HYU 한양대학교 HANYANG UNIVERSITY

한양대학교 ERICA 캠퍼스
Education Research Industry Cluster @ Ansan

딥러닝 심화과정 Day 1

Hanyang University ERICA
AI Labs

이상근

2017. 11. 7

ETRI

강의 소개

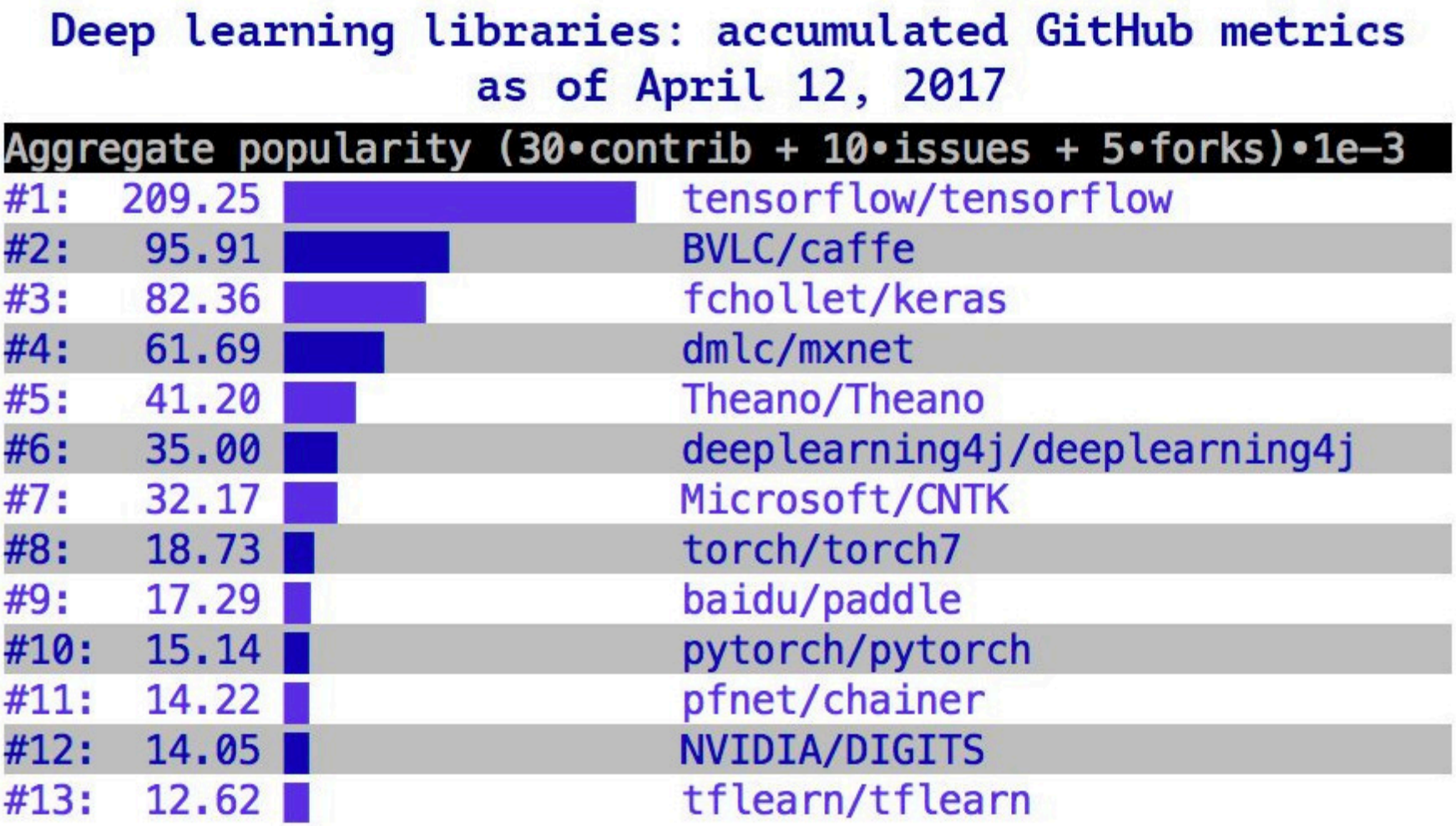
제목: 딥러닝 심화과정

내용: Tensorflow를 이용한 딥러닝 심화 학습

목표:

- Day 1: 딥러닝을 위한 Tensorflow의 고급 기능 이해
- Day 2: 고급 CNN 실습: ImageNet 기반 CNN 재학습 / 사용 실습
- Day 3: 고급 RNN 실습: Word Embedding 학습과 Sentiment Analysis

Deep Learning Frameworks



https://twitter.com/fchollet

Deep Learning Frameworks in 2017

Caffe
(UC Berkeley) → Caffe2
(Facebook)

Torch
(NYU / Facebook) → PyTorch
(Facebook)

Theano
(U Montreal) → TensorFlow
(Google)
v1.4

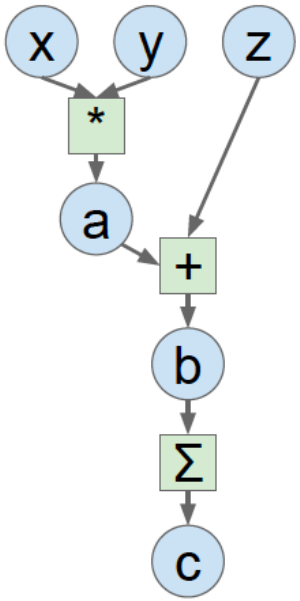
Paddle
(Baidu)

CNTK
(Microsoft)

MXNet
(Amazon)
Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

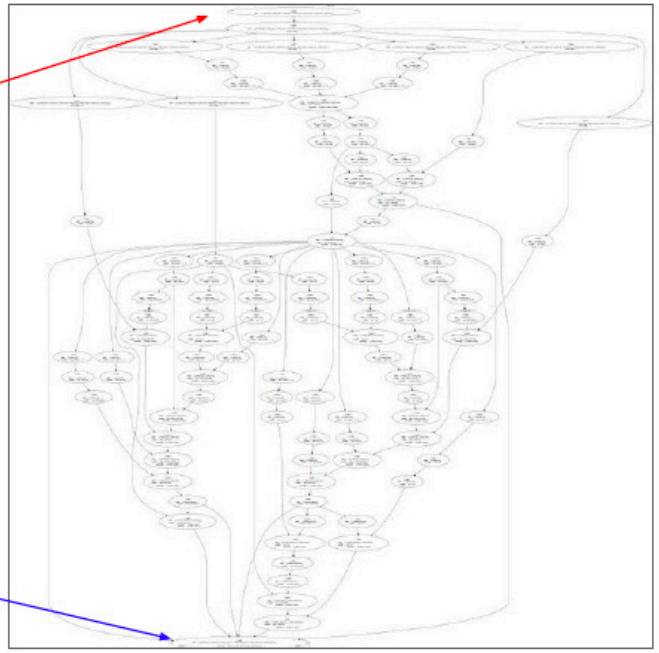
And others...

The Points of DL Frameworks



input image

loss



(1) Easily build big computational graphs
(2) Easily compute gradients in computational graphs
(3) Run it all efficiently on GPU (wrap cuDNN, cuBLAS, etc)

DL in Mobile Platform

CoreML (Apple)

- Optimized for Apple hardwares

Caffe2go (Facebook)

- Can be optimized for GPU servers, Android, and iOS
- Backend: NEON, CoreML, CUDA, …

Tensorflow Lite (Google)

- Optimized for a special hardware ?

Caffe2go (2016)

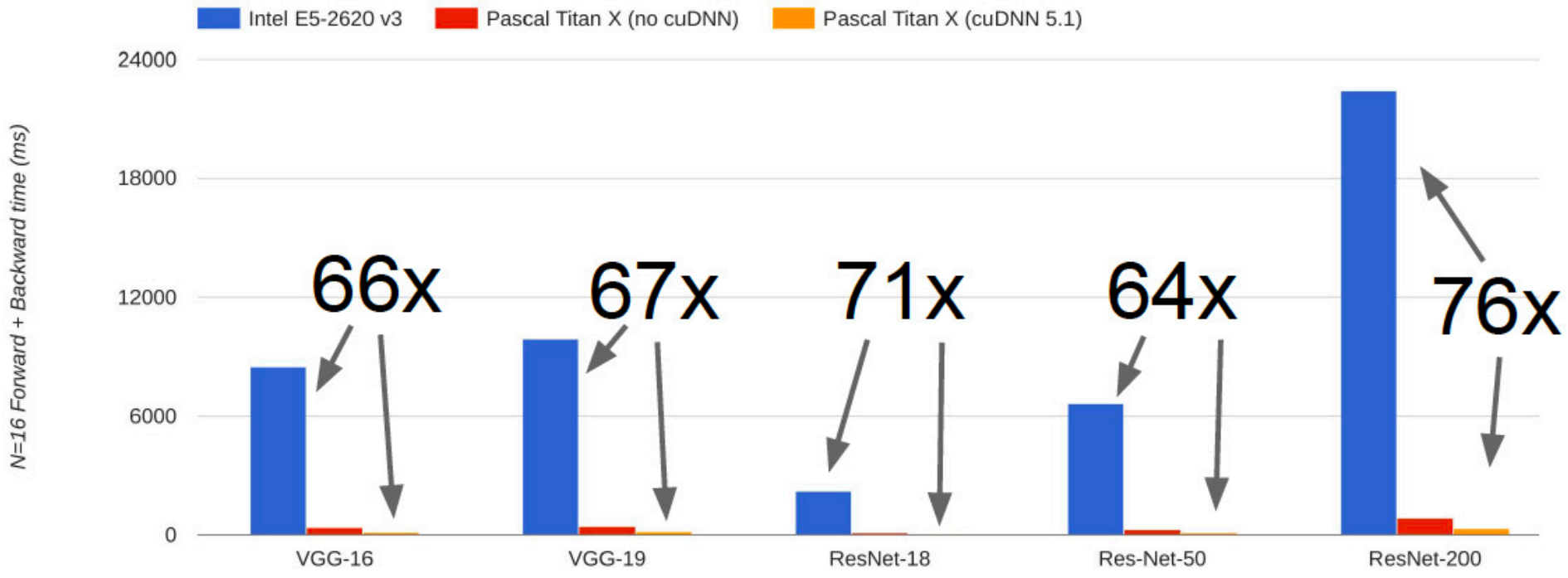
Optimization for ***style transfer*** application:

(at least for now)

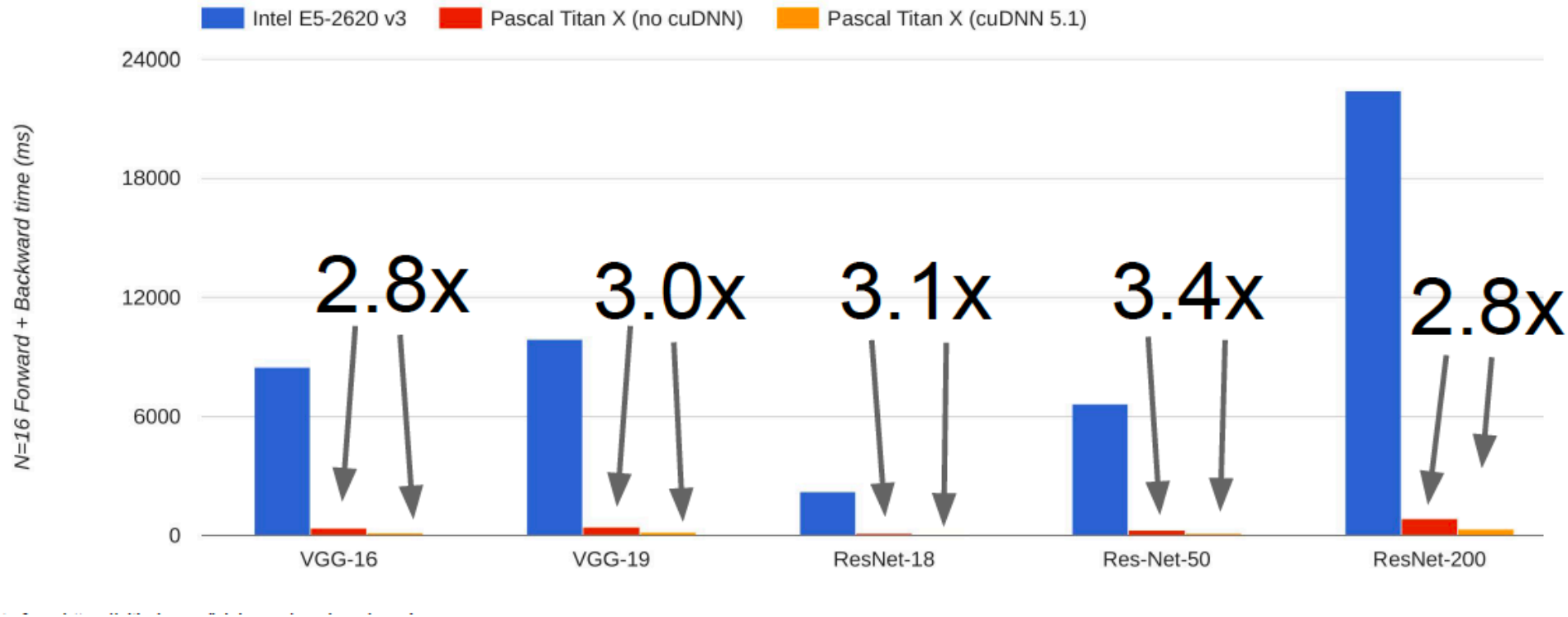
- # of convolution layers
- # of units in each conv layer
- Downsampling before convolution
- …

TensorFlow + GPU

CPU vs. GPU


Intel E5-2620 v3
Pascal Titan X (no cuDNN)
Pascal Titan X (cuDNN 5.1)
N=16 Forward + Backward time (ms)
24000
18000
12000
6000
0
66x
67x
71x
64x
76x
VGG-16
VGG-19
ResNet-18
Res-Net-50
ResNet-200

CPU vs. GPU


Intel E5-2620 v3
Pascal Titan X (no cuDNN)
Pascal Titan X (cuDNN 5.1)
N=16 Forward + Backward time (ms)
24000
18000
12000
6000
0
2.8x
3.0x
3.1x
3.4x
2.8x
VGG-16
VGG-19
ResNet-18
Res-Net-50
ResNet-200

Tensorflow + GPU

Step 1. NVIDIA CUDA Toolkit (incl. gpu driver)

- https://developer.nvidia.com/cuda-toolkit
- Recommended: ver 8.0
- **Current version: ver 9.0**

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows, Linux, Mac OSX
Architecture	x86_64, ppc64le
Distribution	Fedora, OpenSUSE, RHEL, CentOS, SLES, Ubuntu
Version	17.04, 16.04
Installer Type	runfile (local), deb (local), deb (network)

Download Installer for Linux Ubuntu 17.04 x86_64

The base installer is available for download below.

Tensorflow + GPU

Step 2. cuDNN (https://developer.nvidia.com/cudnn)

- **Required: ver 6.0 (by Tensorflow 1.3)**
- Current: ver 7.0

Membership Required

The downloadable file or page you have requested, requires membership of the NVIDIA Developer Program. Please login to gain access or use the button below and complete the short application for this free to join program. Thank you. Join now

cuDNN

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

☑ **I Agree To the Terms of the cuDNN Software License Agreement**

Note: Please refer to the Installation Guide for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.

For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the Deep Learning SDK Documentation web page.

Download cuDNN v7.0.3 (Sept 28, 2017), for CUDA 9.0

Download cuDNN v7.0.3 (Sept 28, 2017), for CUDA 8.0

Download cuDNN v6.0 (April 27, 2017), for CUDA 8.0

Download cuDNN v6.0 (April 27, 2017), for CUDA 7.5

Download cuDNN v5.1 (Jan 20, 2017), for CUDA 8.0

Download cuDNN v5.1 (Jan 20, 2017), for

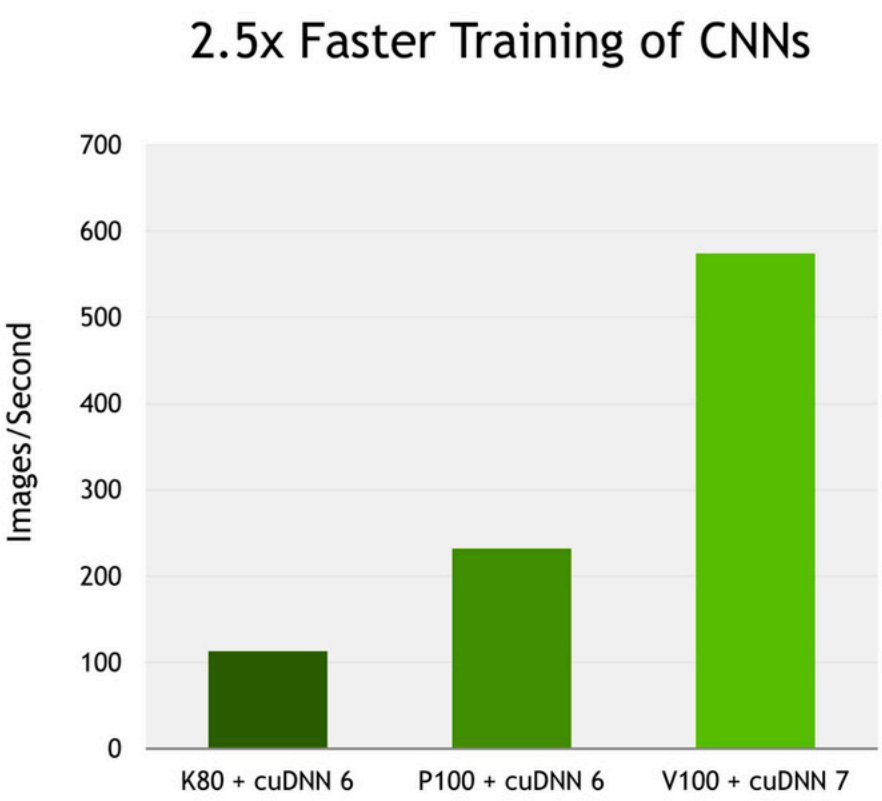
Archived cuDNN Releases

cuDNN v6.0 Runtime Library for Ubuntu16.04 (Deb)

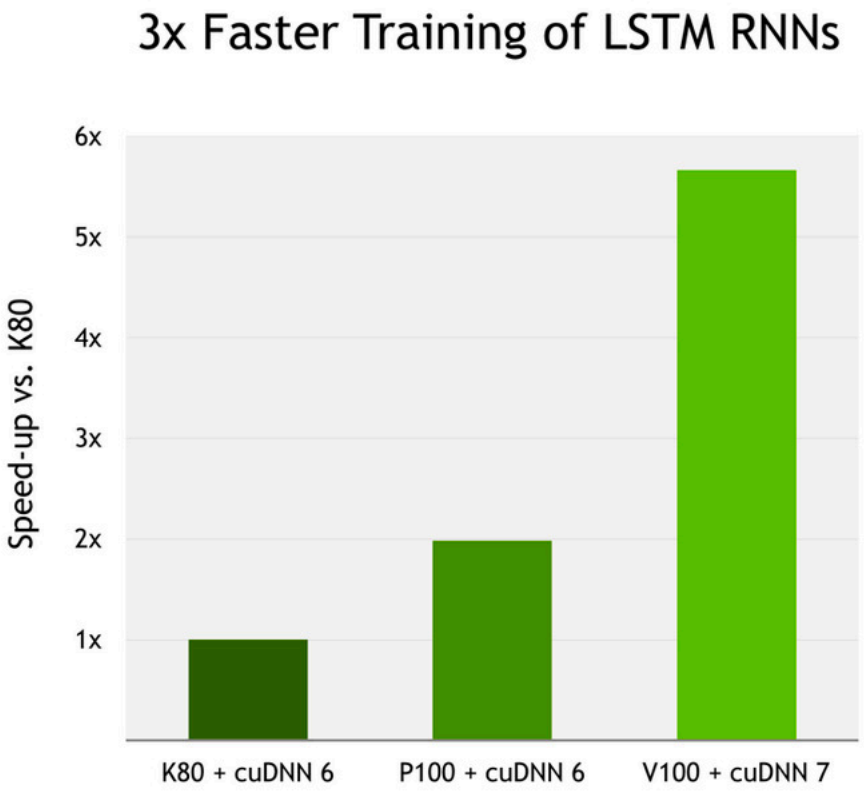
cuDNN v6.0 Developer Library for Ubuntu16.04 (Deb)

cuDNN v6.0 Code Samples and User Guide for Ubuntu16.04 (Deb)

cuDNN



Caffe2 performance (images/sec), Tesla K80 + cuDNN 6 (FP32), Tesla P100 + cuDNN 6 (FP32), Tesla V100 + cuDNN 7 (FP16, pre-release H/W and S/W). ResNet50, Batch size: 64



MXNet performance (min/epoch), Tesla K80 + cuDNN 6 (FP32), Tesla P100 + cuDNN 6 (FP32), Tesla V100 + cuDNN 7 (FP16, pre-release H/W and S/W). NMT seq2seq RNN (https://github.com/mkolod/mxnet_seq2seq)

Tensorflow + GPU

NVIDIA CUDA Profile Tools Interface

```
$ sudo apt-get install libcupti-dev
```

Install Tensorflow with GPU support

```
$ pip3 install tensorflow-gpu
```

TensorFlow + Jupyter Notebook

Steps 1/2

1. **Uninstall Jupyter**

   $ sudo pip3 uninstall jupyter

   $ pip3 uninstall jupyter

2. **Activate the virtual environment**

   $ source ~/tensorflow/bin/activate

3. **Install Jupyter within the virtual environment**

   (tensorflow) $ pip install jupyter

Steps 2/2

1. Uninstall Jupyter
2. Activate the virtual environment
3. Install Jupyter within the virtual environment
4. **Create a Jupyter notebook profile**

   (tensorflow) $ ipython kernelspec install-self --user

5. **Copy the profile**

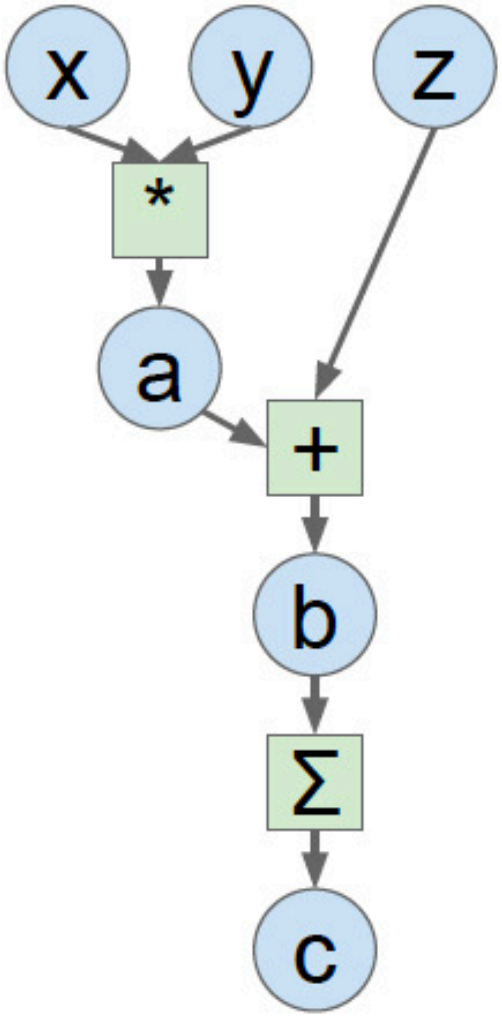
   $ mkdir -p ~/.ipython/kernels

   $ mv ~/.local/share/jupyter/kernels/python3 ~/.ipython/kernels/python3_tf

6. **Edit** ~/.ipython/kernels/<kernel_name>/kernel.json and display_name

TensorFlow

Tensorflow




```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Create Computation Graph

CPU vs GPU

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3000, 4000

with tf.device('/cpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x,

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_
                    feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_va
```

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3000, 4000

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                    feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Tensorflow: Two Parts

Define computation graph

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])
```

Run the graph many times

```
with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Placeholder

Create placeholders for input x, weights w1,w2, and targets y

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Computation Graph

Compute predictions for y and loss

No computation occurs here !!

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Computation Graph

Compute gradients:

No computation happens here!!

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Session

Enter a session:

we can actually r
un the graph

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Fill-in Data

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Fill-in placeholders

Run within a Session

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Run the graph

Training

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    learning_rate = 1e-5
    for t in range(50):
        out = sess.run([loss, grad_w1, grad_w2],
                       feed_dict=values)
        loss_val, grad_w1_val, grad_w2_val = out
        values[w1] -= learning_rate * grad_w1_val
        values[w2] -= learning_rate * grad_w2_val
```

Train the model : run the graph many times, updating weights

Training: Copying Problems

Placeholders are on GPU

Copying weights between CPU & GPU each step

'values' are on CPU

```
N, D, H = 64, 1000, 100
with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32, shape=(N, D))
    y = tf.placeholder(tf.float32, shape=(N, D))
    w1 = tf.placeholder(tf.float32, shape=(D, H))
    w2 = tf.placeholder(tf.float32, shape=(H, D))
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    learning_rate = 1e-5
    for t in range(50):
        out = sess.run([loss, grad_w1, grad_w2],
                       feed_dict=values)
        loss_val, grad_w1_val, grad_w2_val = out
        values[w1] -= learning_rate * grad_w1_val
        values[w2] -= learning_rate * grad_w2_val
```

Variables

Change w1 & w2 from placeholder to Variable (persistent in the graph between calls)

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((H, D)))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

learning_rate = 1e-5
new_w1 = w1.assign(w1 - learning_rate * grad_w1)
new_w2 = w2.assign(w2 - learning_rate * grad_w2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    for t in range(50):
        loss_val, = sess.run([loss], feed_dict=values)
```

Assignment Graph

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((H, D)))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

learning_rate = 1e-5
new_w1 = w1.assign(w1 - learning_rate * grad_w1)
new_w2 = w2.assign(w2 - learning_rate * grad_w2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    for t in range(50):
        loss_val, = sess.run([loss], feed_dict=values)
```

Add 'assign' operators to update w1 & w2, as part of the graph!!

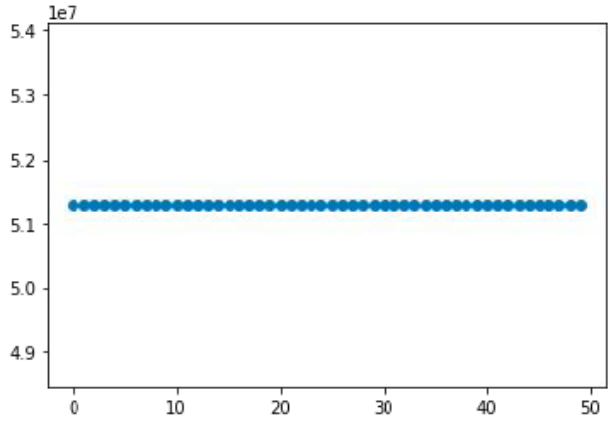
Run

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((H, D)))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

learning_rate = 1e-5
new_w1 = w1.assign(w1 - learning_rate * grad_w1)
new_w2 = w2.assign(w2 - learning_rate * grad_w2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    for t in range(50):
        loss_val, = sess.run([loss], feed_dict=values)
```

Run the graph once to initialize w1 & w2

Run many times to train

Run



Problem: loss not going down! Assign calls not actually being executed!

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
 = tf.Variable(tf.random_normal((H, D)))

 = tf.maximum(tf.matmul(x, w1), 0)
_pred = tf.matmul(h, w2)
ff = y_pred - y
ss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
ad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

arning_rate = 1e-5
w_w1 = w1.assign(w1 - learning_rate * grad_w1)
w_w2 = w2.assign(w2 - learning_rate * grad_w2)

th tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    for t in range(50):
        loss_val, = sess.run([loss], feed_dict=values)
```

Dummy Node

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((H, D)))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

learning_rate = 1e-5
new_w1 = w1.assign(w1 - learning_rate * grad_w1)
new_w2 = w2.assign(w2 - learning_rate * grad_w2)
updates = tf.group(new_w1, new_w2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    losses = []
    for t in range(50):
        loss_val, _ = sess.run([loss, updates],
                               feed_dict=values)
```

Add a dummy graph node that depends on updates

Compute the dummy node

Optimizer

We can use 'optimizer' to compute gradients

Execute the output of the optimizer

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((H, D)))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff * diff, axis=1))

optimizer = tf.train.GradientDescentOptimizer(1e-5)
updates = optimizer.minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    losses = []
    for t in range(50):
        loss_val, _ = sess.run([loss, updates],
                               feed_dict=values)
```

Loss

Predefined loss functions

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((H, D)))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
loss = tf.losses.mean_squared_error(y_pred, y)

optimizer = tf.train.GradientDescentOptimizer(1e-3)
updates = optimizer.minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    for t in range(50):
        loss_val, _ = sess.run([loss, updates],
                               feed_dict=values)
```

Initialization / tf.layers

Xavier initializer

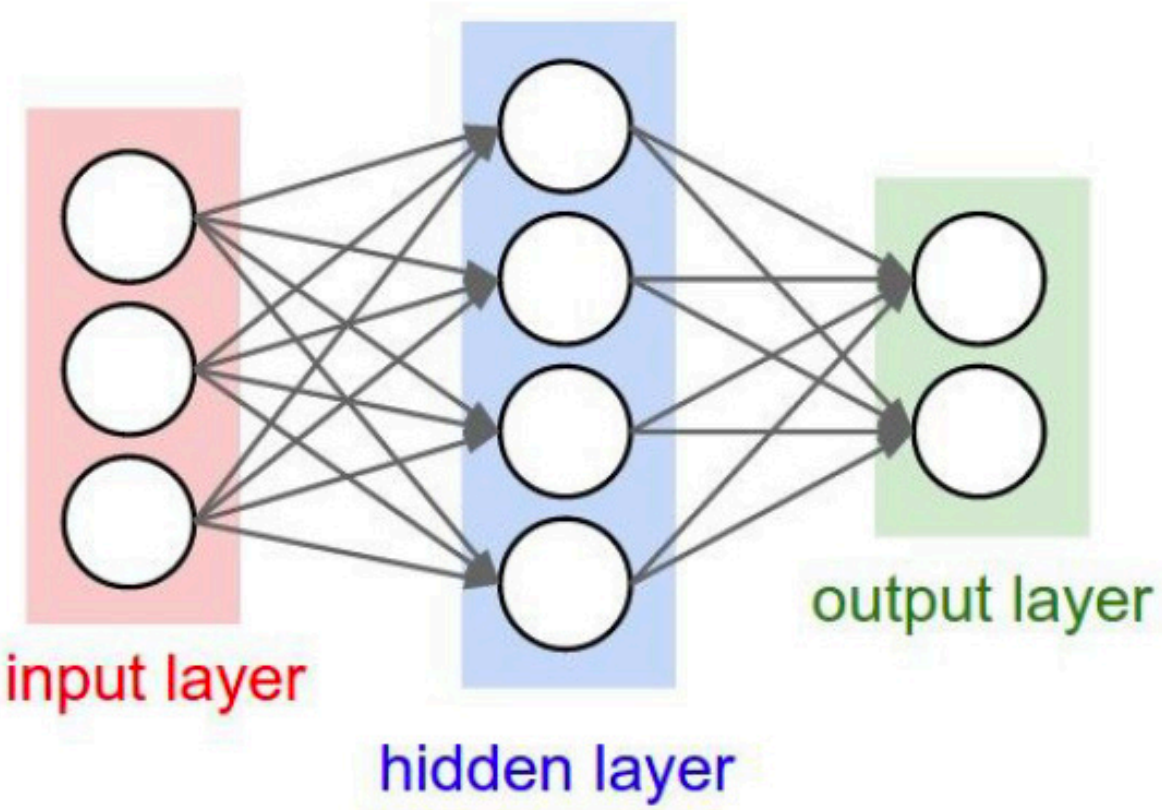
tf.layers automatically sets up weights and biases

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))

init = tf.contrib.layers.xavier_initializer()
h = tf.layers.dense(inputs=x, units=H,
        activation=tf.nn.relu, kernel_initializer=init)
y_pred = tf.layers.dense(inputs=h, units=D,
        kernel_initializer=init)

loss = tf.losses.mean_squared_error(y_pred, y)

optimizer = tf.train.GradientDescentOptimizer(1e0)
updates = optimizer.minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    for t in range(50):
        loss_val, _ = sess.run([loss, updates],
                               feed_dict=values)
```

Initialization

DNN

What happens if we initialize all weights to the zero value?



Small Random Numbers

Gaussian with mean 0 and std 0.01

```
W = 0.01* np.random.randn(D,H)
```

Works okay for small networks, but problematic for deeper networks

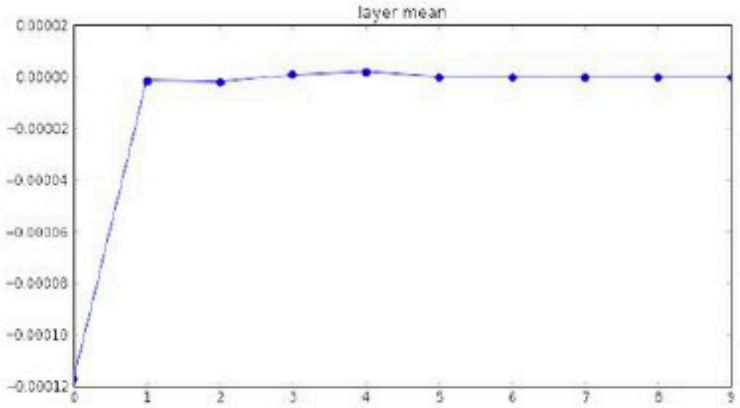
Activation Statisitics

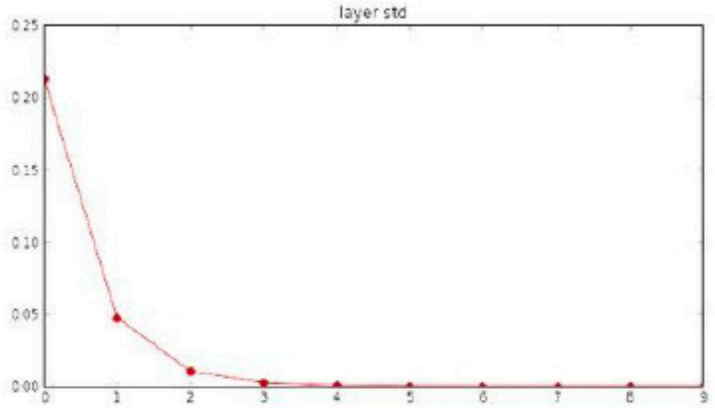
```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```

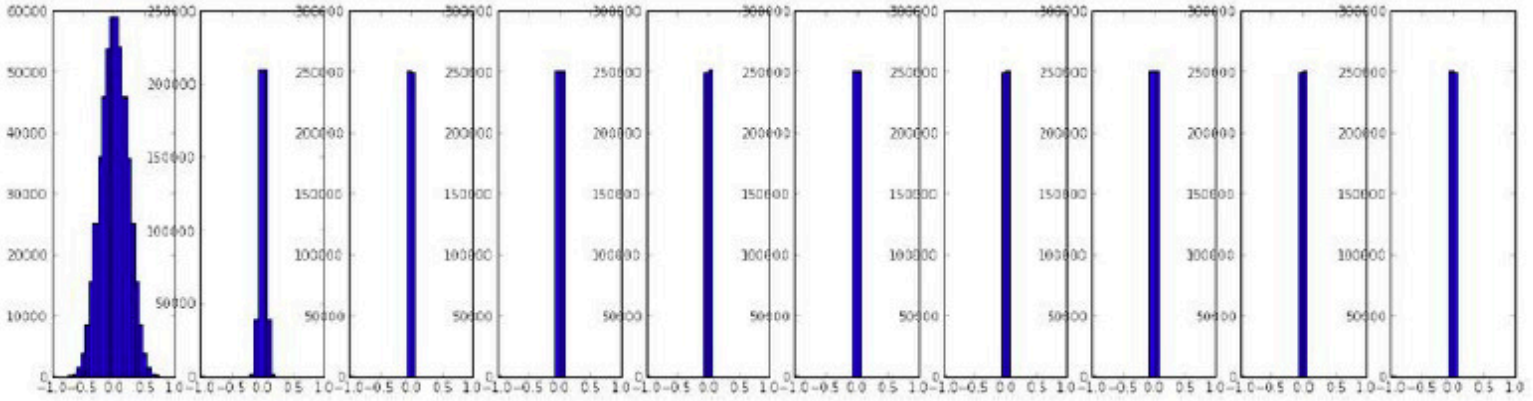
- 10-layer DNN
- 500 units / layer
- Activation: tanh

All activations becomes zero!!

How'd the gradients look like?







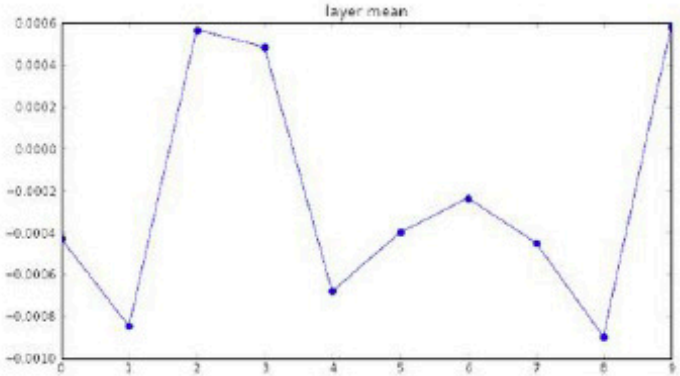
```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

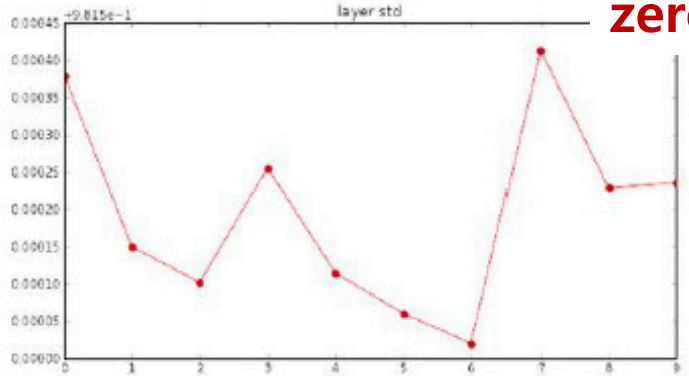
```
input layer had mean 0.001880 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

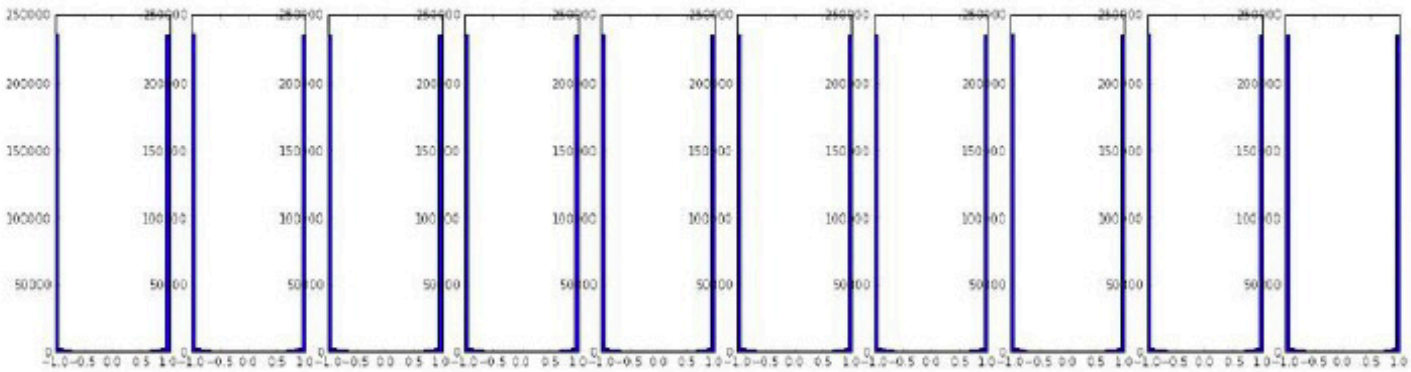
*1.0 instead of *0.01

Almost all neurons a re saturated, either -1 or +1

Gradients will be zero





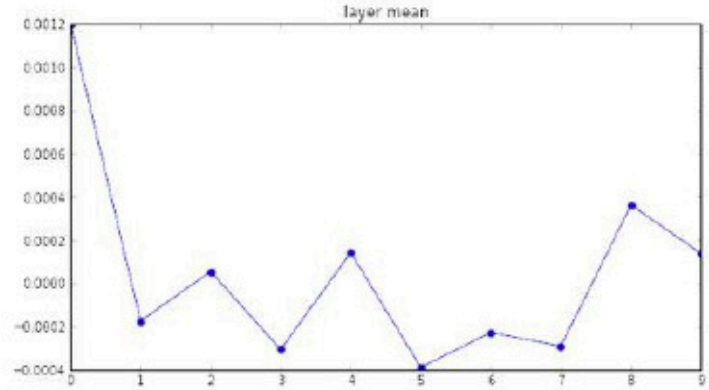


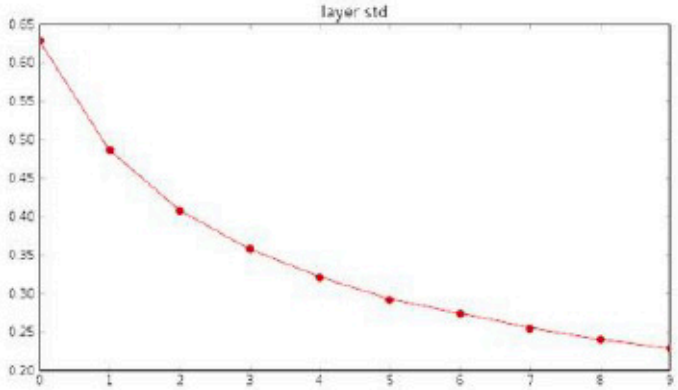
Xavier Initialization

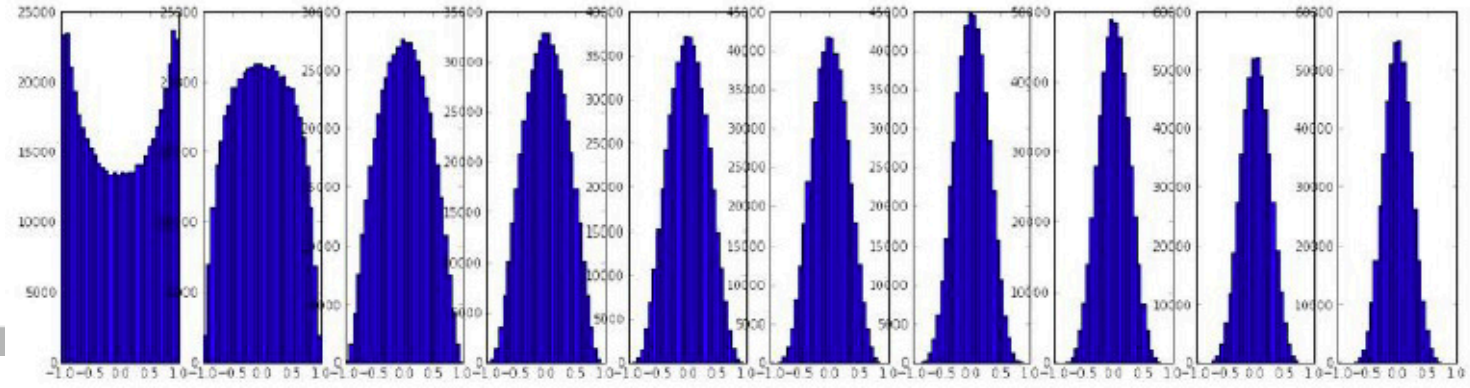
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

Mathematical derivation assumes linear activations



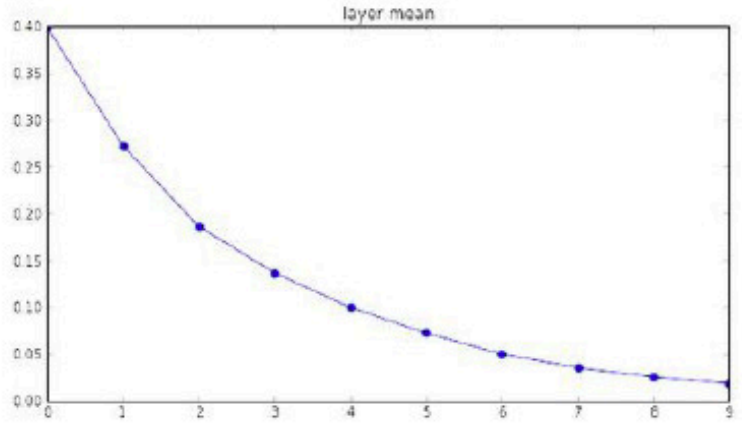


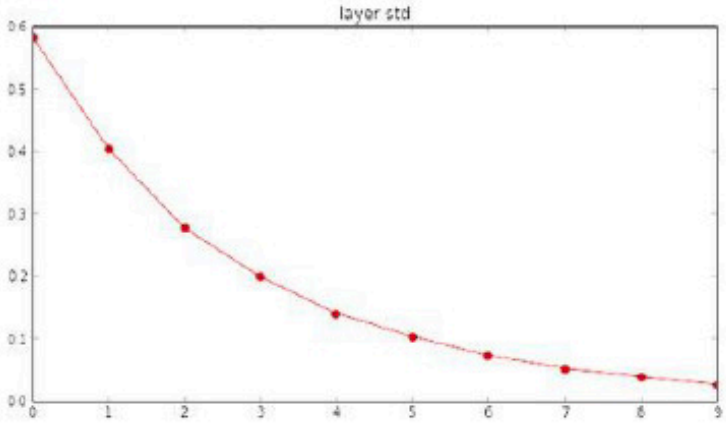


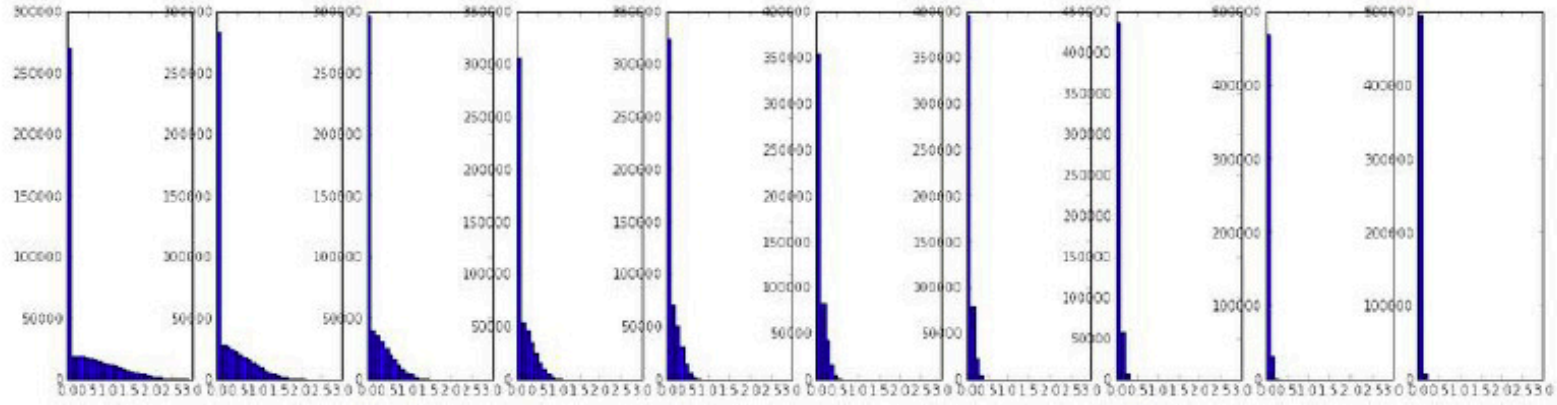
Xavier Initialization

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```

Doesn't work as good with ReLU



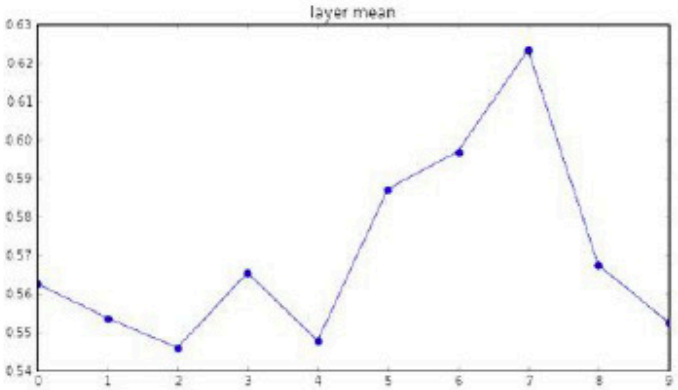


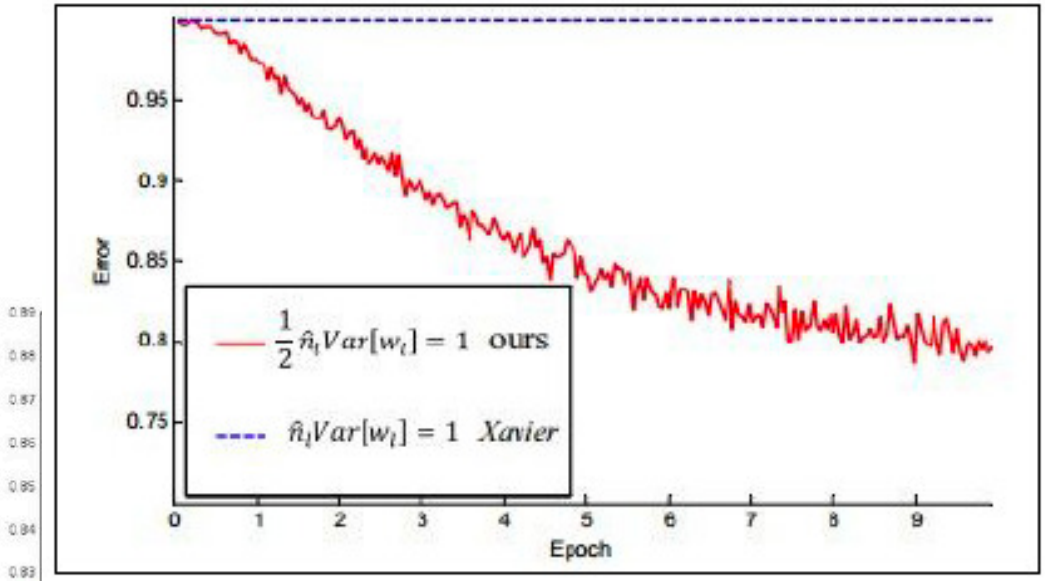


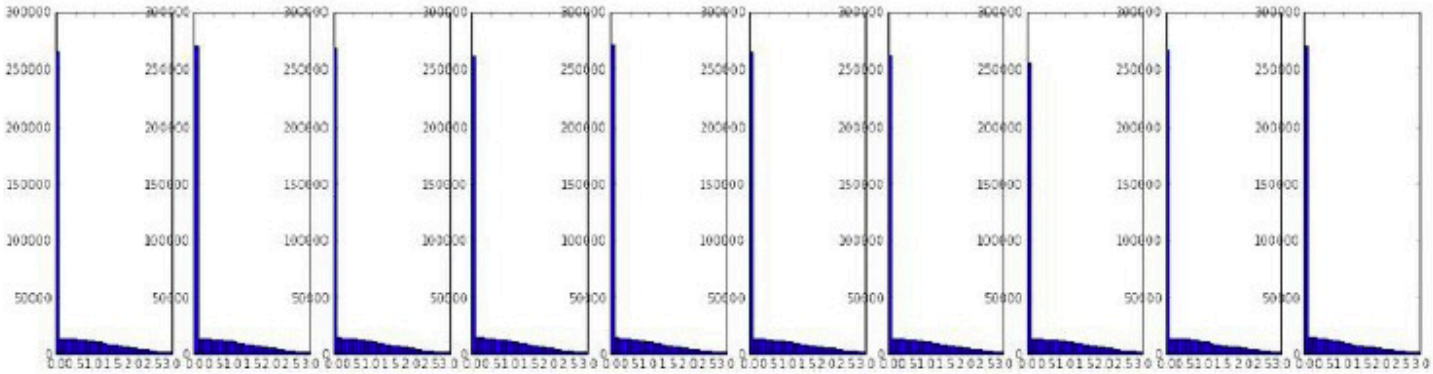
Initialization [He et al., 2015]

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```





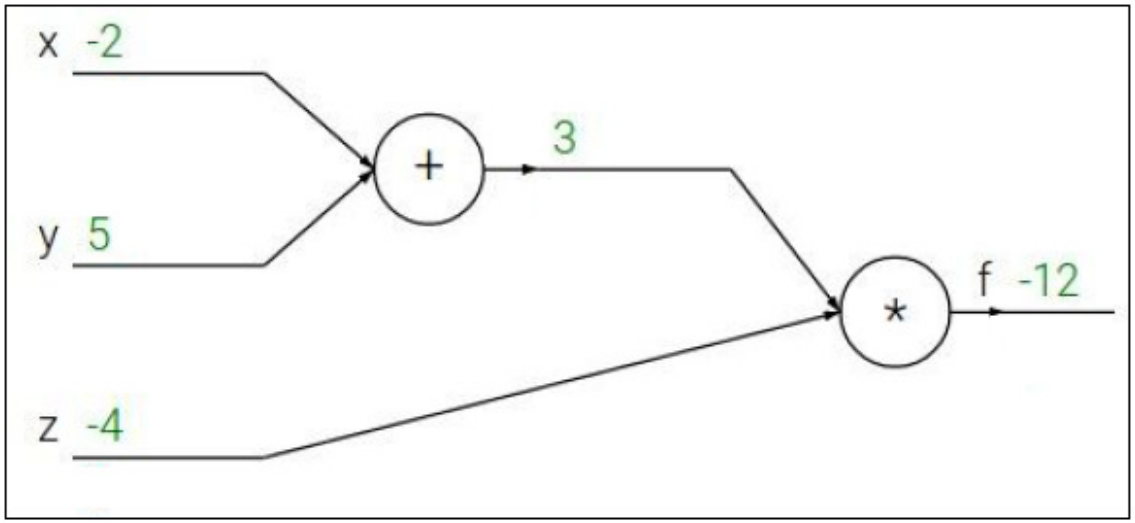


Back Propagation in Computation Graph

Initialization [He et al., 2015]

$$f(x, y, z) = (x + y)z$$

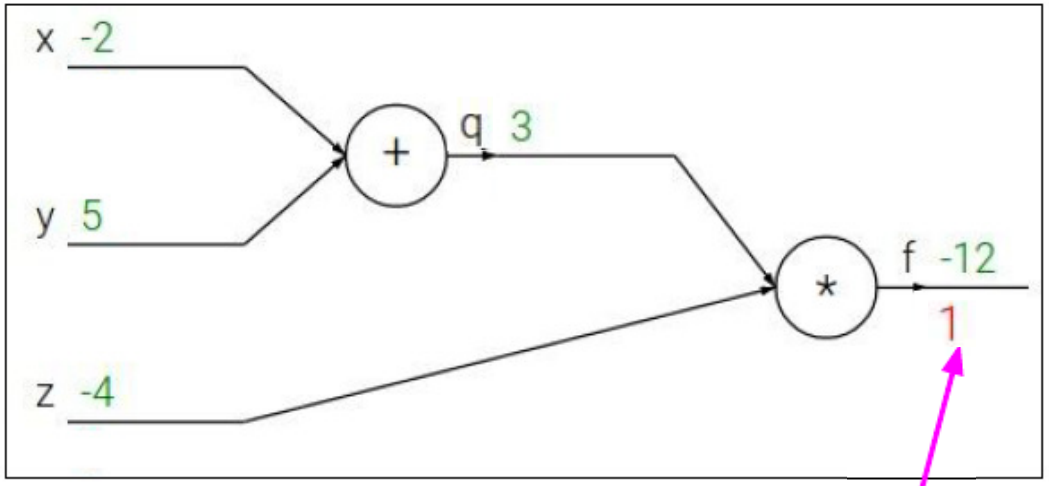
e.g. x = -2, y = 5, z = -4



Initialization [He et al., 2015]

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

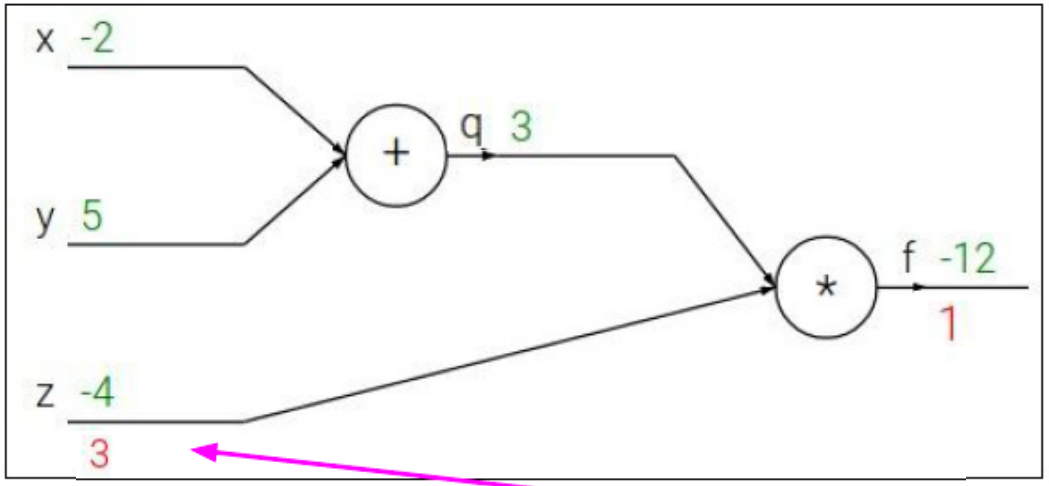
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Initialization [He et al., 2015]

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

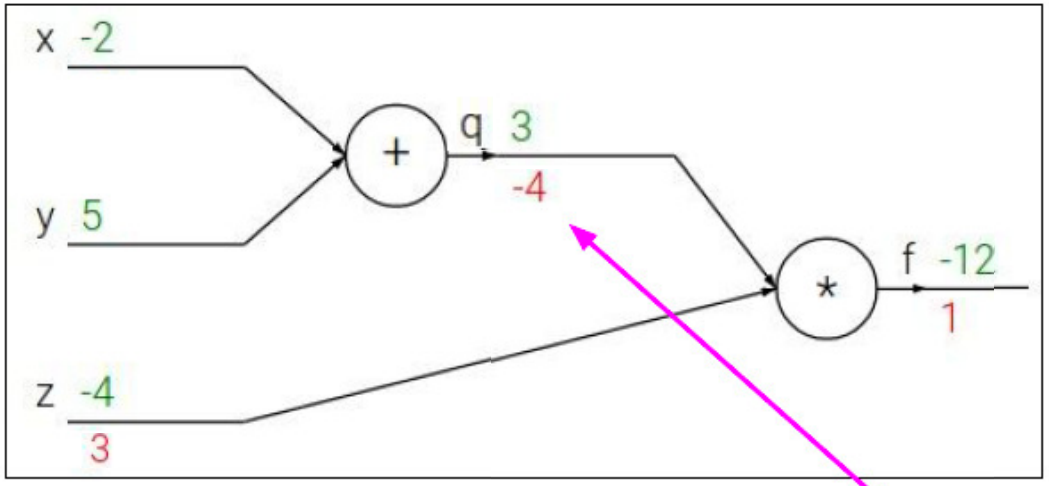
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial z}$

Initialization [He et al., 2015]

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

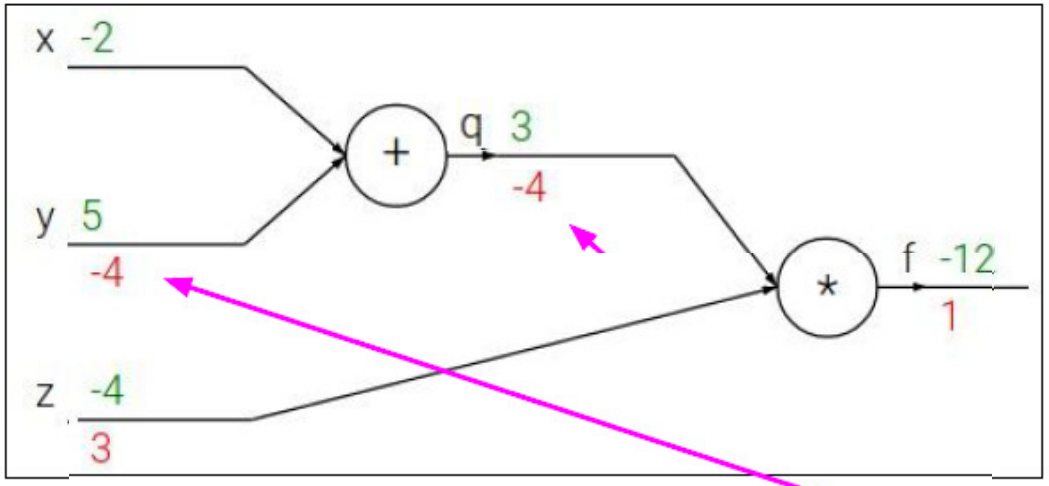
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial q}$

Initialization [He et al., 2015]

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial y}$

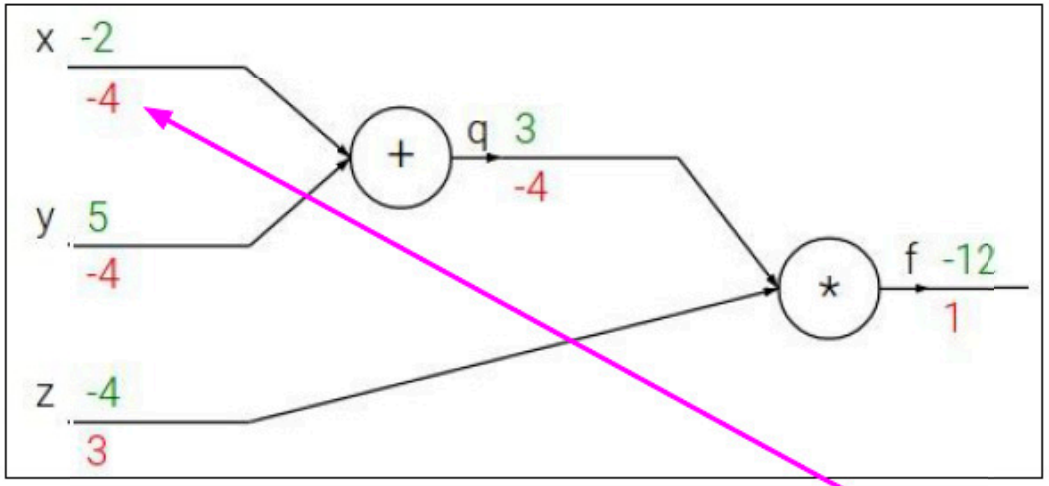
Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Initialization [He et al., 2015]

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

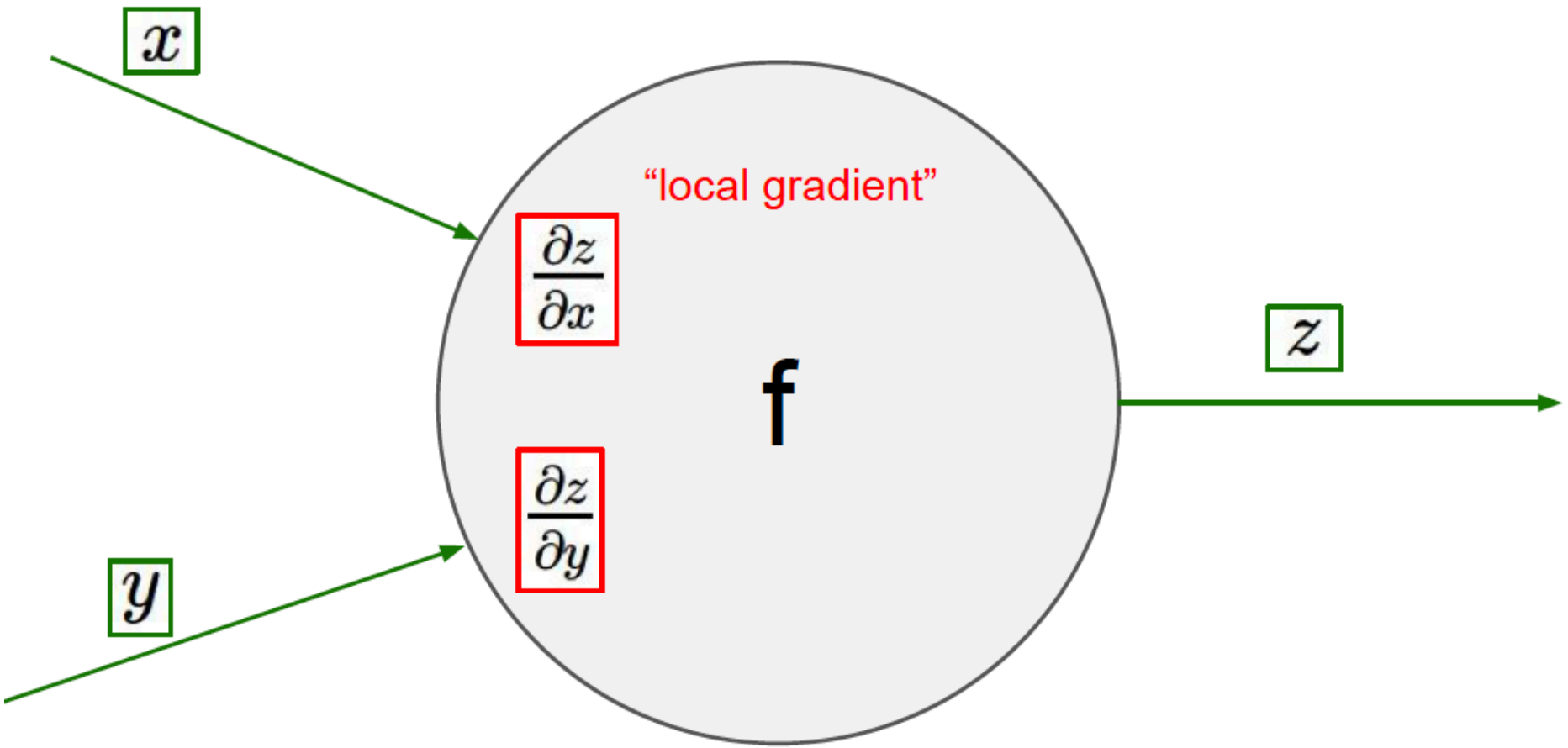
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

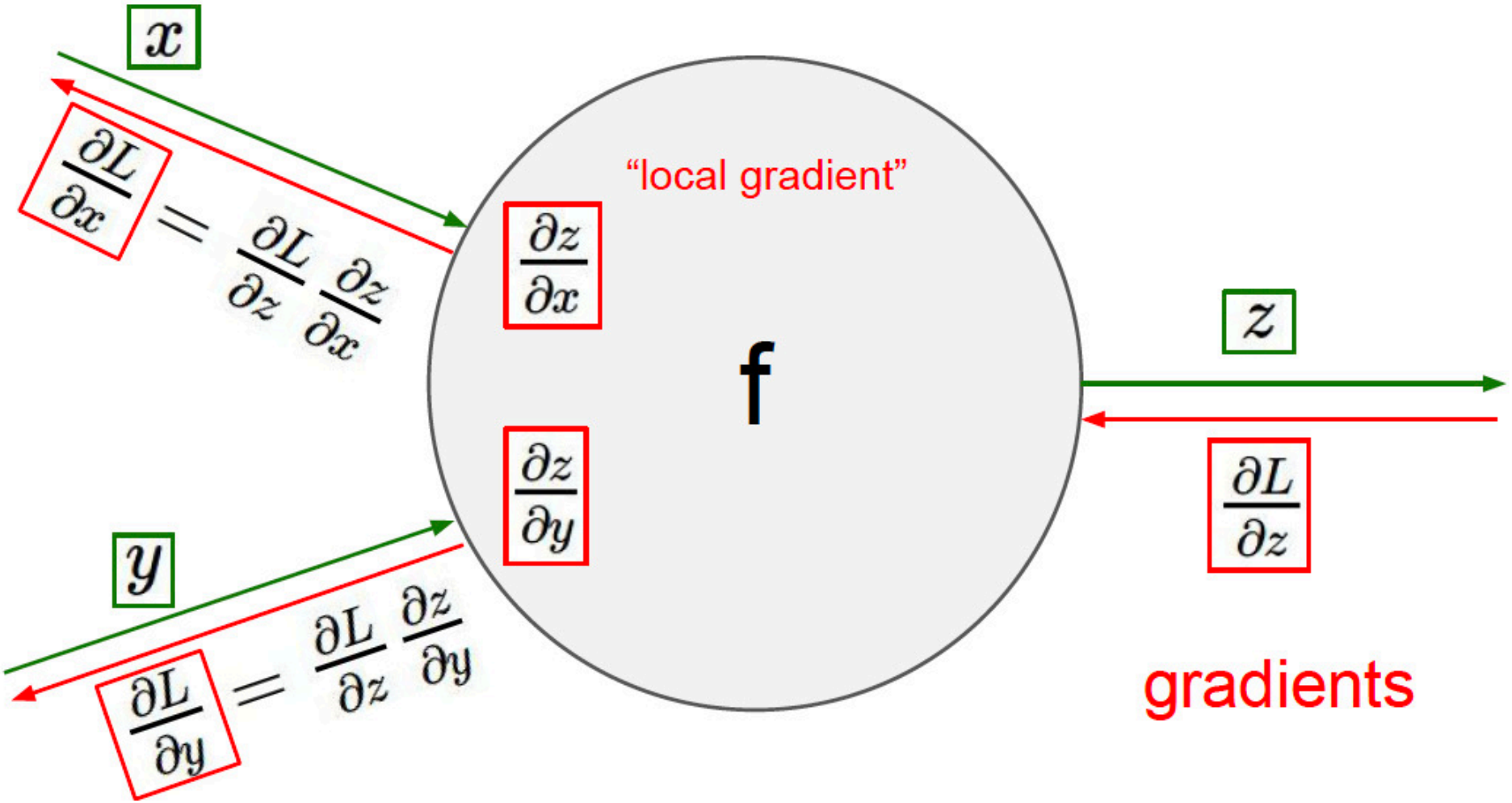
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

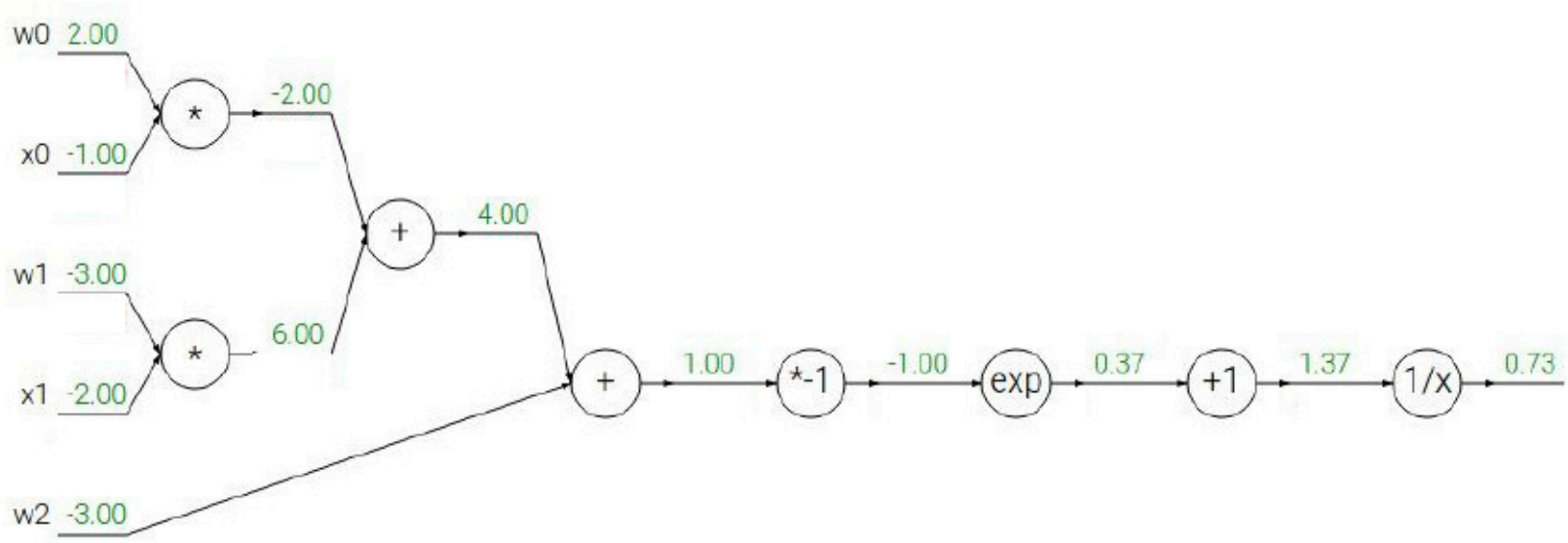
$\frac{\partial f}{\partial x}$


x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

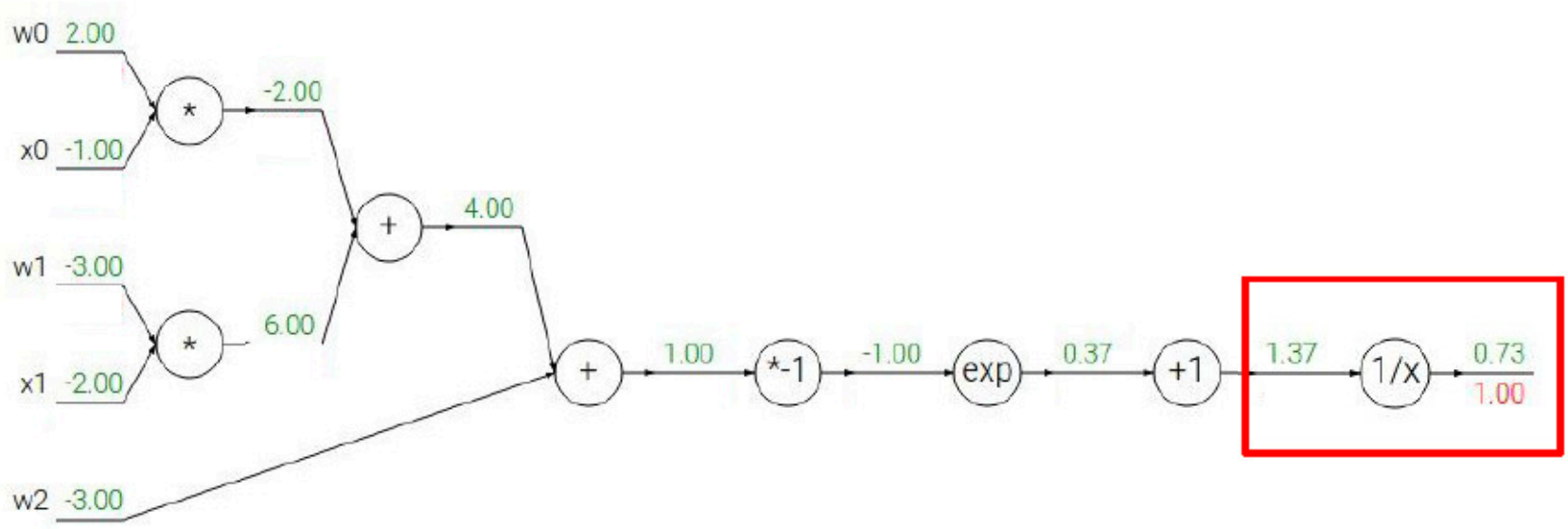
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

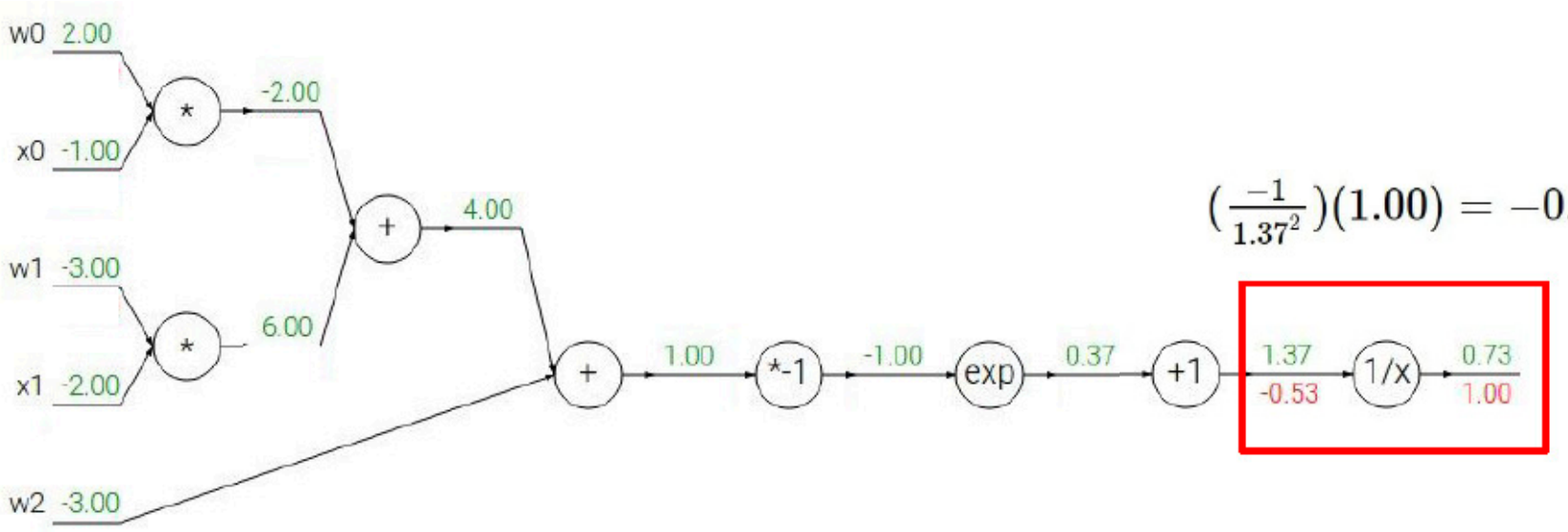
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(\frac{-1}{1.37^2})(1.00) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

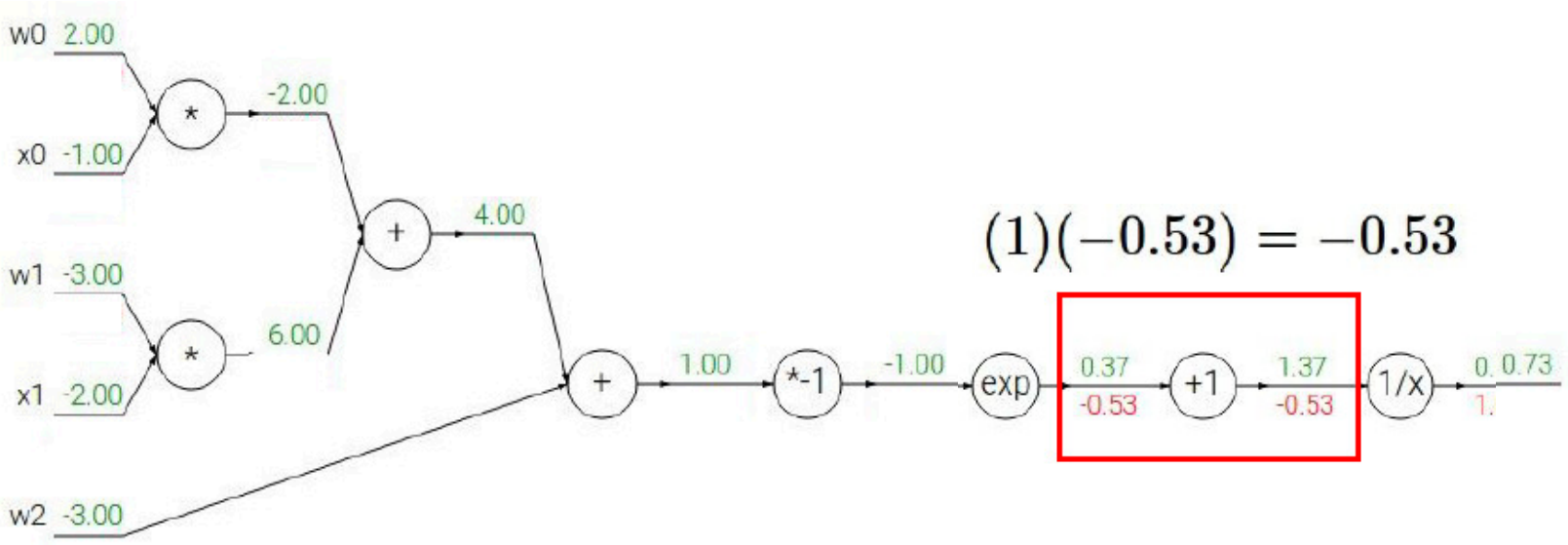
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(1)(-0.53) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

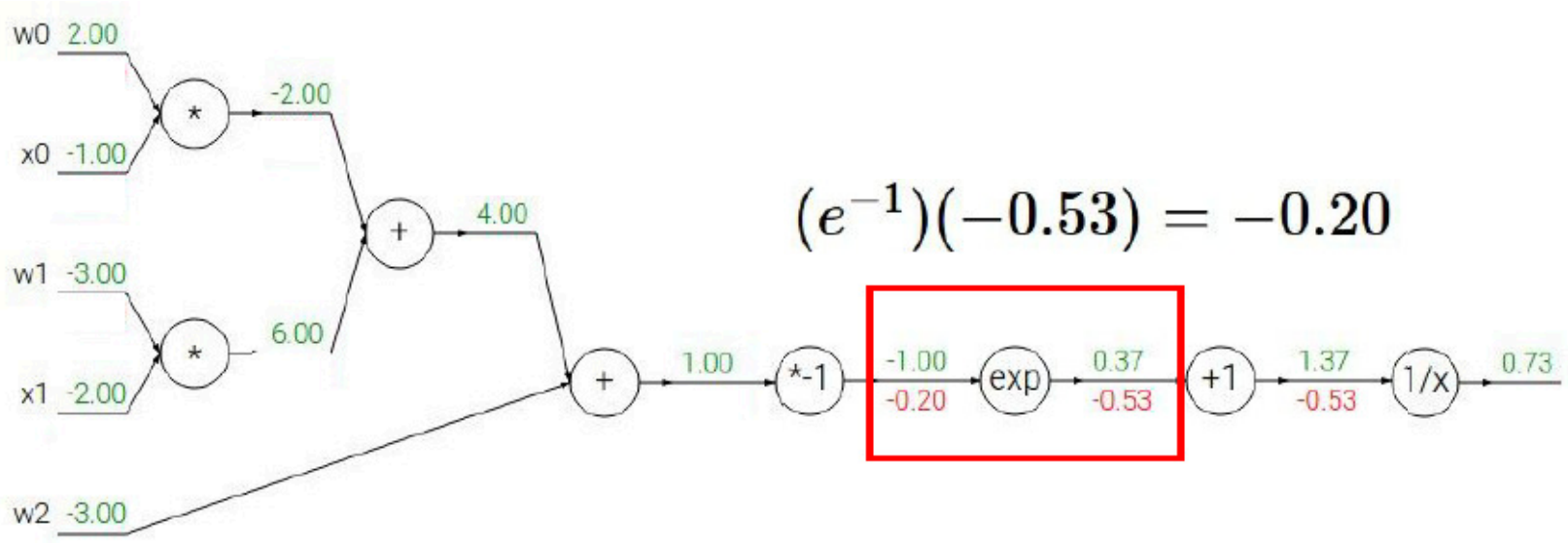
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



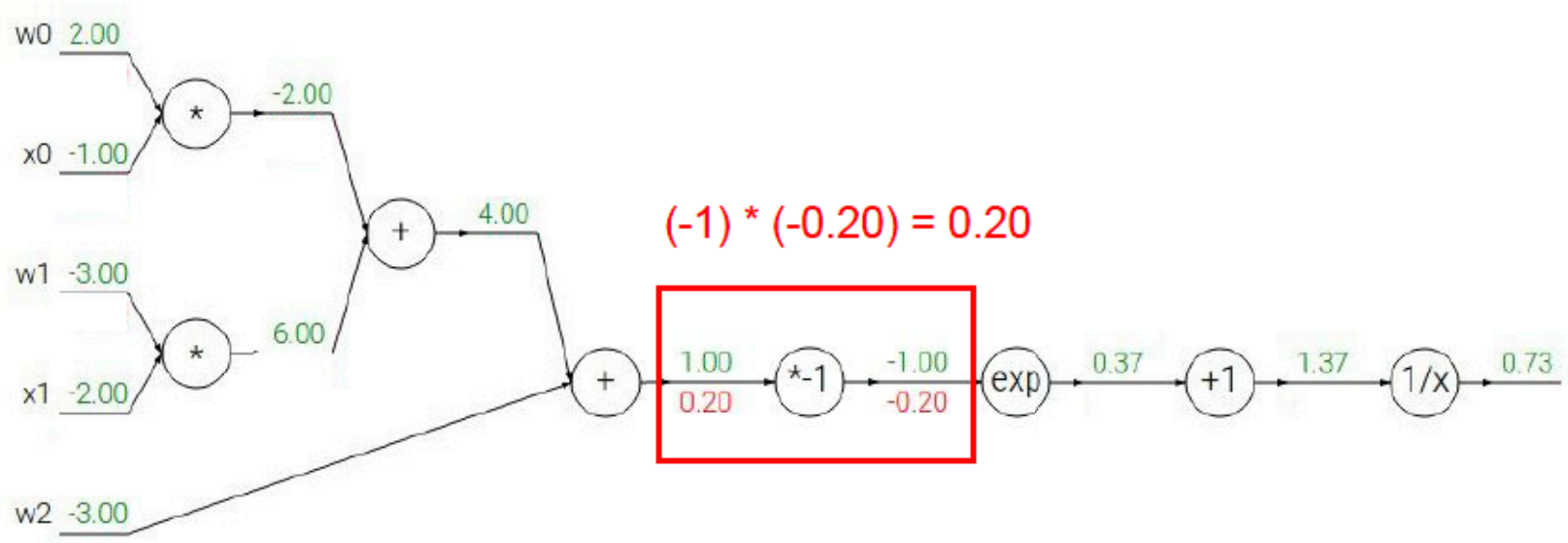
$$(e^{-1})(-0.53) = -0.20$$

$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$

$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

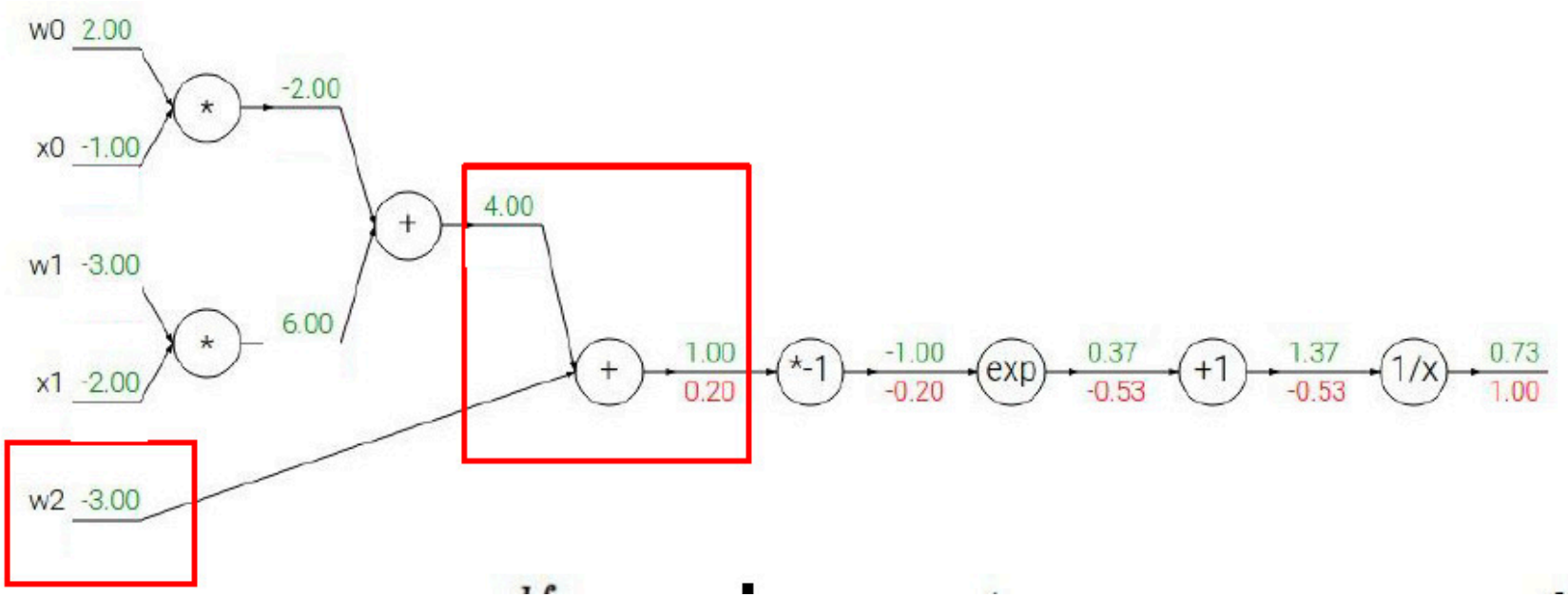
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

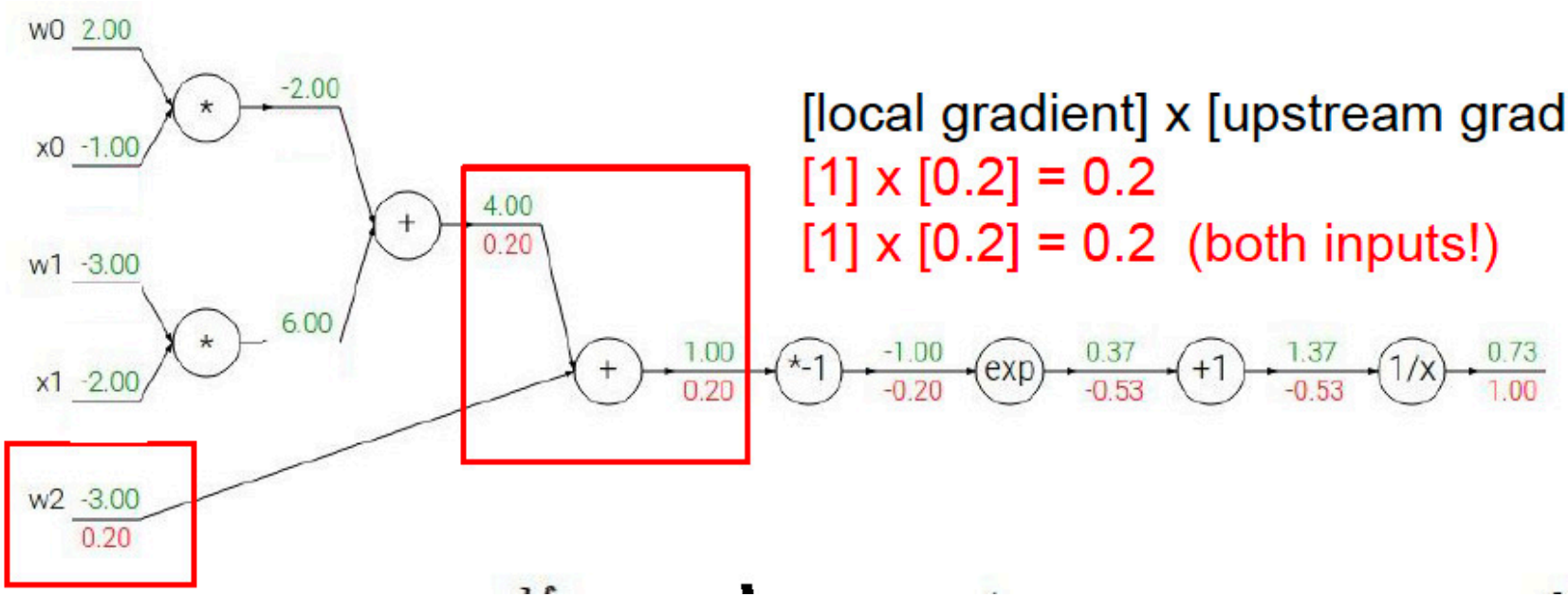
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [upstream gradient]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (both inputs!)

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

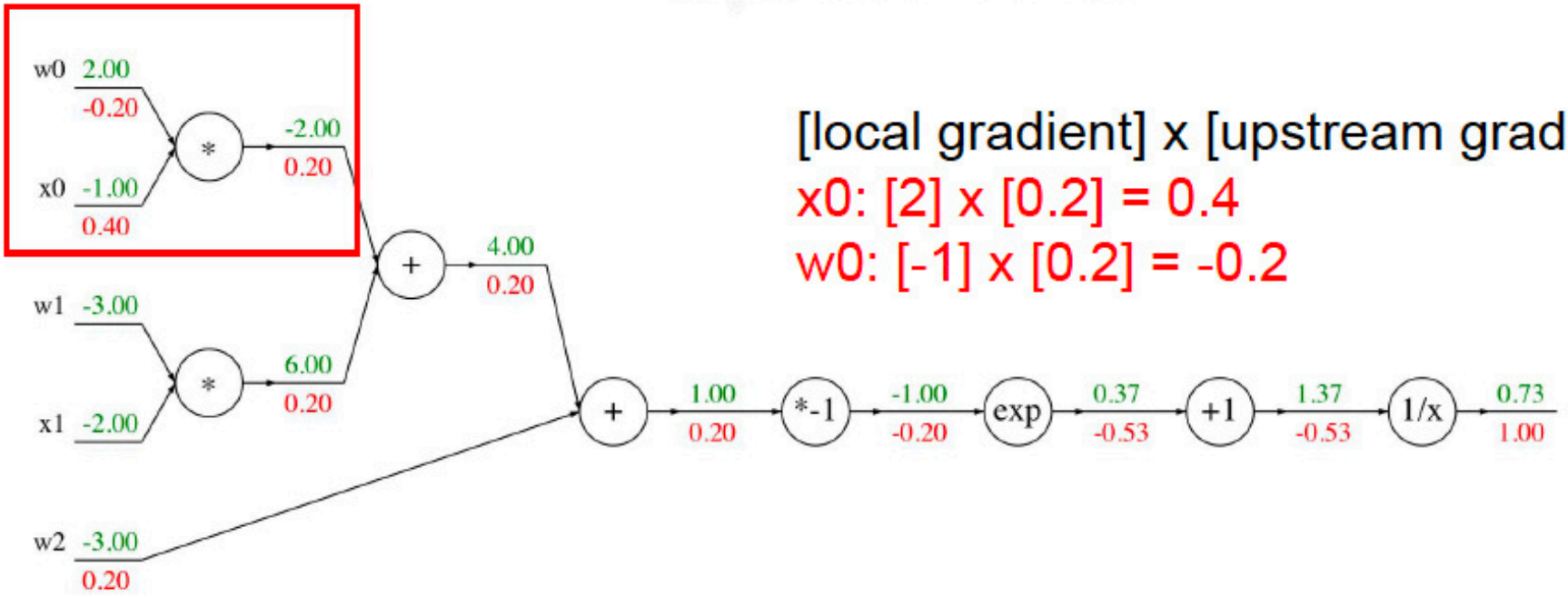
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [upstream gradien

x0: [2] x [0.2] = 0.4

w0: [-1] x [0.2] = -0.2

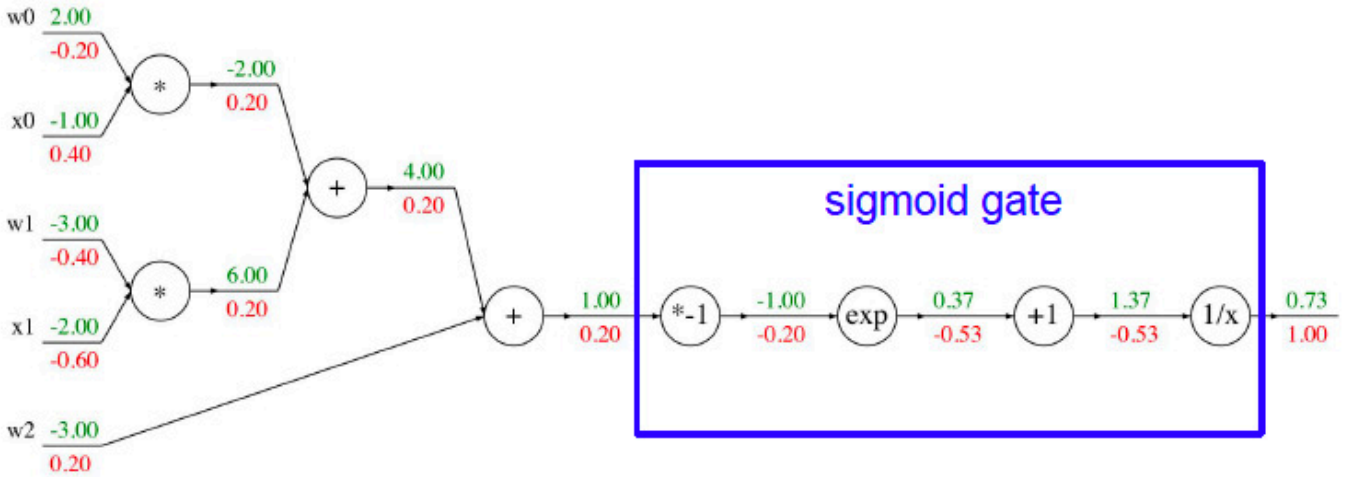
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x \qquad f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a \qquad f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$ sigmoid function

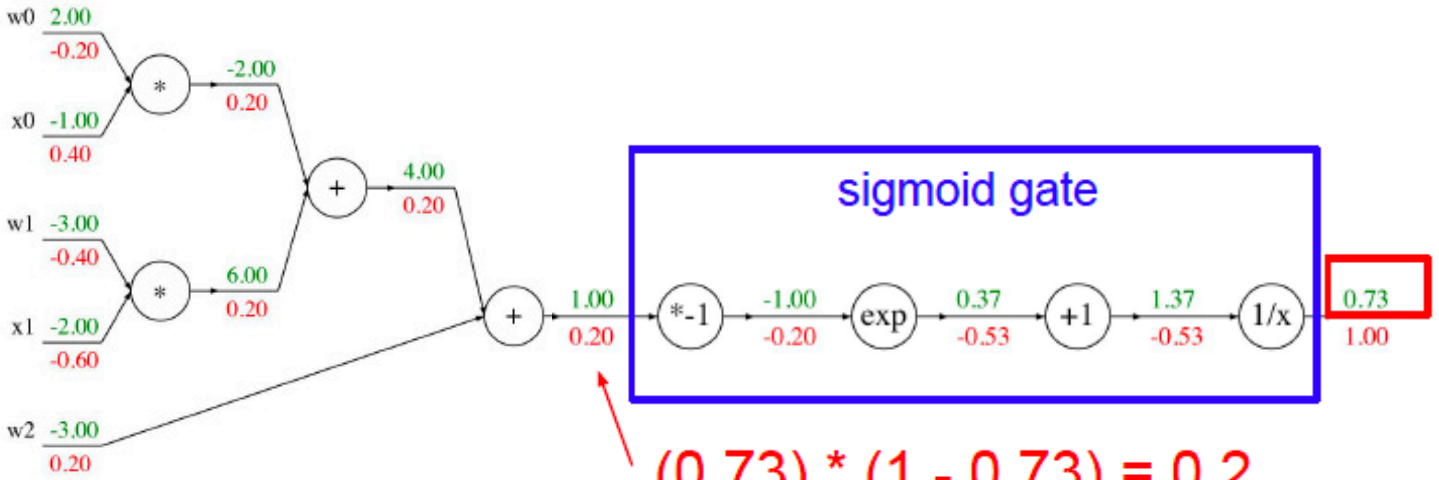
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

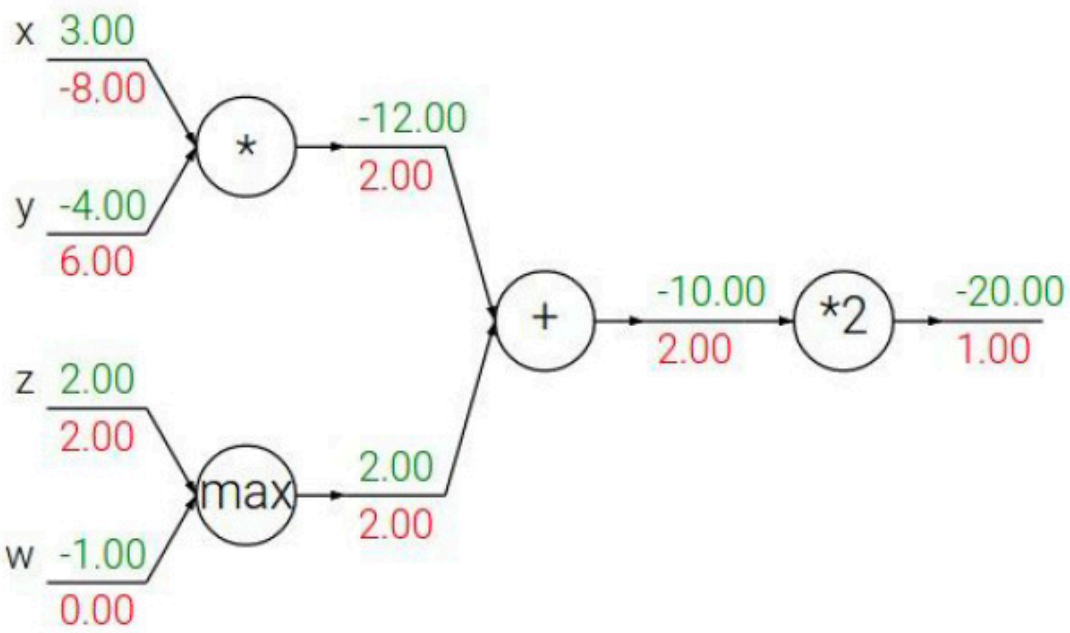
$$\sigma(x) = \frac{1}{1+e^{-x}}$$ sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



Backward Flow Patterns

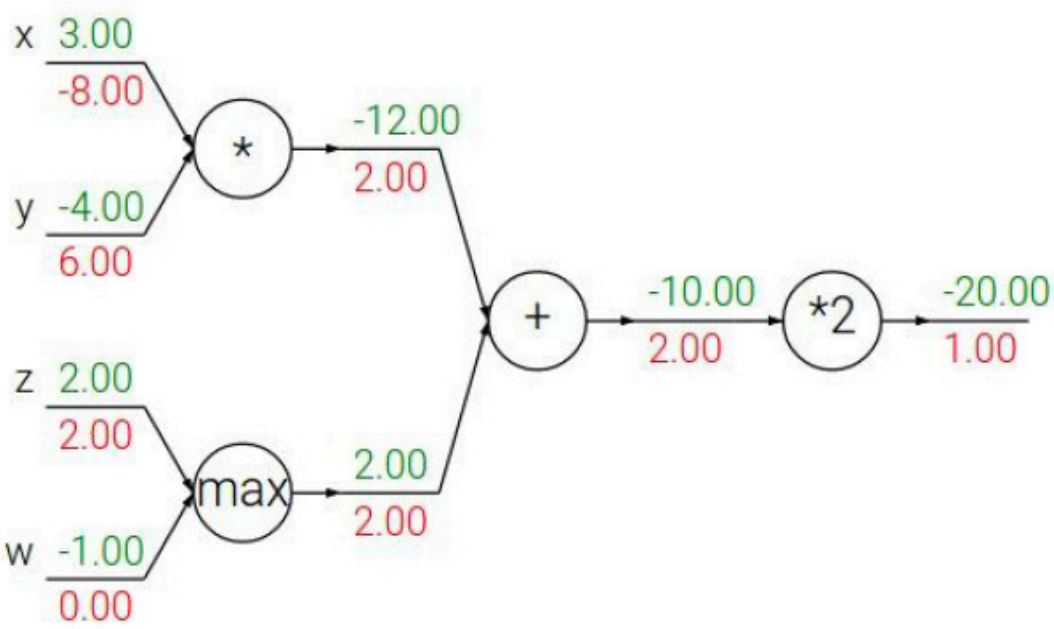
add gate: gradient distributor



Backward Flow Patterns

add gate: gradient distributor

max gate: gradient router

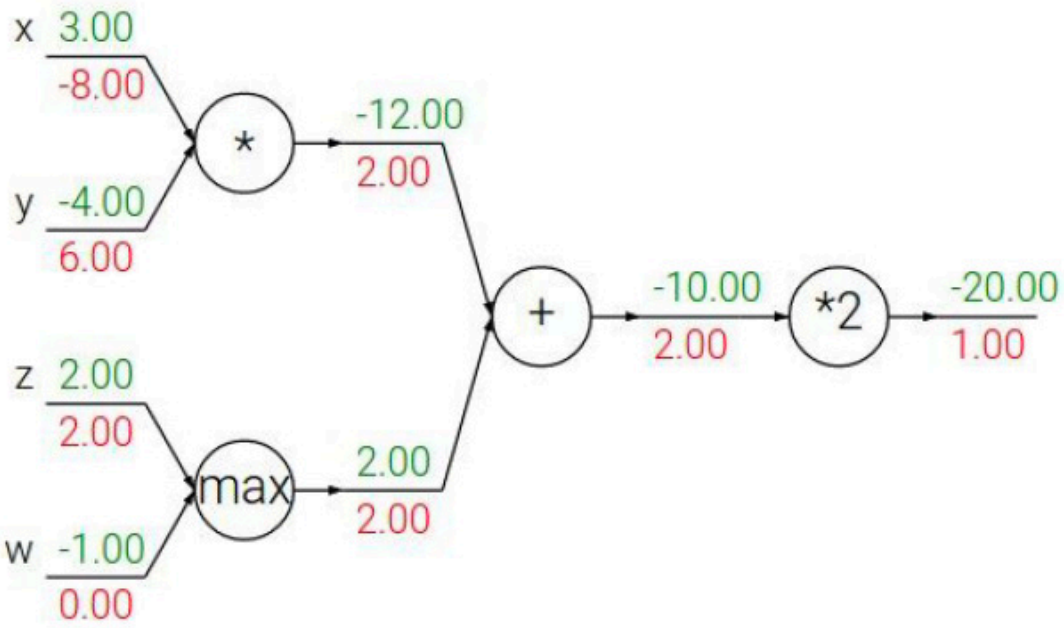


Backward Flow Patterns

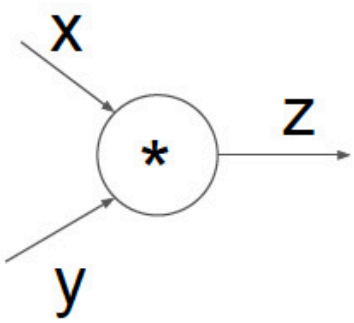
add gate: gradient distributor

max gate: gradient router

mul gate: gradient switcher



Modular Implementation



(x,y,z are scalars)

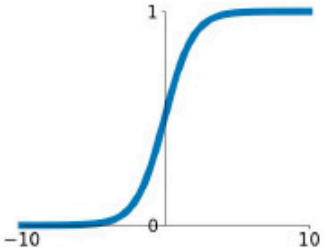
```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Activation Functions

Activation Functions

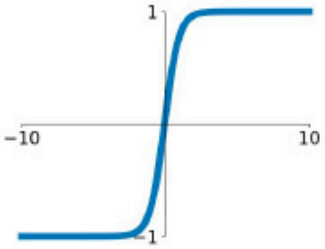
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



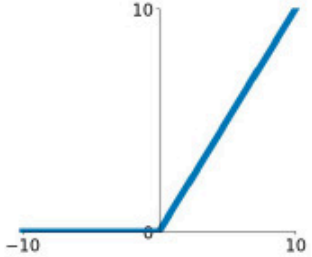
tanh

$$\tanh(x)$$



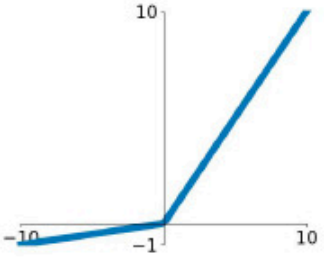
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

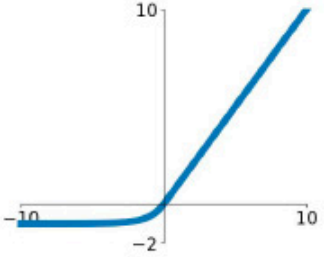


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

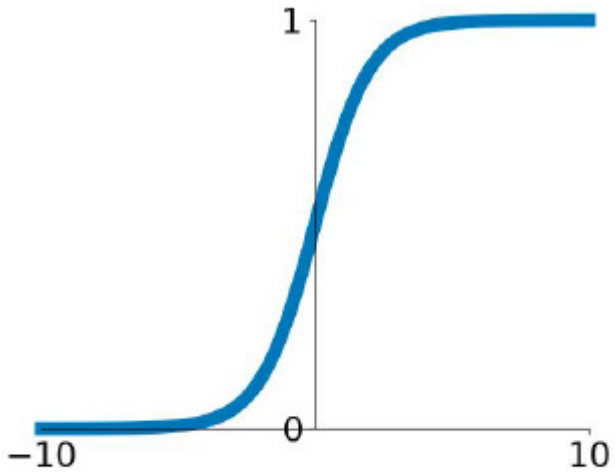
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

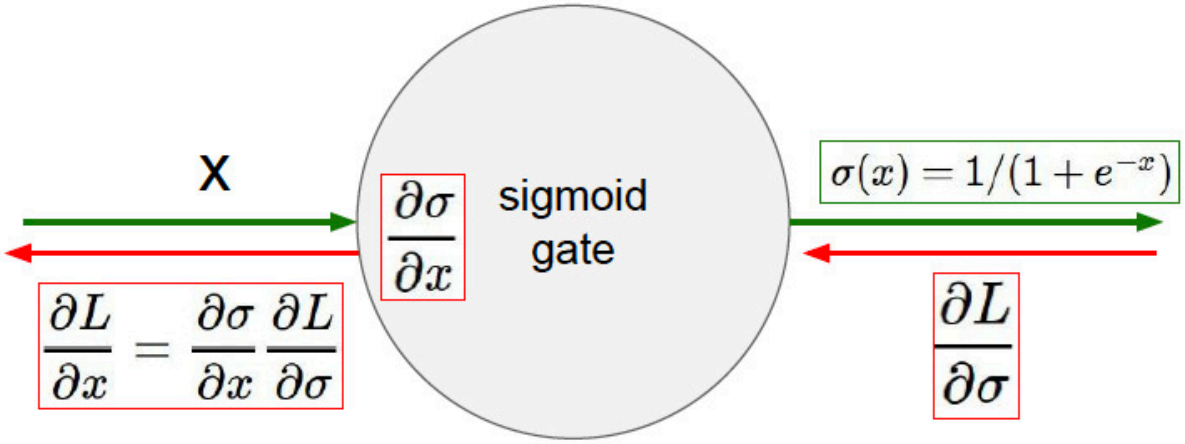


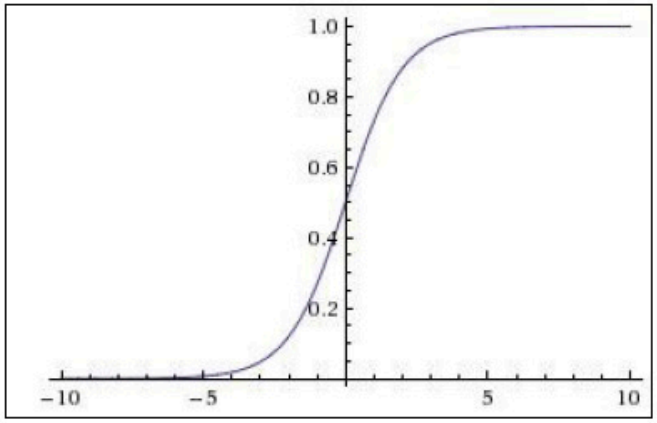
Sigmoid

- Squashes numbers to range [0,1]
- A linear output -> probability

Sigmoid : Problem 1

Saturated neurons "kill" the gradient

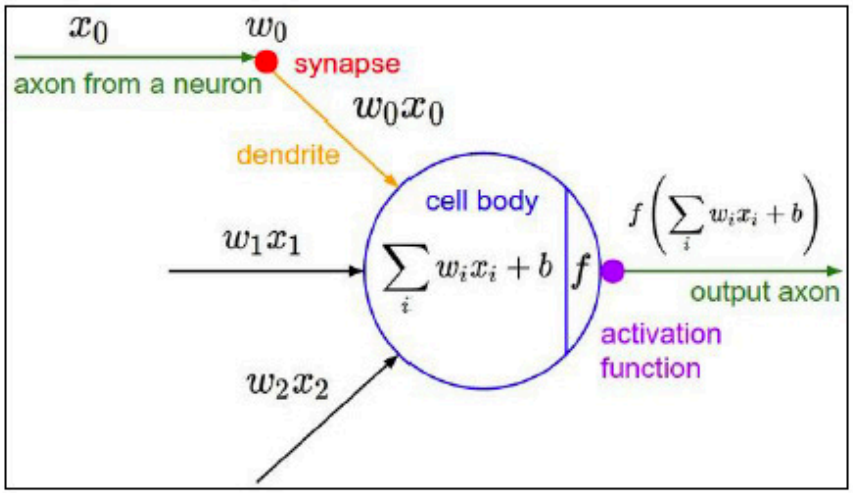




What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Sigmoid: Problem 2

Sigmoid output is not zero-centered

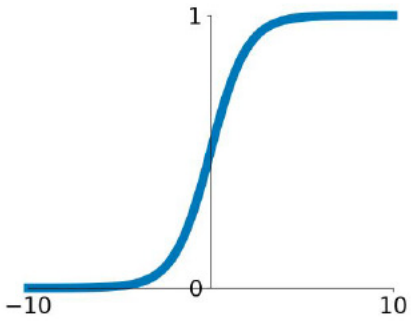


$$f\left(\sum_i w_i x_i + b\right)$$

What happens, if all input to the neuron is always positive?

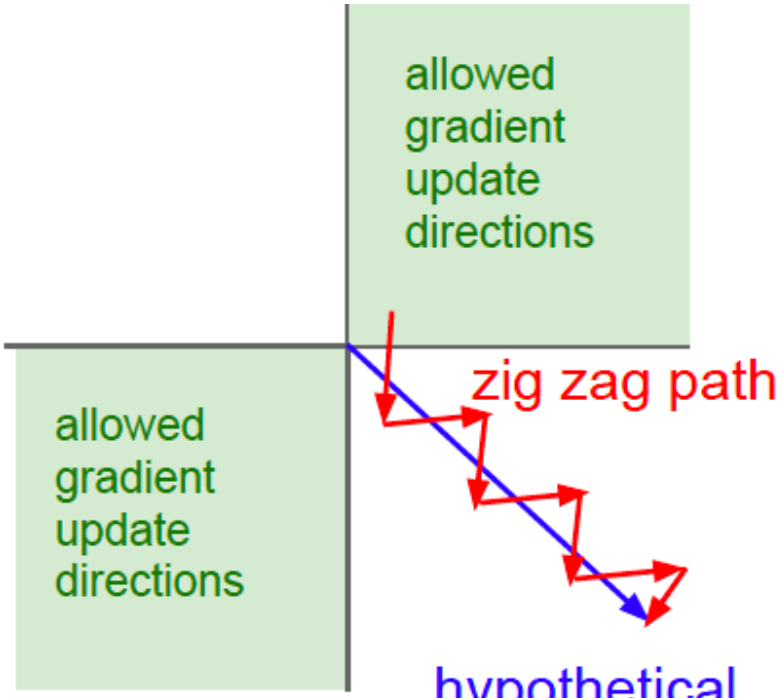
What can we say about the gradients?

Sigmoid: Problem 2



Sigmoid

$$f\left(\sum_i w_i x_i + b\right)$$



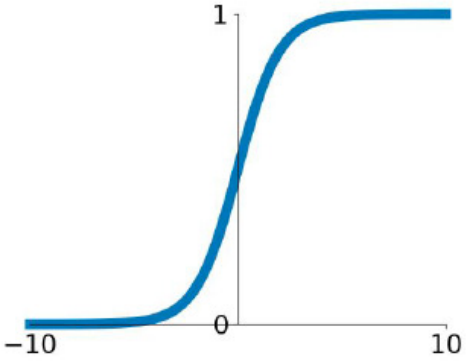
What can we say about the gradients?

Always all negative or all positive!

(this is also why you want zero-mean data)

Sigmoid: Problem 3

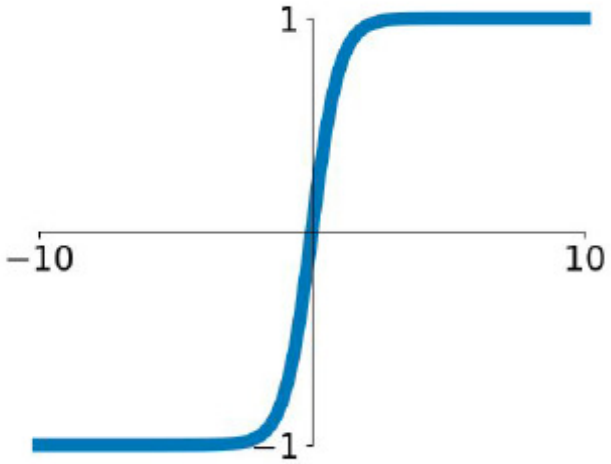
$$\sigma(x) = 1/(1 + e^{-x})$$



Sigmoid

Exp() is computationally expensive

TanH [LeCun et al., 1991]

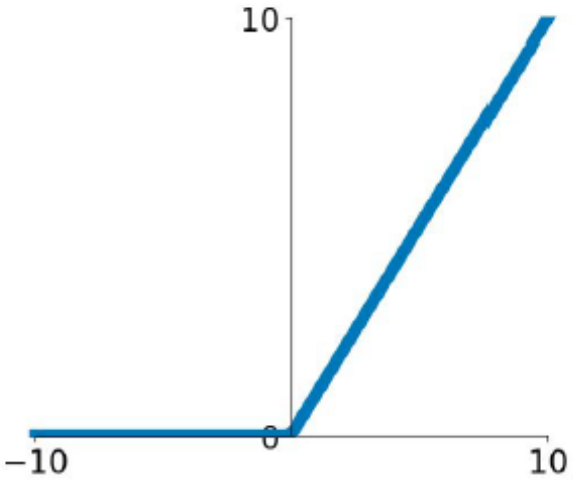


tanh(x)

- Squashes numbers to range [-1,1]
- **Zero-centered**
- **Still kills gradients when saturated**

ReLU [Krizhevsky et al., 2012]

$$f(x) = \max(0,x)$$



ReLU
(Rectified Linear Unit)

- Does not saturate (in + region)
- Very computationally efficient
- In practice, NN tends to converge faster

- Not zero-centered
- Gradient is always zero in - region

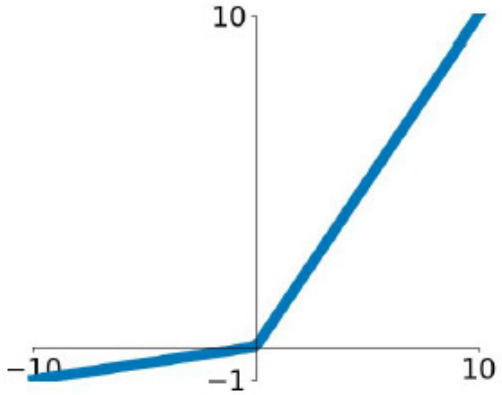
ReLU [Krizhevsky et al., 2012]

Dead ReLU will never activate

➔ Never update

➔ We often initialize ReLU units with slightly positive biases (e.g. 0.01)

Leaky ReLU



- Does not saturate
- Computationally efficient
- In practice, NN tends to converge faster
- Will not die, like ReLU

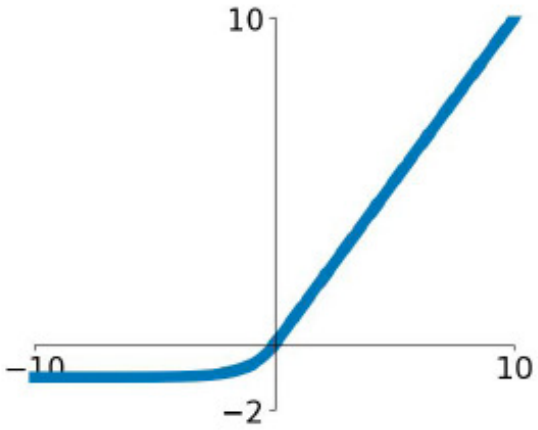
Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

ELU (Exponential Linear Units)



- All benefits of ReLU
- Closer to zero-mean output
- Negative region adds some robust ness, like Leaky ReLU

- Exp() is expensive

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right) & \text{if } x \leq 0 \end{cases}$$

Maxout [Goodfellow et al., 2013]

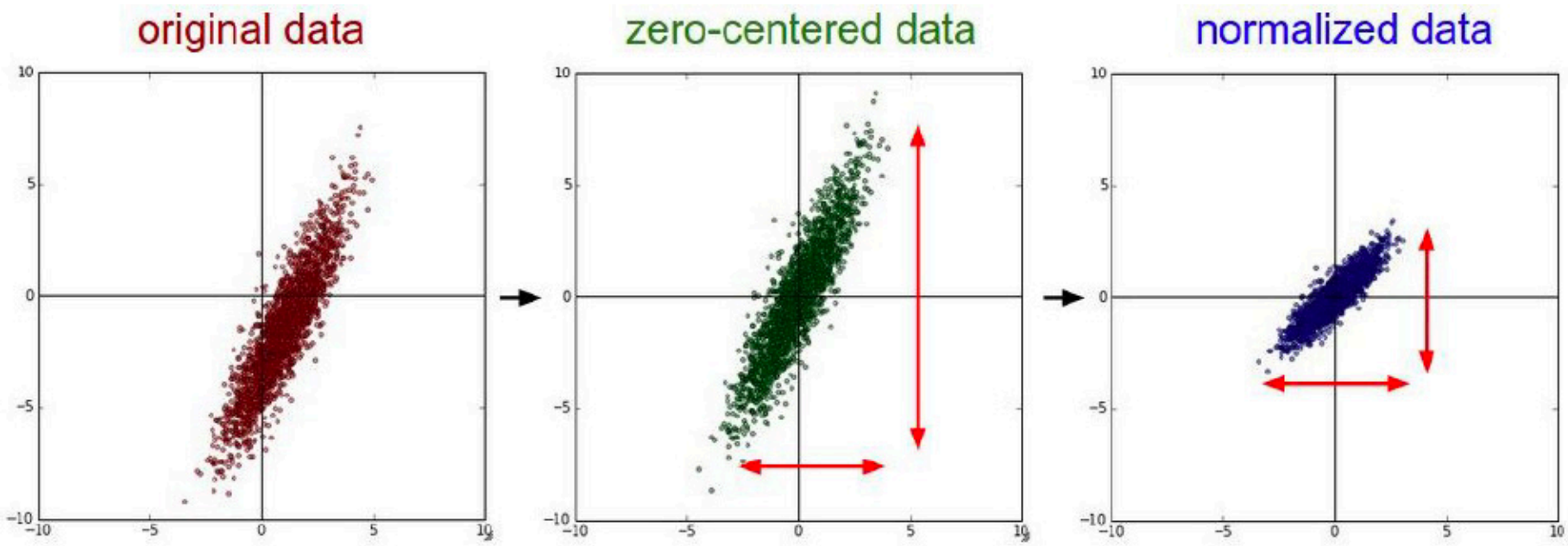
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- Generalizes ReLU and Leaky ReLU
- Linear parts
- Does not saturate
- Does not die

- Double the number of parameters

Data Pre-processing

Centering / Normalization

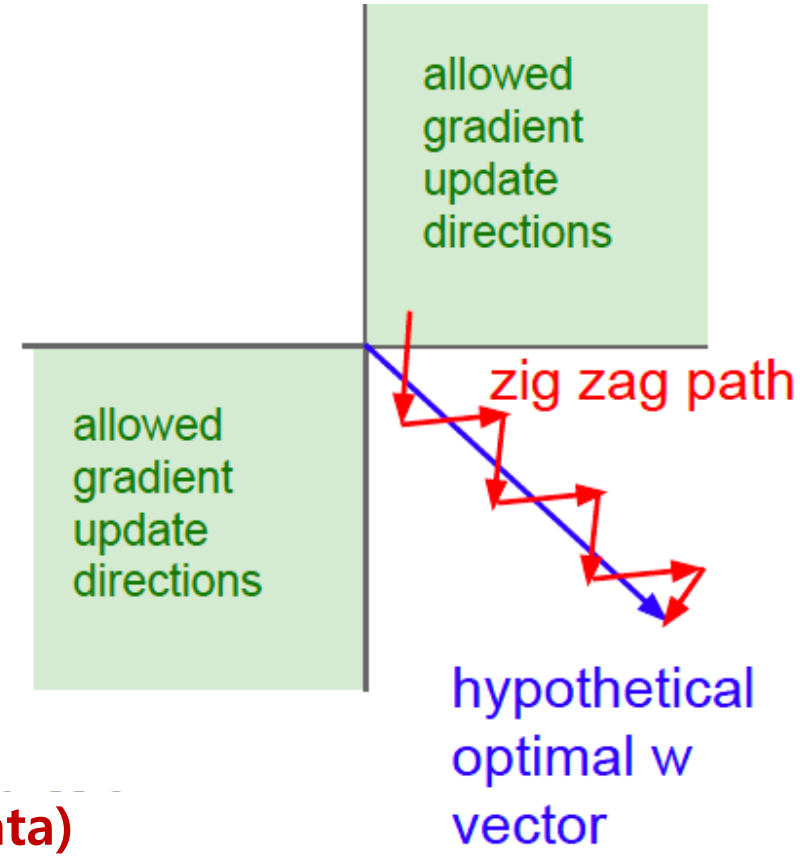



```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

When Inputs to a Neuron is All Positive

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients?

Always all negative or all positive!

(this is also why you want zero-mean data)

In Practice for Images

Ex. CIFAR-10 images of [32,32,3]

- **AlexNet: subtract the mean image**
  - **(mean image = [32,32,3] array)**
- **VGGNet: subtract per-channel mean**
  - **(mean along each channel = 3 numbers)**

Batch Normalization

Batch Normalization [Ioffe & Szegedy, 2015]

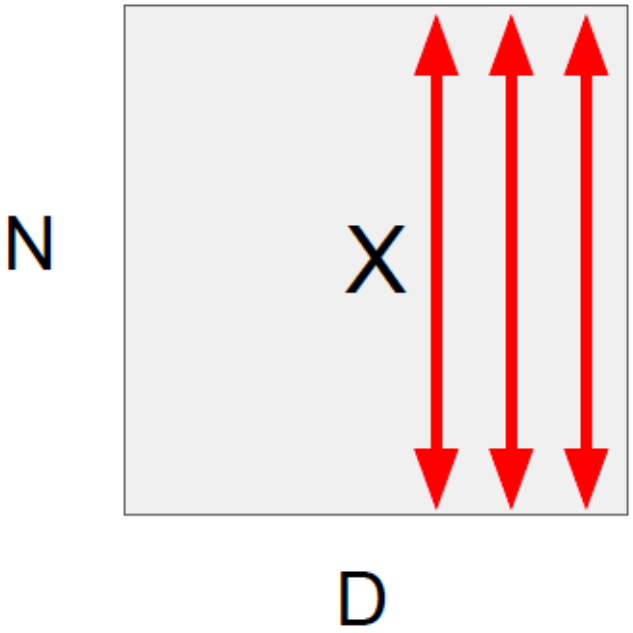
Consider a batch of activations at a layer:

To make each dimension unit Gaussian, apply

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

This is a differentiable function

Batch Normalization

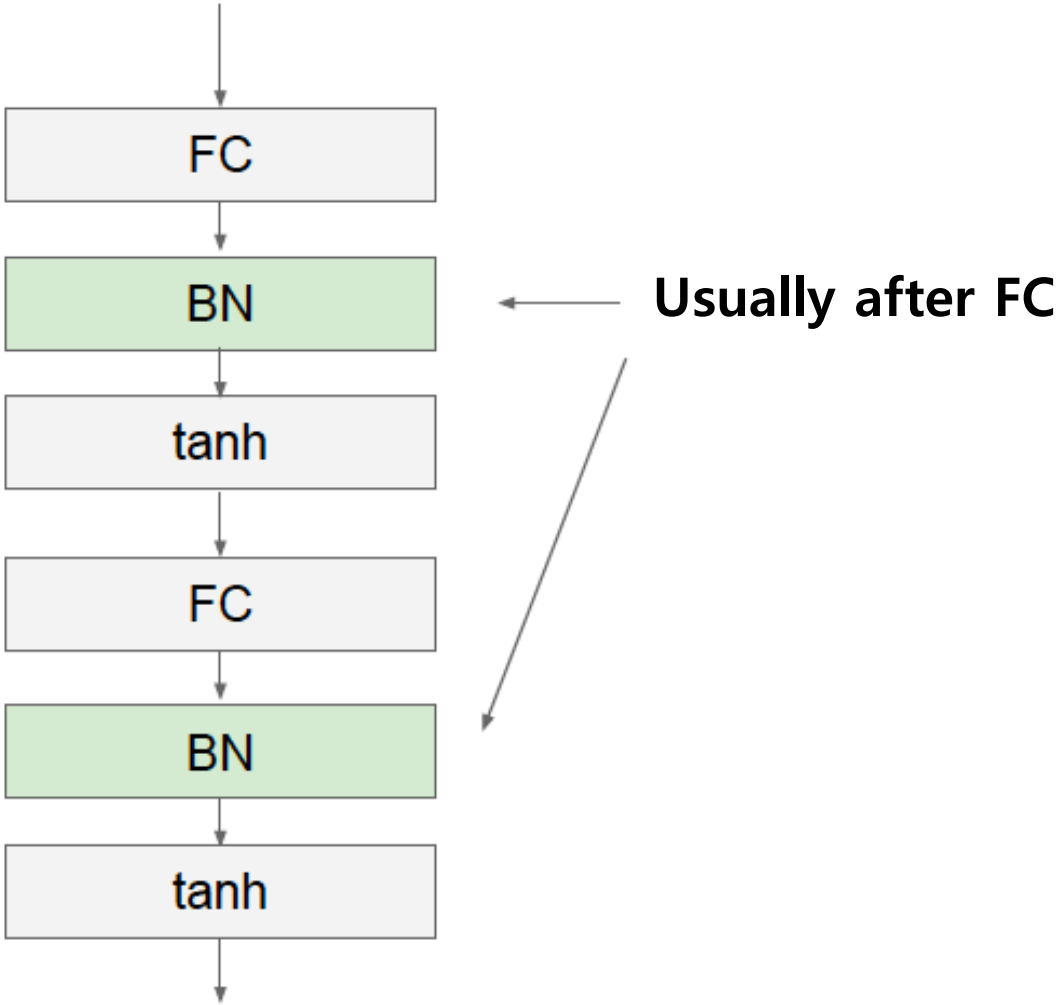


1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization



Usually after FC and before nonlinearity

Batch Normalization

We may not want unit Gaussian input, e.g. to tanh

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- **Allows larger learning rates**
- **Reduces strong dependency on initialization**
- **Acts as a form of regularization**

Hyperparameter Tuning

Hyperparameter Tuning

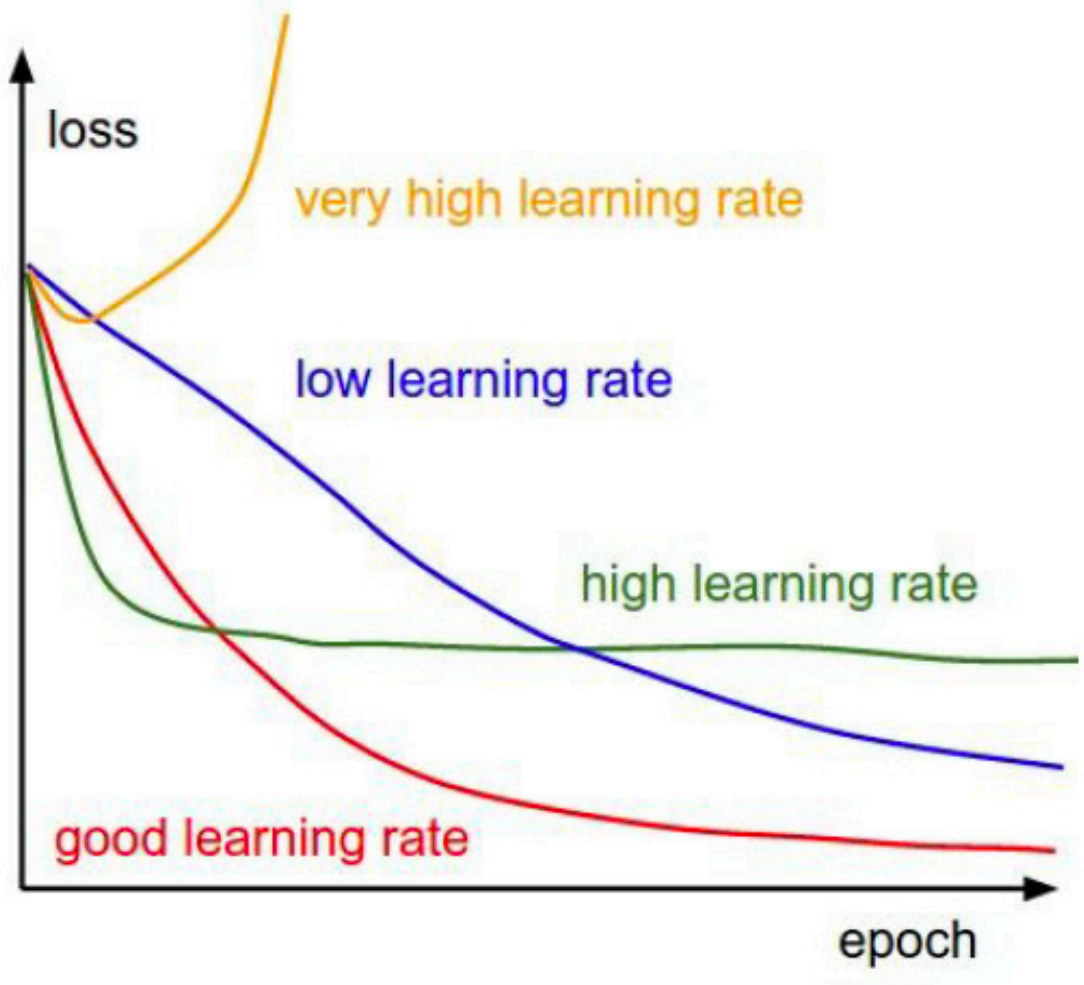
Hyperparameters

- **Network architecture**
- **Learning rate, its decay schedule, update type**
- **Regularization (L2 / Dropout strength)**

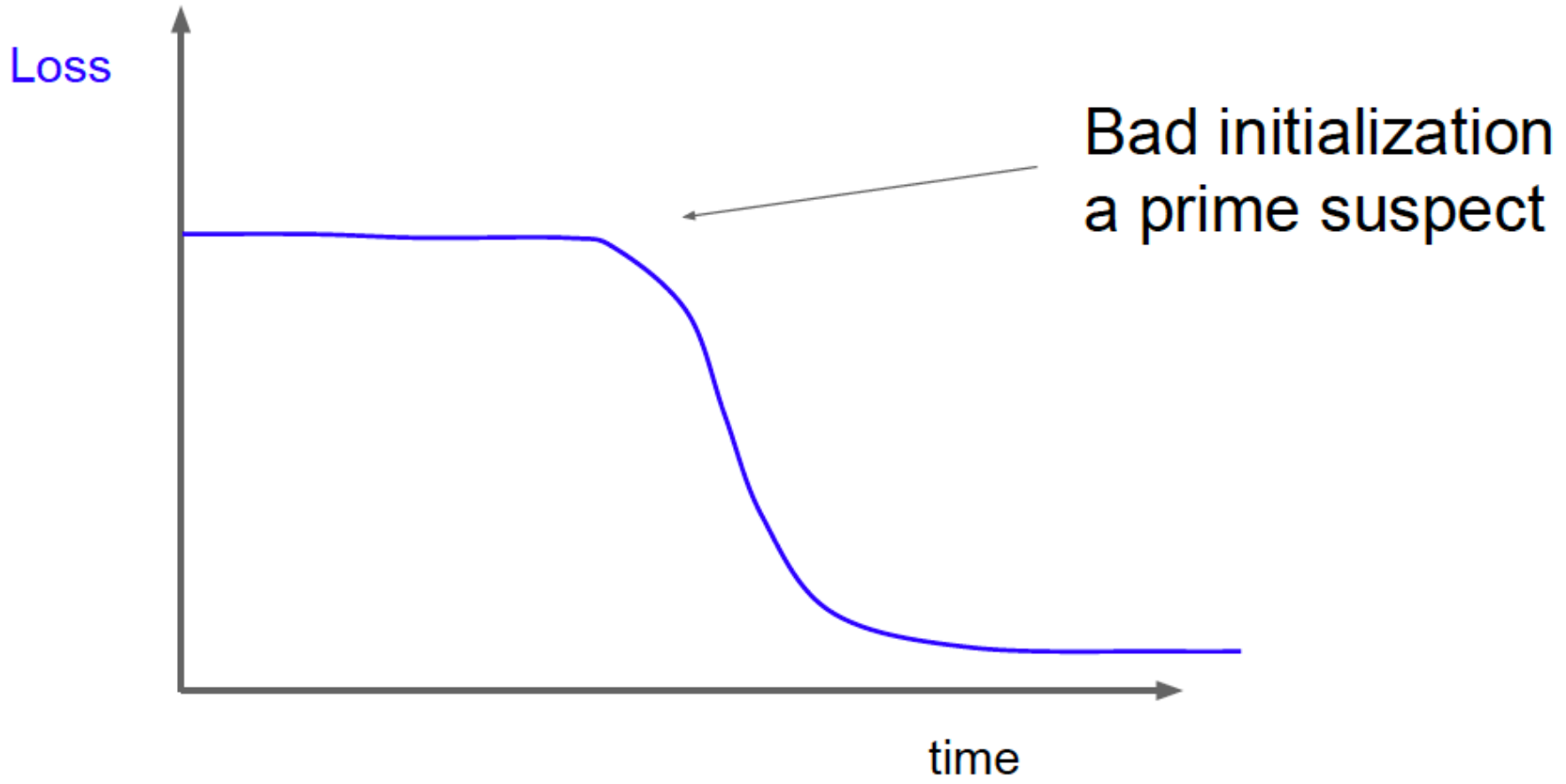
Search strategy: coarse → fine cross validation in stages

- **First stage:** only a few epochs to get rough idea of what parameters work
- **Second stage:** longer running time, finer search

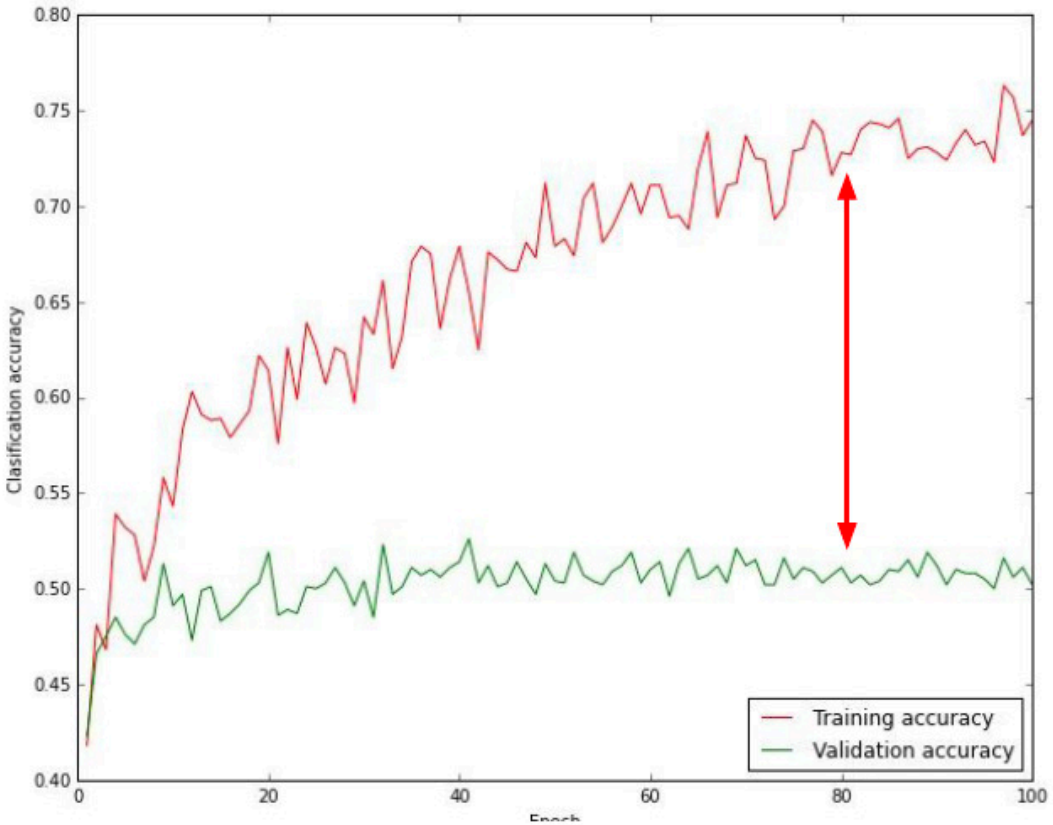
Learning Rates



Initialization



Training vs. Validation Accuracy



big gap = overfitting

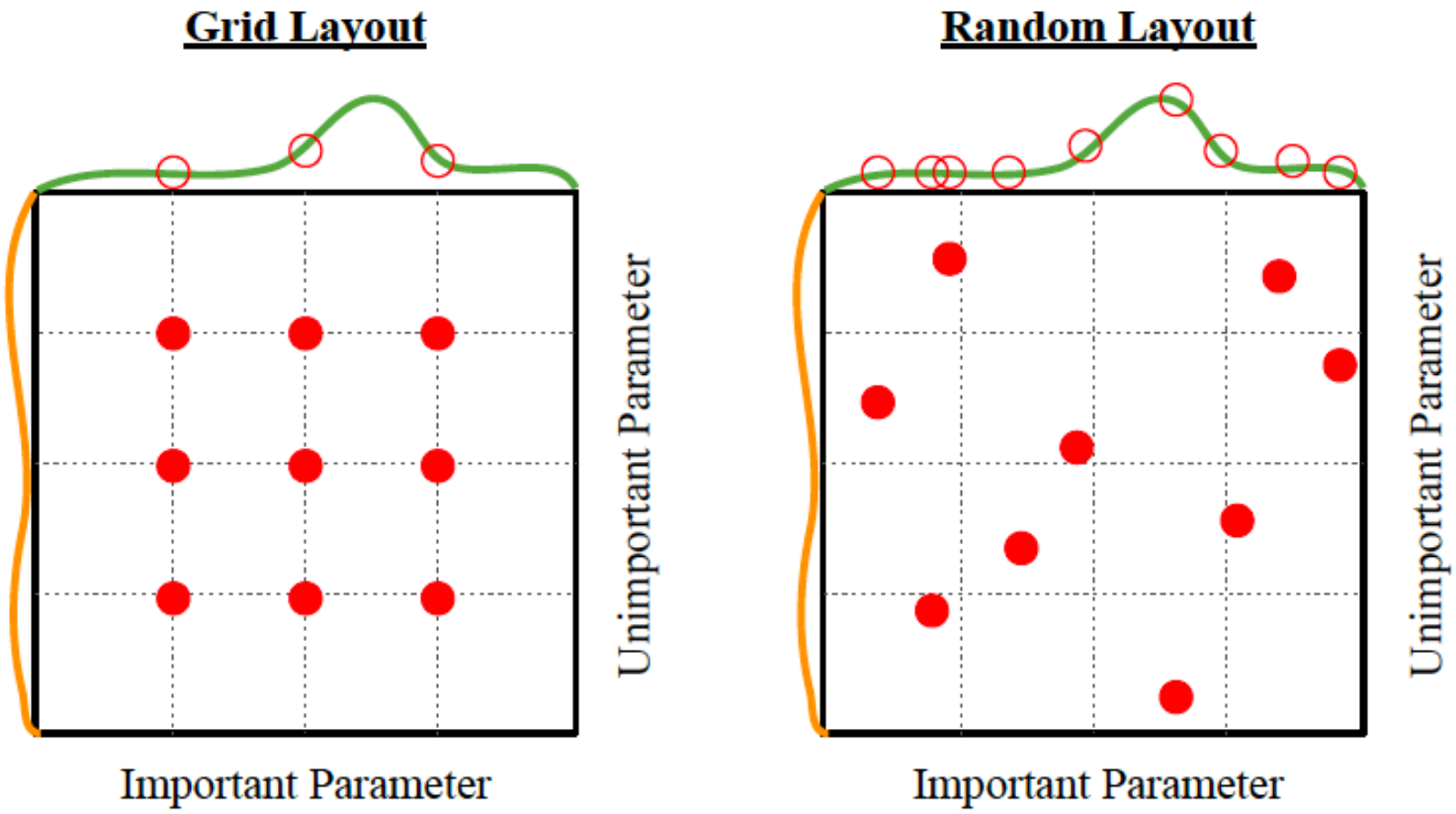
=> increase regularization strength?

no gap

=> increase model capacity?

Random vs. Grid Search

[Bergstra and Bengio, 2012]

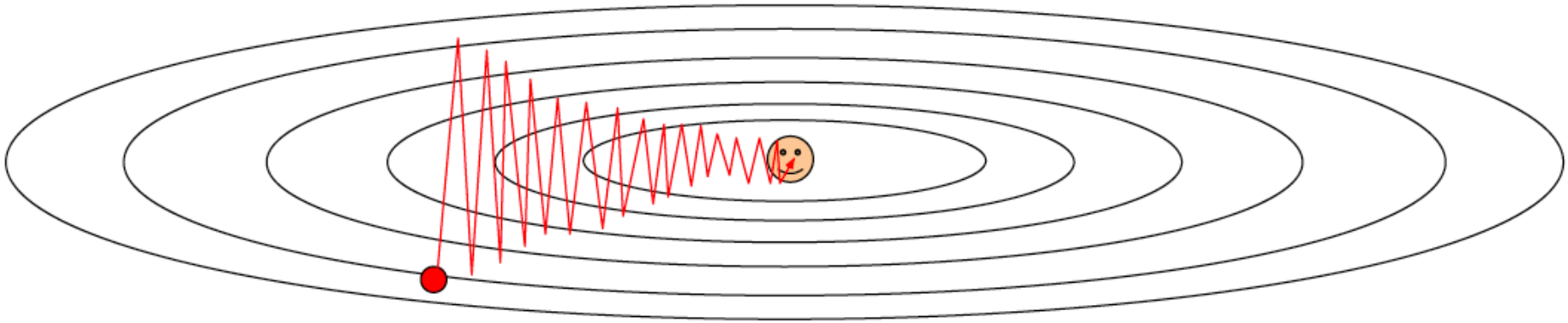


Optimization

Problems of SGD

If the loss function is ***ill-conditioned***, gradient descent does:

Very slow progress along shallow dimension, jitter along steep direction

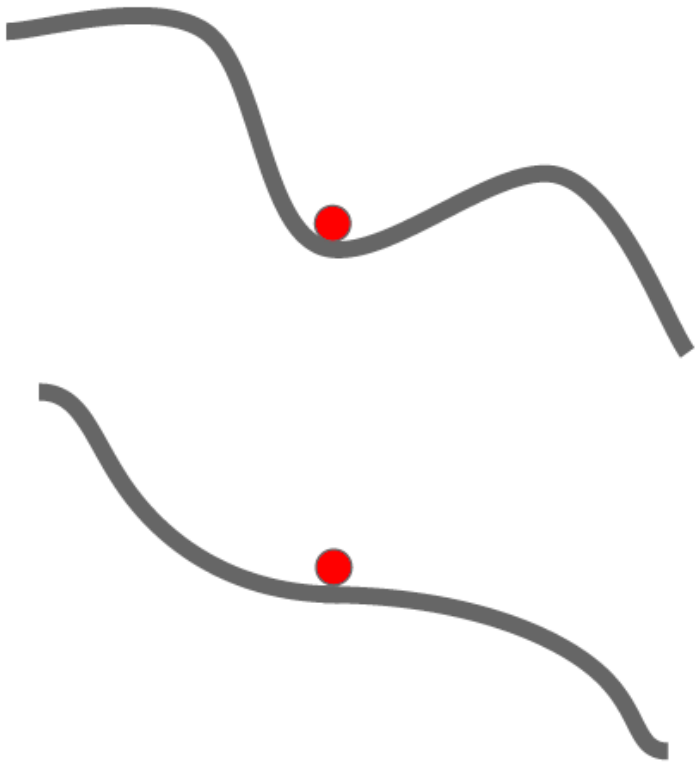


Problems of SGD

At **local minima** or **saddle-point**?

Zero gradient → (S)GD gets stuck

Saddle point is much more common in high dimensions

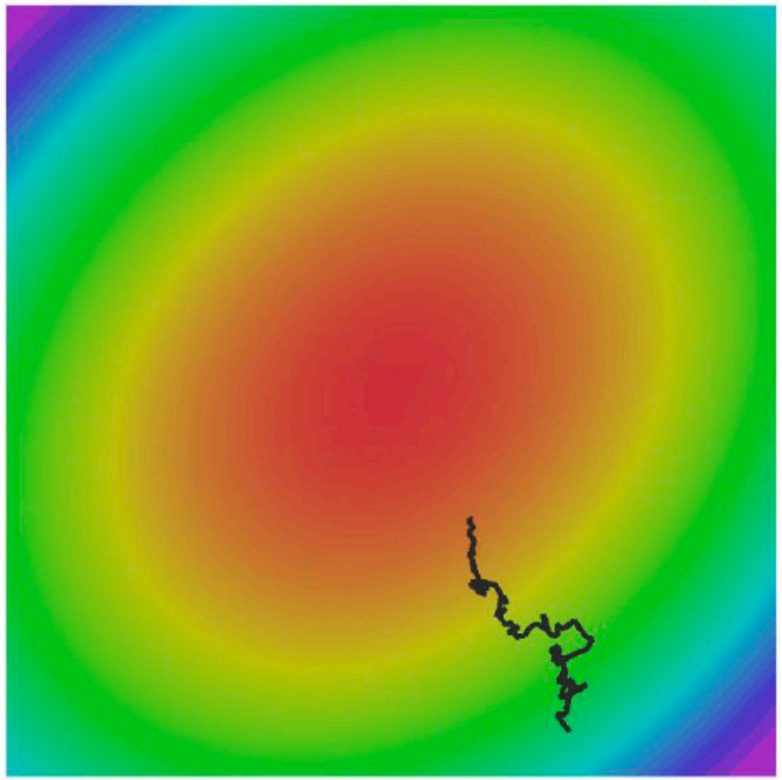


Problems of SGD

Gradients comes from minibatches → can be **noisy**

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

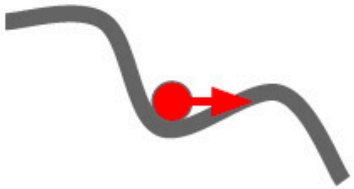
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

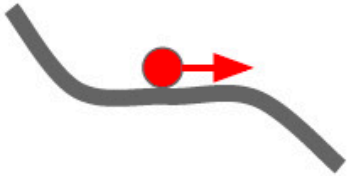
- "Velocity": running mean of gradients
- Rho gives "friction": typically rho = 0.9 or 0.99

SGD + Momentum

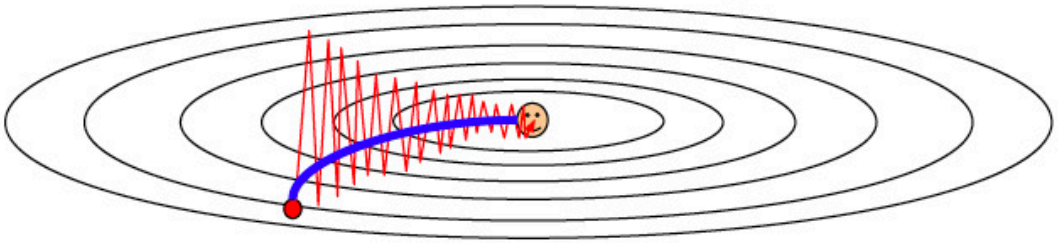
Local Minima



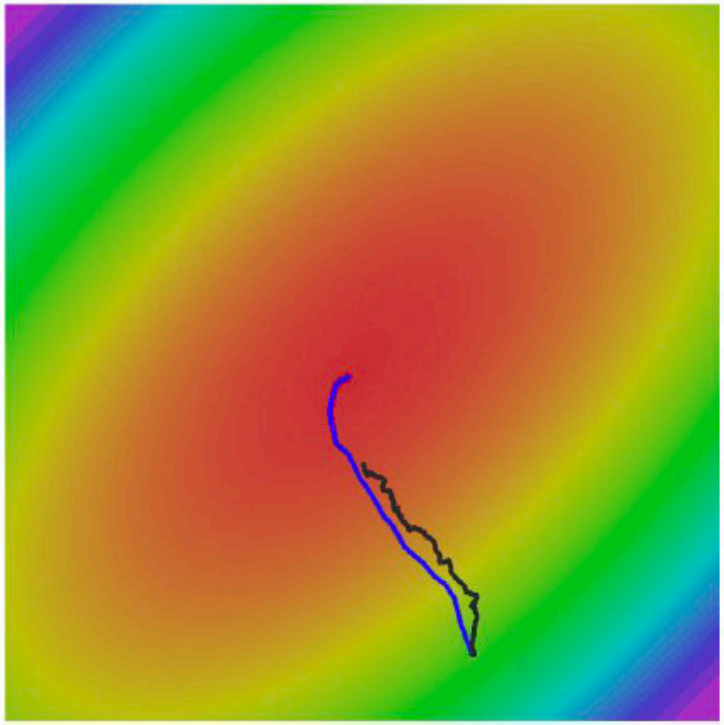
Saddle points



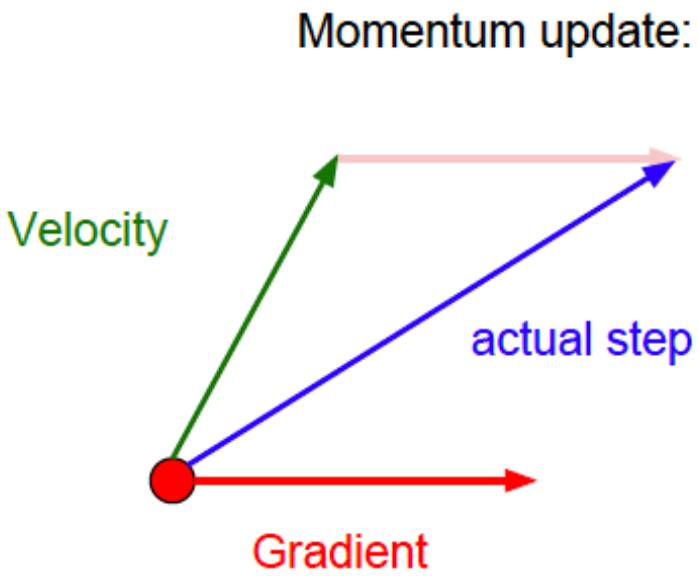
Poor Conditioning

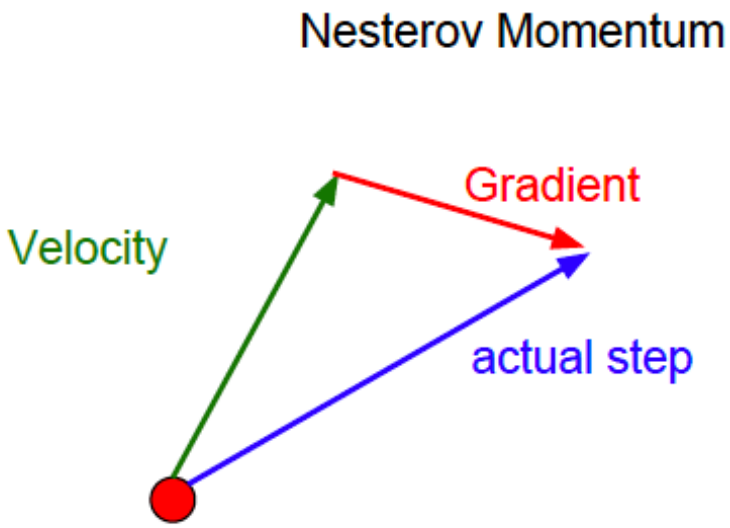


Gradient Noise

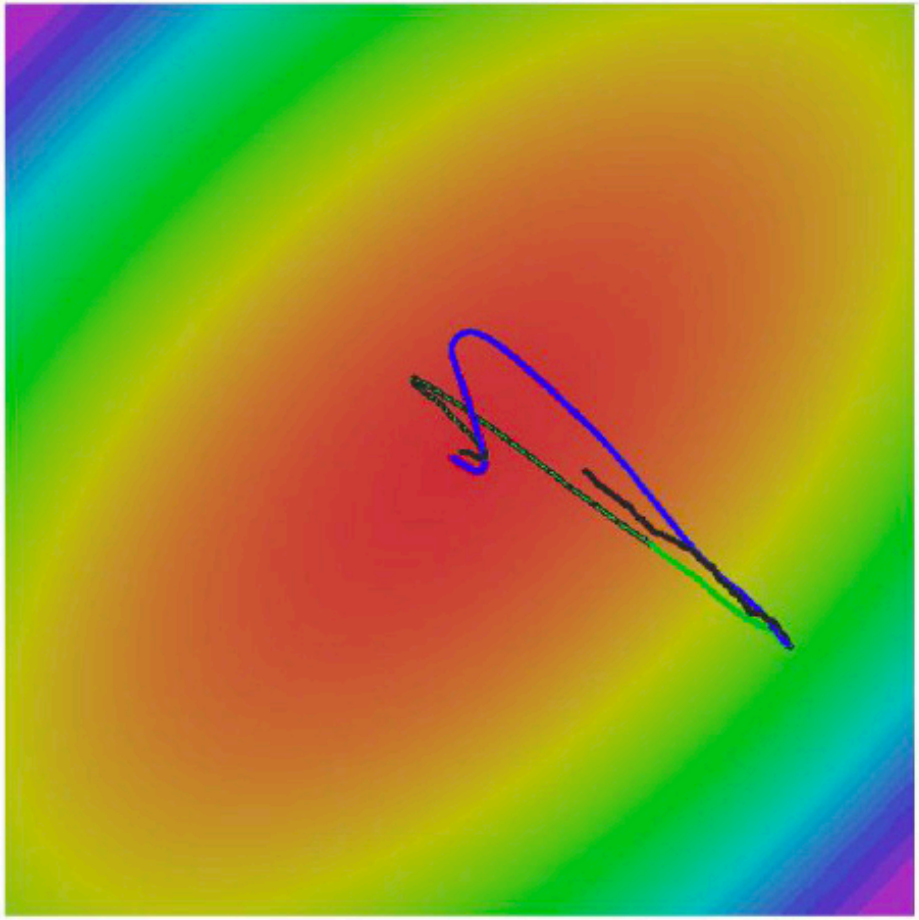


Nesterov Momemtum





Nesterov Momentum



SGD

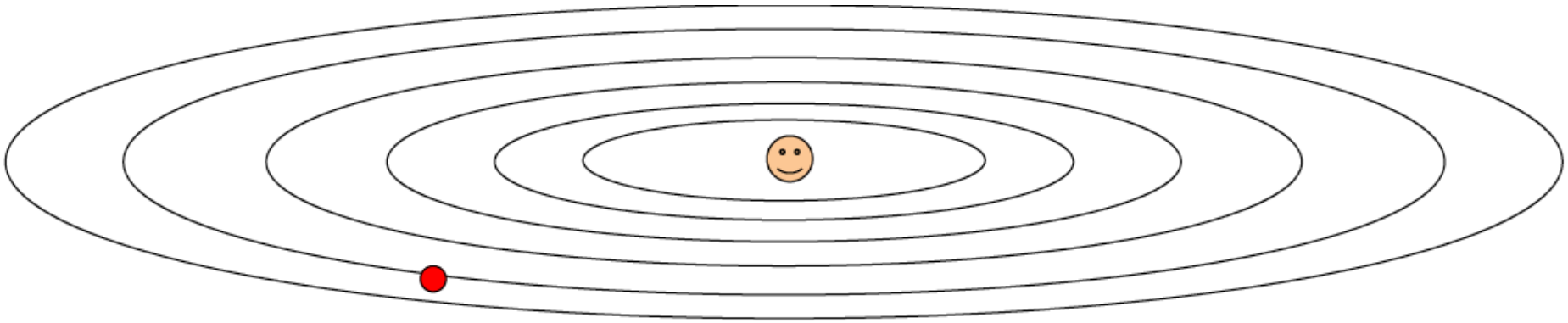
SGD+Momentum

Nesterov

AdaGrad [Duchi et al., 2011]

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension



Q: What happens with AdaGrad?

Q2: What happens to the step size over long time?

RMSProp [Tieleman and Hinton, 2012]

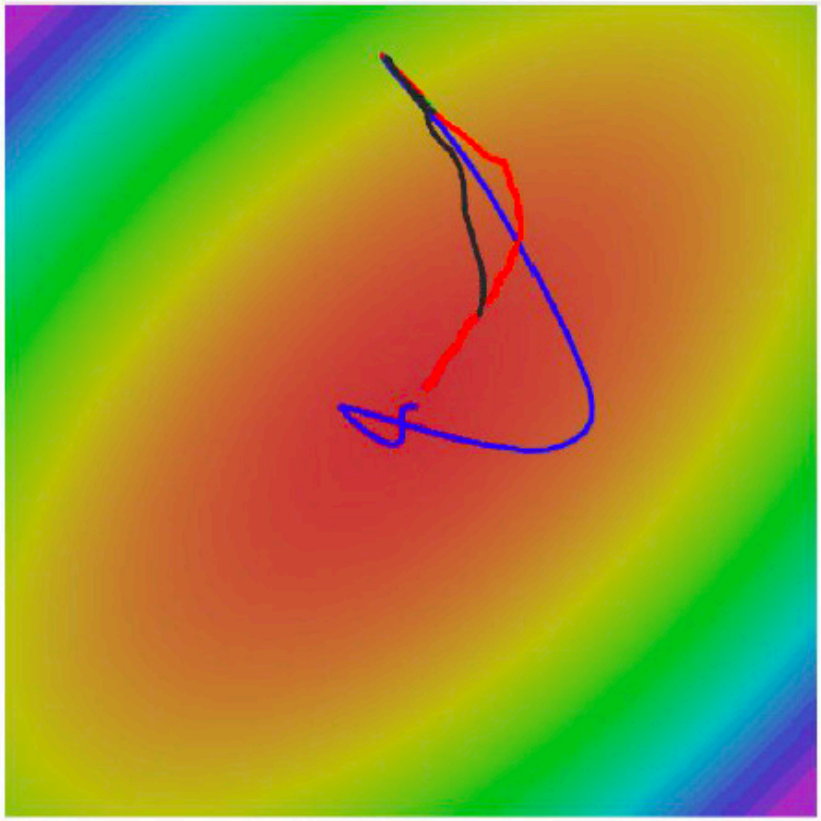
AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp



SGD

SGD+Momentum

RMSProp

Adam [Kingma and Ba, 2015]

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Adam with beta1 = 0.9,
beta2 = 0.999, and learning_rate = 1e-3 or 5e-4
is a great starting point for many models!

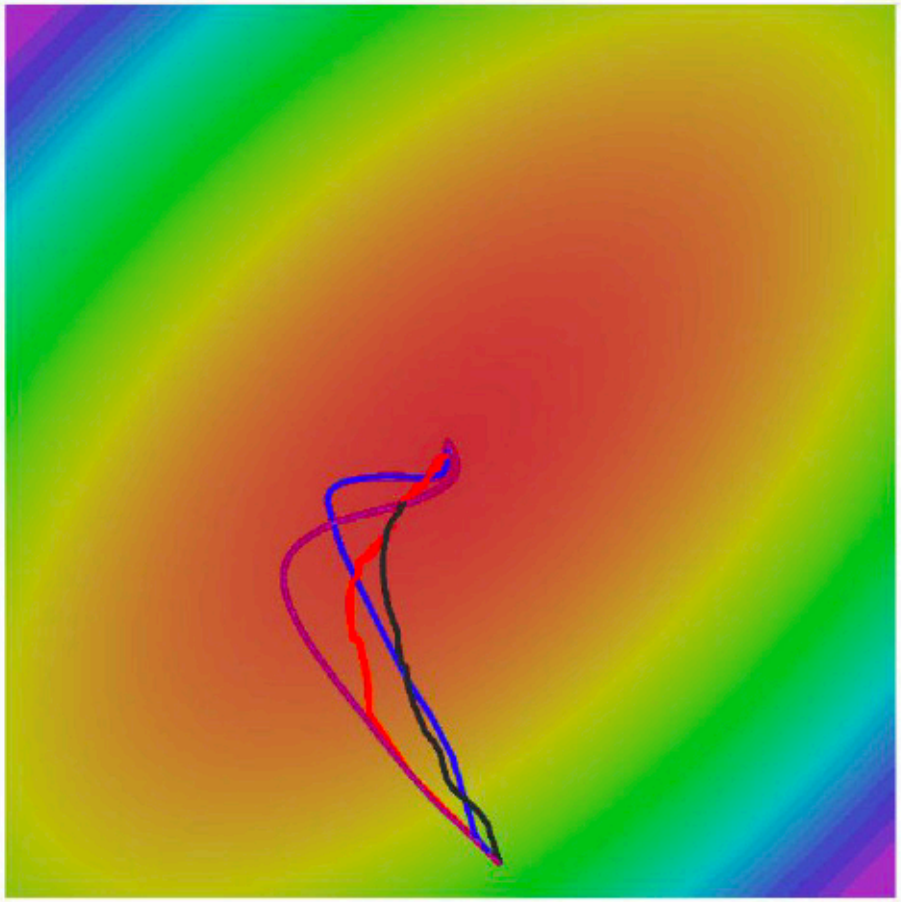
```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Adam



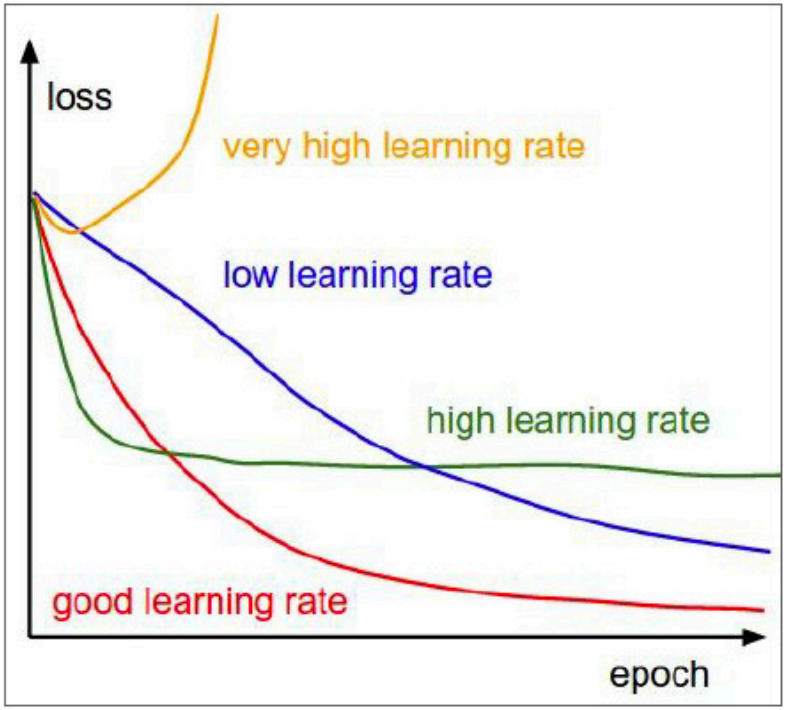
SGD

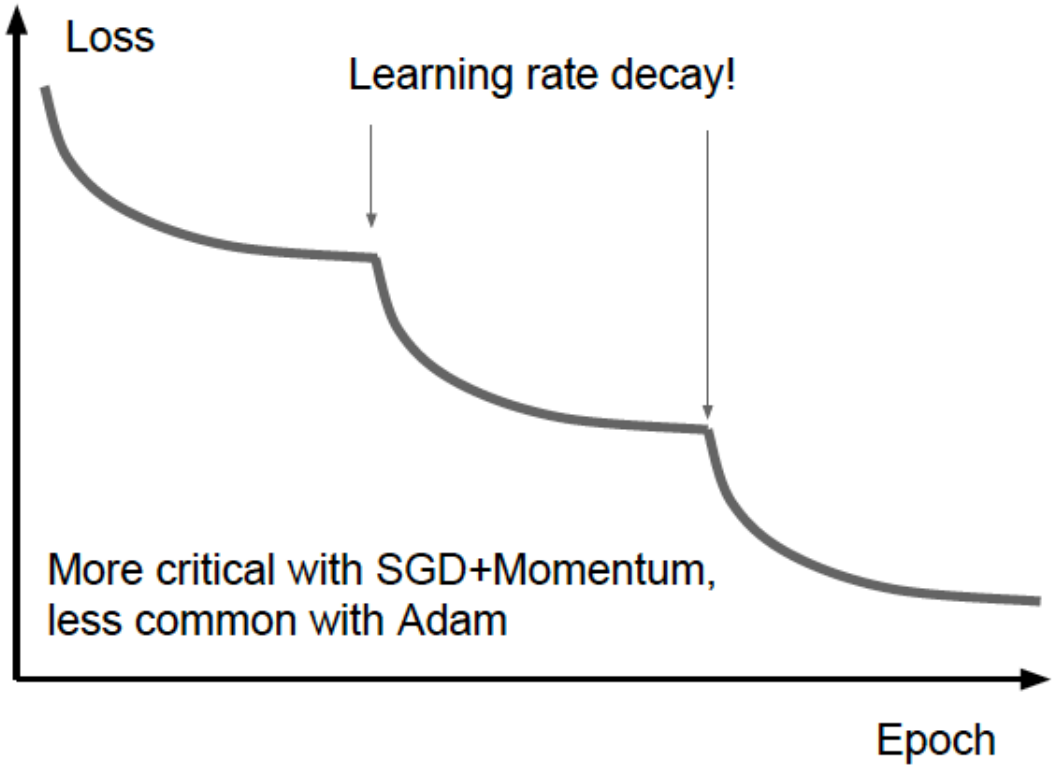
SGD+Momentum

RMSProp

Adam

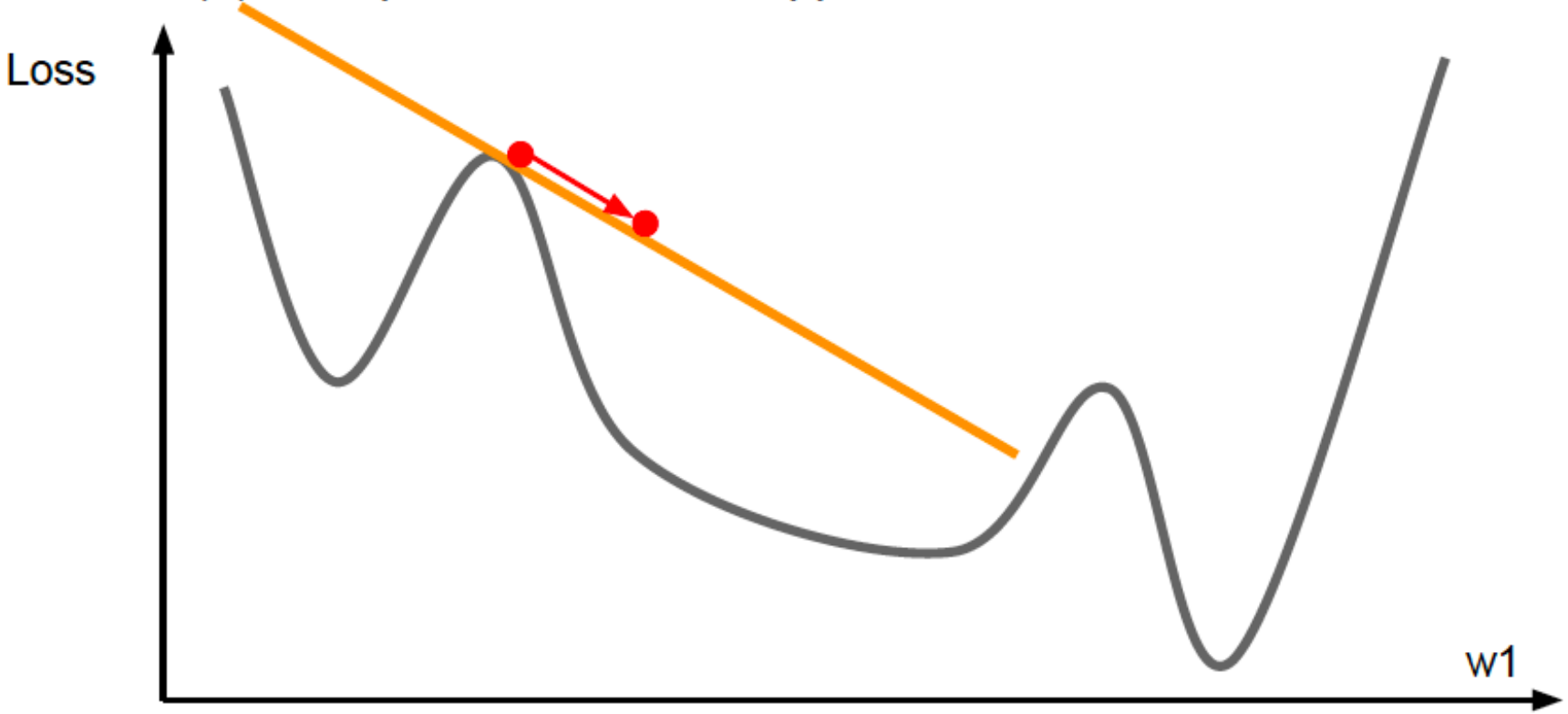
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.





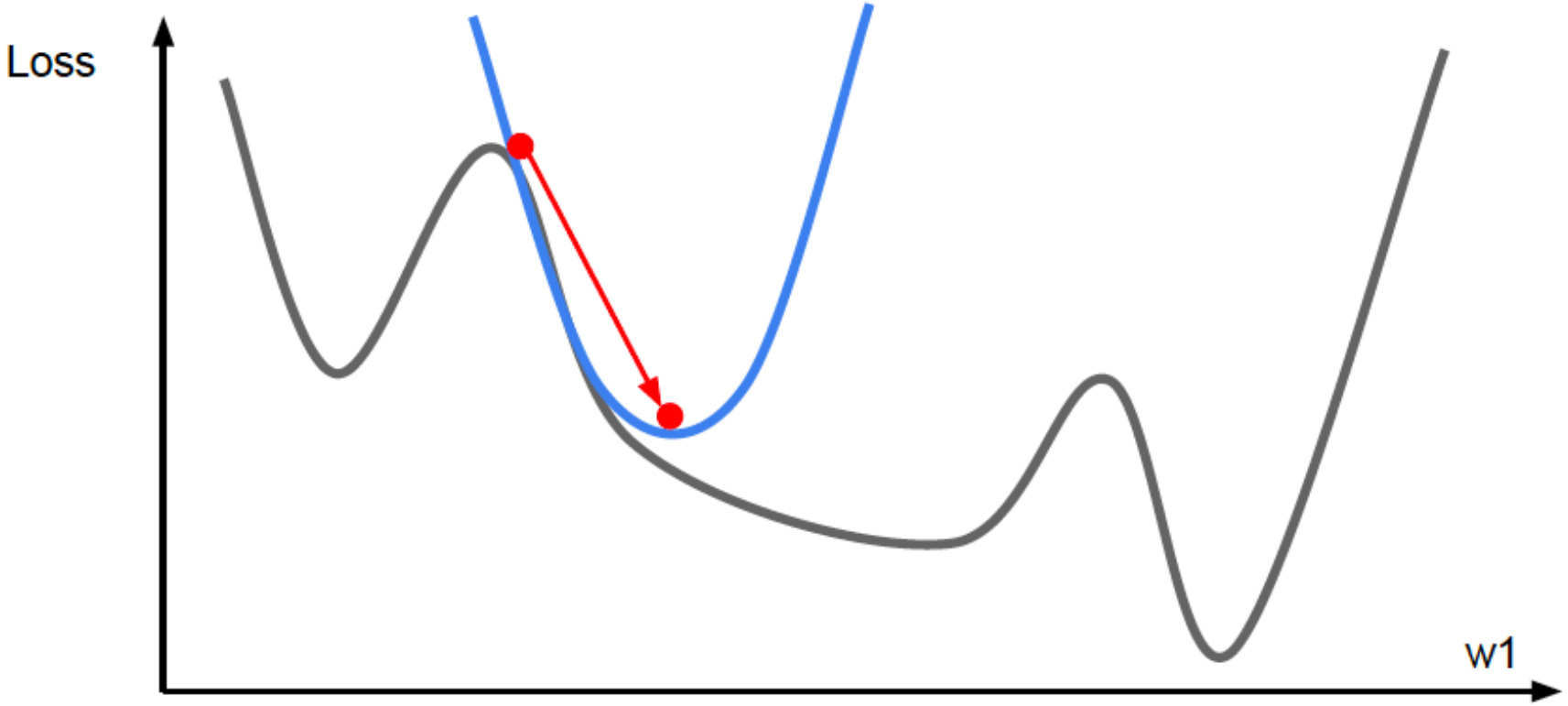
First-Order Optimization

(1) Use gradient form linear approximation
(2) Step to minimize the approximation



Second Order Optimization

(1) Use gradient **and Hessian** to form **quadratic** approximation
(2) Step to the **minima** of the approximation



Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

No hyperparameters!
No learning rate!

Hessian has O(N^2) elements
Inverting takes O(N^3)
N = (Tens or Hundreds of) Millions

Second-Order Optimization

- Quasi-Newton methods (**BGFS** most popular):
  instead of inverting the Hessian (O(n^3)), approximate inverse Hessian with rank 1 updates over time (O(n^2) each).

- **L-BFGS** (Limited memory BFGS):
  Does not form/store the full inverse Hessian.

Works well in full batch.
Does not work very well to mini-batch settings

Tensorboard

Visualizing the Graph

```
writer = tf.summary.FileWriter("/tmp/mnist_demo/1")
writer.add_graph(sess.graph)
```

```
>> tensorboard --logdir /tmp/mnist_demo/1
```

Cleaning the Graph

```
def conv_layer(input, channels_in, channels_out, name="conv"):
  with tf.name_scope(name):
    w = tf.Variable(tf.zeros([5, 5, channels_in, channels_out]), name="W")
    b = tf.Variable(tf.zeros([channels_out]), name="B")
    conv = tf.nn.conv2d(input, w, strides=[1, 1, 1, 1], padding="SAME")
    act = tf.nn.relu(conv + b)
    return tf.nn.max_pool(act, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")

def fc_layer(input, channels_in, channels_out, name="fc"):
  with tf.name_scope(name):
    w = tf.Variable(tf.zeros([channels_in, channels_out]), name="W")
    b = tf.Variable(tf.zeros([channels_out]), name="B")
    return tf.nn.relu(tf.matmul(input, w) + b)
```

```
# Setup placeholders, and reshape the data
x = tf.placeholder(tf.float32, shape=[None, 784], name="x")
x_image = tf.reshape(x, [-1, 28, 28, 1])
y = tf.placeholder(tf.float32, shape=[None, 10], name="labels")

conv1 = conv_layer(x_image, 1, 32, "conv1")
conv2 = conv_layer(conv1, 32, 64, "conv2")

flattened = tf.reshape(conv2, [-1, 7 * 7 * 64])
fc1 = fc_layer(flattened, 7 * 7 * 64, 1024, "fc1")
logits = fc_layer(fc1, 1024, 10, "fc2")
```

Cleaning the Graph

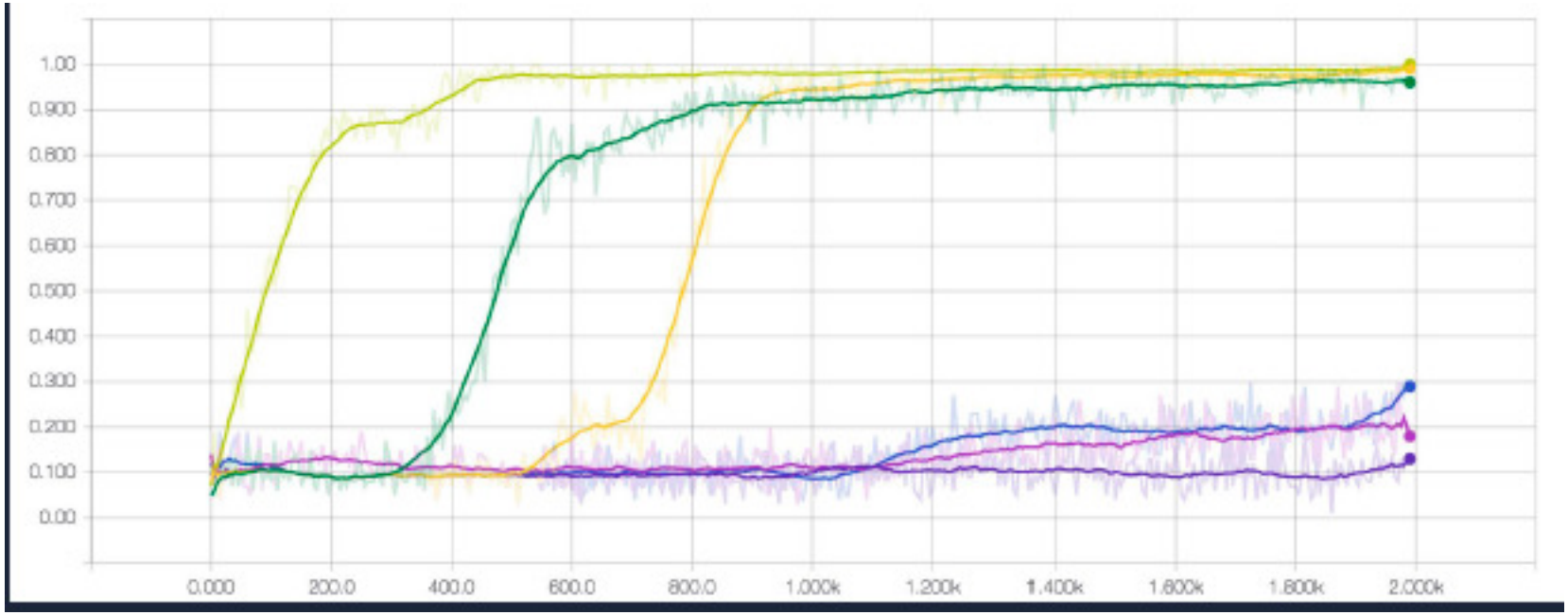
```
with tf.name_scope("xent"):
  xent = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))

with tf.name_scope("train"):
  train_step = tf.train.AdamOptimizer(1e-4).minimize(xent)

with tf.name_scope("accuracy"):
  correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
writer = tf.summary.FileWriter("/tmp/mnist_demo/2")
writer.add_graph(sess.graph)
```

```
>> tensorboard --logdir /tmp/mnist_demo/2
```

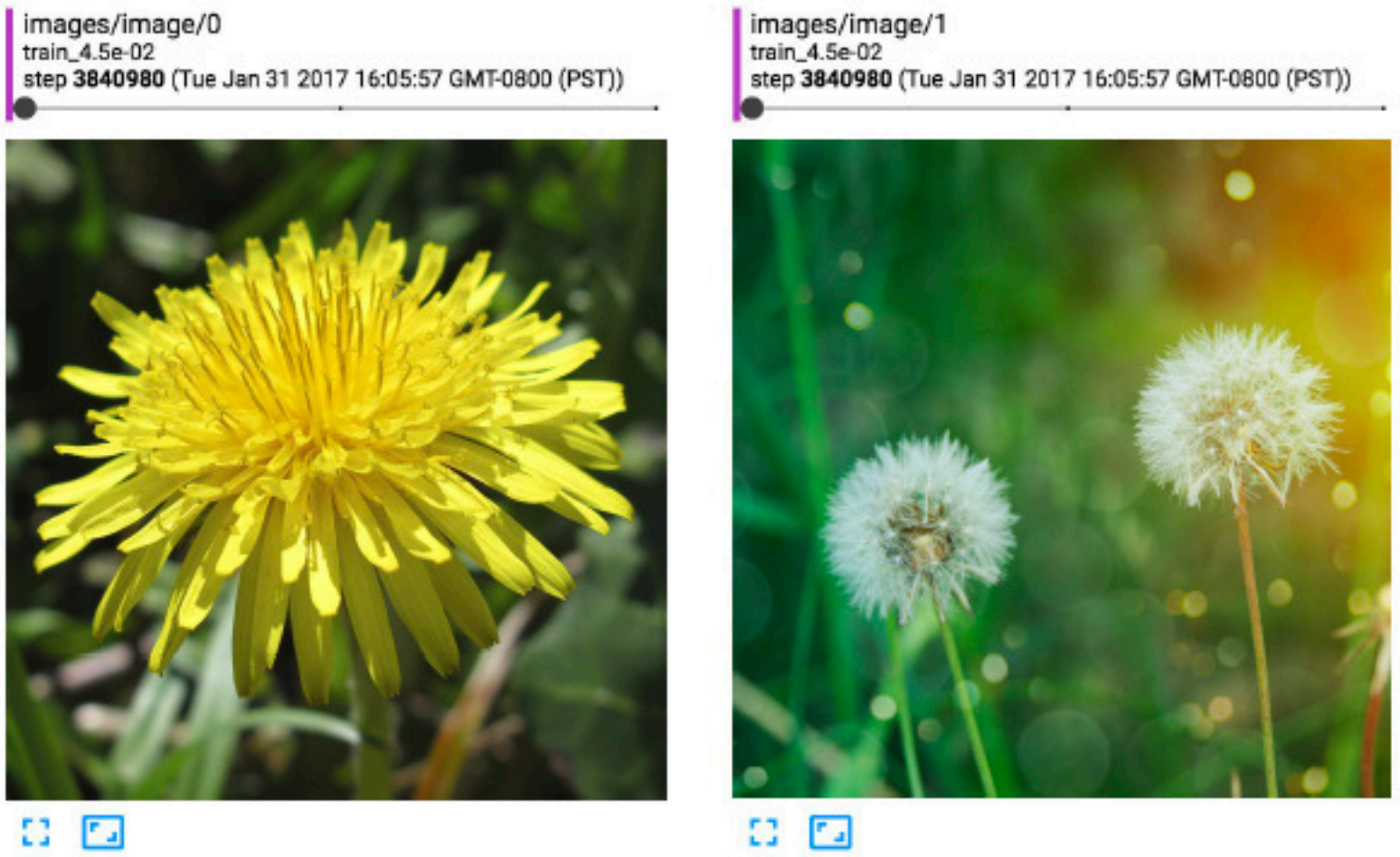
tf.summary.scalar

```
tf.summary.scalar('cross_entropy', xent)
tf.summary.scalar('accuracy', accuracy)
```



tf.summary.image

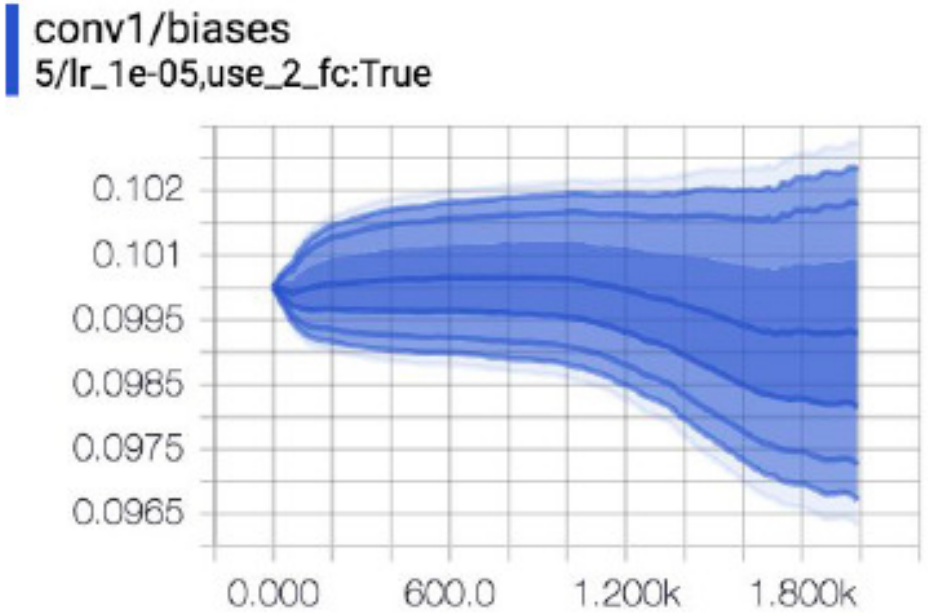
```
tf.summary.image('input', x_image, 3)
```

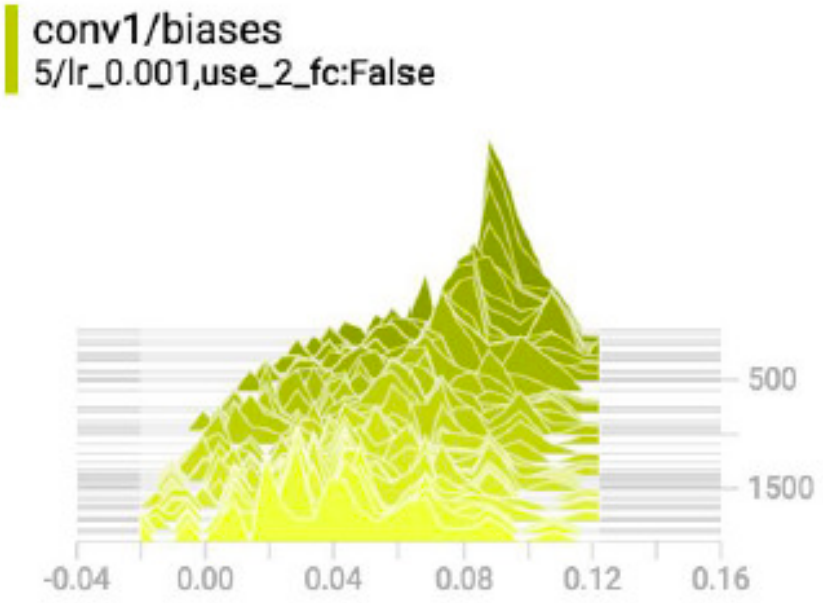


tf.summary.audio

tf.summary.histogram

```
tf.summary.histogram("weights", w)
tf.summary.histogram("biases", b)
tf.summary.histogram("activations", act)
```





Merge -> Write

```
merged_summary = tf.summary.merge_all()
writer = tf.summary.FileWriter("/tmp/mnist_demo/3")
writer.add_graph(sess.graph)
```

```
for i in range(2001):
  batch = mnist.train.next_batch(100)
  if i % 5 == 0:
    s = sess.run(merged_summary, feed_dict={x: batch[0], y: batch[1]})
    writer.add_summary(s, i)
  sess.run(train_step, feed_dict={x: batch[0], y: batch[1]})
```

Hyperparameter Search

```
# Try a few learning rates
for learning_rate in [1E-3, 1E-4, 1E-5]:

  # Try a model with fewer layers
  for use_two_fc in [True, False]:
    for use_two_conv in [True, False]:

      # Construct a hyperparameter string for each one (example: "lr_1E-3,fc=2,conv=2)
      hparam_str = make_hparam_string(learning_rate, use_two_fc, use_two_conv)

      writer = tf.summary.FileWriter("/tmp/mnist_tutorial/" + hparam_str)

      # Actually run with the new settings
      mnist(learning_rate, use_two_fully_connected_layers, use_two_conv_layers, writer)
```

감사합니다.