Deep Learning Tools & Frameworks

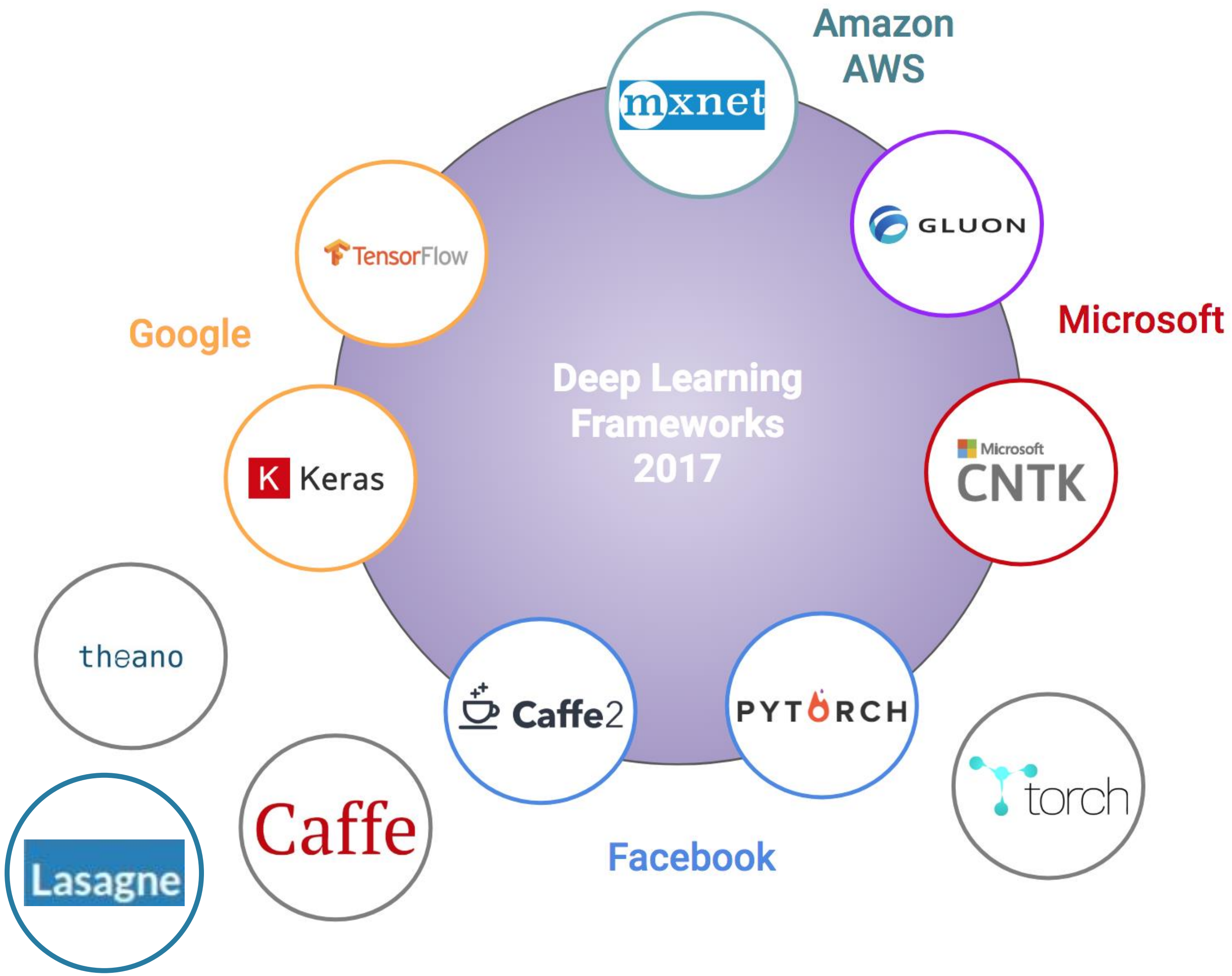
Danilo Pau

Advanced System Technology

Agrate Brianza

life.augmented

Many Deep Learning Frameworks



DL Framework Popularity (Oct.17)

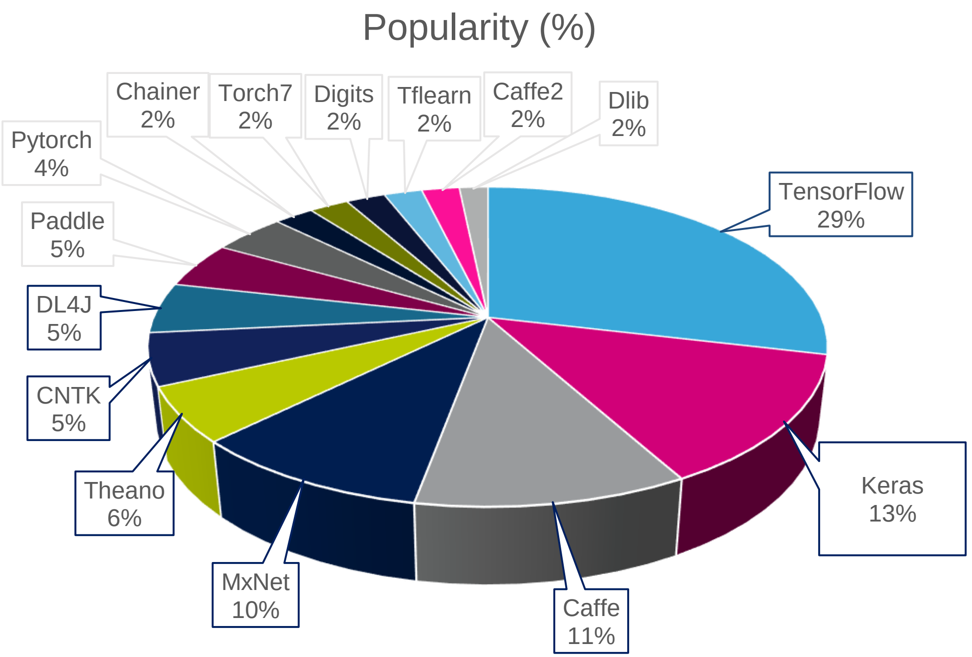
- TensorFlow dominates the field with the largest active community:
  - It can be used as a back-end in Keras and Sonnet
  - Pros: general-purpose deep learning framework, flexible interface, good-looking computational graph visualizations, and Google's significant developer and community resources.
- Keras is the most popular front-end for deep learning:
  - Used as a front-end for TensorFlow, Theano, MXNet, CNTK, or deeplearning4j.
  - Pros: simplicity, ease-of-use, allowing fast protoyping at the cost of some of the flexibility and control that comes from working directly with a framework.
- Caffe has yet to be replaced by Caffe2:
  - Caffe2 is a more lightweight, modular, and scalable version of Caffe that includes recurrent neural networks.
  - Caffe and Caffe2 are separate repos, so data scientists can continue to use the orginial Caffe.
  - However, there are migration tools such as Caffe Translator that provide a means of using Caffe2 to drive existing Caffe models.
- Theano continues to hold a top spot even without large industry support
- Sonnet (Deepmind 2017) is the fastest growing library
  - a high-level object oriented library built on top of TensorFlow. +272% Q3'17vs Q2'17 for Google Search.
  - DeepMind has a focus on Artificial general Intelligence and Sonnet can help a user build on top of their specific AI ideas and research.

GitHub DL Frameworks Aggregated Popularity (Oct.2017)

Aggregate popularity (30•contrib + 20•issues + 3•forks + 1•stars)•1e-3

#1:	377.51	tensorflow/tensorflow
#2:	174.15	fchollet/keras
#3:	143.84	BVLC/caffe
#4:	128.26	dmlc/mxnet
#5:	72.85	Theano/Theano
#6:	69.32	Microsoft/CNTK
#7:	67.30	deeplearning4j/deeplearning4j
#8:	61.54	baidu/paddle
#9:	54.07	pytorch/pytorch
#10:	29.65	pfnet/chainer
#11:	29.35	torch/torch7
#12:	29.33	NVIDIA/DIGITS
#13:	28.42	tflearn/tflearn
#14:	28.09	caffe2/caffe2
#15:	21.41	davisking/dlib

https://twitter.com/fchollet/status/915366704401719296



* = DL Frameworks Callouts with blu line are supported by ST Automatic NN Mapping Tool

https://blog.thedataincubator.com/2017/10/ranking-popular-deep-learning-libraries-for-data-science/

DL Framework Popularity (Oct.17)

DL Framework	Rank	Overall	Github	Stack Overflow	Google Results
tensorflow	1	10.87	4.25	4.37	2.24
keras	2	1.93	0.61	0.83	0.48
caffe	3	1.86	1.00	0.30	0.55
theano	4	0.76	-0.16	0.36	0.55
pytorch	5	0.48	-0.20	-0.30	0.98
sonnet	6	0.43	-0.33	-0.36	1.12
mxnet	7	0.10	0.12	-0.31	0.28
torch	8	0.01	-0.15	-0.01	0.17
cntk	9	-0.02	0.10	-0.28	0.17
dlib	10	-0.60	-0.40	-0.22	0.02
caffe2	11	-0.67	-0.27	-0.36	-0.04
chainer	12	-0.70	-0.40	-0.23	-0.07
paddlepaddle	13	-0.83	-0.27	-0.37	-0.20
deeplearning4j	14	-0.89	-0.06	-0.32	-0.51
lasagne	15	-1.11	-0.38	-0.29	-0.44
bigdl	16	-1.13	-0.46	-0.37	-0.30
dynet	17	-1.25	-0.47	-0.37	-0.42
apache singa	18	-1.34	-0.50	-0.37	-0.47
nvidia digits	19	-1.39	-0.41	-0.35	-0.64
matconvnet	20	-1.41	-0.49	-0.35	-0.58
tflearn	21	-1.45	-0.23	-0.28	-0.94
nervana neon	22	-1.65	-0.39	-0.37	-0.89
opennn	23	-1.97	-0.53	-0.37	-1.07

life.augmented

Interoperability

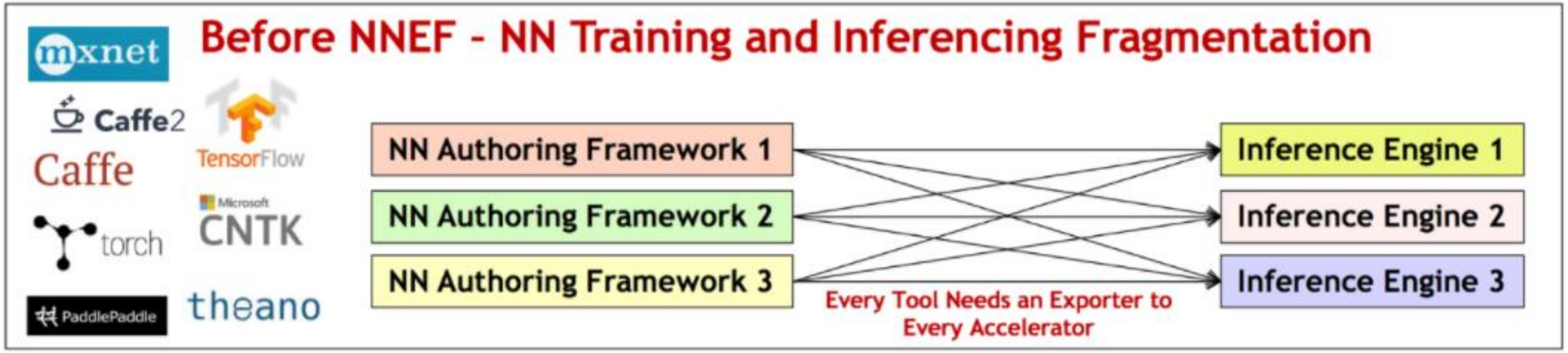
- https://onnx.ai/

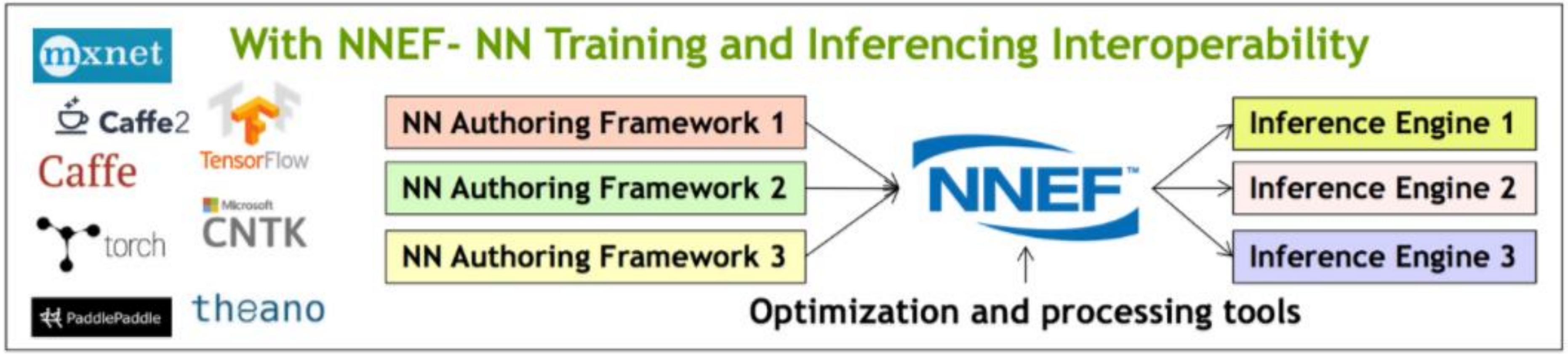
What is ONNX?

ONNX is a open format to represent deep learning models. With ONNX, AI developers can more easily move models between state-of-the-art tools and choose the combination that is best for them. ONNX is developed and supported by a community of partners.

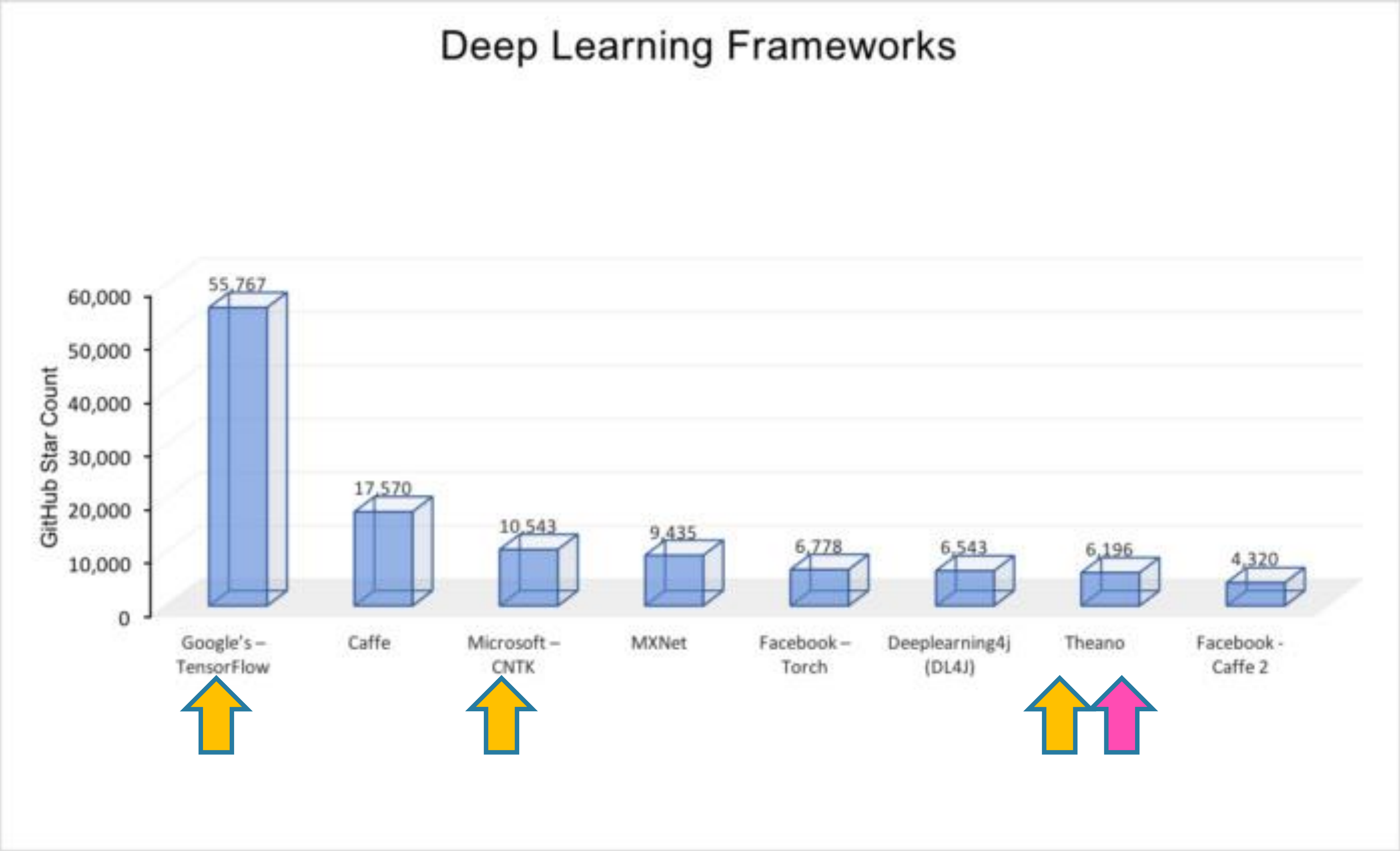
Interoperability

- https://www.khronos.org/nnef





Keras (2017)



= with Keras

= with Lasagne

https://www.cio.com/article/3193689/artificial-intelligence/which-deep-learning-network-is-best-for-you.html

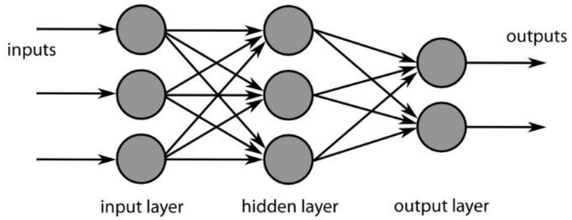
Keras

- A Python based high-level neural networks API
- Designed to be minimalistic & straight forward yet extensive (e.g. Lamba layers)
- Originally built as a wrapper around Theano.
- But now also work on top of TensorFlow or CNTK.
- The focus is making able the developers for prototyping in a fairly quick way with proprietary custom layers.

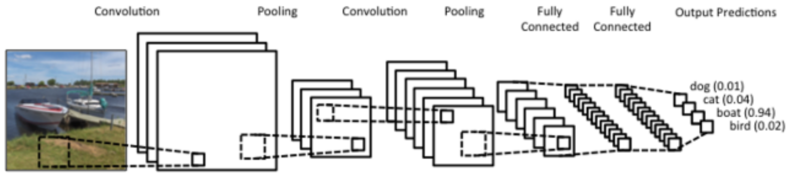
Keras

- Supports
  - Feed-Forward, Convolutional and Recurrent Neural Networks,
  - Reinforcement learning (maximize some notion of cumulative reward)
  - Linear and deep wide models
- Why to use Keras?
  - User friendliness: Simple to get started, simple to keep going, yet deep enough to make some serious complex models.
  - Modularity: Highly modular.
  - Easy extensibility: Easy to expand and add custom definitions.
  - Work with Python: Written python no new training and syntax knowledge required.

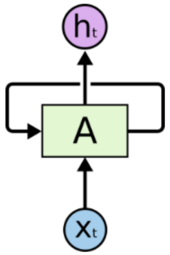
Coverage of Keras



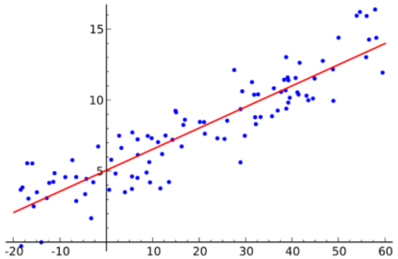
Feed forward neural network



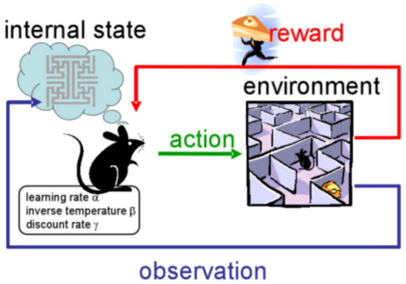
Convolutional neural network



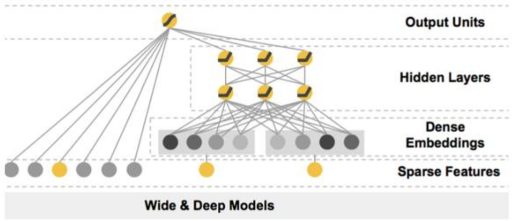
Recurrent neural network



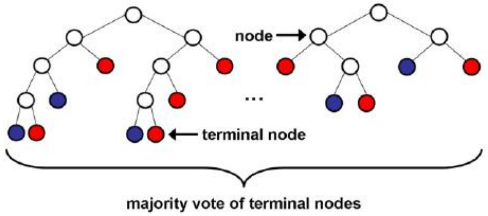
Linear models



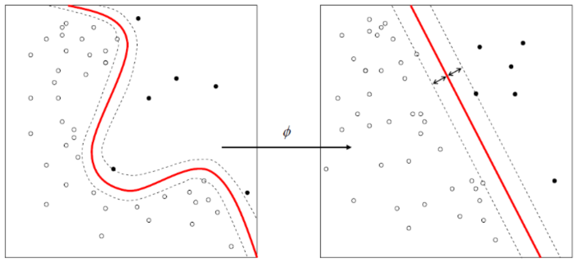
Reinforcement learning



Deep and wide models



Random forests



Support Vector Machines

life.augmented

Keras

- Link: https://keras.io/ (general information, documentation)
- Installation instructions: https://keras.io/#installation (OS related)
- Sample codes: https://github.com/fchollet/keras (openly available)
- A very nice link for starters: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/ (if you are new on Keras, this is highly recommended)

Keras: General Design Principals

General Idea in Keras is that it is based on layers and their inputs/outputs

- Prepare your inputs and output tensors
- Create first layer to handle the input tensor
- Create output layer to handle targets
- Build virtually any model layers you like in between

Keras has a number of built-in layers. Notable examples include

- Regular Dense layer: Fully connected, MLP type

  Syntax is

  keras.layers.core.Dense(output_dim, init = 'glorot_uniform', activation = 'linear', weights = None,
  b_regularizer = None, W_regularizer = None, activity_regularizer = None,
  W_constraint = None, b_constraint = None, input_dim = None)

- 1D Convolutional layer

  Syntax is

  keras.layers.convolutional.Convolution1D(nb_filter, filter_length, init = 'uniform', activation = 'linear',
  weights = None, border_mode = 'valid', input_dim = None
  W_regularizer = None, b_regularizer = None, W_constraint = None
  activity_regularizer = None, b_constraint = None,
  keranal_size=1)

Keras Architecture

- 2D Convolutional layer

  Syntax is

  keras.layers.convolutional.Convolution2D(nb_filter, filter_length, init = 'uniform', activation = 'linear',
  weights = None, border_mode = 'valid', input_dim = None,
  W_regularizer = None, b_regularizer = None, W_constraint = None
  activity_regularizer = None, b_constraint = None,
  kernel_size=(1,1))

- Recurrent layers, LSTM, GRU, etc.

  Syntax is

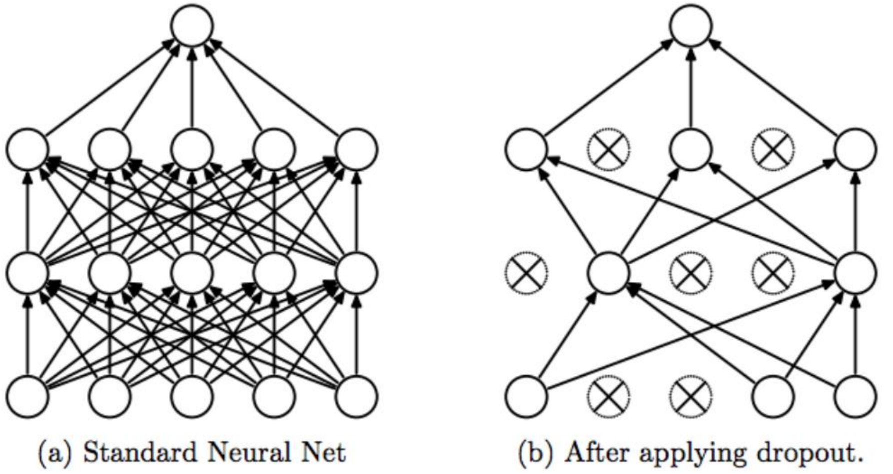
  keras.layers.recurrent.GRU(output_dim, nb_filter, filter_length, init = 'glotot_uniform', inner_init = 'orthogonal',
  activation = 'sigmoid', inner_activation = 'hard_sigmoid', statefull = False,
  go_backward = False, input_dim = None, input_length = None)

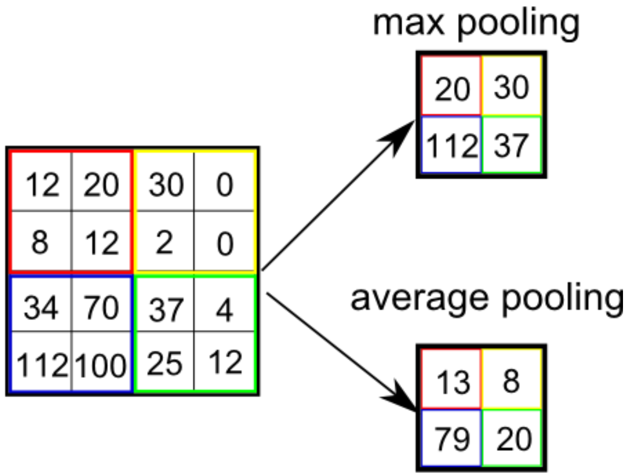
  .

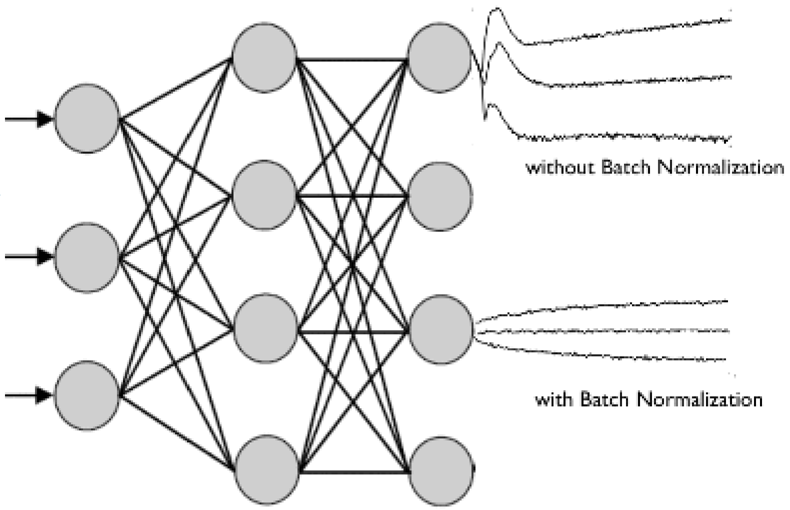
Keras Architecture

Some other types of supported layer includes

- Dropout
- Noise
- Pooling
- Normalization
- Embedding and many more







Keras Activations

- Almost all famous activations are available in Keras and can be added as an activation function to the layer. Such as
  - Sigmoid
  - Tanh
  - ReLu
  - Softmax
  - Softplus
  - Hard_sigmoid
  - Linear
- Advance activations as separate layers, include, LeakyRelu, PRelu, Elue, Parametric Softplus, Threshold linear etc.

Objectives and Optimizers

Objective functions

- Error loss: rmse, mse, mae, mape, msle
- Hinge loss: squared_hinge, hinge
- Class loss: binary_crossentropy, categorical_crossentropy

Optimizers

- Provides SGD, Adagrad, Adadelta, Rmsprop and Adam.
- All optimizers can be customized via parameters.

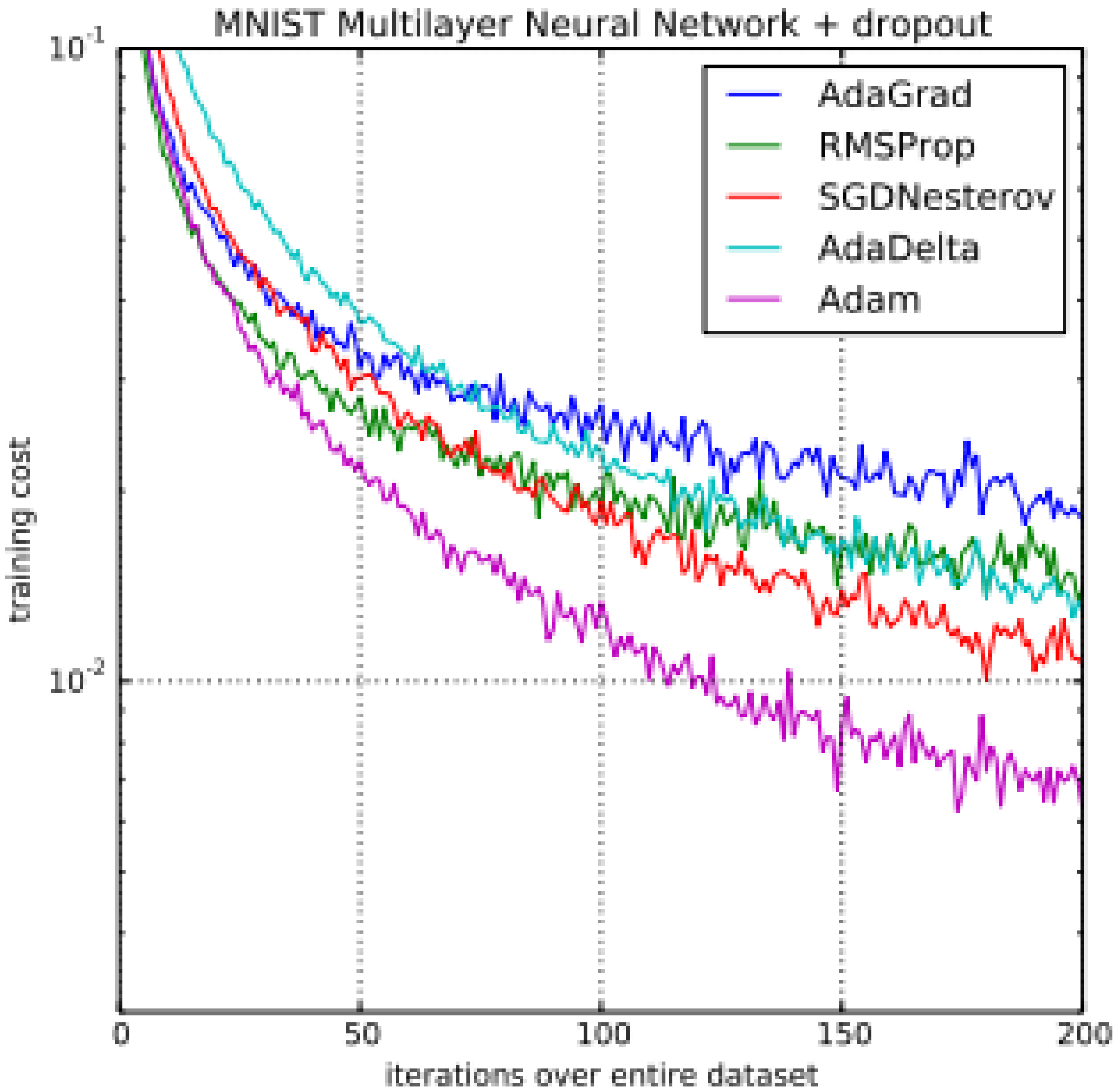
More on Optimizers

- **Adaptive Gradient Algorithm (AdaGrad)** : maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).

- **Root Mean Square Propagation (RMSProp) :** maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

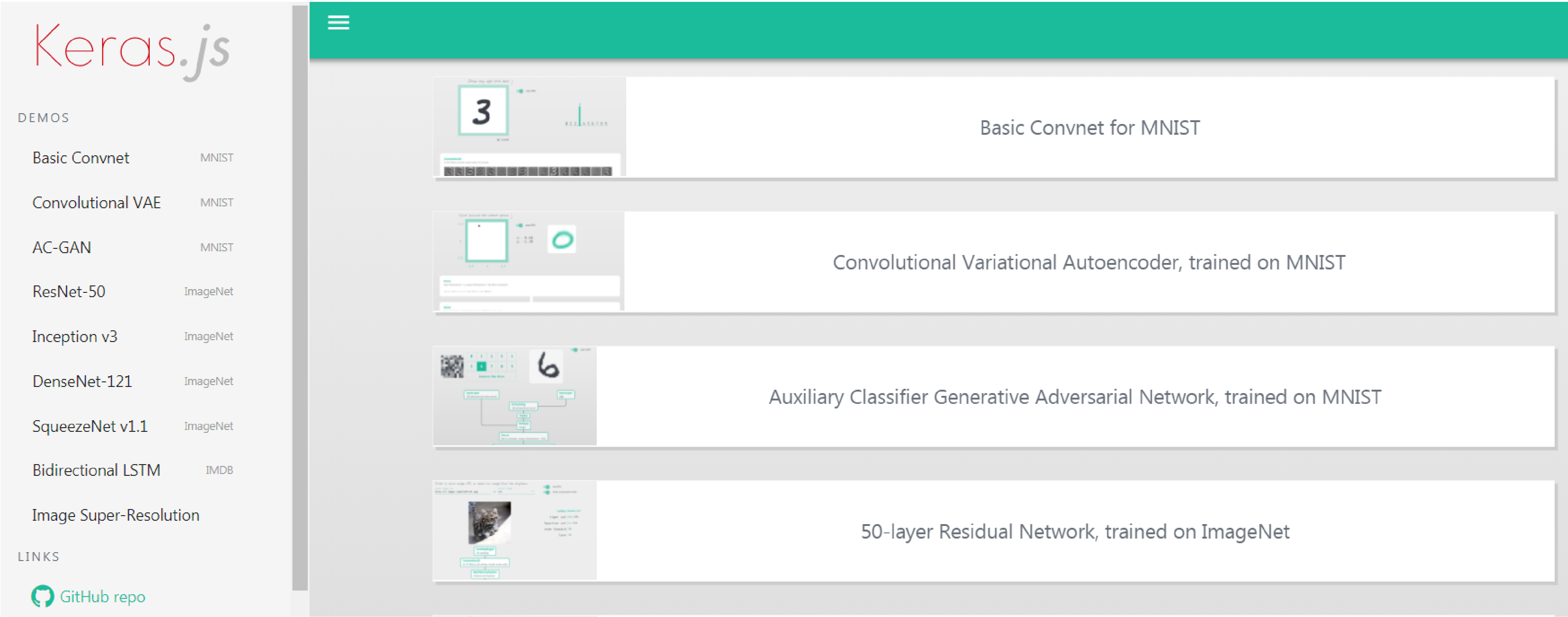
- **Adam** : adapts the parameter learning rates based on the average first moment (the mean) as in RMSProp, and also makes use of the average of the second moments of the gradients (the un centered variance).

More on Optimizers



Let’s see an example network…

https://transcranial.github.io/keras-js/#/



life.augmented

Untitled - Mozilla Firefox

localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3

File Edit View Insert Cell Kernel Widgets Help

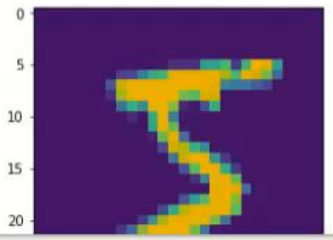
Trusted | Python 3

In [34]:
```python
# Importing the dependencies
%matplotlib inline
import numpy as np, keras.backend as K,matplotlib.pyplot as plt, os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from keras.datasets import mnist
from keras.utils import np_utils
from sklearn import metrics
```

In [22]:
```python
# setting the image convention to theano
K.set_image_dim_ordering('th')
# fix random seed for reproducibility
seed = 611
np.random.seed(seed)
```

In [50]:
```python
# Load and preprocess the database
(X_train, Y_train),(X_test,Y_test) = mnist.load_data()
plt.imshow(np.squeeze(X_train[0,:,:]))
print(Y_train[0])
numTrainImages = X_train.shape[0]
numTestImages = X_test.shape[0]
imRows = X_train.shape[1]
imCols = X_train.shape[2]
numChannels = 1
# reshaping the data to match the theano setup [samples][number of channels][width][height] in our case the number of chan
X_train = X_train.reshape(numTrainImages,numChannels,imCols,imRows).astype('float32')
X_test = X_test.reshape(numTestImages,numChannels,imCols,imRows).astype('float32')
# Normalizing the input images to be between 0 - 1
X_train = X_train/255
X_test = X_test/255
# one hot encode outputs (Mean the value is one if the right category zero otherswise)
Y_train = np_utils.to_categorical(Y_train)
Y_test = np_utils.to_categorical(Y_test)
num_classes = Y_test.shape[1]
```

5



life.augmented