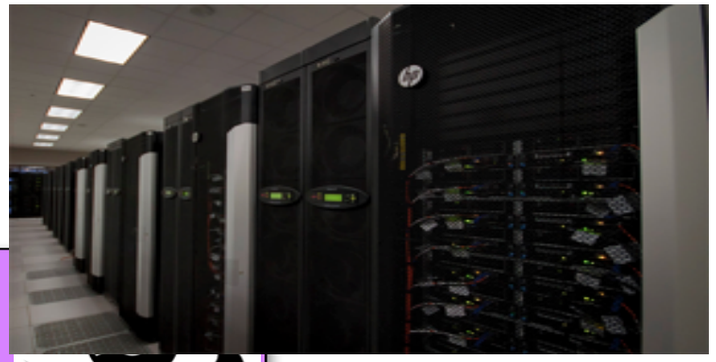
Deep Learning at TACC

Zhao Zhang
Data Intensive Computing
Texas Advanced Computing Center
April 23rd, 2019

TACC at a Glance

- Personnel
  - 150 Staff (~70 PhD)
- Facilities
  - 12 MW Data center capacity
  - Two office buildings, Three Datacenters, two visualization facilities, and a chilling plant.
- Systems and Services
  - A Billion compute hours per year
  - 5 Billion files, 50 Petabytes of Data, Hundreds of Public Datasets
- Capacity & Services
  - HPC, HTC, Visualization, Large scale data storage, Cloud computing
  - Consulting, Curation and analysis, Code optimization, Portals and Gateways, Web service APIs, Training and Outreach

TACC at a Glance



Stampede
HPC Jobs
6400+ Nodes
10 PFlops
14+ PB Storage

Maverick
Vis & Analysis
Interactive
Access
132 K40 GPUs

Lonestar
HTC Jobs
1800+ Nodes
22000+ Cores
146 GB/node

Wrangler
Data Intensive
Computations
10 PB Storage
High IOPS

Stockyard
Shared Workspace
20 PB Storage
1 TB per user
Project Workspace

Corral
Data Collections
6 PB Storage
Databases
IRODS

Rodeo/
Chameleon
Cloud Services
User VMs

Vis Lab
mmersive Vis
Colaborative
Touch Screen
3D

Ranch
Tape Archive
160 PB Tape
1+ PB Access
Cache

High Performance Computing and Machine Learning

- Stampede 2 - 18 Petaflops for simulation and modeling
  - Mixture of Xeon Phi and Xeon Platinum processors
  - Larger memory footprint per node

- Maverick 2 - Nvidia GPU cluster
  - GTX - 23 nodes, 4x GTX 1080 Ti GPUs per node
  - V100 - 4 nodes, 2x V100 GPUs per node
  - P100 - 3 nodes, 2x P100 GPUs per node

- Wrangler - 96 node cluster
  - Data intensive computing, 10PB disk based Lutre file system
  - Flash storage system


FRONTERA
TACC
TEXAS

Frontera

- Coming soon in 2019
- Aimed at the largest problems scientists and engineers currently face
- TACC will support and operate this system for 5 years
- Plan a potential Phase 2 system, with 10x the capability ties, for the future challenges scientists will face

Frontera

- Primary compute system: DellEMC and Intel
  - 35-40 PetaFlops Peak Performance (Next Generation Xeon processors)
- Interconnect: Mellanox HDR and HDR-100 links.
  - Fat Tree topology, 200Gb/s links between switches.
- Storage: DataDirect Networks
  - 50+ PB disk, 3PB of Flash, 1.5TB/sec peak I/O rate.
- Single Precision Compute Subsystem: Nvidia
- Front end for data movers, workflow, API

Agenda

- Deep Learning Overview
- Image Classification Example
- Hands-on Exercise

Deep Learning Overview

- Introduction
- From Linear Regression to Neural Network
- Notions

Deep Learning Overview

- Deep learning is a class of machine learning algorithms in which multiple layers of nonlinear processing units are used for feature extraction and transformation, with each successive layer taking the output from the previous layer as input

Deep Learning Applications

- Autonomous Driving

https://upload.wikimedia.org/wikipedia/commons/1/1b/Google%27s_Lexus_RX_450h_Self-Driving_Car.jpg
https://upload.wikimedia.org/wikipedia/commons/thumb/c/cd/Uber_OTTO_autonomous_driving_truck.jpg/640px-Uber_OTTO_autonomous_driving_truck.jpg

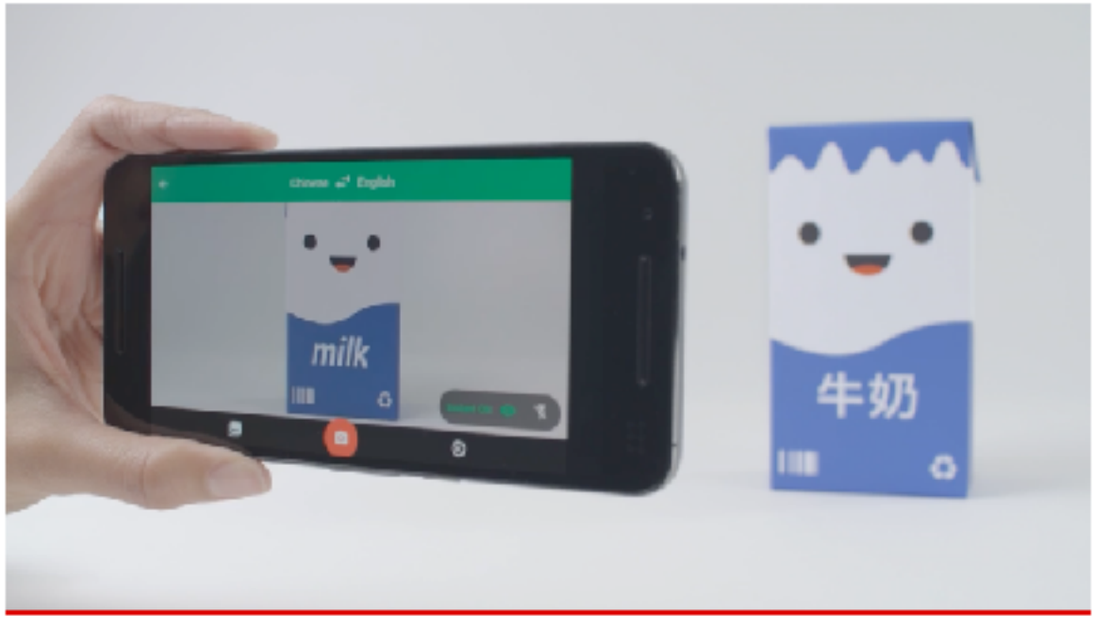
Deep Learning Applications

- The GO game

https://c1.staticflickr.com/2/1626/25708381781_eee5664c65_b.jpg

Deep Learning Applications

- Neural Machine Translation



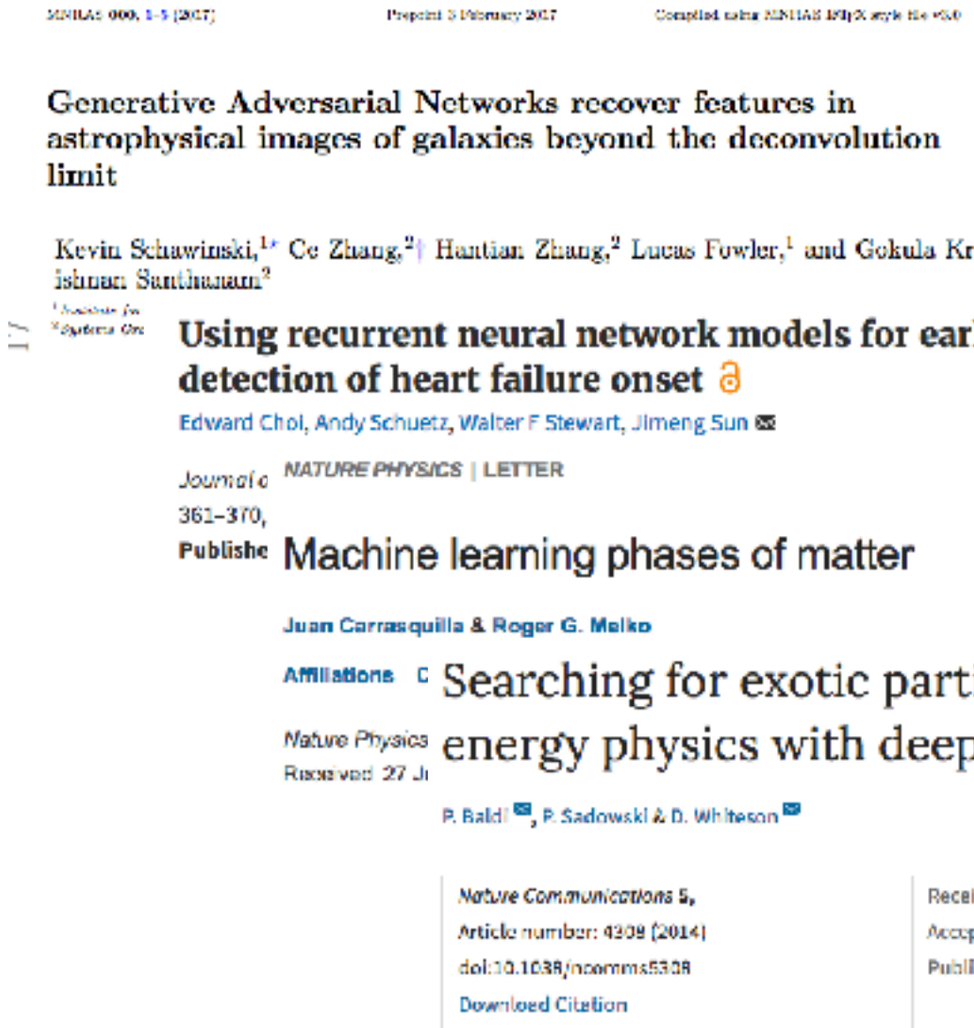
Google's new translation software is powered by brainlike artificial intelligence

By Catherine Matacic | Sep. 27, 2016, 2:45 PM

http://www.sciencemag.org/news/2016/09/google-s-new-translation-software-powered-brainlike-artificial-intelligence
http://mashable.com/2015/01/14/google-translate-word-lens/#WR1bRlKDfOq3

Deep Learning Applications

- Many scientists are exploring and adopting deep learning as the data science methodology to tackle their domain research challenge
  - Astronomy
  - Drug discovery
  - Disease diagnosis
  - Molecular dynamics
  - Neurology
  - Particle physics
  - Social science



MNRAS 000, 1–5 (2017) Preprint 3 February 2017 Compiled using MNRAS LATEX style file v3.0

Generative Adversarial Networks recover features in astrophysical images of galaxies beyond the deconvolution limit

Kevin Schawinski,[1]* Ce Zhang,[2]† Hantian Zhang,[2] Lucas Fowler,[1] and Gokula Krishnan Santhanam[2]

Using recurrent neural network models for early detection of heart failure onset

Edward Choi, Andy Schuetz, Walter F Stewart, Jimeng Sun

Journal o
361–370,
Publishe

NATURE PHYSICS | LETTER

Machine learning phases of matter

Juan Carrasquilla & Roger G. Melko

Affiliations C

Searching for exotic particles in high-energy physics with deep learning

Nature Physics
Received: 27 Ju

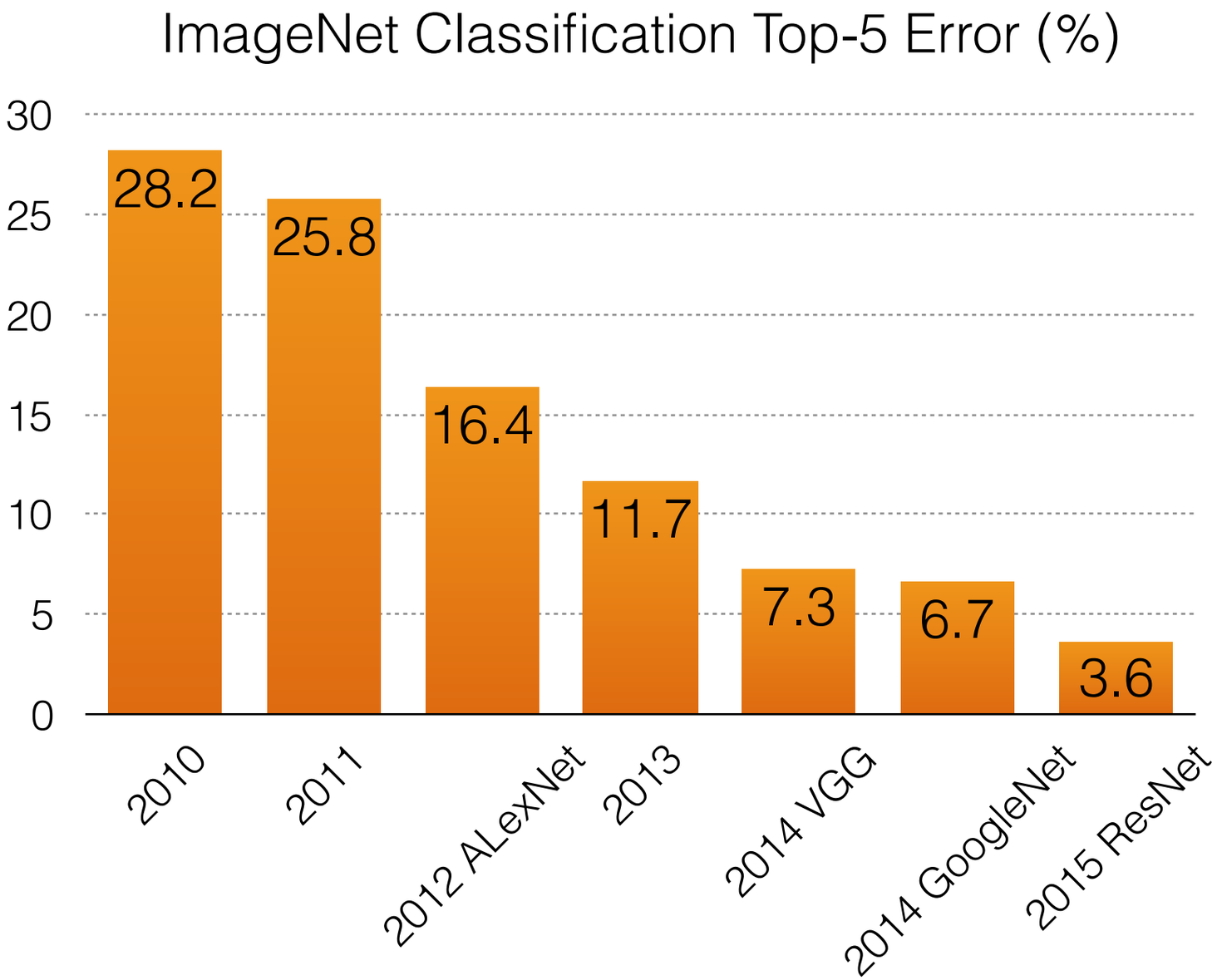
P. Baldi, P. Sadowski & D. Whiteson

Nature Communications 5,
Article number: 4308 (2014)
doi:10.1038/ncomms5308
Download Citation

Received: 19 February 2014
Accepted: 04 June 2014
Published online: 03 July 2014

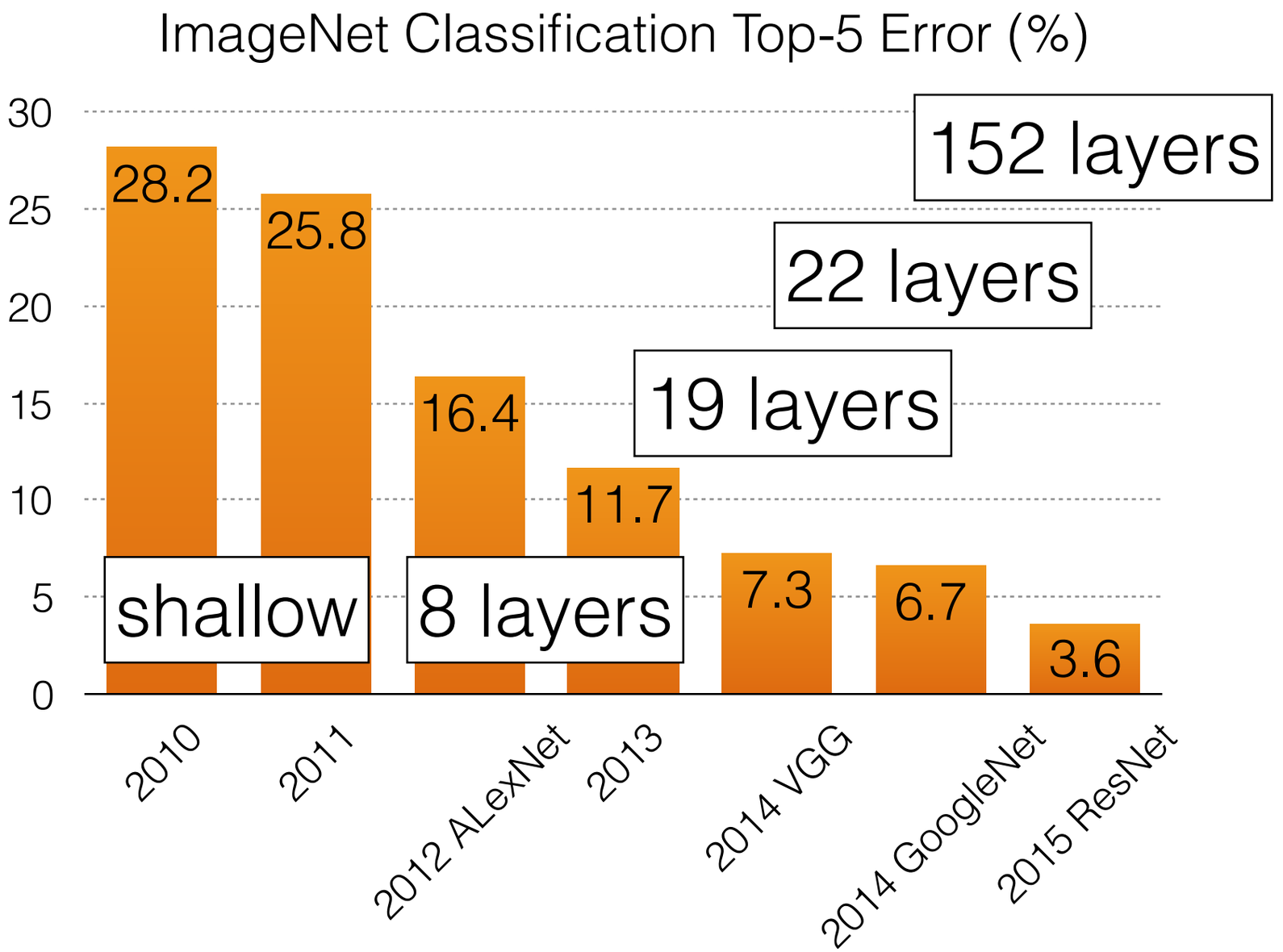
Deep Learning Applications

- Image Recognition



Deep Learning Applications

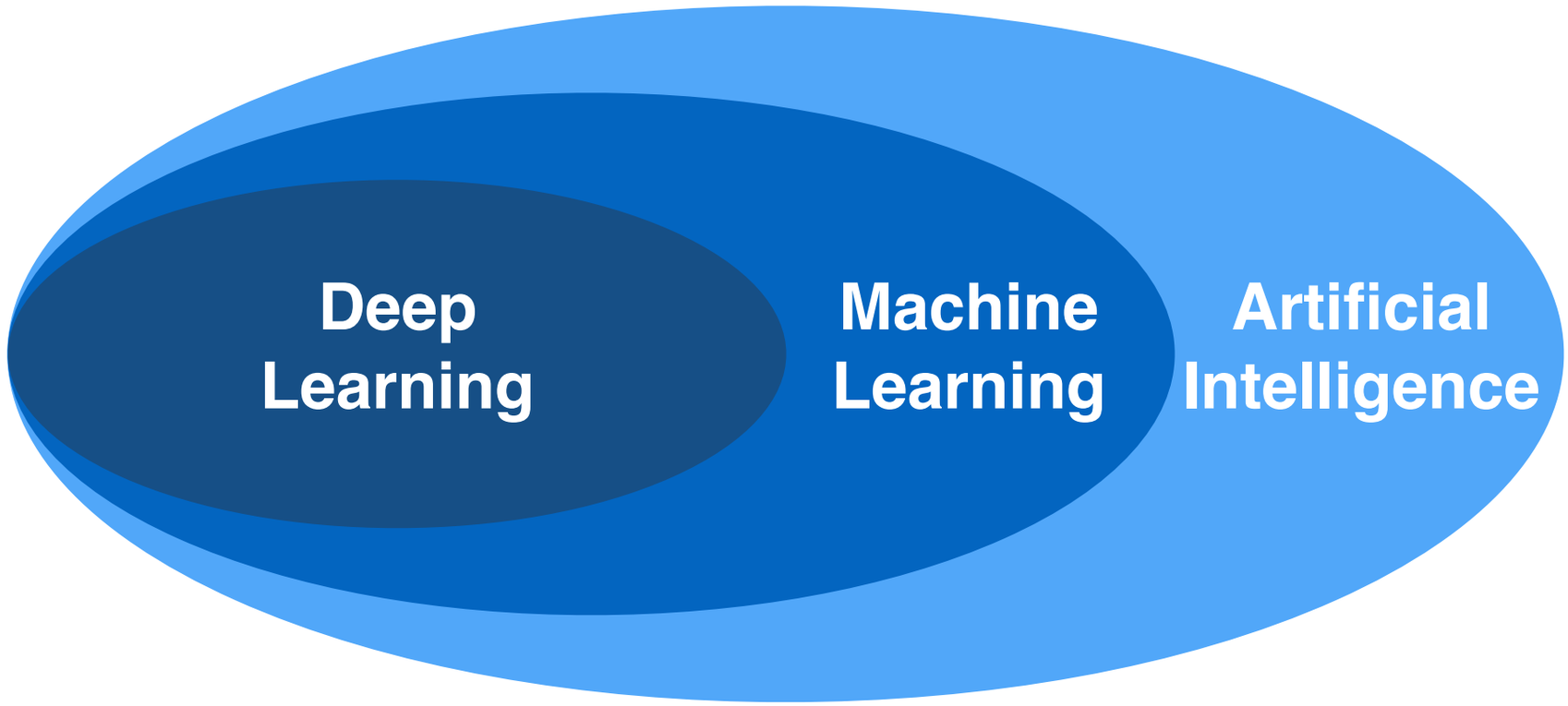
- Image Recognition



History

- 60s — Cybernetics
- 90s — Connectionism + Neural Networks
- 10s — Deep Learning
  - Two key factors for the on-going renaissance
  - Computing capability
  - Big Data

Deep Learning Landscape



Artificial Intelligence Scope

*"The true challenge to artificial intelligence proved to be solving **the tasks that are easy for people to perform but hard for people to describe formally**—problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images."*

— Ian Goodfellow, Yoshua Bengio, Aaron Courville

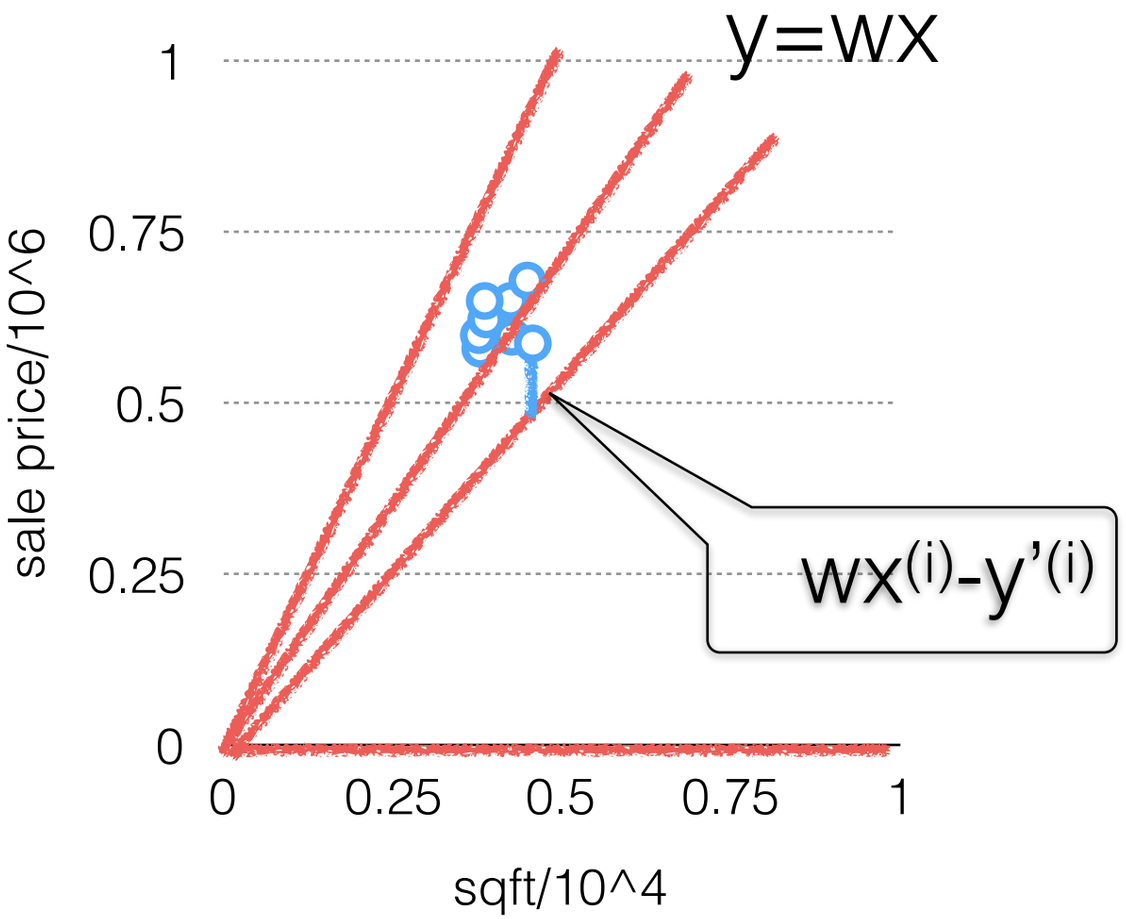
Deep Learning Overview

- Introduction
- From Linear Regression to Neural Network
- Notions

Linear Regression

- Example: Predicting house price with square footage

ID	$x^{(i)}$, sqft/10^4	$y'^{(i)}$, price/10^6
1	0.3801	0.58
2	0.4271	0.5975
3	0.4580	0.588
4	0.3780	0.6
5	0.3890	0.623
6	0.4250	0.65
7	0.4500	0.68
8	0.3867	0.65



Train a function y=w*x to minimize
Loss =1/2 * $\sum(wx^{(i)}-y'^{(i)})^2$

Linear Regression

- Example: Predicting house price with square footage
- We use the gradient descent method to find the value of w to be 1.427
- Now y=1.427*x, is this good or bad?

Model Generalization

- In practice, we divide a labeled training dataset into two parts. E.g., 80% and 20%, referred as training and validation dataset, respectively
- We derive the value of w using the training dataset.
  - value of w can be referred as model
- Then we apply the model to the validation dataset and compare the prediction with the labels
  - The difference between the prediction and the label is referred as error or loss
- A good model has low training error and low validation error
  - This is referred as good generalization

Deep Learning Introduction

- Example: Predicting house price with square footage
  - Example: Predicting house price with square footage
  - We use the gradient descent method to find the value of w to be 1.427
  - Now y=1.427*x, is this good or bad?

ID	$x^{(i)}$, sqft/10^4	$y'^{(i)}$, price/10^6	$y^{(i)}$, price/10^6
1	0.3905	0.57	0.56
2	0.3650	0.48	0.52
3	0.4230	0.62	0.60
4	0.3980	0.47	0.57

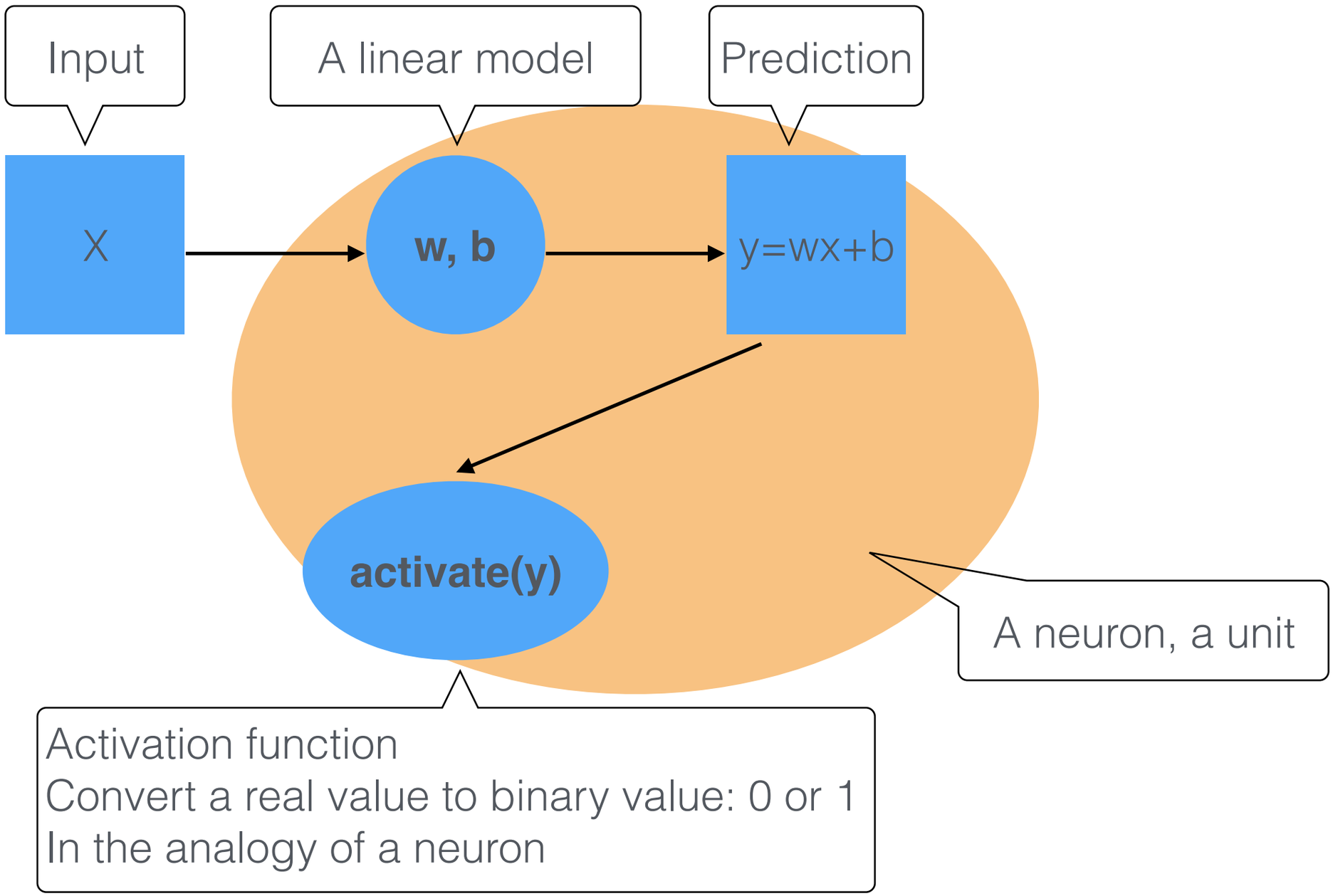
Deep Learning Introduction

- From Linear Regression to Neural Network
  - In practice
  - We use a bias item: $y = w*x + b$, or even a regularization item: $y = w*x + 0.5*\lambda*w^2$
  - We use a vector of $X = \{x_1, x_2, \ldots, x_n\}$ as the set of features
  - We use the gradient descent algorithm to find the w with minimum error
  - We use cross-entropy as error/loss instead of the distance

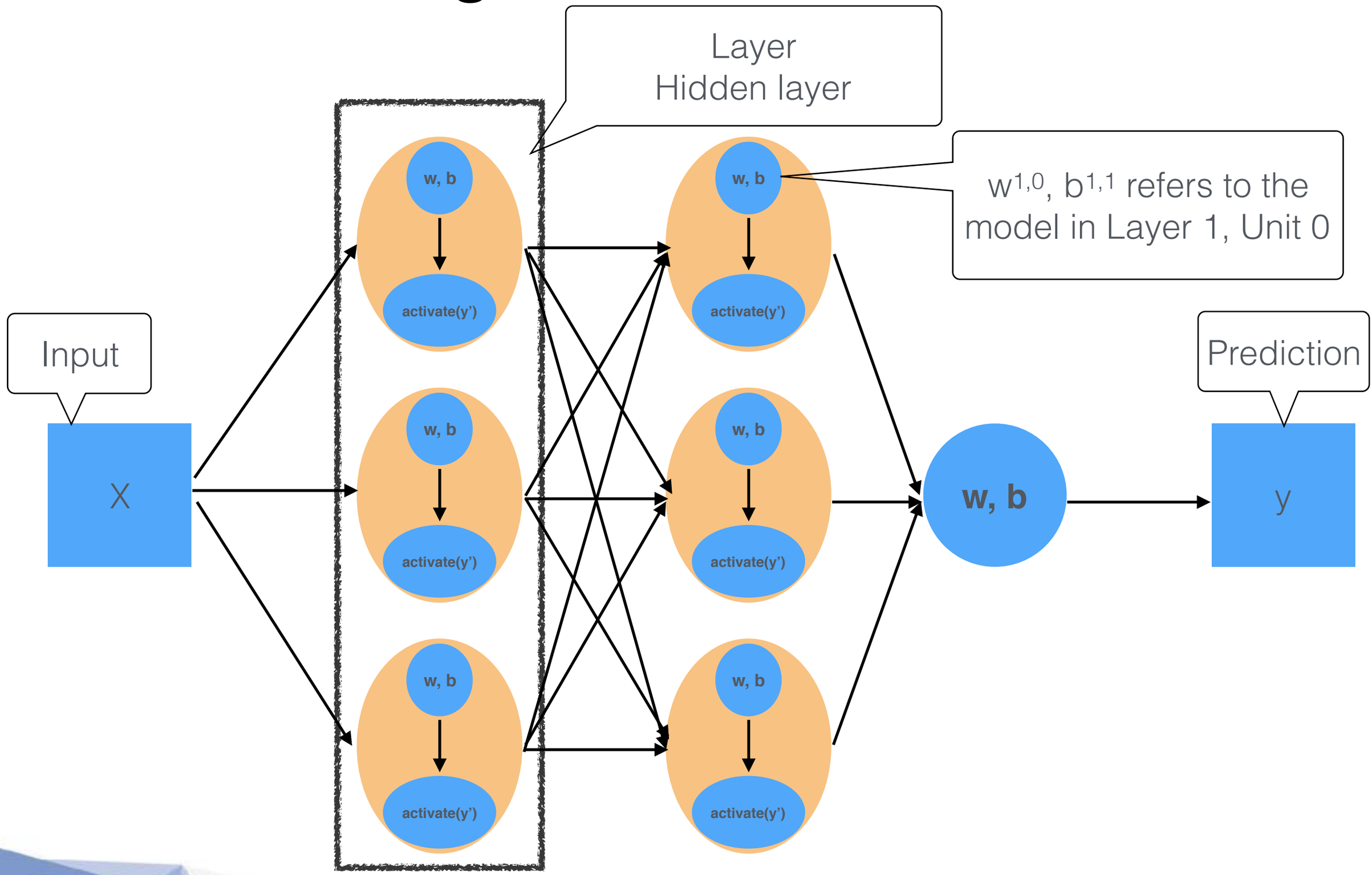
A Few Key Concepts

- For a machine learning/deep learning application
  - Labeled data
  - Training dataset and validation dataset
  - Error/Loss
  - Model generalization
    - A good model: low training error and low validation error

From Linear Regression to Neural Network

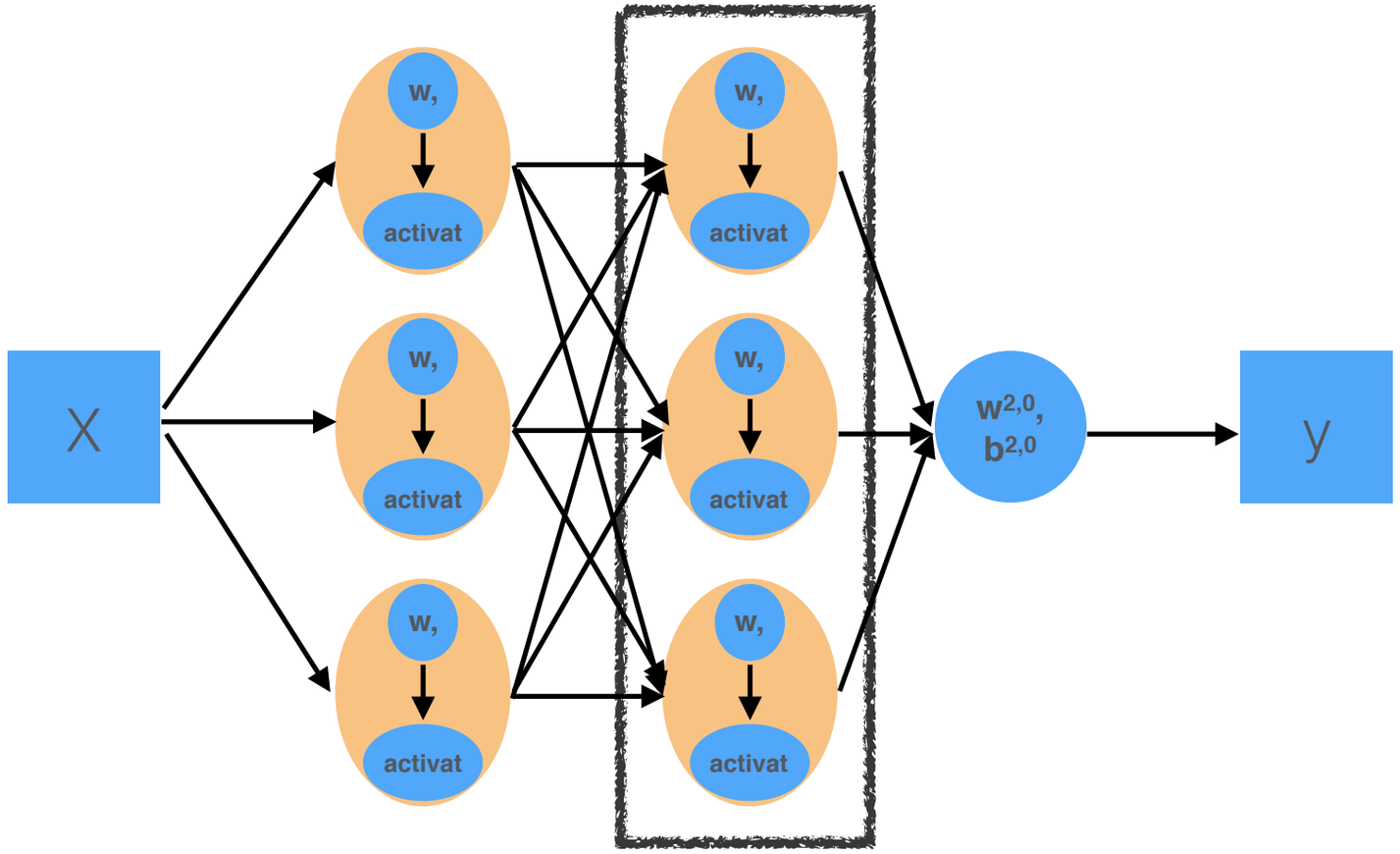


From Linear Regression to Neural Network



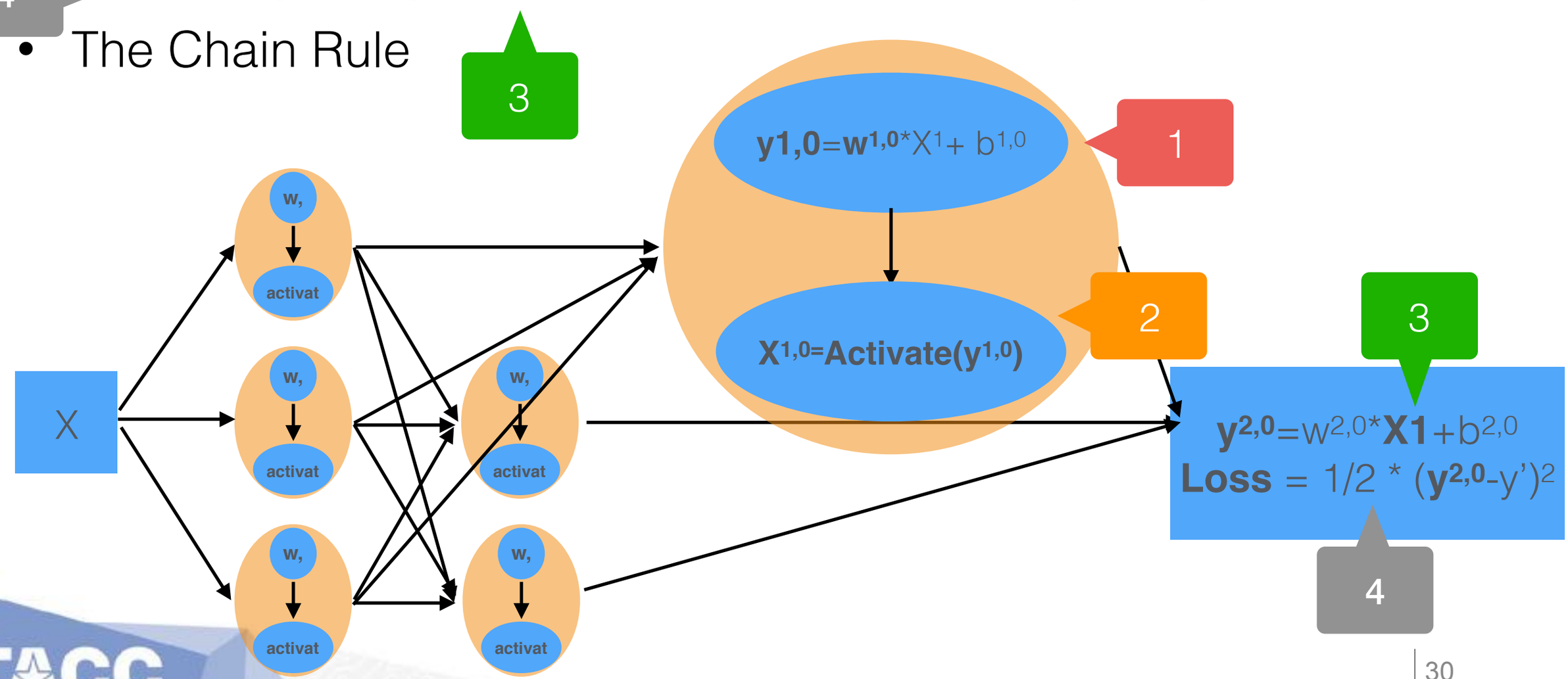
From Linear Regression to Neural Network

- Now we have labeled data
- We can calculate y and the error with label y’
- We can then update $w^{2,0}$
- How can we update $w^{1,0}$, $w^{1,1}$, $w^{1,2}$?



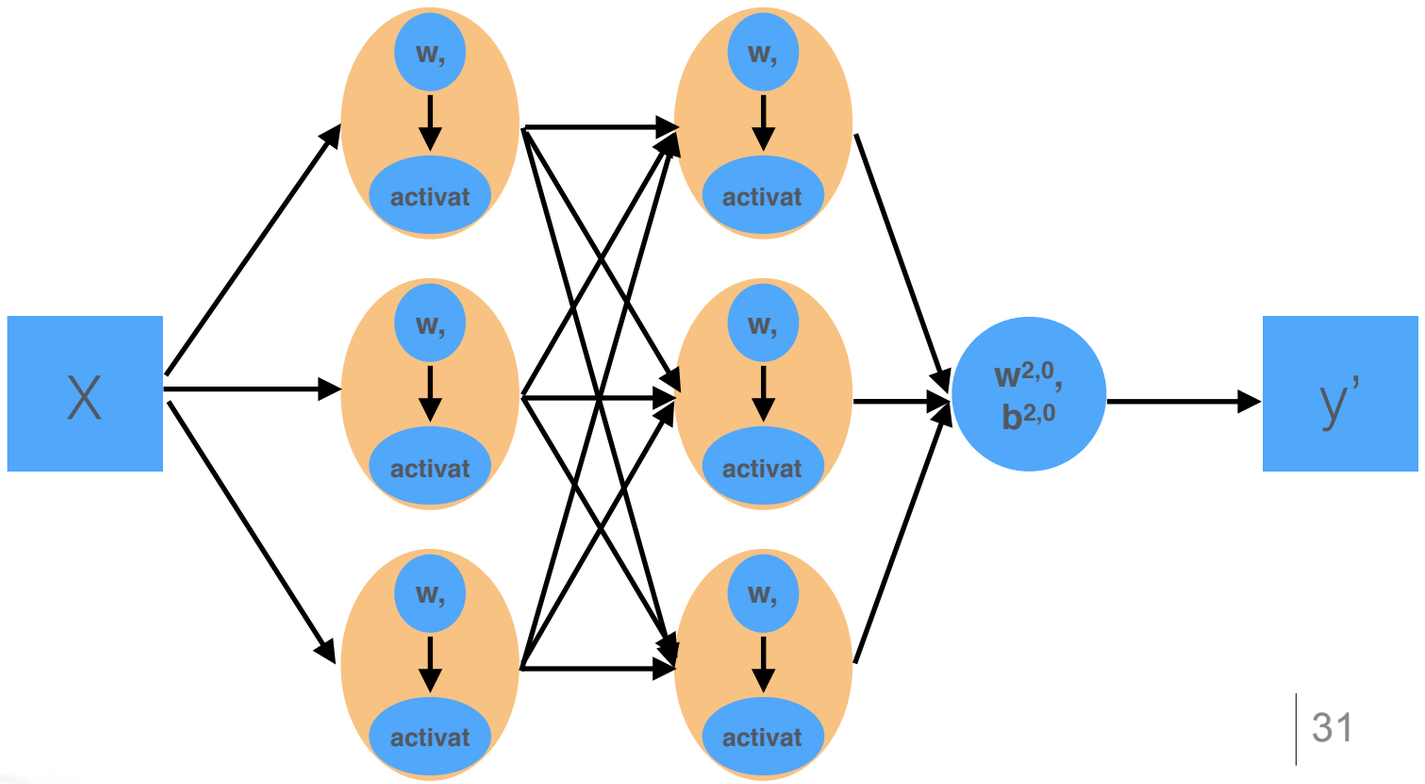
From Linear Regression to Neural Network

- The back-propagation algorithm
- $W^{1,0}$ should be updated as $W^{1,0}=W^{1,0} - \lambda * \partial Loss/\partial W^{1,0}$
- $\partial Loss/\partial W^{1,0} =$
  $\partial Loss/\partial y^{2,0} * \partial y^{2,0}/\partial Activate^{1,0} * \partial Activate^{1,0}/\partial y^{1,0} * \partial y^{1,0}/\partial W^{1,0}$
- The Chain Rule



From Linear Regression to Neural Network

- Stochastic Gradient Descent
- In the ImageNet-1K dataset, we have 1.2 million images
- So we have N=1.2 million X
- Traversing every single image takes too long
- So for each pass, we take a small size of n (e.g., n=512), and update the parameters based on the average loss



From Linear Regression to Neural Network

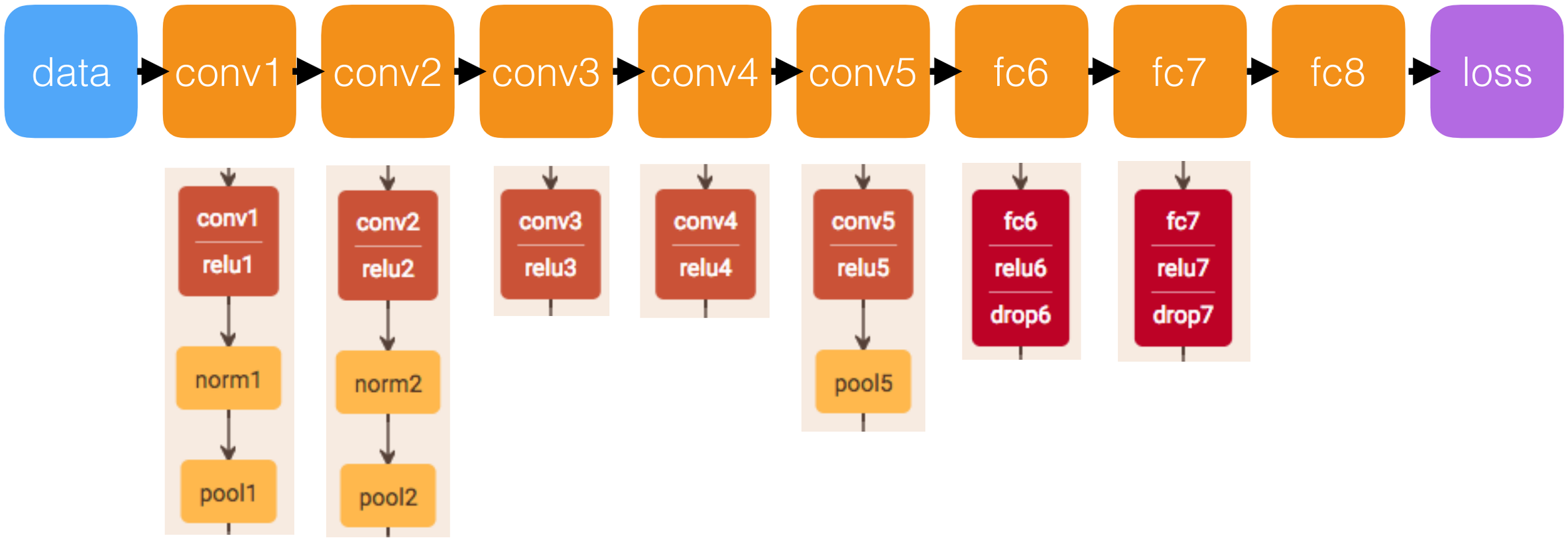
- The notion of Epoch
- The time by which every training data item is visited once
- So for1,200,000 images with a 512 mini-batch size, an epoch roughly take 2,400 iterations

- How many epochs is enough?
  - A somewhat standard practice uses 100 epochs for AlexNet and 90 epochs for ResNet-50

From Linear Regression to Neural Network

- What we just saw is a feed-forward network
- If in any layer, there is a convolution operations, it is called convolutional neural network
- Often coupled with pooling operation
- Example applications:
  - Image classification
  - Object detection
  - Autonomous driving

AlexNet

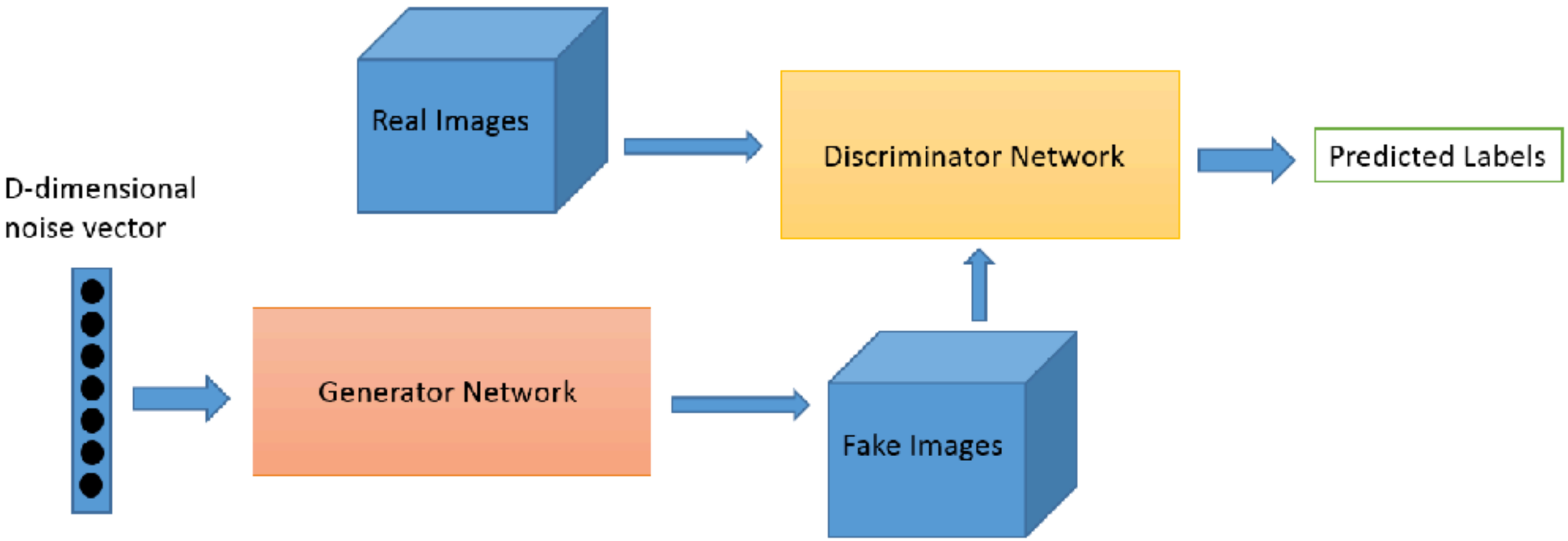


- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

From Linear Regression to Neural Network

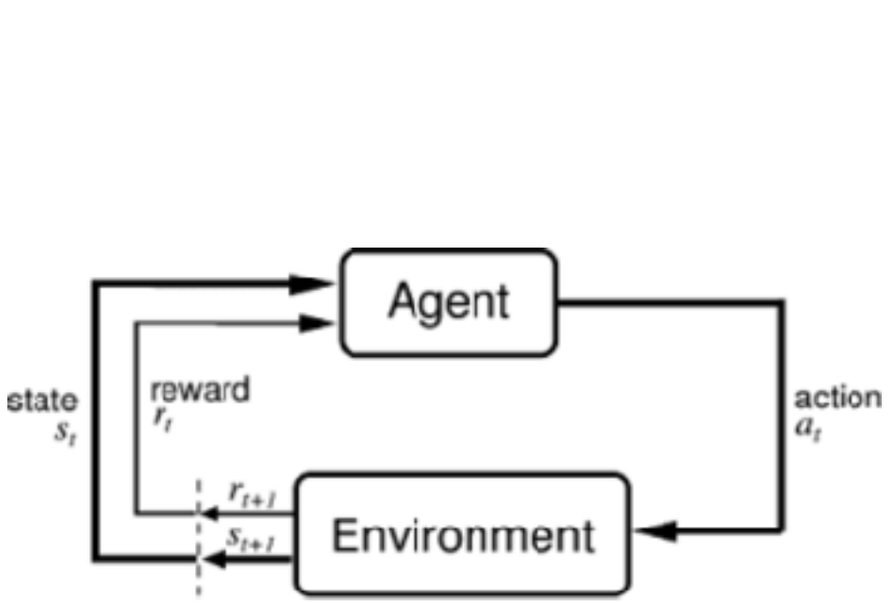
- Recursive Neural Network is another typical neural network architecture, mainly used for ordered/sequence input
- RNNs provide a way of use information about X_{t-i}, …, X_{t-1} for inferring X_t
- Example applications:
  - Language models, i.e. auto correction
  - Machine Translation
  - Auto image captioning
  - Speech Recognition
  - Autogenerating Music

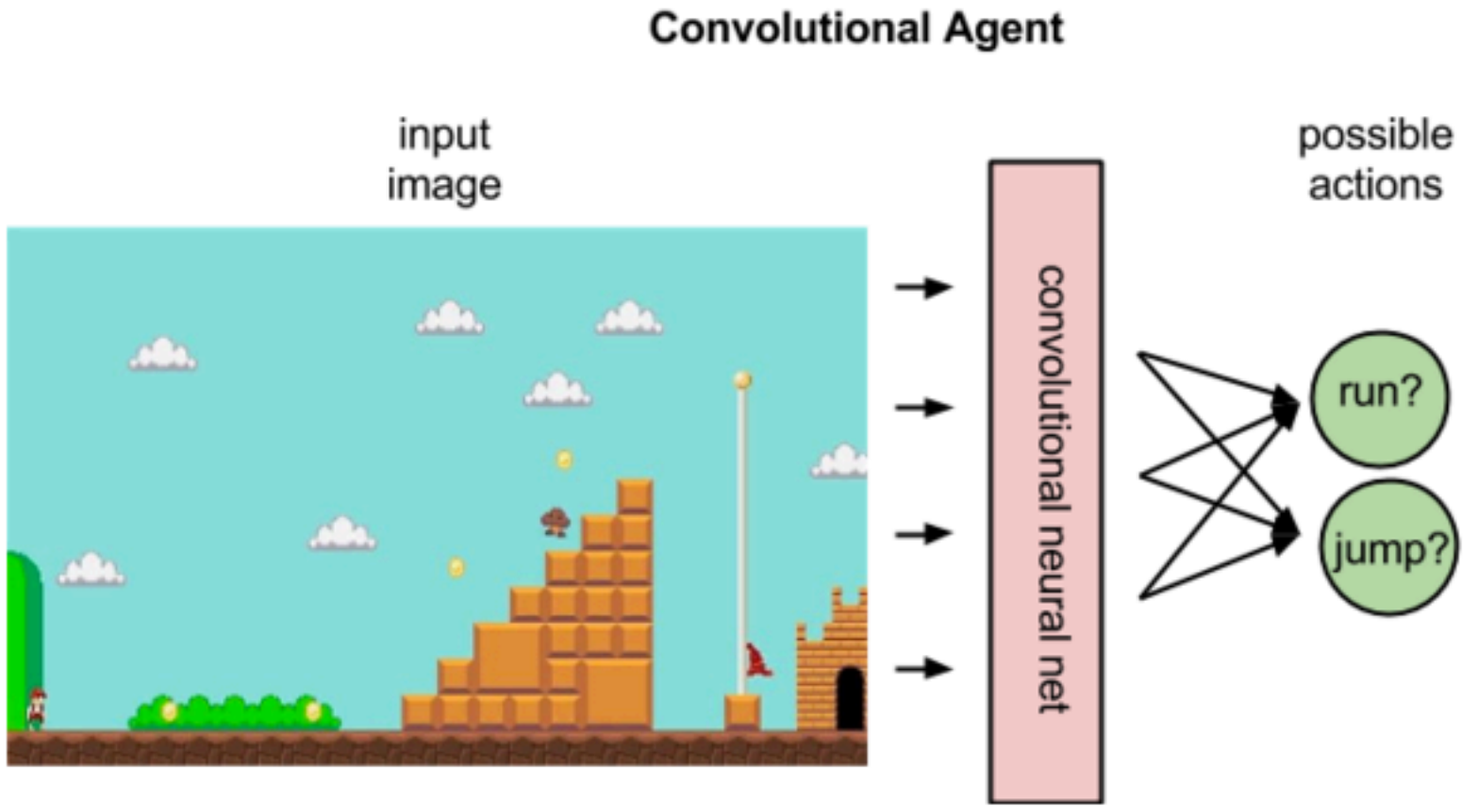
Generative Adversarial Network



Courtesy image from O'Reilly

Deep Reinforcement Learning





https://skymind.ai/wiki/deep-reinforcement-learning

Deep Learning Overview

- Introduction
- From Linear Regression to Neural Network
- Notions

Notions

- Neural Network Architecture
  - Multi-layer Perceptron
  - Convolutional Neural Network
  - Recurrent Neural Network
- Activation, Loss, and Optimization
  - Activation Function
  - Loss Function
  - Back-propagation
  - Gradient Descent
  - Stochastic Gradient Descent
- Training and Validating
  - Training Dataset
  - Validation/Test Dataset
  - Training Accuracy
  - Validation/Test Accuracy
    Training Loss
  - Validation/Test Loss
  - Epoch
  - Iteration/Step

Introduction to Keras and TensorFlow

David Walling, Zhao Zhang
Data Intensive Computing
Texas Advanced Computing Center
April 23rd, 2019

Goals

- Keras and TensorFlow Overview
- Sequential vs. Functional API
- Data Processing
- Handling Data -> Data Generators
- Example - Linear Regression

TensorFlow

- Product of Google Brain team.
- Open source symbolic math library ideal for DL computations.
- Build up computational graphs operating on n-dimensional arrays (tensors)
- Low level API, difficult to program
- Initial release 2015
- Version 1.0.0 release Feb 2017
- Current 1.13.1 release Feb 2019

Keras

- Keras is a Python API wrapping lower level Deep Learning (DL) frameworks including Tensorflow, Theano, and CNTK.
- Philosophy: “Being able to go from idea to result with the least possible delay is key to doing good research.”
- Original author: Google engineer François Chollet
- Provides many common building blocks for building DL models: layers, optimizers, activation functions
- Convenience functions for processing common data types: image and text

Example

```
# Linear Regression
from keras.models import Sequential
from keras.layers import Dense
from keras.datasets import boston_housing

# Load and Split Data
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
```

Example

```
# Build Model
model = Sequential()


model.add(Dense(8, init='normal', input_dim=13, activation='sigmoid'))
# Input + Hidden 1
model.add(Dense(8, init='normal', activation='sigmoid')) #  Hidden 2
model.add(Dense(1, init='normal')) # Output

model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=['mse', 'mape'])
```

Example

```
# Train
model.fit(x_train, y_train, epochs=100, batch_size=32)


# Evaluate
model.evaluate(x_test, y_test, batch_size=32)
```

Example

Epoch 1/10

404/404 [==============================] - 0s 885us/step - loss: 285.5638 - mean_squared_error: 285.5638 - mean_absolute_percentage_error: 57.4356

Epoch 2/10

404/404 [==============================] - 0s 46us/step - loss: 87.3172 - mean_squared_error: 87.3172 - mean_absolute_percentage_error: 33.3023

Epoch 3/10

404/404 [==============================] - 0s 42us/step - loss: 85.0584 - mean_squared_error: 85.0584 - mean_absolute_percentage_error: 34.2751

Epoch 4/10

404/404 [==============================] - 0s 39us/step - loss: 84.6952 - mean_squared_error: 84.6952 - mean_absolute_percentage_error: 35.4184

Epoch 5/10

404/404 [==============================] - 0s 41us/step - loss: 85.0854 - mean_squared_error: 85.0854

…

Example

…

Epoch 10/10

404/404 [==============================] - 0s 37us/step - loss: 84.7952 - mean_squared_error: 84.7952 - mean_absolute_percentage_error: 36.0463

102/102 [==============================] - 0s 1ms/step

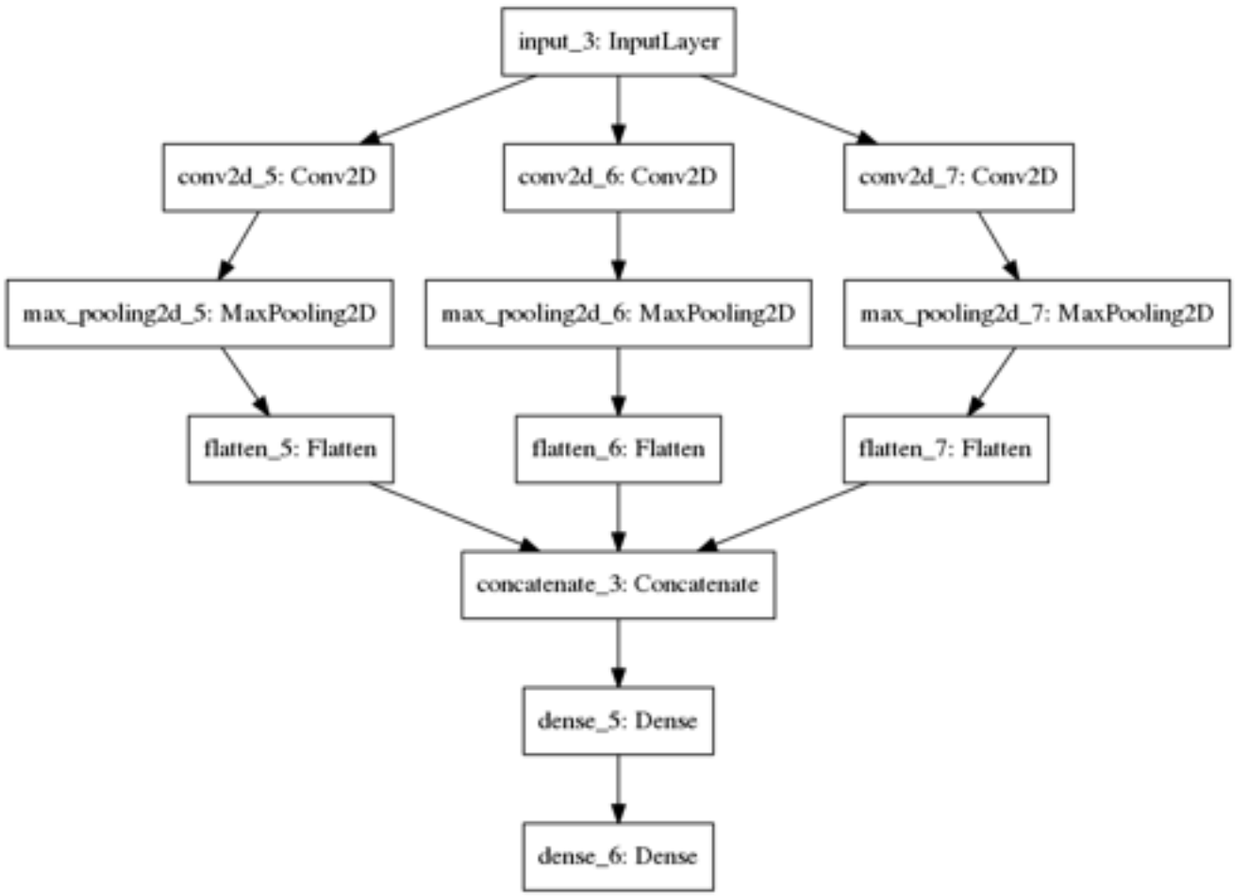
model.evaluate(x_test, y_test, batch_size=32)

102/102 [==============================] - 0s 40us/step

Out[28]: [83.557204003427543, 83.557204003427543, 37.89191358229693]

Sequential vs. Functional API

- Keras provides both a Sequential and Functional API
- Sequential: simpler for beginners, most tutorials use this
- Functional: advantages for complex models - Ex. DAGs or shared layers



Functional API

```
# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
```

Data Processing

- Keras provides a host of utility classes for dealing with common data sets
- Text Data
  - keras.preprocessing.text.Tokenizer
    - Transform text into integer representation
  - keras.preprocessing.sequence.pad_sequences
    - Ensure sequence inputs are all the same length
- Image Data
  - keras.preprocessing.image.ImageDataGenerator
    - Automatic image augmentation
    - Ensure images are the same size

Callbacks

- Keras provides a callback mechanism to easily implement
- Examples:
  - Checkpointing
  - ReduceLROnPlateau
  - EarlyStopping

```
filepath = output_path + "resnet-{epoch:02d}.hdf5"
checkpoint = ModelCheckpoint(filepath, period=5)
callbacks_list = [checkpoint]
model.compile(loss='categorical_crossentropy',
                    optimizer=’sgd’,
                    metrics=['categorical_accuracy'],
                    callbacks=callbacks_list)
```

Data Generators

- What happens if your input data does not fit into memory of CPU?
- Each iteration of forward/backward pass only works on data set = batch size, relatively small.
- Keras allows you to write custom ‘data generators’ to dynamically load data equal to each batch size.
- Automatically handles keeping track on indexes and pulling in next batch

Data Generators

```
datagen = ImageDataGenerator(
        # set input mean to 0 over the dataset
        featurewise_center=False,

        # set each sample mean to 0
        samplewise_center=False,

        # divide inputs by std of dataset
        featurewise_std_normalization=False,

        # divide each input by its std
        samplewise_std_normalization=False,

        # apply ZCA whitening
        zca_whitening=False,
        …
)
```

```
# Compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is applied).

datagen.fit(x_train)

# Fit the model on the batches generated by datagen.flow().

model.fit_generator(
    datagen.flow(x_train, y_train,
        batch_size=batch_size),
    validation_data=(x_test, y_test),
    epochs=epochs, verbose=1, workers=4,
    callbacks=callbacks)
```

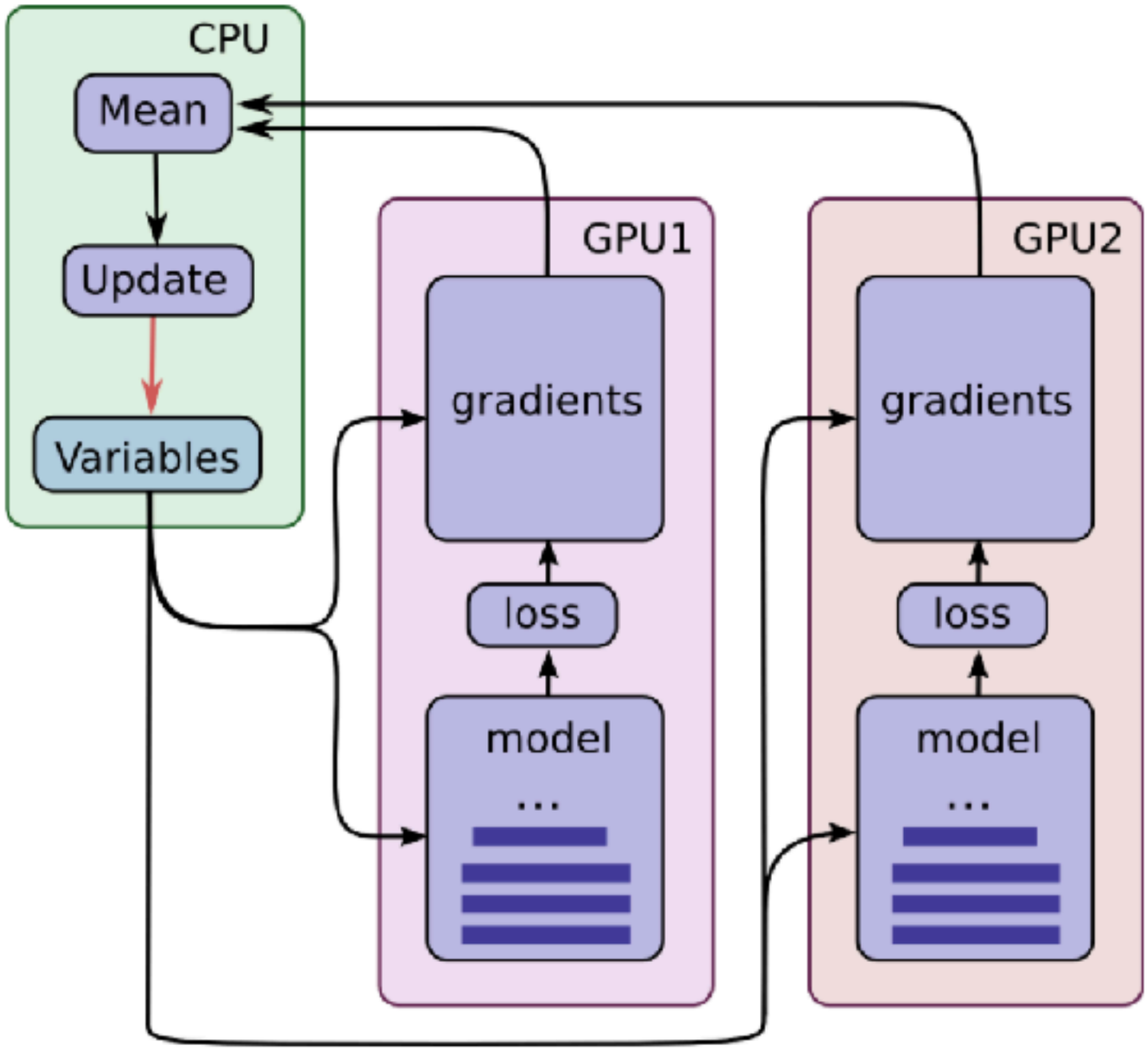
CPU vs. GPU

- Keras can support running on CPU vs GPU, generally with no changes to code.
- Execution will depend on which backend has been specified and which version of that backend is installed on the given resource.
- CPUs: good enough for development against small datasets to work through bugs and data cleansing issues. Can run larger batch sizes. Generally need to scale to multiple nodes.
- GPUs: great match for the type of heavy calculations involved in back-propagation algorithms, i.e. generally faster.

Multi-GPU Support

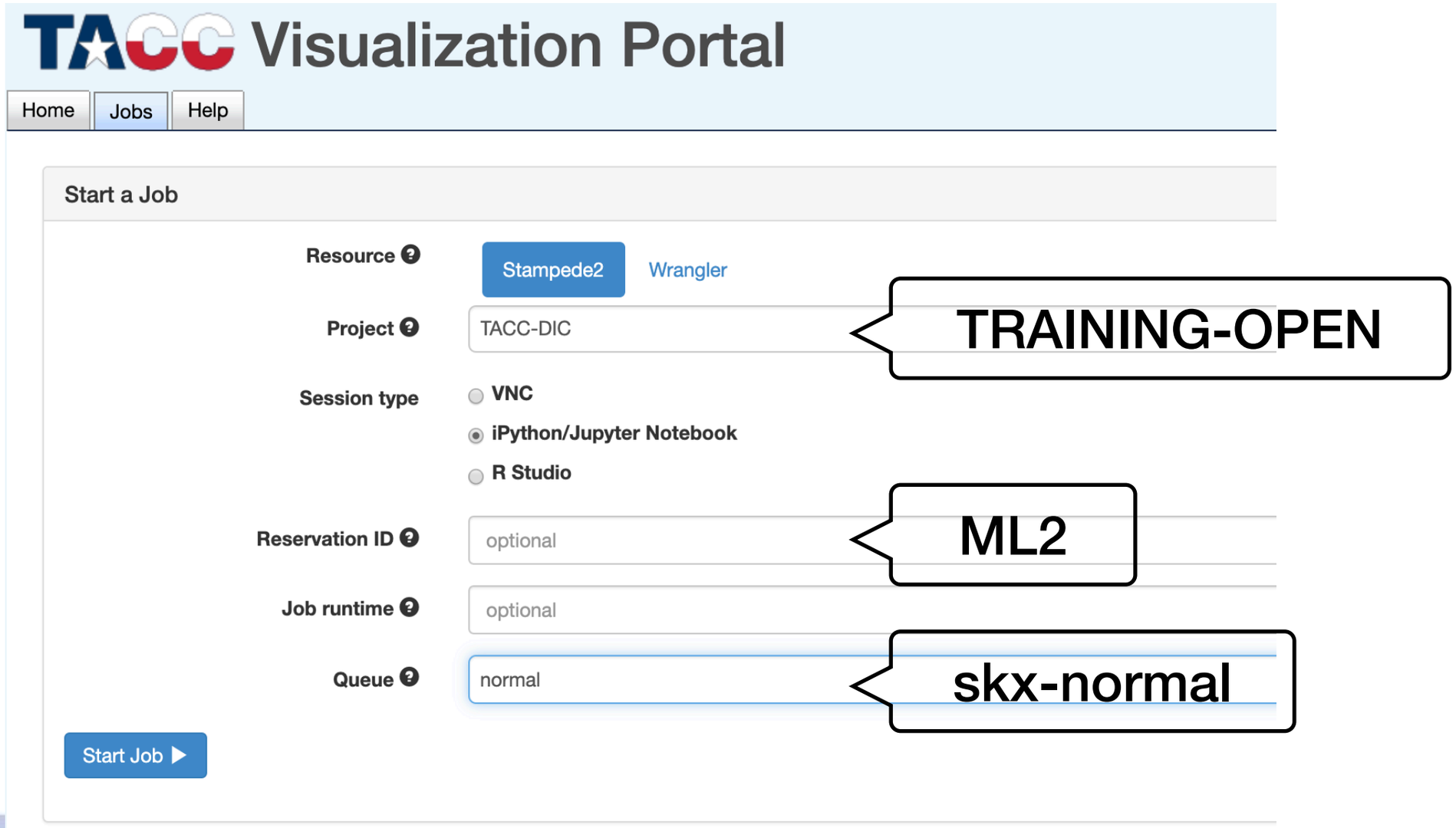
- Keras provides ability to utilize multi-gpus with slight code modifications.
- keras.utils.multi_gpu_model
- Workflow
  - Build model normally
  - Pass model template to multi_gpu_model -> parallel_model
  - Compile parallel_model
  - Run fit against parallel_model
  - Save weights using model.save_weights()

Multi-GPU Support



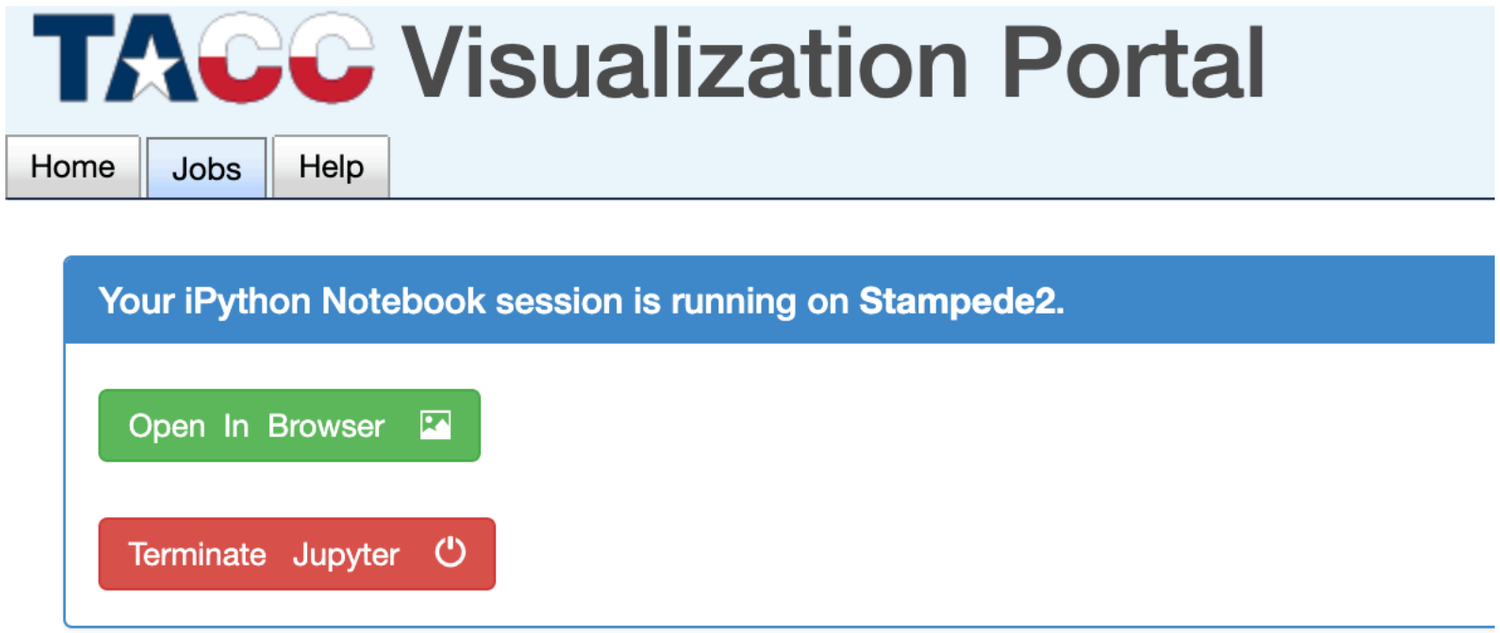
Hands-on Prep #1

- Jupiter Notebook via TACC VIS Portal
- Login https://vis.tacc.utexas.edu with your TACC account
-



Hands-on Prep #1

```
In [2]: ! cp -r /work/00946/zzhang/stampede2/ml-2019 $HOME/
```



Hands-on Prep #1

Hands-on Prep #1

- Open install_kera_tf.ipynb
  - Run every line
- Open runner.ipynb
  - Run every line

Hands-on Prep #1

- Open runner.ipynb
- Execute: %run boston_housing.py

Hands-on Prep #2

- Open runner.ipynb
- Execute: %run cifar10_keras.py