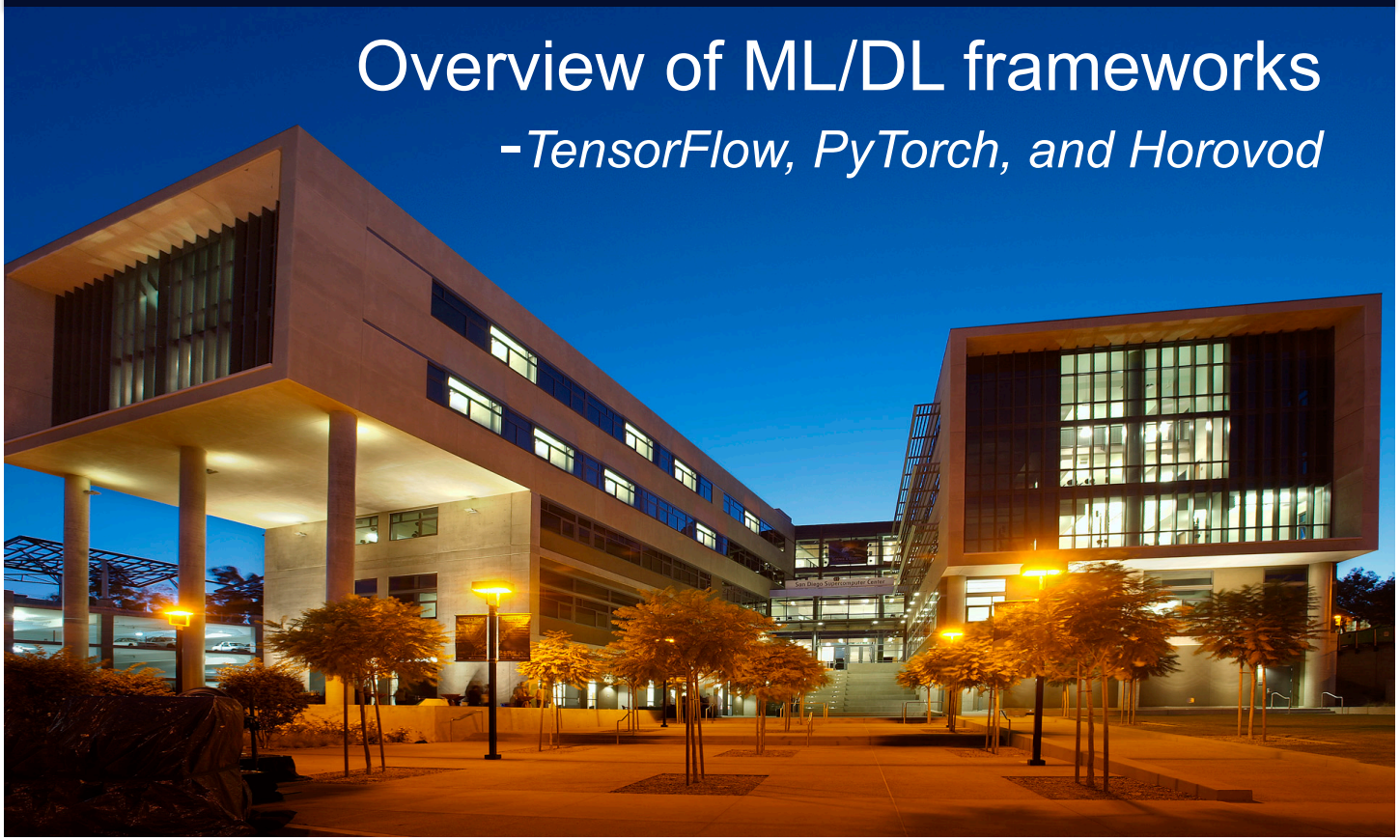
Overview of ML/DL frameworks

-*TensorFlow, PyTorch, and Horovod*



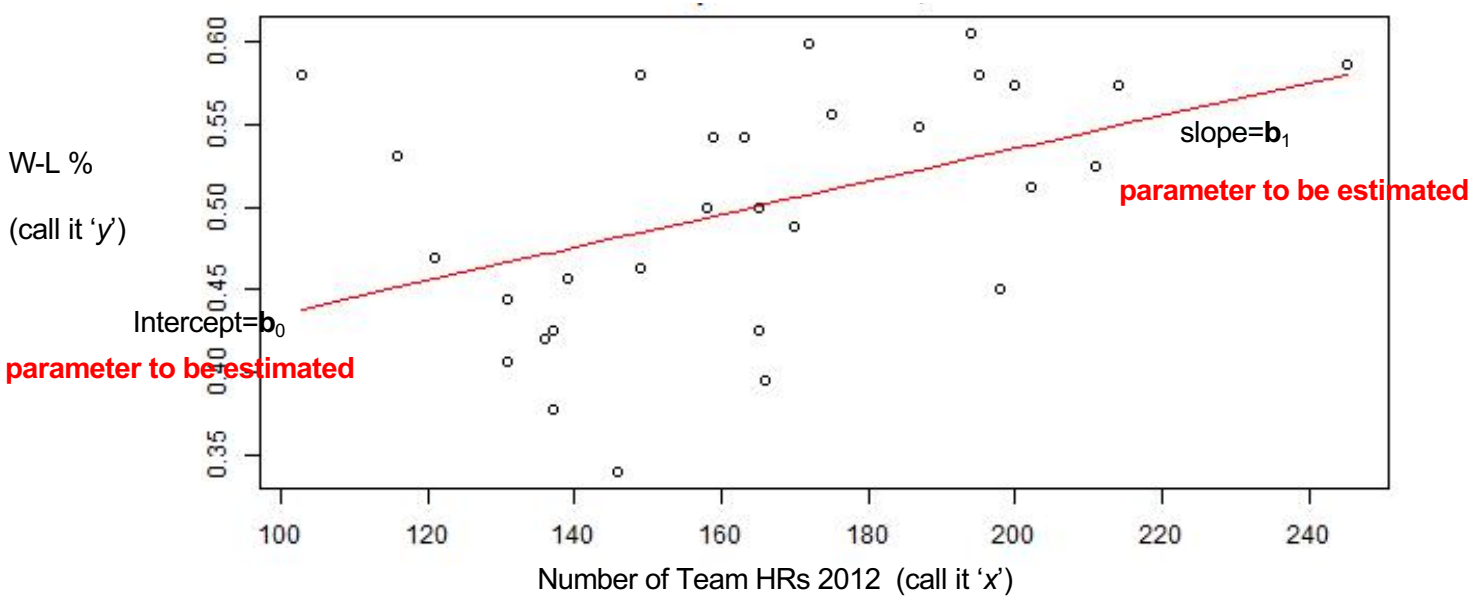
Overview

- **Quick Deep Learning intro – leveraging Summer Institute talk from Paul Rodriguez**
- **TensorFlow**
  - Hands on MNIST example
  - Running on Comet via Singularity containers
- **PyTorch**
  - Comet Example
- **Horovod**
  - Distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.

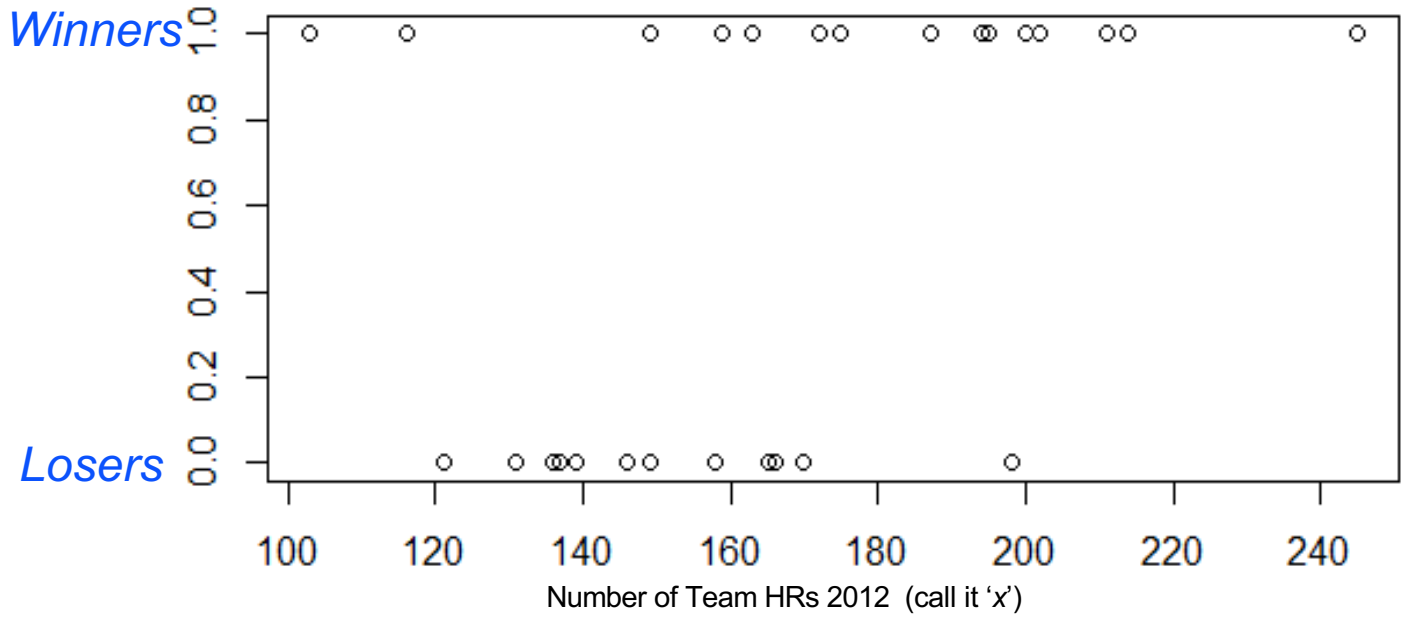
Quick Deep Learning
Paul Rodriguez SDSC

Linear Regression is Fitting a Line

the Model: $y = f(x, b) = bo * 1 + b_1 * x$

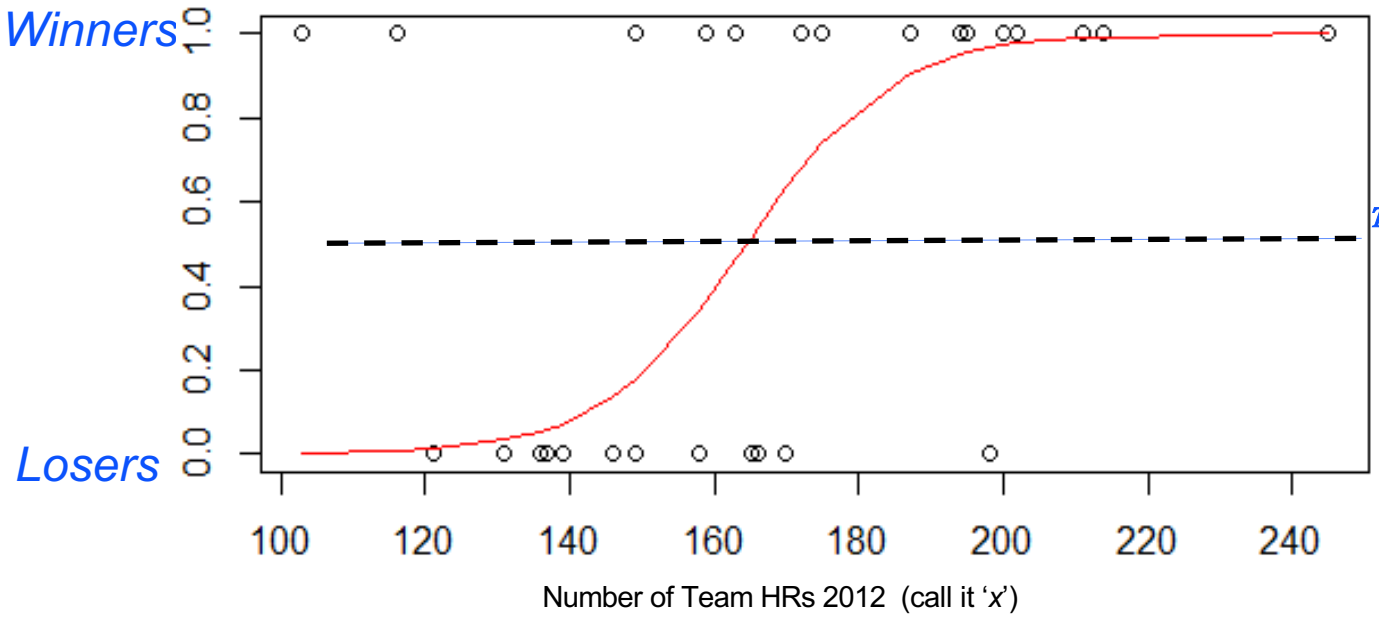


Classification uses labelled outcomes



Can do better: fit a nonlinear function

the Model: $y = f(x, b) = 1/(1 + \exp[-(b_o * 1 + b_1 * x)]$

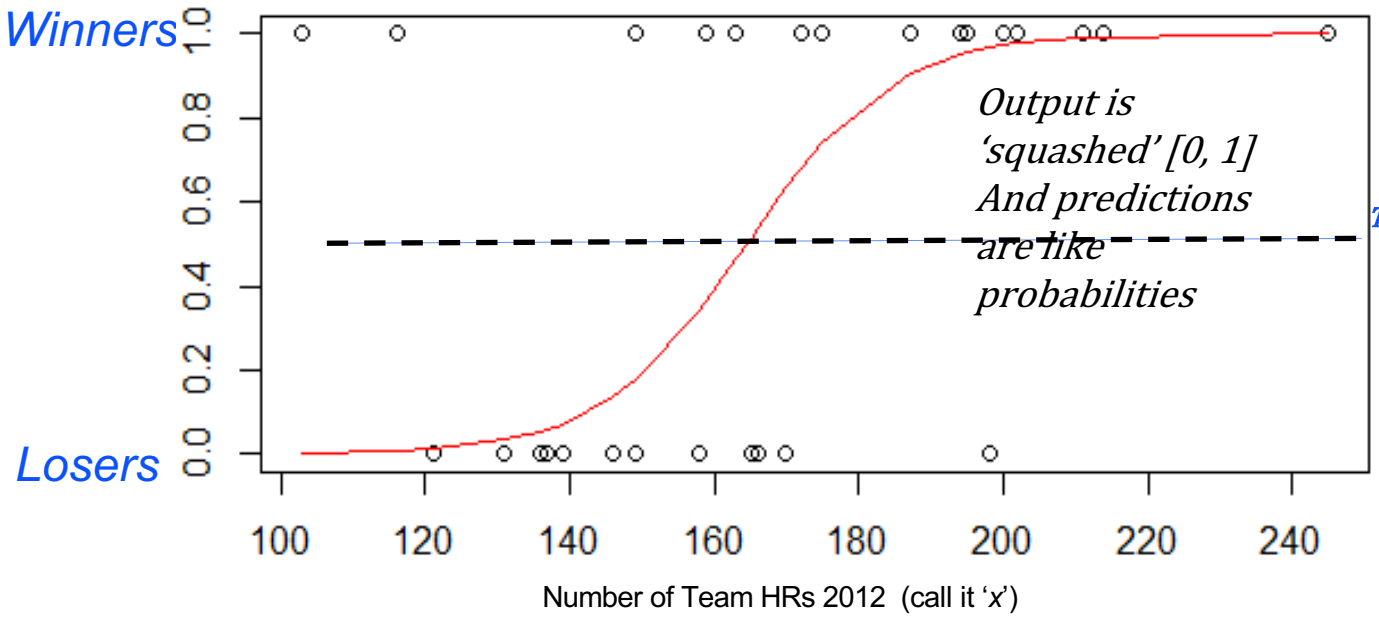


Use threshold to label prediction:

for $y > 0.5$, classify x as Winner

Can do better: fit a nonlinear function

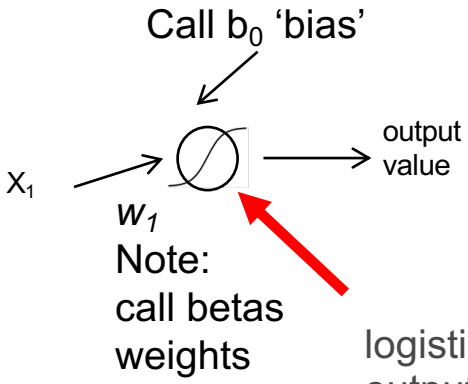
the Model: $y = f(x, b) = 1/(1 + \exp[-(b_o * 1 + b_1 * x)]$



Use threshold to label prediction:

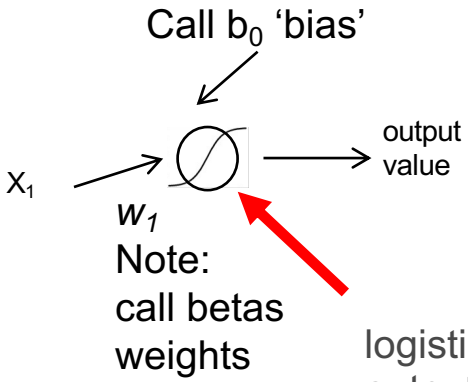
for $y > 0.5$, classify x as Winner

Logistic Regression as 1 node network



logistic function to transform input to output – call it the 'activation' function

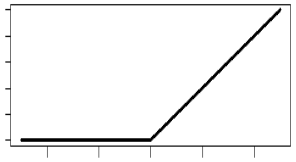
Logistic Regression as 1 node network



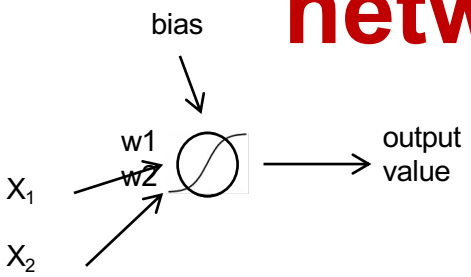
logistic function to transform input to output – call it the 'activation' function

Note: other activations are possible,

RELU (rectified linear unit)



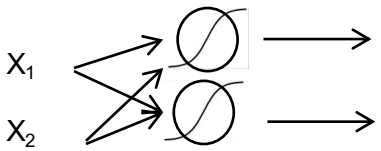
Next step: More general networks



Add input variables

More general networks

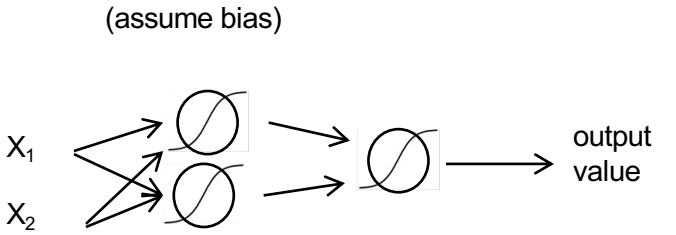
(assume bias present)



Add input variables

Add logistic transformations …

More general networks

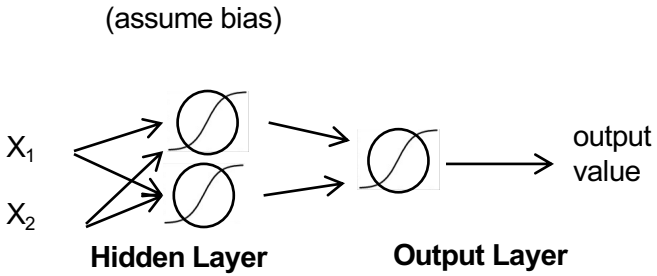


Combine transformations!

Add input variables

Add logistic transformations …

More general networks



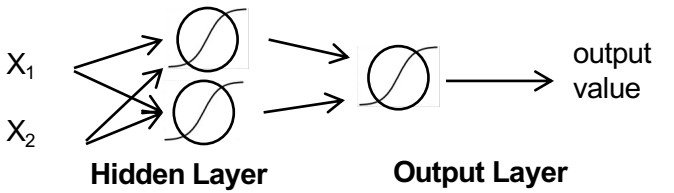
Combine transformations!

Add input variables

Add logistic transformations …

But parameter fitting is harder too

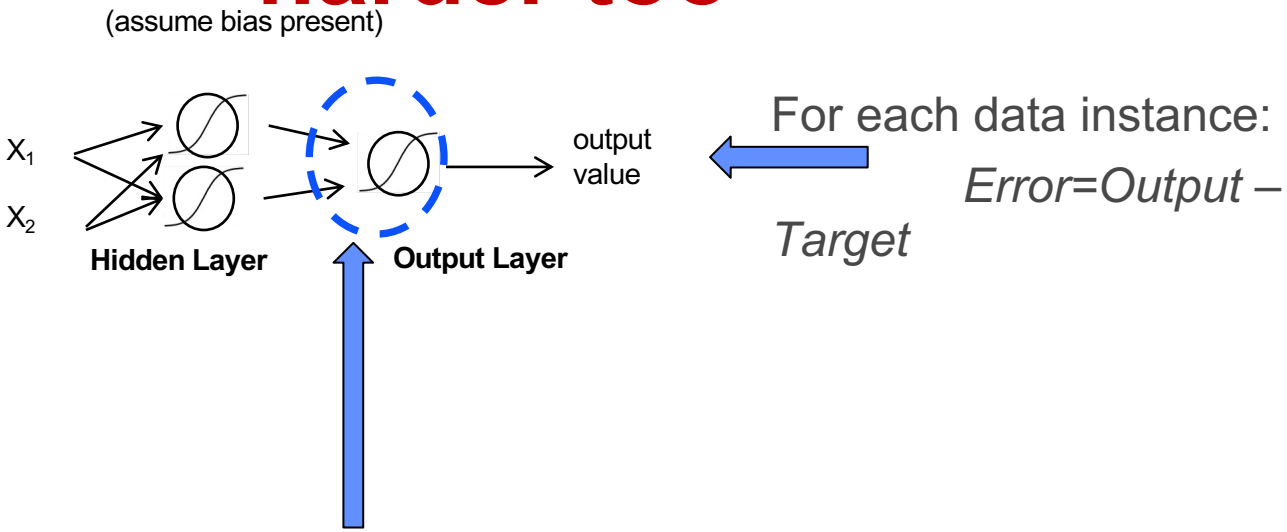
(assume bias present)



For each data instance:

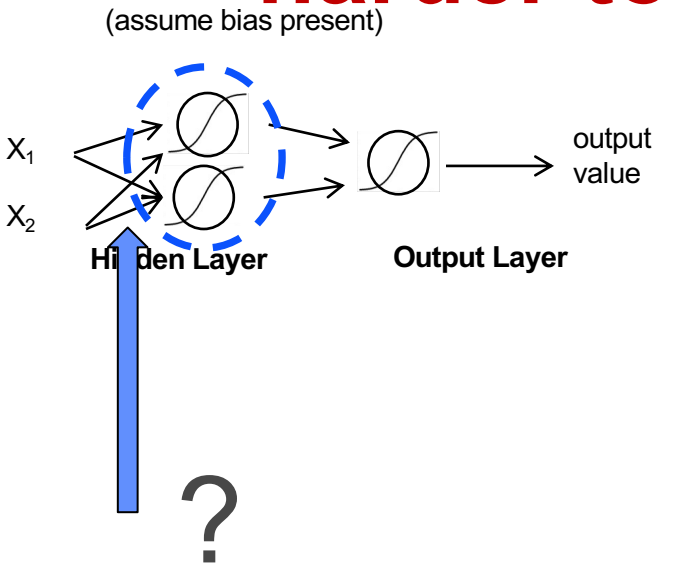
Error=Output – Target

But parameter fitting is harder too



The objective is to minimize *Error* related to output weights (same as for logistic regression)

But parameter fitting is harder too

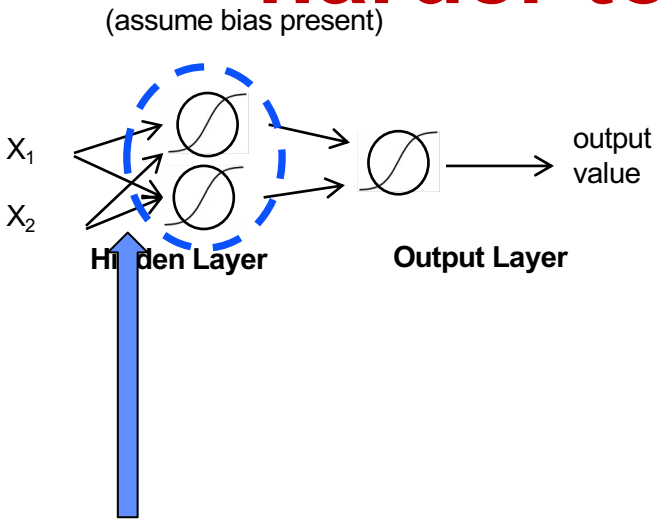


For each data instance:

Error=Output – Target

But, error signals are only known for output layer, what is error for hidden layer?

But parameter fitting is harder too



For each data instance:

Error=Output – Target

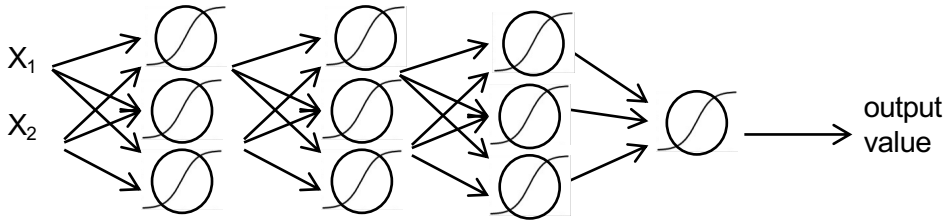
But, error signals are only known for output layer, what is error for hidden layer?

Minimize *Error* related to output weights, that is also related to hidden weights

(Use derivatives to 'back-propagate' errors, "stochastic gradient descent")

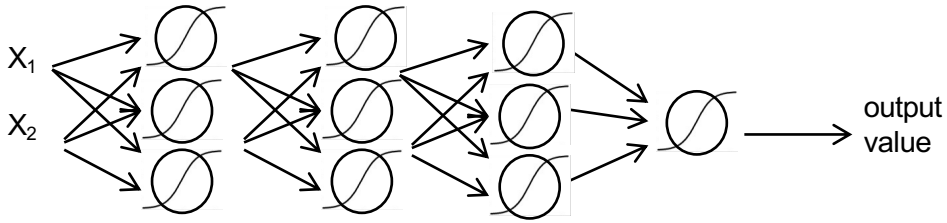
Why stop at 1 hidden layer?

- More hidden layers => More varied features, or ‘Deep’ Learning



Train with Care

- More hidden layers => More varied features, or ‘Deep’ Learning



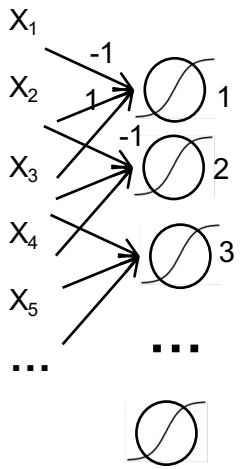
Many more parameters, and error signal at final output layer gets drowned out at lower layers-

but penalizing weight sizes, varied activation functions, and more data help!

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

*(*assume *b=0*)



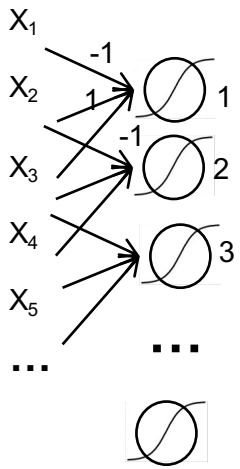
For node 1 let W= [w_1 w_2 w_3] = [-1 1 -1]

What values of x_1 ,x_2 , x_3 will give maximum node 1 output? (assuming -1 ≤ x ≤1)

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

*(*assume *b=0*)



For node 1 let W= [w_1 w_2 w_3] = [-1 1 -1]

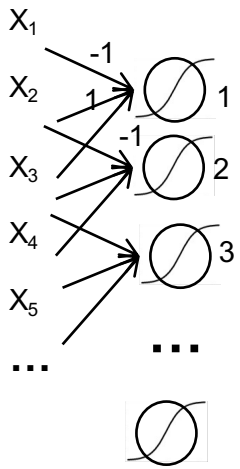
What values of x_1 ,x_2 , x_3 will give maximum node 1 output? (assuming -1 ≤ x ≤1)

Informally, node 1 has max activation for a 'spike', e.g. when [x_1 ,x_2 , x_3] = *[-1 +1 -1]*

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

*(*assume *b=0*)



For node 1 let W= [w_1 w_2 w_3] = [-1 1 -1]

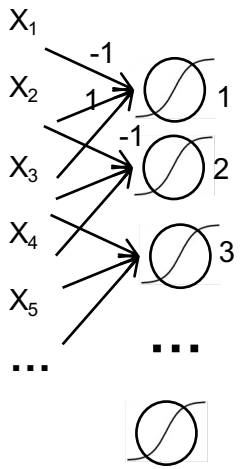
For node 2,3, etc… copy W for node 1

What is the hidden layer doing?

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume $b=0$)



For node 1 let W=[w_1 w_2 w_3] = [-1 1 -1]

For node 2,3, etc… copy W for node 1

What is the hidden layer doing?

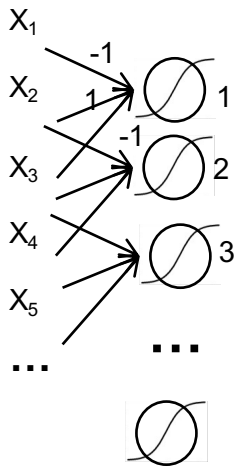
Informally, looking for a spike everywhere.

This is essentially a convolution operator, where W is the kernel.

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume $b=0$)



For node 1 let W=[w_1 w_2 w_3] = [-1 1 -1]

For node 2,3, etc… copy W for node 1

What is the hidden layer doing?

Informally, looking for a spike everywhere

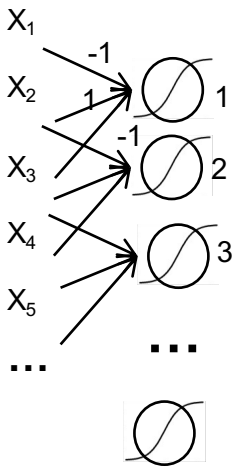
This is essentially a convolution operator, where W is the kernel.

Note: sharing weights is like *sliding* W across input

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume $b=0$)



For node 1 let W=[w_1 w_2 w_3] = [-1 1 -1]

For node 2,3, etc… copy W for node 1

What is the hidden layer doing?

Informally, looking for a spike everywhere

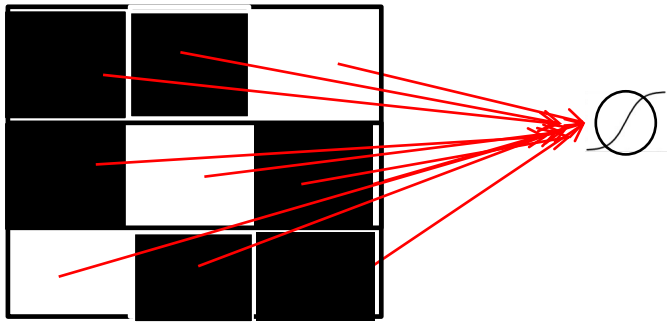
This is essentially a convolution operator, where W is the kernel.

Note: sharing weights is like *sliding* W across input

Note: if we take max activation across nodes ('Max Pool') then it's like looking for a spike *anywhere*.

2D Convolution

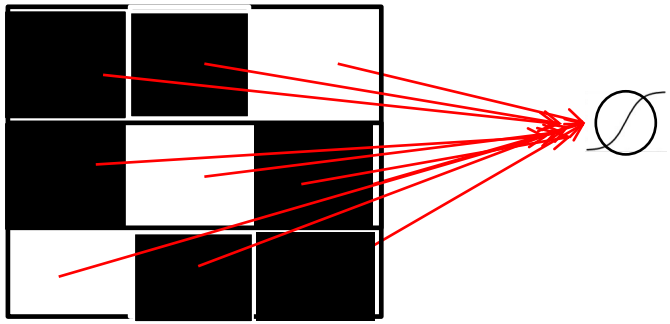
Now let input be a 2D binary matrix, e.g. a binary image) fully connected to 1 node



What W matrix would 'activate' for a upward-toward-left diagonal line?

2D Convolution

Now let input be a 2D binarized 3x3 matrix fully connected to 1 node



(black= -1 white=1)

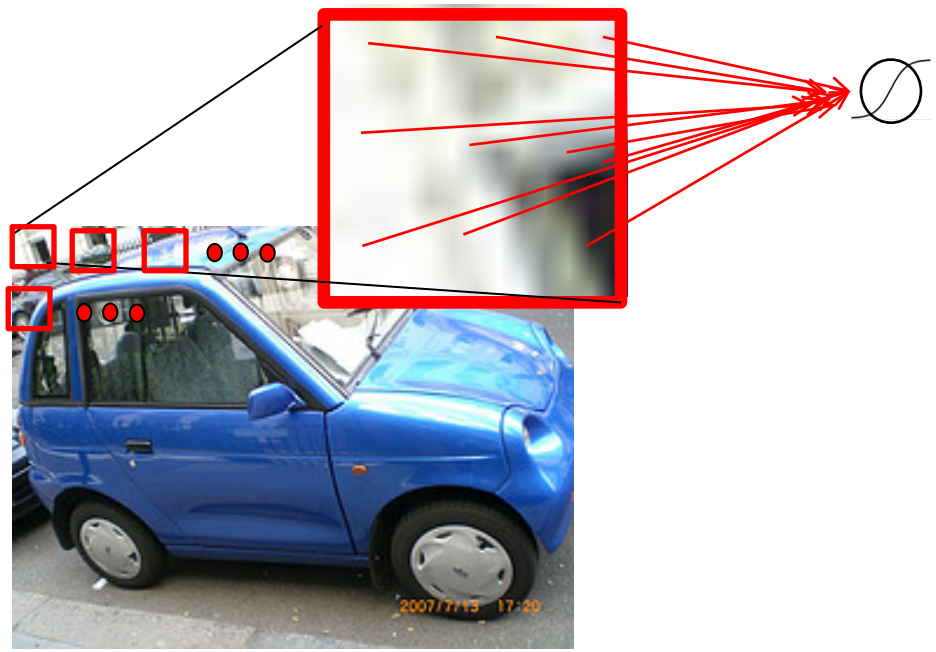
What W matrix would 'activate' for a upward-toward-left diagonal line?

How about:

W= [-1 -1 1
-1 1 -1
1 -1 -1]

2D Convolution

For full image, 1 filter is applied to 1 region in 1 color channel at a time, and then slid across regions (or done in parallel with shared weights) and produces 1 new 2D image (hidden) layer



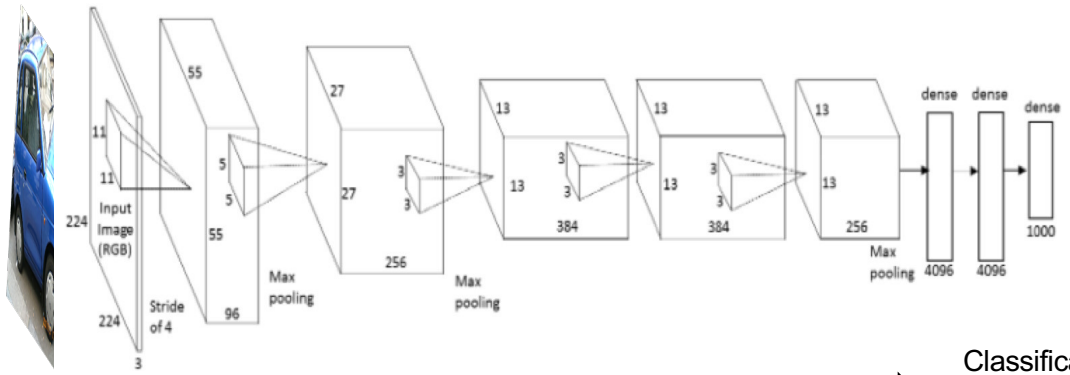
Convolution Layer parameters:

- filter size depends on input:
  - smaller filters for smaller details
    - 2 layers of 3x3 ~ 1 layer of 5x5
- sliding amount
  - smaller better but less efficient
- number of filters
  - depends on task
  - each filter is a new 2D layer

Convolution Network :
- many layers and architecture options

Large Scale Versions

- Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)
- Need large amounts of data and many heuristics to avoid overfitting and increase efficiency

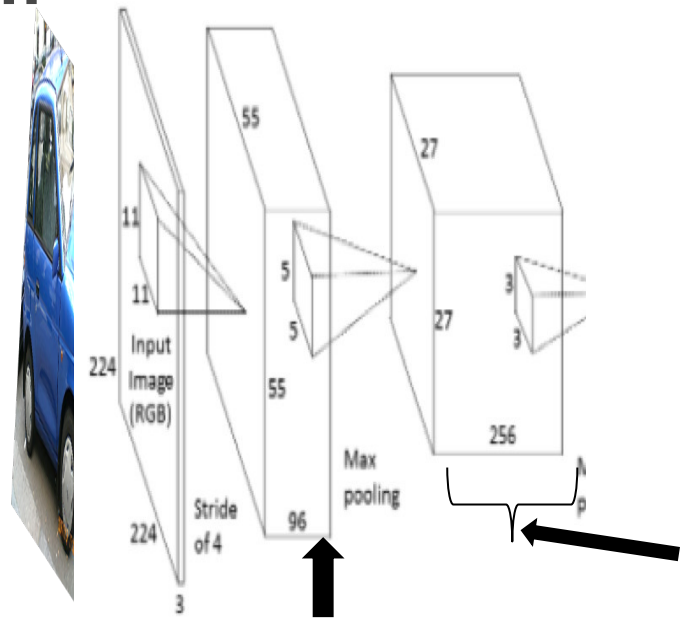


Convolution layers

Classification layers and output

Large Scale Versions

- **Zooming in:**

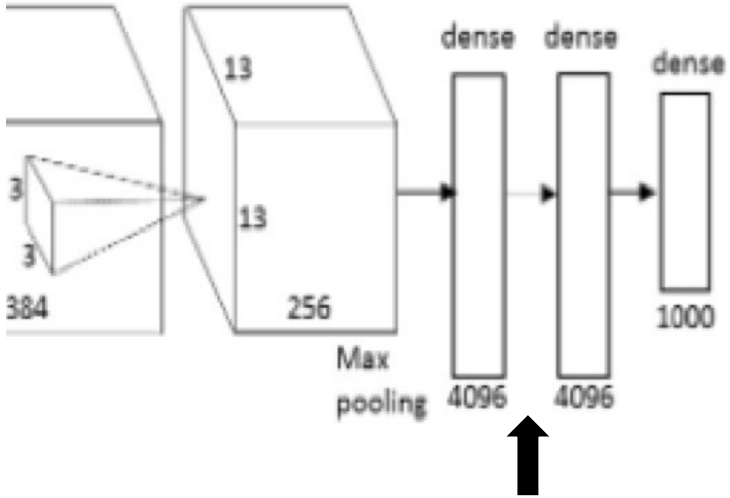


The thickness is the number of different convolutions, i.e. different transformations, sometimes called "channels"

Each convolution layer uses RELU (rectified linear activation units instead of logistic function) and is followed by Max Pooling layer (over 2D regions with sliding)

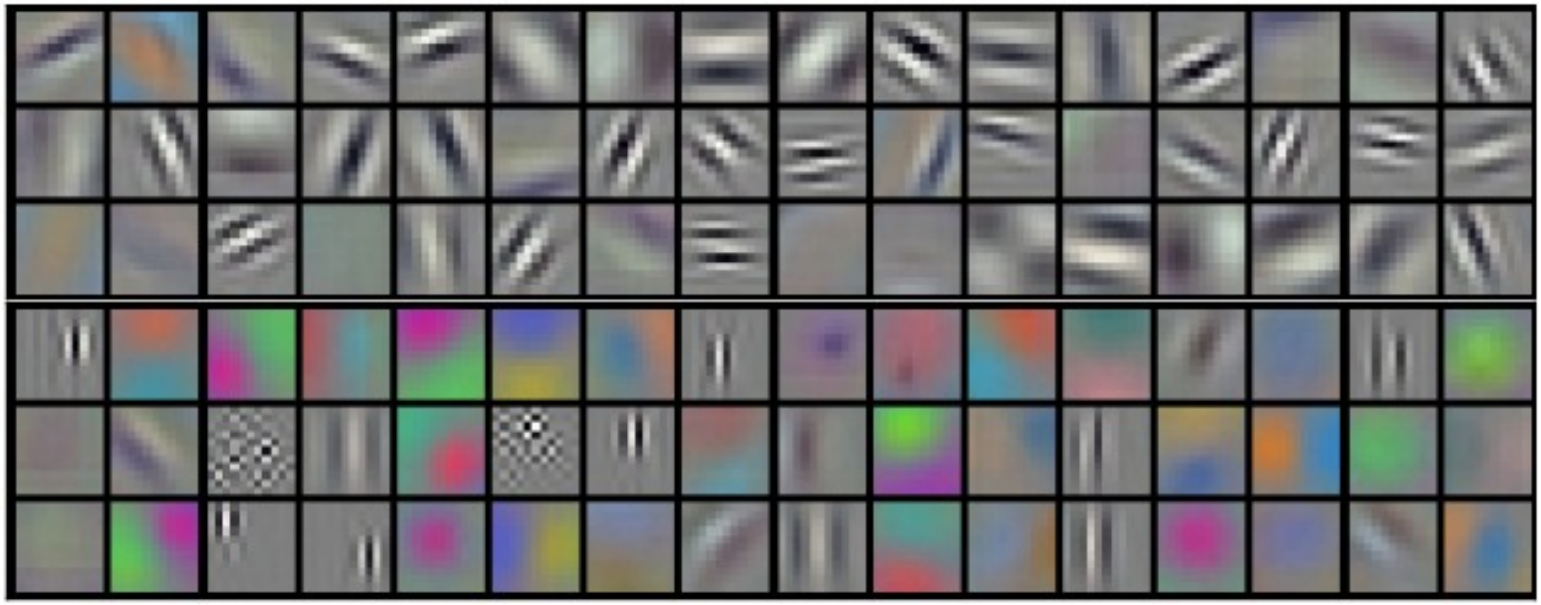
Large Scale Versions

- **Zooming in:**



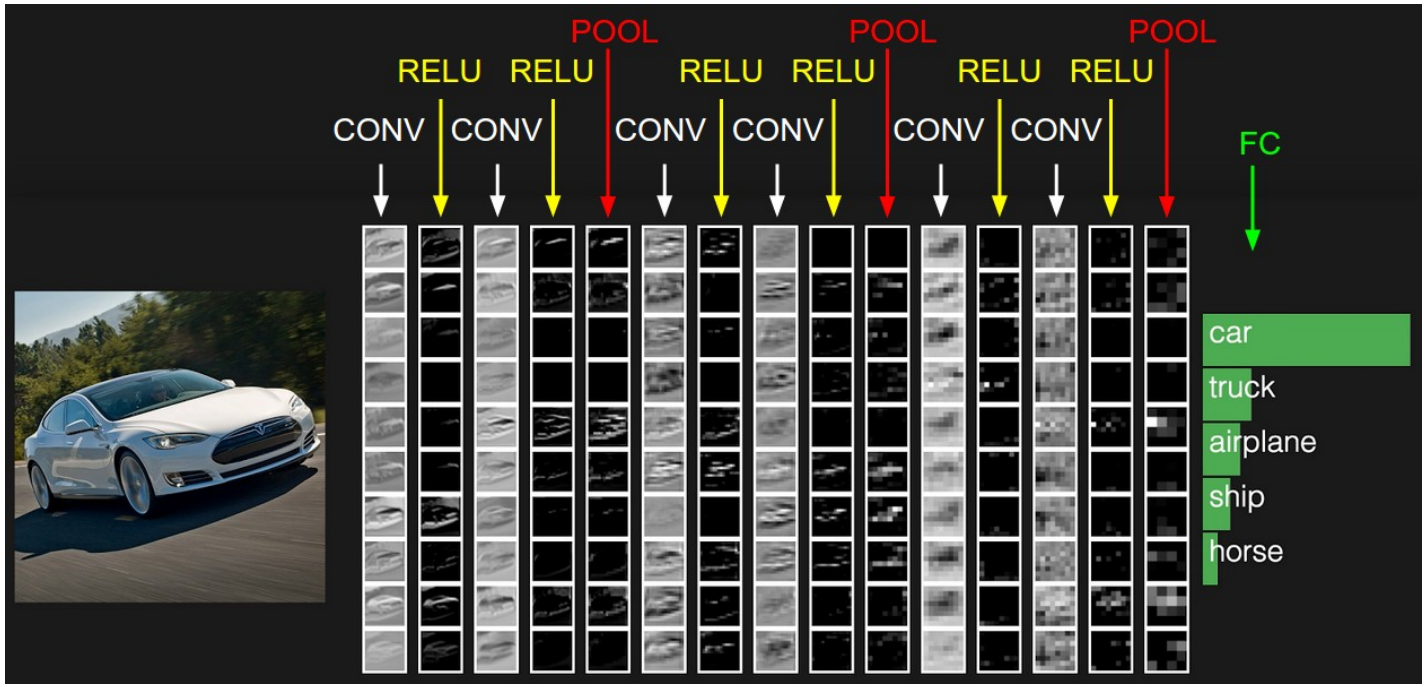
Last convolution layer is laid out as a vector for input into classification layers.
Classification uses dense, i.e. fully connected, hidden layers and output layer.

What Learned Convolutions Look Like



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." Advances in neural information processing systems. 2012.

What Learned Convolutions Look Like



Summarizing Deep Layers

- **Hidden layers transform input into new features:**
  - Feature can be highly nonlinear
  - Features as a new space of input data
  - Features as projection onto lower dimensions (compression)
  - Features as filters, which can be used for convolution
- **But also:**
  - Many algorithm parameters
  - Many weight parameters
  - Many options for stacking layers

References

- **Book: https://mitpress.mit.edu/books/deep-learning**
- **Documentation: https://keras.io/**
- **Tutorials I used (borrowed):**
  - http://cs231n.github.io/convolutional-networks/
  - https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59
  - https://github.com/julienr/ipynb_playground/blob/master/keras/convmnist/keras_cnn_mnist.ipynb

Overview of TensorFlow, PyTorch, Horovod

Mahidhar Tatineni, SDSC

Implementing DL techniques in Practice

- **Frameworks/Libraries like TensorFlow, Torch, Keras, and Horovod make it easy to program DL tasks**
  - High level APIs and interfaces from python, C++ etc.
  - Implementations of commonly used neural-network building blocks.
  - Scalable, distributed options

- **Quick overviews today. Lot of training material online if you are interested.**

TensorFlow

- **Open-source machine learning library originally developed by Google Brain team.**
- **High level Keras API in python**
  - Modular building blocks to create and train DL models.
  - Allows assembly of layers, lot of common use cases supported out of the box
  - Allows for custom blocks, can create new layers, loss functions etc.
- **APIs for Eager Execution, Importing data**
- **Pre-made Estimators for training, evaluation, prediction, and export.**

Links: https://www.tensorflow.org/guide/
https://www.tensorflow.org/tutorials/

Machine Learning/Deep Learning on Comet via Singularity

- Bulk of the Singularity usage on Comet is for machine learning/deep learning applications.
- Lot of these packages are constantly upgraded and the dependency list is difficult to update in the standard Comet environment.
- Install options
  - Singularity image provides dependencies and user can compile actual application from source. e.g. Torch
  - Entire dependency stack and the application is in the image. e.g Keras with backend to TensorFlow and some additional python libraries
- Run options
  - Most cases are run on single GPU nodes (4 GPUs at most)
  - Can access this via Jupyter notebooks
  - Multi-node options are possible but difficult to set up via singularity.

Tutorial

- MNIST database of handwritten printed digits
- The ‘hello world’ of Conv. Neural Networks
- Use Keras front end (high level neural functions) to Tensorflow engine (neural math operations)
- Works with GPU or CPUs

MNIST Example

- **Keras, deep learning library in python - backend to Tensorflow or Theano. Today we will use Tensorflow.**

- **Deep learning refers to neural networks with multiple hidden layers that can learn increasingly abstract representations of the input data. For example in a image classification case**
  - first layers might learn local edge pattern
  - each subsequent layer (or filter) gets more complex reps
  - eventually the last layer can classify images - car, tree etc.

- **Convolutional Neural Networks - images are the input data - reduce the parametric space for tuning and effectively handle the high dimensionality of raw images.**

- **References:**
  - https://cambridgespark.com/content/tutorials/deep-learning-for-complete-beginners-recognising-handwritten-digits/index.html

MNIST Example via Jupyter Notebook

- **Step 1: Follow the instructions (next page) for spinning up the Jupyter notebook**

- **Step 2: From the browser window, open the following notebook:**

  - LabMNIST_Final.ipynb

  (See /share/apps/examples/SCC/ML_Tensorflow)

Accessing via Jupyter Notebook

[1] Get an interactive node:

```
srun --pty --nodes=1 --ntasks-per-node=24 -p compute --t 02:00:00 --wait 0 /bin/bash
```

[2] Load the singularity module and get an interactive shell

```
module load singularity
singularity shell /share/apps/gpu/singularity/sdsc_ubuntu_tf1.1_keras_R.img
```

[3] Launch the notebook

```
ipython notebook --no-browser --ip=`/bin/hostname`
```

This will give you an address which has localhost in it and a token. Something like:

http://comet-14-0-4:8888/?token=389587c9d1b69f8f595e7d8bfdd83c9961ed26b8b3f1bb3e

You can then paste it into your browser. That should get you into the running notebook. From there everything should be working as a regular notebook. Note: This token is your auth so don't email/send it around.

PyTorch

- **Deep learning platform**
  - Hybrid front-end w/ ease of use/flexibility in eager mode, graph mode for speed, optimization
  - Deep integration with Python
  - C++ front end
- **Pre-trained model repository, can be customized**
- **Ecosystem of tools built on top of Torch.**
- **Quick intro here:**
  - https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

PyTorch on Comet

- Enable via Singularity
- Examples: /share/apps/examples/pytorch

```
#SBATCH --account=use300      # Modify this line to use your Project ID!

#SBATCH --partition=compute   # Request 1 'compute' nodes for up to 15 minutes.
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=24
#SBATCH --time=00:15:00

#SBATCH --job-name=pytorch-compute
#SBATCH --output=pytorch-compute.o%j.%N

#SBATCH --no-requeue          # Do not requeue job under any circumstances.

declare -xr LOCAL_SCRATCH="/scratch/${USER}/${SLURM_JOB_ID}"
declare -xr LUSTRE_SCRATCH="/oasis/scratch/comet/${USER}/temp_project"

declare -xr SINGULARITY_MODULE='singularity/2.3.2'
declare -xr SINGULARITY_IMAGE_DIR='/oasis/scratch/comet/mkandes/temp_project/sin
gularity/images'

module purge
module load "${SINGULARITY_MODULE}"
module list
printenv

time -p singularity exec "${SINGULARITY_IMAGE_DIR}/pytorch-cpu.img" python "${SL
URM_SUBMIT_DIR}/examples/mnist/main.py" --no-cuda
```

Horovod

- Distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.
- Distributed TensorFlow with parameter servers can be tricky to setup and involve code changes.
- Horovod leverages MPI to make things easier and also picks up performance by using MPI functions which are optimized for the hardware involved.

Typical Horovod code additions

- Initialization (hvd.init())
- Assign GPUs
- Set number of workers (scale up)
- Change optimizer to use Horovod (hvd.DistributedOptimizer). This will allow leverage of MPI functions like AllReduce, AllGather for the gradient computations.
- Broadcast global variables
- Checkpoints written from worker 0.
- MNIST example here:
  - https://github.com/horovod/horovod/blob/master/examples/keras_mnist.py

Summary

- Several frameworks available for developing/training/testing machine learning/deep learning models. TensorFlow, PyTorch commonly used on Comet system.
- Both TensorFlow and PyTorch are available via Singularity on Comet.
- High level APIs available for use via python. Easy to incorporate into Jupyter notebooks.