Introduction to Deep Learning at TACC

David Walling

Texas Advanced Computing Center
The University of Texas at Austin

TACC

Schedule

- Introduction to DL
- Introduction to Keras and TensorFlow
- DL and HPC at TACC

History

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

Image Classification

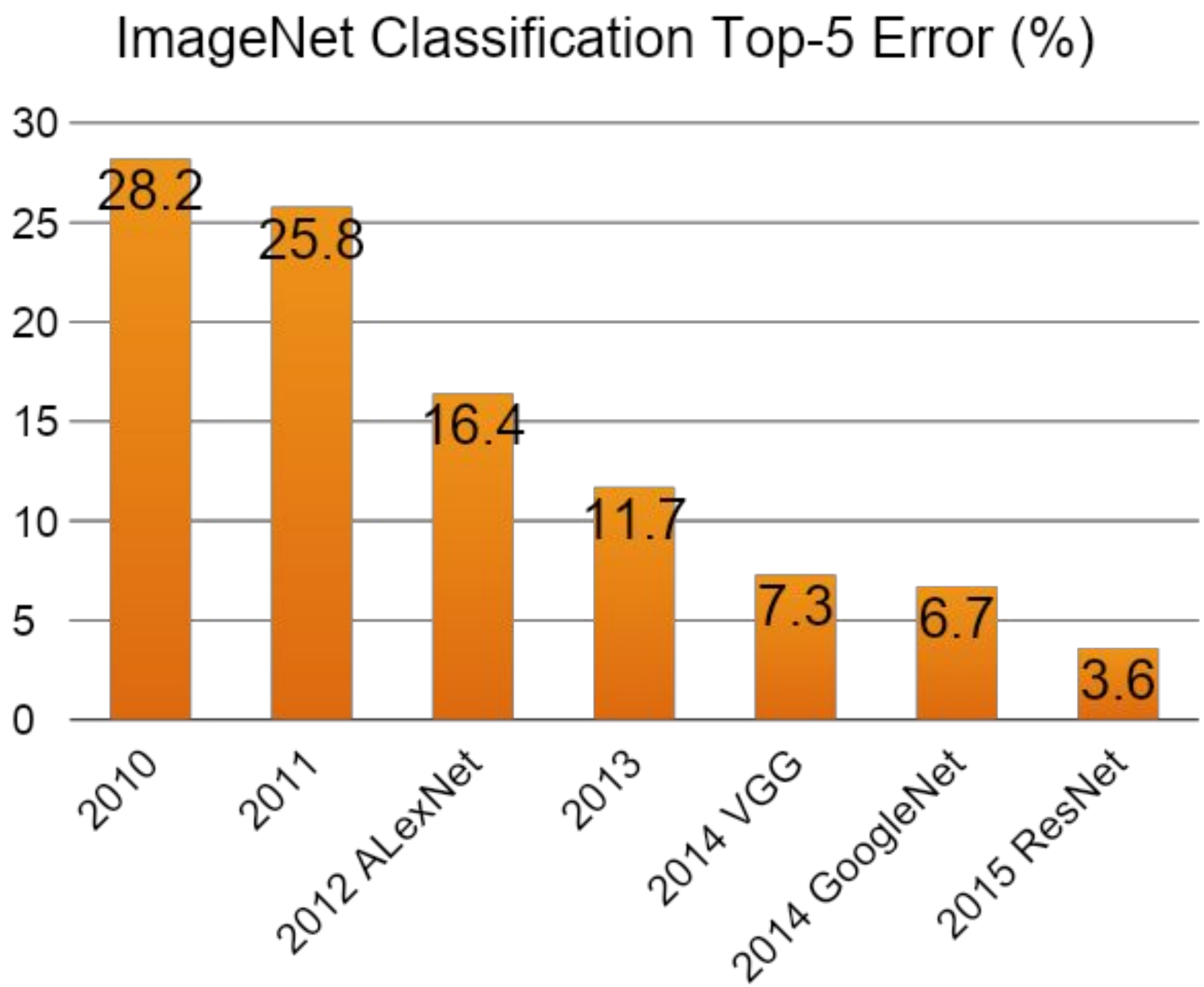
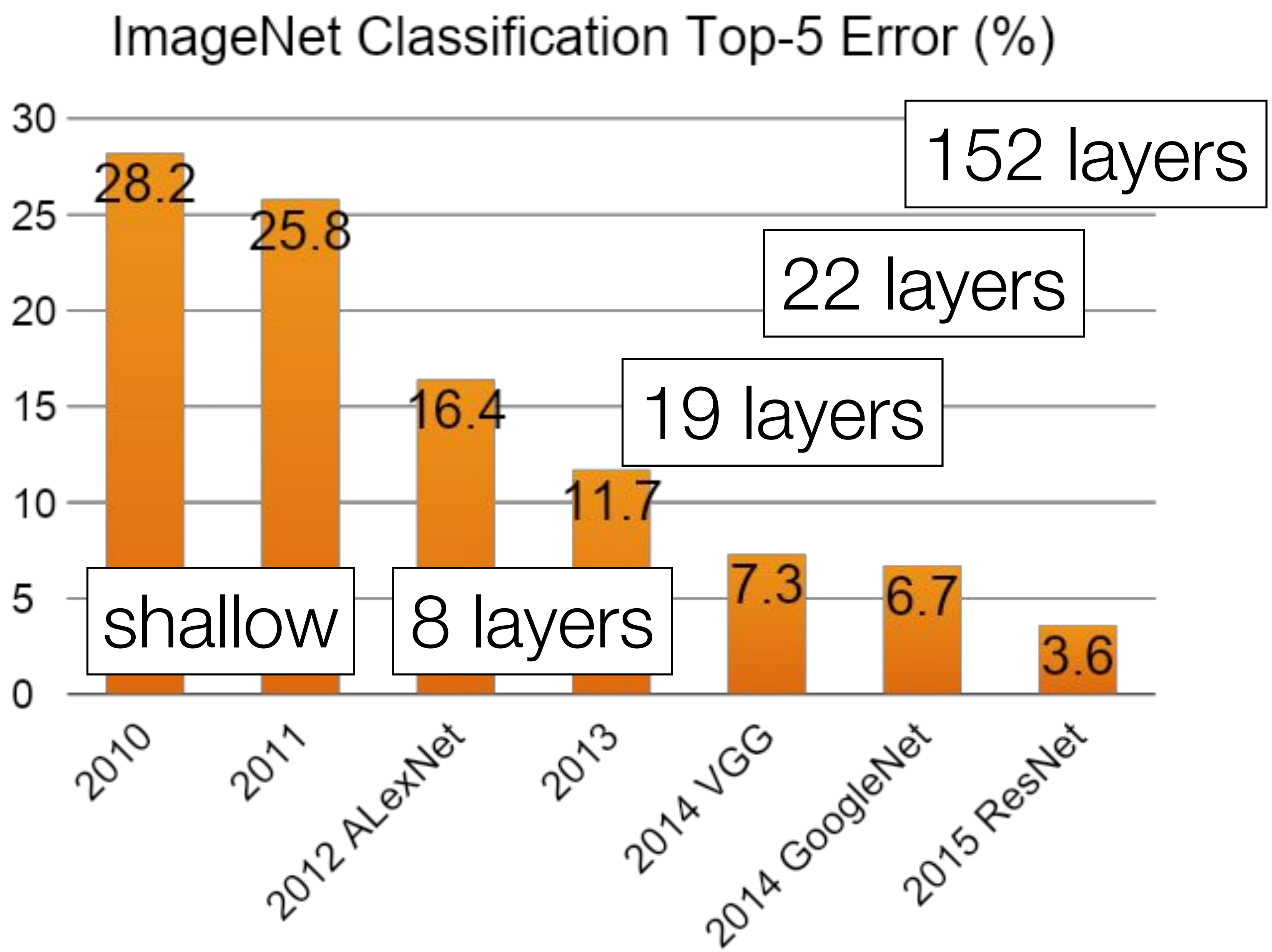
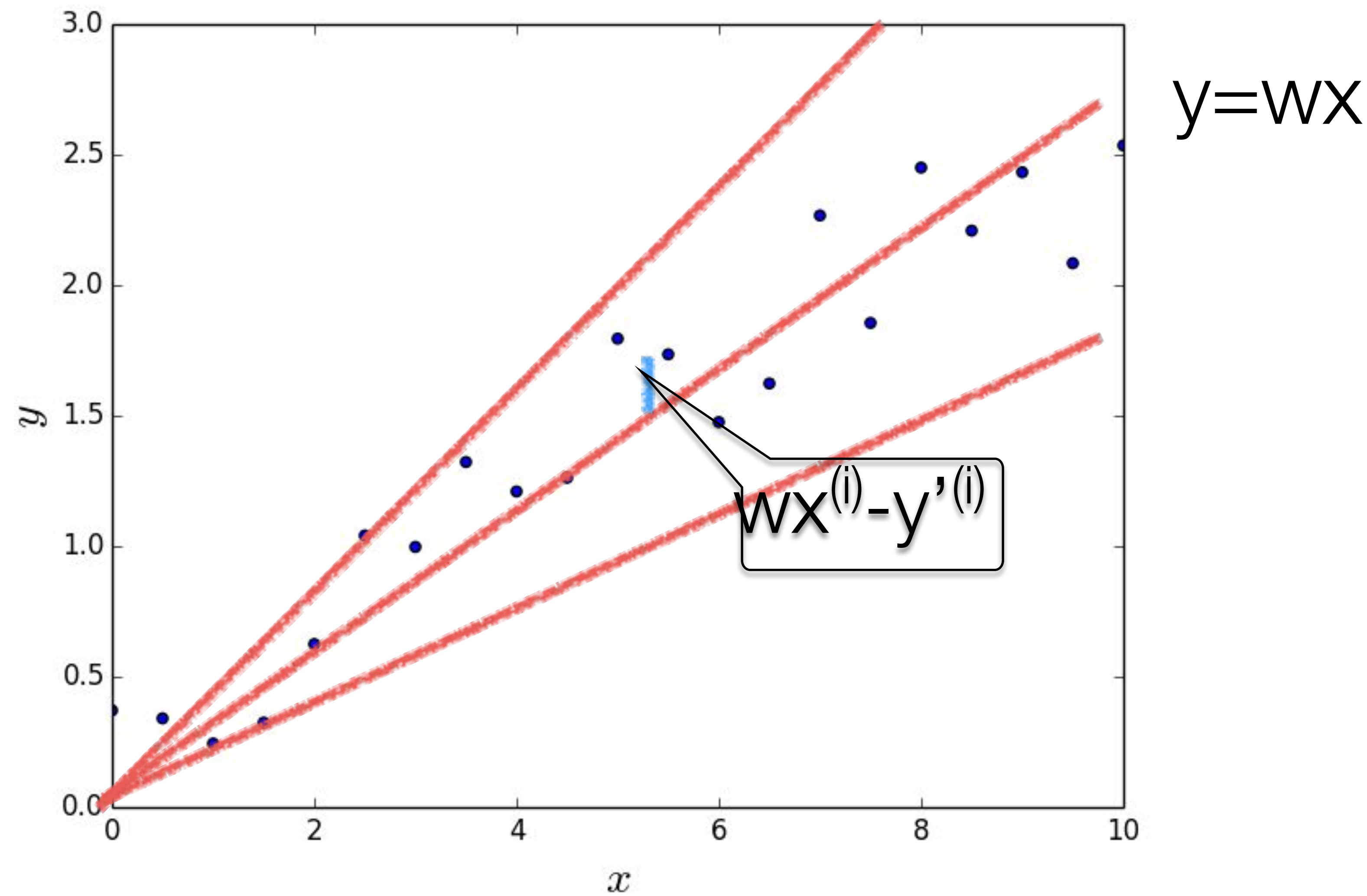


Image Classification



Linear Regression

- Example: Predicting house price with square footage



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

Model Generalization

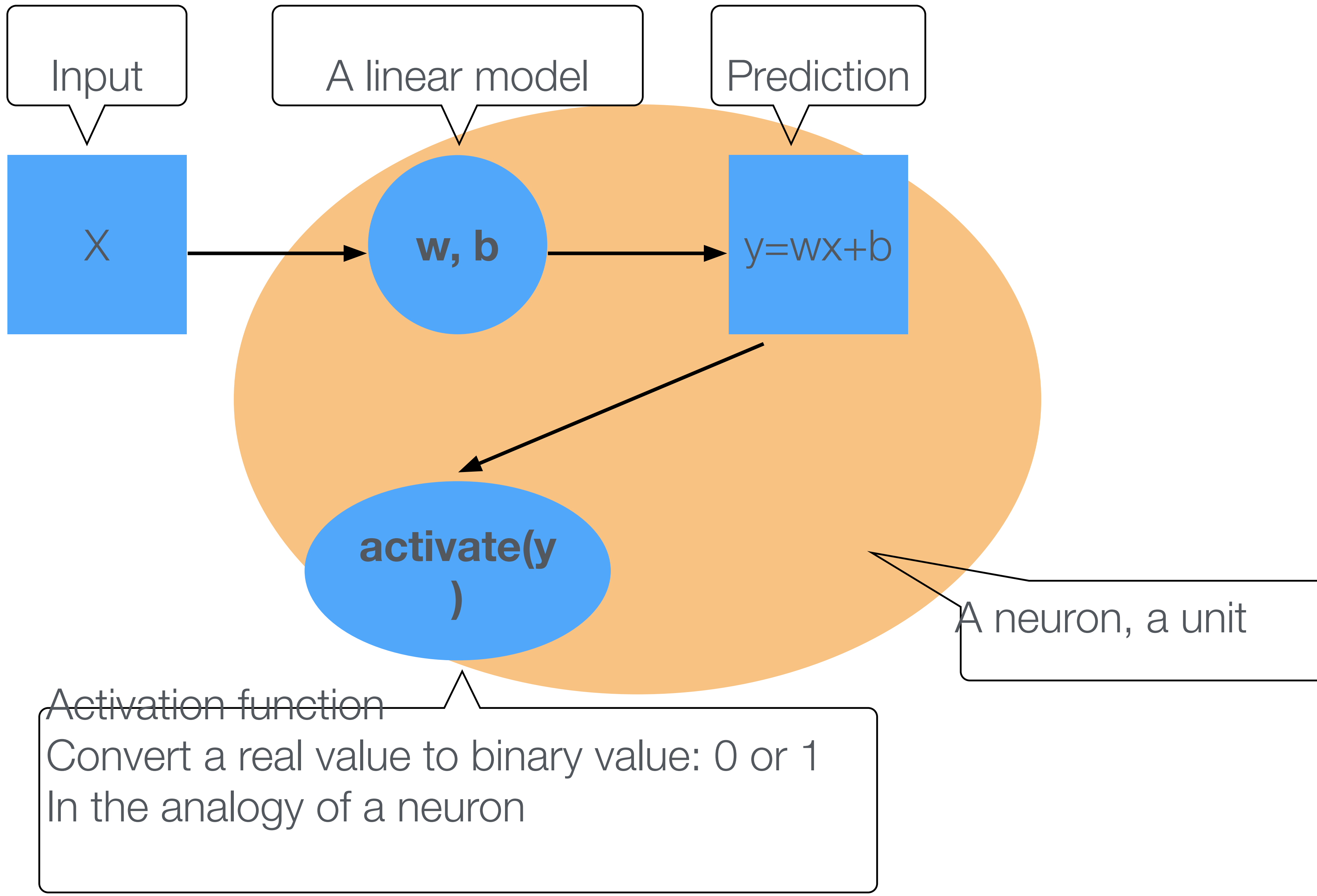
- In practice, we divide a labeled training dataset into two parts. E.g., 80% and 20%, referred as training and validation dataset, respectively
- We derive the value of w using the training dataset.
  - value of w can be referred as model
- Then we apply the model to the validation dataset and compare 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

TACC

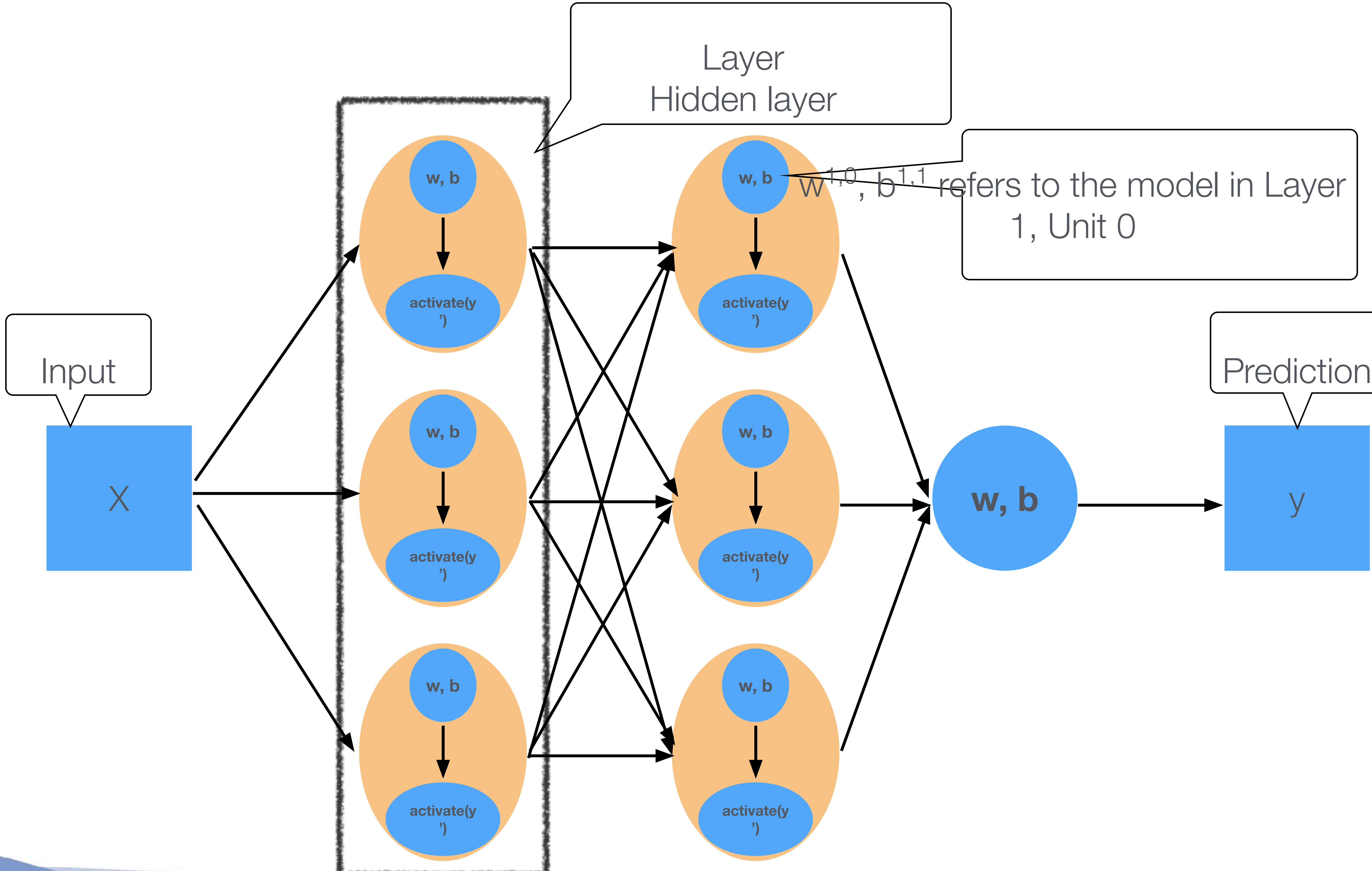
In practice

- We may 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 may use the gradient descent algorithm to find the w with minimum error
- We may use cross-entropy as error/loss instead of the distance

From Linear Regression to Neural Network

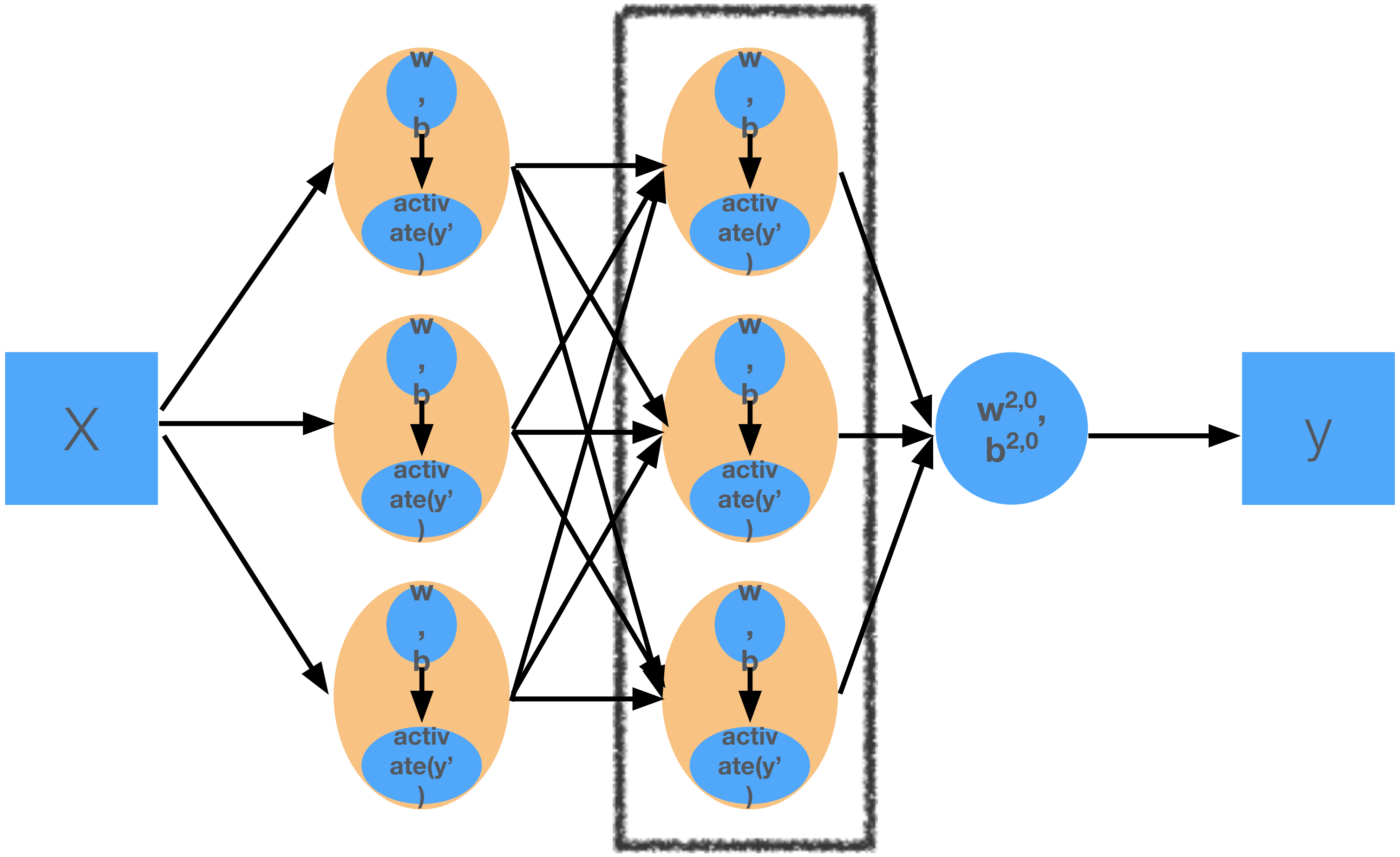


From Linear Regression to Neural Network



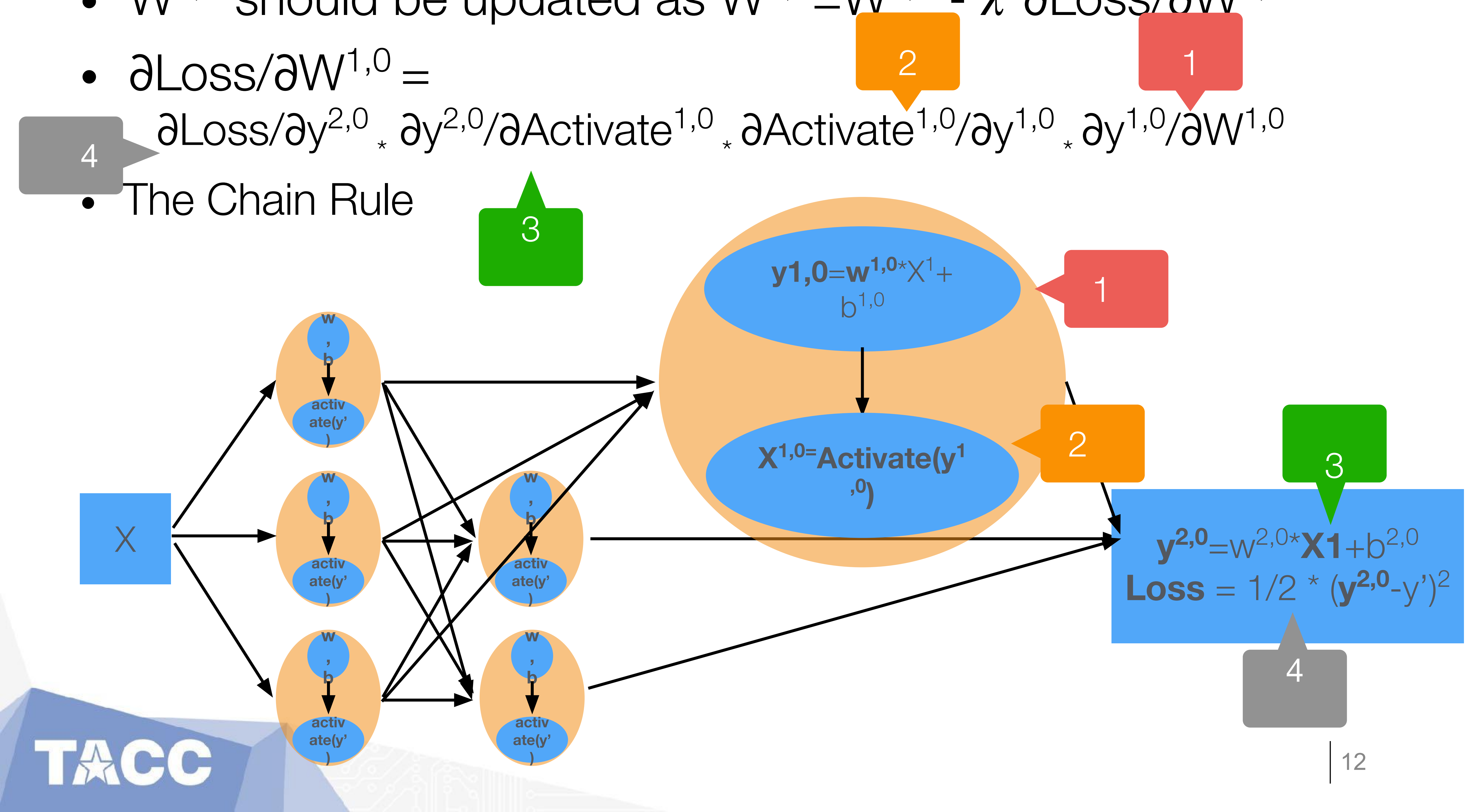
From Linear Regression to Neural Network

- 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 $w^{1,0}$, $w^{1,1}$, $w^{1,2}$?



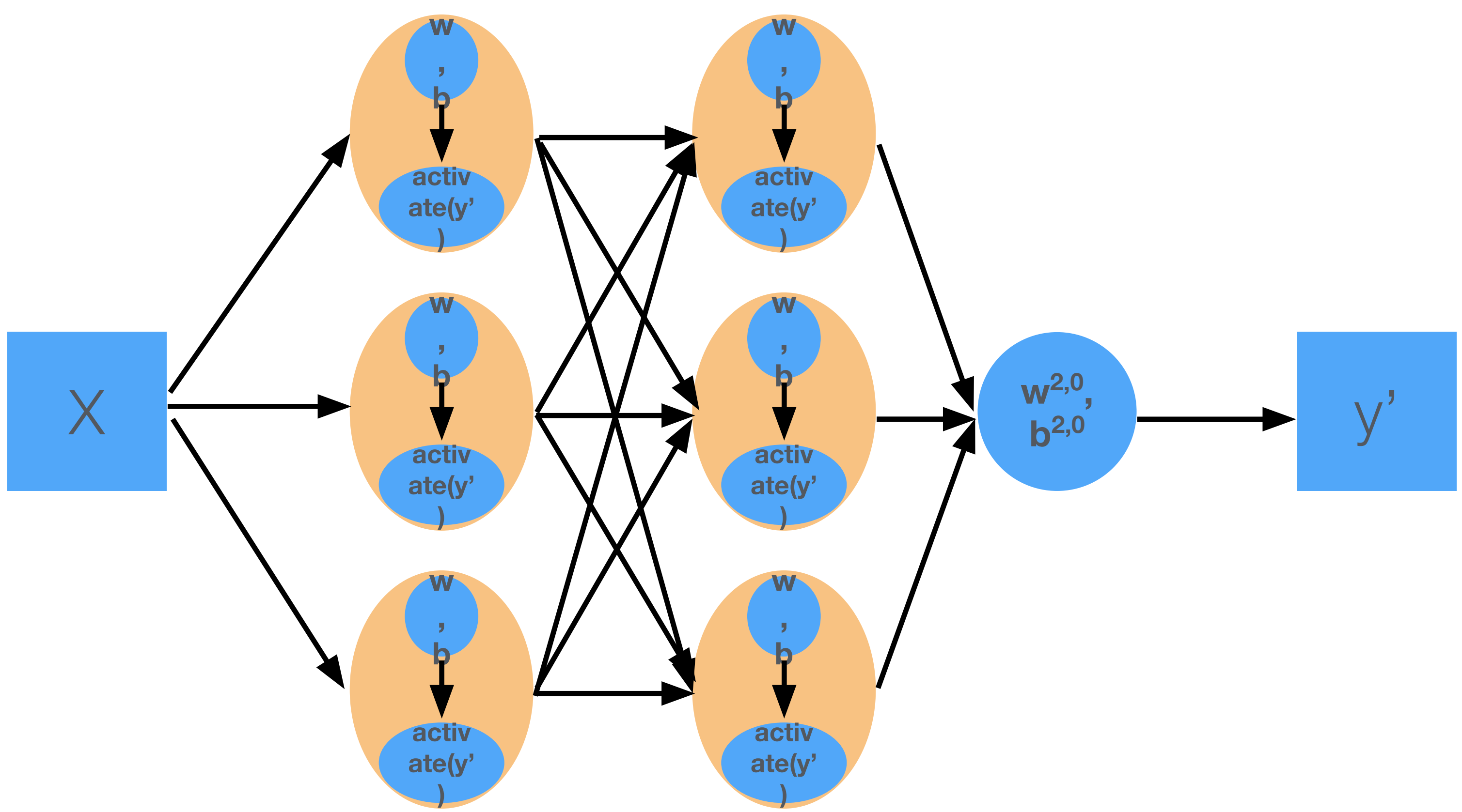
From Linear Regression to Neural Network

- 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
- So for each iteration, we take a small size of n (e.g., n=512), and update the parameters based on the averaged gradients

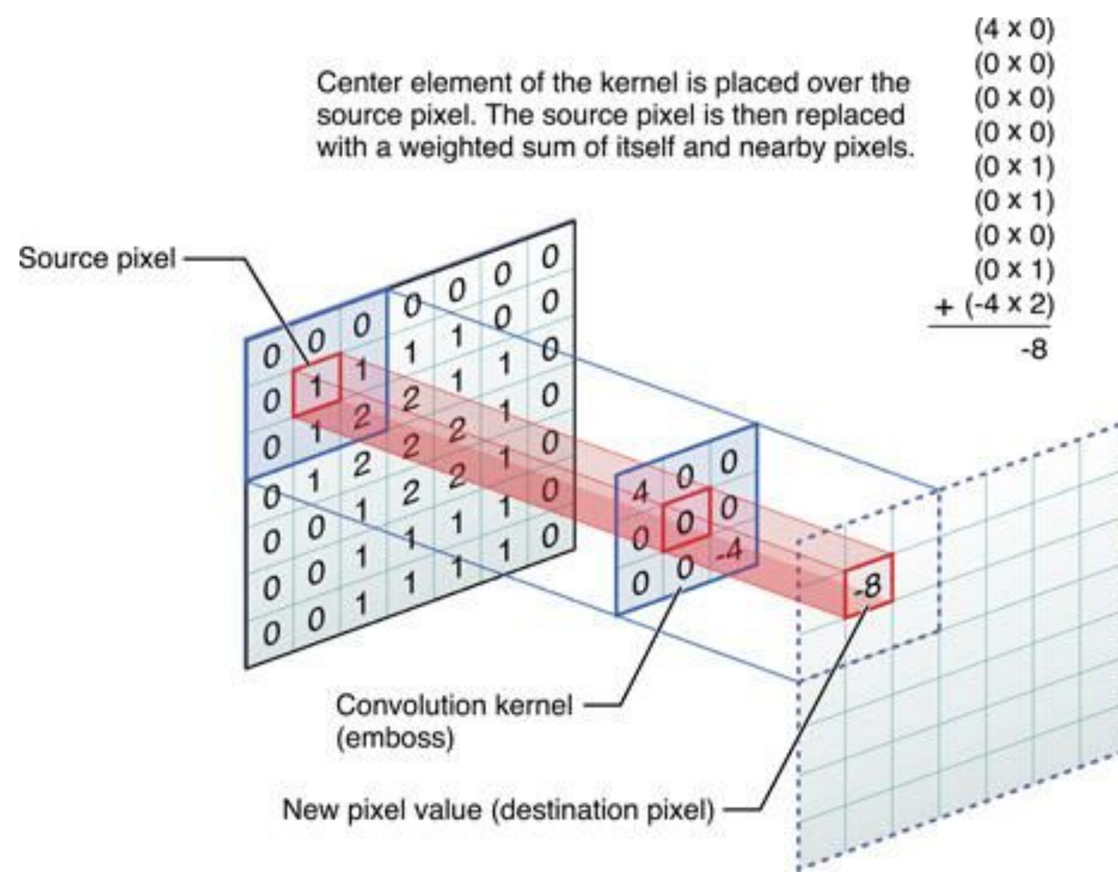


From Linear Regression to Neural Network

- The notion of Epoch
  - The time by which every training data item is visited once
  - So for 1,200,000 images with a 512 mini-batch size, an epoch roughly take 2,400 iterations
- How many epochs is enough?
  - Case by case
  - A somewhat standard practice uses 100 epochs for AlexNet and 90 epochs for ResNet-50
  - In practice, limited by ‘time’

Convolutional Neural Network

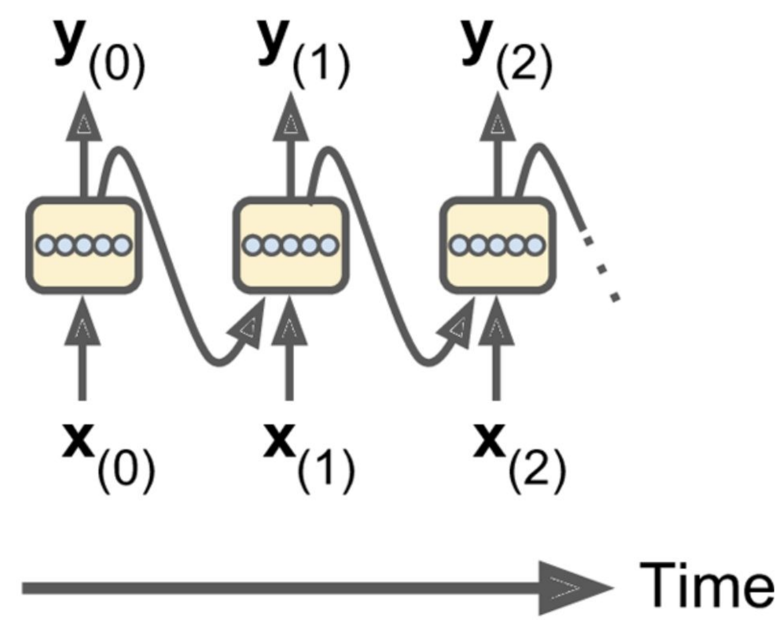
- What we just saw is a multi-layer perceptron (MLP) network
- If in any layer, there is a convolution operations, it is called convolutional neural network
- Often coupled with pooling operation
- Example applications:
  - Image classification
  - Object detection
  - Autonomous driving



https://ikhlestov.github.io/pages/machine-learning/convolutions-types/

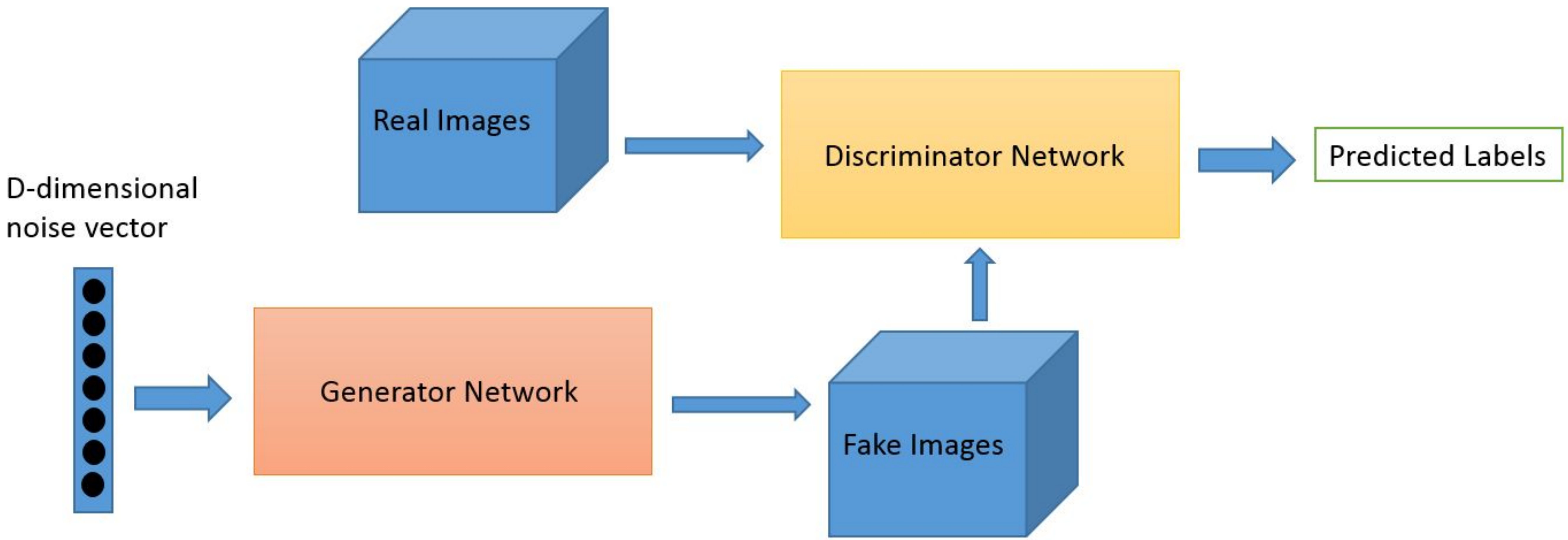
Recurrent Neural Network

- Recurrent 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



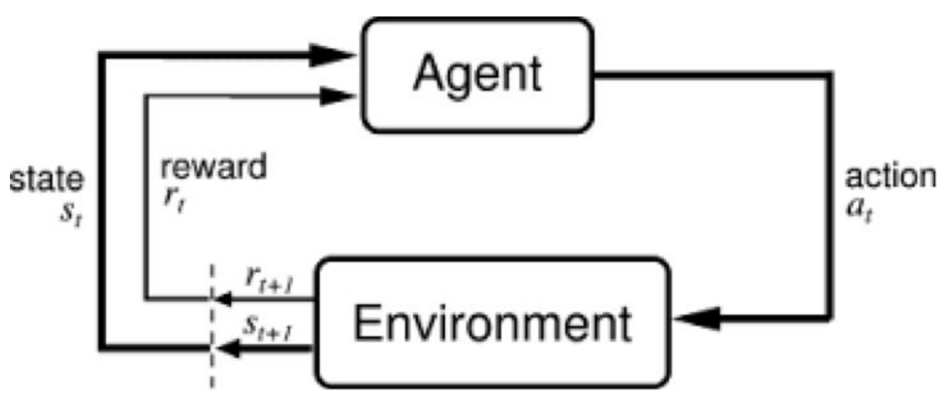
https://www.oreilly.com/library/view/neural-networks-and/9781492037354/ch04.html

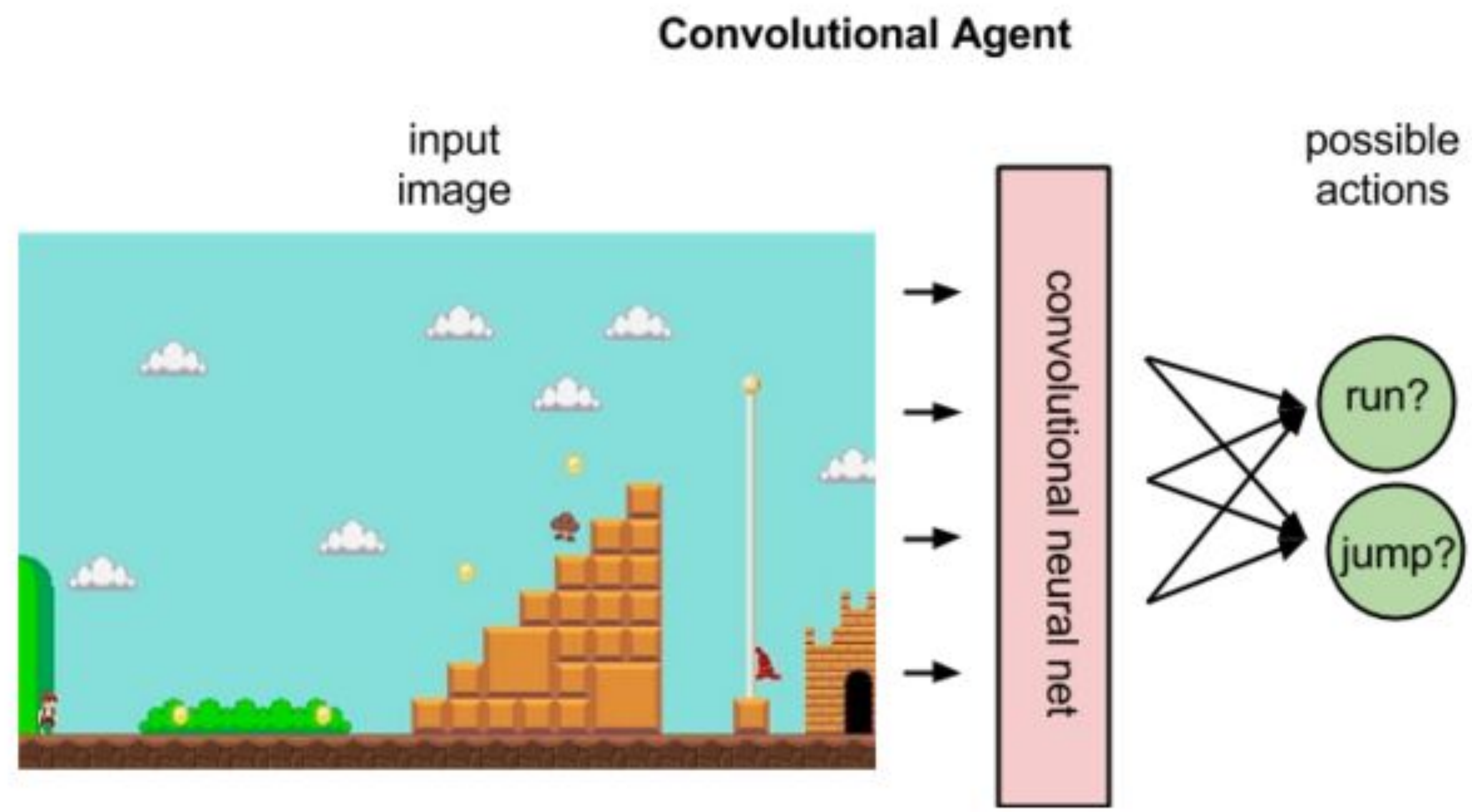
Generative Adversarial Network



Courtesy image from O'Reilly

Deep Reinforcement Learning





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

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

Schedule

- Introduction to DL
- Introduction to Keras and TensorFlow
- DL and HPC at TACC
- An DL Example in Natural Hazards
  - Damage classification from images with Deep Learning with Hurricane Harvey datasets

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 1.15.2 and 2.1.0 release Jan 2020

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

Keras Programming Interface

- Constructing Models — Sequential and Functional API
- Setup Input Stream — Data Generator API
- Instrumenting Training — Callback API
- Inference/Serving — Prediction API

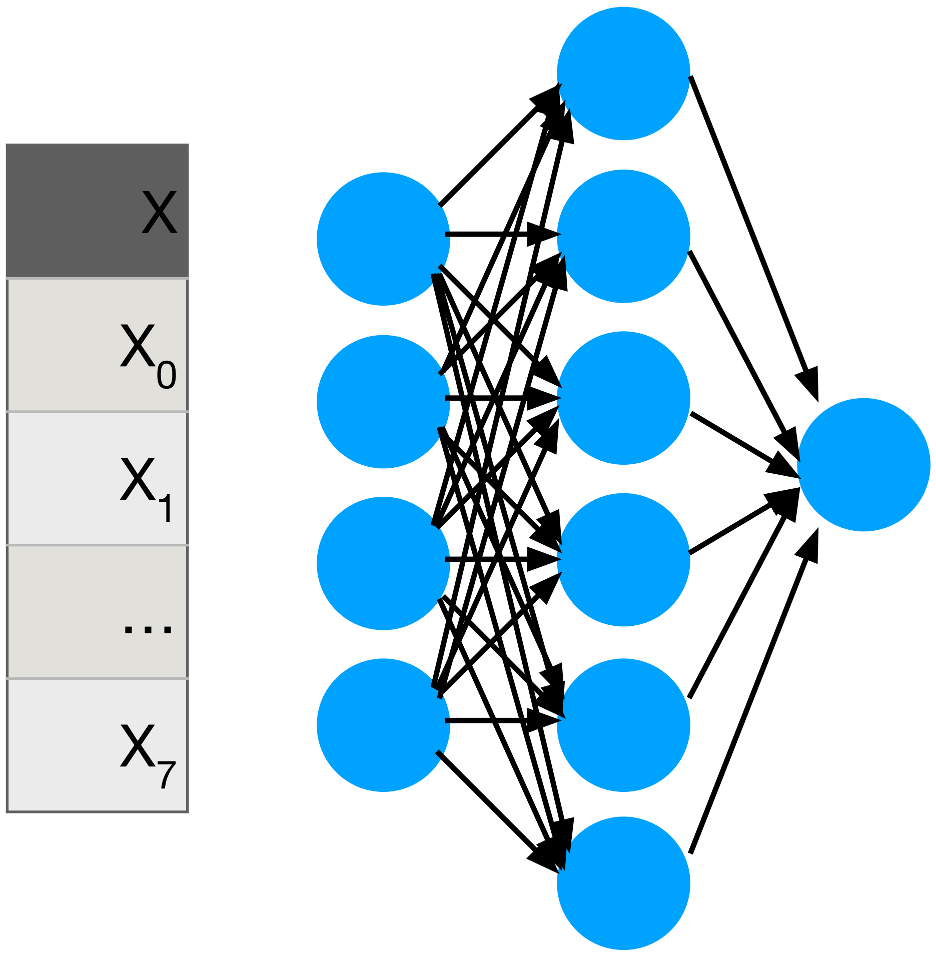
Constructing Models — Sequential API

- model = Sequential()

- model.add(Dense(4, input_dim=8, activation='relu'))

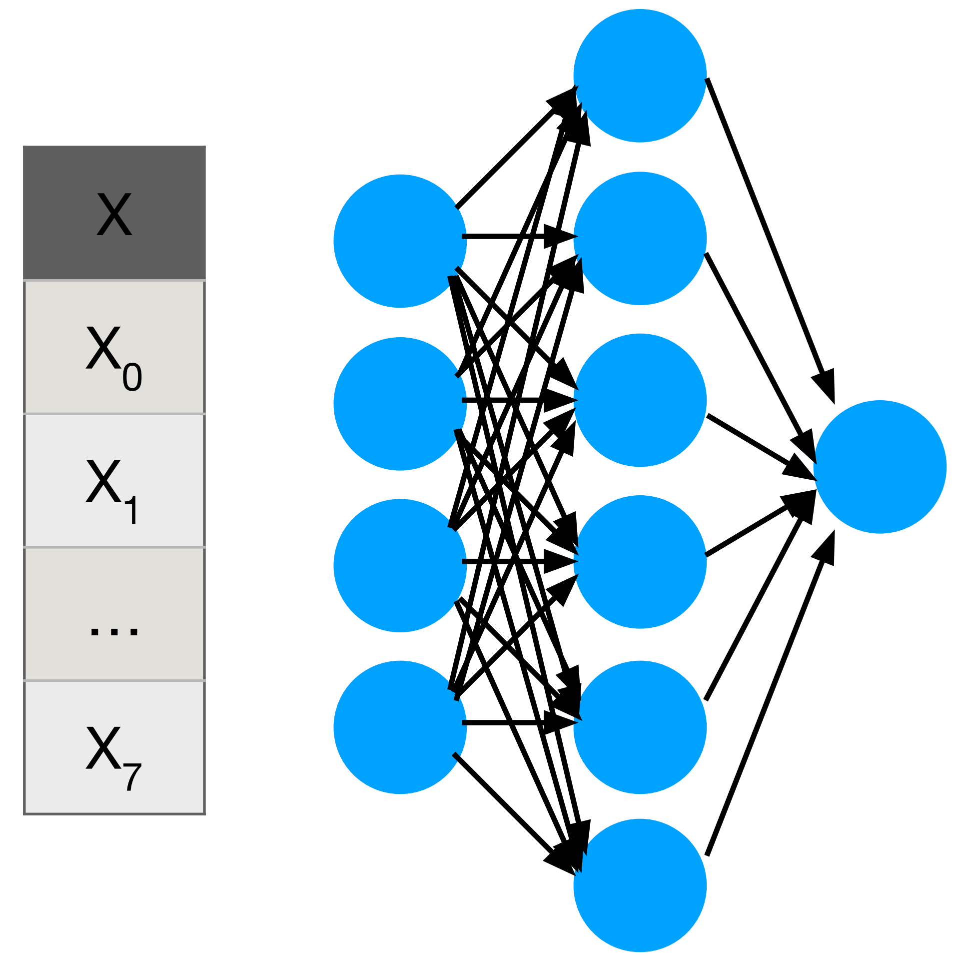
- model.add(Dense(6, activation=‘relu'))

- model.add(Dense(1, activation='sigmoid'))

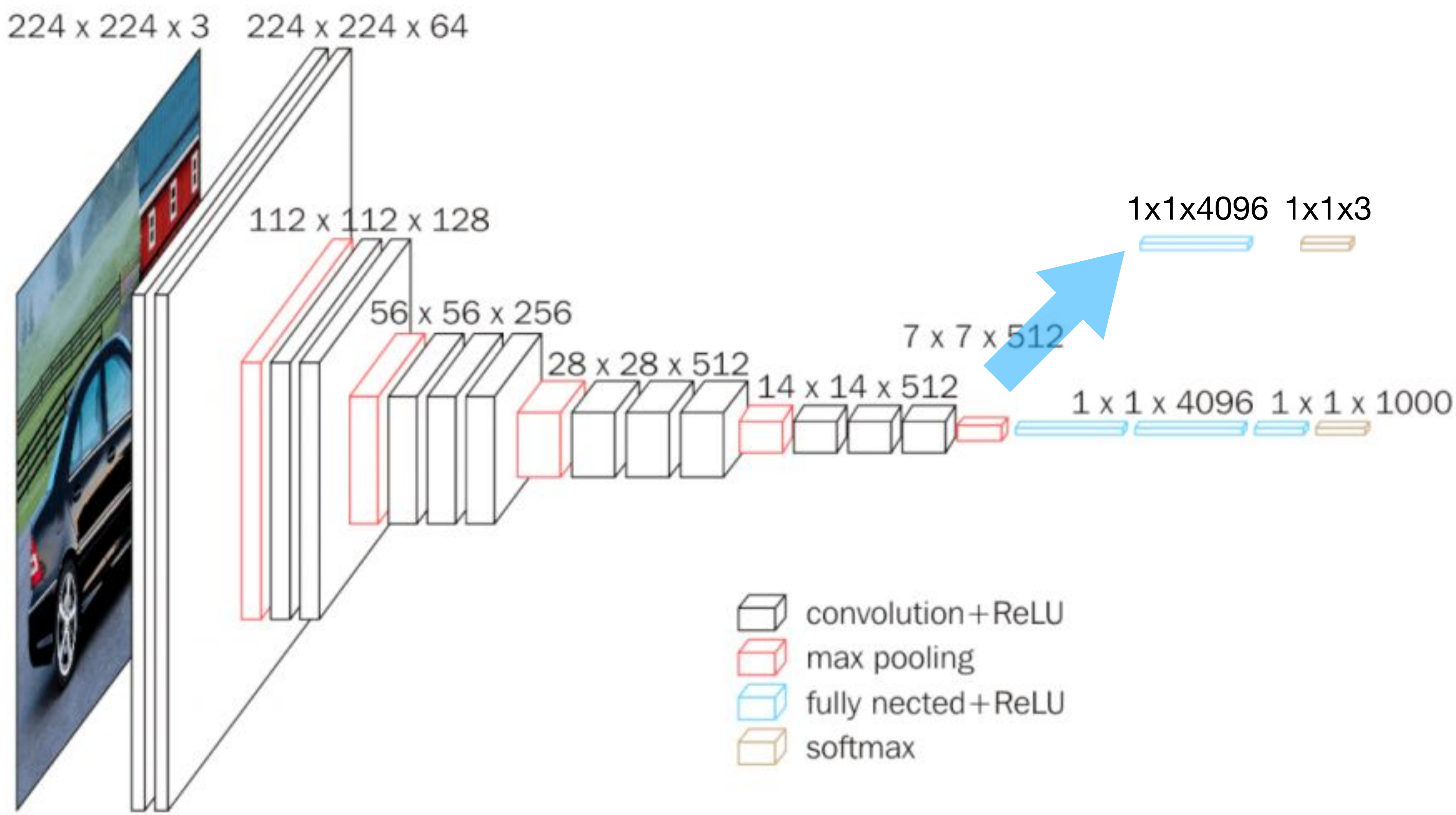


Constructing Models — Functional API

- inputs = Input(shape=(8,0))
- x = Dense(4, activation=‘relu')(inputs)
- x = Dense(8, activation=‘relu')(x)
- predictions = Dense(1, activation=‘sigmoid’)(x)
- model = Model(inputs=inputs, outputs=predictions)



Extend a Pre-trained Model



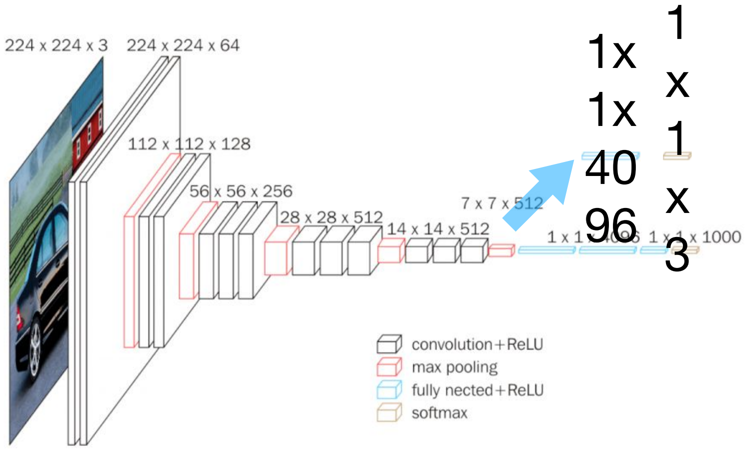
credits: https://neurohive.io/en/popular-networks/vgg16/

Loading a Pre-trained Model

- input_tensor = Input(shape=(224,224,3))
- vgg_model = VGG16(weights='imagenet', include_top=False, input_tensor=input_tensor)

- x = vgg_model.get_layer(‘block5_pool').output
- x = Flatten()(x)
- x = Dense(4,096, activation=‘relu')(x)
- x = Dense(3, activation=‘softmax')(x)

- model = Model(input=vgg_model.input, output=x)



Data Generator API

- A natural way to feed training data to models is to
  - Placing training items in file system, with each category in one directory
  - Organizing the validation data the same way

```
├── Train
│   ├── C0
│   │   ├── 001c4ec9-a5a3-4ef8-8154-876d3f54a7eb.jpg
│   │   ├── 0073a751-e33f-4748-b0e0-12ab9306fe8d.jpg
│   │   ├── …
│   └── C4
│       ├── 000f1691-619b-4d89-849c-33e416dff150.jpg
│       ├── 0056732e-9b52-4b7e-ac67-e41a05f37116.jpg
│       ├── …
```

Data Generator API

- datagen = ImageDataGenerator()

- train_it = datagen.flow_from_directory('Dataset_binary/Train/', target_size=(224,224), class_mode='categorical', batch_size=16, shuffle=True)

- val_it = datagen.flow_from_directory('Dataset_binary/Validation/', target_size=(224,224), class_mode='categorical', batch_size=1, shuffle=False)

Data Augmentation

- datagen = ImageDataGenerator(
- rotation_range=40,
- width_shift_range=0.2,
- height_shift_range=0.2,
- shear_range=0.2,
- zoom_range=0.2,
- horizontal_flip=True,
- fill_mode=‘nearest'
-)

TACC

Configuring a Model

- model.compile(loss='categorical_crossentropy',
- optimizer=opt,
- metrics=['accuracy'])
- print(model.summary())

Configuring a Model

Layer (type)	Output	Shap e			Param
input_1 (InputLayer)	(None,	224,	224,	3)	0
block1_conv1 (Conv2D)	(None,	224,	224,	64)	1792
block1_conv2 (Conv2D)	(None,	224,	224,	64)	36928
block1_pool (MaxPooling2D)	(None,	112,	112,	64)	0
block2_conv1 (Conv2D)	(None,	112,	112,	128)	73856
block2_conv2 (Conv2D)	(None,	112,	112,	128)	147584
block2_pool (MaxPooling2D)	(None,	56, 5	6, 12	8)	0
block3_conv1 (Conv2D)	(None,	56, 5	6, 25	6)	295168
block3_conv2 (Conv2D)	(None,	56, 5	6, 25	6)	590080
block3_conv3 (Conv2D)	(None,	56, 5	6, 25	6)	590080
block3_pool (MaxPooling2D)	(None,	28, 2	8, 25	6)	0
block4_conv1 (Conv2D)	(None,	28, 2	8, 51	2)	1180160
block4_conv2 (Conv2D)	(None,	28, 2	8, 51	2)	2359808
block4_conv3 (Conv2D)	(None,	28, 2	8, 51	2)	2359808
block4_pool (MaxPooling2D)	(None,	14, 1	4, 51	2)	0
block5_conv1 (Conv2D)	(None,	14, 1	4, 51	2)	2359808
block5_conv2 (Conv2D)	(None,	14, 1	4, 51	2)	2359808
block5_conv3 (Conv2D)	(None,	14, 1	4, 51	2)	2359808

Callbacks

- Callbacks let you instrument the training process
- Examples:
  - Checkpointing
  - ReduceLROnPlateau

Callbacks

- reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1, patience=5, min_lr=1e-8)

- filepath="model-{epoch:02d}-{val_accuracy:.2f}.hdf5"
- checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode=‘max')

TACC

Training

- model.fit_generator(train_it,
- steps_per_epoch=83,
- callbacks = [reduce_lr, checkpoint],
- validation_data=val_it,
- validation_steps=363,
- epochs=5)

Training

- Epoch 1/5
- 1/83 [..............................] - ETA: 11:42 - loss: 9.0094 - accuracy: 0.2500
- 2/83 [..............................] - ETA: 5:50 - loss: 8.9650 - accuracy: 0.2812
- 3/83 [>.............................] - ETA: 5:43 - loss: 8.8843 - accuracy: 0.2917
- 4/83 [>.............................] - ETA: 5:28 - loss: 9.3953 - accuracy: 0.2656
- 5/83 [>.............................] - ETA: 5:29 - loss: 8.7584 - accuracy: 0.3125
- …
- 80/83 [===========================>..] - ETA: 12s - loss: 7.5598 - accuracy: 0.3901
- 81/83 [============================>.] - ETA: 8s - loss: 7.5693 - accuracy: 0.3899
- 82/83 [============================>.] - ETA: 4s - loss: 7.5603 - accuracy: 0.3897
- 83/83 [==============================] - 436s 5s/step - loss: 7.5605 - accuracy: 0.3903 - val_loss: 0.8355 - val_accuracy: 0.5179

Inference/Serving

- l_model = load_model(“models/model-12-0.71.hdf5")
- img = image.load_img('Dataset_2/Validation/C4/8108cbbf-60ca-47d8-af13-2e3603a5c30e.jpg', target_size=(224,224))
- img = np.expand_dims(img, axis=0)
- y_pred = l_model.predict(img)
- print(np.argmax(y_pred))

Tuning — Model Structure

- Number of layers
- Unit count
- Variable initialization

Tuning — Hyperparameter

- Learning rate
- Momentum
- Penalty in logistic regression
- Loss in SGD

Schedule

- Introduction to DL
- Introduction to Keras and TensorFlow
- AI/ML/DL and HPC at TACC
- An ML/DL Example in Natural Hazards
  - Damage classification from images with Deep Learning with Hurricane Harvey datasets

Deep Learning at TACC

- Hardware
- Software
- Interface

AI Hardware at TACC

- In general, we support AI on every platform.
- For this purpose, we will focus mostly on GPUs
  - Frontera
  - Longhorn
  - Maverick
  - Chameleon

Frontera Single Precision Subsystem

- Frontera is the #5 supercomputer in the world, with more than 450,000 processors achieving 40 PetaFlops at double precision.
- It also has a smaller subsystem optimized for System Features:
  - 90 nodes/360 GPUs
  - 2x Broadwell processors
  - 128 GB RAM
  - 4x NVIDIA Turing Quadro RTX 5000 GPUs per node
  - 150 GB local SSD
- Infiniband connected to Frontera main filesystems (50 Petabytes).

Software Support for Deep Learning

- While you can produce custom code for about any method, most of what you need is easiest to get too from common frameworks.
  - For Deep Learning, PyTorch, Keras, TensorFlow
  - Many ML methods in data science frameworks like Pandas
- The typical language of choice for these methods is Python — you don't really need to know Python for today's exercises.
- The best way to work interactively in Python is through Jupyter notebooks.

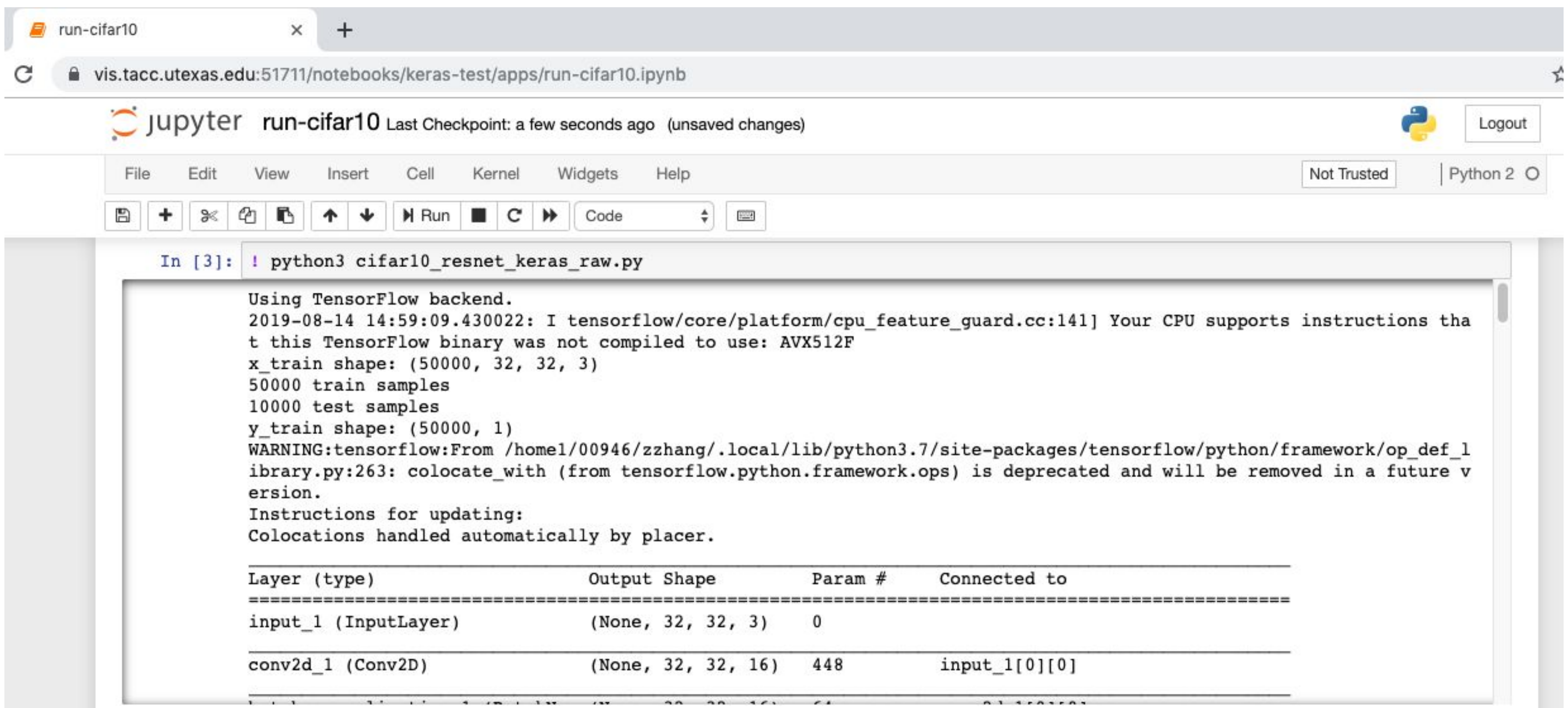
Software

	Frontera (CPU)	Frontera (GPU)	Longhorn (GPU)
Keras/TensorFlow/ Horovod	✔	✔	✔
PyTorch/Horovod	✔	✔	✔
MXNet/Horovod		✔	✔
Caffe/Intel MLSL	✔		

TACC

Front-end

- Command Line
- Jupyter Notebook



Schedule

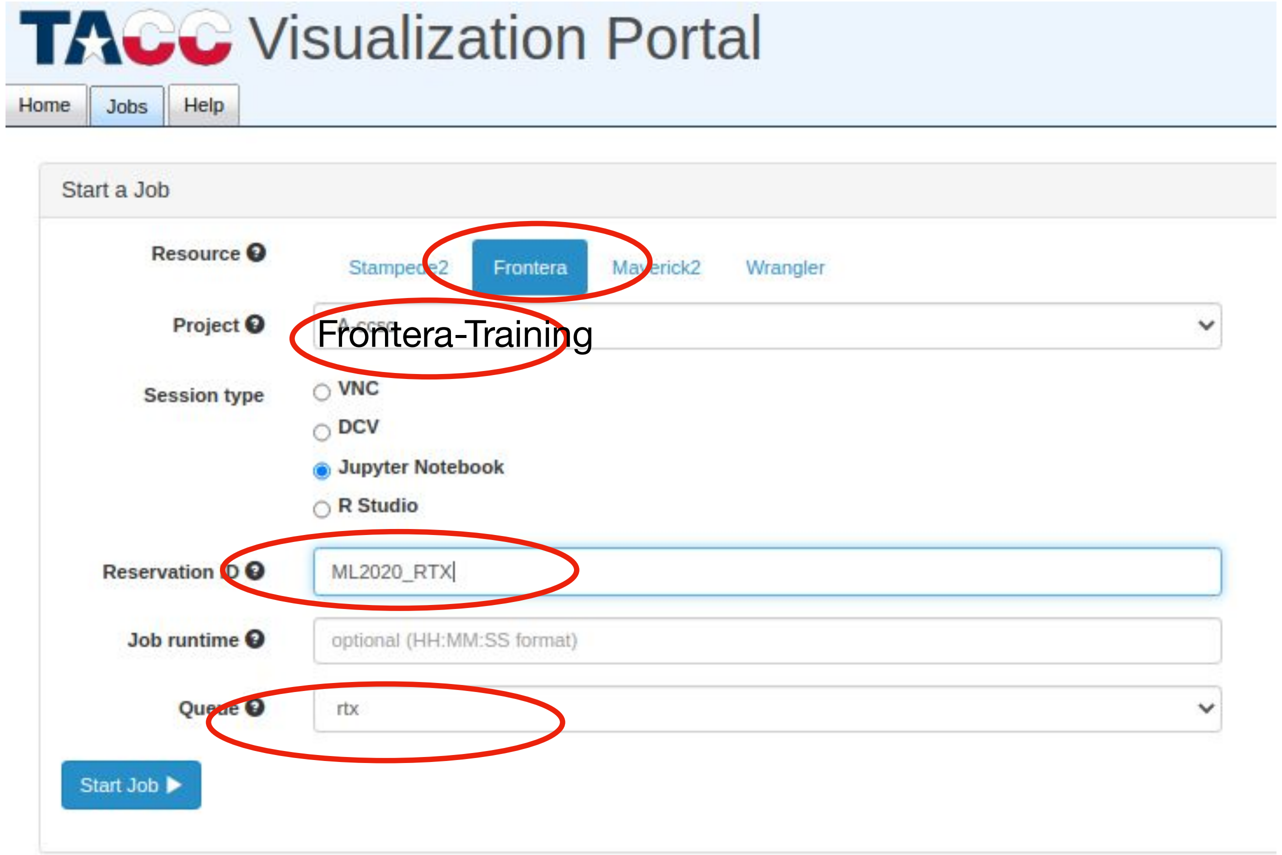
- Introduction to DL
- Introduction to Keras and TensorFlow
- DL and HPC at TACC
- An ML/DL Example in Natural Hazards
  - Damage classification from images with Deep Learning with Hurricane Harvey datasets

Today's examples

- Deep Learning:
  - A Jupyter notebook via TACC's vis portal
  - on a Fronter RTX node,
  - using Keras over TensorFlow (GPU) to
  - build, train and infer with a CNN.
- Data sets from Hurricane Harvey reconnaissance.
- About GPU Nodes (usually the best choice)
  - TACC Systems with GPUs
    - Frontera, Longhorn, Maverick, Chameleon
  - Also possible use Stampede2 (CPU)

Starting Jupyter

- TACC Visualization Portal:
  - Go to https://vis.tacc.utexas.edu
  - Login with your training account credentials
  - Reservation ID: ML_Institute_day4



Basic Setup

- Launch Jupyter Notebook on Frontera RTX reservation via vis portal
- Open Terminal
  - *cd $SCRATCH*
  - *cp -rf   /scratch1/00157/walling/ml-2021/dl_tutorial   ./*
  - *cd $HOME*
  - *ln -s $SCRATCH/dl_tutorial   ./dl_tutorial*
- Run install_tf_keras.ipynb

TACC

Exercise 1

- Open train-1st.ipynb
- Run through the cells
- Train for the 1st time
- Tasks:
  1. Monitor val_accuracy change along epochs
  2. Monitor val_accuracy vs. train_accuracy

TACC

Exercise 2

- Open train-2nd.ipynb
- Run through the cells
- Train for the 2nd time
- Tasks:
  1. Pay attention to the data augmentation code
  2. Monitor val_accuracy vs. train_accuracy and check if overfitting exists

Exercise 3

- Open train-3rd.ipynb
- Run through the cells
- Train for the 3rd time
- Tasks:
  1. Pay attention to label smoothing in the loss function
  2. Pay attention to the learning rate reducer
  3. Monitor val_accuracy change along epochs

TACC

Exercise 4

- Open infer.ipynb
- Run through the cells
- Visualize selected image then predict using the trained-model
- Tasks:
    1. See if predictions match labels
    2. Randomly choose images and run predictions

Questions?

- Contact Information
  - David Walling (walling@tacc.utexas.edu)
  - Zhao Zhang (zzhang@tacc.utexas.edu)
  - Weijia Xu (xwj@tacc.utexas.edu)

TACC