Deep Learning

Hands-on

Elisa Ricci


Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
from theano import *
What I actually do

Outline

- Deep Learning Frameworks
- Introduction to TensorFlow
  - Examples (linear regression, MNIST)
- Introduction to Keras
  - Examples (MNIST MLP & CNN)

Deep Learning Frameworks

Deep Learning Frameworks

- Many different frameworks over the past few years…

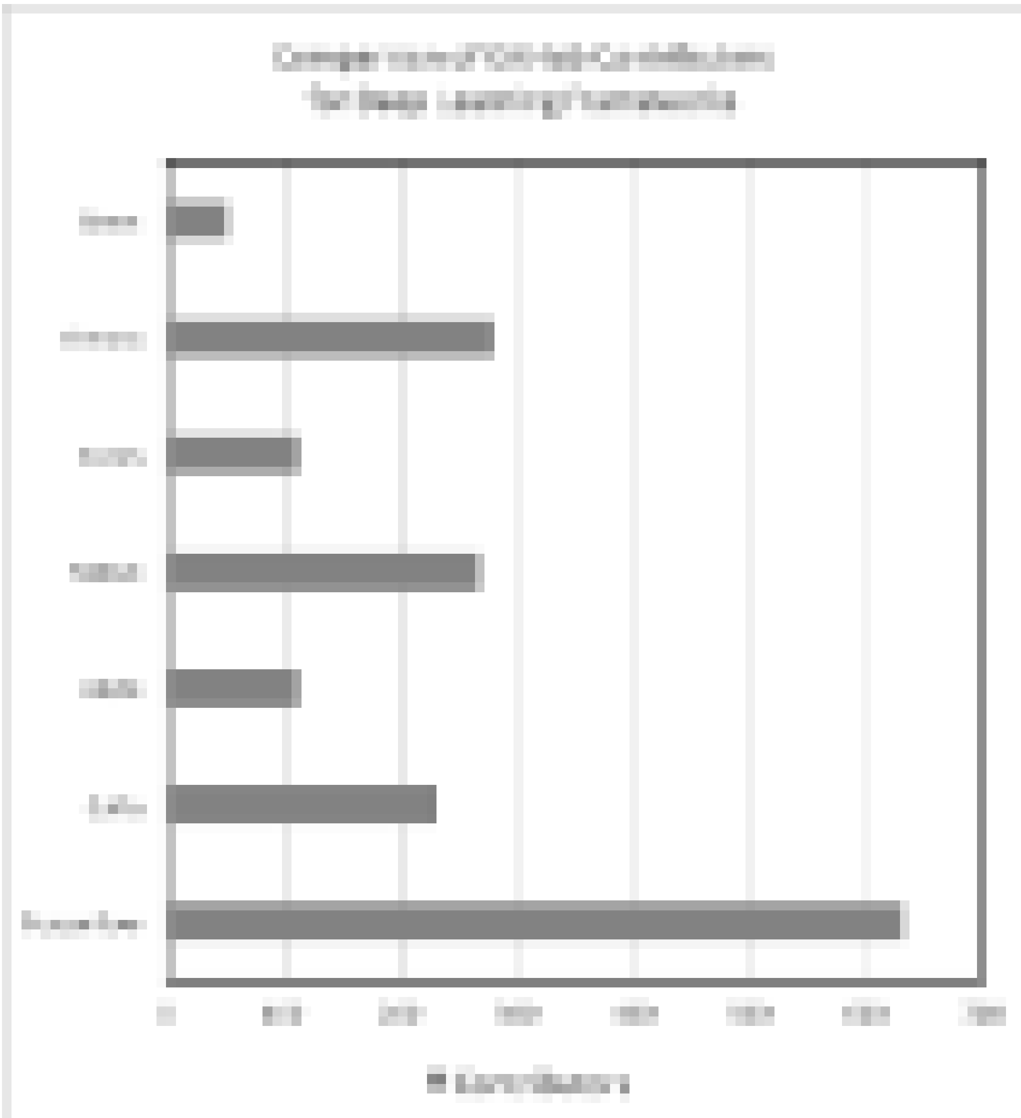
Deep Learning Frameworks

TensorFlow	Google Brain, 2015 (rewritten DistBelief)
Theano	University of Montréal, 2008
Keras	François Chollet, 2015 (now at Google)
Torch	Facebook AI Research, Twitter, Google DeepMind
Caffe	Berkeley Vision and Learning Center (BVLC), 2013

Deep Learning Frameworks

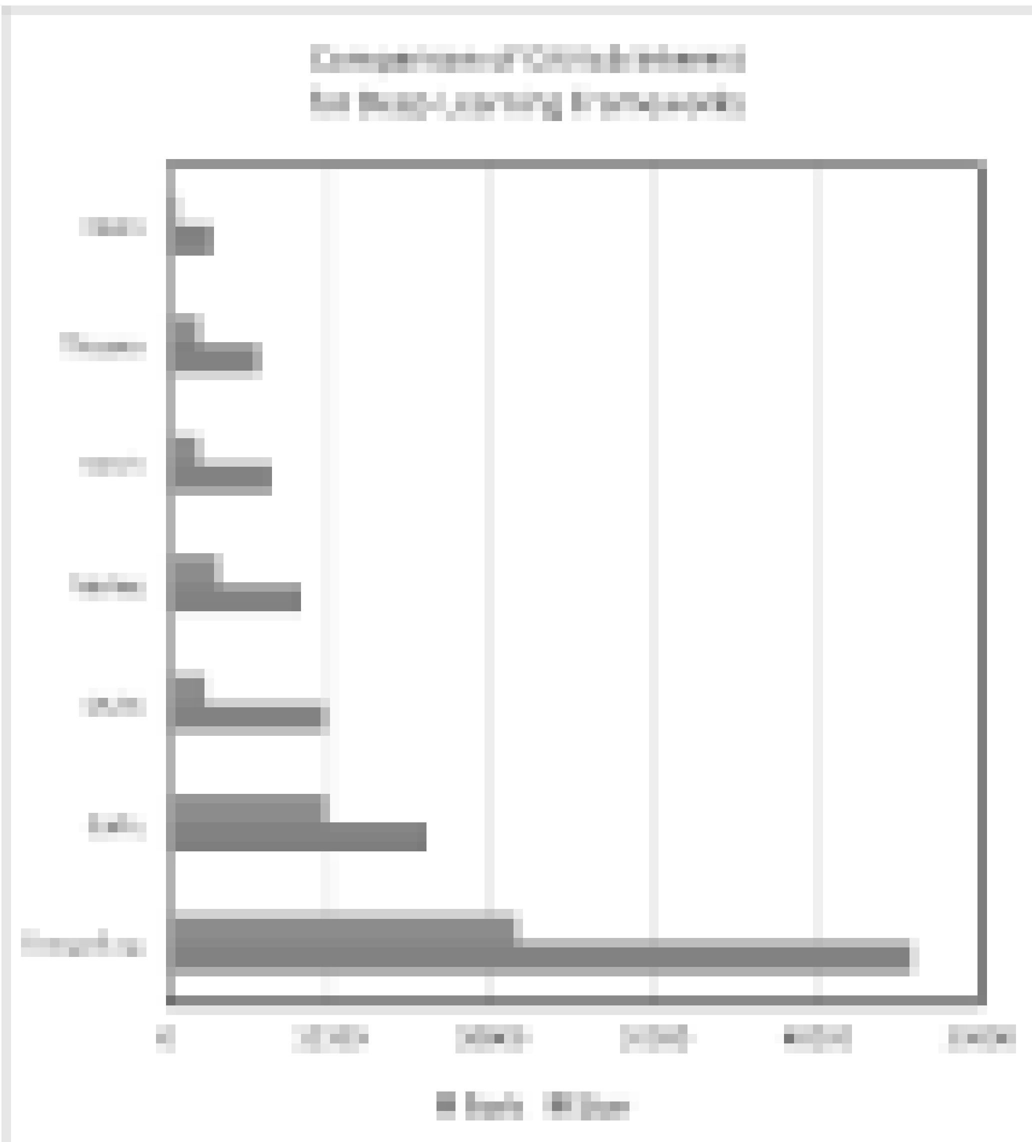
- Which framework to choose? Look at GitHub…

Deep Learning Frameworks



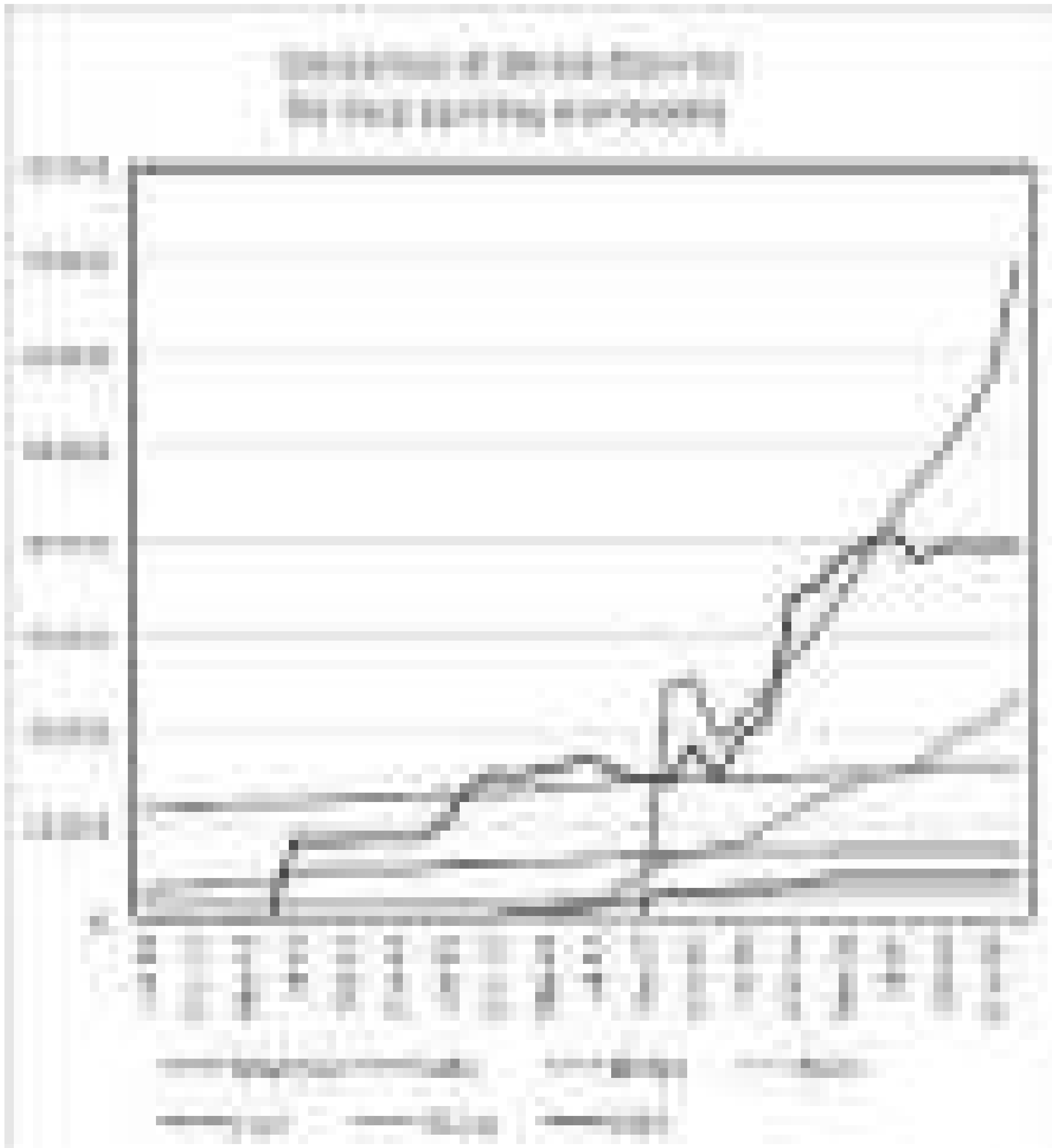
[Rubashkin]

Deep Learning Frameworks



[Rubashkin]

Deep Learning Frameworks



[Rubashkin]

Community and Resources

- (Github, groups, discussions...)
  - For CNNs Caffe has the largest community
  - TensorFlow' s is already large and growing
  - Keras' community is growing
  - Theano' s and Lasagne' s community are declining

Theano

- Maintained by Montréal University group
- Pioneered the use of a computational graph
- General machine learning tool
- Symbolic differentiation
- Use of Lasagne and Keras
- Very popular in the research community, but not much elsewhere. Falling behind

Torch

- Mixed language:
  - C/CUDA backend built on common backend libraries
  - Lua frontend
- Flexibility: existing building blocks from the community can be easily integrated
- Automatic differentiation
- Modularity
- Speed
- *(People hate Lua)* → *very recently PyTorch*

Caffe

- Pros:
  - Especially good for CNN and Computer Vision
  - Extremely easy to code
  - Easy to use pretrained models
  - Matlab and Python interface
  - Easy to include different libraries
  - Layer as building block and many layers already implemented online

Caffe

- Cons:
  - No auto-differentiation
  - Need to write C++/CUDA for new GPU layers
  - Not good for RNN
  - Cumbersome for big networks (ResNet)

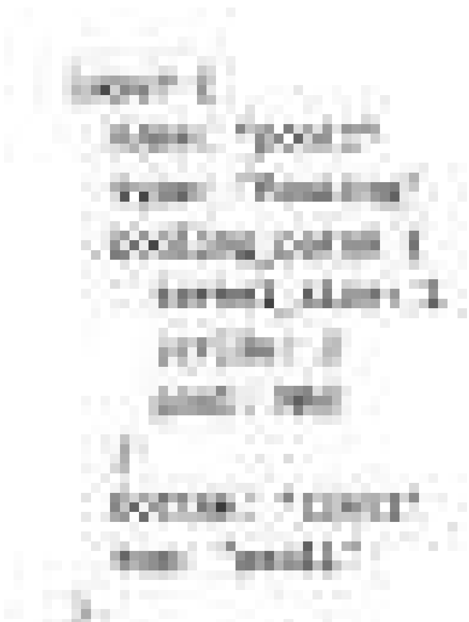
Caffe

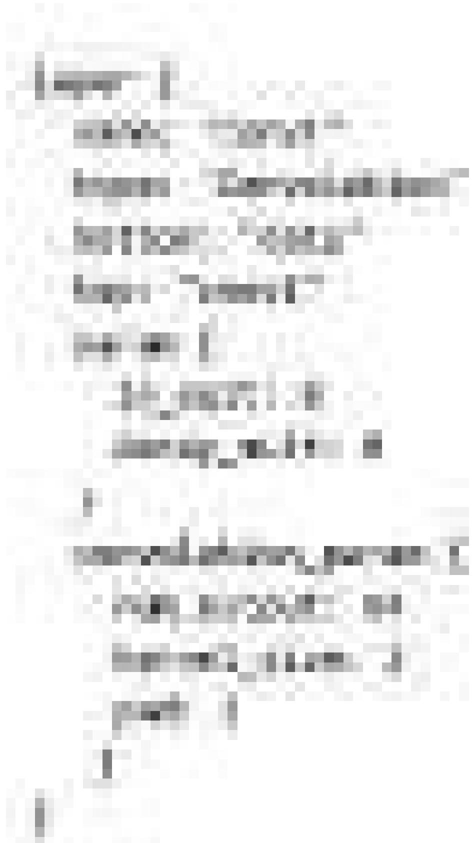
Caffe

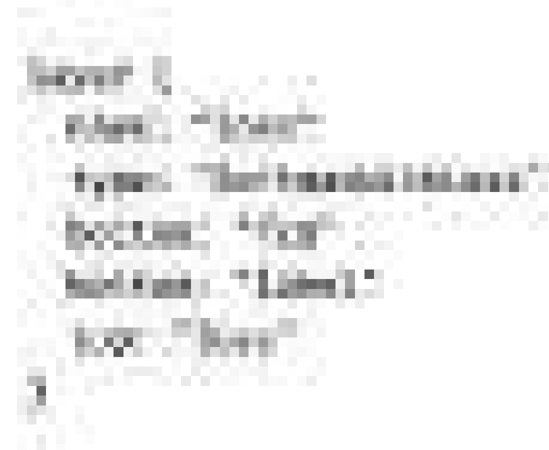
- Main steps:
  - creation of the training network for learning and test network(s) for evaluation
  - iterative optimization by calling forward/backward and parameter updating
  - (periodical) evaluation of the test networks
  - snapshotting of the model and solver state throughout the optimization

Caffe

- Models:







Caffe

Caffe

- Solver:

```
base_lr: 0.01         # begin training at a learning rate of 0.01 = 1e-2

lr_policy: "step"     # learning rate policy: drop the learning rate in "steps"
                      # by a factor of gamma every stepsize iterations

gamma: 0.1            # drop the learning rate by a factor of 10
                      # (i.e., multiply it by a factor of gamma = 0.1)

stepsize: 100000      # drop the learning rate every 100K iterations

max_iter: 350000      # train for 350K iterations total
```

Which framework to chose

	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	++	++	++	+	++	+	+
[illegible]	[illegible]	+++	+++	++	+++	++	++	+
[illegible]	[illegible]	+	+++	++	++	+++	++	
[illegible]	[illegible]	+	++		+	+	+	

[Rubashkin]

Which framework to chose

- You work in industry:
  - TensorFlow, Caffe
- You want to work “seriously” on new models (research-oriented):
  - TensorFlow, Theano, (Torch)
- You don’t have time and you are just curious about deep learning:
  - Keras, Caffe
- You want to use deep learning for educational purposes:
  - Keras, Caffe

TensorFlow

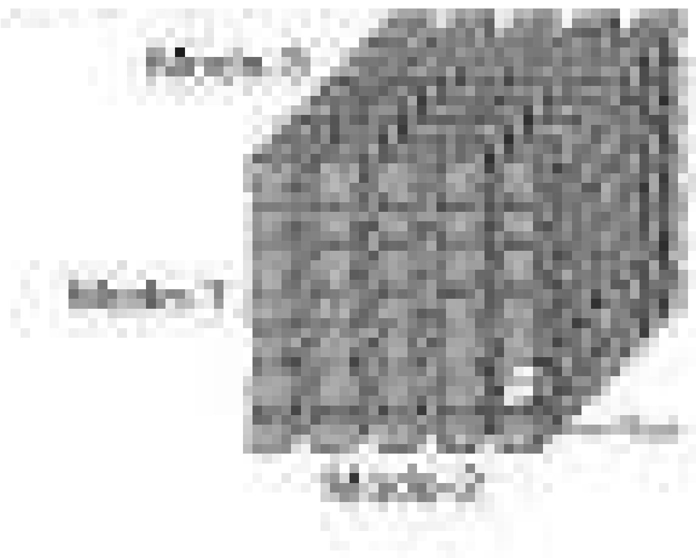
TensorFlow

- An open-source software library for Machine Intelligence
- Especially useful for Deep Learning
- For research & industry

TensorFlow

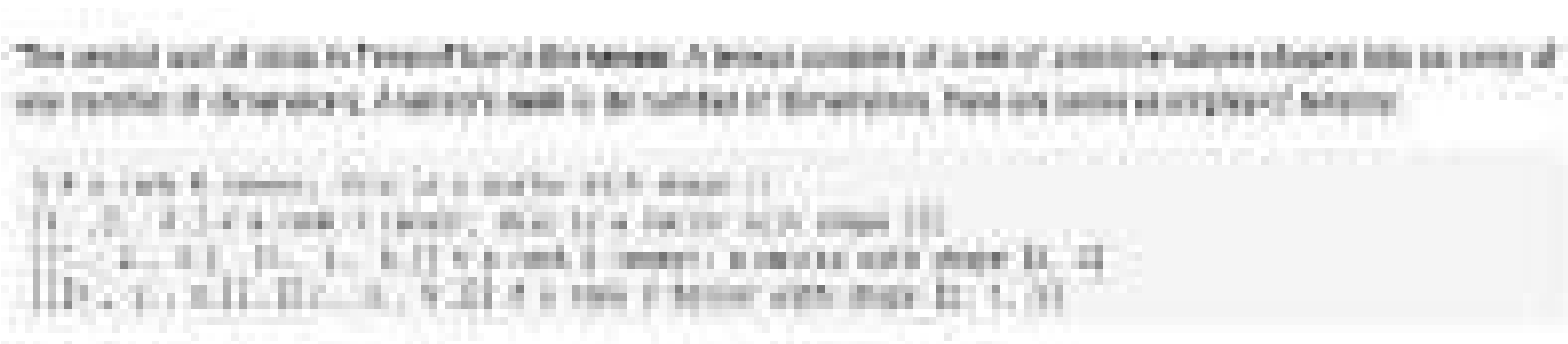
TensorFlow

Tensors: multidimensional arrays



TensorFlow

Tensors: multidimensional arrays



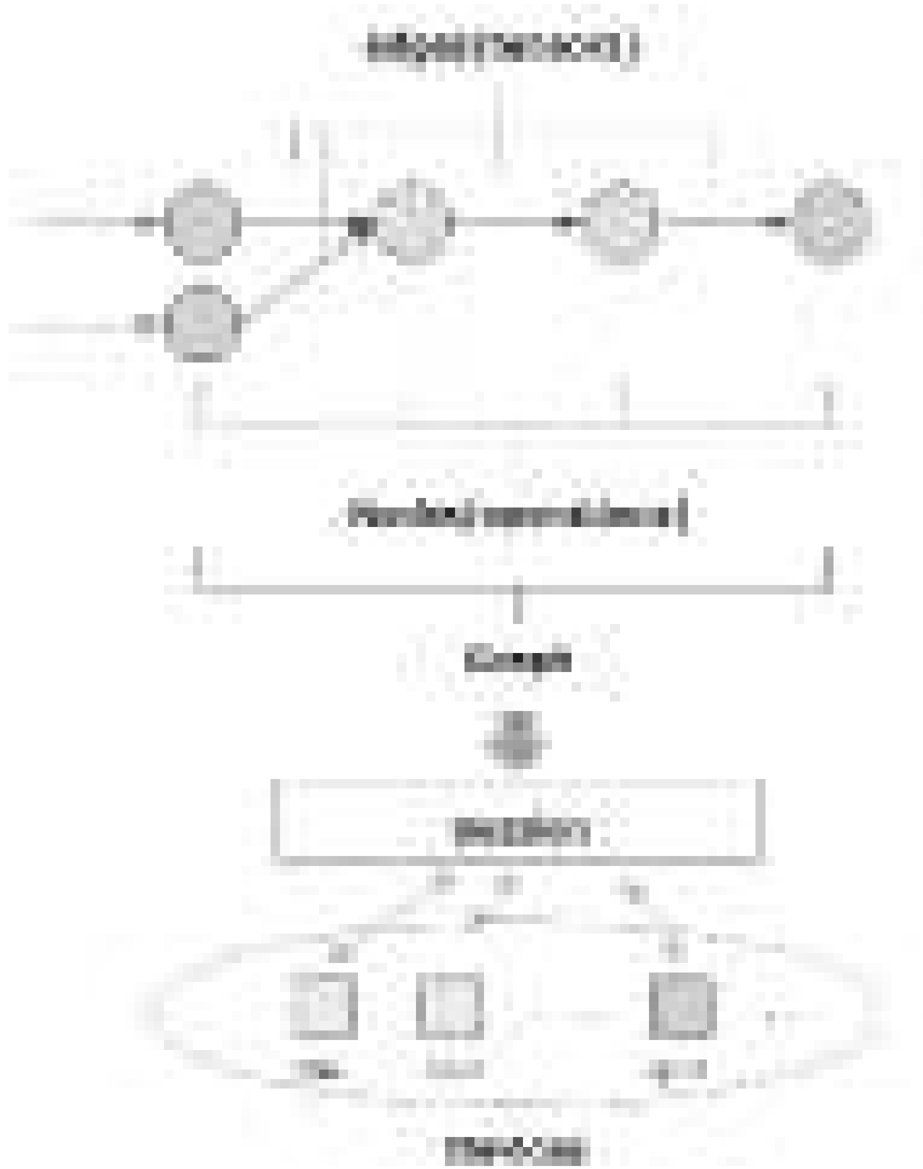
TensorFlow

Flow: Graph describing operations

DataFlow Graph

- Computation is defined as a directed acyclic graph (DAG) to optimize an objective function
- Graph is defined in high-level language (Python, C++)
- Graph is compiled and optimized
- Graph is executed (in parts or fully) on available low level devices (CPU, GPU, Android)
- Data (tensors) flow through the graph

TensorFlow Idea



Automatic differentiation

- TensorFlow can compute gradients automatically
  - Reverse automatic differentiation
  - In a nutshell:
    - When you define an operator (op), you also define together how its derivatives are computed (of course most of the common ops are already provided).
    - After you write a function by stacking a series of ops, the program can figure out by itself how should the corresponding derivatives be computed (usually by keeping some computation graphs and using the chain rule).
    - The benefit is obvious as it saves us from working out the math, writing the code, verifying the derivatives numerically…

Main Components

- The main components of Tensorflow:
  - **Variables:** Retain values between sessions, use for weights/bias
  - **Nodes:** The operations
  - **Tensors:** Signals that pass from/to nodes
  - **Placeholders:** used to send data between your program and the tensorflow graph
  - **Session:** Place when graph is executed.

What we do

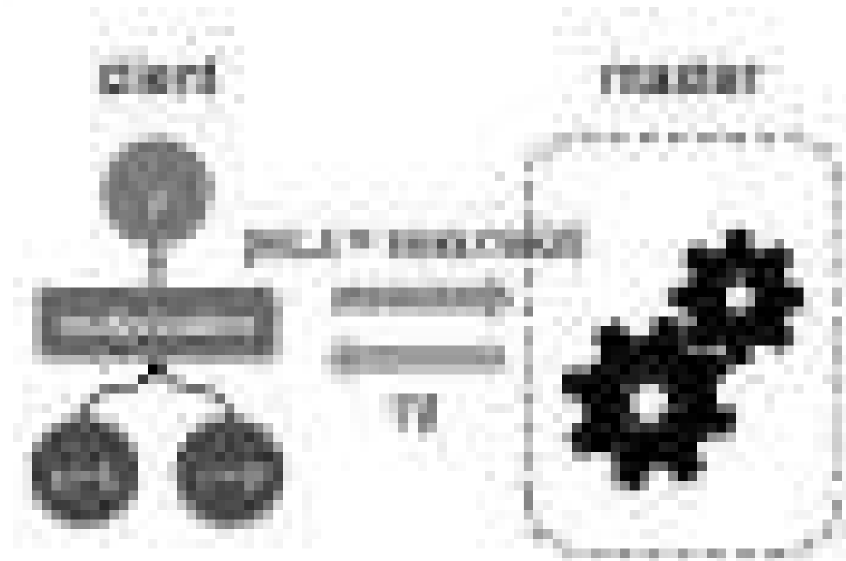
- Create a graph using code C++ or Python and ask TensorFlow to execute this graph.

What we do

- Execution

Hello world

- Multiply two numbers
- Main phases:
  - Import TensorFlow library
  - Build the graph
  - Create a session
  - Run the session



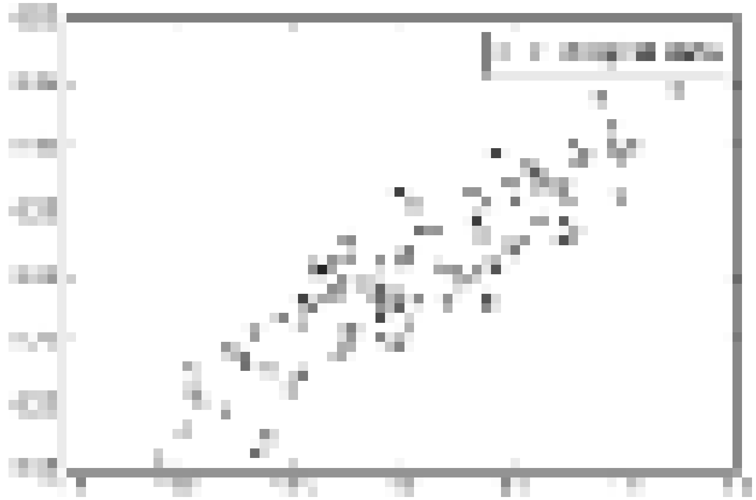
Hello world

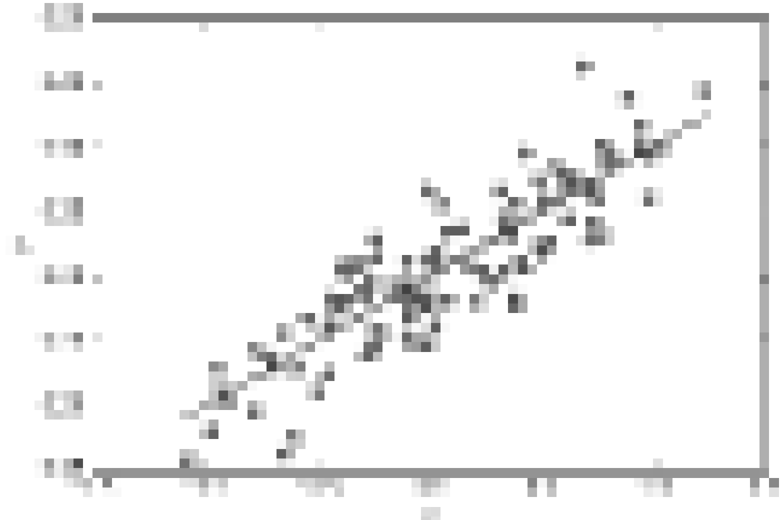
- Multiply two numbers

Placeholders

- Allow exchanging data with your graph variables through "placeholders".
- They can be assigned when we ask the session to run

Linear Regression





Linear Regression

```
import numpy as np
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], tf.float32)
b = tf.Variable([-.3], tf.float32)

# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W * x + b
y = tf.placeholder(tf.float32)

# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
```

Linear Regression

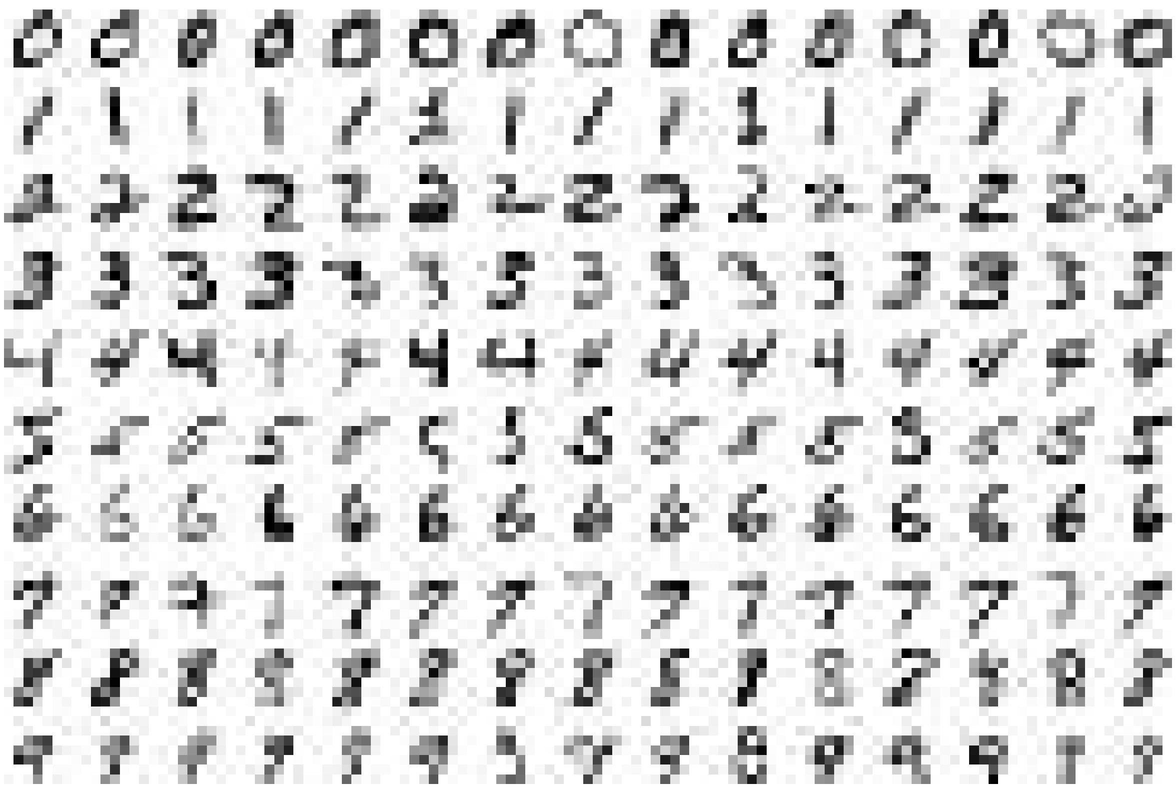
```
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

# training data
x_train = [1,2,3,4]
y_train = [0,-1,-2,-3]

# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
    sess.run(train, {x:x_train, y:y_train})
    # evaluate training accuracy
    curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x:x_train, y:y_train})
    print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```

MNIST

- Classification of hand-written digits (0-9) from 28x28 pixel greyscale images (MNIST data set).
- Full data set of 70k examples: http://yann.lecun.com/exdb/mnist

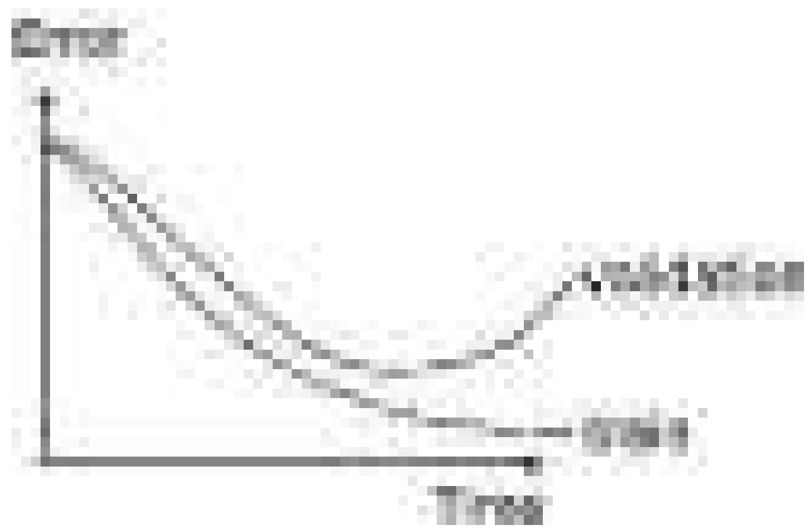


MNIST

- As common in machine learning, the MNIST data is split into three parts:
  - Training: 55,000 images
  - Test: 10,000 images
  - Validation: 5,000 images.
  - Dataset contains pair of images and labels.
  - Useful to test hyper parameters and generalization performance

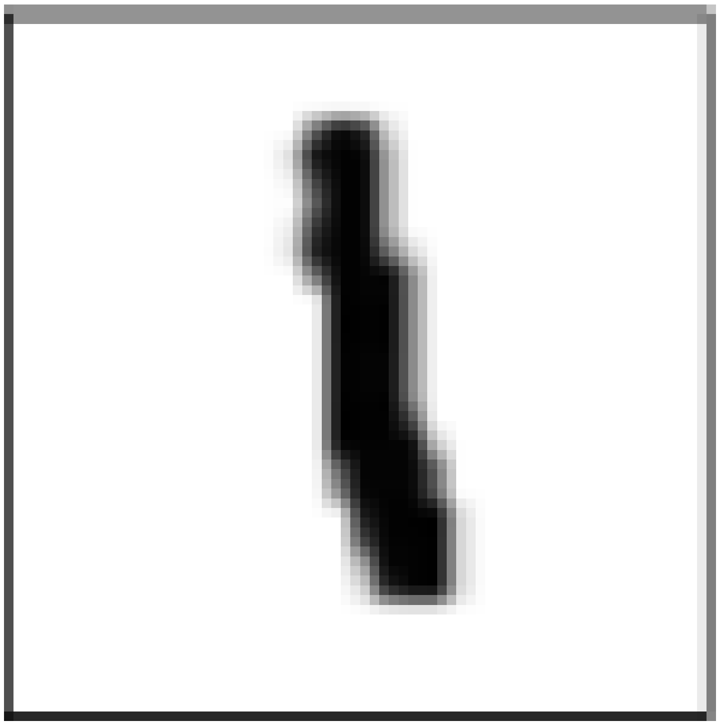
MNIST

- As common in machine learning, the MNIST data is split into three parts:
  - Training: 55,000 images
  - Test: 10,000 images
  - Validation: 5,000 images.
  - Dataset contains pair of images and labels.
  - Useful to test hyper parameters and generalization performance

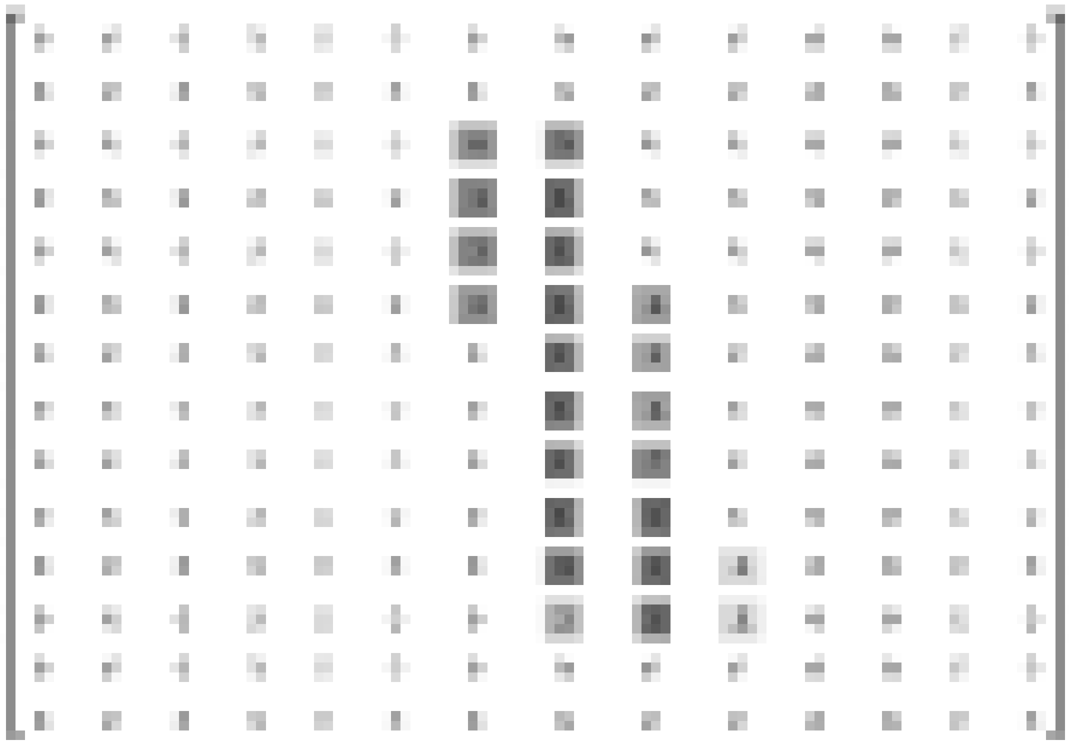


MNIST

- Each image is 28 pixels by 28 pixels.
  - We can flatten this array into a vector of 28x28 = 784 numbers.
  - Vector representation but loosing structure.



≈



Import data

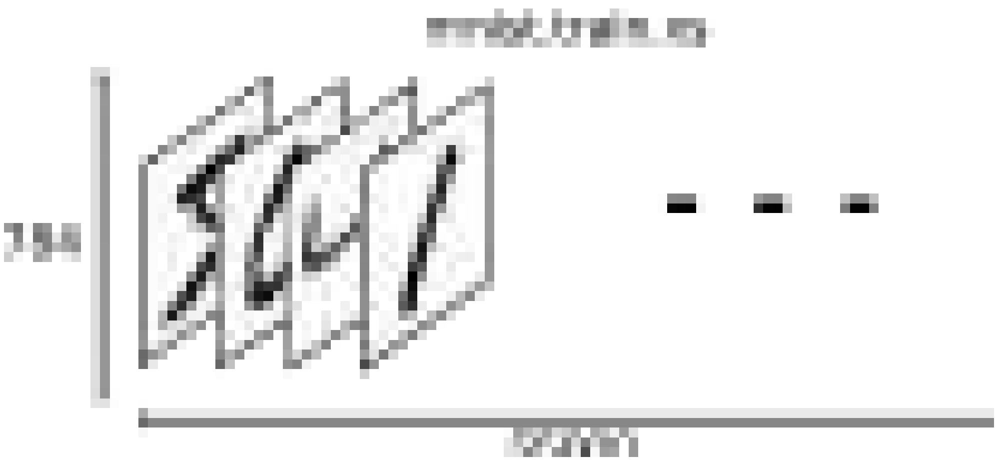
- Download and read the data automatically:

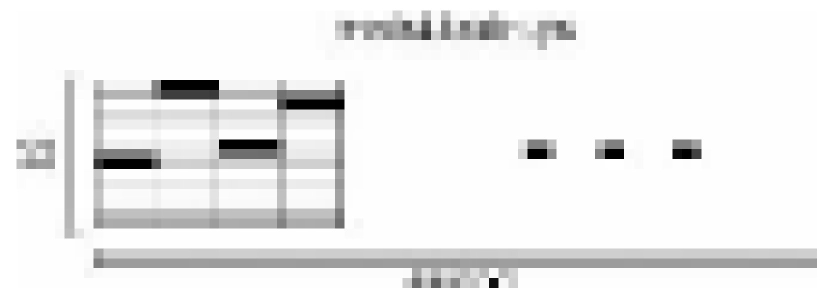
Import data

- We get:

  mnist.train.images: tensor with a shape of [55000, 784]

  *mnist.train.labels:*a [55000, 10] array of floats – vector notation for class labels.



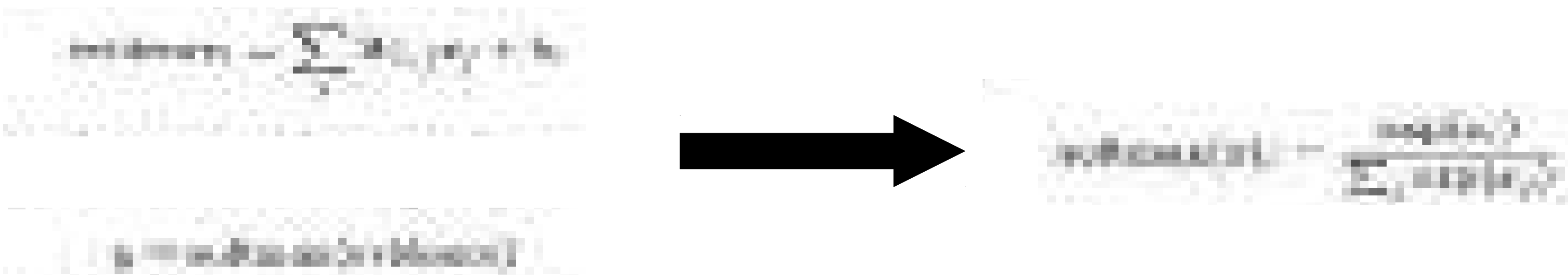


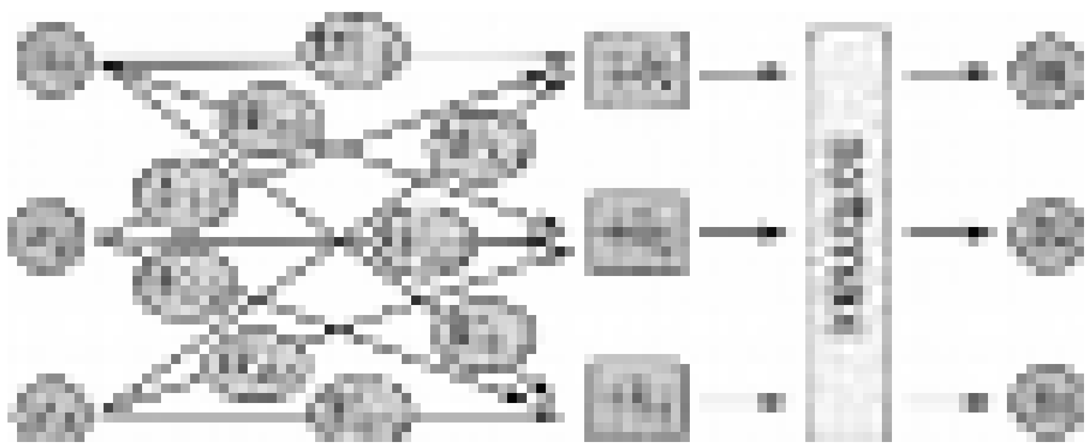
NN training

- Several things to decide (data, hyperparameters):
  - Training data
    - Representation (vectors, images, text).
    - Normalization
  - Architecture
    - Layers: type, shape, number.
    - Activation functions
    - Output type (according to task, e.g. classification/regression) and loss function.
  - Learning algorithm
    - Initialization.
    - Update scheme.
    - Learning rate.
    - Momentum.
    - Regularization (weight decay, dropout).
    - Batch normalization
    - Stopping criteria

Softmax regression

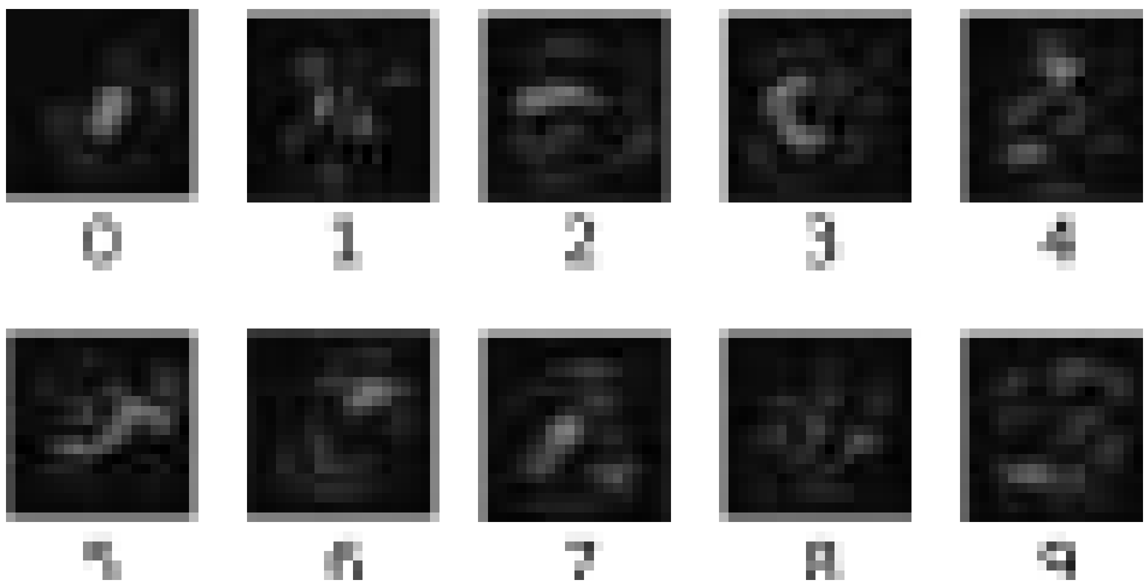
- Recap: softmax regression to output probabilities
- Two steps: add up the evidence of our input being in certain classes and then convert evidences into probabilities.





Softmax regression

- Output: As we do a weighted sum of the pixel intensities we can inspect them.
- Red: negative weights.
- Blue: positive weights.



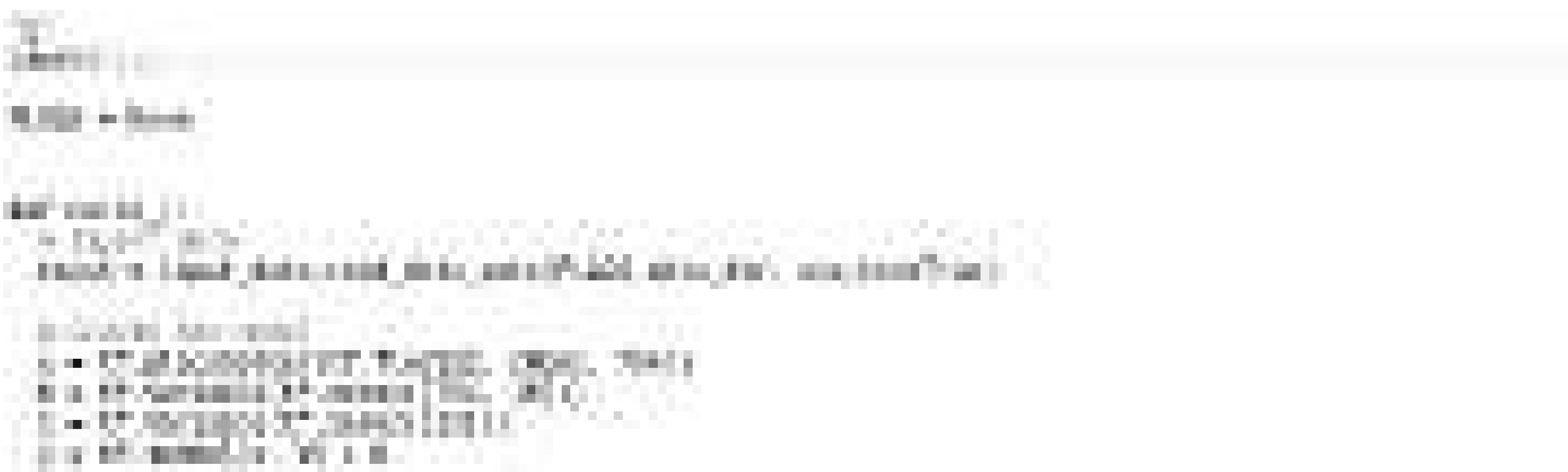
Softmax regression

- Matrix Notation

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right)$$

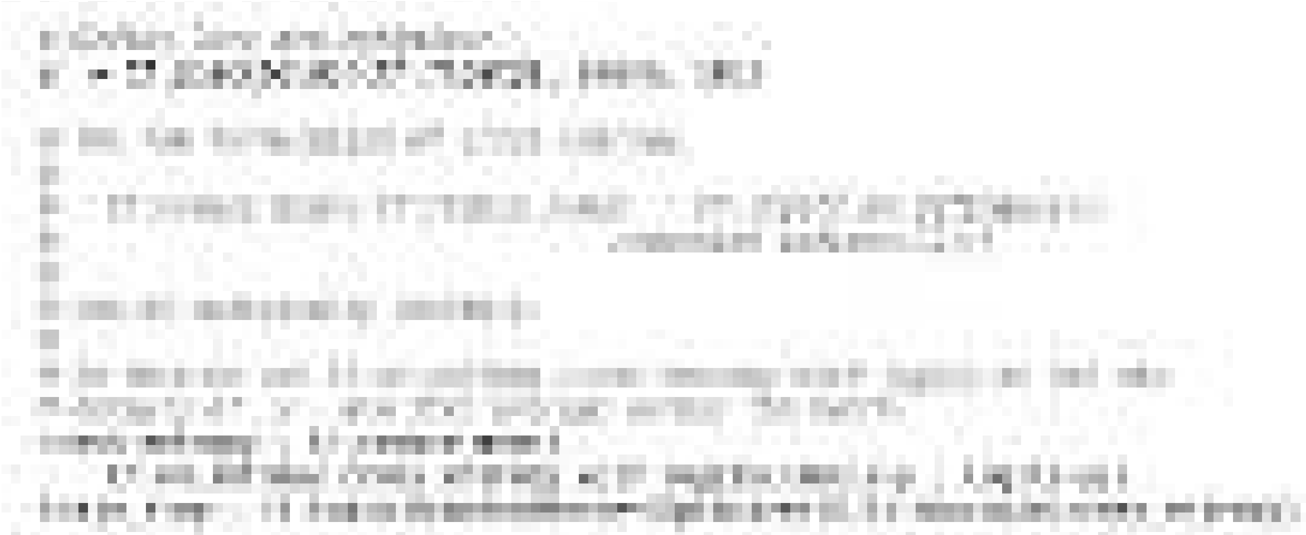
MNIST

- We use variables and placeholders to create the model:
  - Look at the dimensionality
  - What is missing?



MNIST

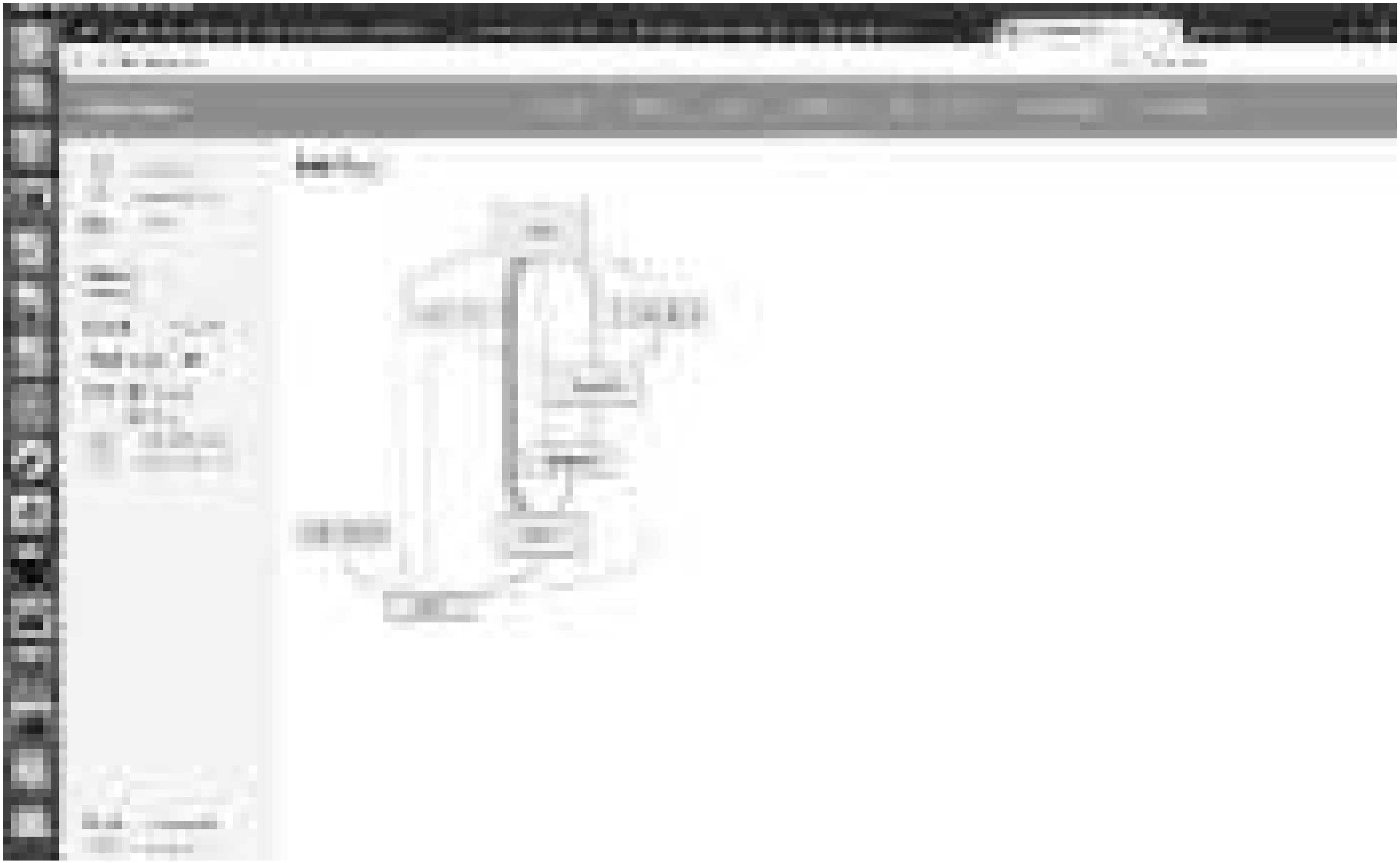
- Model training:
  - Use cross-entropy 
  - Optimize with gradient descent with a learning rate 0.5.
  - Many other optimizers (link)



MNIST

- Run the session
  - Training considering mini-batches
  - Evaluate performance (are they good?)

TensorBoard



TensorBoard

- Training a massive deep neural network can be complex and confusing.
- TensorBoard: visualization tools to facilitate models understanding and debug.
- Visualize graph, plot quantitative metrics about the execution of the graph, show additional data like images used, visualize statistics.

TensorBoard

- Modify code to generate summary data.

  (1) Create graph and decide which nodes you would like to collect summary data.

  Example MNIST:

  - Monitor learning rate and loss.
  - Use `tf.summary.scalar` for to the nodes that output the learning rate and loss respectively.

TensorBoard

- Modify code to generate summary data.

  (1) Create graph and decide which nodes you would like to collect summary data.

  Example MNIST:

  - Visualize the distributions of activations coming off a particular layer, or the distribution of gradients or weights.
  - Use `tf.summary.histogram`.

TensorBoard

- Modify code to generate summary data.

  (1) Create graph and decide which nodes you would like to collect summary data.

  The summary nodes are peripheral nodes added to the graph: none of the ops we are currently running depend on them.

TensorBoard

- Modify code to generate summary data.

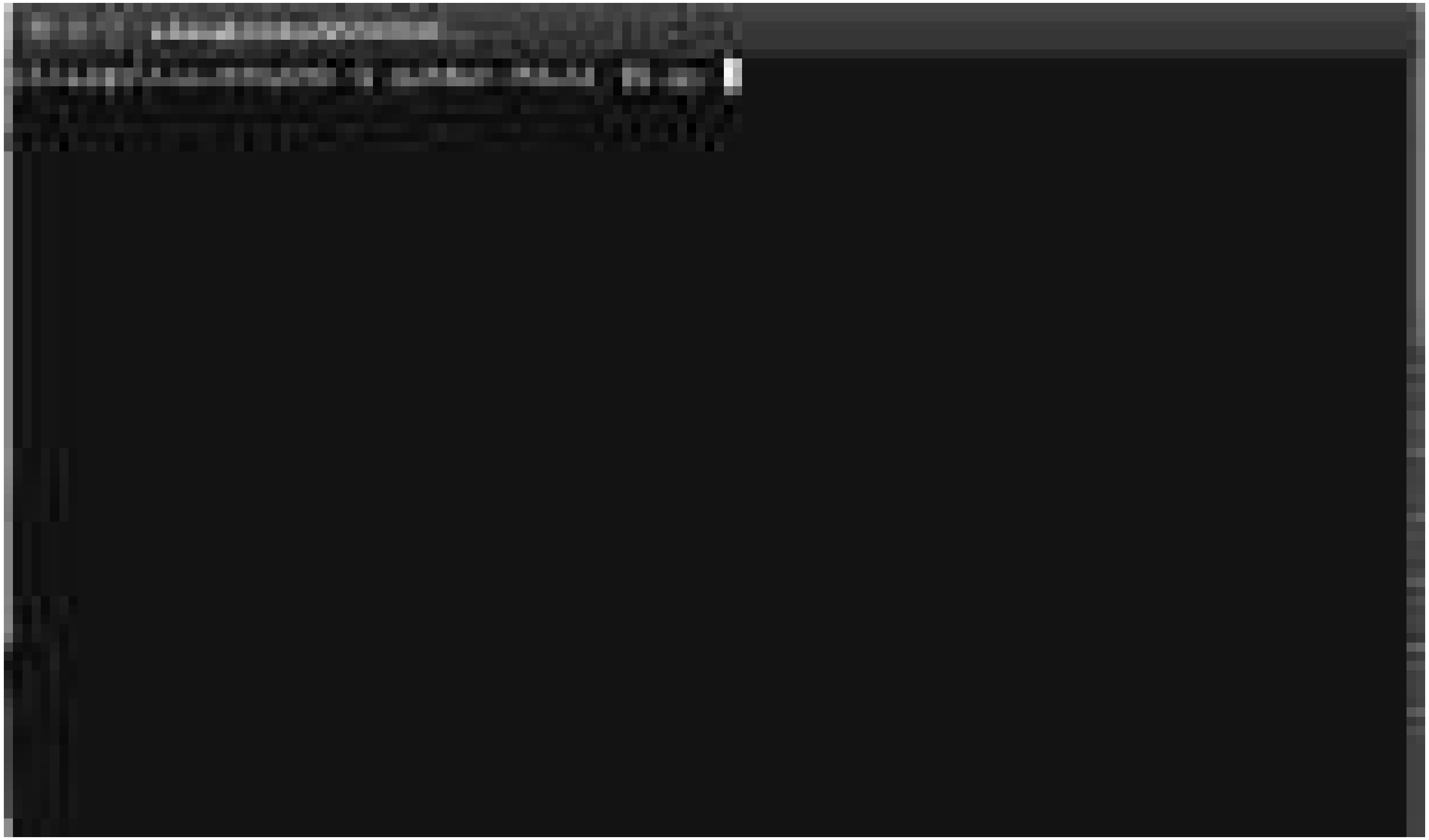
  (2) To generate summaries, run all of the summary nodes.

  (2a) Use *`tf.summary.merge_all`* to combine them.

  (2b) Run the merged summary op, which will generate a serialized Summary protobuf object with all of your summary data at a given step.

  (5) Write summary data to disk, pass the summary protobuf to a *`tf.summary.FileWriter`*.

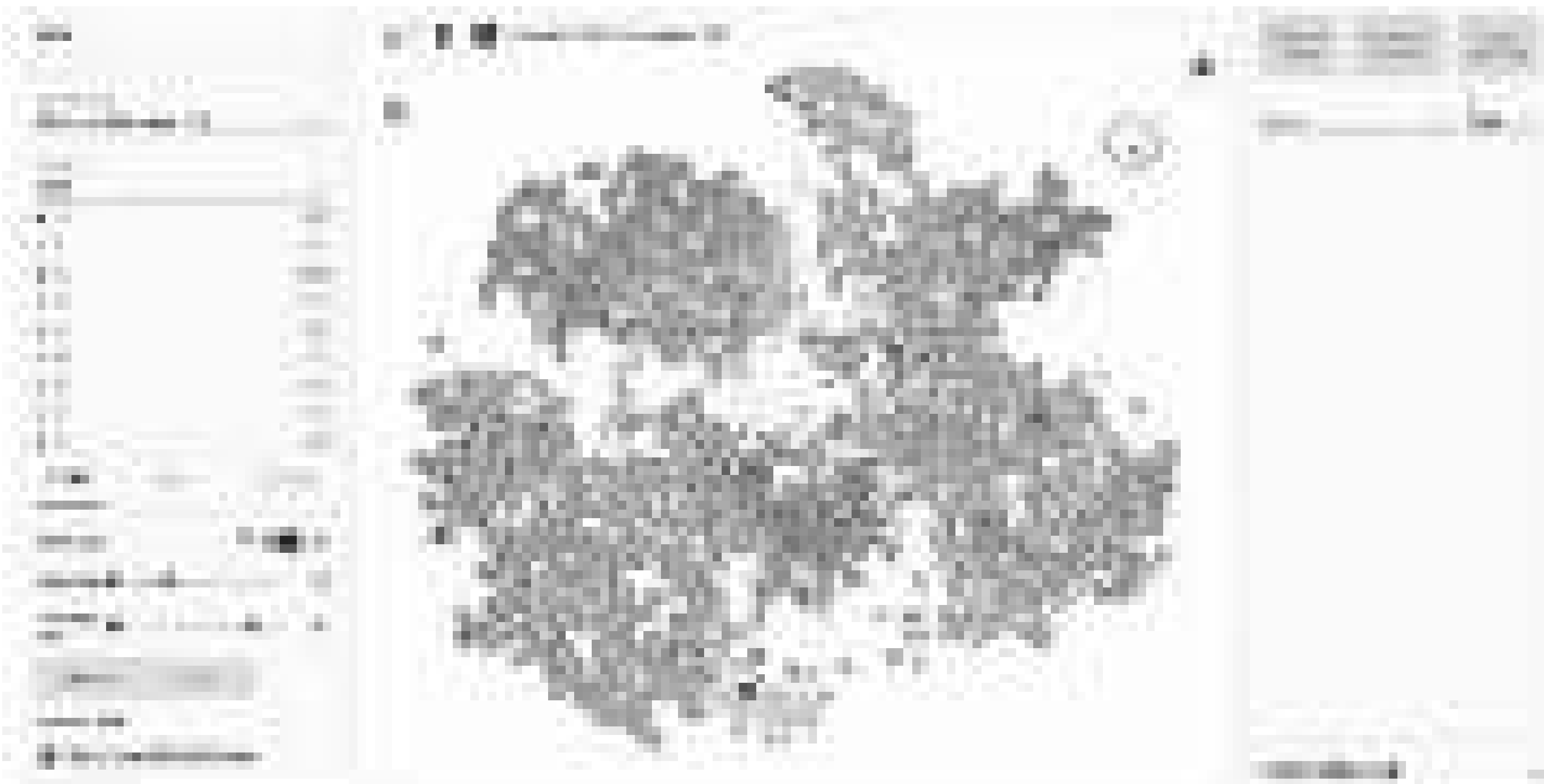
TensorBoard



TensorBoard

TensorBoard

- Other features: Embedding visualization.



Keras

Keras

- Keras (κέρας) means *horn* in Greek.
- In the Odyssey it is mentioned that dream spirits are divided between:
  - those who deceive men with false visions, who arrive to Earth through a gate of ivory
  - those who announce a future that will come to pass, who arrive through a gate of horn.

Keras

- Easy-to-use Python library
- Why Python? Easy to learn, powerful libraries (scikit-learn, matplotlib...)
- It wraps Theano and TensorFlow (it benefits from the advantages of both)
- Guiding principles: modularity, minimalism, extensibility.

Keras

- Use both GPU and CPUs
- Easy to use both convolutional networks and recurrent networks and combinations of the two.
- Supports arbitrary connectivity schemes (including multi-input and multi-output training)
- Many easy-to-use tools: real-time data augmentation, callbacks (Tensorboard visualization)

Keras

- Keras gained official Google support

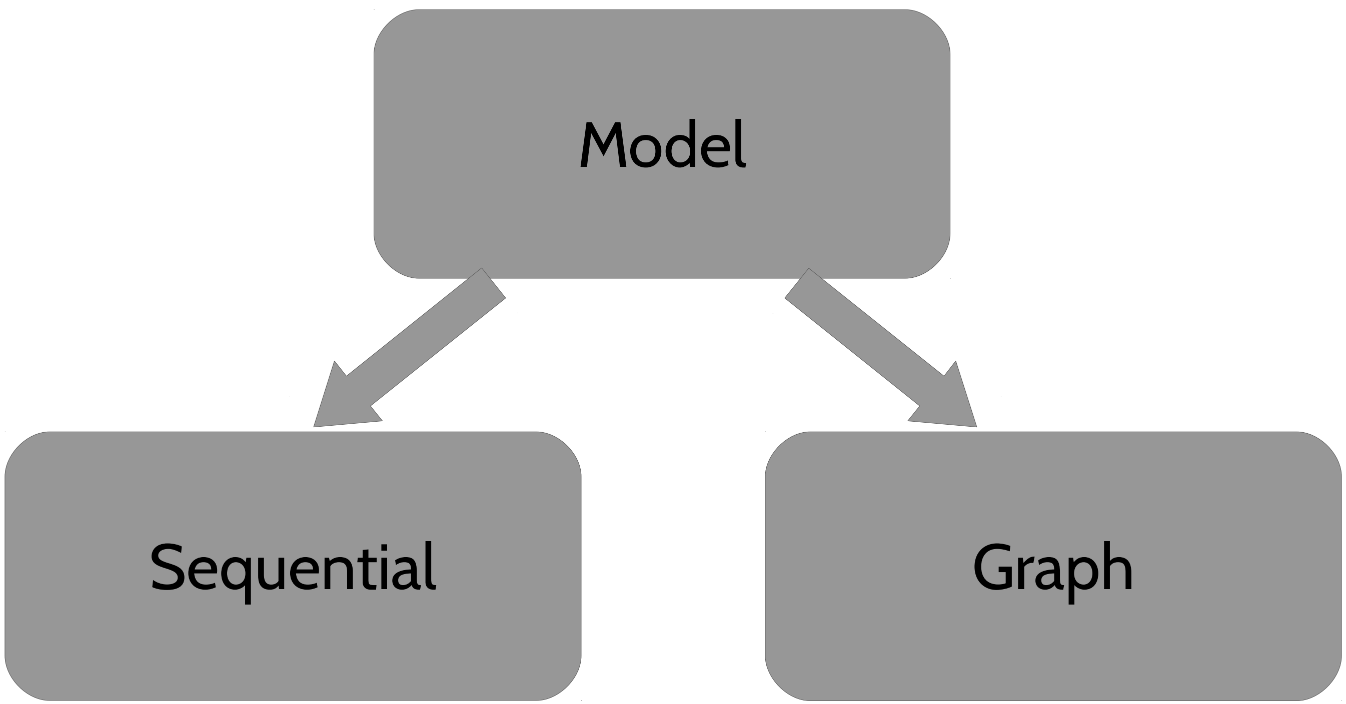
Big deep learning news: Google Tensorflow chooses Keras

Keras

- Weaknesses:
  - Less flexible
  - Some stuff not there yet (no RBM for example)
  - Less projects available online (e.g. with respect to Caffe)

Model

- A model is a ***sequence*** or a ***graph*** of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible.



Modularity

- A model is a ***sequence*** or a ***graph*** of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible.
- Modules:
  - neural layers
  - cost functions
  - optimizers
  - initialization schemes
  - activation functions
  - regularization schemes
  - ***your own module***

Keras

- Extensibility: modules are easy to add.
- Simplicity: modules should be made extremely simple.

TensorFlow:

Keras:

Install Keras

- Extremely easy:

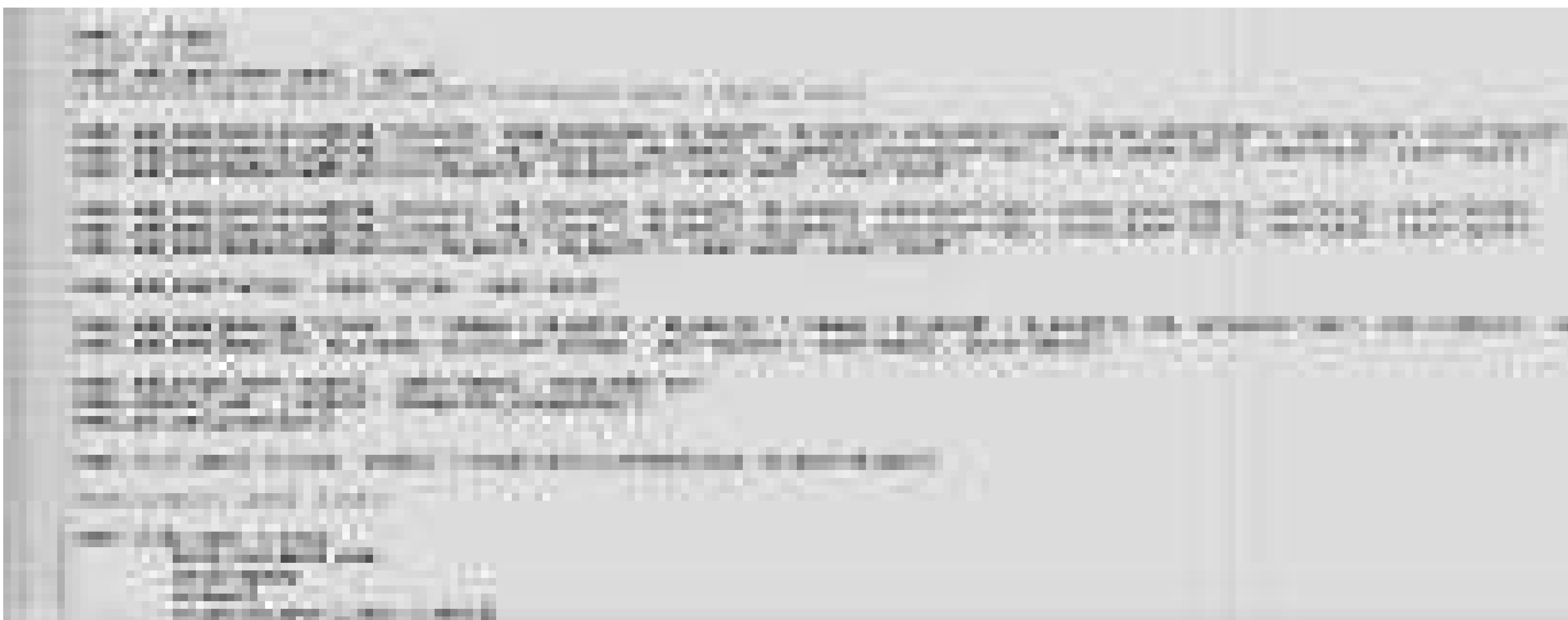
```
>> source tensorflow/bin/activate
>> python
>> pip install keras
>> import keras as k
```

Sequential model

- Sequential models are linear stack of layers
- Treat each layer as object that feeds the next layer

Graph model

- Useful to create two or more independent networks to diverge or merge
- Useful to create multiple separate inputs or outputs
- Different merging layers (sum or concatenate)



Let's run MNIST again

- Homepage

  https://keras.io/

  https://keras.io/getting-started/sequential-model-guide/#getting-started-with-the-keras-sequential-model

- Examples:

  https://github.com/fchollet/keras/tree/master/examples

- Let's compare a MLP and a CNN…

Questions?