Overdoing linear regression with TensorFlow

Oliver Dürr

27.09.18

Outline

Background

- What is deep learning
- What is linear regression
- Why linear regression is of interest in this area
- Solving linear regression the DL way
- Gradient Descent

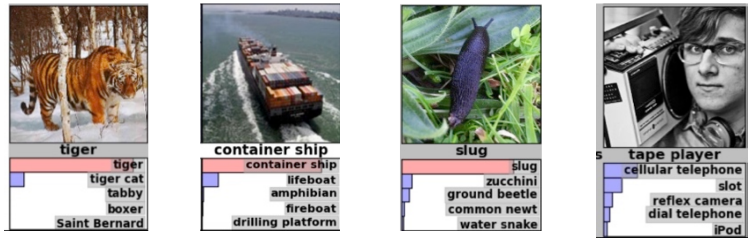
TensorFlow (as an example of a DL framework)

- Computational Graph
- Gradient Flow in a computational graph

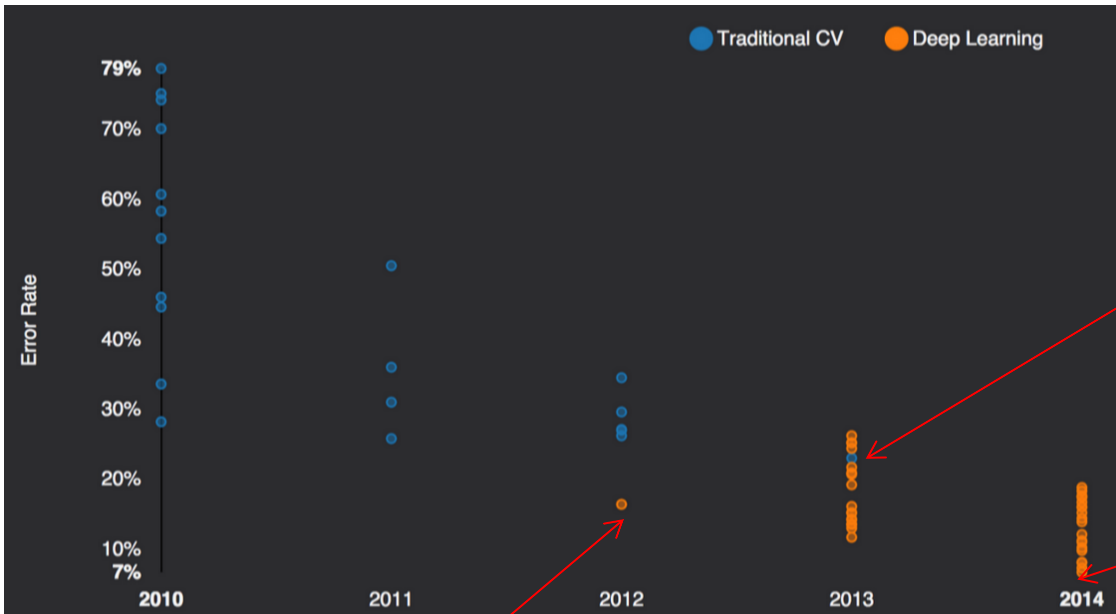
What is DL (in 4 slides)

Why DL: Imagenet 2012, 2013, 2014, 2015

1000 classes
1 Mio samples



...



Human: 5% misclassification

Only one non-CNN approach in 2013

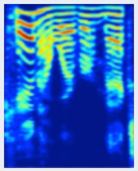
GoogLeNet 6.7%

A. Krizhevsky
first CNN in 2012
Und es hat zoom gemacht

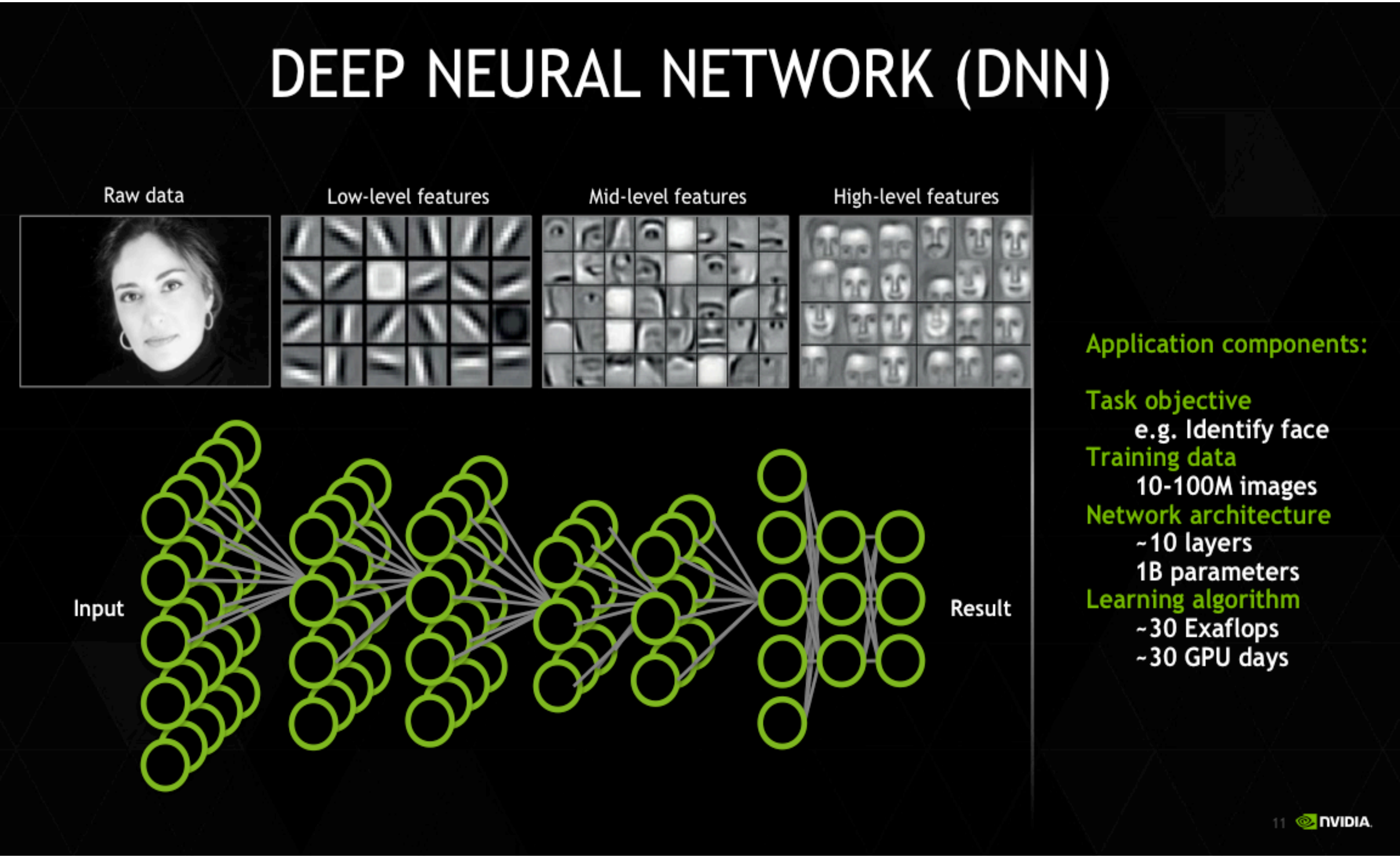
2015: It gets tougher
- 4.95% Microsoft (Feb 6 surpassing human performance 5.1%)
- 4.8% Google (Feb 11) -> further improved to 3.6 (Dec)?
- 4.58% Baidu (May 11 banned due too many submissions)
- 3.57% Microsoft (Resnet winner 2015)

Figure: https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519

Application Areas of DL

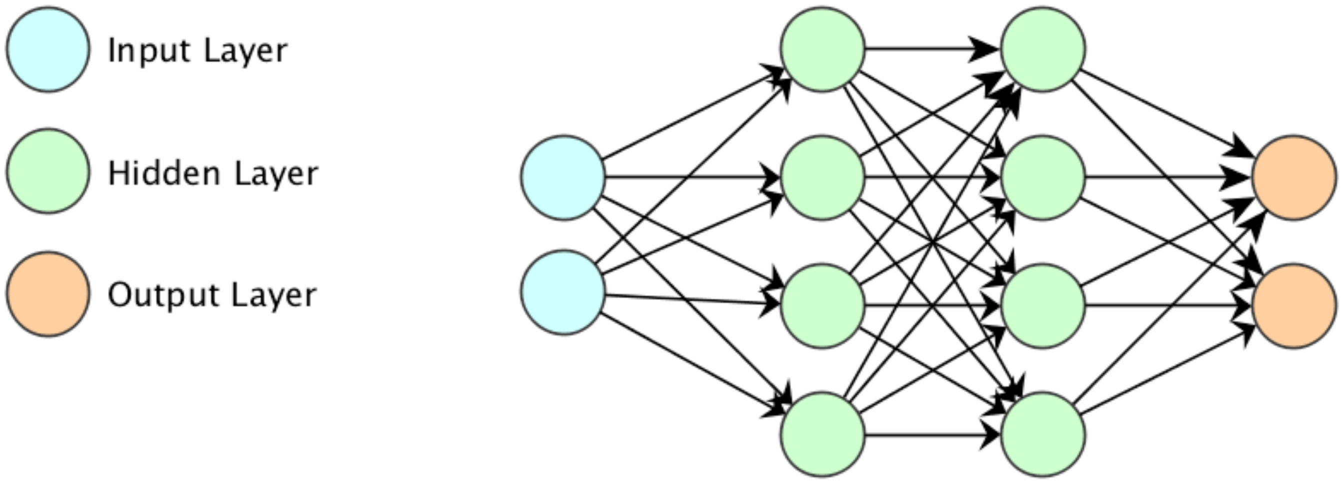
Input x to DL model	Output y of DL model	Application
Images 	Label "Tiger"	Image classification
Audio 	Sequence / Text "see you tomorrow"	Voice Recognition
ASCII-Sequences "Hallo, wie gehts?"	Unicode-Sequences "你好，你好吗?"	Translation
ASCII-Sequence This movie was rather good	Label (Sentiment) positive	Sentiment Analysis
Structured Data city='london', device='mobile'	P("user clicks on add")	Click prediction
 Reward for last Action State of the world	 Action	Deep Reinforcement Learning e.g. GO

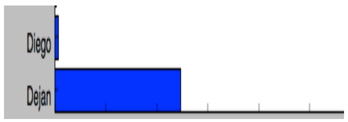
Main Idea in DL



Learn hierarchy of features

A DL model: Fully Connected aka MLP





- The input: e.g. intensity values of pixels of an image
- Information is processed layer by layer
- Output: probability that image belongs to certain person
- Arrows are **weights (these need to be learned)**
- For image and text there are specialized architectures (CNN, RNN)

The learning process

- Three ingredients
  - A **model with weights**, which needs to be learned
  - **Data with labels** (reinforcement is a bit different)
  - A **loss function**, describing who good the data is fit with the model

- Learning is tuning the weights to fit the training data

Linear Regression

An introductory remark

Judea Pearl – fellow ACM, Turing Award winner

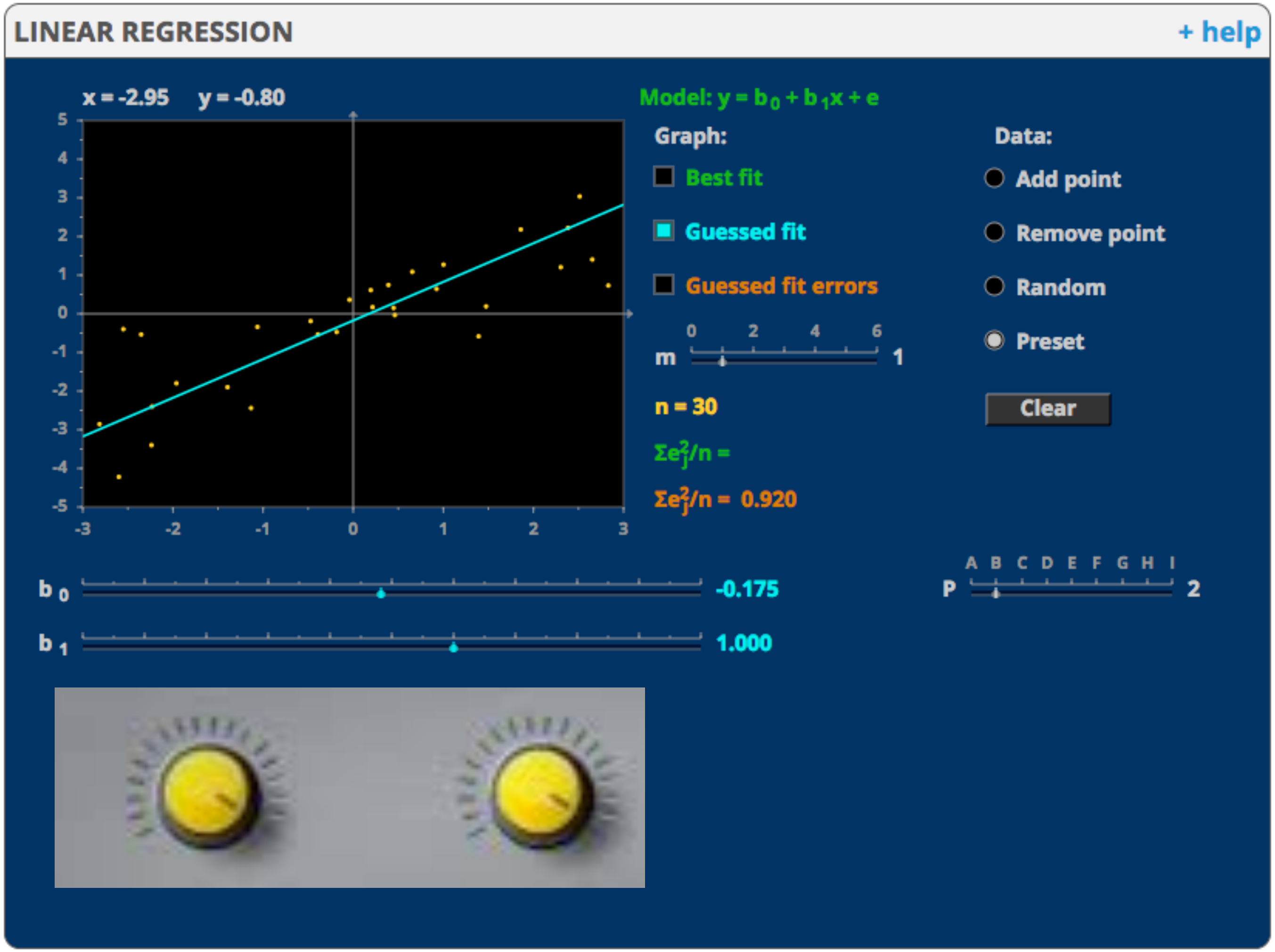
<<All the impressive achievements of deep learning amount to just curve fitting>>
Judea Pearl, 2018

Let's look at the simplest curve fitting model: linear regression

Linear Regression: See Backbord

- Model \hat y = a * x + b
- Linear Regression as a mother of all networks
- Training Data
  - (x and y pairs)
  - Plot
- Kriterium RSS

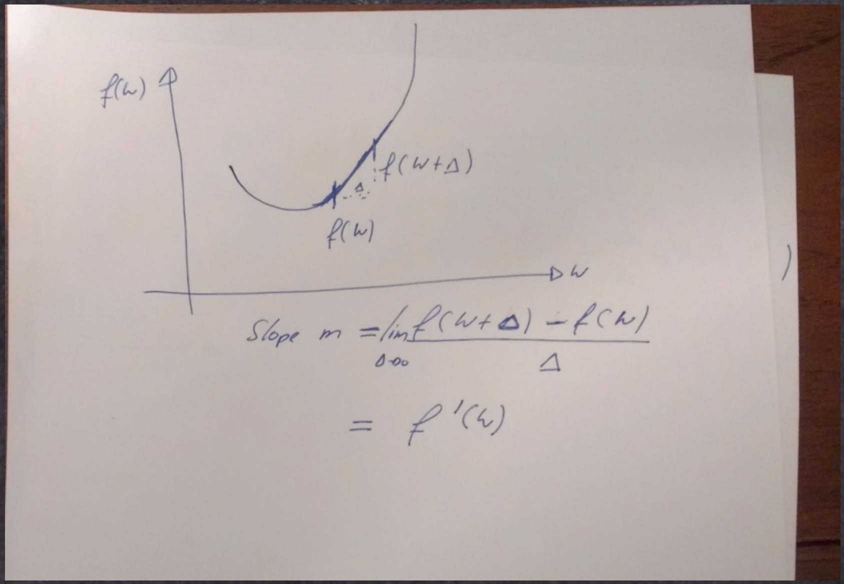
Tuning the weights

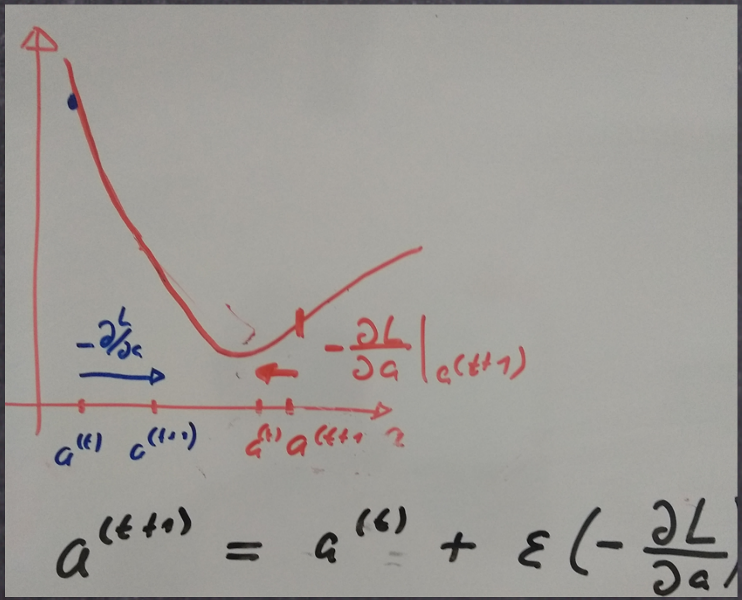


http://mathlets.org/mathlets/linear-regression/

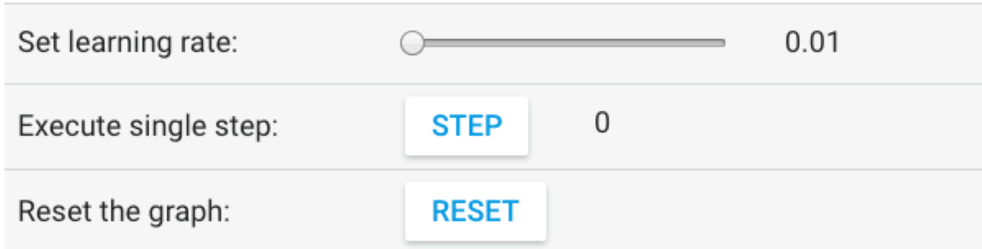
Gradient Descent: 1-D function See Backbord

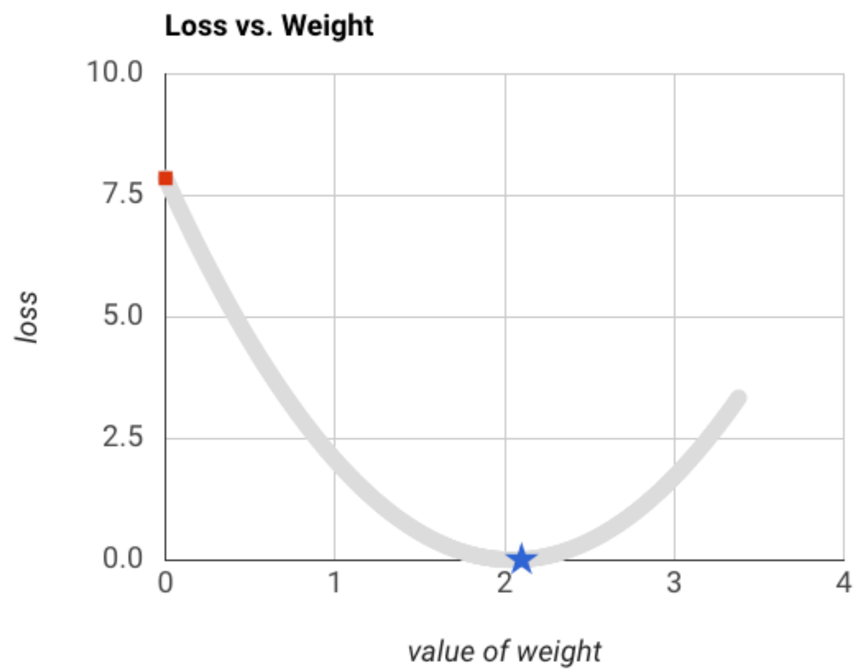
Gradient and Gradient Descent





Demo for influence of step size

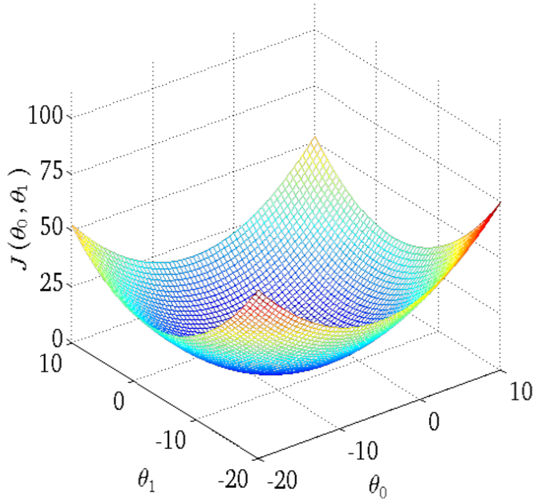


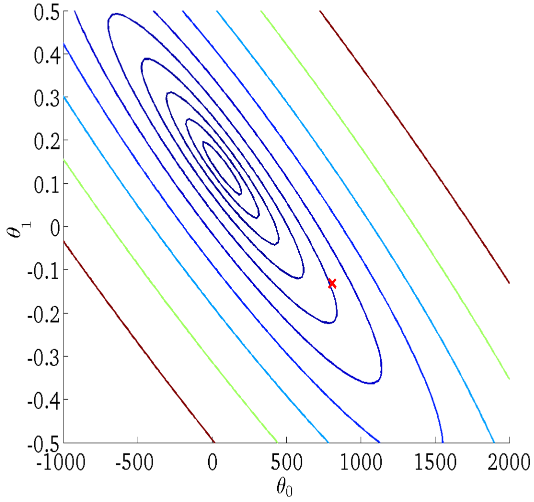


https://developers.google.com/machine-learning/crash-course/fitter/graph

Optimization in 2-D

- 2 equivalent representations





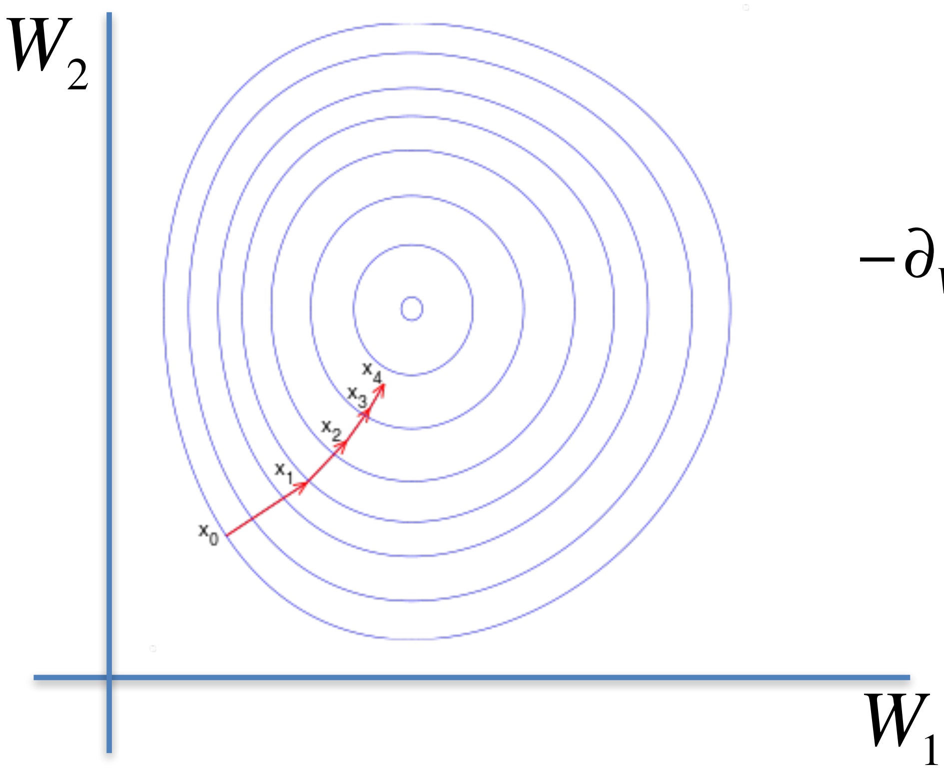
Slide credit: Thilo, wikipedia

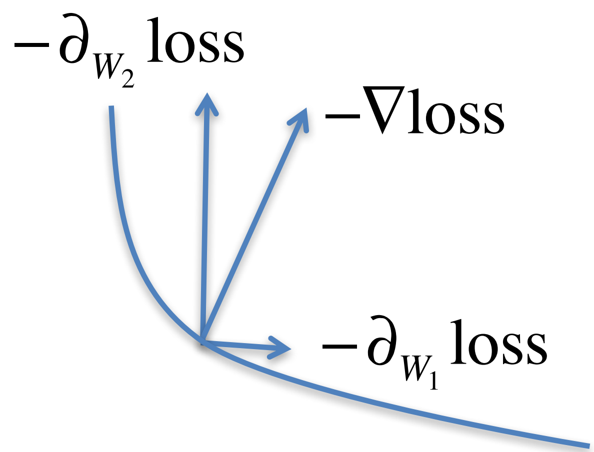
Optimization in 2-D

- Gradient Descent

  Gradient is perpendicular to levels

$$W_i^{t+1} = W_i^{t} - \varepsilon\, \partial_{W_i} \text{loss}$$





Slide credit: wikipedia

Gradient Descent

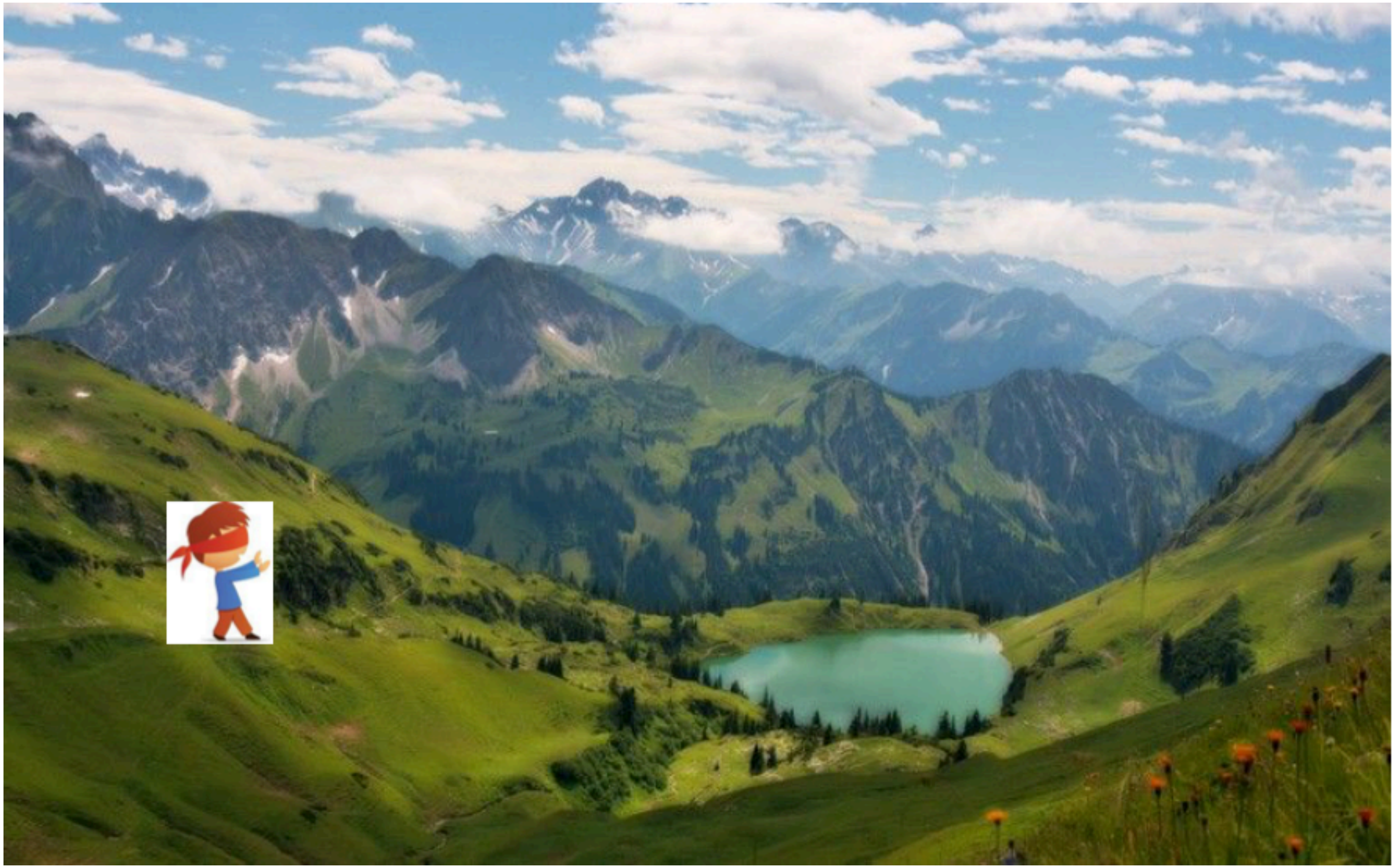
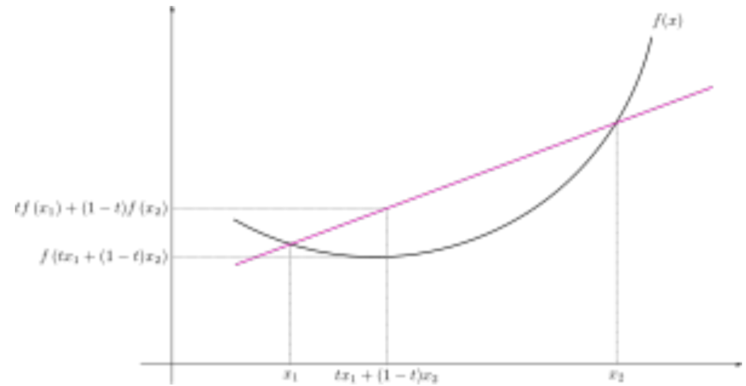


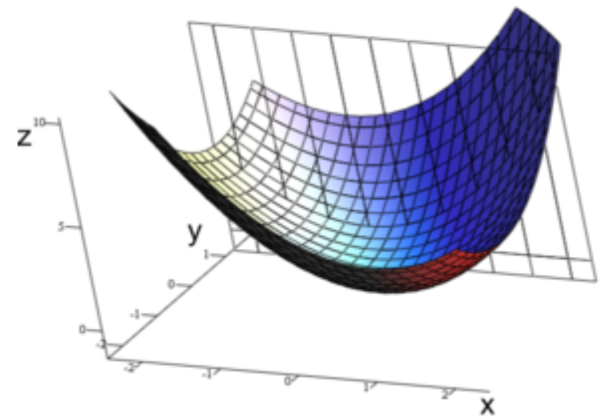
Figure shows a 2 dimensional loss function.
In DL Millions!
We just know the current value (blind)

Slide from cs229

Local vs. Global Minima

- If the loss is convex, gradient descent converges to local minima if step-size is small enough
- Linear regression is a convex problem
- Deep Learning is by far not a convex problem. Still works in practice (one of the miracles of DL)





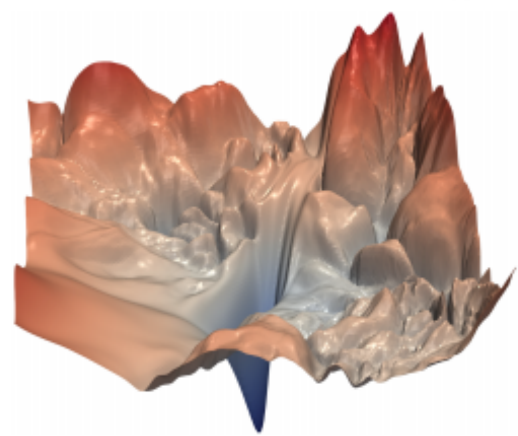


Image credit: Wikipedia, https://openreview.net/pdf?id=HkmaTz-0W

Details left out

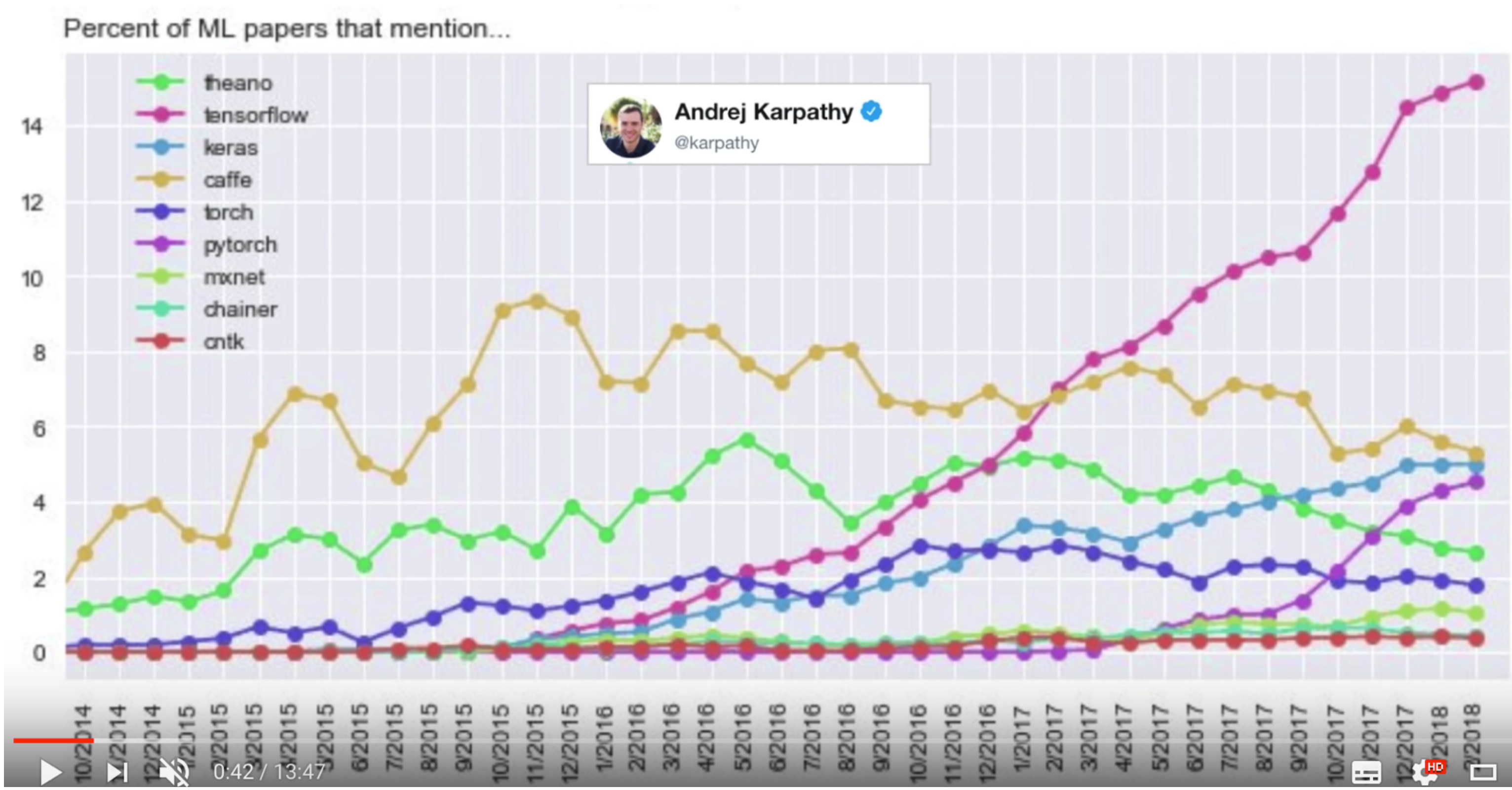
- Mini-batch stochastic gradient descent
  - Sometimes we cannot use all of the data points ➔ just use a random subset

- Overfitting problematic
  - When the models get to complicated (many weights) models can learn the particularities of the training data

- Deep Learning is often used for classification problems
  - Here we had regression with RSS
  - Classification similar but different loss functions

Introduction to TF

Deep Learning frameworks



- TensorFlow

Keras
pytorch

Two mayor library designs

We need gradients, of functions with 100 million+ weights.

Two design principles

- Static Computational Graph (build and run the graph in 2 steps)
  - Theano
  - TensorFlow

- Autograd
  - Pytorch
  - TensorFlow Eager

What is TensorFlow

- It's API about **tensors**, which flow in a computational graph

https://www.tensorflow.org/

- What are **tensors**?

What is a tensor?

In this course we only need the simple and easy accessible definition of Ricci:

Definition. A tensor of type (p, q) is an assignment of a multidimensional array

$$T^{i_1 \dots i_p}_{j_1 \dots j_q}[\mathbf{f}]$$

to each basis $\mathbf{f} = (\mathbf{e}_1,\dots,\mathbf{e}_n)$ of a fixed n-dimensional vector space such that, if we apply the change of basis

$$\mathbf{f} \mapsto \mathbf{f} \cdot R = \left(\mathbf{e}_i R^i_1, \dots, \mathbf{e}_i R^i_n\right)$$

Just kidding...

then the multidimensional array obeys the transformation law

$$T^{i'_1 \dots i'_p}_{j'_1 \dots j'_q}[\mathbf{f} \cdot R] = (R^{-1})^{i'_1}_{i_1} \cdots (R^{-1})^{i'_p}_{i_p} \, T^{i_1, \dots, i_p}_{j_1, \dots, j_q}[\mathbf{f}] \, R^{j_1}_{j'_1} \cdots R^{j_q}_{j'_q} \, .$$

Sharpe, R. W. (1997). Differential Geometry: Cartan's Generalization of Klein's Erlangen Program. Berlin, New York: Springer-Verlag. p. 194. ISBN 978-0-387-94732-7.

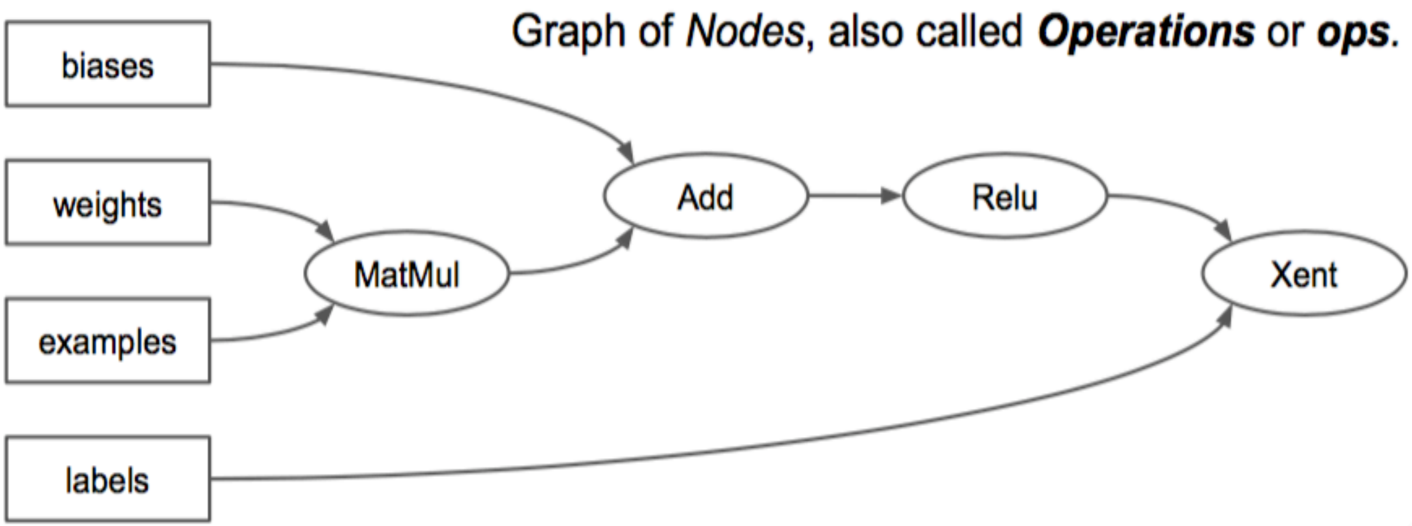
What is a tensor?

For TensorFlow: A tensor is an array with several indices (like in numpy). Order are number of indices and shape is the range.

```
In [1]: import numpy as np

In [2]: T1 = np.asarray([1,2,3]) #Tensor of order 1 aka Vector
        T1

Out[2]: array([1, 2, 3])

In [3]: T2 = np.asarray([[1,2,3],[4,5,6]]) #Tensor of order 2 aka Matrix
        T2

Out[3]: array([[1, 2, 3],
               [4, 5, 6]])

In [4]: T3 = np.zeros((10,2,3)) #Tensor of order 3 (Volume like objects)

In [6]: print(T1.shape)
        print(T2.shape)
        print(T3.shape)

        (3,)
        (2, 3)
        (10, 2, 3)
```

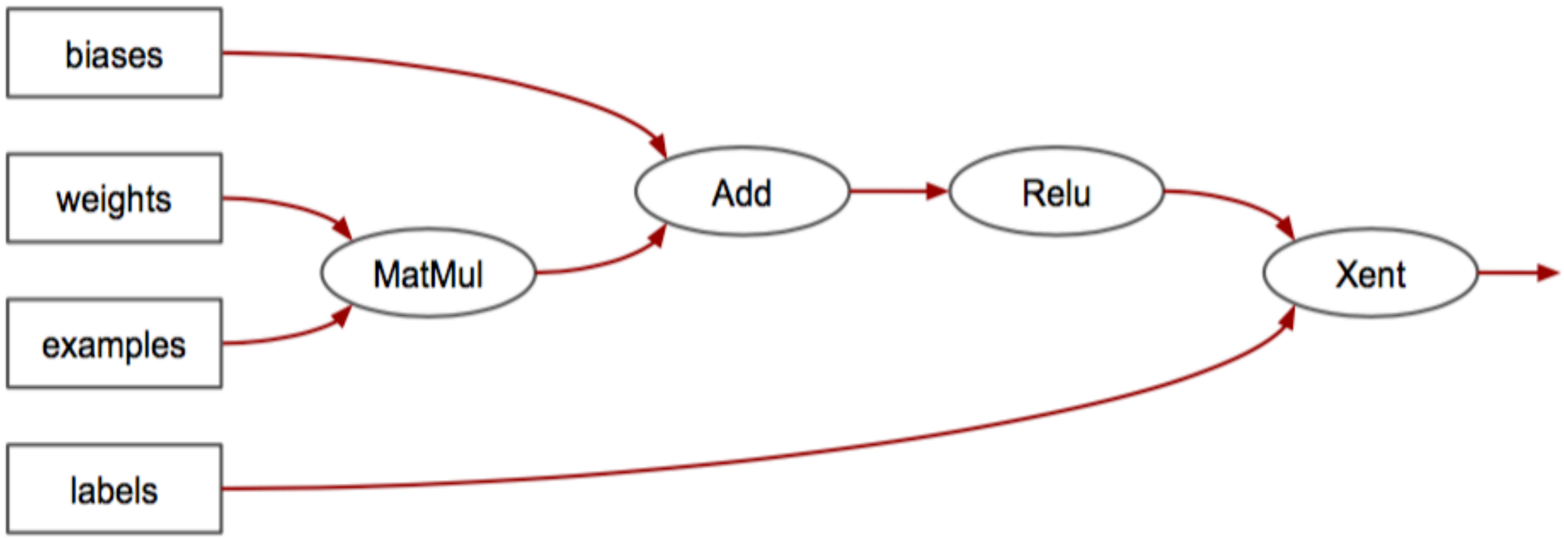
Computations in TensorFlow (and Theano)

- Computation is expressed as a dataflow graph



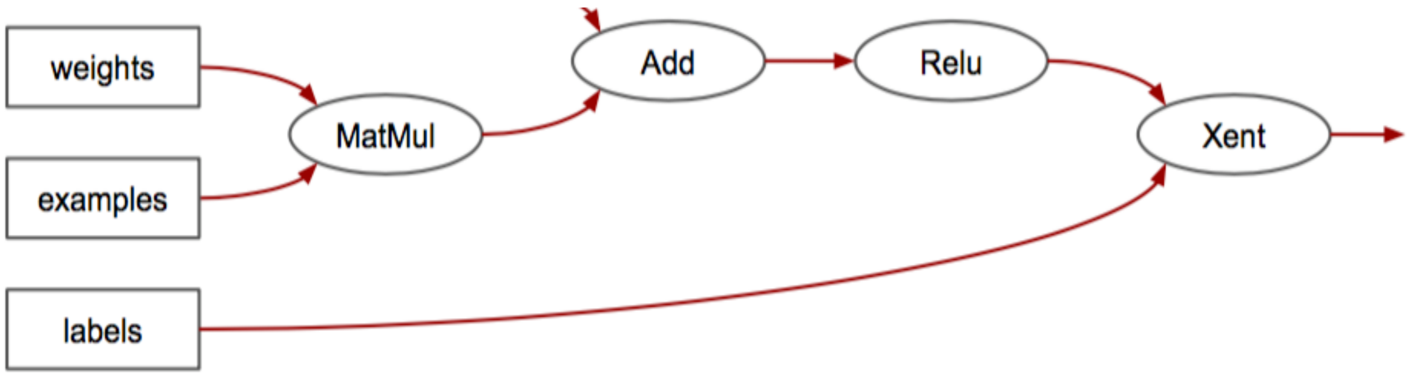
Computations in TensorFlow (and Theano)

- Edges are N-dimensional Arrays: Tensors



Summary

- The computation in TF is done via a computational graph



- The nodes are ops
- The edges are the flowing tensors

TensorFlow: Computation in 2 steps

- Computations are **done in 2 steps**
  - **First:** Build the graph
  - **Second:** Execute the graph
- Both steps can be done in many languages (python, C++, R)
  - Best supported so far is python
- Graph can be trained and ported on different devices
  - TPU
  - GPU
  - Embedded System like mobile phones
- Graph can be optimized
  - XLA optimization

Building the graph (python)

$$10\begin{pmatrix} 3 & 3 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix}=120$$

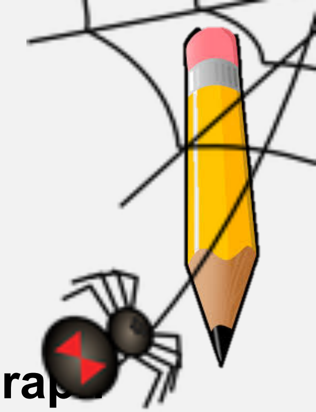
In [1]: numpy

```
import numpy as np
m1 = np.array([[3., 3.]])
m2 = np.array([[2.],[2.]])
10 * np.dot(m1,m2)
```

Out[1]:

```
array([[ 120.]])
```

Be the spider who knits a computational graph



$$10\begin{pmatrix} 3 & 3 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix} = 120$$

Translate the following TF code in a graph

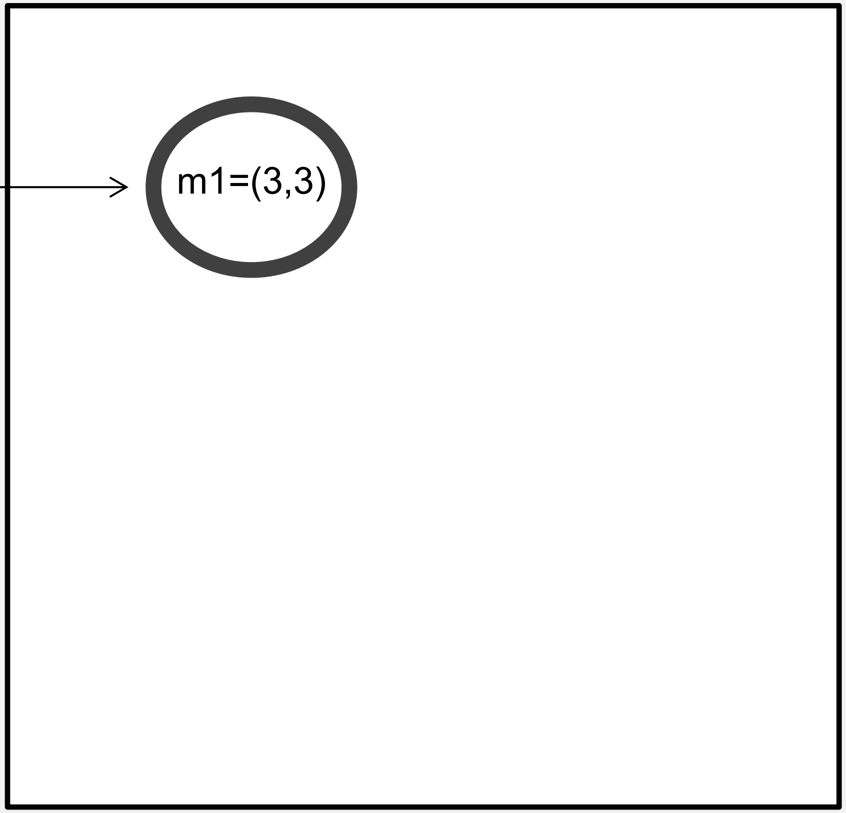
TensorFlow: Building the graph

wipes the graph

```
import tensorflow as tf
# We construct a graph (we write to the default graph)
# make first sure the default graph is empty
tf.reset_default_graph()
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

Quite much happen in here!

Finish the computation graph



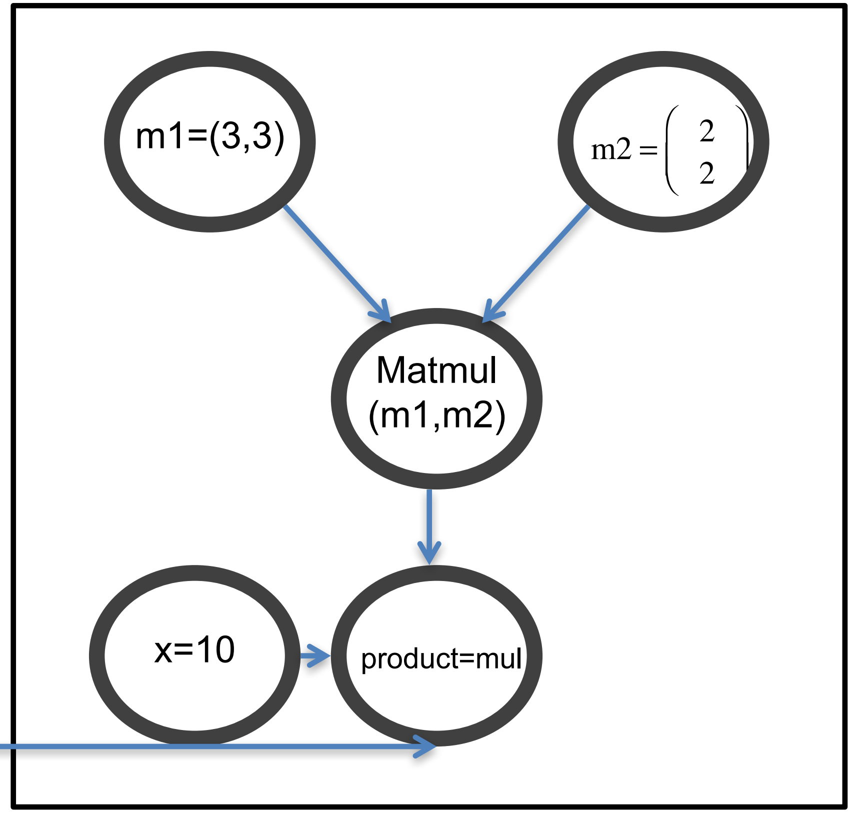
Be the spider who knits the computational graph

Translate the following TF code in a graph

TensorFlow: Building the graph

```
import tensorflow as tf
# We construct a graph (we write to the default graph)
# make first sure the default graph is empty
tf.reset_default_graph()
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

Finish the computation graph



TensorFlow: Executing the graph

```
In [4]:

sess = tf.Session()
res = sess.run(product)
print(res)
sess.close()

[[ 120.]]
```

Building the graph (Numpy vs TensorFlow)

$$10\begin{pmatrix}3 & 3\end{pmatrix}\begin{pmatrix}2\\2\end{pmatrix}=120$$

In [1]: numpy

```python
import numpy as np
m1 = np.array([[3., 3.]])
m2 = np.array([[2.],[2.]])
10 * np.dot(m1,m2)
```

Out[1]:

```
array([[ 120.]])
```

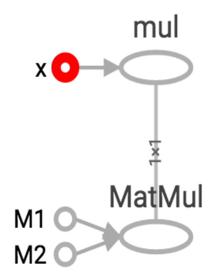
TensorFlow: Building the graph

```python
import tensorflow as tf
# We construct a graph (we write to the default graph)
# make first sure the default graph is empty
tf.reset_default_graph()
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

In [4]: TensorFlow: Executing the graph

```python
sess = tf.Session()
res = sess.run(product)
print(res)
sess.close()
```

```
[[ 120.]]
```



mul/x
Operation: Const

Attributes (2)
dtype {"type":"DT_FLOAT"}
value {"tensor": {"dtype":"DT_FLOAT","tensor_shape":{},"float_val":10}}

Inputs (0)

Outputs (0)

Remove from main graph

Session vs Graph

- A graph is the abstract definition of the calculation

- A session **is a concrete realization**
  - It places the ops on physical devices such as GPUs
  - It initializes variables
  - We can feed and fetch a session (see next slides)

```
sess = tf.Session()
… #do stuff
sess.close() #Free the resources (TF eats all mem on GPU!)
```

Alternatively use the with construct

```
with tf.Session()as sess:
    … #do stuff
#Free the resources when leaving the scope of with
```

Gradient Descent in TensorFlow

- In Theano and TensorFlow the Framework does the calculation of the gradient for you (autodiff)
- You just have to provide a graph

```
# loss has to be defined symbolically
train_op = tf.train.GradientDescentOptimizer(0.0001).minimize(loss)

…
for e in range(epochs): #Fitting the data for some epochs
  _, res = sess.run([train_op, loss], feed_dict={x:x_data, y:y_data})
```

Look at the source luke

Exercises at:

https://tensorchiefs.github.io/linear_regression/

Excercises

Backup

Gradient flow in a computational graph: local junction

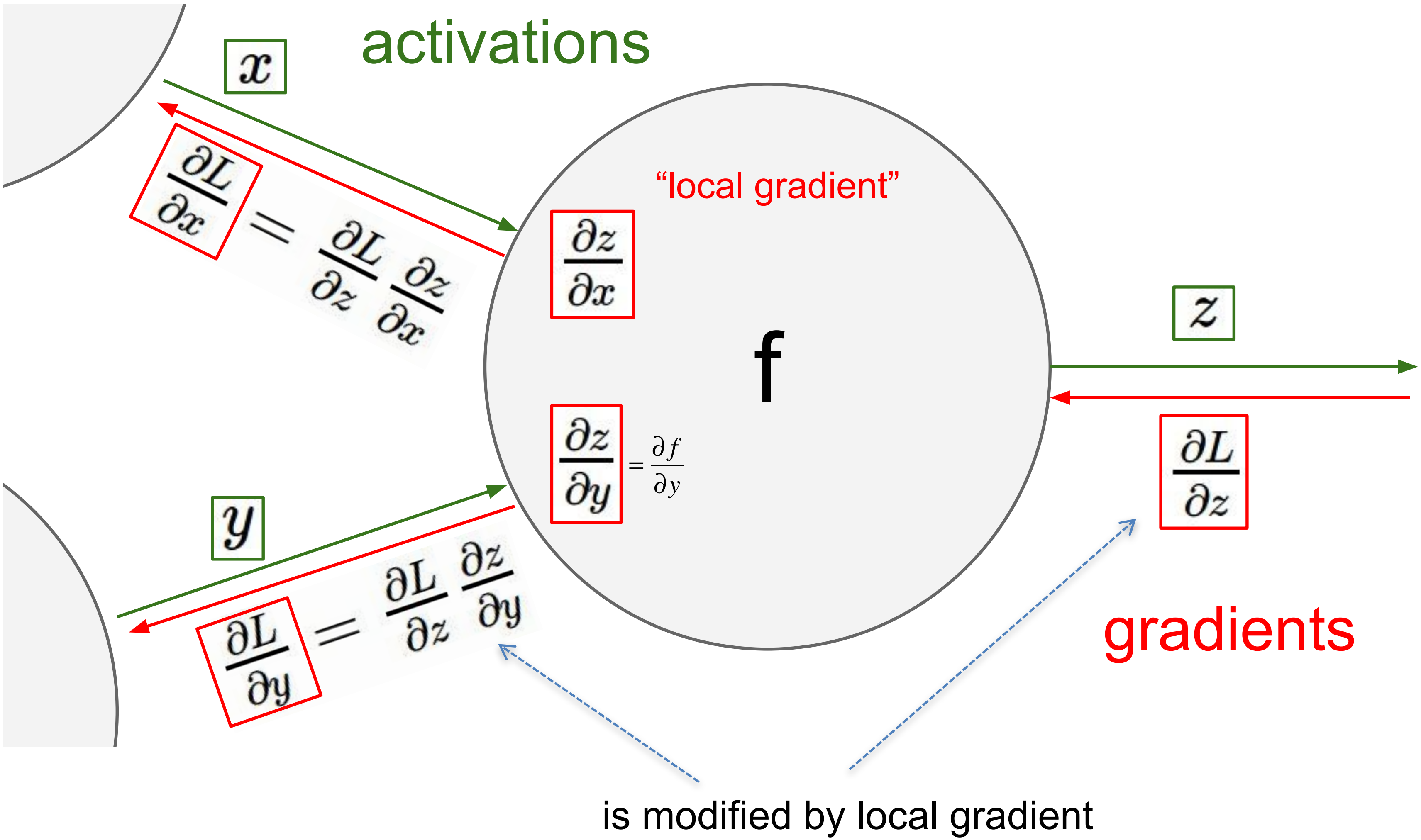
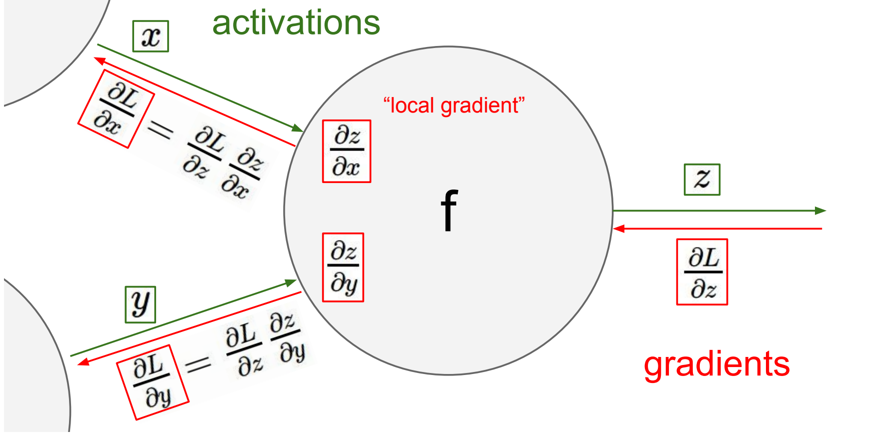


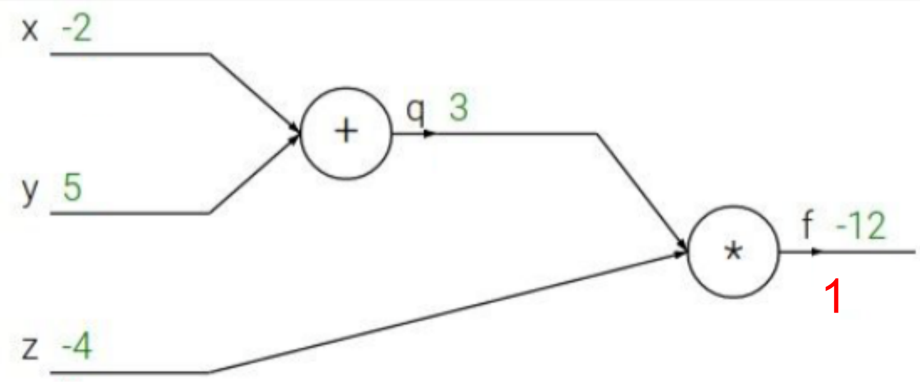
Illustration: http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

Example

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

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

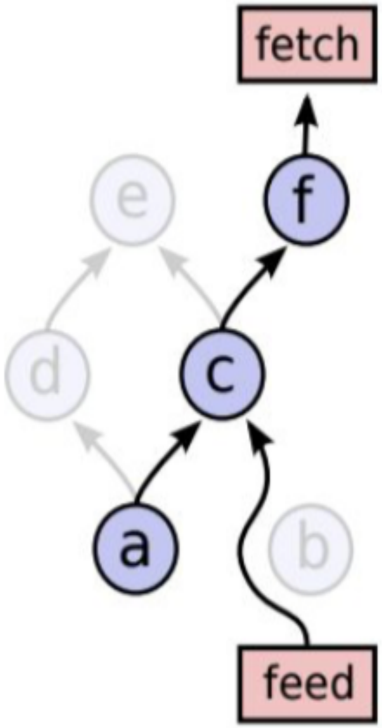




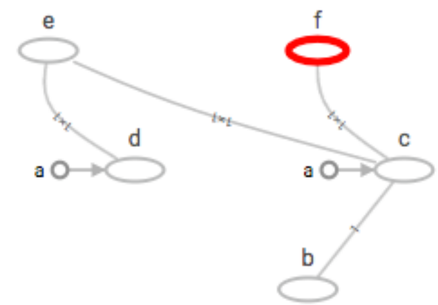
$$\frac{\partial(\alpha + \beta)}{\partial \alpha} = 1 \qquad \frac{\partial(\alpha * \beta)}{\partial \alpha} = \beta$$

➔Multiplication do a switch

Computations using feeding and fetching




```
a = tf.constant([[1]], name='a')
b = tf.placeholder(dtype='int32', shape=[1], name='b')
d = tf.identity(a,name='d')
c = tf.multiply(a,b,name='c')
e = tf.multiply(d,c,name='e')
f = tf.identity(c, name='f')
```




```
res = sess.run(f, feed_dict={b:[2]})
```

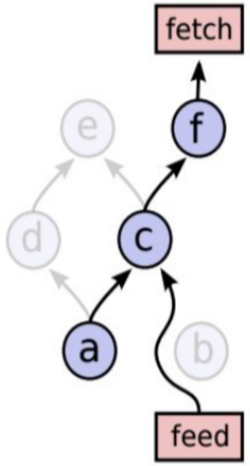
fetch
(the numeric value)

Fetch
f (symbolic)

symbolic

values

Feed and Fetch



- Fetches can be a list of tensors
- Feed (from TF docu)
  - A feed temporarily replaces the **output of an operation** with a tensor value. You supply feed data as an argument to a run() call. The feed is only used for the run call to which it is passed. The most common use case involves designating specific operations to be “feed” operations by using tf.placeholder() to create them.

```
res = sess.run(f, feed_dict={b:data[:,0]})
```

A more general example

```
x = tf.placeholder(tf.float32, shape=(1024, 1024))
res1, res2 = sess.run([loss, loss2], feed_dict={x:data[:,0], y:data[:,1]})
```

fetches
(the numeric values)

fetches
(symbolic)

two inputs (feeds)