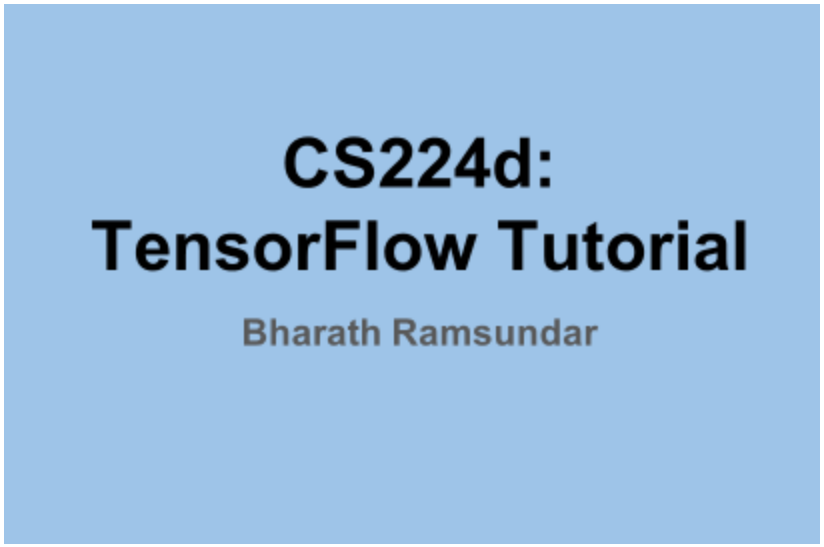
Deep Learning & TensorFlow with Keras

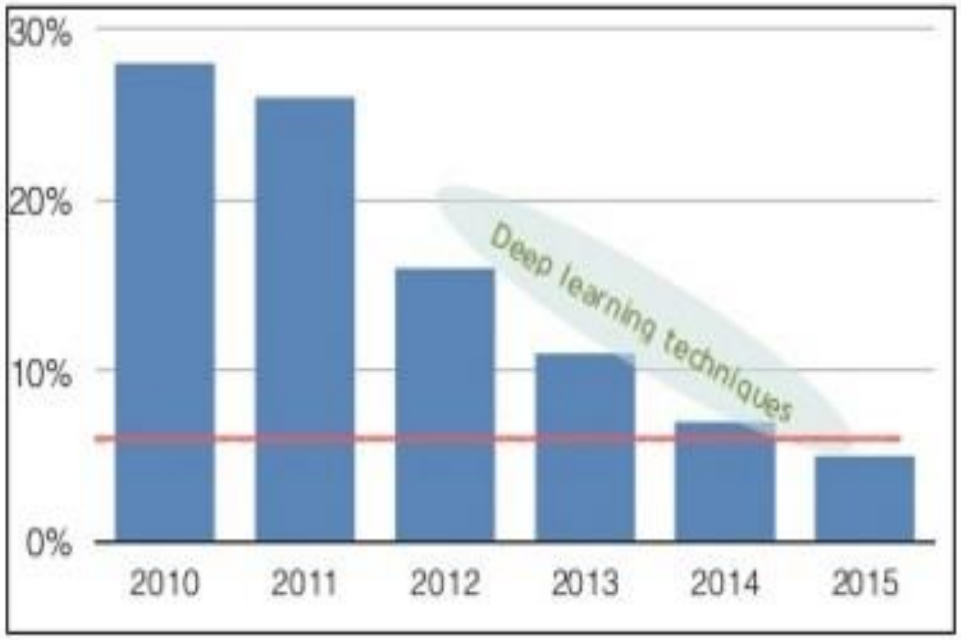
These slides are based on:

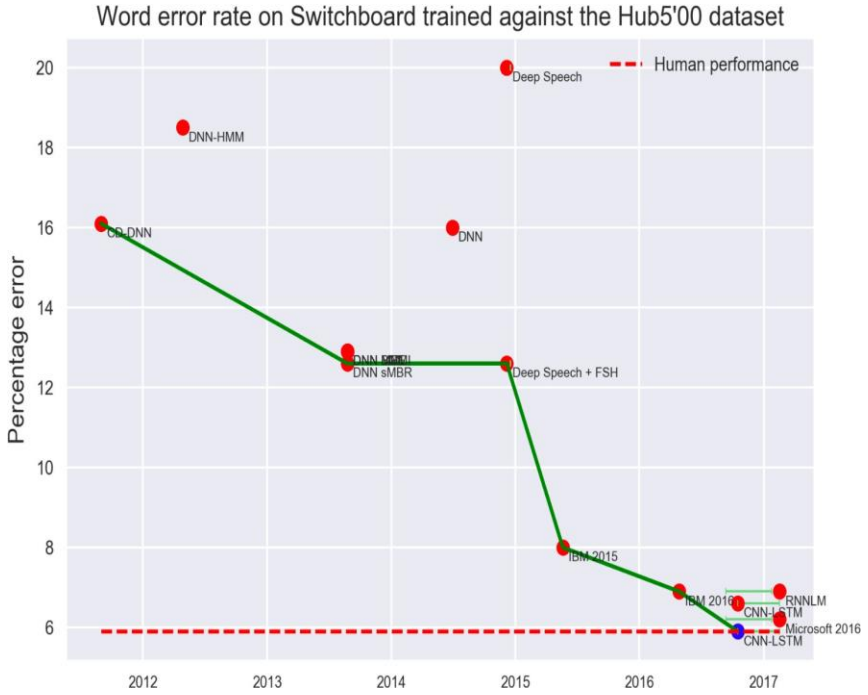
https://www.slideshare.net/kerveros99/deep-learning-for-recommender-systems-recsys2017-tutorial



https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Why Deep Learning?





ImageNet challenge [error rates] (red line = human performance)

Why Deep Learning?

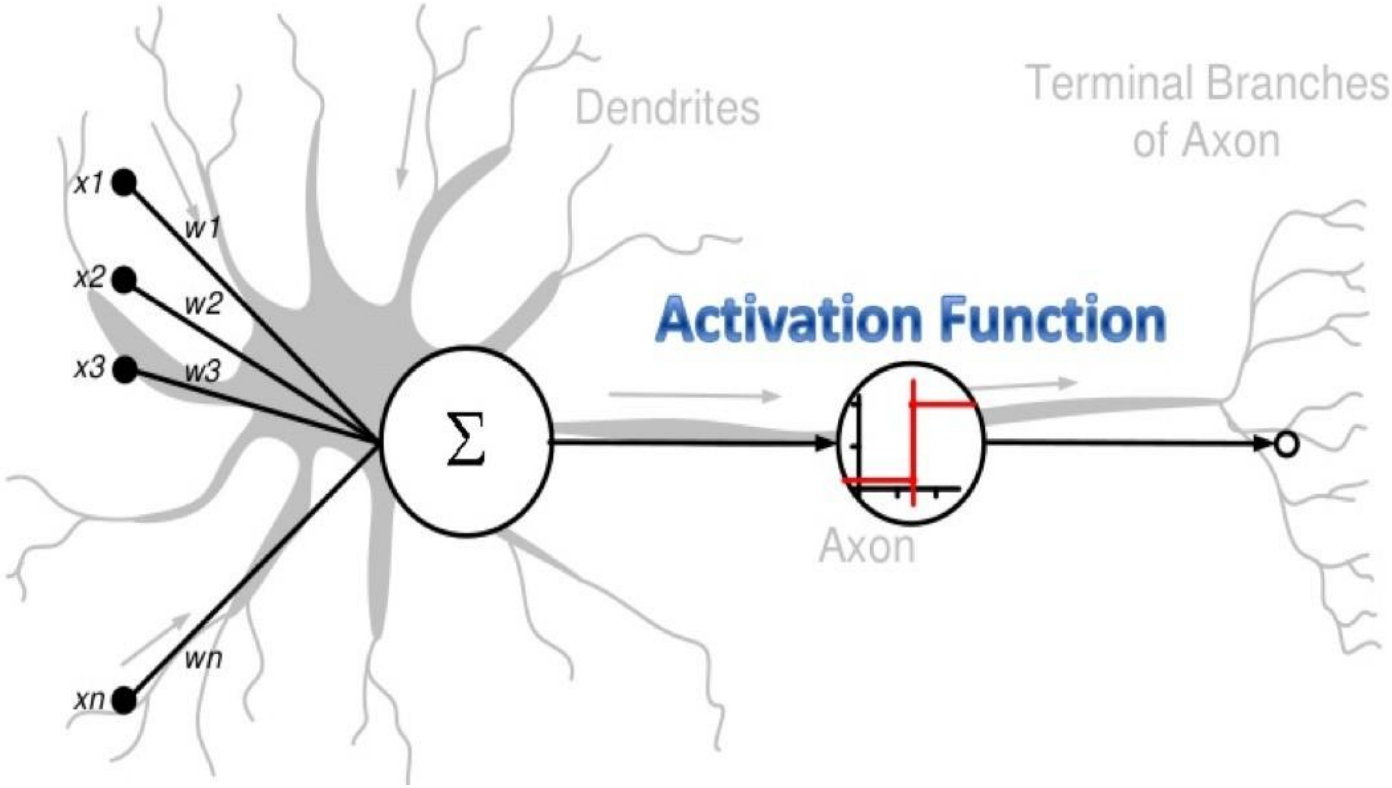
hand-crafted
Feature Extractor

"Simple" Trainable
Classifier

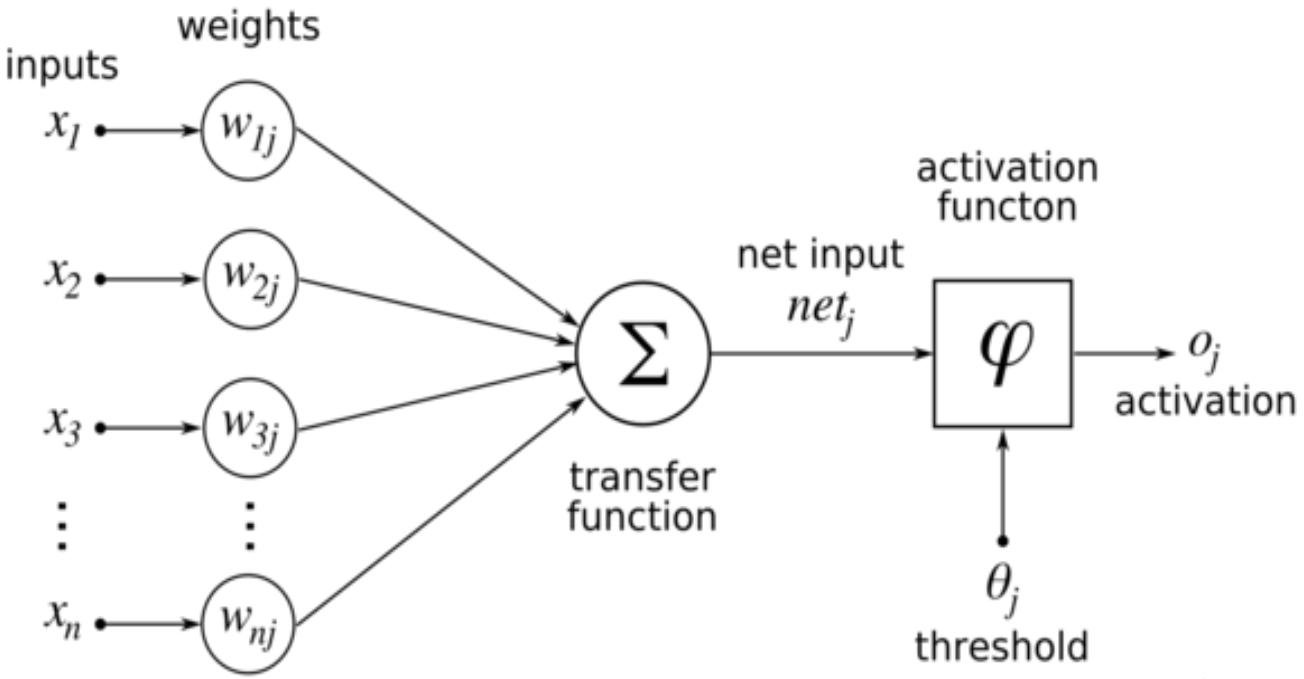
Trainable
Feature Extractor

Trainable
Classifier

Neural Model

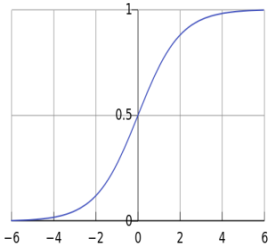


Neuron a.k.a. Unit

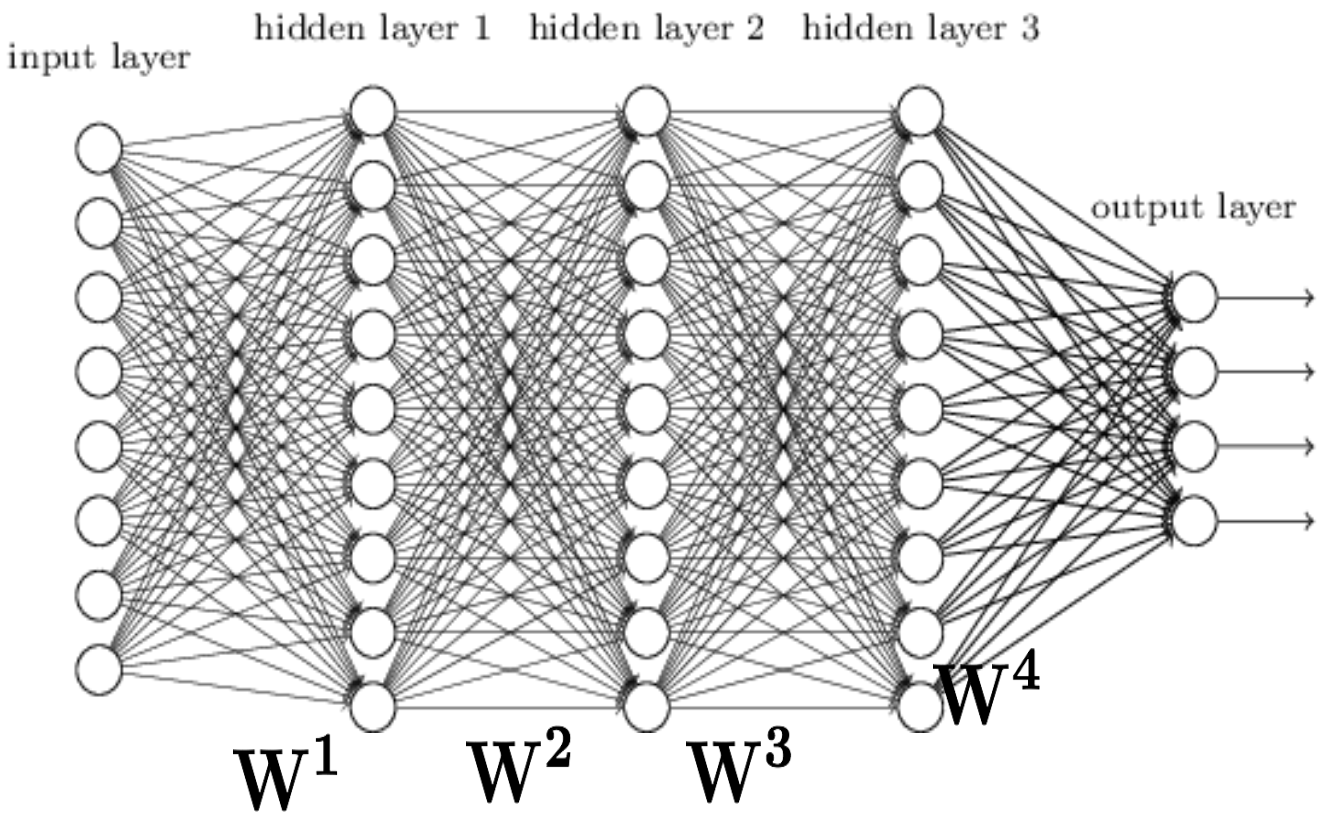


$$P(o_j = 1|x) = \phi\left(\sum_{i=1}^{n} w_{ij}x_i + \theta_j\right)$$

$$\phi = \frac{1}{1 + e^{-\Sigma}}$$



Learning


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer
W^1
W^2
W^3
W^4

Stochastic Gradient Descent

- Generalization of (Stochastic) Gradient Descent

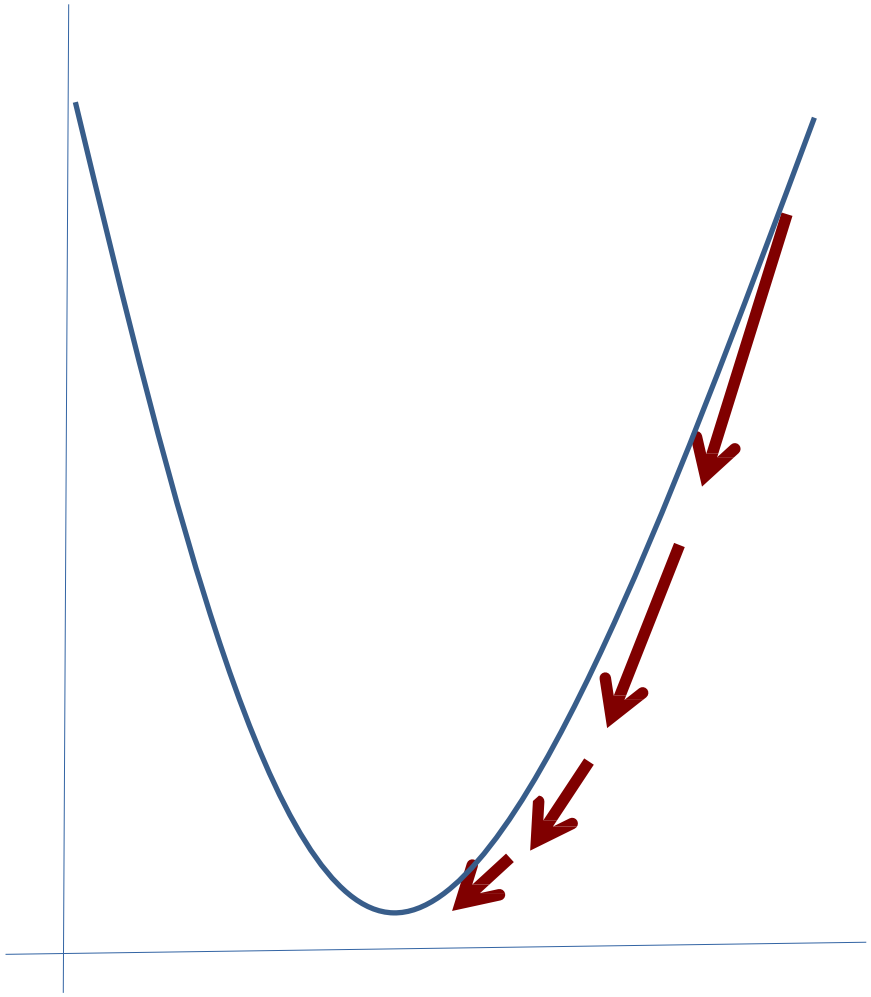
$$E = \frac{1}{2}(f - y)^2$$

$$f = \mathbf{w}^T \mathbf{x}$$

$$for \quad i = 1, 2, \ldots, n$$

$$\mathbf{w} := \mathbf{w} - \eta \nabla_f E \mathbf{x}_i$$

Stochastic Gradient Descent



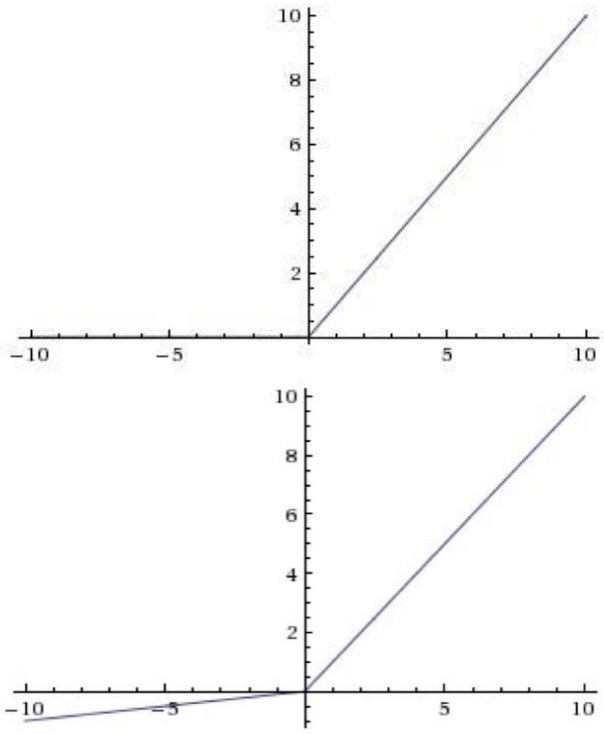
Modern Deep Networks

- Ingredients:

- Rectified Linear Activation function a.k.a. ReLu

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

$$\sigma(x) = max(\alpha x, x) \quad \alpha < 1$$



Modern Deep Networks

- Mini-batches:
  - Stochastic Gradient Descent (or its evolutions)
  - Compute gradient over many (50 -100) data points (minibatch) and update.
- Shared parameters
- Several layer / network-level templates
  - CNN, RNN, 2vec models, Autoencoders, Generative Adversal Networks (GAN),...
- Transfer learning (from one problem to another)

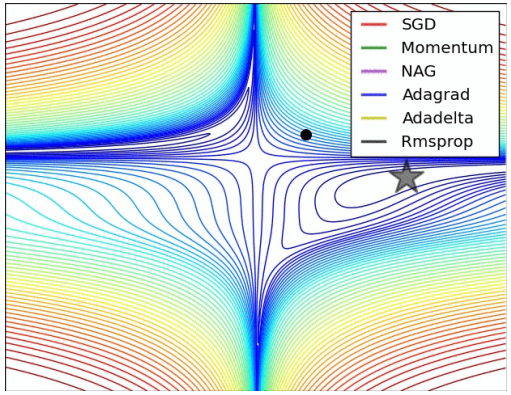
Modern Feedforward Networks

- Ingredients:
- Adagrad a.k.a. adaptive learning rates

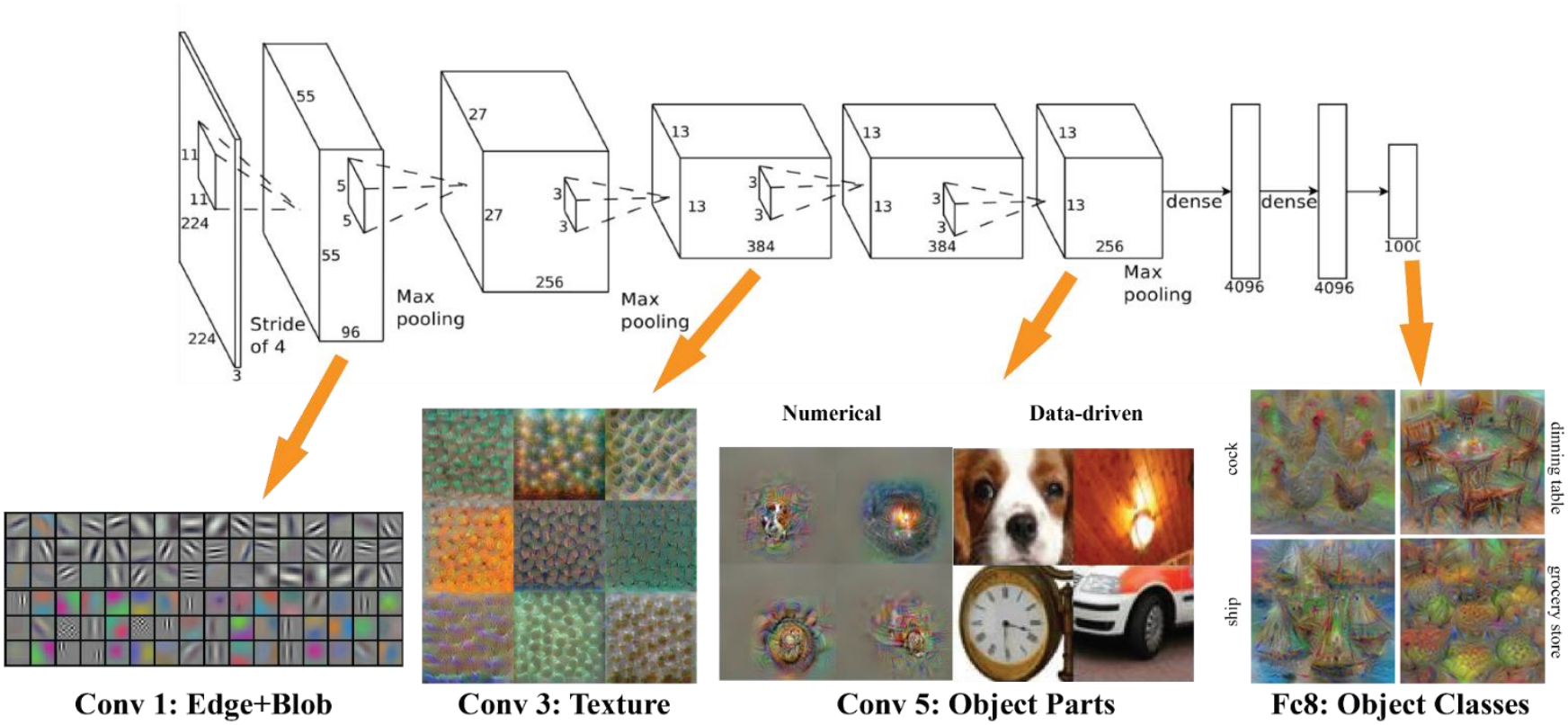
$$\mathbf{w} := \mathbf{w} - \frac{\eta}{\sqrt{\sum_{j=0}^{j=i} \nabla_w E_j^2}} \nabla_f E \mathbf{x}_i$$

Decrease step size over time

In a factor-wise fashion



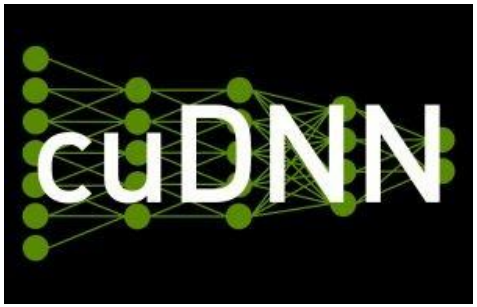
Convolutional deep network

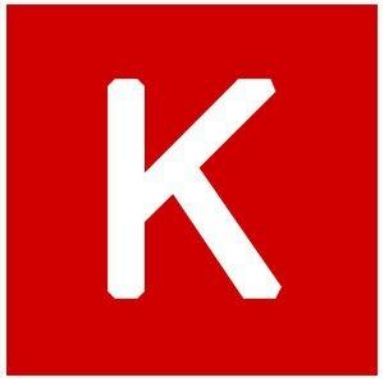


[Fig source: http://vision03.csail.mit.edu/cnn_art/index.html]

Deep-Learning Package Zoo

- Torch
- *Caffe*
- Theano (Keras, Lasagne)
- CuDNN
- Tensorflow
- *Mxnet*
- Etc.

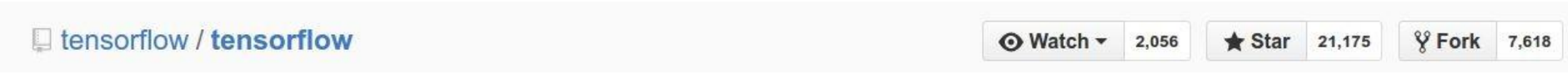




Deep-Learning Package Design Choices

- Model specification: **Configuration file** (e.g. Caffe, DistBelief, CNTK) versus **programmatic generation** (e.g. Torch, Theano, Tensorflow)
- For programmatic models, choice of high-level language: Lua (Torch) vs. Python (Theano, Tensorflow) vs others.
- We chose to work with **python** because of rich community and library infrastructure.

- *However, the most important is DEAD or ALIVE*
  - *TensorFlow is pretty much alive☺*



What is TensorFlow?

- TensorFlow is a deep learning library recently(2017?) open-sourced by Google.
- But what does it actually do?
  - TensorFlow provides primitives for defining functions on tensors and automatically computing their derivatives.
  - Much more nowadays
  - https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/

But what's a Tensor?

- Formally, tensors are multilinear maps from vector spaces to the real numbers (V vector space, and V^* dual space)

$$f : \underbrace{V^* \times \cdots V^*}_{p \text{ copies}} \times \underbrace{V \times \cdots V}_{q \text{ copies}} \to \mathbb{R}$$

- A scalar is a tensor ($f : \mathbb{R} \to \mathbb{R}, \ f(e_1) = c$)
- A vector is a tensor ($f : \mathbb{R}^n \to \mathbb{R}, \ f(e_i) = v_i$)
- A matrix is a tensor ($f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}, \ f(e_i, e_j) = A_{ij}$)
- Common to have fixed basis, **so a tensor can be represented as a multidimensional array of numbers.**

Why the „Flow“?

- Tensorflow constructs a graph of the evaluation G
- *The nodes in the graph are evaluated upon request (.eval() / session.run()) – TF1.0*
- TenorFlow can automatically calculate derivates for each of the nodes and therefor we can use SGD easily to optimize w.r.t. Arbitrary error metric

- *Significant changes in TF2.0 interface (but internally it remains a graph all the same)*

TensorFlow 1.0 code

```
In [31]: import tensorflow as tf
#import tensorflow.compat.v1 as tf
In [32]: tf.InteractiveSession()

In [33]: a=tf.zeros((2,2)); b=tf.ones((2,2))

In [34]: tf.reduce_sum(b, reduction_indices=1).eval()
Out[34]: array([ 2.,  2.], dtype=float32)

In [35]: a.get_shape()
Out[35]: TensorShape([Dimension(2), Dimension(2)])

In [36]: tf.reshape(a, (1, 4)).eval()
Out[36]: array([[ 0.,  0.,  0.,  0.]], dtype=float32)
```

More on Session soon

More on .eval() in a few slides

TensorShape behaves like a python tuple.

TensorFlow Session Object (1) # TF1.0

- “A Session object encapsulates the environment in which Tensor objects are evaluated” - TensorFlow Docs

```
In [20]: a = tf.constant(5.0)

In [21]: b = tf.constant(6.0)

In [22]: c = a * b

In [23]: with tf.Session() as sess:
    ....:     print(sess.run(c))
    ....:     print(c.eval())
    ....:
30.0
30.0
```

c.eval() is just syntactic sugar for sess.run(c) in the currently active session!

Placeholders and Feed Dictionaries (2)

```
In [96]: input1 =tf.placeholder(tf.float32)

In [97]: input2 =tf.placeholder(tf.float32)

In [98]: output =tf.mul(input1, input2)

In [99]: with tf.Session() as sess:
   ....:         print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))
   ....:

[array([ 14.], dtype=float32)]
```

Define tf.placeholder *objects for data entry.*

Fetch value of output from computation graph.

Feed data into computation graph.

Ex: Linear Regression in TensorFlow (3)

```
# Define variables to be learned
with tf.variable_scope("linear-regression"):
    W = tf.get_variable("weights", (1, 1),
                        initializer=tf.random_normal_initializer())
    b = tf.get_variable("bias", (1,),
                        initializer=tf.constant_initializer(0.0))
    y_pred = tf.matmul(X, W) + b
    loss = tf.reduce_sum((y - y_pred)**2/n_samples)
```

Note reuse=False so these tensors are created anew

$$J(W,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (Wx_i + b))^2$$

TensorFlow 2.0

- Eager evaluation (no Sessions anymore)
- Model building with TF.Keras (high level abstraction)
- Compile model with error metric and solver
  - Train -> evaluate -> predict
- TF.data.Dataset handling the input data
  - (instead of placeholders)
  - Support for pipelining

- *Seemingly much more single-purpose than TF1.0*
  - *But common things are done easier*
  - *Heavily researched (outdated tutorials & StOF)*

TensorFlow 2.0 Datasets

- Use existing data or create your own e.g. from numpy
- Support for pipelined pre-processing

```
import tesorflow as tf
train, test = tf.keras.datasets.fashion_mnist.load_data()

images, labels = train
images = images/255

dataset = tf.data.Dataset.from_tensor_slices((images, labels))
dataset
```

```
>> <TensorSliceDataset shapes: ((28, 28), ()), types: (tf.float64, tf.uint8)>
```

Then you want to create a way to iterate through data
More details: https://www.tensorflow.org/guide/data

TensorFlow 2.0 Models (TF.Keras)

- Common types of network layers
  - (abstraction on tensors)
- Sequential model
  - Simple pipeline processing (e.g. via dense layers)

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
# Define Sequential model with 3 layers
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Call model on a test input
x = tf.ones((3, 3))
y = model(x)
```

More details: https://www.tensorflow.org/guide/keras/sequential_model

TensorFlow 2.0 Models (TF.Keras)

- Continue with tf2_basics.ipynb

„Large“ homework:
- see nprg067_04_task.ipynb

- Build a siammese network:
  - Is the pair of data from the same class?
  - One-shot classification task
    - What if a new class emerges?
    - Re-train model (standard ML) or use similarity of embeddings
  - Two „towers“ that share parameters and the similarity calculation on top of it

  - Use tf.keras layers + model