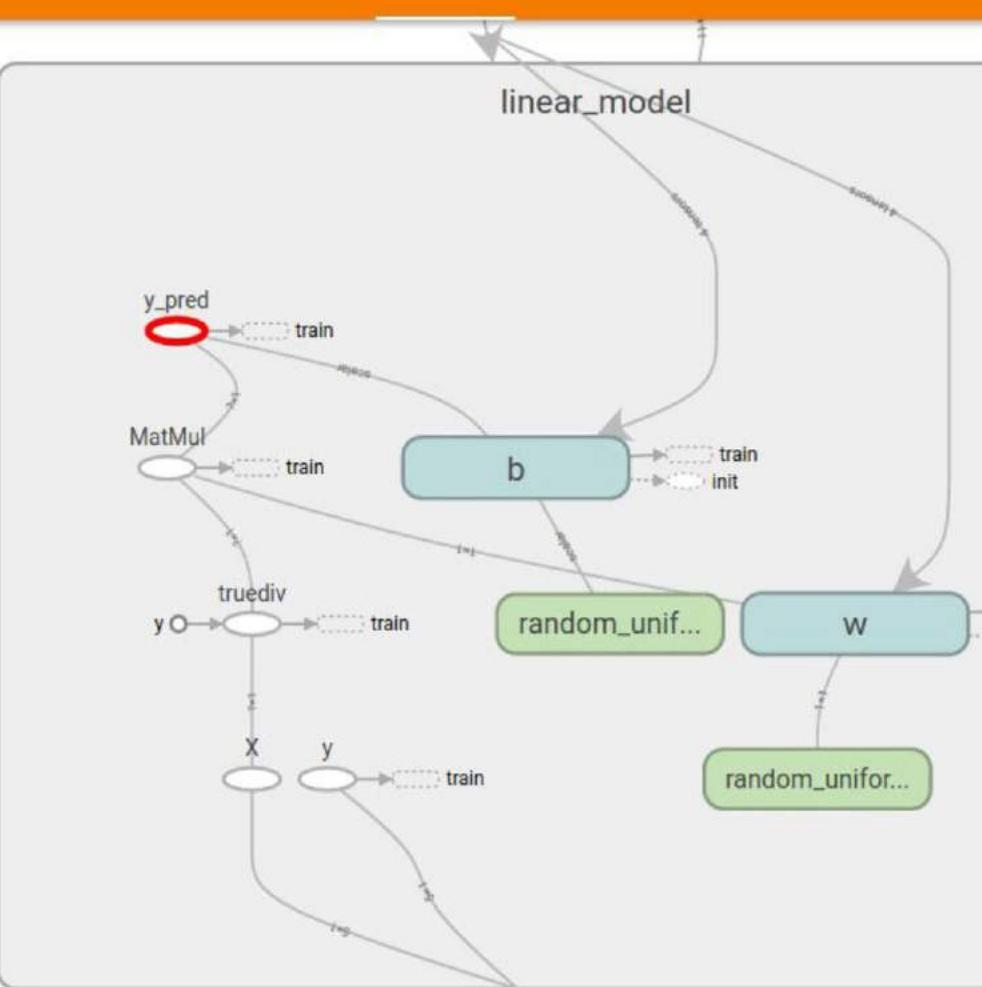
Dataflow edge



Control dependency edge



Reference edge

**linear_model/y_pred**

Operation: Add

Attributes (1)

T {"type": "DT_FLOAT"}

Inputs (2)

linear_model/MatMul

linear_model/b/read

scalar

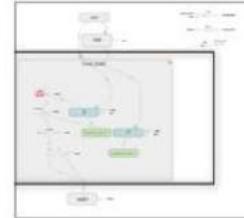
Outputs (2)

train/sub

train/gradients/train/sub_grad/Shape

?x1

Remove from main graph



TensorBoard > Plotting the MSE During Training

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     [...]
...     with tf.name_scope("train"):
...         mse = tf.reduce_mean(tf.square(y_pred - y),
...                             name="mse")
...         mse_summary = tf.summary.scalar('MSE', mse)
...     [...]
```

TensorBoard > Plotting the MSE During Training

```
>>> with tf.Session(graph=graph) as sess:  
...     [...]  
...     while True:  
...         _, mse_summary_val, step_val = (  
...             sess.run([training_op, mse_summary, global_step])  
...         [...]
```

TensorBoard > Plotting the MSE During Training

```
>>> with tf.Session(graph=graph) as sess:  
...     [...]  
...     while True:  
...         _, mse_summary_val, step_val = (  
...             sess.run([training_op, mse_summary, global_step])  
...         if step_val % 10 == 0:  
...             summary_writer.add_summary(mse_summary_val, step_val)  
...     [...]
```

Write a regex to create a tag group

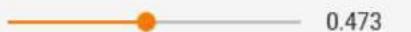


Split on underscores

Data download links

Tooltip sorting method: **default** ▾

Smoothing



Horizontal Axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

run-20170402201131

run-20170402201251

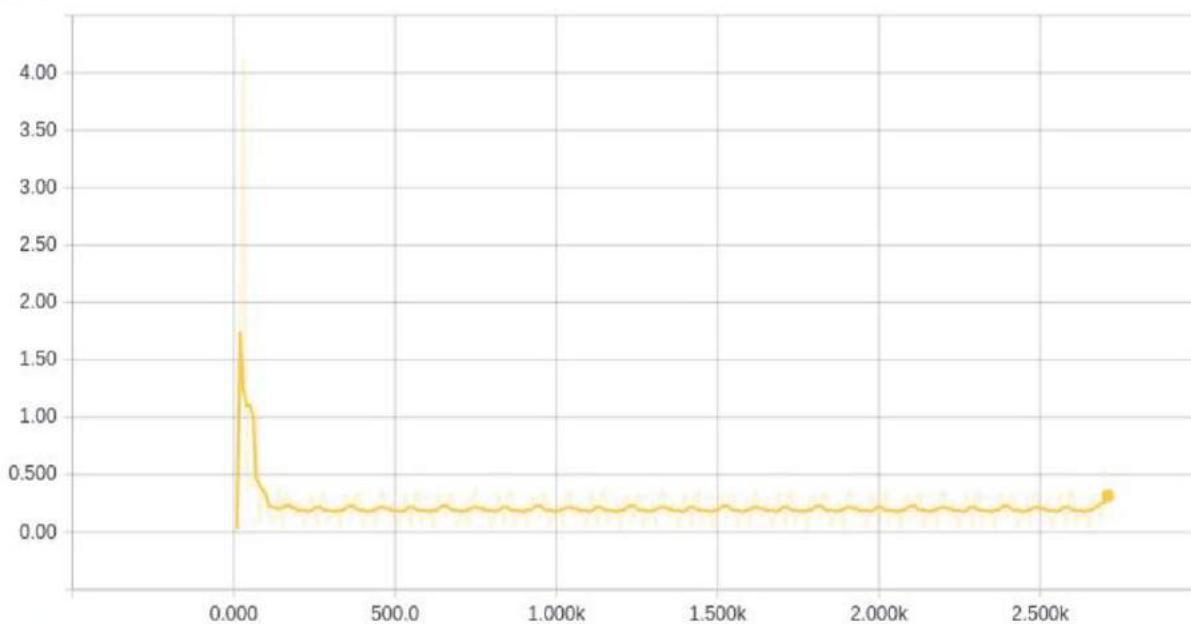
TOGGLE ALL RUNS

tf_logs

train

1

train/MSE



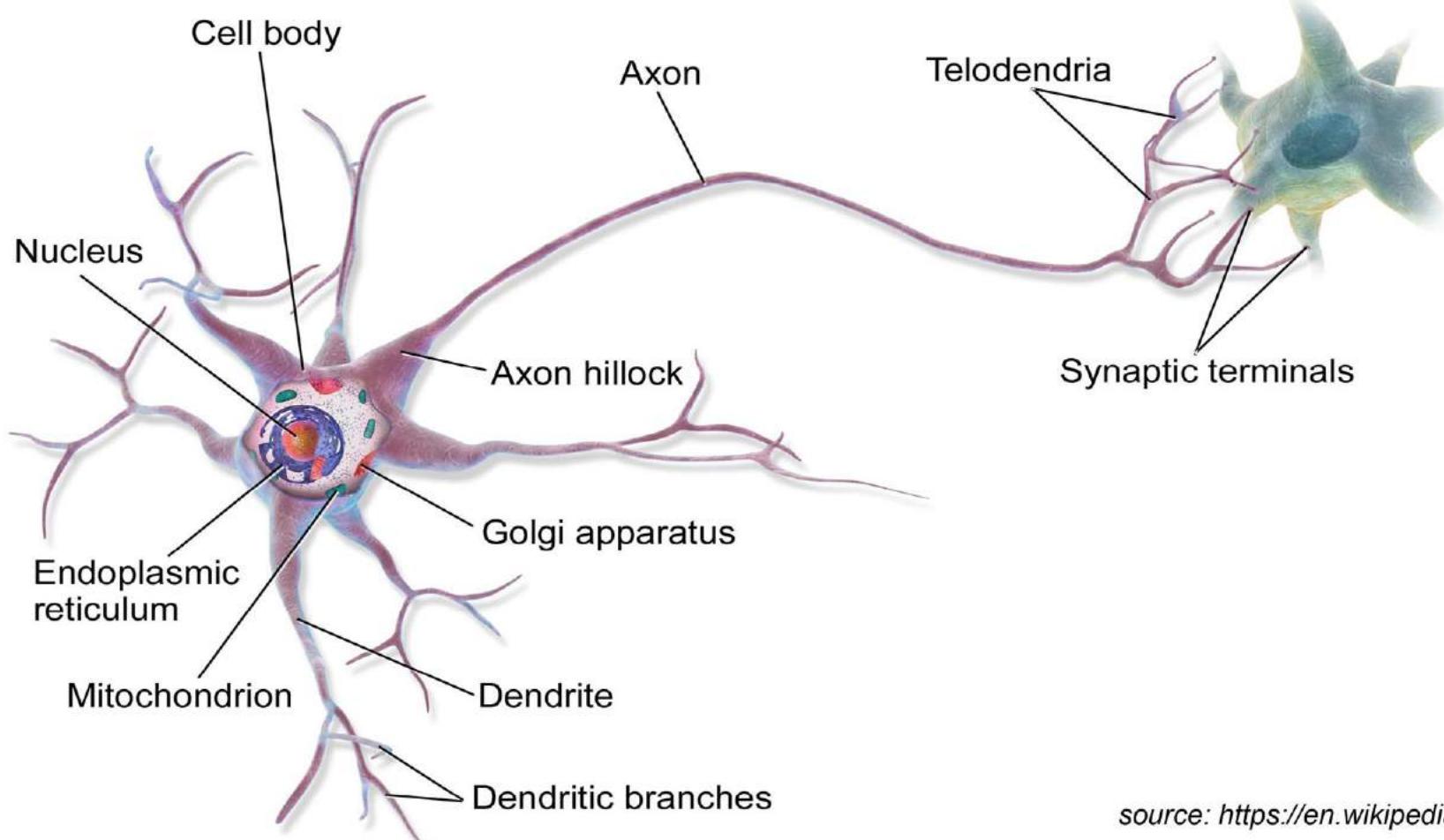
20 minutes
(homework)

Exercise 6

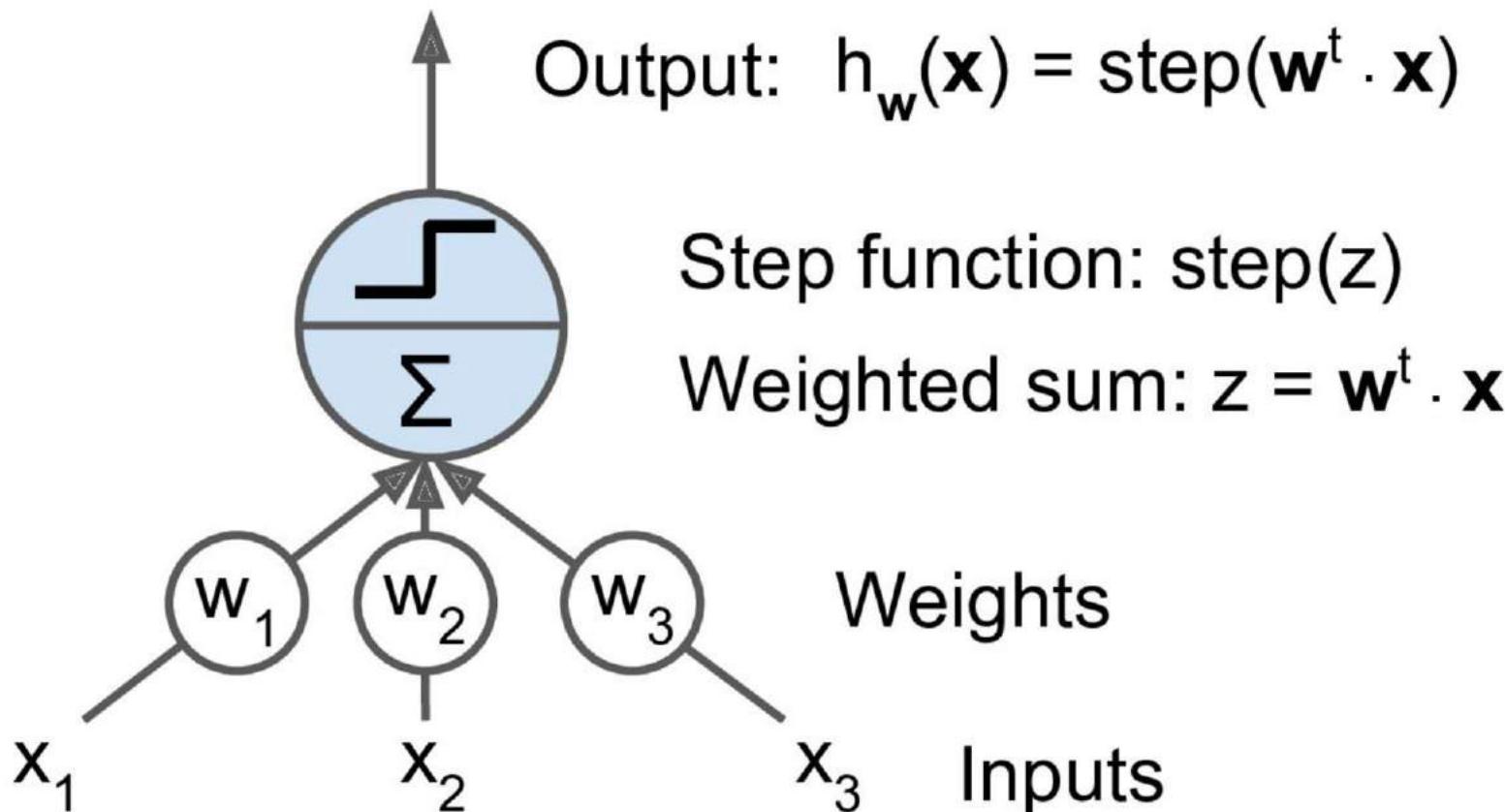


Artificial Neural Networks

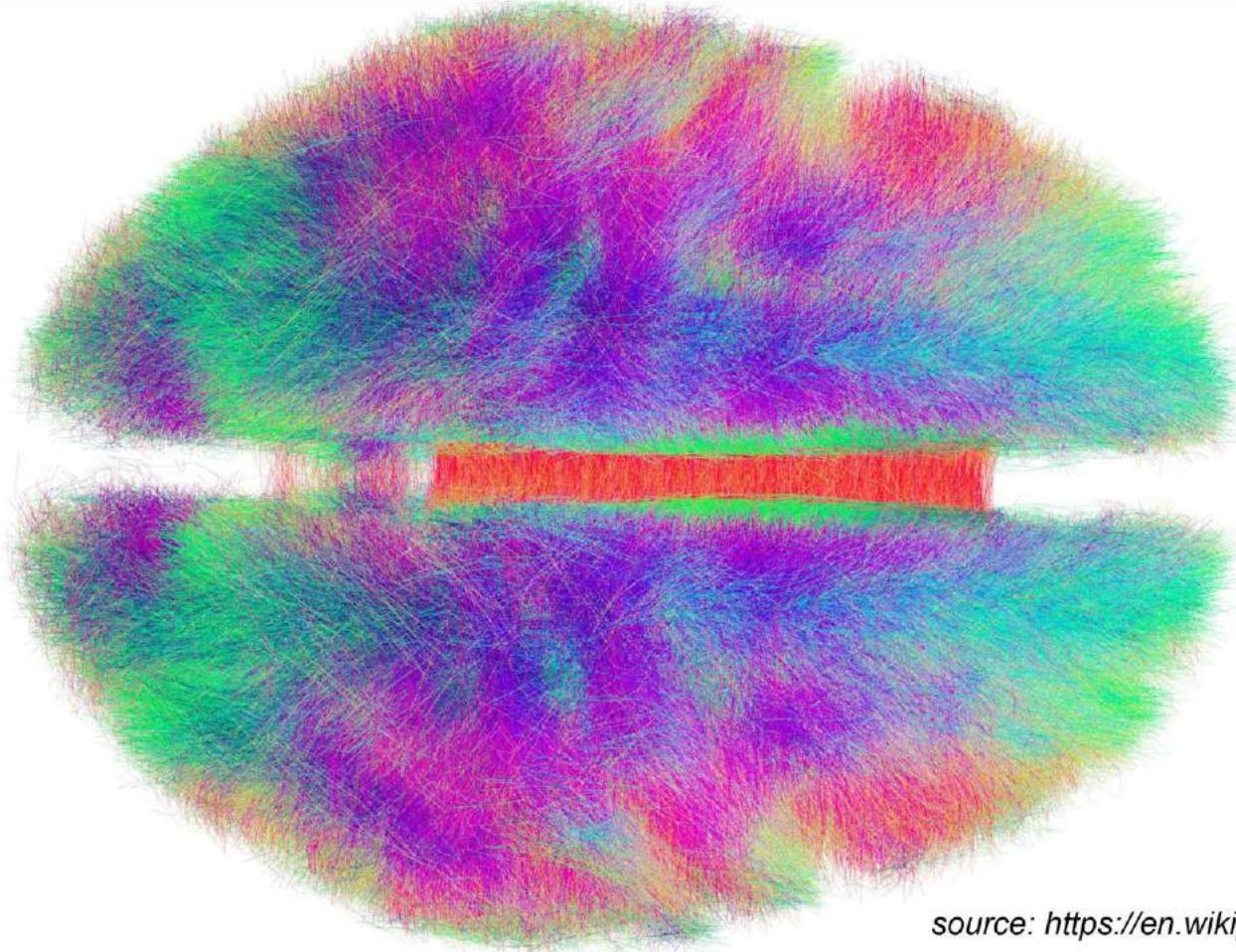
Biological Neuron



Linear Threshold Unit (LTU)

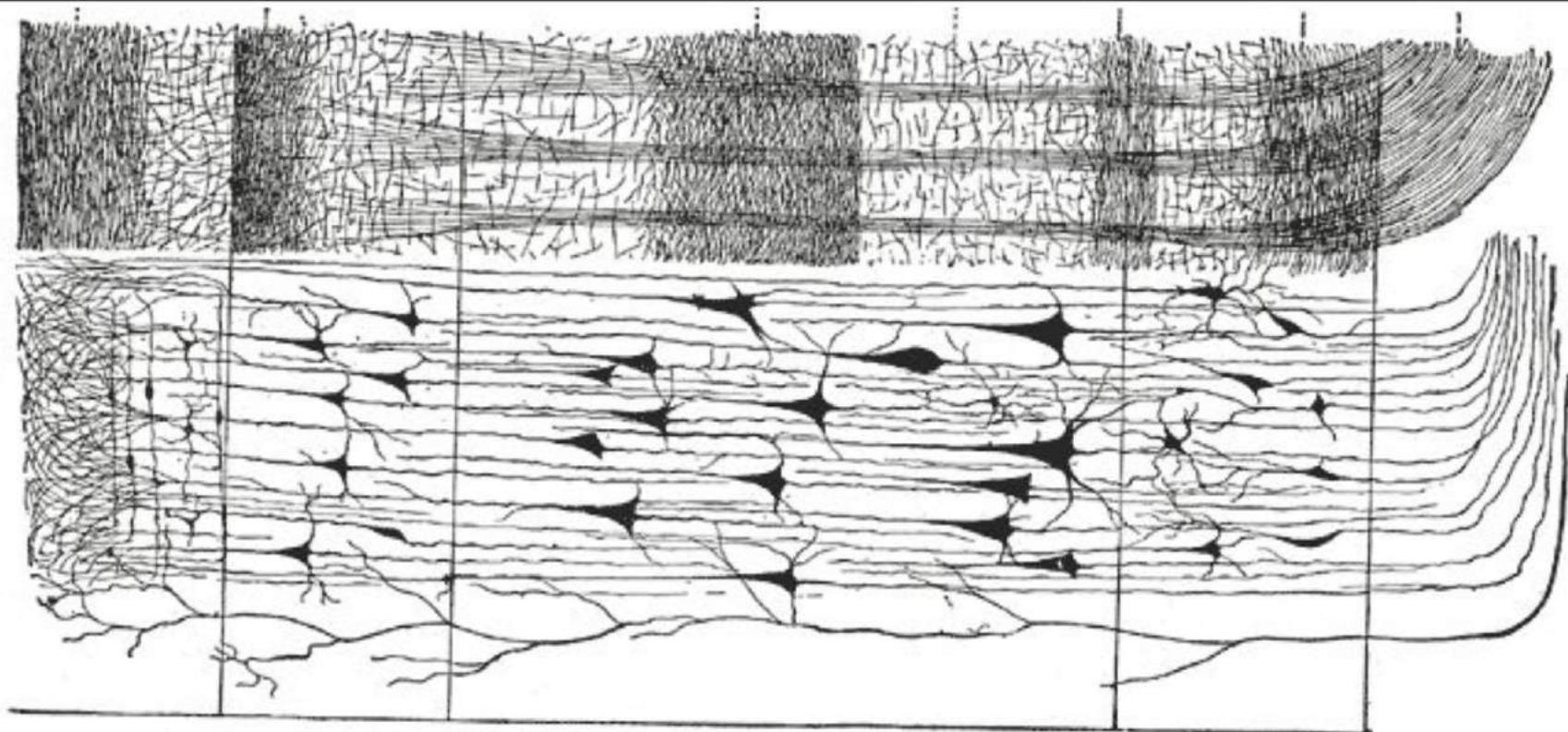


Connectome



source: <https://en.wikipedia.org/wiki/Connectome>

Layers in the Human Cortex



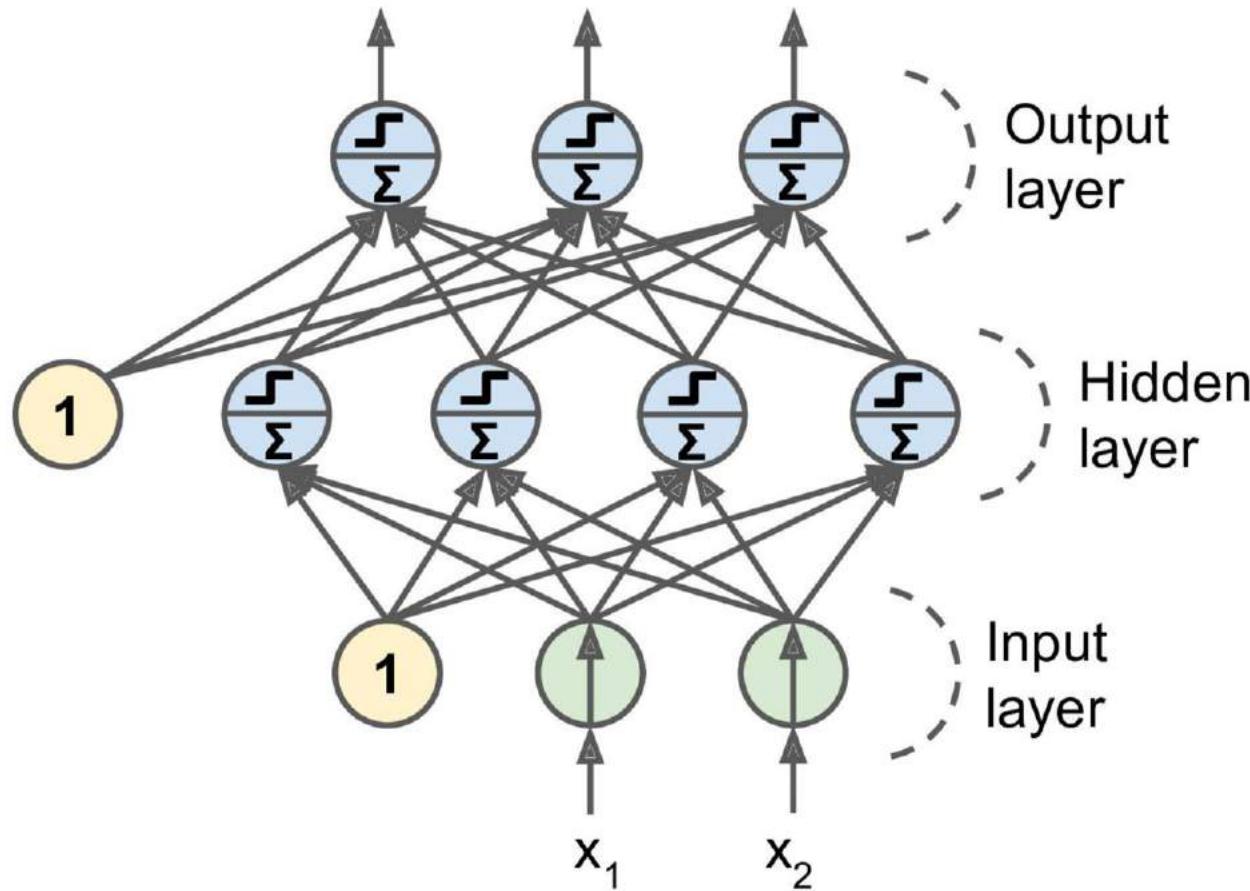
Molecular
layer

Layer of small
pyramidal cells

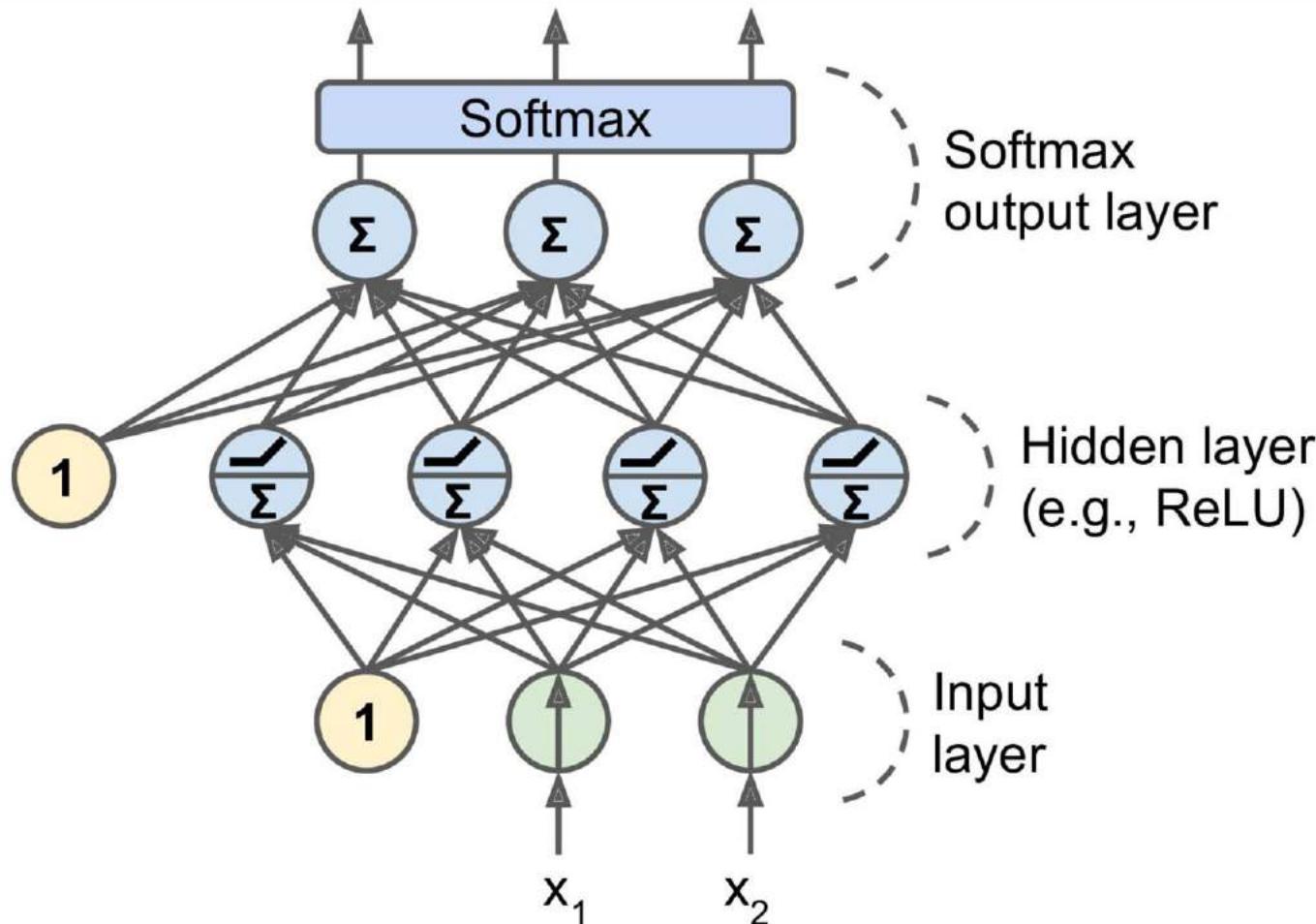
Layer of
pyramidal cells

Layer of poly-
morphous cells

Multi-Layer Perceptron (MLP)



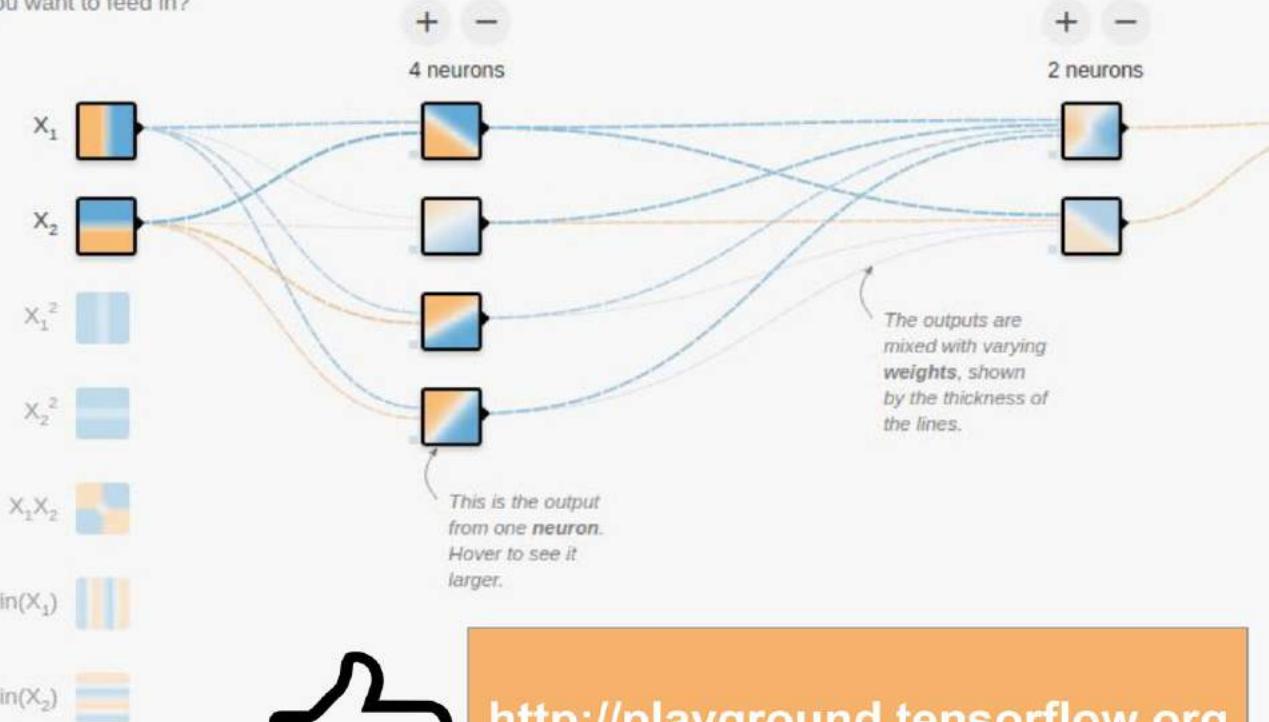
Modern MLP for Classification



Neural Nets > TensorFlow Playground

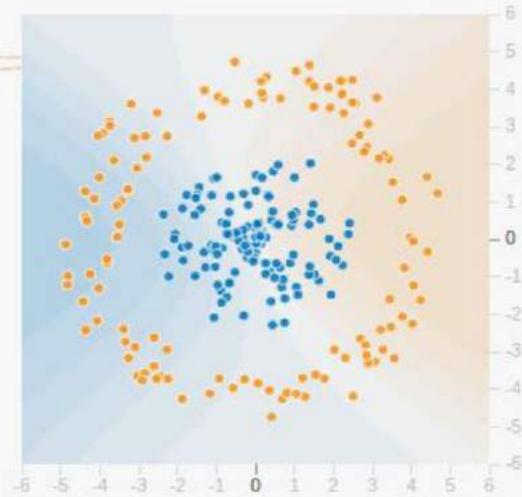
FEATURES

Which properties do you want to feed in?



OUTPUT

Test loss 0.525
Training loss 0.528



Colors shows data, neuron and weight values.

-1 0 1



<http://playground.tensorflow.org>

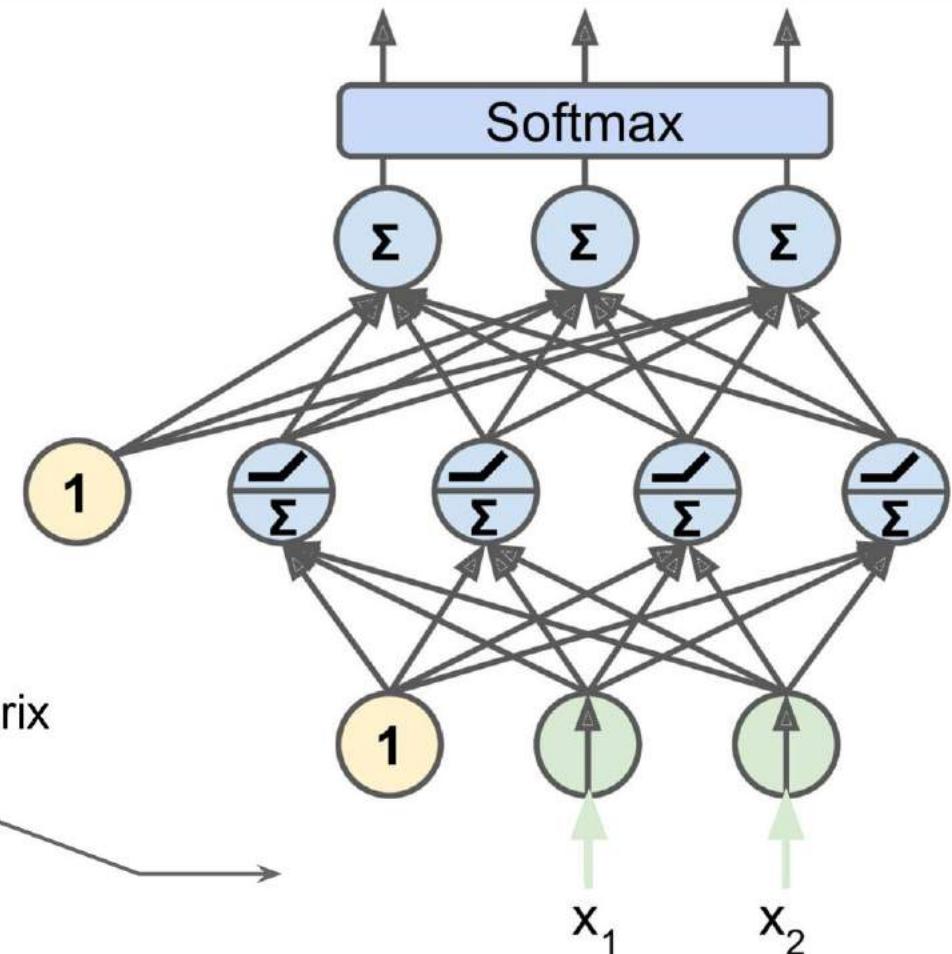
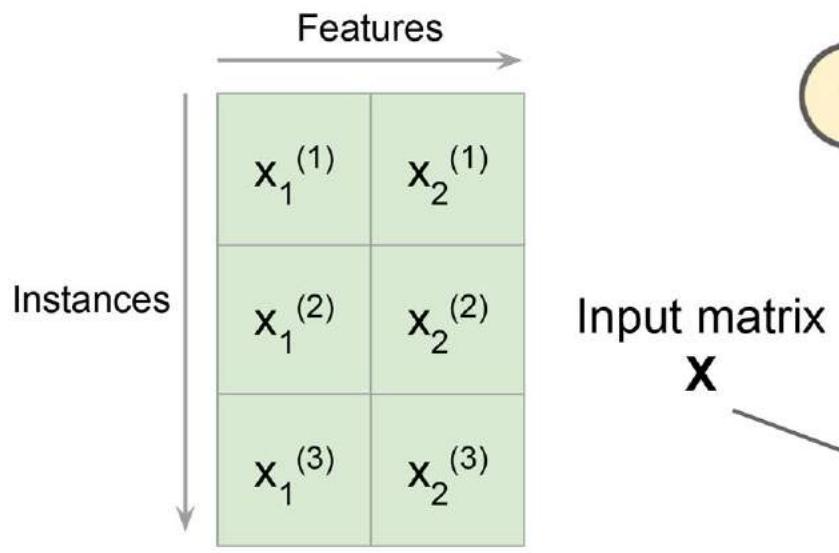
Show test data Discretize output

10 minutes

Exercise 7



Modern MLP > Input Matrix \mathbf{X}

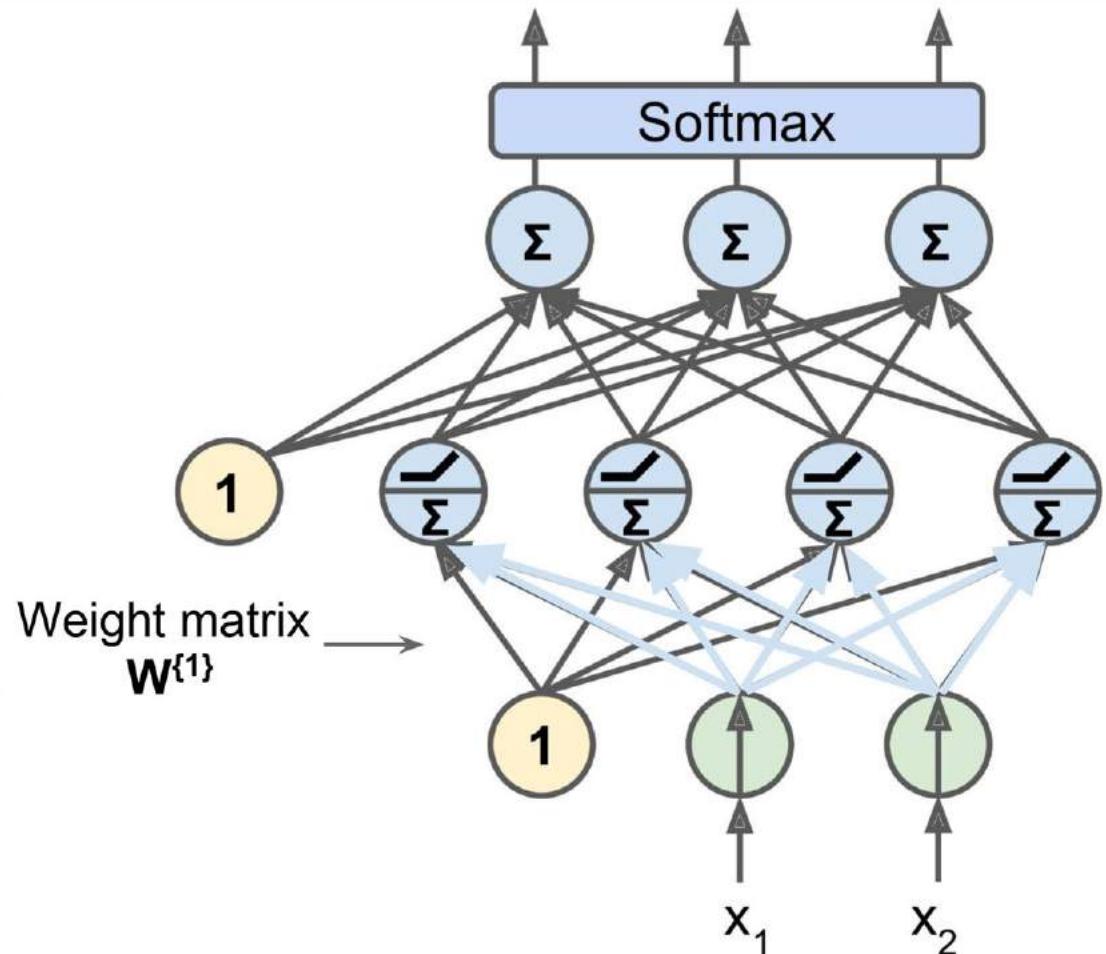


Modern MLP > Weight Matrix $\mathbf{W}^{(1)}$

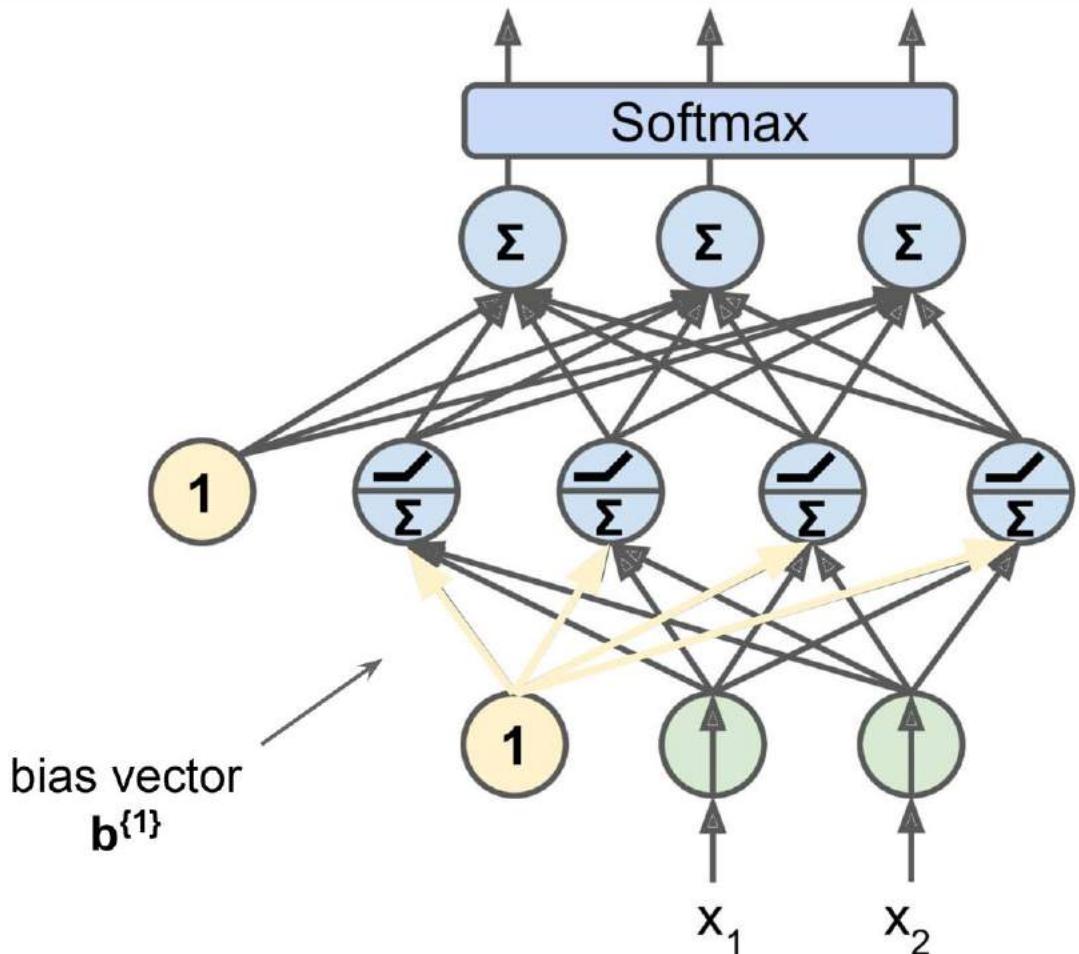
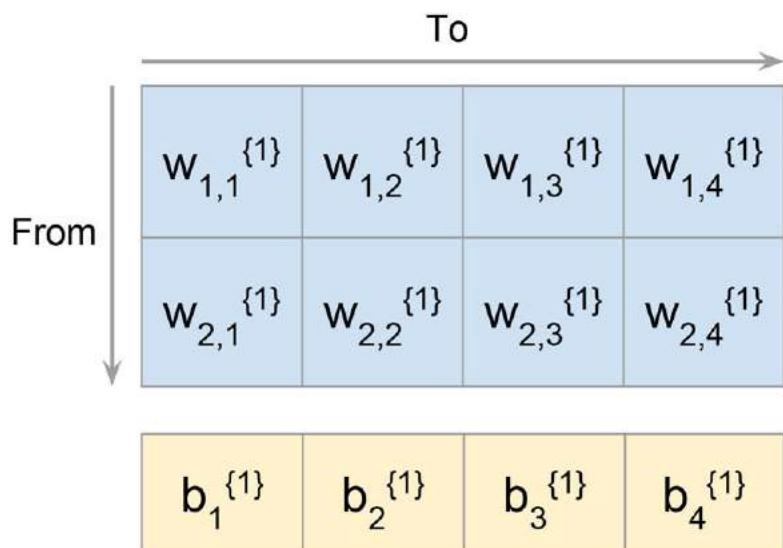
From

To

	$w_{1,1}^{(1)}$	$w_{1,2}^{(1)}$	$w_{1,3}^{(1)}$	$w_{1,4}^{(1)}$
$w_{2,1}^{(1)}$				
$w_{2,2}^{(1)}$				
$w_{2,3}^{(1)}$				
$w_{2,4}^{(1)}$				



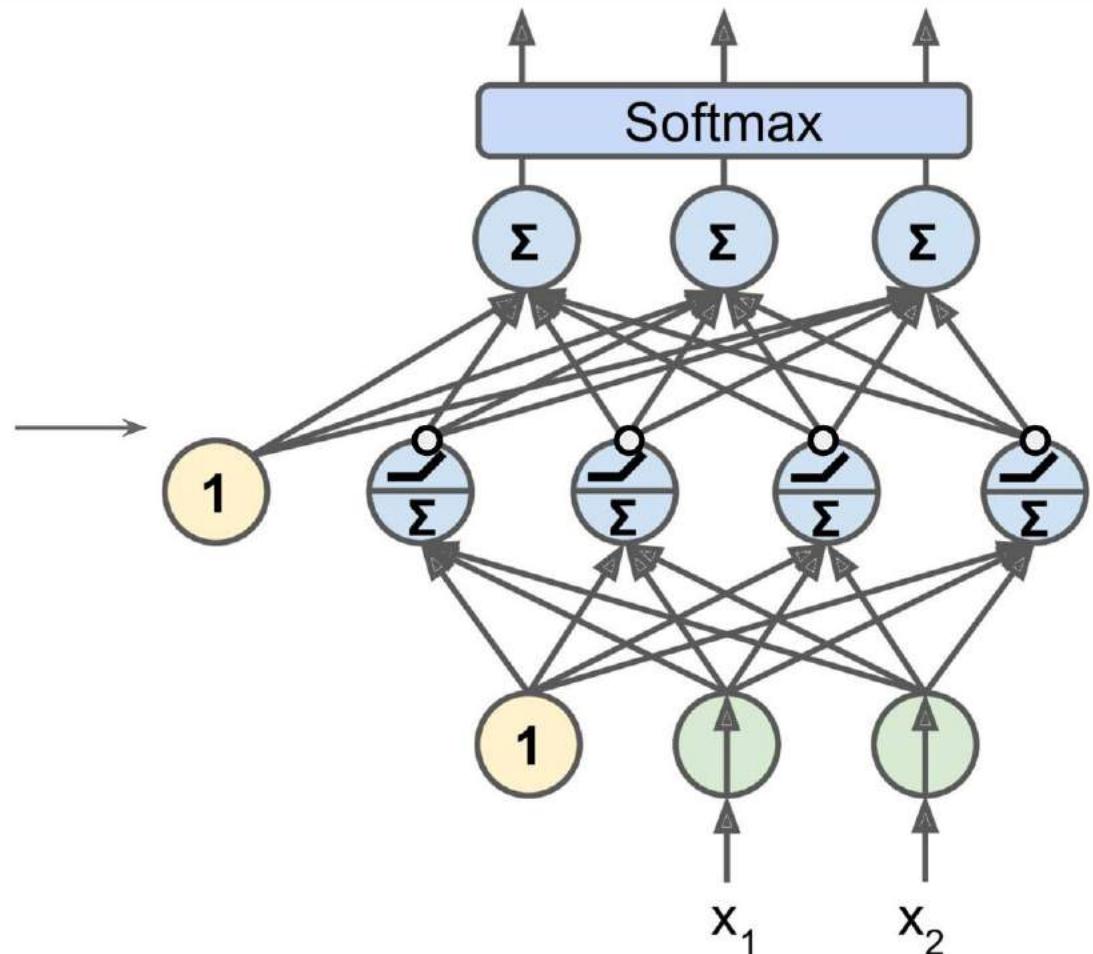
Modern MLP > Bias Vector $\mathbf{b}^{(1)}$



Neural Nets > Modern MLP > Output Matrix $H^{(1)}$

Output matrix of the 1st layer:

$$H^{(1)} = \text{ReLU}(X \cdot W^{(1)} + b^{(1)})$$

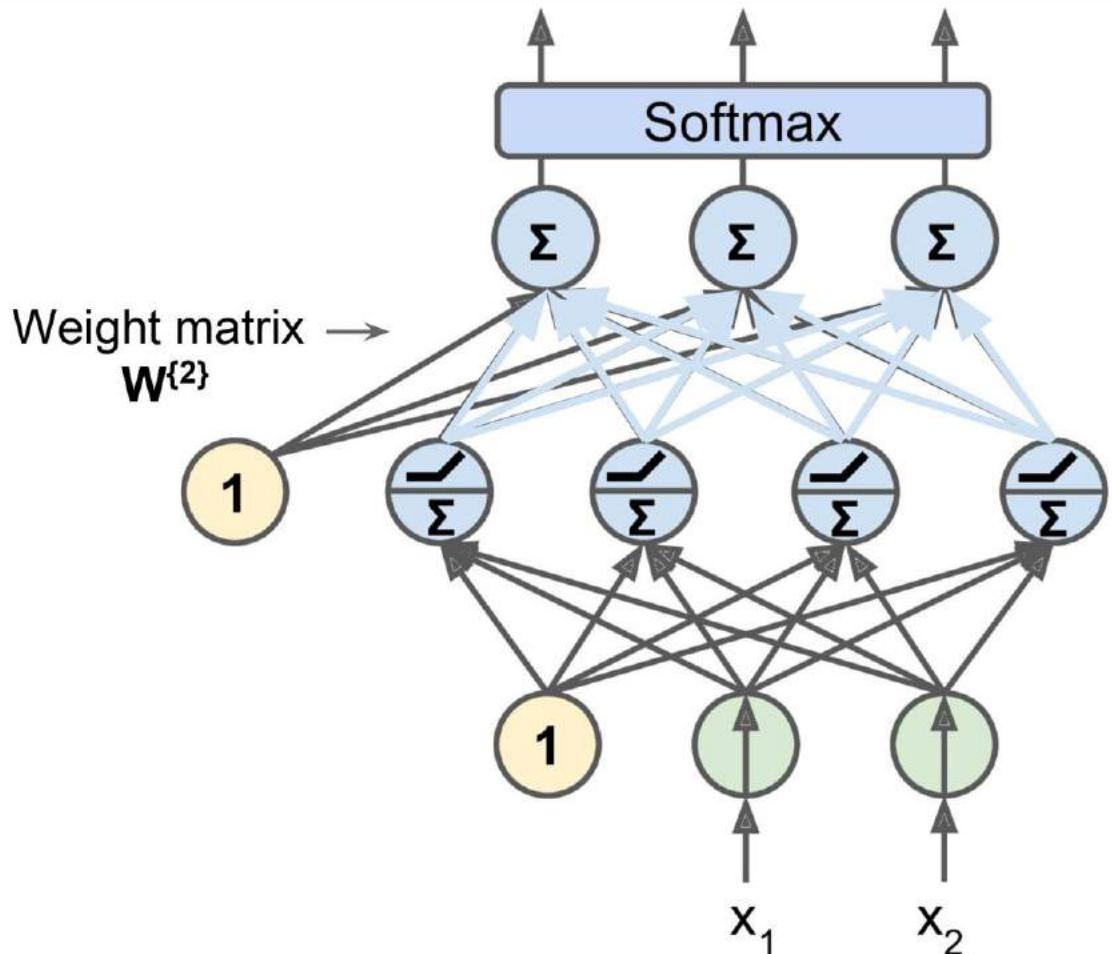


Modern MLP > Weight Matrix $\mathbf{W}^{(2)}$

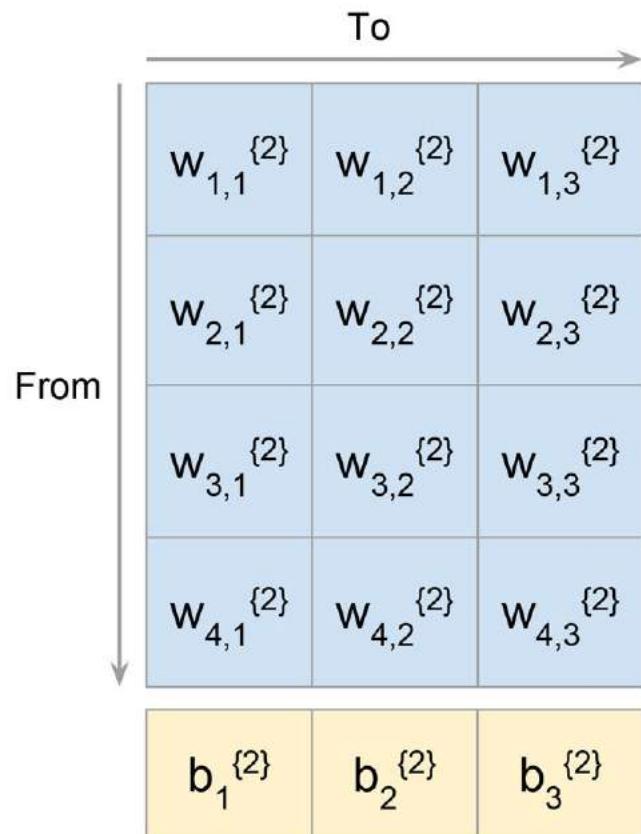
To

$w_{1,1}^{(2)}$	$w_{1,2}^{(2)}$	$w_{1,3}^{(2)}$
$w_{2,1}^{(2)}$	$w_{2,2}^{(2)}$	$w_{2,3}^{(2)}$
$w_{3,1}^{(2)}$	$w_{3,2}^{(2)}$	$w_{3,3}^{(2)}$
$w_{4,1}^{(2)}$	$w_{4,2}^{(2)}$	$w_{4,3}^{(2)}$

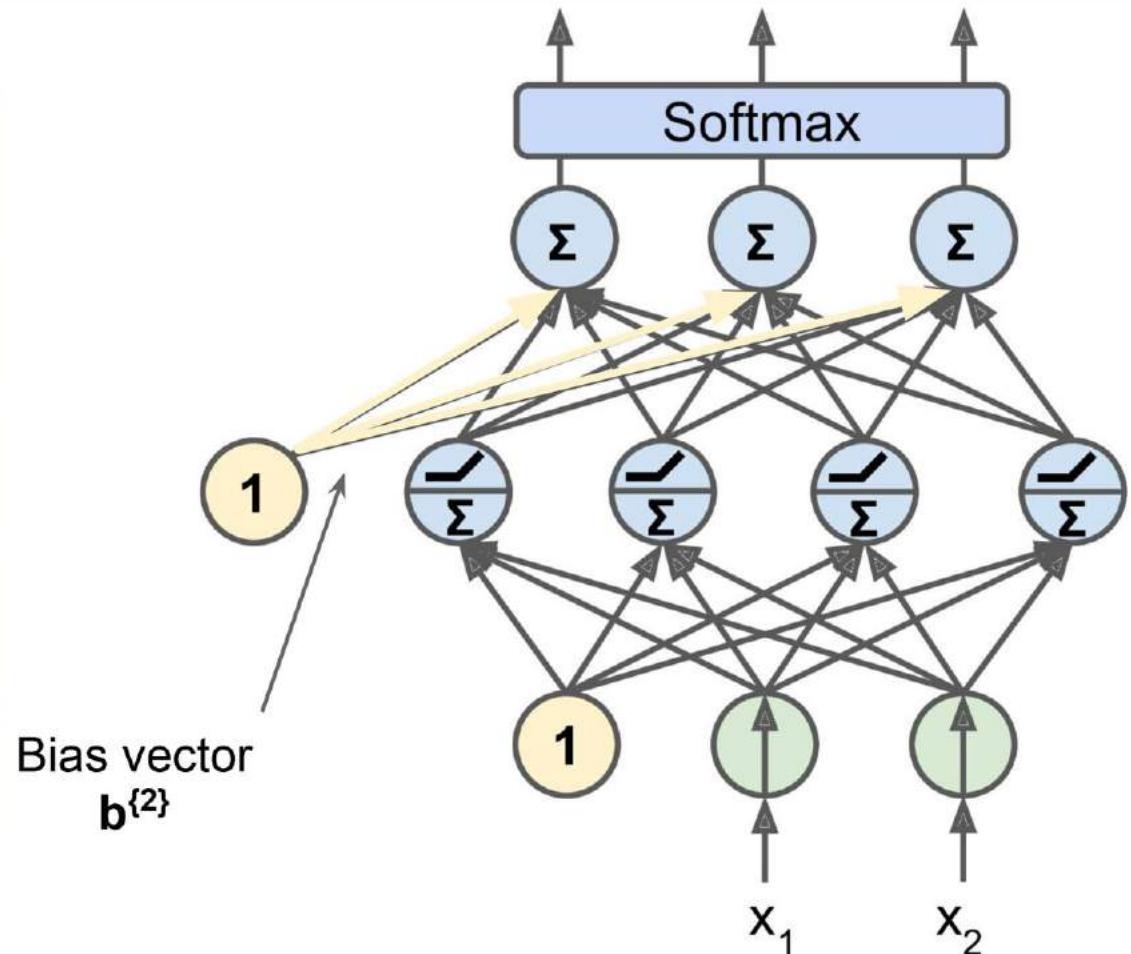
From



Modern MLP > Bias Vector $\mathbf{b}^{\{2\}}$



Bias vector
 $\mathbf{b}^{\{2\}}$



Modern MLP > Prediction Matrix \hat{Y}

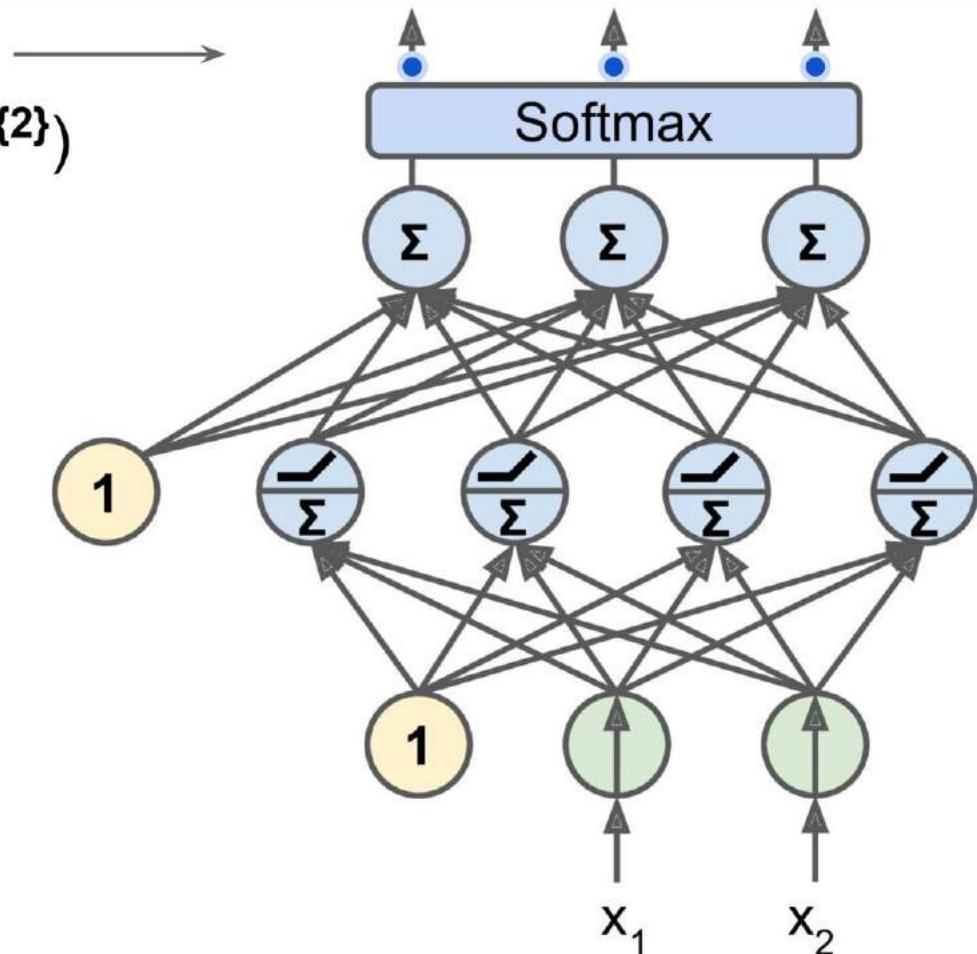
Prediction matrix:

$$\hat{Y} = \text{softmax}(\mathbf{H}^{\{1\}} \cdot \mathbf{W}^{\{2\}} + \mathbf{b}^{\{2\}})$$

Instances ↓

	$\hat{y}_1^{(1)}$	$\hat{y}_2^{(1)}$	$\hat{y}_3^{(1)}$
	$\hat{y}_1^{(2)}$	$\hat{y}_2^{(2)}$	$\hat{y}_3^{(2)}$
	$\hat{y}_1^{(3)}$	$\hat{y}_2^{(3)}$	$\hat{y}_3^{(3)}$
	$\hat{y}_1^{(4)}$	$\hat{y}_2^{(4)}$	$\hat{y}_3^{(4)}$

Class →



Modern MLP > Target Matrix \mathbf{Y}

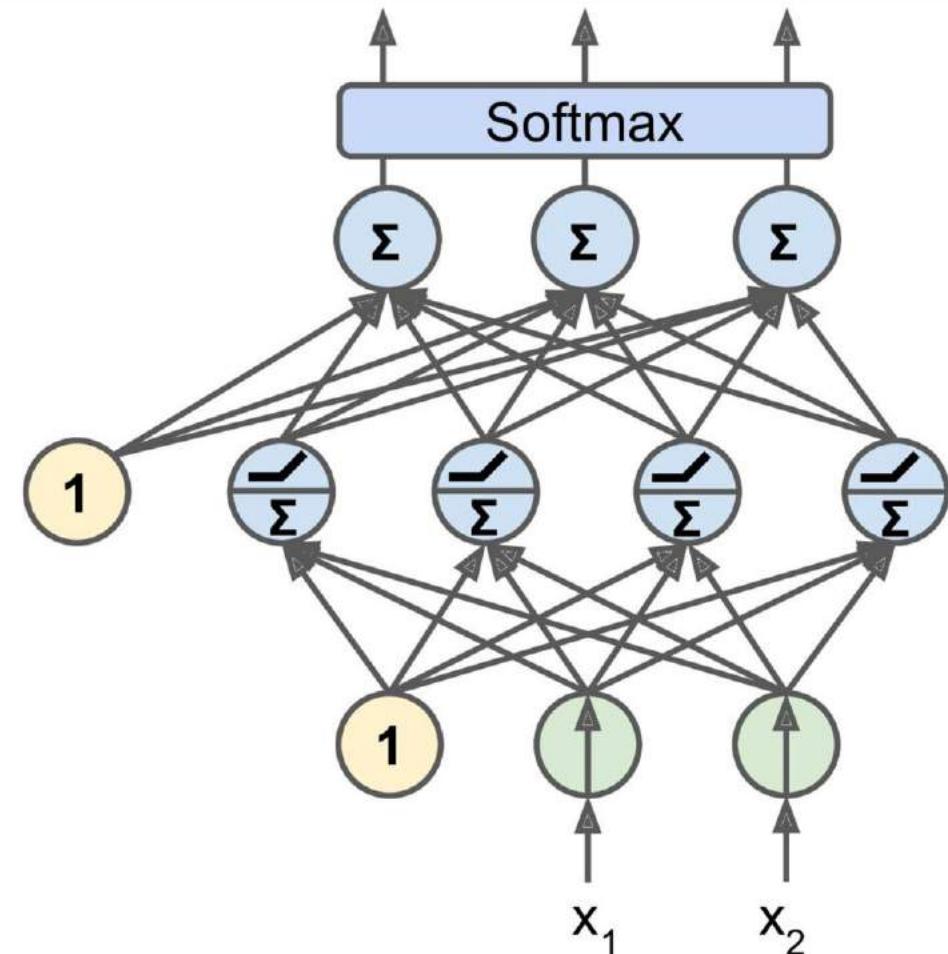
Target matrix:

\mathbf{Y}

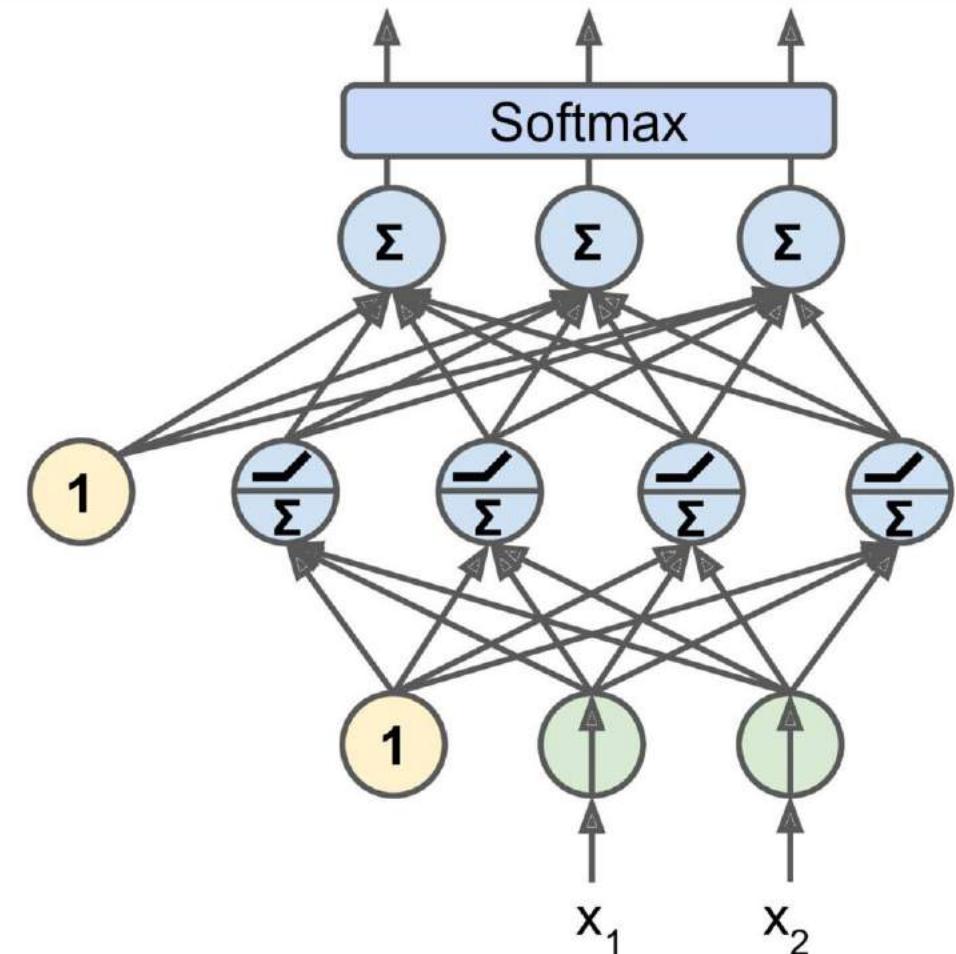
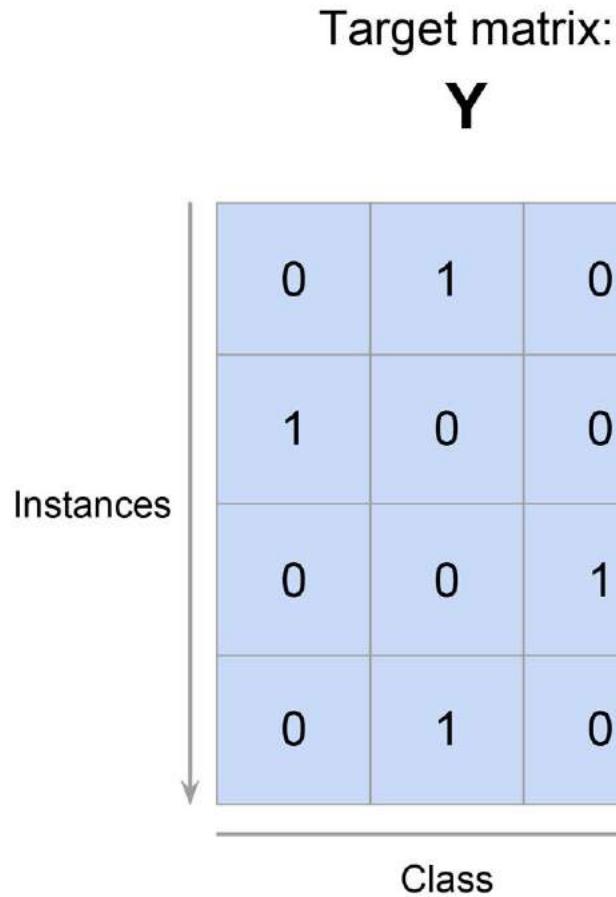
Instances

$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$
$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$
$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$
$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$

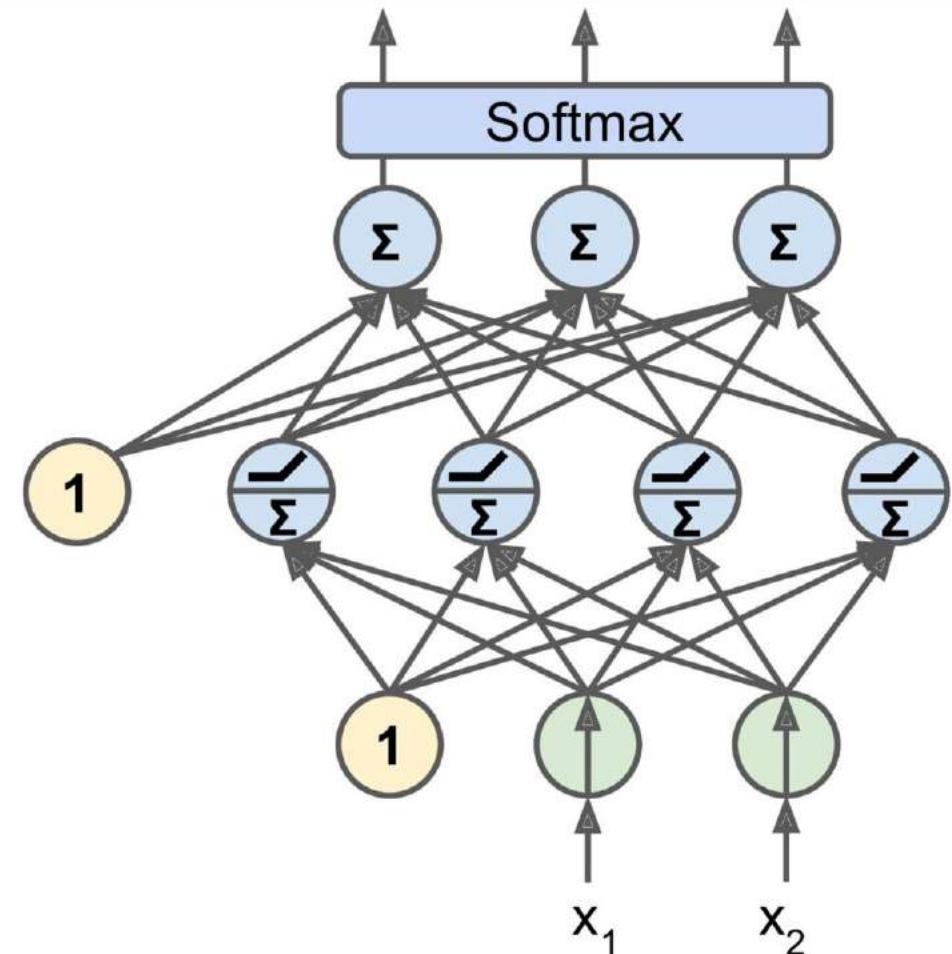
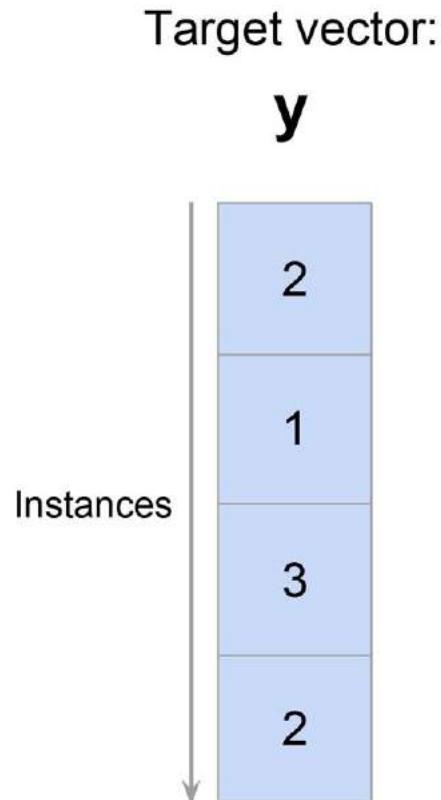
Class



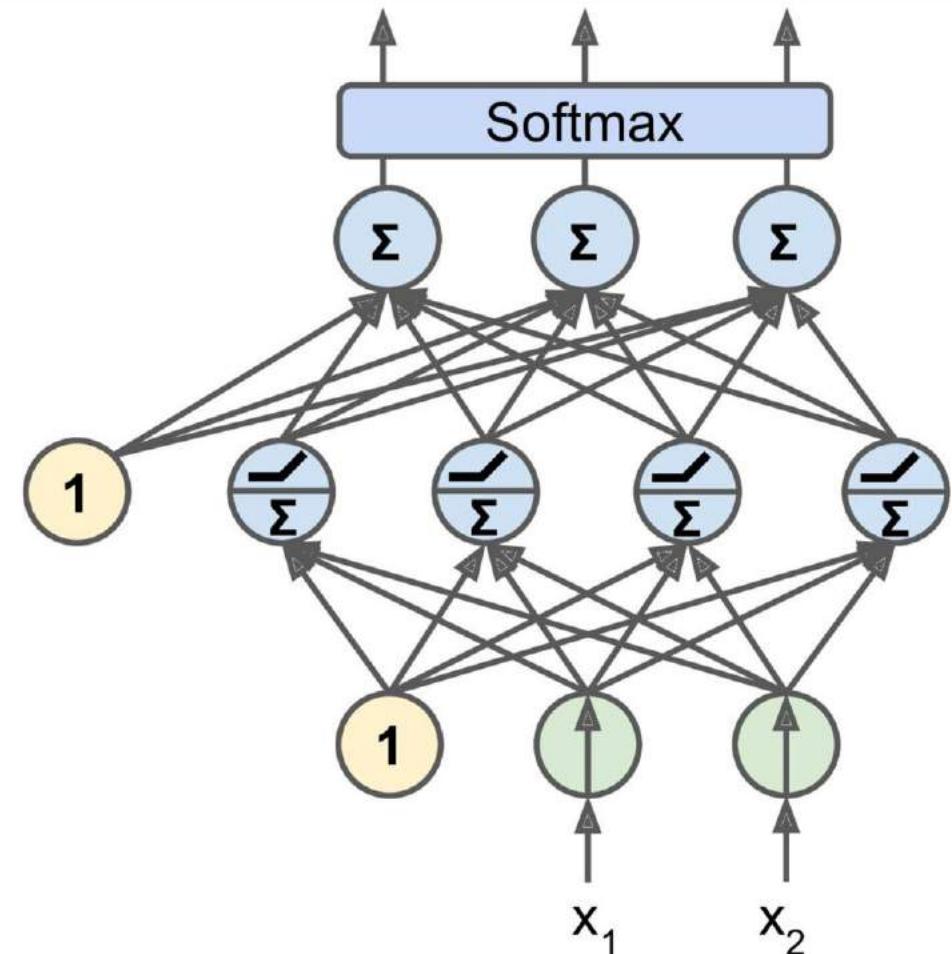
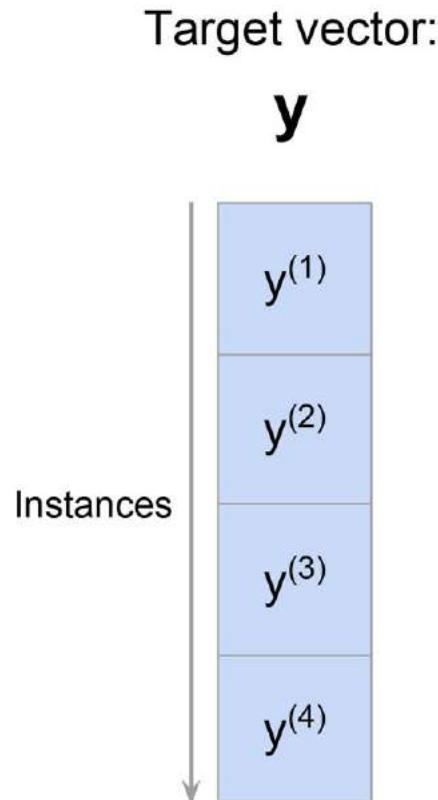
Modern MLP > Target Matrix \mathbf{Y}

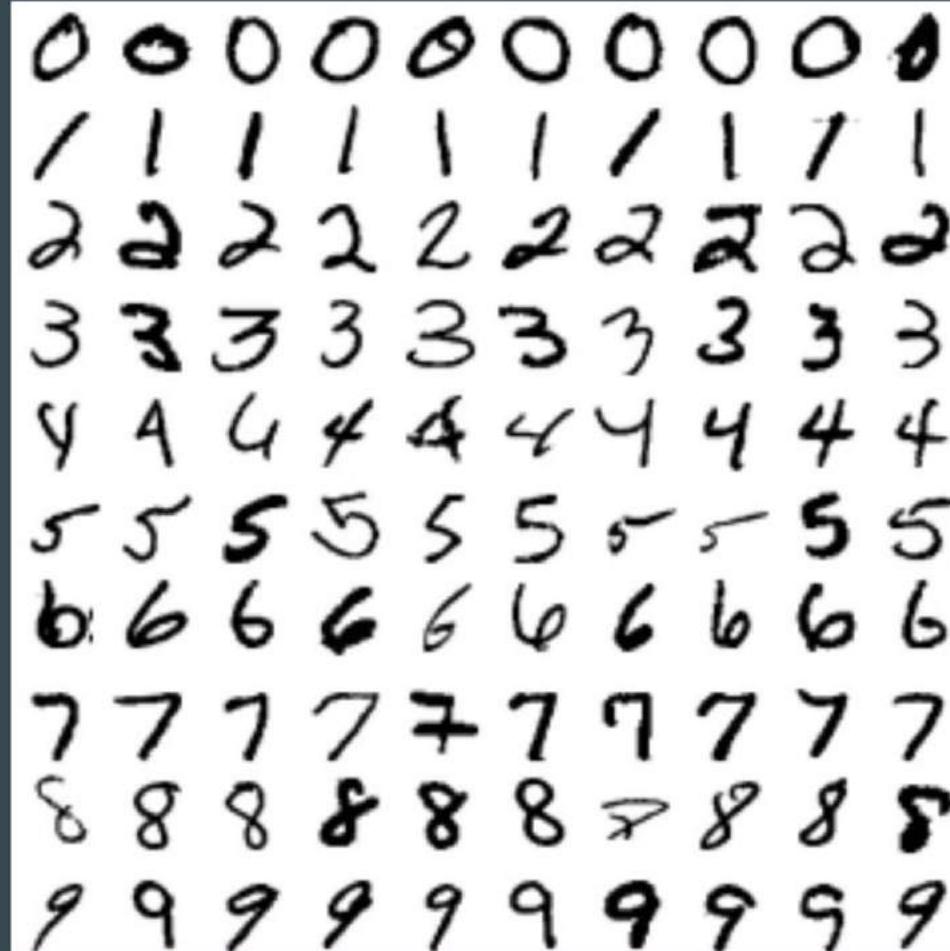


Modern MLP > Target Vector y



Modern MLP > Target Vector y





Neural Nets > **MNIST** > Load the MNIST Dataset

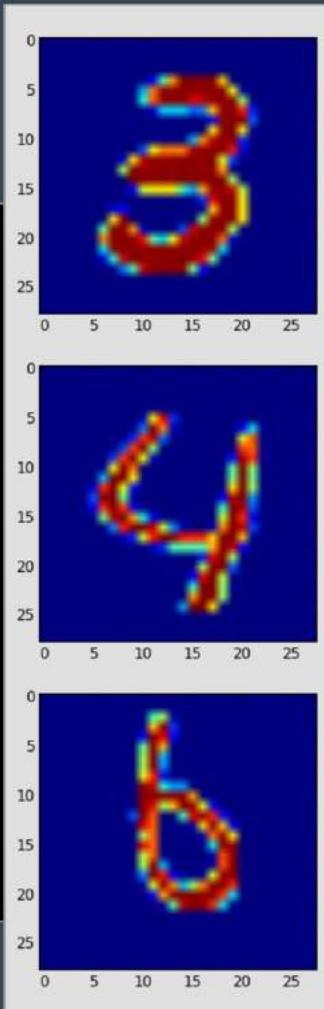
```
>>> from tensorflow.examples.tutorials.mnist import input_data  
>>> mnist = input_data.read_data_sets("/tmp/data/")  
Extracting /tmp/data/train-images-idx3-ubyte.gz  
Extracting /tmp/data/train-labels-idx1-ubyte.gz  
Extracting /tmp/data/t10k-images-idx3-ubyte.gz  
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

Neural Nets > MNIST > Load the MNIST Dataset

```
>>> from tensorflow.examples.tutorials.mnist import input_data  
>>> mnist = input_data.read_data_sets("/tmp/data/")  
Extracting /tmp/data/train-images-idx3-ubyte.gz  
Extracting /tmp/data/train-labels-idx1-ubyte.gz  
Extracting /tmp/data/t10k-images-idx3-ubyte.gz  
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz  
>>> x_batch, y_batch = mnist.train.next_batch(batch_size=3)  
>>> x_batch.shape  
(3, 784)  
>>> y_batch.shape  
(3, )
```

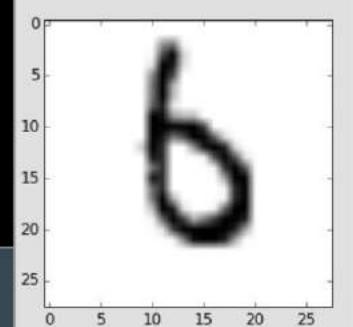
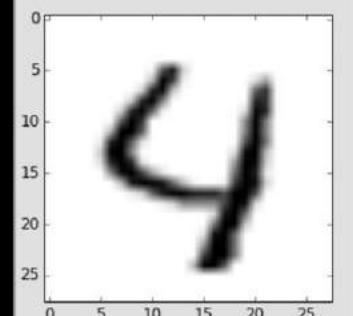
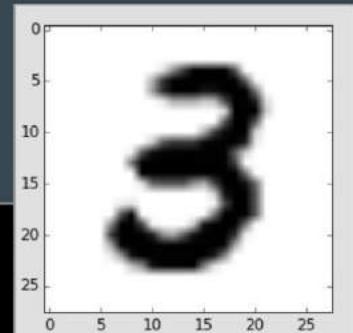
Neural Nets > MNIST > A Peek at the Digits

```
>>> for image_data in X_batch:  
...     plt.imshow(image_data.reshape([28, 28]))  
...     plt.show()
```



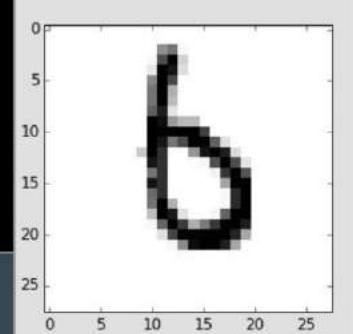
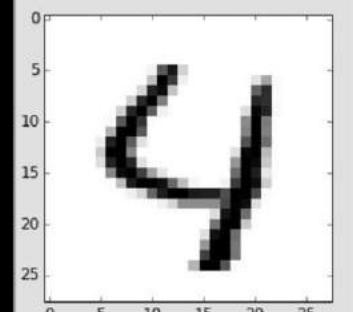
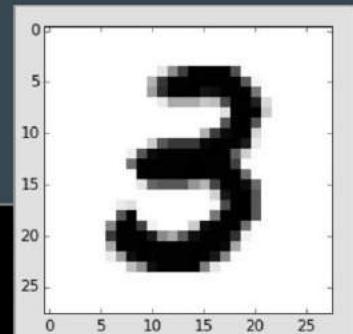
Neural Nets > MNIST > A Peek at the Digits

```
>>> for image_data in X_batch:  
...     plt.imshow(image_data.reshape([28, 28]),  
...                 cmap="binary")  
...     plt.show()
```



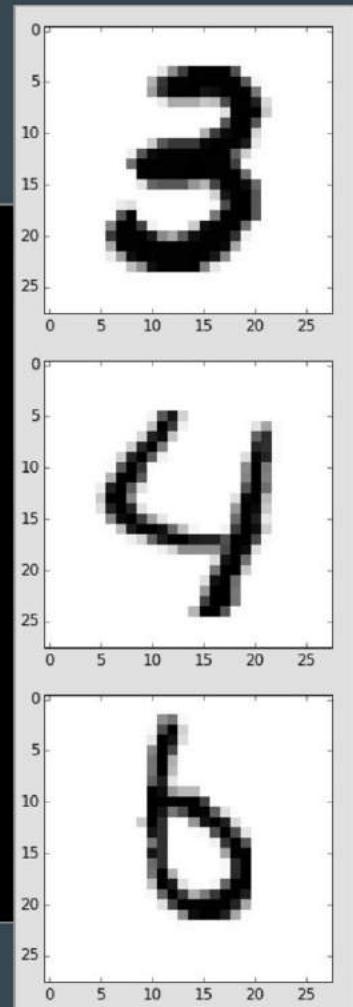
Neural Nets > MNIST > A Peek at the Digits

```
>>> for image_data in X_batch:  
...     plt.imshow(image_data.reshape([28, 28]),  
...                 cmap="binary",  
...                 interpolation="nearest")  
...  
...     plt.show()
```



Neural Nets > MNIST > A Peek at the Digits

```
>>> for image_data in X_batch:  
...     plt.imshow(image_data.reshape([28, 28]),  
...                 cmap="binary",  
...                 interpolation="nearest")  
...     plt.show()  
>>> y_batch  
array([3, 4, 6], dtype=uint8)
```



Neural Nets > Build the Neural Network > Constants

```
>>> n_inputs = 28*28
>>> n_hidden1 = 100
>>> n_outputs = 10
```

Neural Nets > Build the Neural Network > Inputs

```
>>> n_inputs = 28*28
>>> n_hidden1 = 100
>>> n_outputs = 10
>>>
>>> graph = tf.Graph()
>>> with graph.as_default():
...     with tf.name_scope("inputs"):
...         x = tf.placeholder(tf.float32, shape=[None, n_inputs],
...                           name="X")
...
...         y = tf.placeholder(tf.int32, shape=[None], name="y")
```

Neural Nets > Build the Neural Network > Hidden Layer

```
>>> with graph.as_default():
...     with tf.name_scope("hidden1"):
...         b1 = tf.Variable(tf.zeros([n_hidden1]), name="b1")
...         W1 = tf.Variable(
...             tf.random_uniform([n_inputs, n_hidden1], -1.0, 1.0),
...             name="W1")
...         hidden1 = tf.nn.relu(tf.matmul(X, W1) + b1)
```

Neural Nets > Build the Neural Network > Output Layer

```
>>> with graph.as_default():
...     with tf.name_scope("output"):
...         b2 = tf.Variable(tf.zeros([n_outputs]), name="b2")
...         W2 = tf.Variable(
...             tf.random_uniform([n_hidden1, n_outputs], -1.0, 1.0),
...             name="W2")
...         logits = tf.matmul(hidden1, W2) + b2
...         Y_proba = tf.nn.softmax(logits, name="Y_proba")
...         y_pred = tf.argmax(Y_proba, axis=1)
```

Neural Nets > Build the Neural Network > Training ops

```
>>> with graph.as_default():
...     with tf.name_scope("train"):
...         xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
...             logits=logits, labels=y)
...         loss = tf.reduce_mean(xentropy)
...         optimizer = tf.train.AdamOptimizer()
...         training_op = optimizer.minimize(loss)
```

Neural Nets > Build the Neural Network > Training ops

```
>>> with graph.as_default():
...     with tf.name_scope("train"):
...         xentropy = tf.nn.sparse softmax cross entropy with logits(
...             logits=logits, labels=y)
...         loss = tf.reduce_mean(xentropy)
...         optimizer = tf.train.AdamOptimizer()
...         training_op = optimizer.minimize(loss)
```

$$\hat{p}_i = \text{Softmax}(\mathbf{z})_i = \exp(z_i) / \sum_j \exp(z_j)$$

$$\text{Cross-Entropy}(\mathbf{p}, \hat{\mathbf{p}}) = \sum_i p_i \log(\hat{p}_i)$$

Neural Nets > Build the Neural Network > Evaluation

```
>>> with graph.as_default():
...     with tf.name_scope("eval"):
...         correct = tf.equal(y_pred, y)
...         accuracy = tf.reduce_mean(tf.to_float(correct))
```

Neural Nets > Build the Neural Network > Evaluation

```
>>> with graph.as_default():
...     with tf.name_scope("eval"):
...         correct = tf.equal(y_pred, y)
...         accuracy = tf.reduce_mean(tf.to_float(correct))
```



tf.metrics = streaming metrics

Neural Nets > Build the Neural Network > Init & Saver

```
>>> with graph.as_default():
...     with tf.name_scope("init_and_save"):
...         init = tf.global_variables_initializer()
...         saver = tf.train.Saver()
```

12 minutes

Exercise 8



Train the Neural Network

```
>>> n_epochs = 20
>>> batch_size = 50
>>> n_batches_per_epoch = mnist.train.num_examples // batch_size
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     for epoch in range(n_epochs):
...         for iteration in range(n_batches_per_epoch):
...             x_batch, y_batch = mnist.train.next_batch(batch_size)
...             sess.run(training_op,
...                     feed_dict={X: x_batch, y: y_batch})
...     save_path = saver.save(sess, "./my_mnist_model")
```

Neural Nets > Load the Model

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     saver = tf.train.import_meta_graph("./my_mnist_model.meta")
...
>>> with tf.Session(graph=graph) as sess:
...     saver.restore(sess, "./my_mnist_model")
...     X = graph.get_tensor_by_name("inputs/X:0")
...     Y_proba = graph.get_tensor_by_name("output/Y_proba:0")
```

Load the Model & Make Predictions

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     saver = tf.train.import_meta_graph("./my_mnist_model.meta")
...
>>> with tf.Session(graph=graph) as sess:
...     saver.restore(sess, "./my_mnist_model")
...     X = graph.get_tensor_by_name("inputs/X:0")
...     Y_proba = graph.get_tensor_by_name("output/Y_proba:0")
...     Y_proba_val = Y_proba.eval(feed_dict={X: mnist.test.images})
```

Visualize the Errors

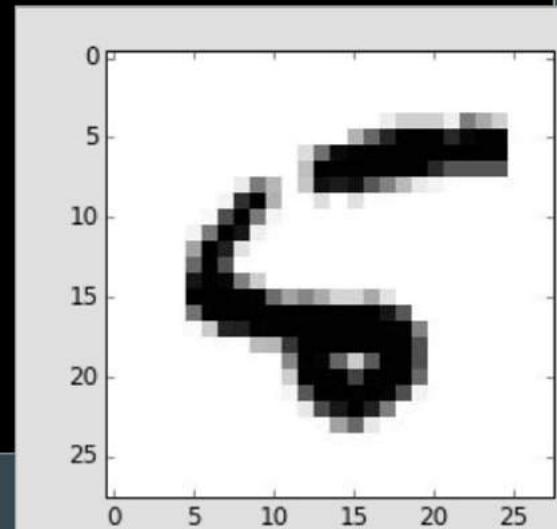
```
>>> for example_index in range(200):
...     y_proba = Y_proba_val[example_index]
...     y_pred = np.argmax(y_proba)
...     y_label = mnist.test.labels[example_index]
...     if y_pred != y_label:
...         plt.imshow(...)
...         plt.show()
```

Neural Nets >

Visualize the Errors

```
>>> for example_index in range(200):
...     y_proba = Y_proba_val[example_index]
...     y_pred = np.argmax(y_proba)
...     y_label = mnist.test.labels[example_index]
...     if y_pred != y_label:
...         plt.imshow(...)
...         plt.show()
```

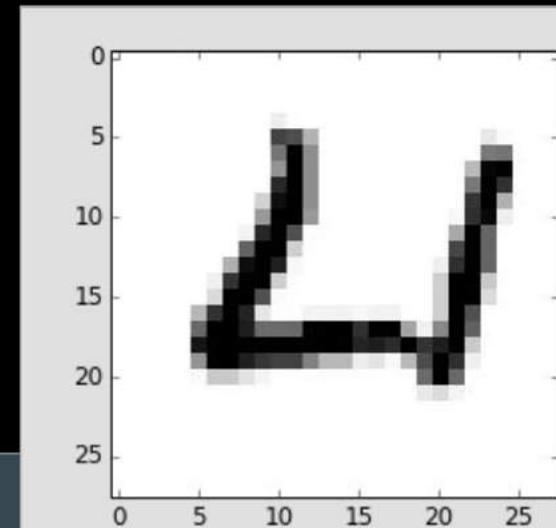
Largest predicted probabilities: 5=4.9% 6=94.9%
⇒ Actually a 5



Visualize the Errors

```
>>> for example_index in range(200):  
...     y_proba = Y_proba_val[example_index]  
...     y_pred = np.argmax(y_proba)  
...     y_label = mnist.test.labels[example_index]  
...     if y_pred != y_label:  
...         plt.imshow(...)  
...         plt.show()
```

Largest predicted probabilities: **0=100%**
⇒ Actually a 4

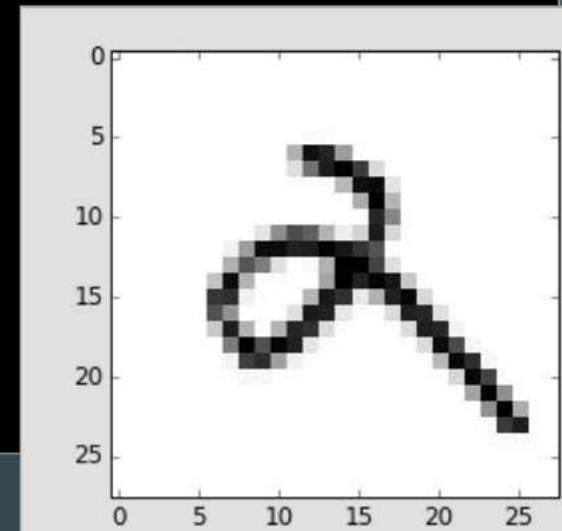


Neural Nets >

Visualize the Errors

```
>>> for example_index in range(200):  
...     y_proba = Y_proba_val[example_index]  
...     y_pred = np.argmax(y_proba)  
...     y_label = mnist.test.labels[example_index]  
...     if y_pred != y_label:  
...         plt.imshow(...)  
...         plt.show()
```

Largest predicted probabilities: 2=30.3% **4=54.8%** 9=14.7%
⇒ Actually a 2



Organizing Your Code

Organizing Your Code > Simplifying Your Code > Before

```
>>> with graph.as_default():
...     with tf.name_scope("hidden1"):
...         b1 = tf.Variable(tf.zeros([n_hidden1]), name="b1")
...         W1 = tf.Variable(
...             tf.random_uniform([n_inputs, n_hidden1], -1.0, 1.0),
...             name="W1")
...         hidden1 = tf.nn.relu(tf.matmul(X, W1) + b1)
```

Building a Reusable Model Component

```
>>> def neural_net_layer(inputs, n_neurons, activation=None):
...     n_inputs = int(inputs.get_shape()[1])
...     b = tf.Variable(tf.zeros([n_neurons], name="b"))
...     W = tf.Variable(
...         tf.random_uniform([n_inputs, n_neurons], -1.0, 1.0),
...         name="W")
...     logits = tf.matmul(inputs, W) + b
...     if activation:
...         return activation(logits)
...     else:
...         return logits
```

Organizing Your Code > Simplifying Your Code > After

```
>>> with graph.as_default():
...     with tf.name_scope("hidden1"):
...         hidden1 = neural_net_layer(
...             x, n_hidden1, activation=tf.nn.relu)
```

Organizing Your Code > Simplifying Your Code > After

```
>>> with graph.as_default():
...     with tf.name_scope("hidden1"):
...         hidden1 = neural_net_layer(
...             X, n_hidden1, activation=tf.nn.relu)
...     with tf.name_scope("output"):
...         logits = neural_net_layer(hidden1, n_outputs)
...         Y_proba = tf.nn.softmax(logits)
```

Organizing Your Code >

Adding a name Parameter

```
>>> def neural_net_layer(..., name=None):  
...     with tf.name_scope(name, default_name="layer"):  
...         [...]
```

Organizing Your Code > Simplifying Your Code > After

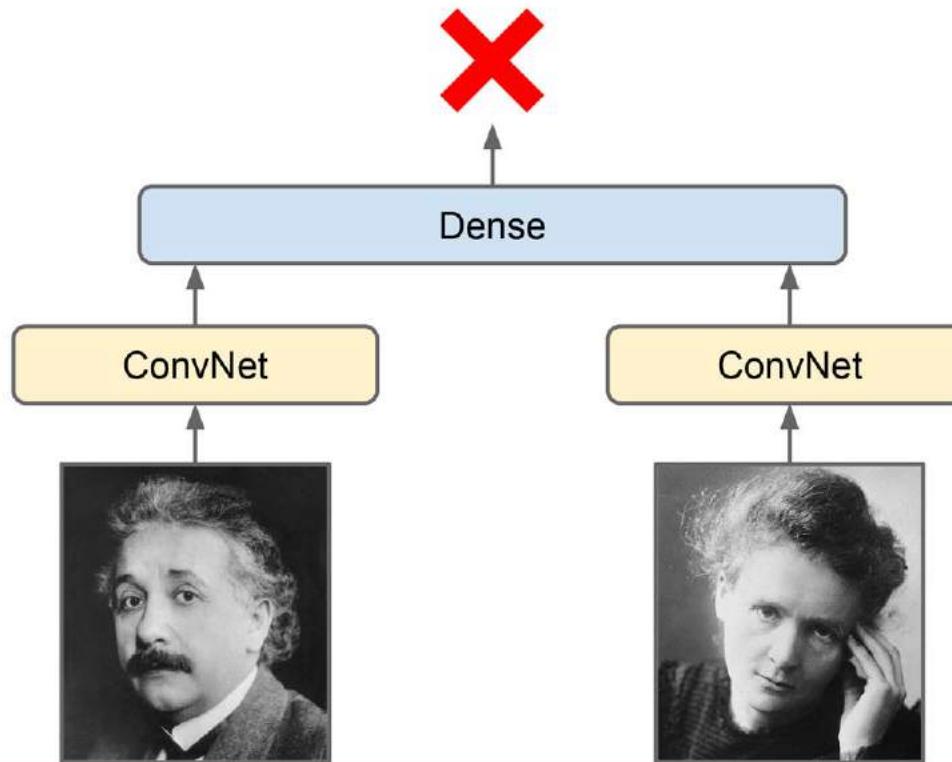
```
>>> with graph.as_default():
...     hidden1 = neural_net_layer(
...         X, n_hidden1, activation=tf.nn.relu, name="hidden1")
...     logits = neural_net_layer(
...         hidden1, n_outputs, name="output")
...     Y_proba = tf.nn.softmax(logits)
```

Organizing Your Code > **Using `tf.layers.dense()` Instead**

```
>>> with graph.as_default():
...     hidden1 = tf.layers.dense(
...         X, n_hidden1, activation=tf.nn.relu, name="hidden1")
...     logits = tf.layers.dense(
...         hidden1, n_outputs, name="output")
...     Y_proba = tf.nn.softmax(logits)
```

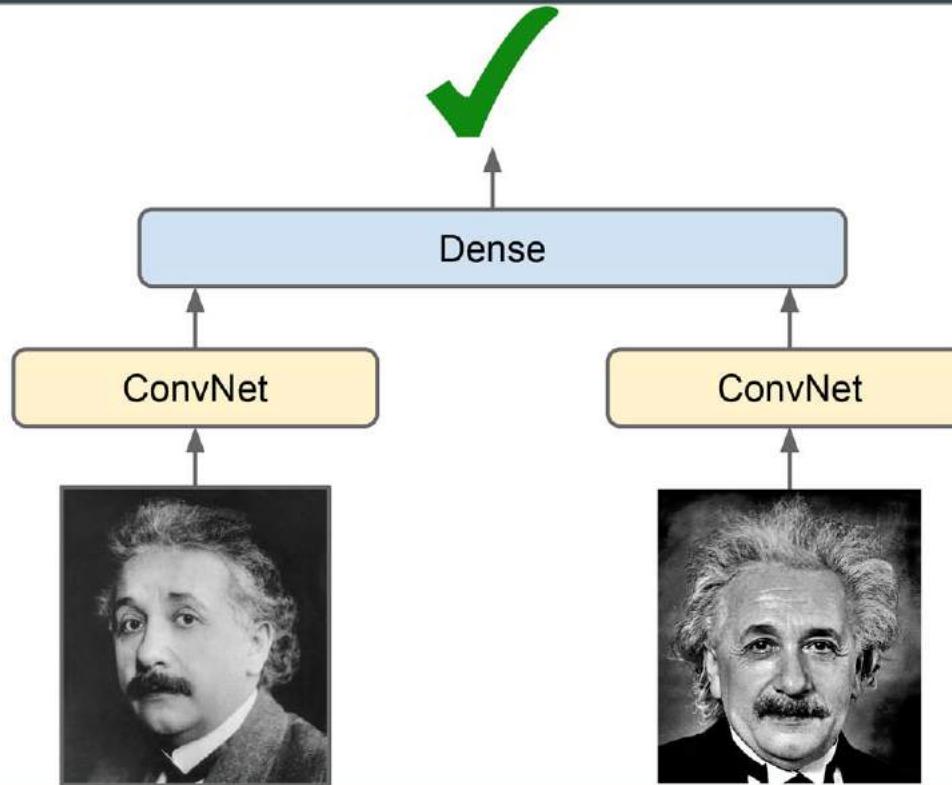
Organizing Your Code >

Sharing Variables

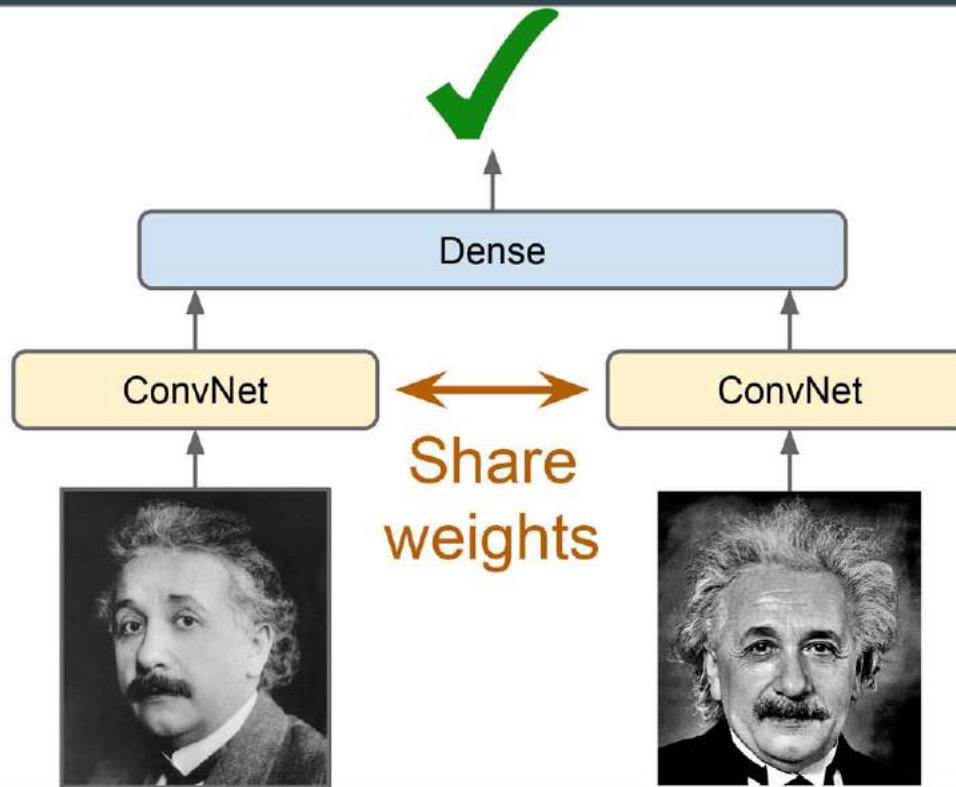


Organizing Your Code >

Sharing Variables



Organizing Your Code > Sharing Variables



Organizing Your Code > Sharing Variables

Organizing Your Code > Using Variable Scopes

Variable Scopes and Name Scopes

```
>>> with tf.variable_scope("foo"):  
...     a = tf.constant(1., name="a")  
...     with tf.name_scope("bar"):  
...         b = tf.constant(2., name="b")  
...         with tf.name_scope("baz"):  
...             c = tf.get_variable("c", ...)  
...             s = tf.add_n([a,b,c], name="s")
```

Organizing Your Code >

Variable Scopes and Name Scopes

```
>>> with tf.variable_scope("foo"):  
...     a = tf.constant(1., name="a")  
...     with tf.name_scope("bar"):  
...         b = tf.constant(2., name="b")  
...         with tf.name_scope("baz"):  
...             c = tf.get_variable("c", ...)  
...             s = tf.add_n([a,b,c], name="s")  
  
>>> a.name  
'foo/a:0'
```

Organizing Your Code >

Variable Scopes and Name Scopes

```
>>> with tf.variable_scope("foo"):  
...     a = tf.constant(1., name="a")  
...     with tf.name_scope("bar"):  
...         b = tf.constant(2., name="b")  
...         with tf.name_scope("baz"):  
...             c = tf.get_variable("c", ...)  
...             s = tf.add_n([a,b,c], name="s")  
  
>>> a.name  
'foo/a:0'  
  
>>> b.name  
'foo/bar/b:0'
```

Organizing Your Code >

Variable Scopes and Name Scopes

```
>>> with tf.variable_scope("foo"):  
...     a = tf.constant(1., name="a")  
...     with tf.name_scope("bar"):  
...         b = tf.constant(2., name="b")  
...         with tf.name_scope("baz"):  
...             c = tf.get_variable("c", ...)  
...             s = tf.add_n([a,b,c], name="s")  
  
>>> c.name  
'foo/bar/baz:c:0'
```

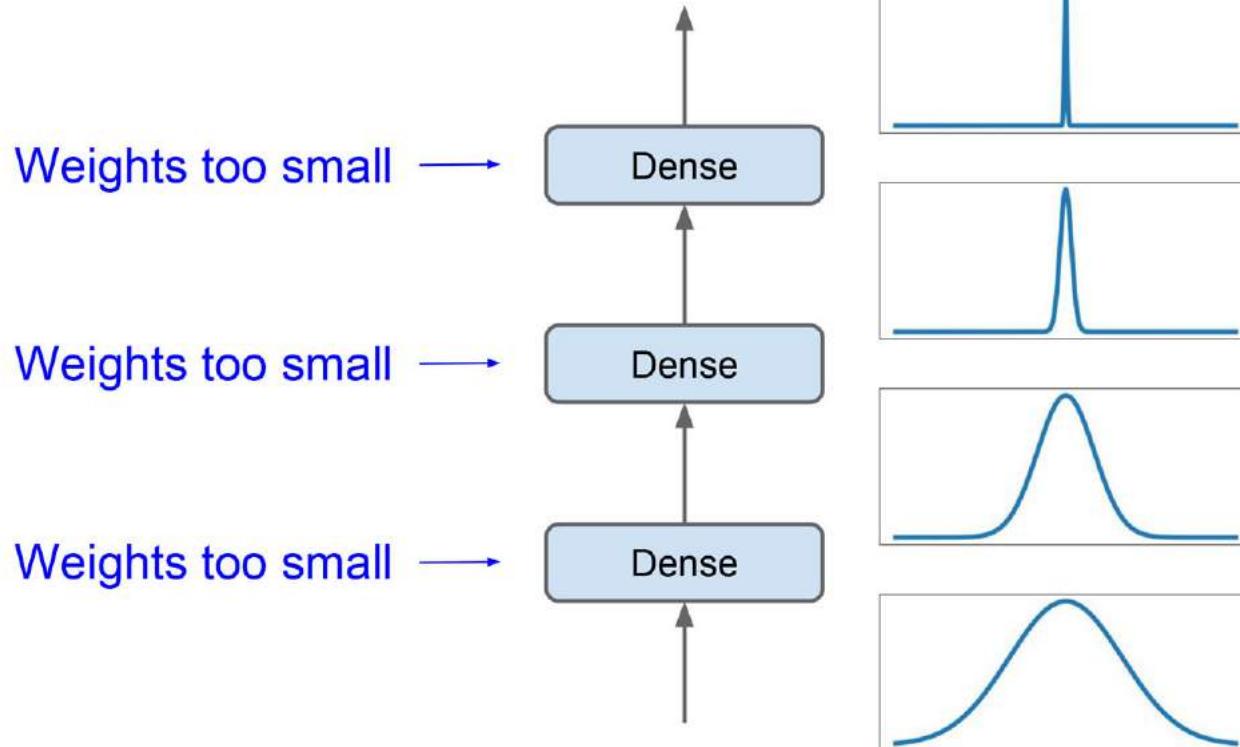
Organizing Your Code >

Variable Scopes and Name Scopes

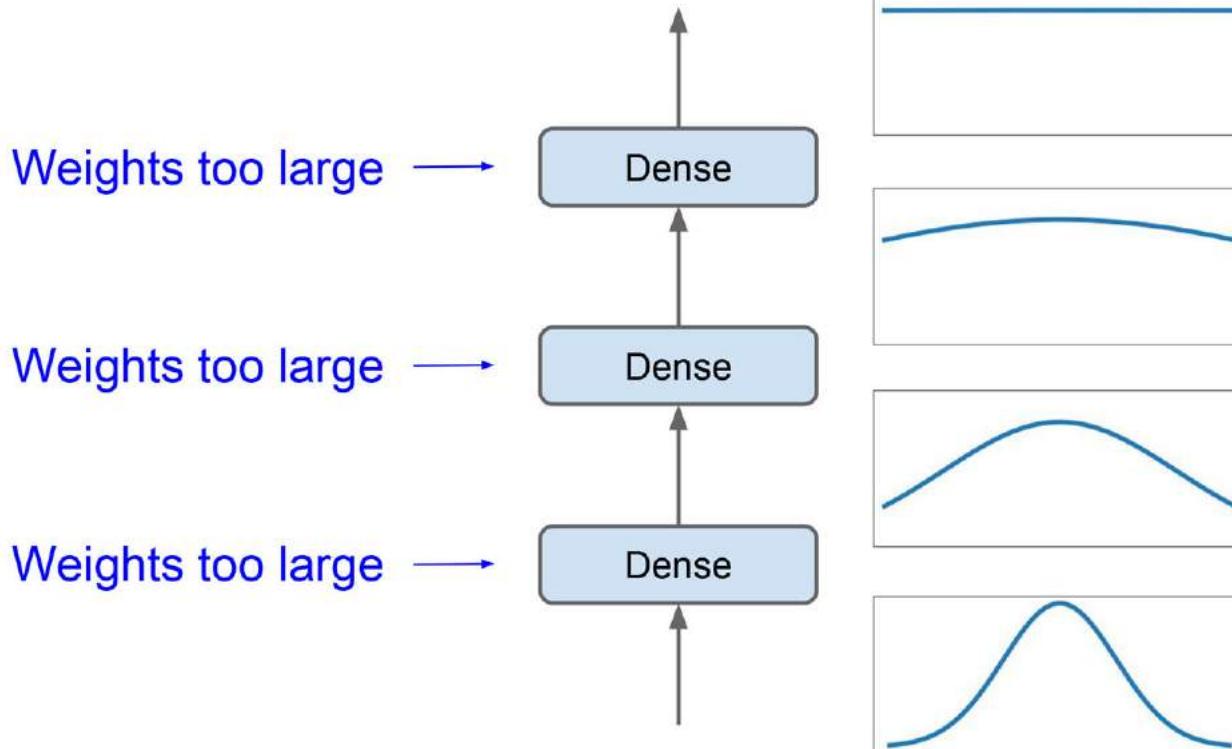
```
>>> with tf.variable_scope("foo"):  
...     a = tf.constant(1., name="a")  
...     with tf.name_scope("bar"):  
...         b = tf.constant(2., name="b")  
...         with tf.name_scope("baz"):  
...             c = tf.get_variable("c", ...)  
...             s = tf.add_n([a,b,c], name="s")  
  
>>> c.name  
'foo/c:0'  
  
>>> s.name  
'foo/bar/baz/s:0'
```

Techniques for Training Deep Nets

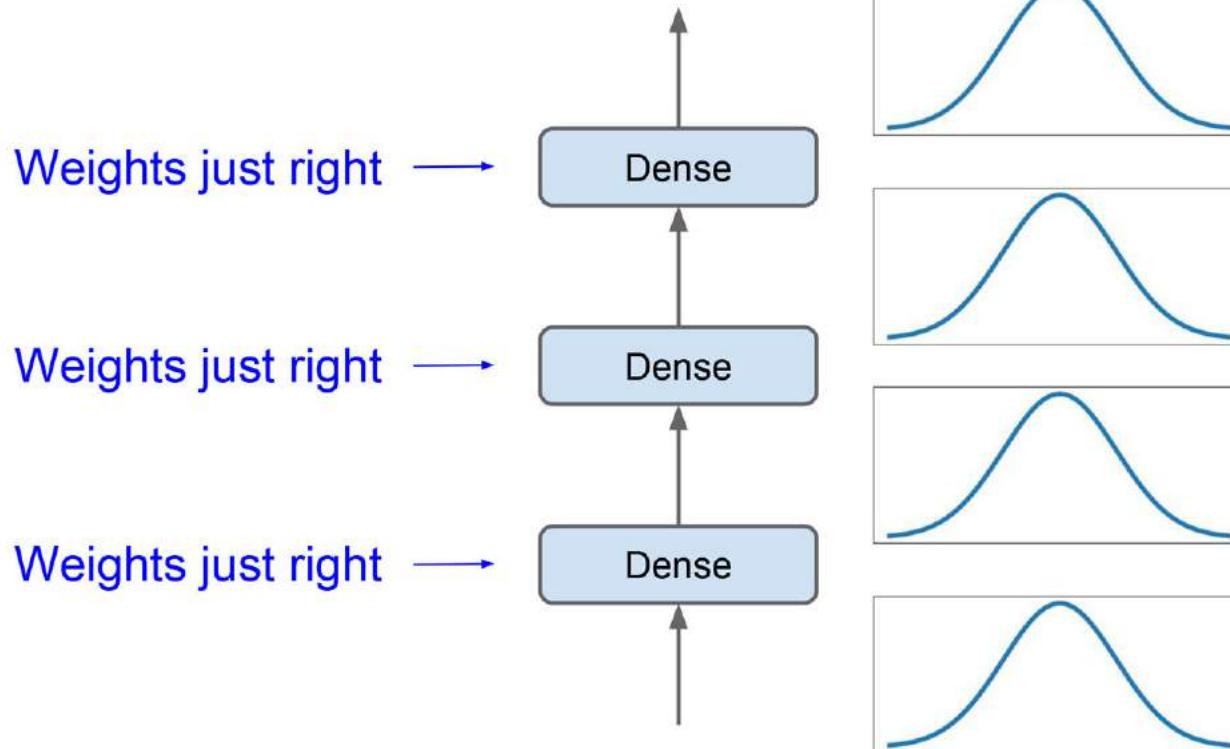
Variable Initialization



Variable Initialization



Variable Initialization



Techniques for DNNs >

Variable Initialization

```
>>> he_init = tf.contrib.layers.variance_scaling_initializer()
```

Techniques for DNNs > Variable Initialization

Techniques for DNNs

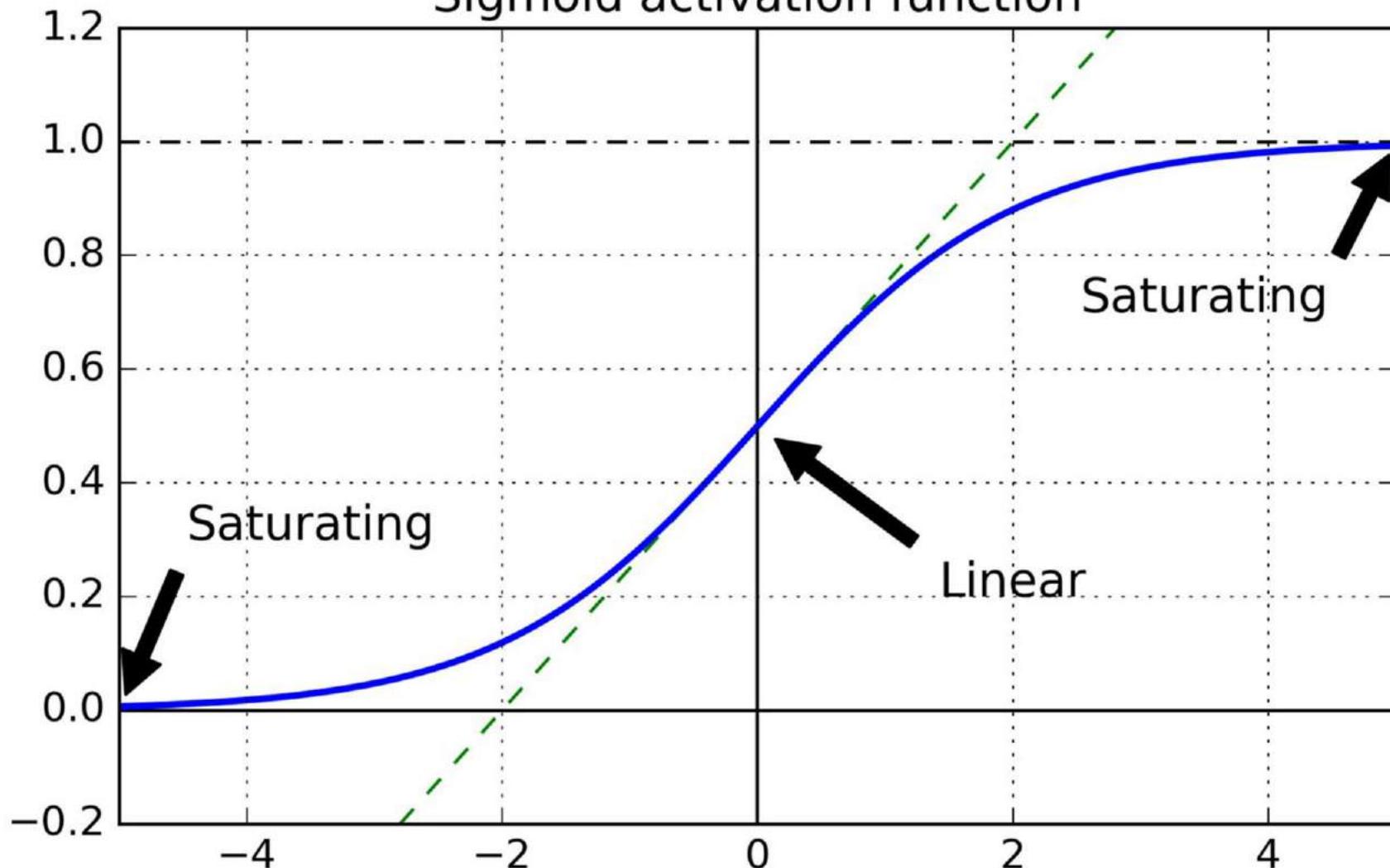
Techniques for DNNs > Variable Initialization

Techniques for DNNs > Python's partial()

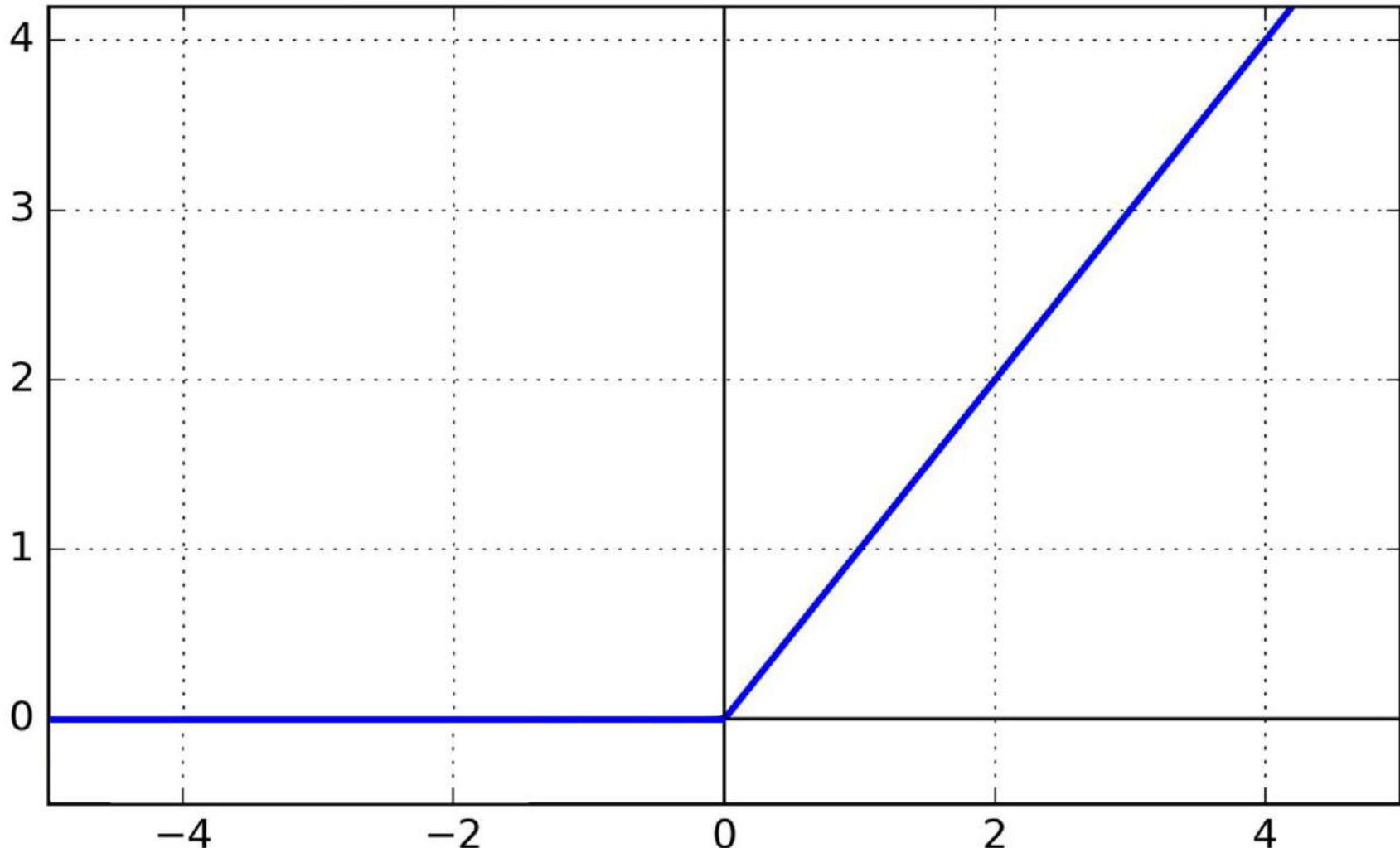
Techniques for DNNs > Python's partial()

```
>>> from functools import partial
>>> he_init = tf.contrib.layers.variance_scaling_initializer()
>>> dense_layer = partial(tf.layers.dense,
...                         activation=tf.nn.relu,
...                         kernel_initializer=he_init)
...
>>> hidden1 = dense_layer(X, n_hidden1, name="hidden1")
>>> hidden2 = dense_layer(hidden1, n_hidden2, name="hidden2")
>>> logits = dense_layer(hidden2, n_outputs, name="output",
...                       activation=None)
```

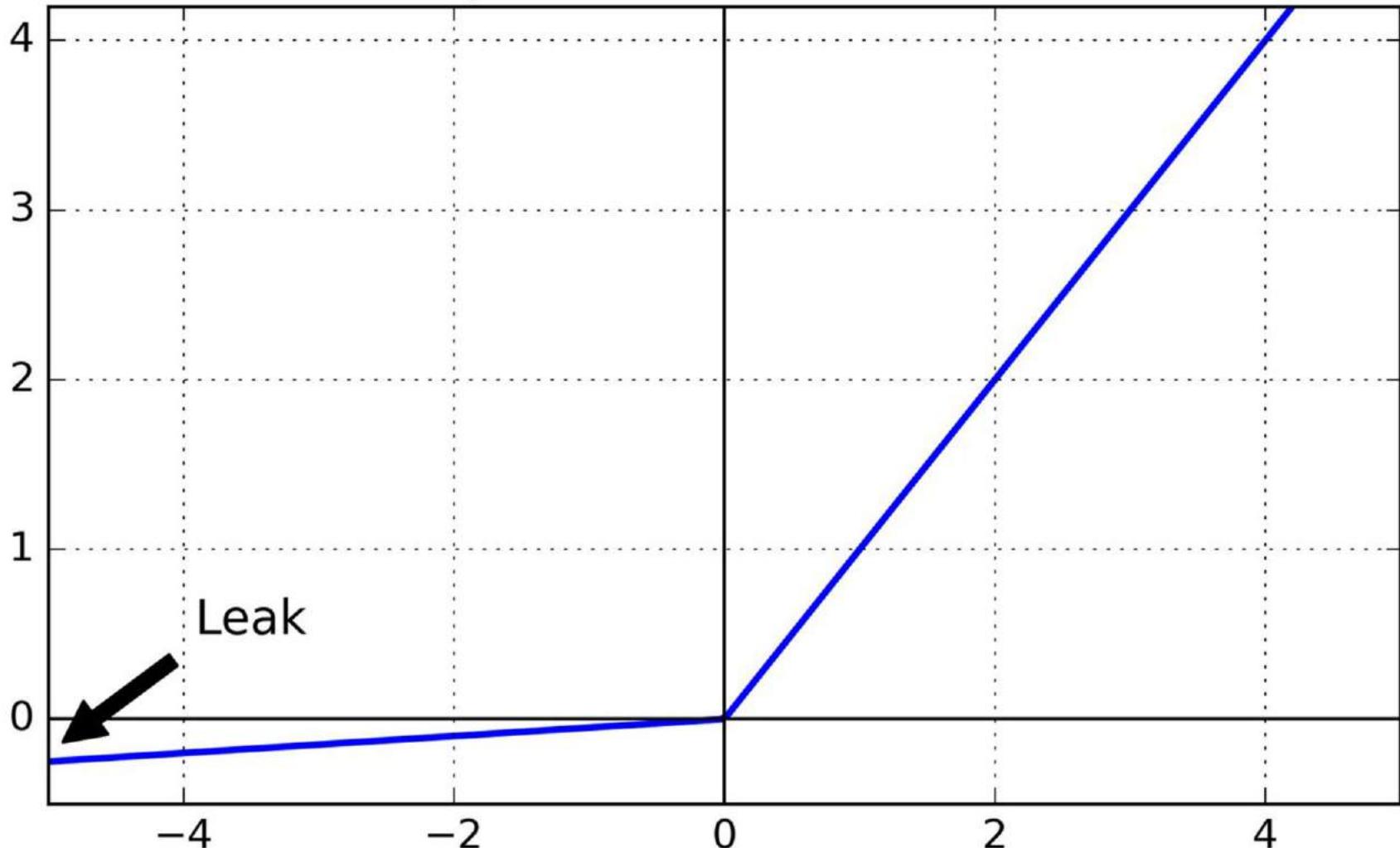
Sigmoid activation function



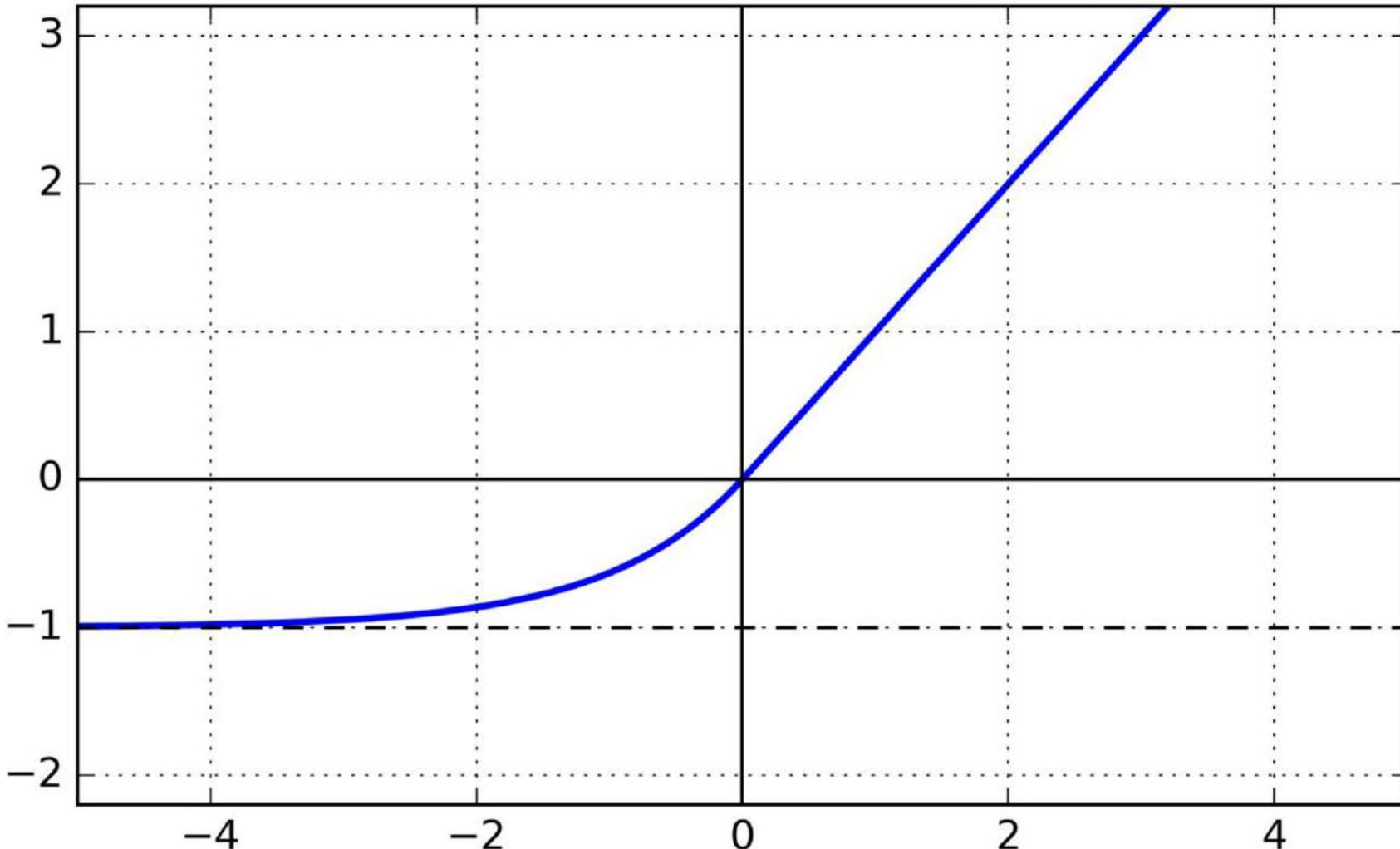
ReLU activation function



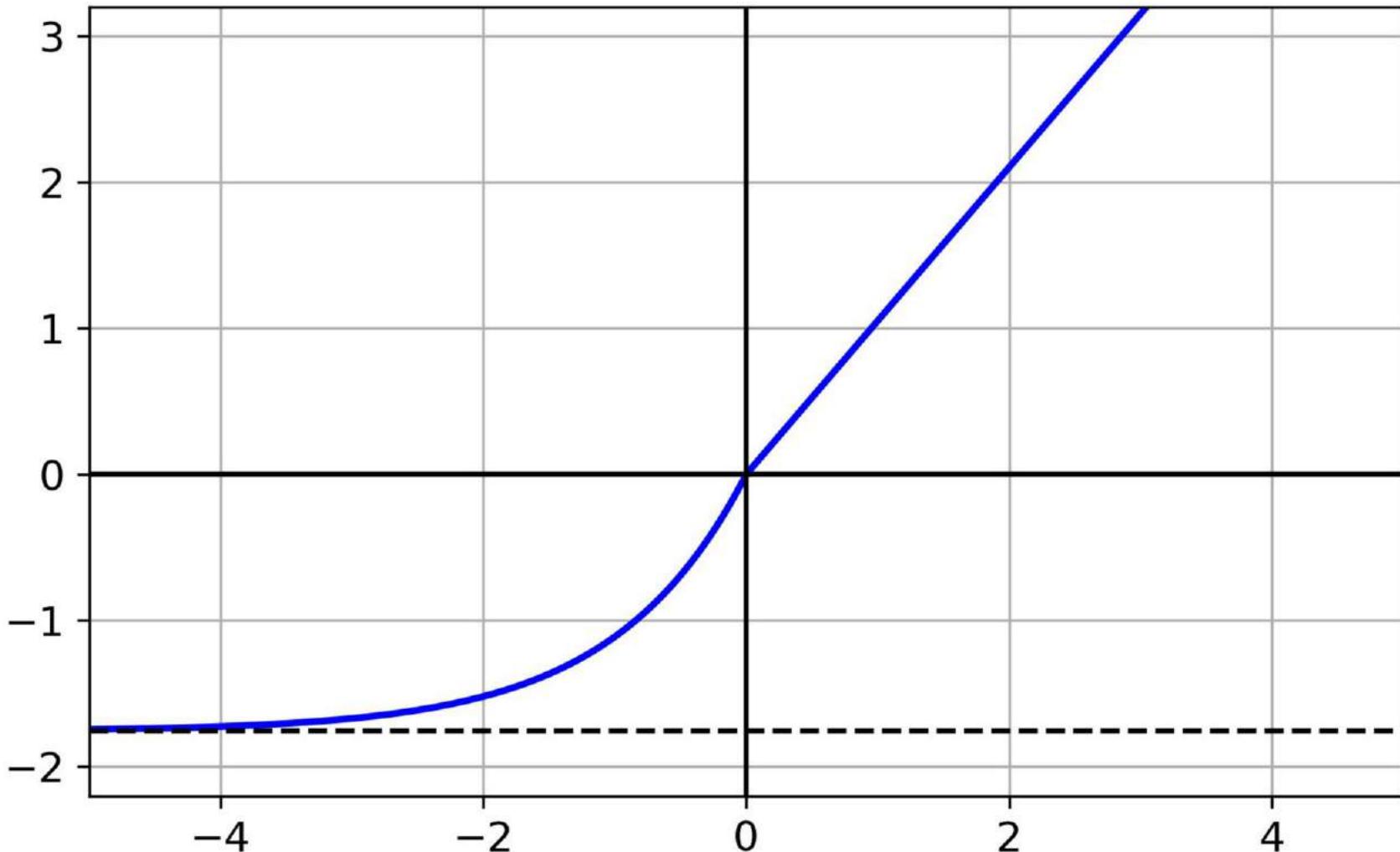
Leaky ReLU activation function



ELU activation function ($\alpha=1$)



SELU activation function



Using the SELU activation function

```
>>> from functools import partial
>>> he_init = tf.contrib.layers.variance_scaling_initializer()
>>> dense_layer = partial(tf.layers.dense,
...                         activation=tf.nn.selu,
...                         kernel_initializer=he_init)
...
>>> hidden1 = dense_layer(X, n_hidden1, name="hidden1")
>>> hidden2 = dense_layer(hidden1, n_hidden2, name="hidden2")
>>> logits = dense_layer(hidden2, n_outputs, name="output",
...                       activation=None)
```

Techniques for DNNs > Batch Normalization > Training

$$\frac{x_{batch} - x_{batch}.mean(axis=0)}{x_{batch}.std(axis=0)}$$

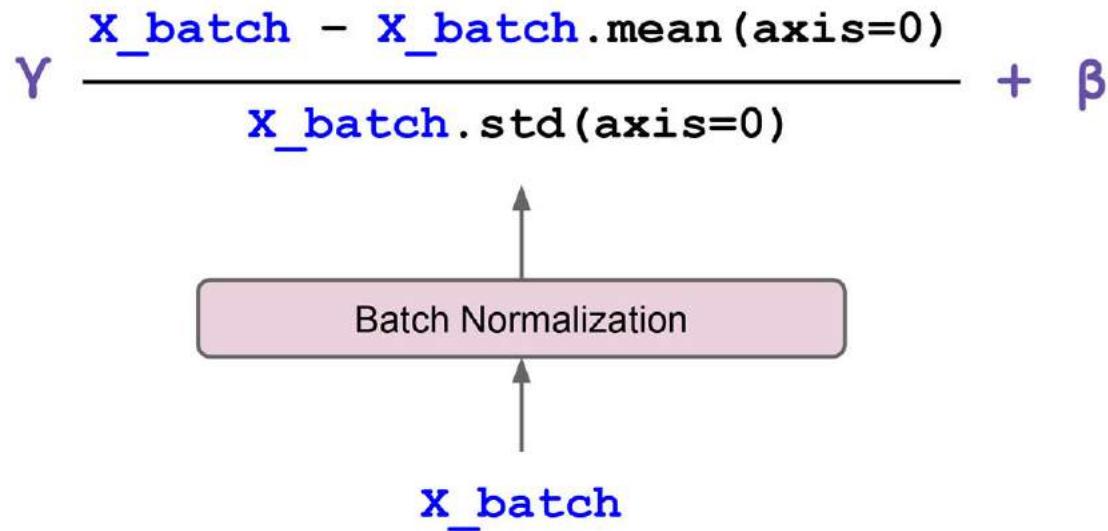
Batch Normalization

The diagram illustrates the formula for Batch Normalization. At the bottom, the input `x_batch` is shown in blue. An arrow points upwards from this input to a pink rounded rectangle labeled "Batch Normalization". From the top of this rectangle, two arrows point upwards to the terms in the formula: `x_batch - x_batch.mean(axis=0)` and `x_batch.std(axis=0)`. These terms are separated by a horizontal line.

Techniques for DNNs > Batch Normalization > Training

$$\gamma \frac{x_{\text{batch}} - x_{\text{batch}}.\text{mean}(\text{axis}=0)}{x_{\text{batch}}.\text{std}(\text{axis}=0)} + \beta$$

Batch Normalization



Techniques for DNNs > Batch Normalization > Testing

$$\gamma = \frac{x_{\text{new}} - x_{\text{train}}.\text{mean}(\text{axis}=0)}{x_{\text{train}}.\text{std}(\text{axis}=0)} + \beta$$

Batch Normalization

```
graph TD; x[x_new] --> BN[Batch Normalization]; BN --> formula
```

Batch Normalization > Testing

Typically estimated using an exponential moving average during training

$$\gamma = \frac{x_{\text{new}} - x_{\text{train}}.\text{mean}(\text{axis}=0)}{x_{\text{train}}.\text{std}(\text{axis}=0)} + \beta$$

Batch Normalization

The diagram illustrates the formula for testing batch normalization. It shows the input x_{new} in green, the mean of the training set $x_{\text{train}}.\text{mean}(\text{axis}=0)$ in red, and the standard deviation of the training set $x_{\text{train}}.\text{std}(\text{axis}=0)$ in red. The formula is $\gamma = \frac{x_{\text{new}} - x_{\text{train}}.\text{mean}(\text{axis}=0)}{x_{\text{train}}.\text{std}(\text{axis}=0)} + \beta$. A blue arrow points from the parameter β to the addition sign. The entire formula is enclosed in a pink rounded rectangle labeled "Batch Normalization".

Techniques for DNNs > **Batch Normalization > Construction Phase**

```
>>> training = tf.placeholder_with_default(False, shape=())
>>> hidden1 = tf.layers.dense(X, n_hidden1,
...                             activation=tf.nn.relu)
...
...
>>> bn1 = tf.layers.batch_normalization(hidden1,
...                                       training=training,
...                                       momentum=0.9)
```

Techniques for DNNs > **Batch Normalization > Construction Phase**

```
>>> training = tf.placeholder_with_default(False, shape=())
>>> hidden1 = tf.layers.dense(X, n_hidden1,
...                             activation=tf.nn.relu)
...
...
>>> bn1 = tf.layers.batch_normalization(hidden1,
...                                       training=training,
...                                       momentum=0.9)
```

Training = use batch statistics

Not training = use full training set statistics

Techniques for DNNs > **Batch Normalization > Construction Phase**

```
>>> training = tf.placeholder_with_default(False, shape=())
>>> hidden1 = tf.layers.dense(X, n_hidden1,
...                             activation=tf.nn.relu)
...
...
>>> bn1 = tf.layers.batch_normalization(hidden1,
...                                       training=training,
...                                       momentum=0.9)
```



For computing the exponential moving averages:

$$\begin{aligned} \text{mean}_{(i)} &= \text{momentum} * \text{mean}_{(i-1)} + \text{batch_mean}_{(i)} \\ \text{std}_{(i)} &= \text{momentum} * \text{std}_{(i-1)} + \text{batch_std}_{(i)} \end{aligned}$$

Techniques for DNNs > **Batch Normalization > Construction Phase**

```
>>> training = tf.placeholder_with_default(False, shape=())
>>> hidden1_pre_activation = tf.layers.dense(X, n_hidden1)
>>> bn1 = tf.layers.batch_normalization(
...     hidden1_pre_activation, training=training, momentum=0.9)
...
>>> hidden1 = tf.nn.relu(bn1)
```

Techniques for DNNs > **Batch Normalization > Execution Phase**

```
>>> extra_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
>>> with tf.Session() as sess:
...     [...]
...     sess.run(
...         [training_op, extra_ops],
...         feed_dict={training: True, x: x_batch, y: y_batch})
```

Techniques for DNNs > **Batch Normalization > Execution Phase**

```
>>> extra_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
>>> with tf.Session() as sess:
...     [...]
...     sess.run(
...         [training_op, extra_ops],
...         feed_dict={training: True, x: x_batch, y: y_batch})
```

For computing the exponential moving averages

Techniques for DNNs > **Batch Normalization > Execution Phase**

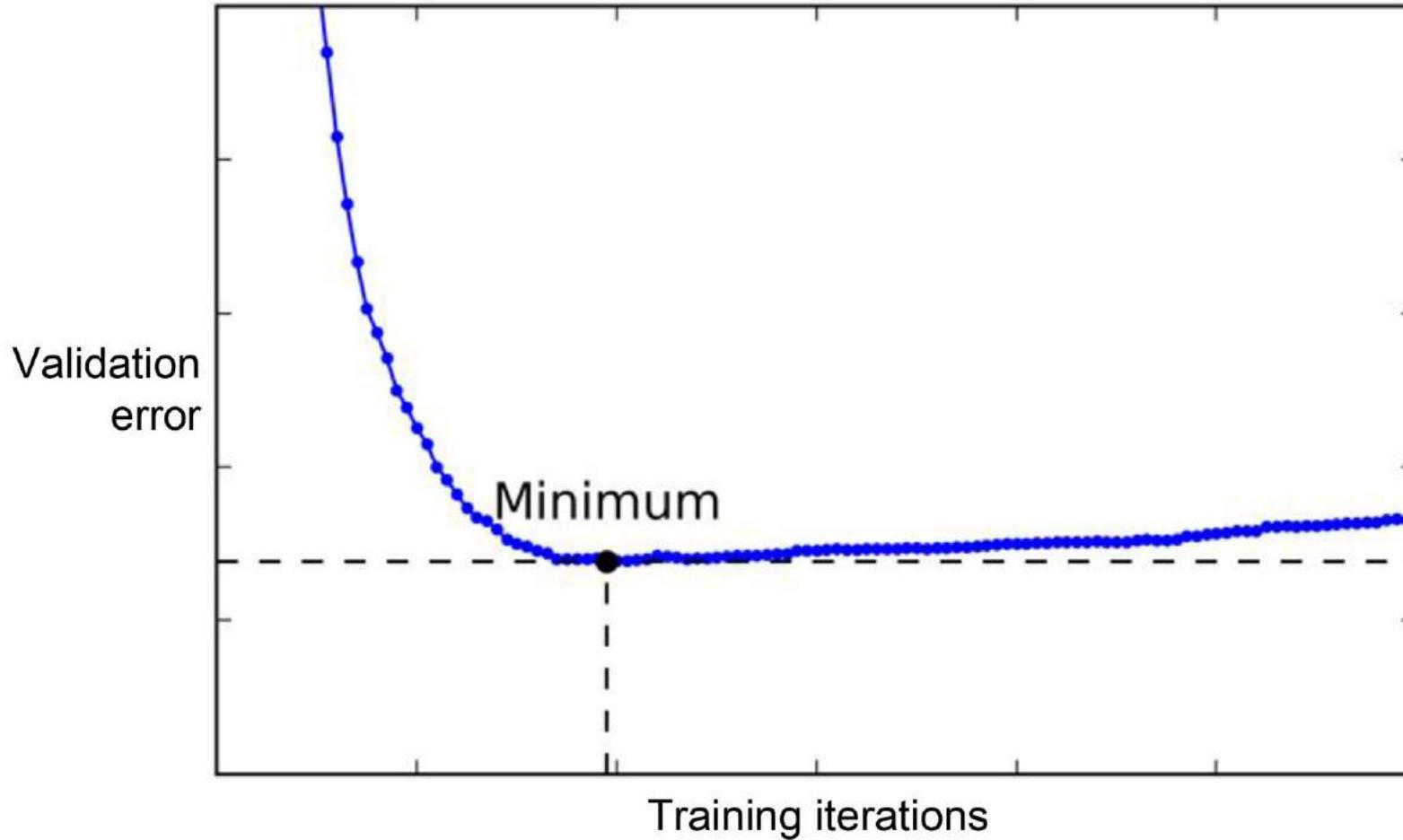
```
>>> extra_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
>>> with tf.Session() as sess:
...     [...]
...     sess.run(
...         [training_op, extra_ops],
...         feed_dict={training: True, x: x_batch, y: y_batch})
```



Training: use batch statistics

Not training: use full training set statistics

Early Stopping



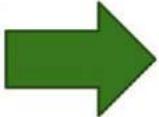
Techniques for DNNs >

Early Stopping

```
>>> best_acc_val = 0.0
>>> check_interval = 100
>>> with tf.Session(graph=graph) as sess:
...     ...
...     sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
...     if iteration % check_interval == 0:
...         acc_val = accuracy.eval(feed_dict={X: X_val, y: y_val})
...         if acc_val > best_acc_val:
...             best_acc_val = acc_val
...             saver.save(sess, "./my_best_model_so_far")
```

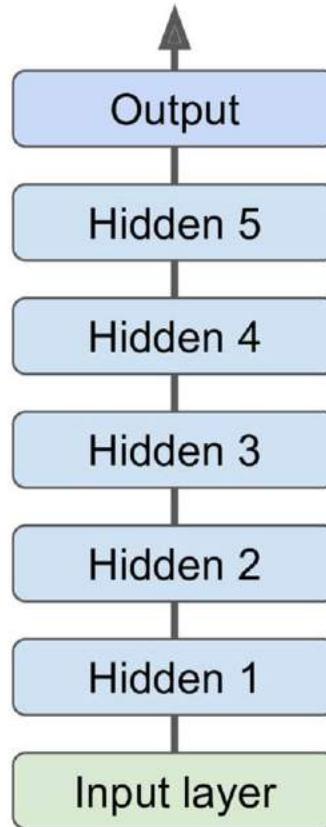
Techniques for DNNs >

Data Augmentation



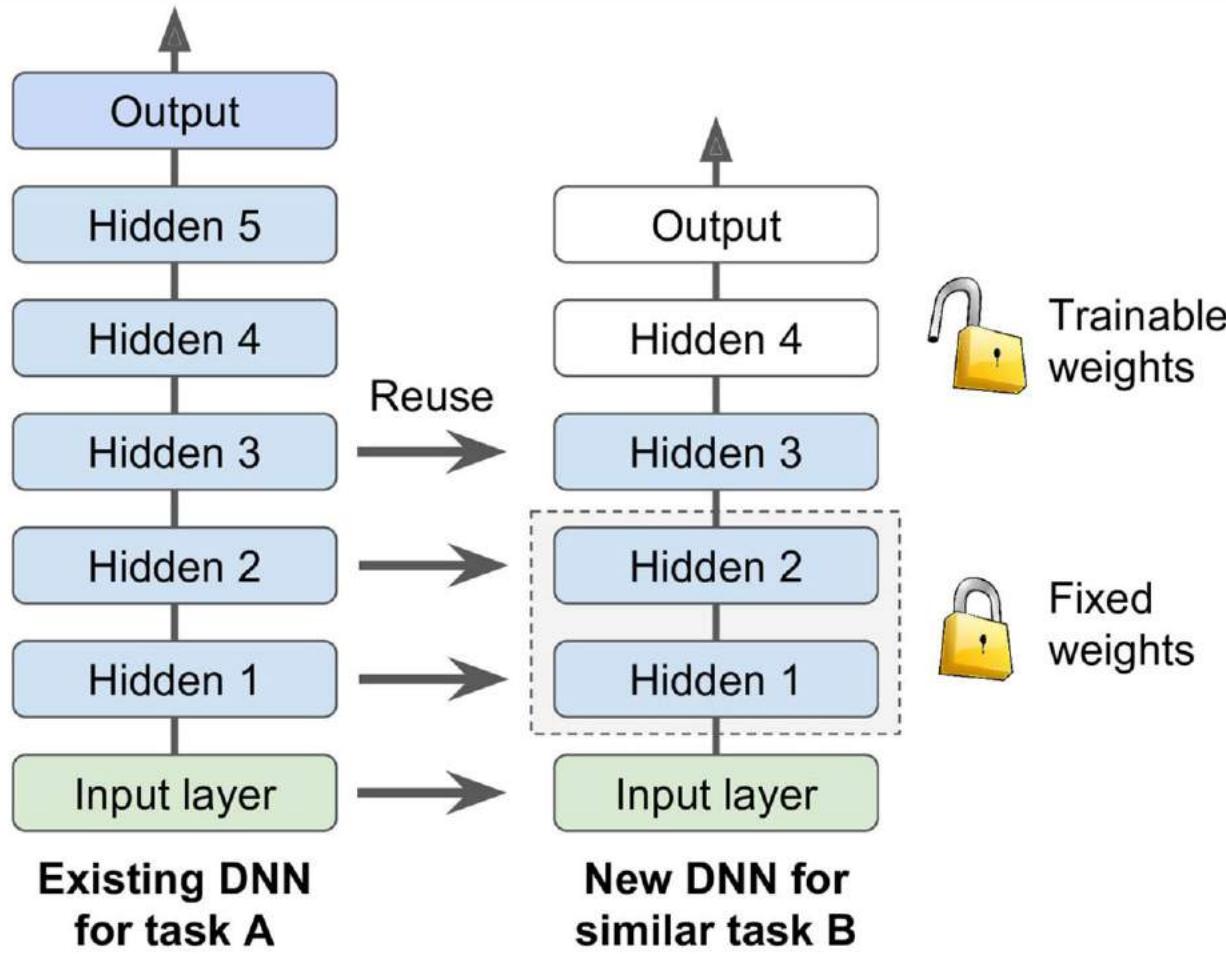
Check out the `tf.image` module for many image transformation functions.

Transfer Learning



**Existing DNN
for task A**

Transfer Learning



Techniques for DNNs >

Freezing Layers

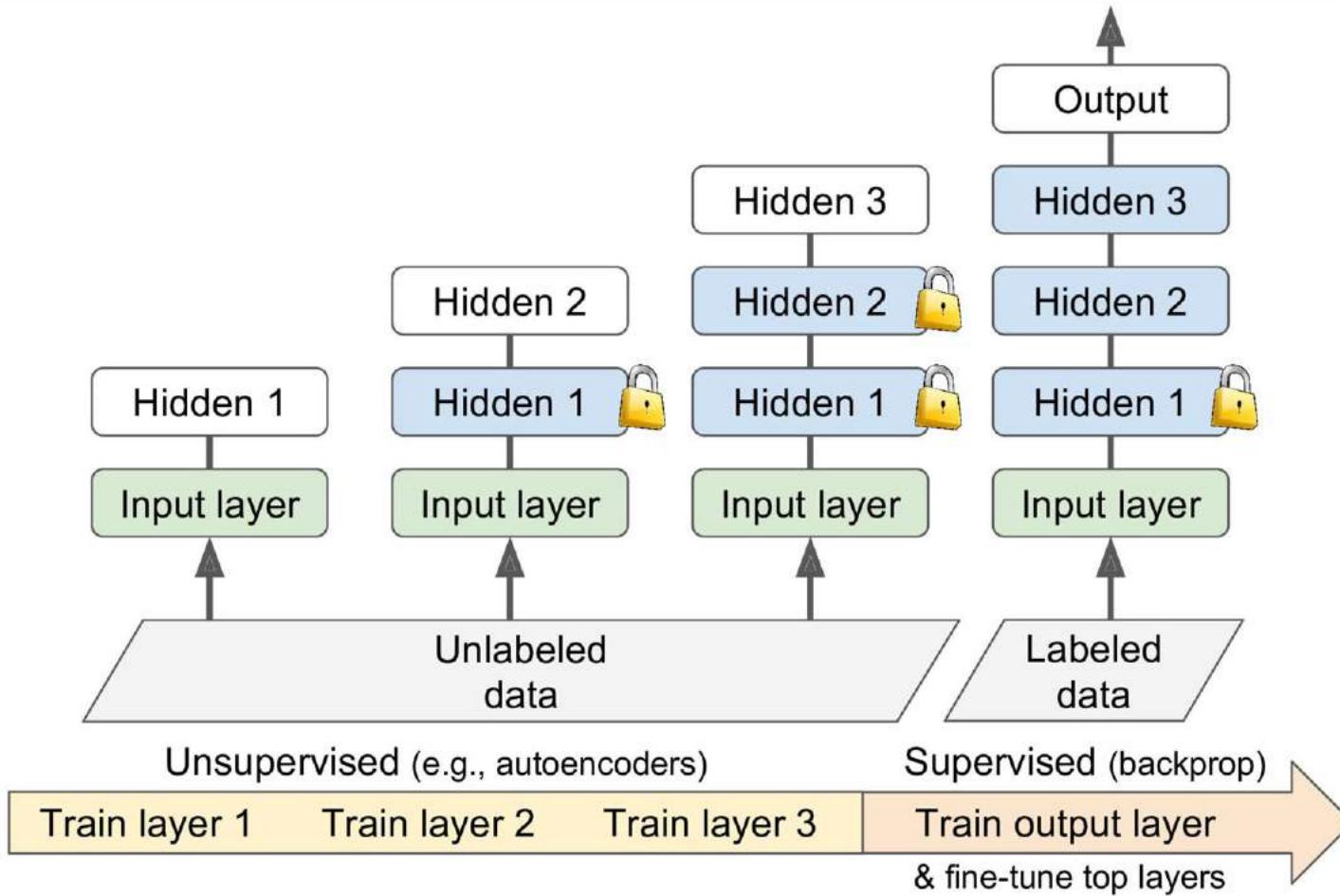
```
>>> tvars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,  
...  
    scope="hidden[34] | outputs")
```

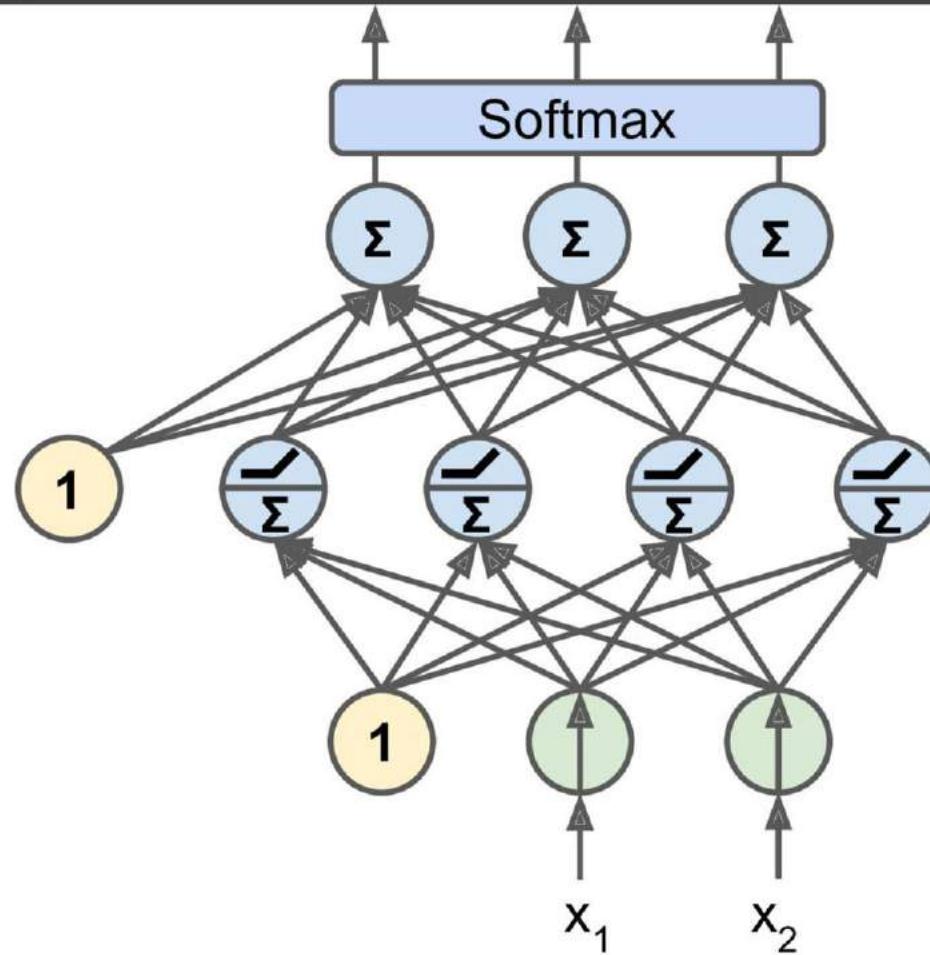
Techniques for DNNs >

Freezing Layers

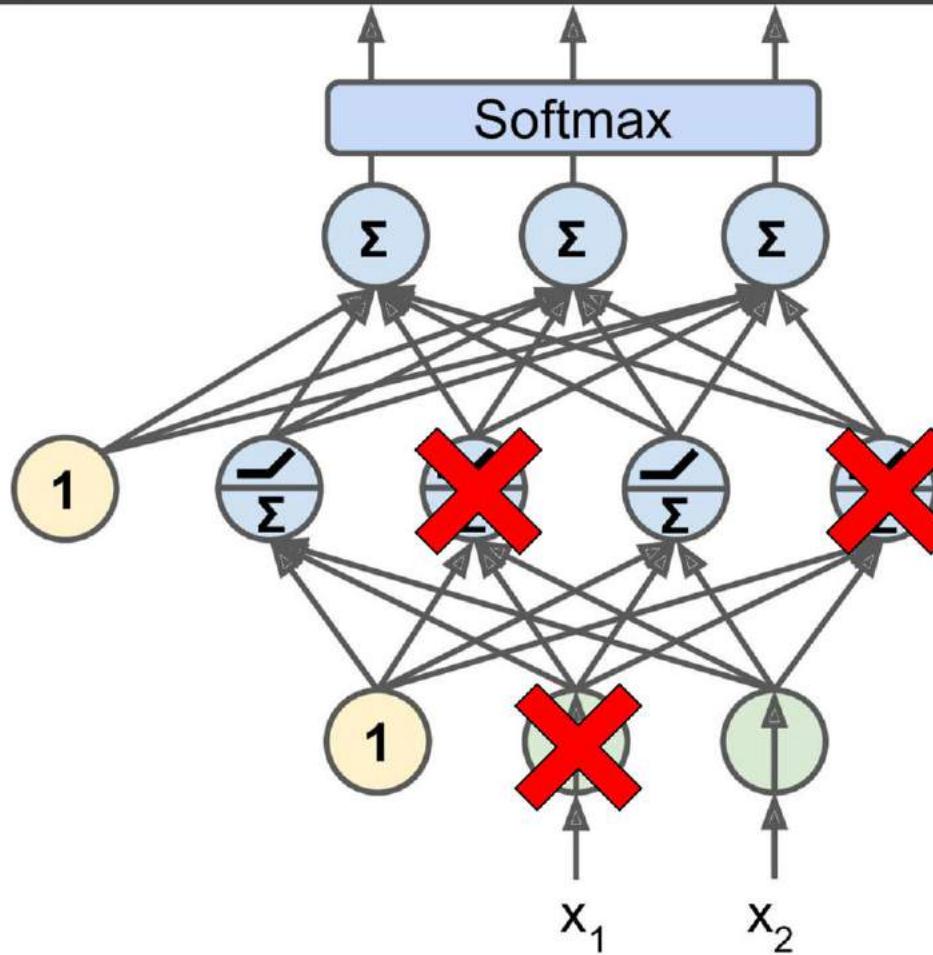
```
>>> tvars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,  
...                           scope="hidden[34]|output")  
>>> training_op = optimizer.minimize(loss, var_list=train_vars)
```

Unsupervised Pretraining

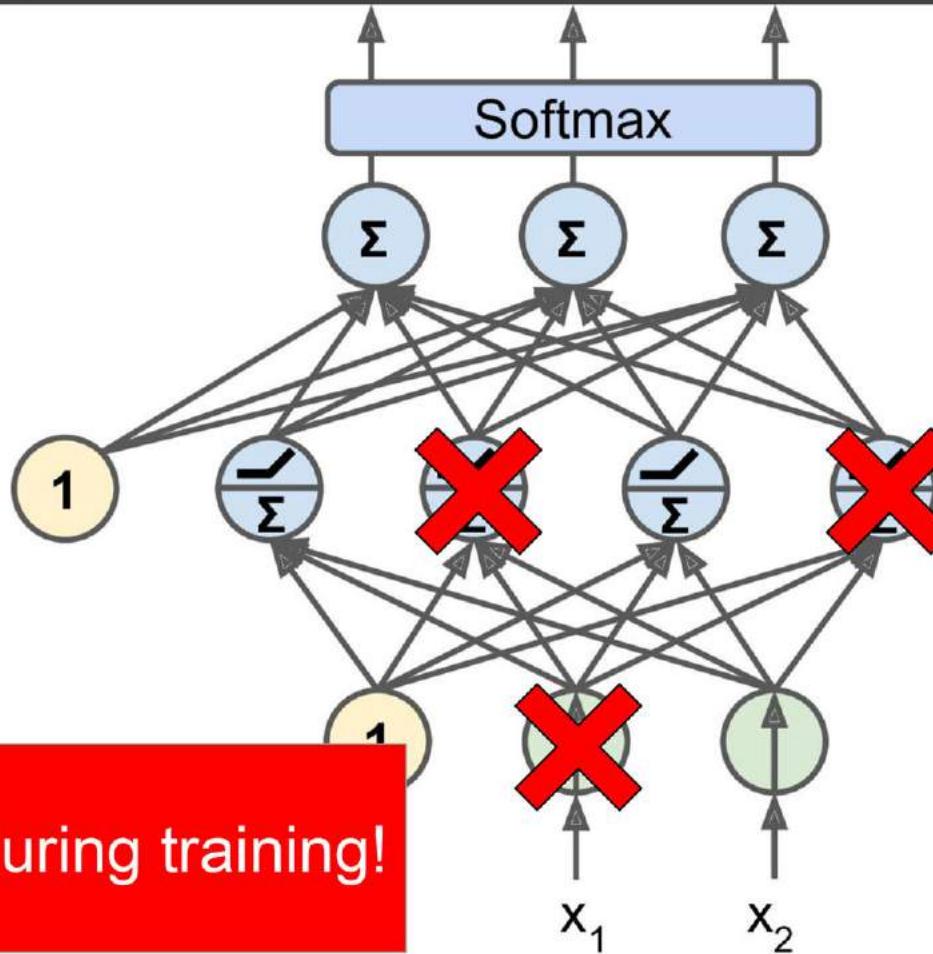




Techniques for DNNs > Dropout



Techniques for DNNs > Dropout



Only during training!

5 minutes

Exercise 9



Techniques for DNNs > **Implementing Dropout**

```
>>> dropout_rate = 0.5
```

Techniques for DNNs >

Implementing Dropout

```
>>> dropout_rate = 0.5
>>> with tf.name_scope("inputs"):
...     x = tf.placeholder(...)
...     y = tf.placeholder(...)
...     training = tf.placeholder_with_default(
...         False, shape=[], name='training')
```

Techniques for DNNs >

Implementing Dropout

```
>>> dropout_rate = 0.5
>>> with tf.name_scope("inputs"):
...     x = tf.placeholder(...)
...     y = tf.placeholder(...)
...     training = tf.placeholder_with_default(
...         False, shape=[], name='training')
...     x_drop = dropout(x, dropout_rate, training=training)
```

Techniques for DNNs >

Implementing Dropout

```
>>> dropout_rate = 0.5
>>> with tf.name_scope("inputs"):
...     x = tf.placeholder(...)
...     y = tf.placeholder(...)
...     training = tf.placeholder_with_default(
...         False, shape=[], name='training')
...     x_drop = dropout(x, dropout_rate, training=training)
...
>>> hidden1 = tf.layers.dense(x_drop, n_hidden1, ...)
```

Techniques for DNNs > **Implementing Dropout > Training**

```
>>> with tf.Session(graph=graph) as sess:  
...     init.run()  
...     for epoch in range(n_epochs):  
...         for iteration in range(...):  
...             x_batch, y_batch = ...  
...             sess.run(training_op, feed_dict={  
...                 X: x_batch, y: y_batch, training: True})
```

What Next?

Develop

GET STARTED

PROGRAMMER'S GUIDE

TUTORIALS

PERFORMANCE

Mandelbrot Set

Partial Differential Equations

Convolutional Neural Networks

Image Recognition

[How to Retrain Inception's Final Layer
for New Categories](#)

Vector Representations of Words

Recurrent Neural Networks

Sequence-to-Sequence Models

A Guide to TF Layers: Building a
Convolutional Neural NetworkLarge-scale Linear Models with
TensorFlow

TensorFlow Linear Model Tutorial

TensorFlow Wide & Deep Learning
Tutorial

Using GPUs

TensorFlow Versions

How to Retrain Inception's Final Layer for New Categories

Modern object recognition models have millions of parameters and can take weeks to fully train. Transfer learning is a technique that shortcuts a lot of this work by taking a fully-trained model for a set of categories like ImageNet, and retrains from the existing weights for new classes. In this example we'll be retraining the final layer from scratch, while leaving all the others untouched. For more information on the approach you can see [this paper on Decaf](#).

Though it's not as good as a full training run, this is surprisingly effective for many applications, and can be run in as little as thirty minutes on a laptop, without requiring a GPU. This tutorial will show you how to run the example script on your own images, and will explain some of the options you have to help control the training process.

Training on Flowers



[ageron / handson-ml](#)[Watch](#)

510

[Star](#)

6,818

[Fork](#)

3,430

[Code](#)[Issues 18](#)[Pull requests 4](#)[Projects 0](#)[Insights](#)

A series of Jupyter notebooks that walk you through the fundamentals of Machine Learning using Scikit-Learn and TensorFlow.

[tensorflow](#)[scikit-learn](#)[machine-learning](#)[python](#)[deep-learning](#)[neural-network](#)[ml](#)[306 commits](#)[1 branch](#)[0 releases](#)Branch: [master](#) ▾[New pull request](#)

 **ageron** Merge branch 'master' of [github.com:ageron/handson-ml](#)

[datasets](#)

Update README.md

[docker](#)

Dependencies for OpenAI gym in Chapter 16

[images](#)

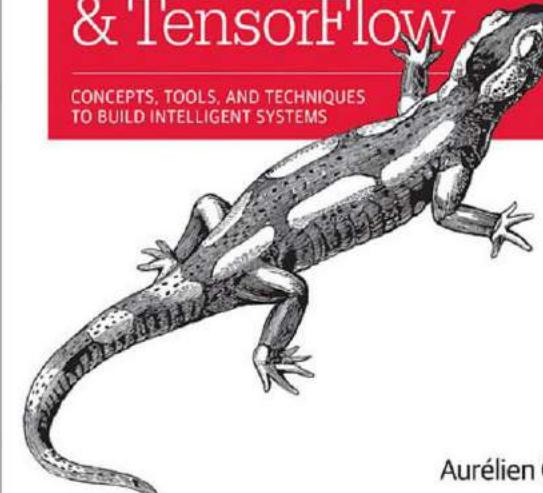
Upgrade chapter 2 to sklearn 0.18 and ensure python 2 a

[.gitignore](#)

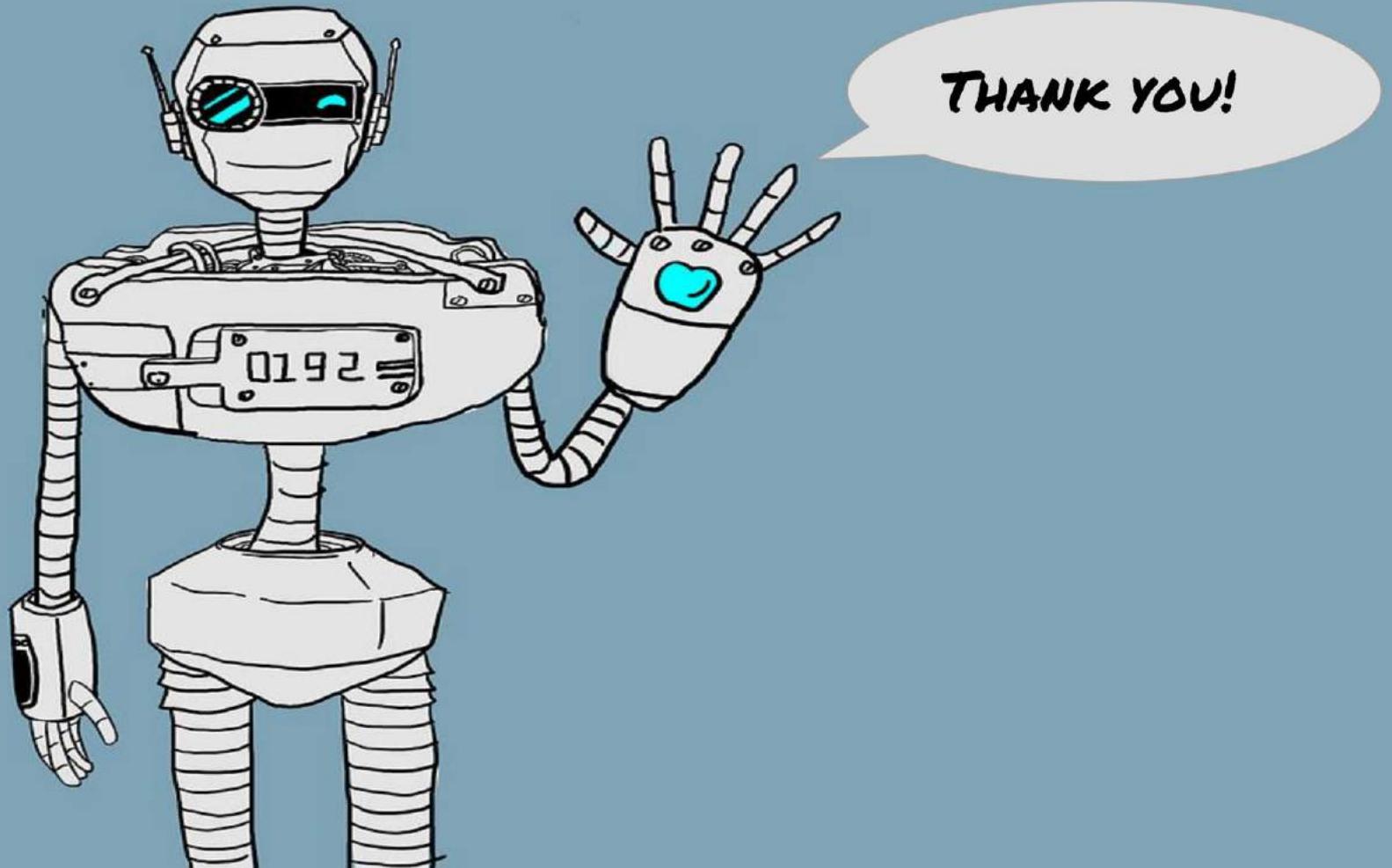
sync with upstream

O'REILLY

Hands-on Machine Learning with Scikit-Learn & TensorFlow

CONCEPTS, TOOLS, AND TECHNIQUES
TO BUILD INTELLIGENT SYSTEMS

Aurélien Géron

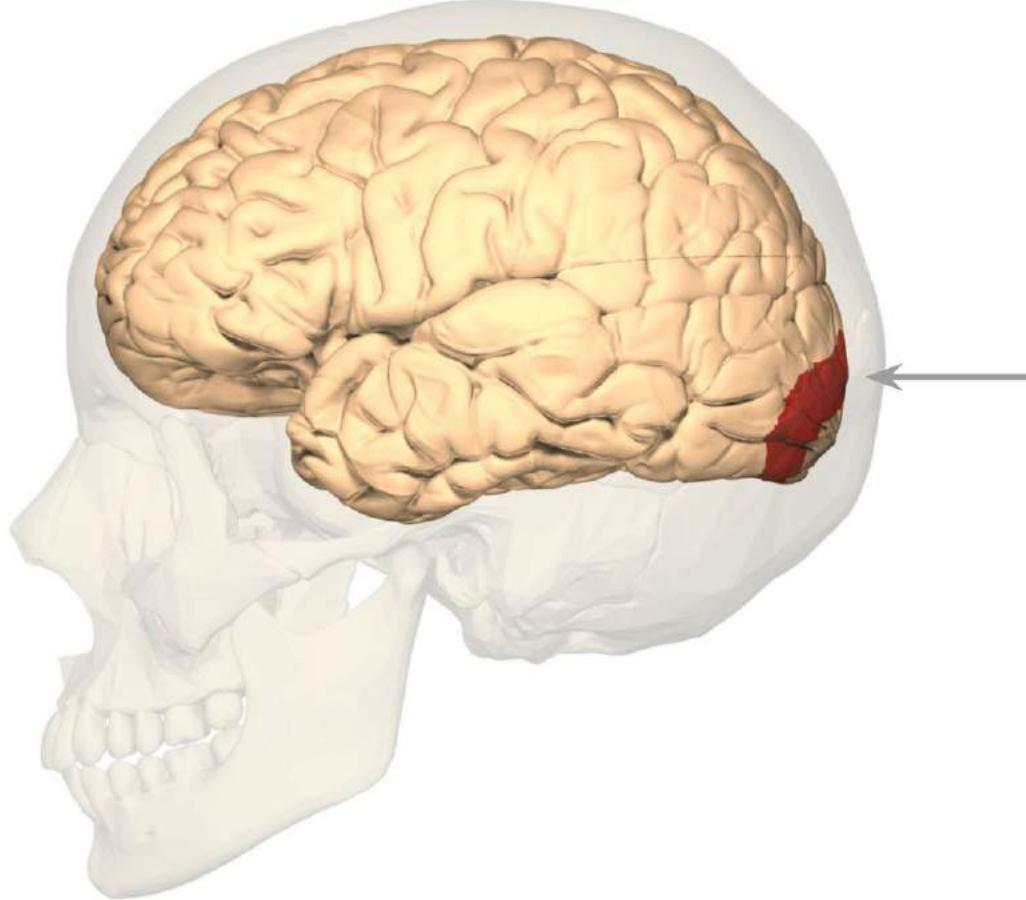


THANK YOU!

Bonus Slides

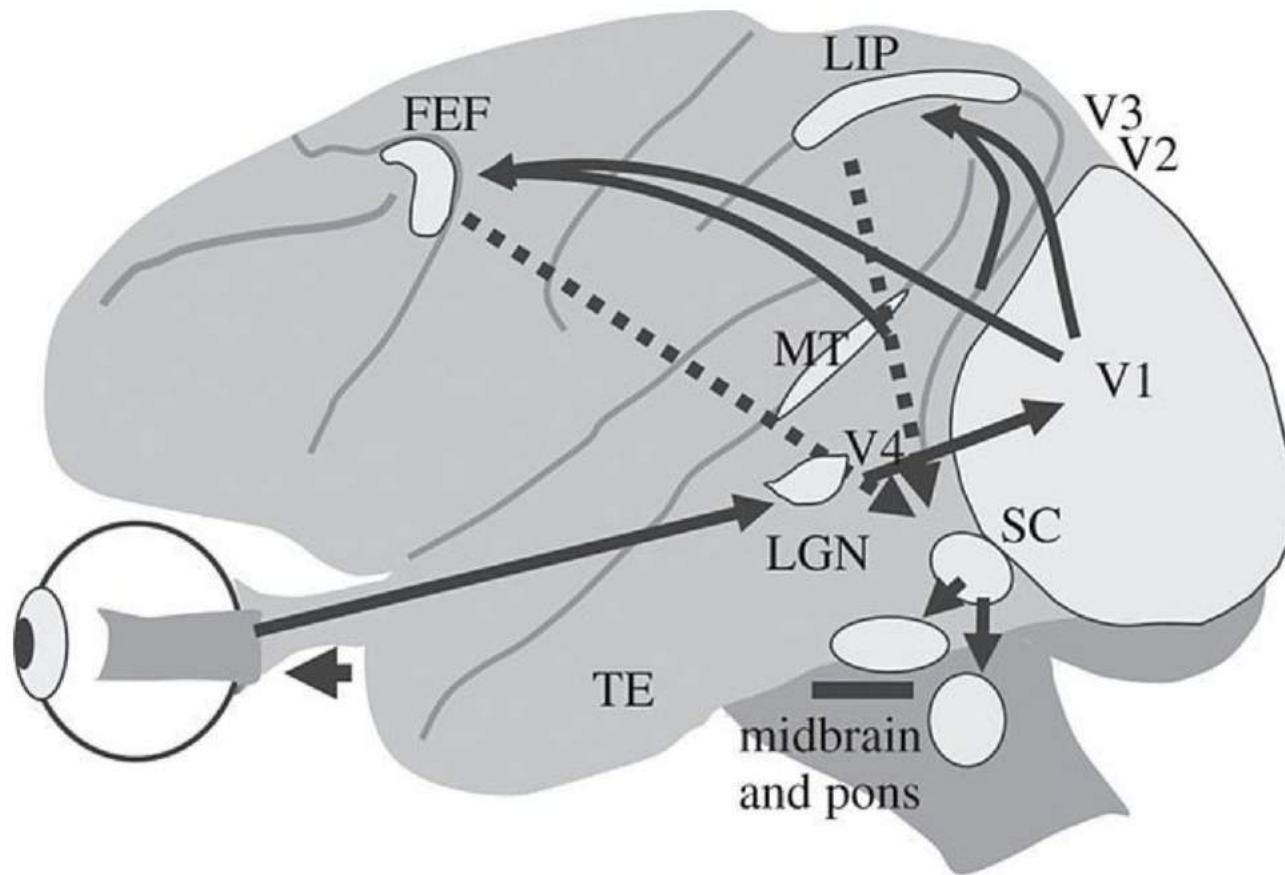
Convolutional Neural Nets for Image Classification

CNNs > Visual Cortex



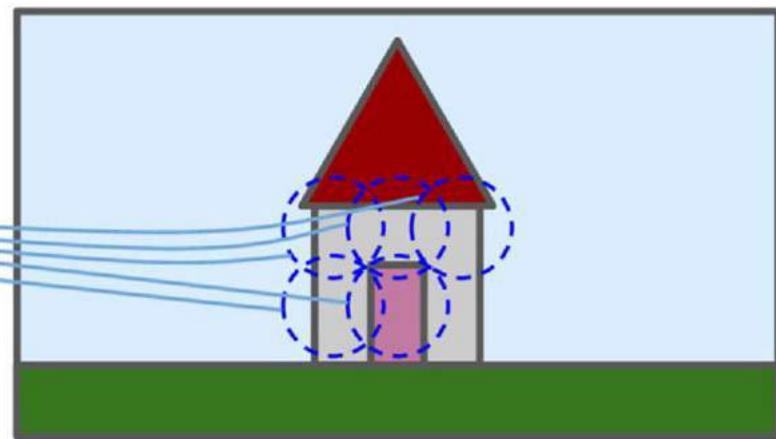
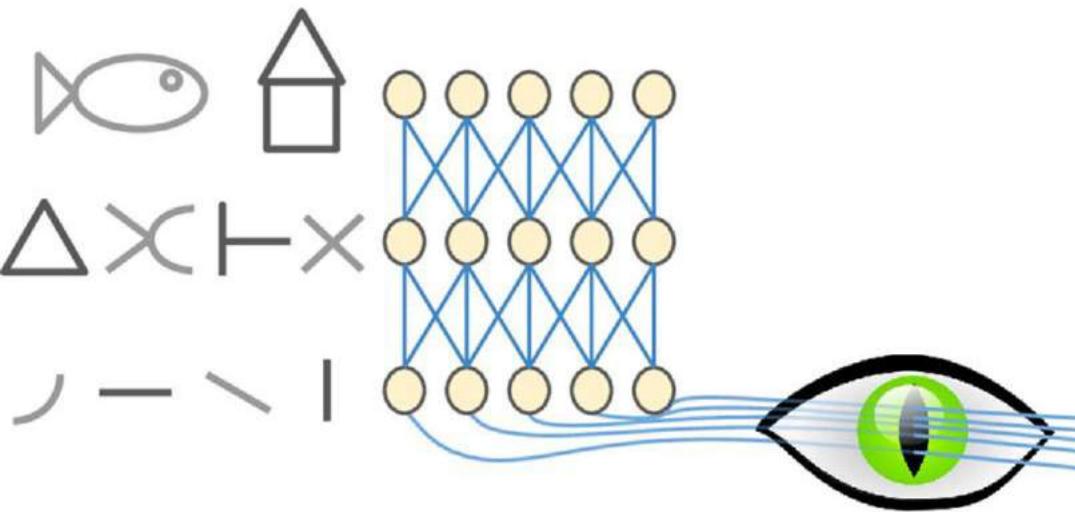
Visual Cortex
(V2)

CNNs > Visual Cortex > V1, V2...

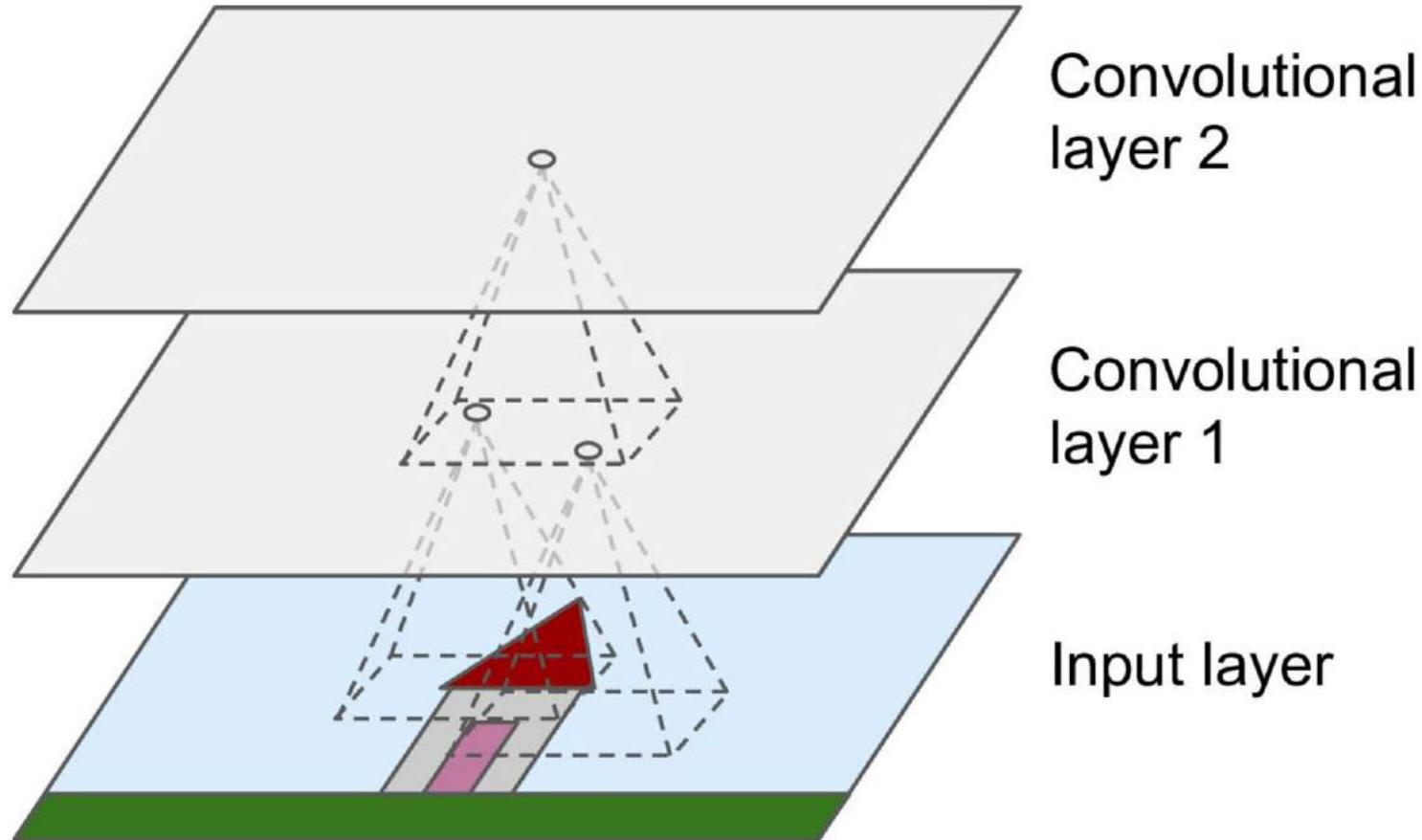


source: https://commons.wikimedia.org/wiki/File:Brain_circuits_for_visually_guided_saccades.jpg

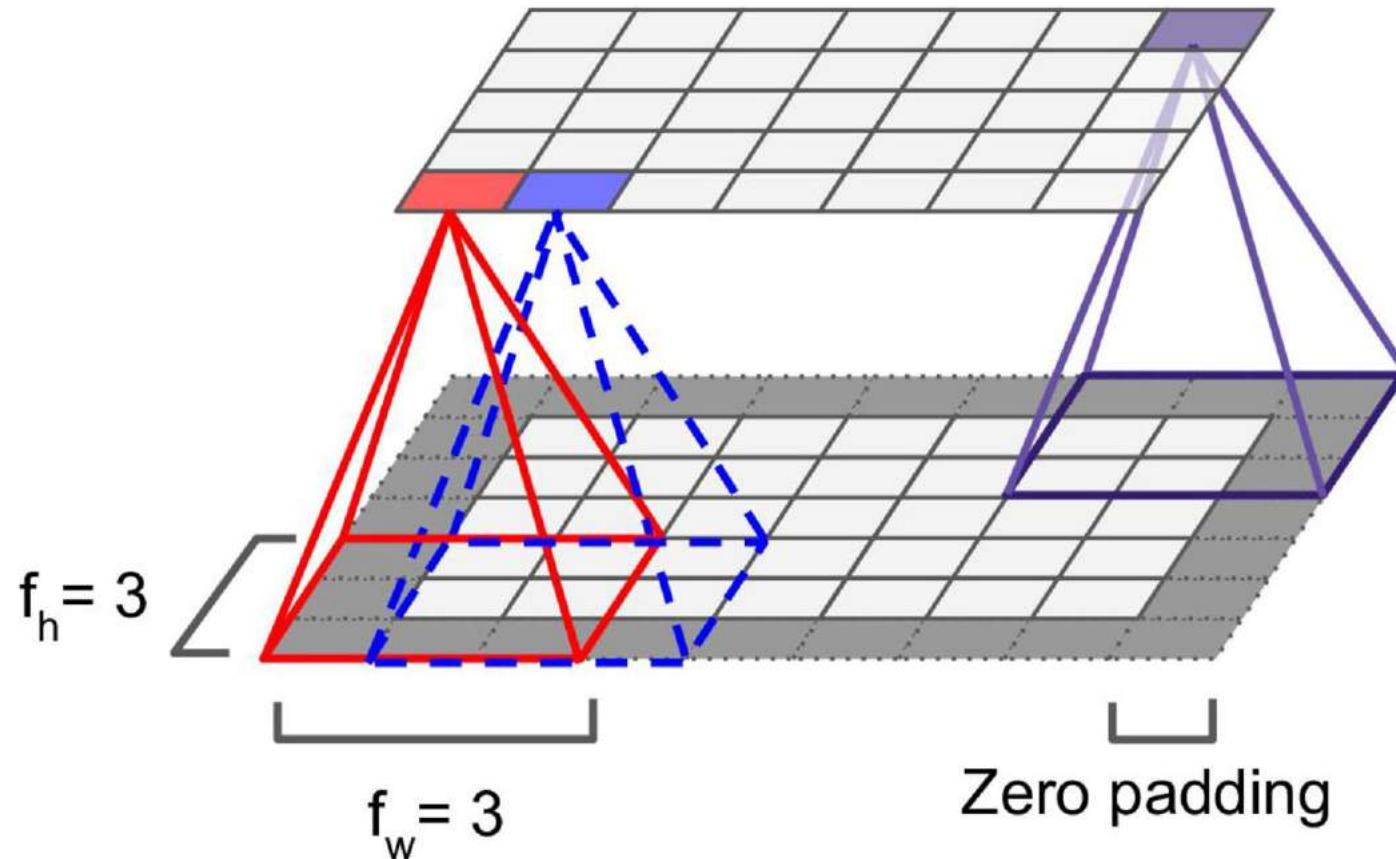
CNNs > Visual Cortex > Receptive Fields



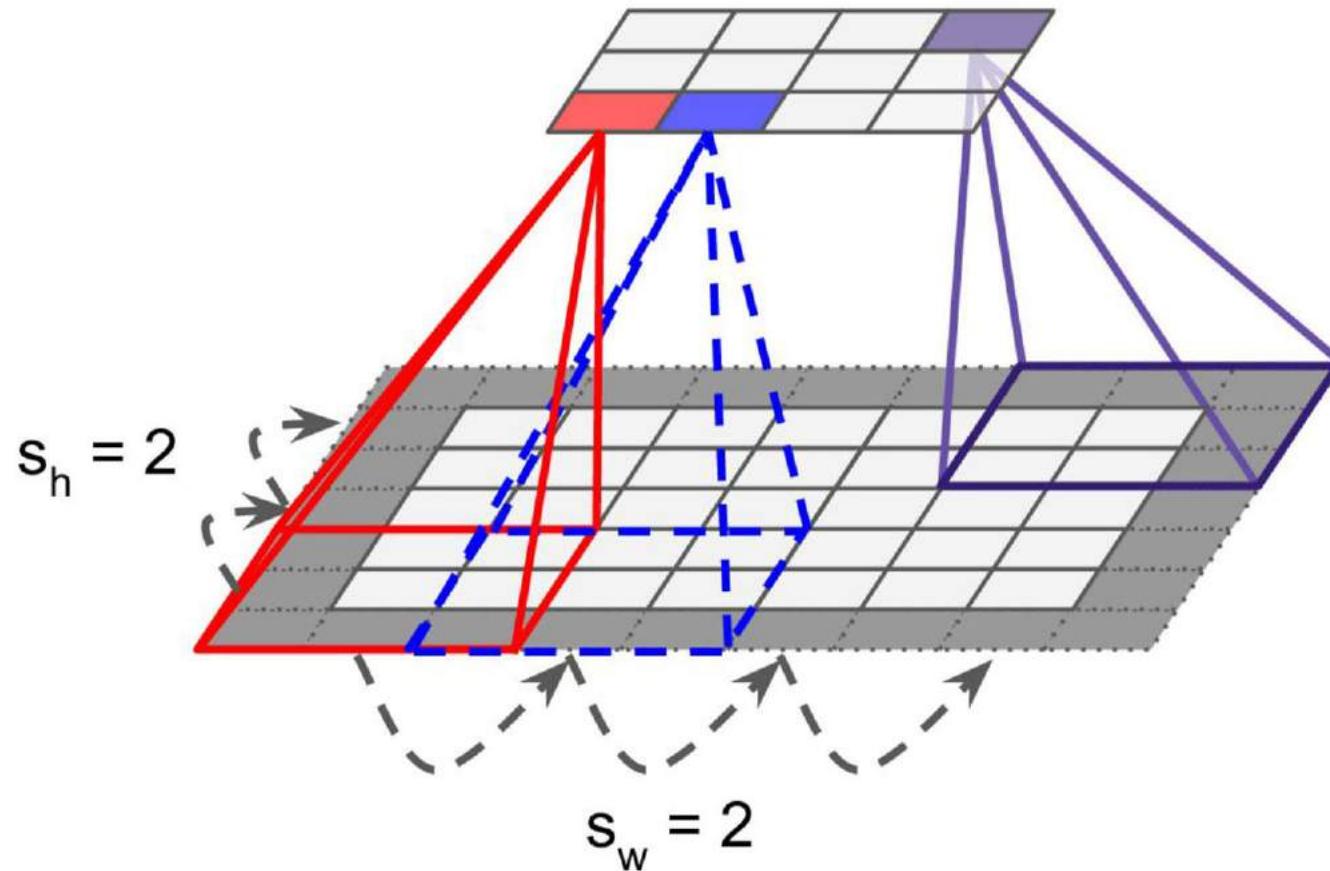
CNNs > Convolutional Layers



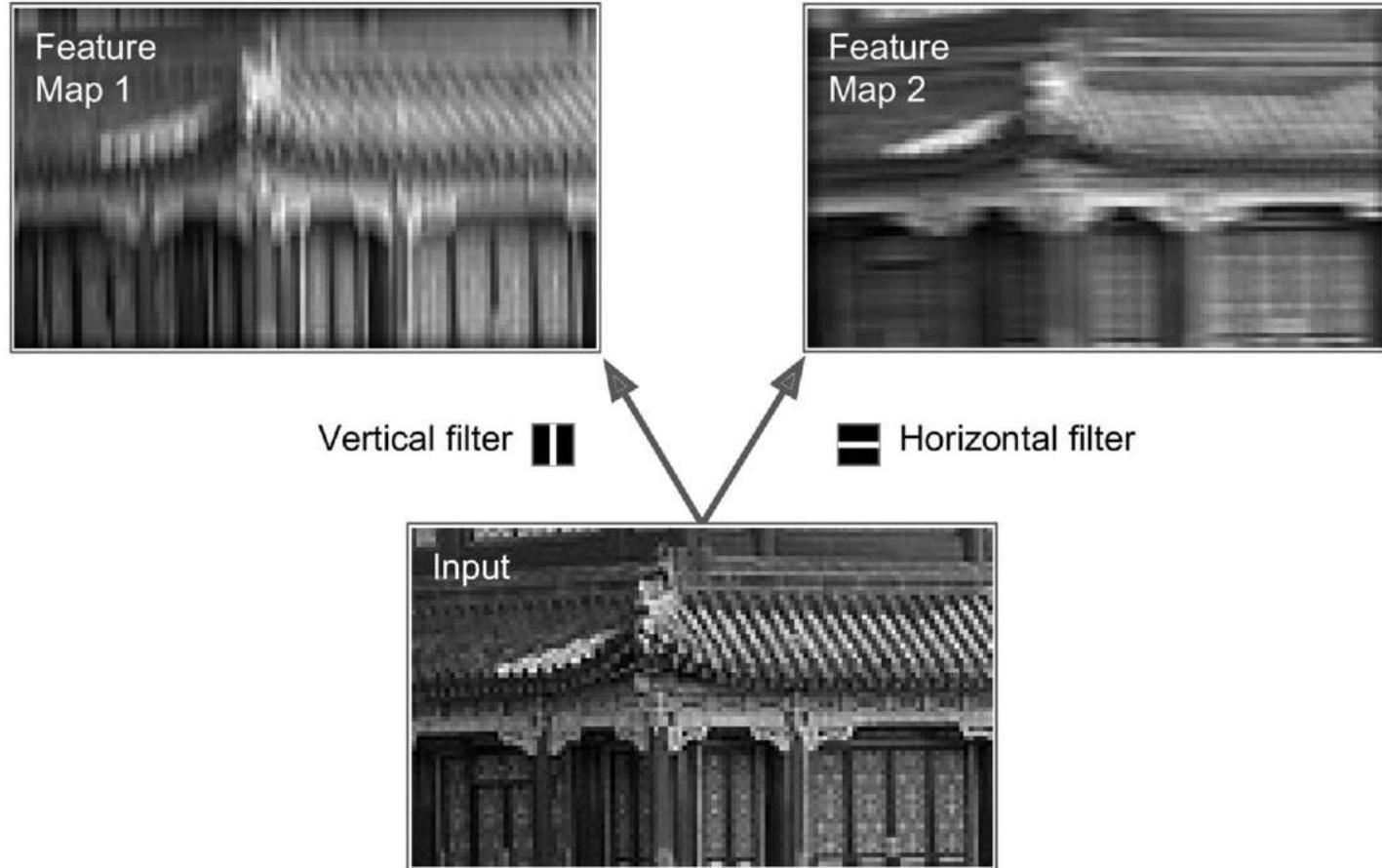
CNNs > Receptive Field Size & Padding



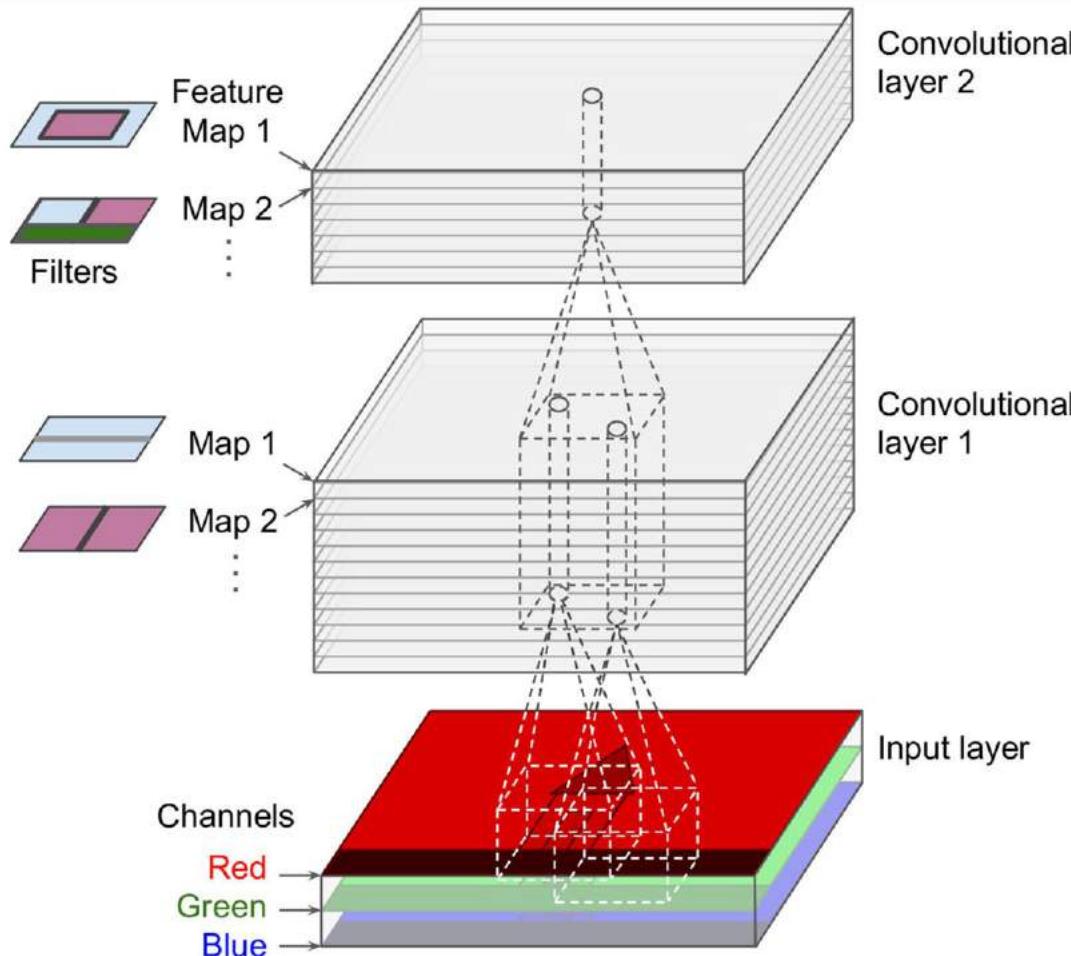
CNNs > Stride



CNNs > Feature Maps



CNNs > Convolutional Layers



CNNs > Using `tf.layers.conv2d()`

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.placeholder(
...         tf.float32, shape=(None, height, width, channels))
...     convolution = tf.layers.conv2d(
...         x, filters=2, kernel_size=7, strides=(1,1),
...         padding="SAME", activation=tf.nn.relu,
...         name="conv1")
...     init = tf.global_variables_initializer()
```

CNNs > Loading and Preprocessing an Image

```
>>> from scipy.misc import imread  
>>> china = imread("./images/china.png")  
>>> china.shape  
(427, 640, 3)
```



CNNs > Loading and Preprocessing an Image

```
>>> from scipy.misc import imread  
>>> china = imread("./images/china.png")  
>>> china.shape  
(427, 640, 3)  
>>> image = china[150:220, 130:250].mean(axis=2).astype(np.float32)  
>>> image.shape  
(70, 120)  
>>> height, width = image.shape  
>>> channels = 1 # grayscale
```



CNNs >

Running the Convolutional Layer

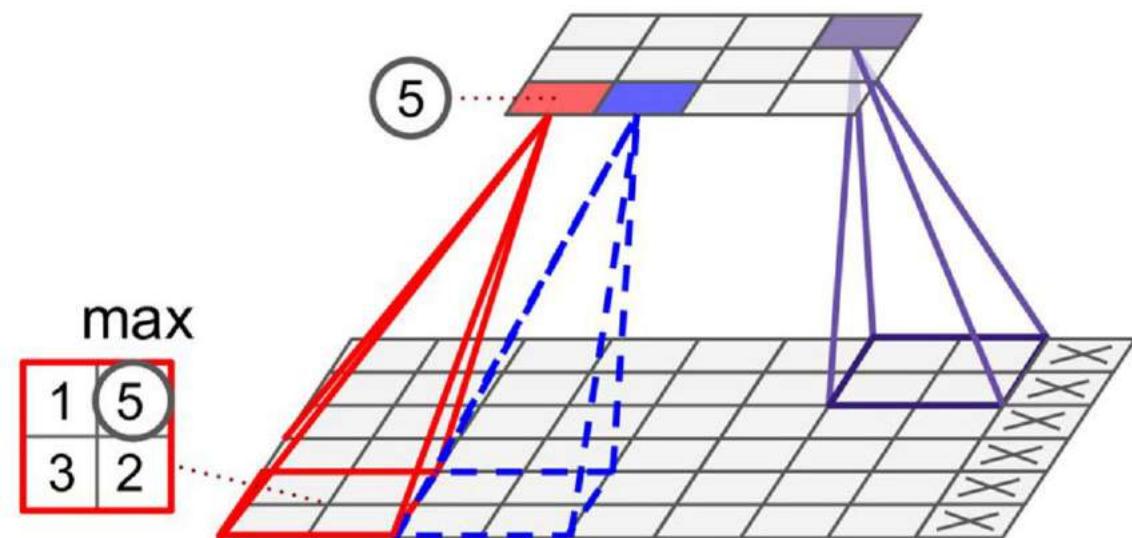
```
>>> X_batch = image.reshape(1, height, width, 1)
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     output = convolution.eval(feed_dict={X: X_batch})
...
```

CNNs > Running the Convolutional Layer

```
>>> X_batch = image.reshape(1, height, width, 1)
>>> with tf.Session(graph=graph) as sess:
...     init.run()
...     output = convolution.eval(feed_dict={X: X_batch})
...
>>> plt.imshow(output[0, :, :, 0], cmap="gray")
>>> plt.imshow(output[0, :, :, 1], cmap="gray")
```



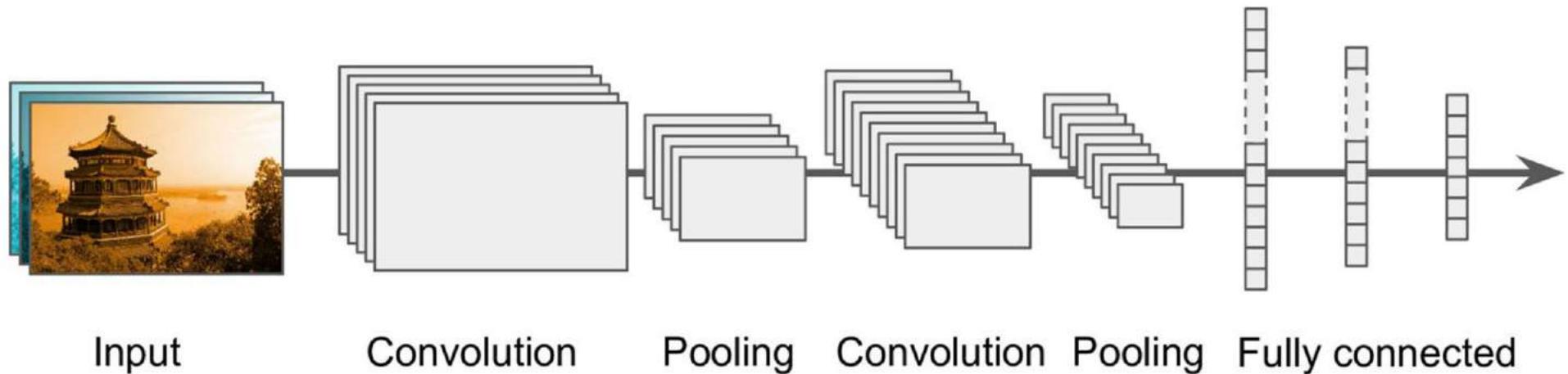
CNNs > Pooling Layers



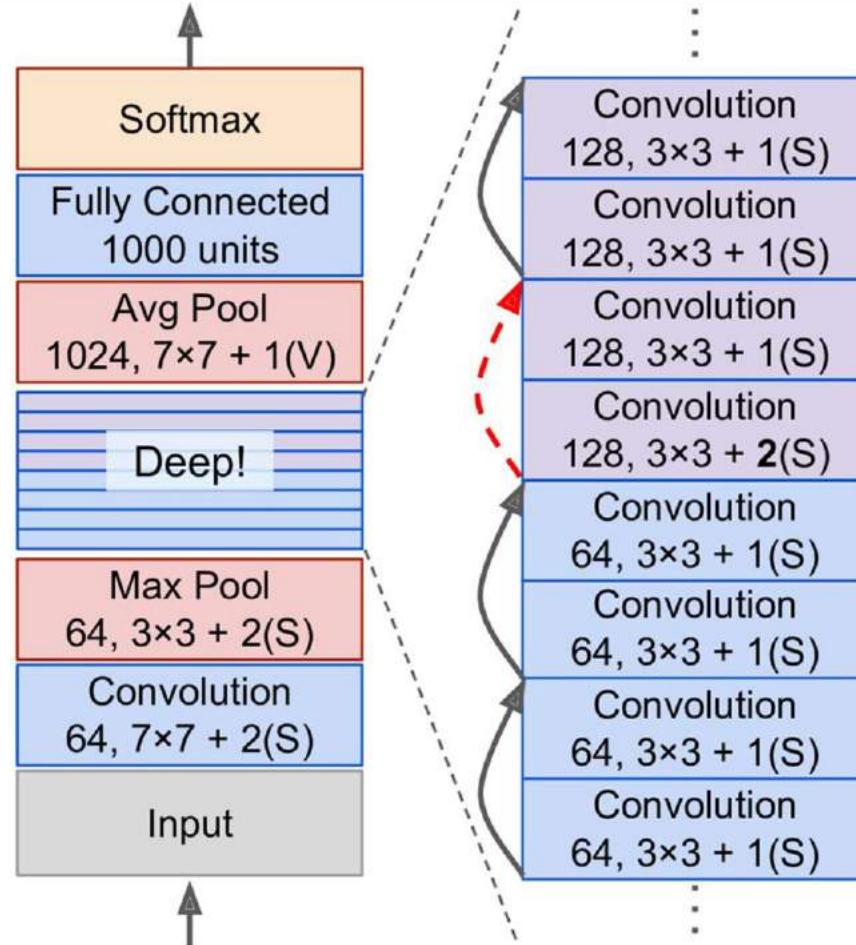
CNNs > Adding a Max Pooling Layer

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x = tf.placeholder(
...         tf.float32, shape=(None, height, width, channels))
...     convolution = tf.layers.conv2d(x, ...)
...     max_pool = tf.nn.max_pool(convolution,
...                               ksize=[1, 2, 2, 1],
...                               strides=[1, 2, 2, 1],
...                               padding="VALID")
...     init = tf.global_variables_initializer()
```

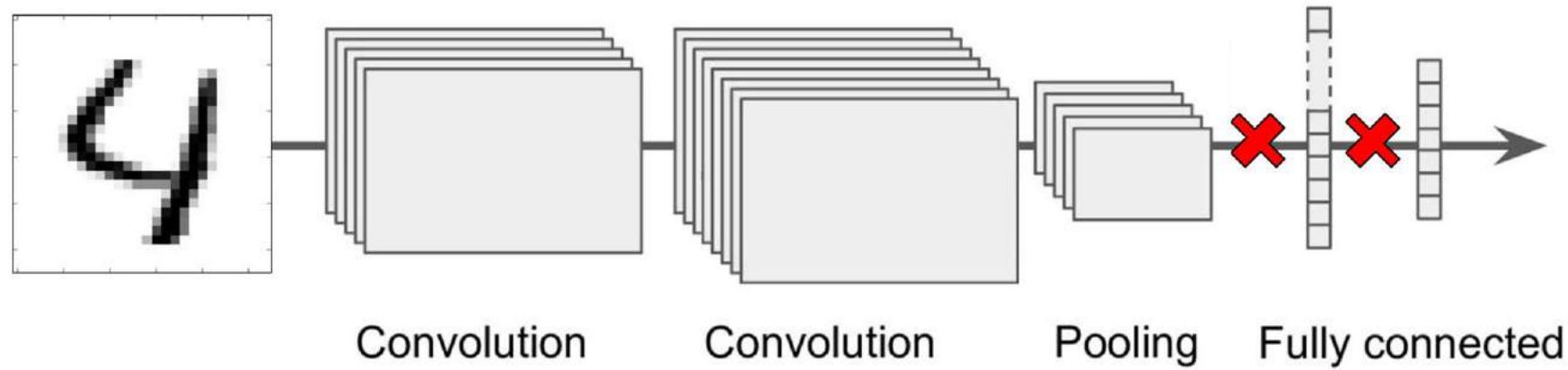
CNNs > Model Architecture



CNNs > ResNet Architecture



CNNs > Model Architecture



CNNs > Tackling MNIST with a CNN > Difficulties

```
>>> X = tf.placeholder(tf.float32, shape=[None, n_inputs])  
>>> X_reshaped = tf.reshape(X, shape=[-1, 28, 28, 1])
```

CNNs > Tackling MNIST with a CNN > Difficulties

```
>>> X = tf.placeholder(tf.float32, shape=[None, n_inputs])
>>> X_reshaped = tf.reshape(X, shape=[-1, 28, 28, 1])

>>> conv1 = tf.layers.conv2d(
...     X_reshaped, filters=32, kernel_size=3, strides=1,
...     padding="SAME", activation=tf.nn.relu, name="conv1")
```

CNNs > Tackling MNIST with a CNN > Difficulties

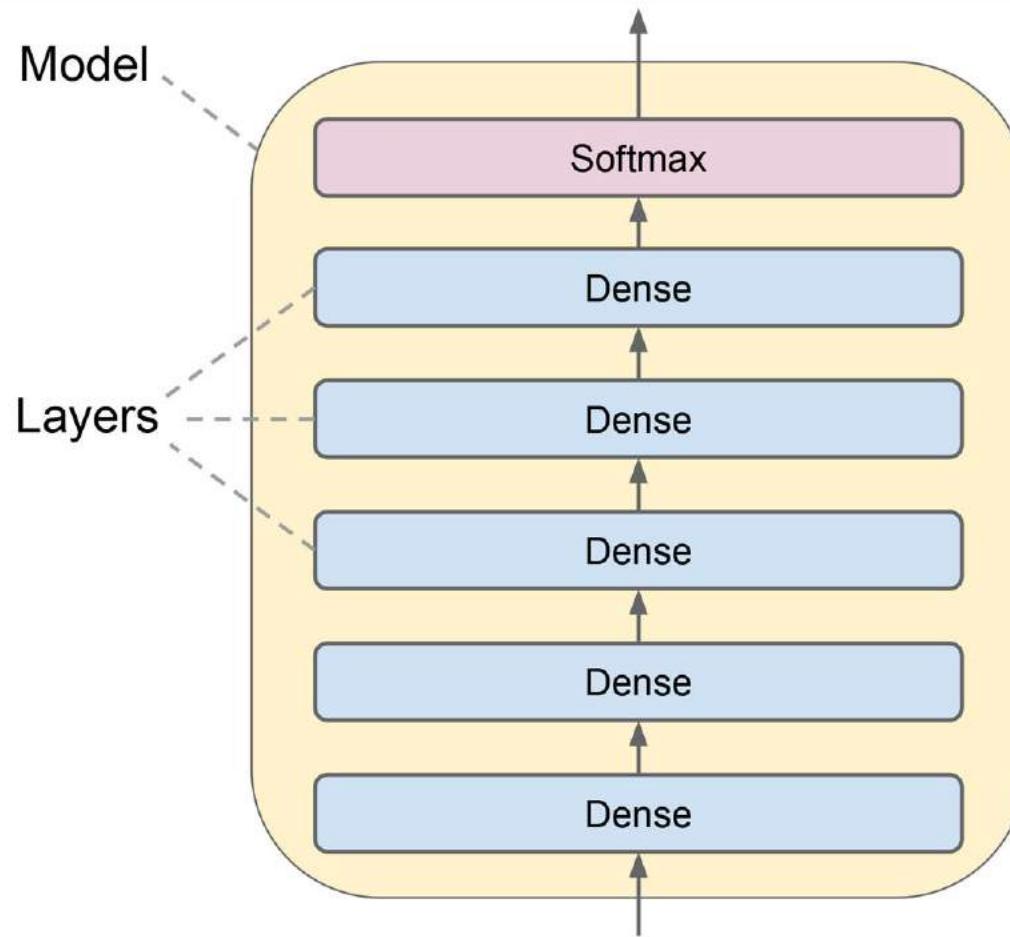
45 minutes
(homework)

Exercise 10



Keras

Keras > Sequential API



Keras > Sequential API

```
>>> import keras
```

Keras > Sequential API

```
>>> import keras  
>>> from keras.models import Sequential
```

Keras > Sequential API

```
>>> import keras  
>>> from keras.models import Sequential  
>>> from keras.layers import Dense
```

Keras > Sequential API

```
>>> import keras
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> model = Sequential()
```

Keras > Sequential API

```
>>> import keras
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> model = Sequential()
>>> model.add(Dense(100))
```

Keras > Sequential API

```
>>> import keras
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> model = Sequential()
>>> model.add(Dense(100, activation="selu"))
```

Keras > Sequential API

```
>>> import keras
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> model = Sequential()
>>> model.add(Dense(100, activation="selu", input_dim=784))
```

Keras > Sequential API

```
>>> import keras
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> model = Sequential()
>>> model.add(Dense(100, activation="selu", input_dim=784))
>>> model.add(Dense(100, activation="selu"))
>>> model.add(Dense(100, activation="selu"))
>>> model.add(Dense(100, activation="selu"))
>>> model.add(Dense(10), activation="softmax"))
```

Keras > Sequential API

```
>>> import keras
>>> from keras.models import Sequential
>>> from keras.layers import Dense, Softmax
>>> model = Sequential()
>>> model.add(Dense(100, activation="selu", input_dim=784))
>>> model.add(Dense(100, activation="selu"))
>>> model.add(Dense(100, activation="selu"))
>>> model.add(Dense(100, activation="selu"))
>>> model.add(Dense(10))
>>> model.add(Softmax())
```

Keras > Sequential API

```
>>> model.compile(loss="categorical_crossentropy",
...                  optimizer="SGD")
```

Keras > Sequential API

Keras > Sequential API

```
>>> model.compile(loss="categorical_crossentropy",
...                  optimizer="SGD", metrics=["accuracy"])

...
>>> model.fit(mnist.train.images, mnist.train.labels, epochs=50)
```

Keras > Sequential API

```
>>> model.compile(loss="categorical_crossentropy",
...                     optimizer="SGD", metrics=["accuracy"])

...
>>> model.fit(mnist.train.images, mnist.train.labels, epochs=50)
Epoch 1/50
55000/55000 [=====] - 5s 85us/step - loss: 0.3948 - acc: 0.8856
Epoch 2/50
55000/55000 [=====] - 4s 71us/step - loss: 0.2447 - acc: 0.9283
...
Epoch 50/50
55000/55000 [=====] - 4s 67us/step - loss: 0.0038 - acc: 0.9999
```

Keras > Sequential API

```
>>> model.evaluate(mnist.test.images, mnist.test.labels)
10000/10000 [=====] - 1s 54us/step
[0.09420637757958539, 0.9776]
```

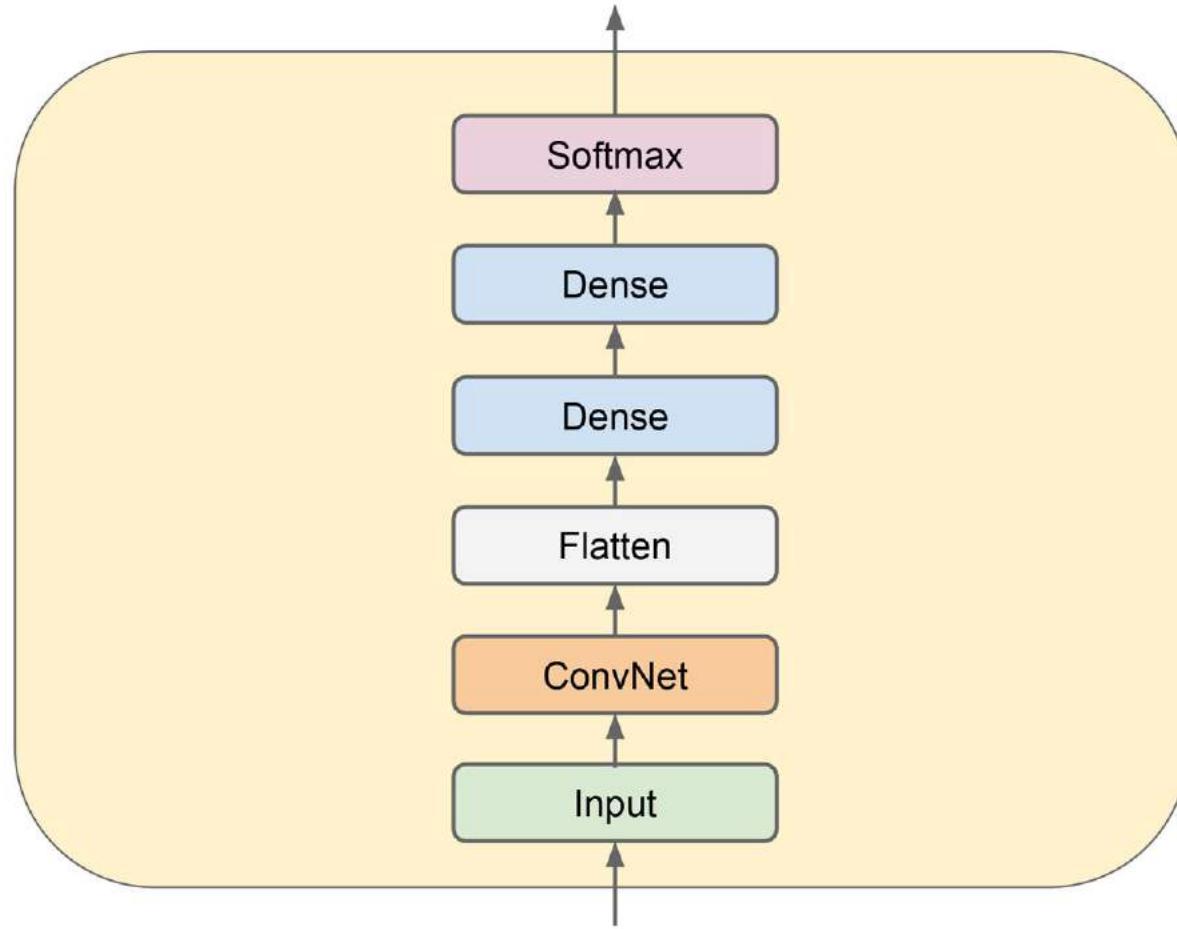
Keras > Sequential API

```
>>> model.evaluate(mnist.test.images, mnist.test.labels)
10000/10000 [=====] - 1s 54us/step
[0.09420637757958539, 0.9776]
>>> y_proba = model.predict(mnist.test.images)
```

Keras > Sequential API

```
>>> model.evaluate(mnist.test.images, mnist.test.labels)
10000/10000 [=====] - 1s 54us/step
[0.09420637757958539, 0.9776]
>>> y_proba = model.predict(mnist.test.images)
>>> y_proba.shape
(10000, 10)
```

Keras > Functional API



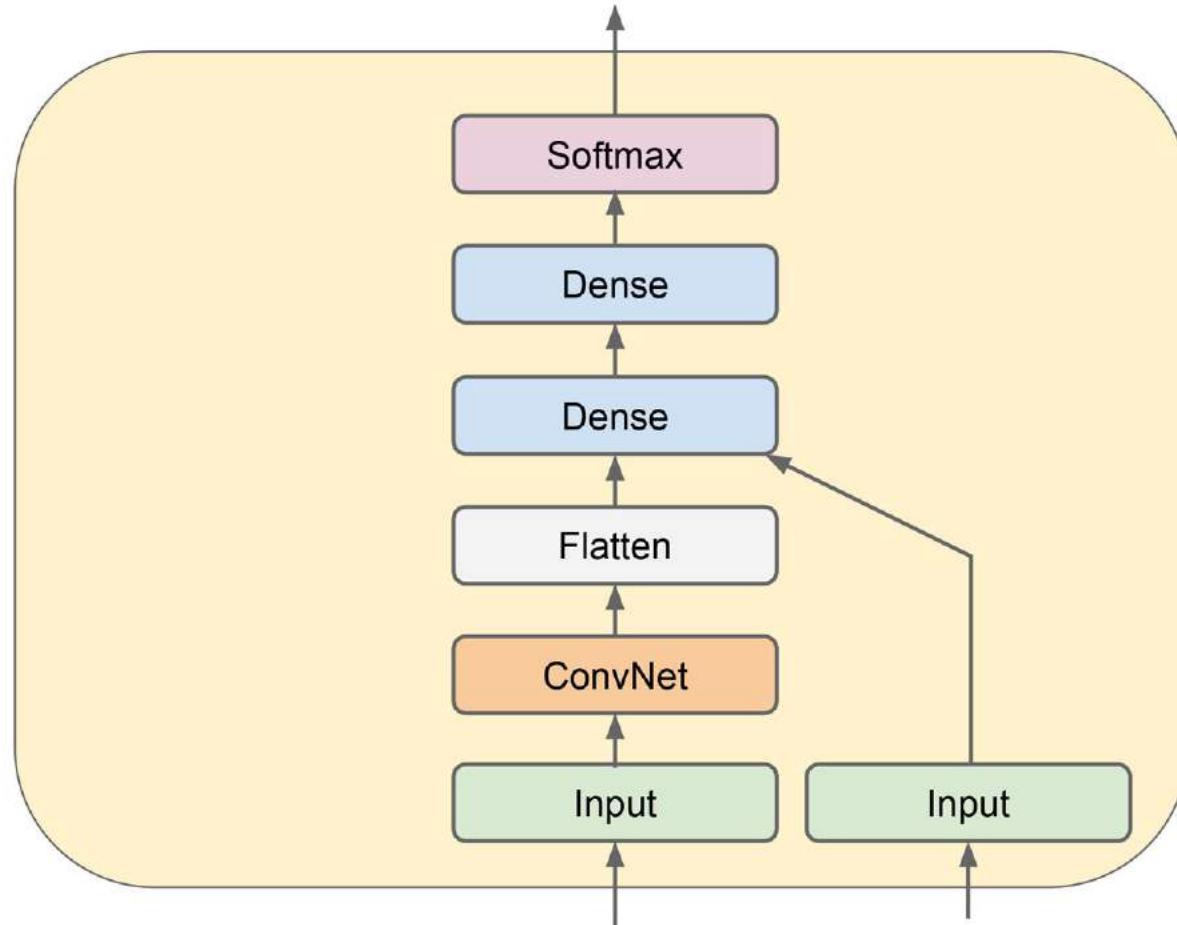
Keras > Functional API

```
>>> from keras.layers import Dense, Flatten, Input, Softmax  
>>> image = Input(shape=[100, 100, 3])  
>>> conv = my_conv_net_model(image)  
>>> flat = Flatten()(conv)  
>>> h1 = Dense(100, activation="selu")(flat)  
>>> h2 = Dense(5)(h1)  
>>> y_proba = Softmax()(h2)
```

Keras > Functional API

```
>>> from keras.models import Model  
>>> from keras.layers import Dense, Flatten, Input, Softmax  
>>> image = Input(shape=[100, 100, 3])  
>>> conv = my_conv_net_model(image)  
>>> flat = Flatten()(conv)  
>>> h1 = Dense(100, activation="selu")(flat)  
>>> h2 = Dense(5)(h1)  
>>> y_proba = Softmax()(h2)  
>>> model = Model(inputs=image, outputs=y_proba)
```

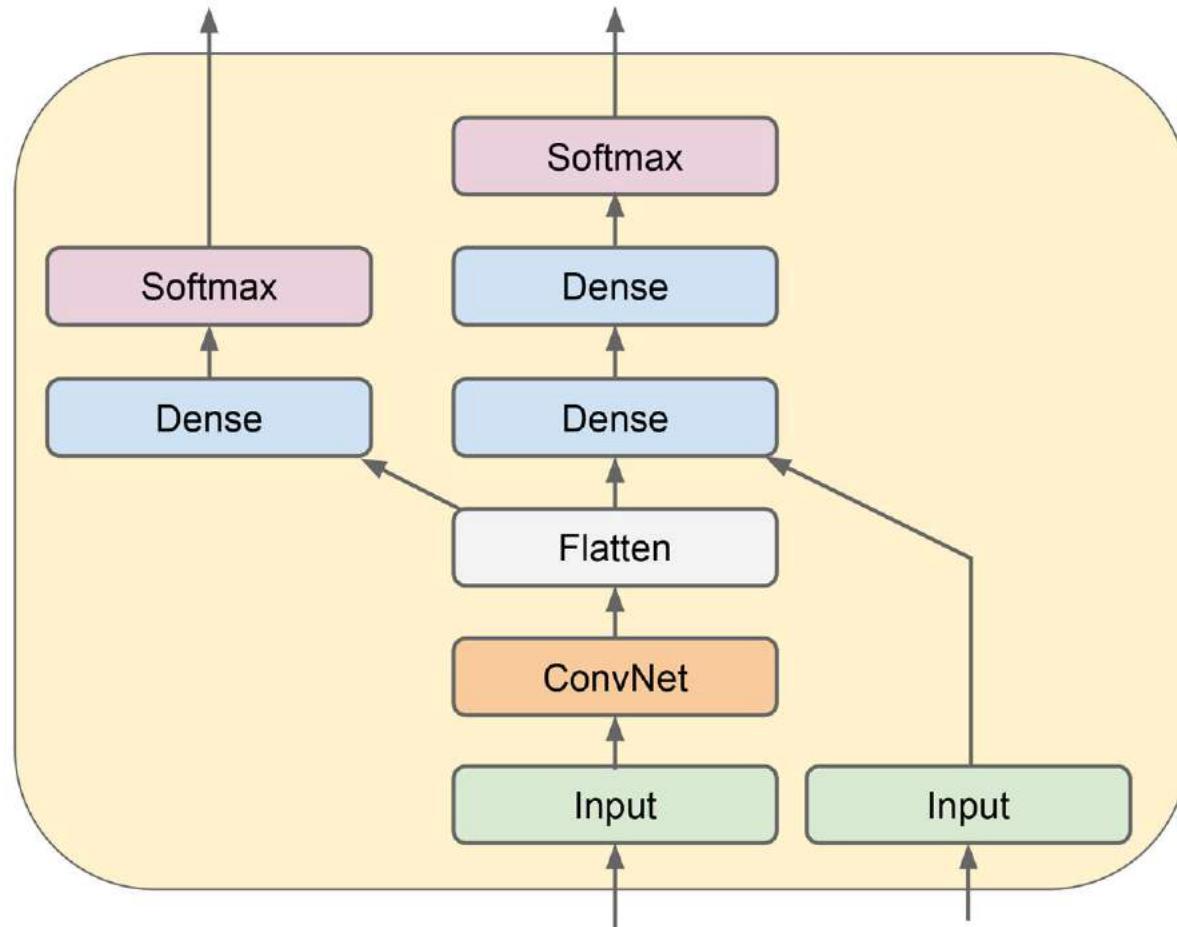
Keras > Functional API



Keras > Functional API

```
>>> from keras.models import Model  
>>> from keras.layers import Dense, Flatten, Input, Softmax  
>>> image = Input(shape=[100, 100, 3])  
>>> conv = my_conv_net_model(image)  
>>> flat = Flatten()(conv)  
>>> extra_input = Input(shape=[10])  
>>> z = keras.layers.concatenate([flat, extra_input])  
>>> h1 = Dense(100, activation="selu")(z)  
>>> h2 = Dense(5)(h1)  
>>> y_proba = Softmax()(h2)  
>>> model = Model(inputs=[image, extra_input], outputs=y_proba)
```

Keras > Functional API



Keras > Functional API

```
>>> # imports
>>> image = Input(shape=[100, 100, 3])
>>> conv = my_conv_net_model(image)
>>> flat = Flatten()(conv)
>>> extra_input = Input(shape=[10])
>>> z = keras.layers.concatenate([flat, extra_input])
>>> h1 = Dense(100, activation="selu")(z)
>>> h2 = Dense(5)(h1)
>>> y_proba = Softmax()(h2)
>>> aux_output = Dense(5, activation="softmax")(flat)
>>> model = Model(inputs=[image, extra_input],
...                  outputs=[y_proba, aux_output])
```

Keras > Functional API

```
>>> model.compile(optimizer='sgd',  
...                 loss='categorical_crossentropy',  
...                 loss_weights=[1., 0.1])  
...
```

Keras > Functional API

```
>>> model.compile(optimizer='sgd',  
...                  loss='categorical_crossentropy',  
...                  loss_weights=[1., 0.1])  
...  
>>> model.fit([image_data, extra_data], [labels, labels],  
...             epochs=100, batch_size=50)  
...
```



Search docs

Home

Why use Keras

Getting started

Guide to the Sequential model

Guide to the Functional API

Getting started with the Keras functional API

First example: a densely-connected network

All models are callable, just like layers

Multi-input and multi-output models

Shared layers

The concept of layer "node"

More examples

Docs » Getting started » Guide to the Functional API

Edit on GitHub

Getting started with the Keras functional API

The Keras functional API is the way to go for defining complex models, such as multi-output models, directed acyclic graphs, or models with shared layers.

This guide assumes that you are already familiar with the `Sequential` model.

Let's start with something simple.

First example: a densely-connected network

The `Sequential` model is probably a better choice to implement such a network, but it helps to start with something really simple.

- A layer instance is callable (on a tensor), and it returns a tensor
- `Input tensor(s)` and `output tensor(s)` can then be used to define a `Model`