TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

Machine Learning on HPC: Deep Learning

Scalable Computational Intelligence Group
Texas Advanced Computing Center
University of Texas at Austin

Juliana Duncan
July 2024

Workshop Series Overview

- There are five workshops this week:
  - Workshop 1: July 15, 2024
    - Introduction on computing cluster environment, Python for machine learning and containers
  - Workshop 2: July 16, 2024
    - Common supervised machine learning methods with examples and hands-on exercises
  - Workshop 3: July 17, 2024
    - Common unsupervised machine learning methods with examples and hands-on exercises
  - **Workshop 4: July 18, 2024**
    - **Deep learning concepts, common neural network structure and frameworks with examples and hands-on exercises.**
  - Workshop 5: July 19, 2024
    - Advanced ML topics
      - Introduction to Reinforcement Learning and LLMs

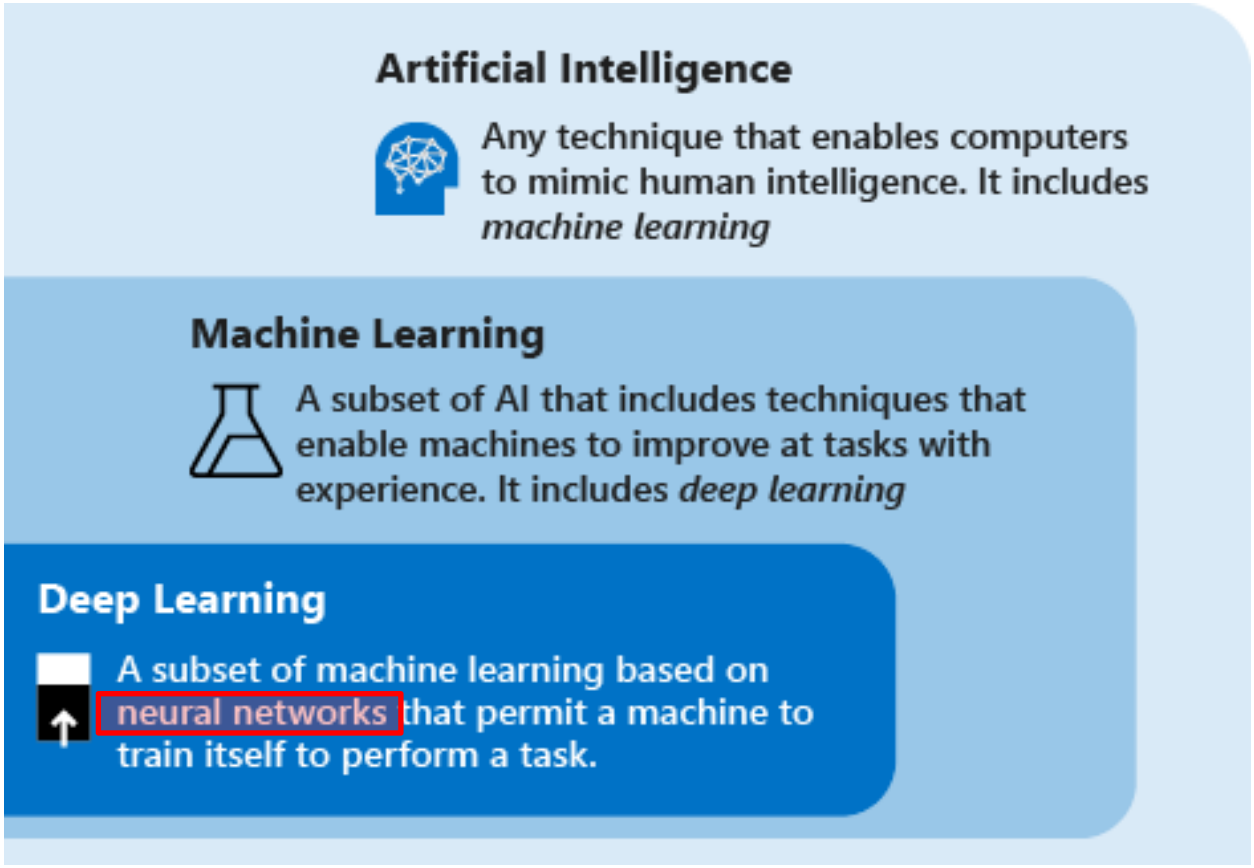
Today's Outline

- Morning (9-12)
  - Lecture: Introduction to Deep Learning
  - Hands on Exercise: Training a CNN Classifier
- Lunch (12-1)
- Afternoon (1-4)
  - Lecture: Introduction to Distributed Deep Learning
  - Hands on Exercise: Train a CNN Classifier with multiple GPUs

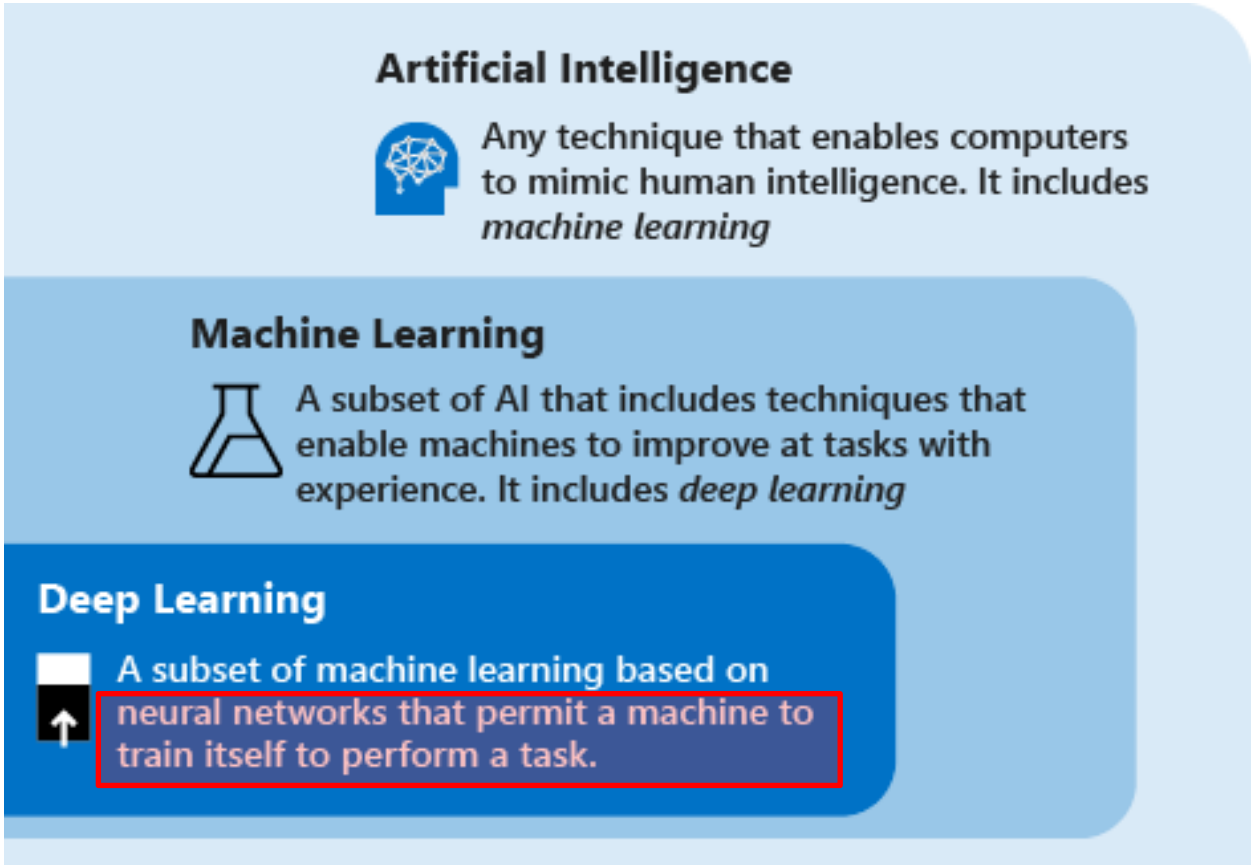
Morning Outline

- Introduction on basic ML/DL models
  - Introduction to Deep Learning
  - Review of Supervised Learning
  - Multi-layer Perceptron (MLP)
    - Architecture
    - Training
  - Convolutional Neural Network (CNN)
    - Residual Networks
    - Transfer Learning
- Case study: Natural Hazard Detection
  - Hands-on session

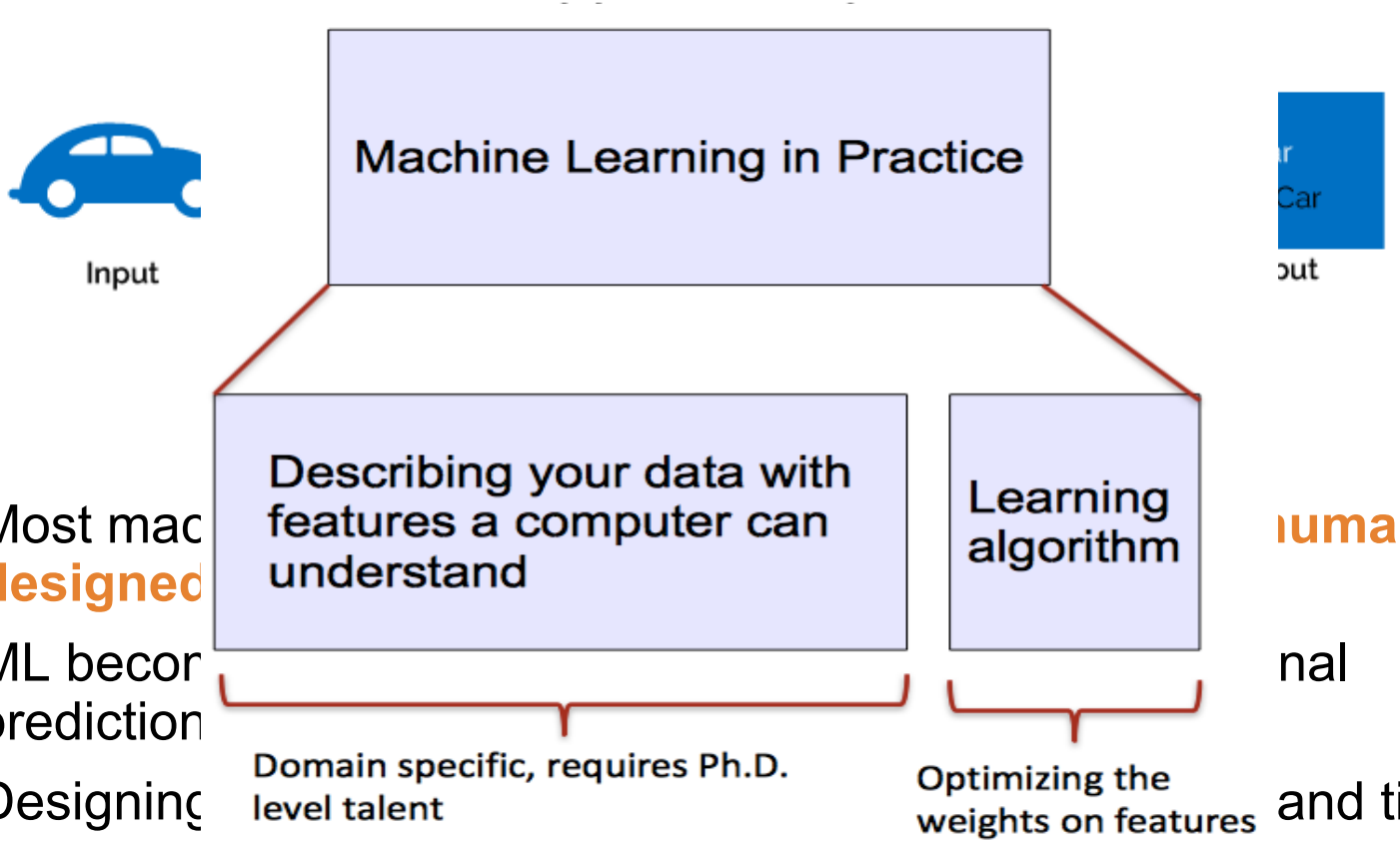
What is Deep Learning?



What is Deep Learning?

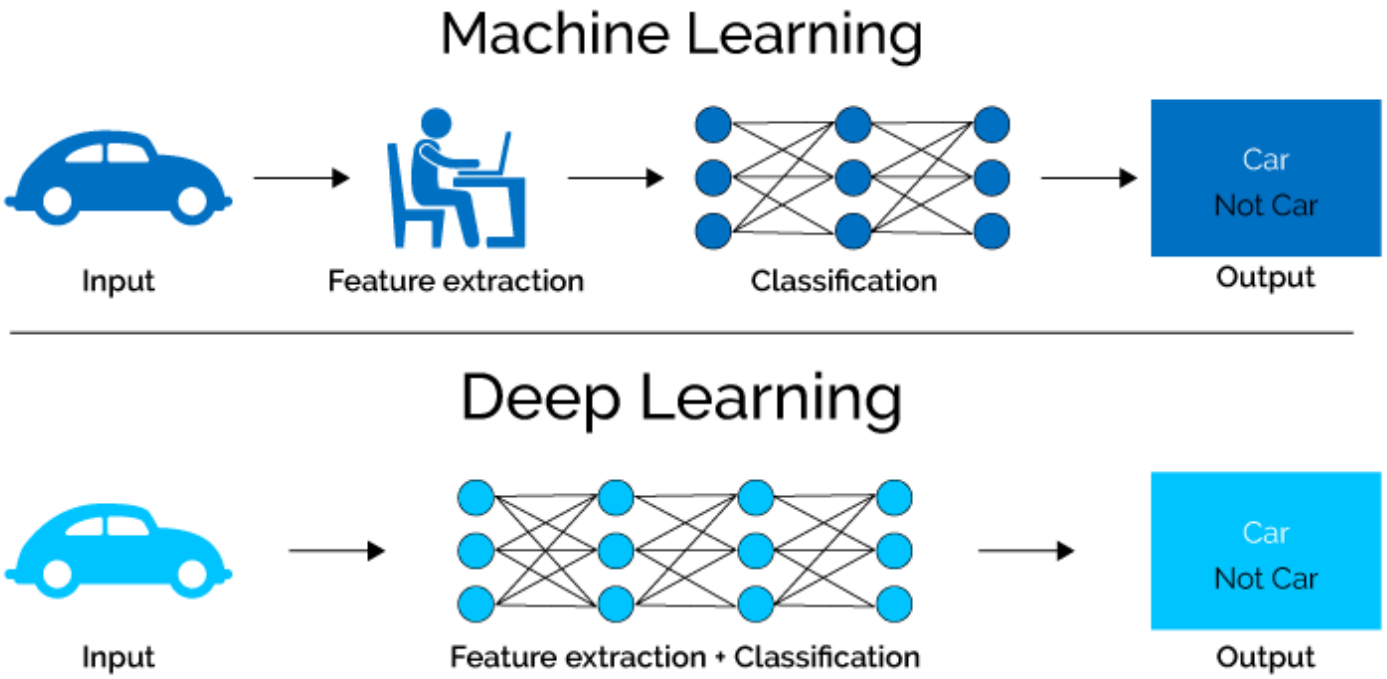


Machine Learning vs Deep Learning



- Most mac … uman-designed
- ML becor … nal prediction
- Designing … and time.

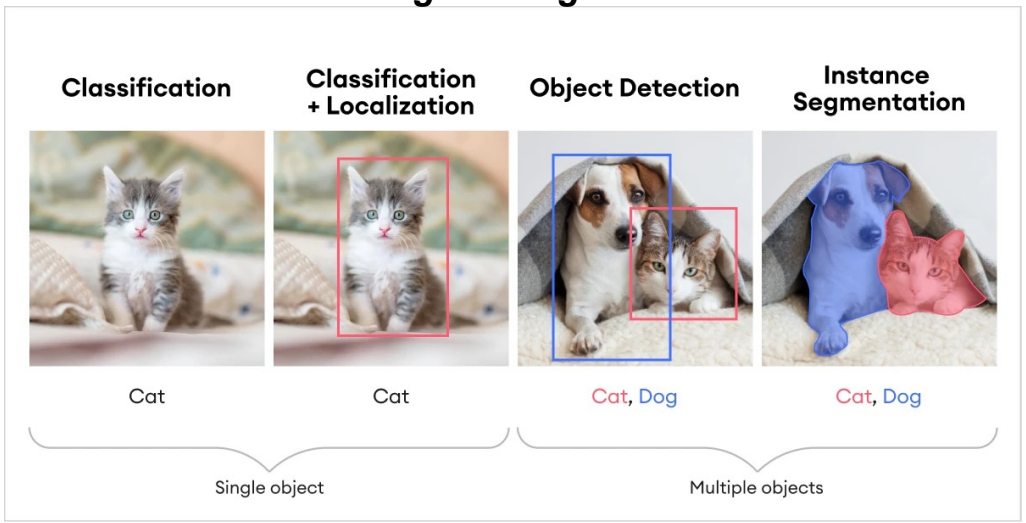
Machine Learning vs Deep Learning



Deep learning algorithms attempt to do the feature extraction for you by increasing the complexity of the model via multiple layers in neural network.

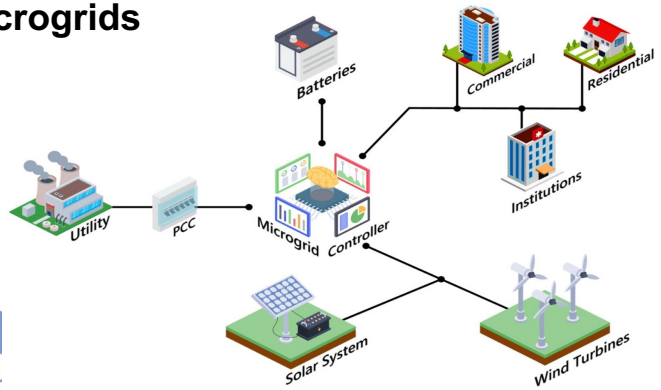
Applications of Deep Learning

Image Recognition

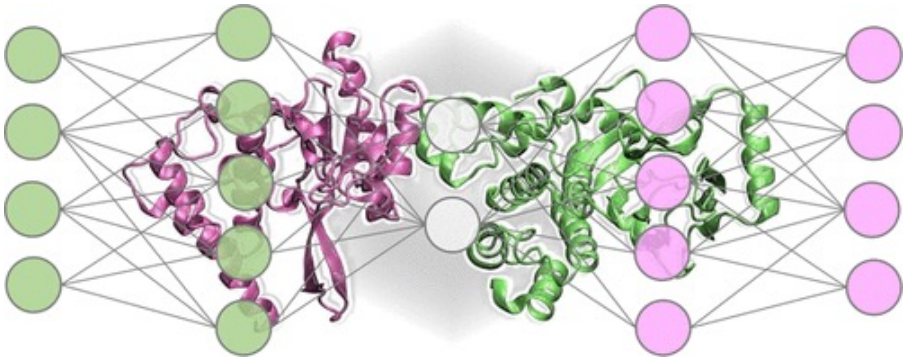


ChatGPT

Deep Reinforcement Learning for Managing Microgrids



Deep Learning Force Fields



Morning Outline

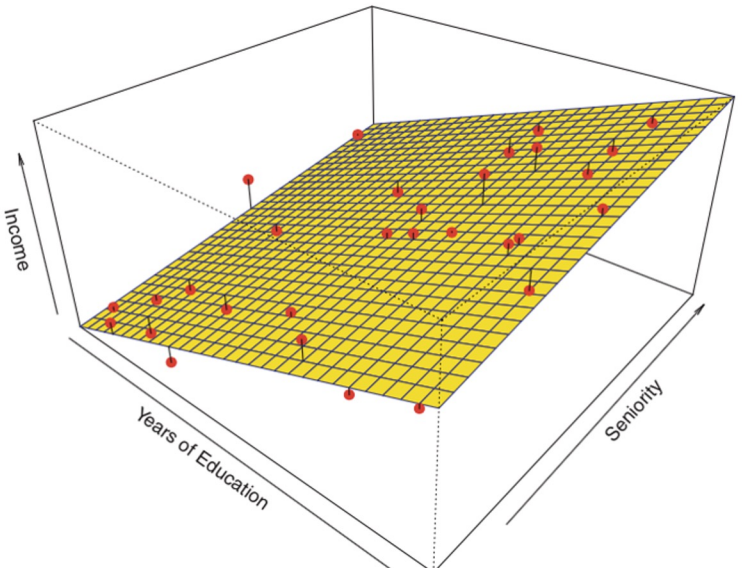
- Introduction on basic ML/DL models
  - Introduction Deep Learning
  - **Review of Supervised Learning**
  - **Multi-layer Perceptron (MLP)**
    - Architecture
    - Training
  - Convolutional Neural Network (CNN)
    - Residual Networks
    - Transfer Learning
- Case study: Natural Hazard Detection
  - Hands-on session

Review: Multiple Linear Regression

- More than one independent variable
- x is a vector of features, $x \in R^N$
- $\boldsymbol{\theta}$ is a vector of weights, $\boldsymbol{\theta} \in R^N$
- $\theta = \langle b, a_0, a_1, \ldots a_N \rangle$

$$f(x, \theta) = b + a_0 x_0 + a_1 x_1 + \cdots + a_N x_N$$

Years of Education	Years experience	Salary
13	1.1	41,000
17	1.7	52,000
19	3	53,000
22	4.2	70,000



Review: Logistic Regression

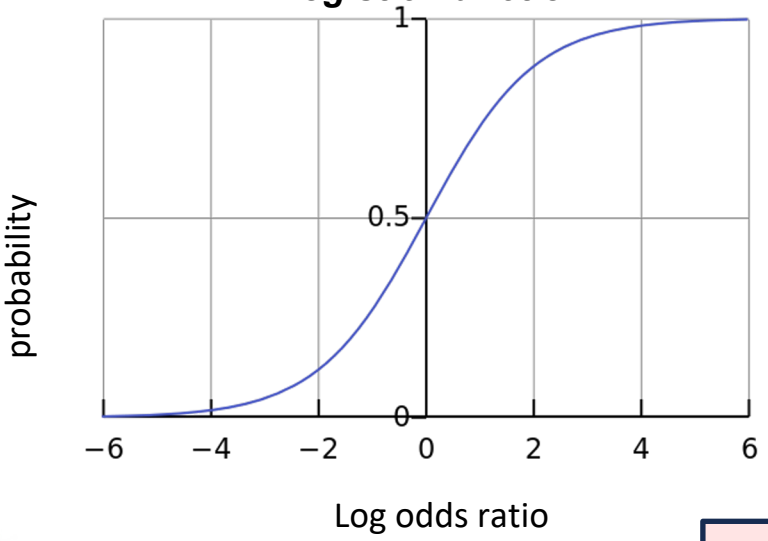
Linear Regression for classification?

$$pr \quad y = f(x,\theta) = b + a_0x_0 + a_1x_1 + \cdots + a_Nx_N$$

Instead linear model predicts the log odds ratio

$$t = ln\left(\frac{p}{1-p}\right) = f(x,\theta) = b + a_0x_0 + a_1x_1 + \cdots + a_Nx_N$$

Logistic Function



Logistic Function

Converts log odds ratio to probabilities

$$\sigma(t) = \frac{1}{1+e^{-t}}$$

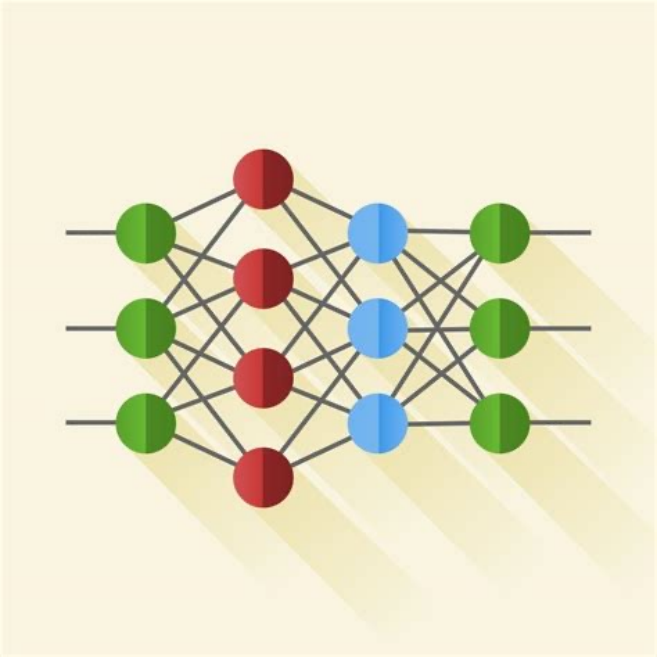
$$p(x,\theta) = \sigma(b + a_0x_0 + a_1x_1 + \cdots + a_Nx_N)$$

Multi Layer Perceptron(MLP)

Morning Outline

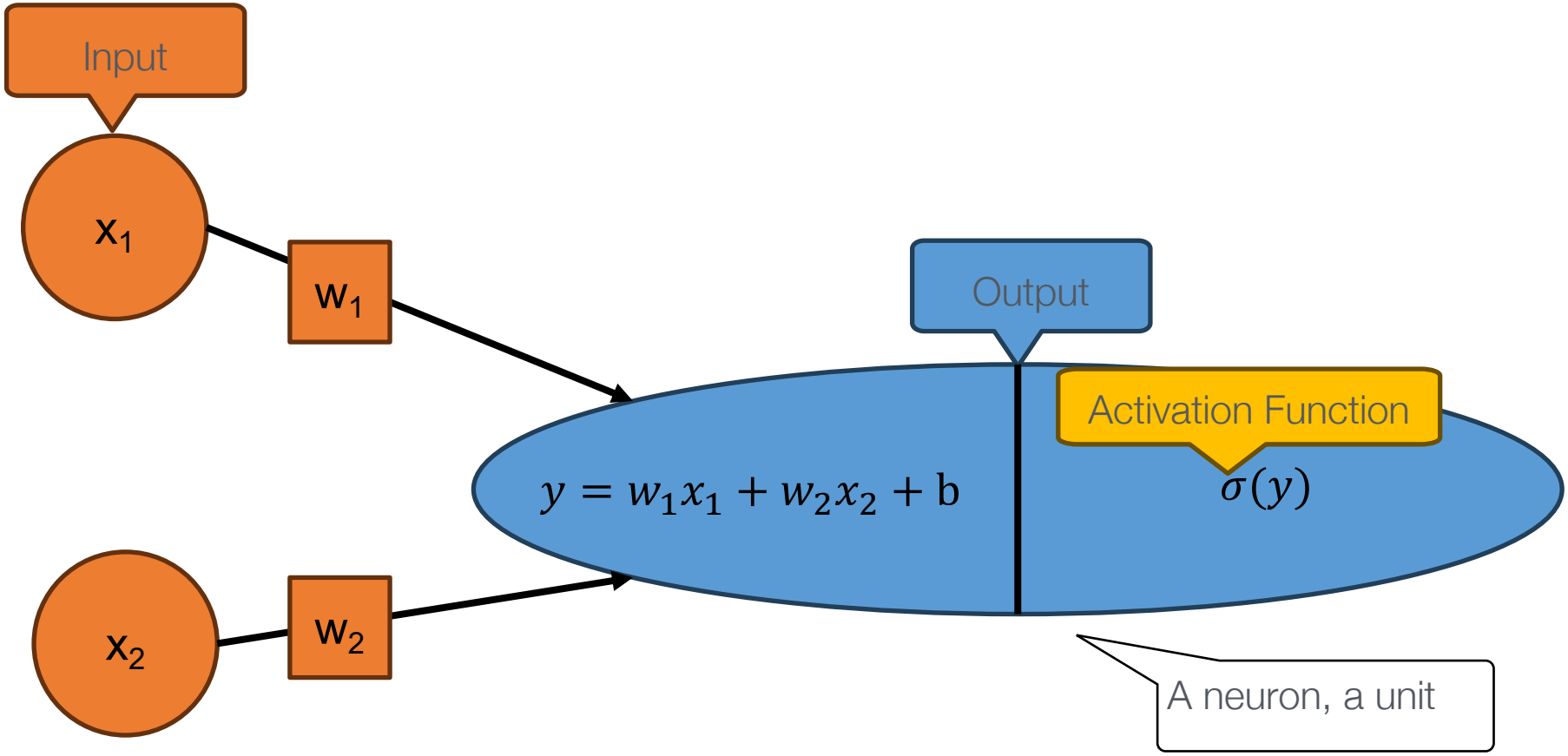
- Introduction on basic ML/DL models
  - Introduction Deep Learning
  - **Review of Supervised Learning**
  - **Multi-layer Perceptron (MLP)**
    - **Architecture**
    - Training
  - Convolutional Neural Network (CNN)
    - Residual Networks
    - Transfer Learning
- Case study: Natural Hazard Detection
  - Hands-on session

Multilayer Perceptrons (MLPs)



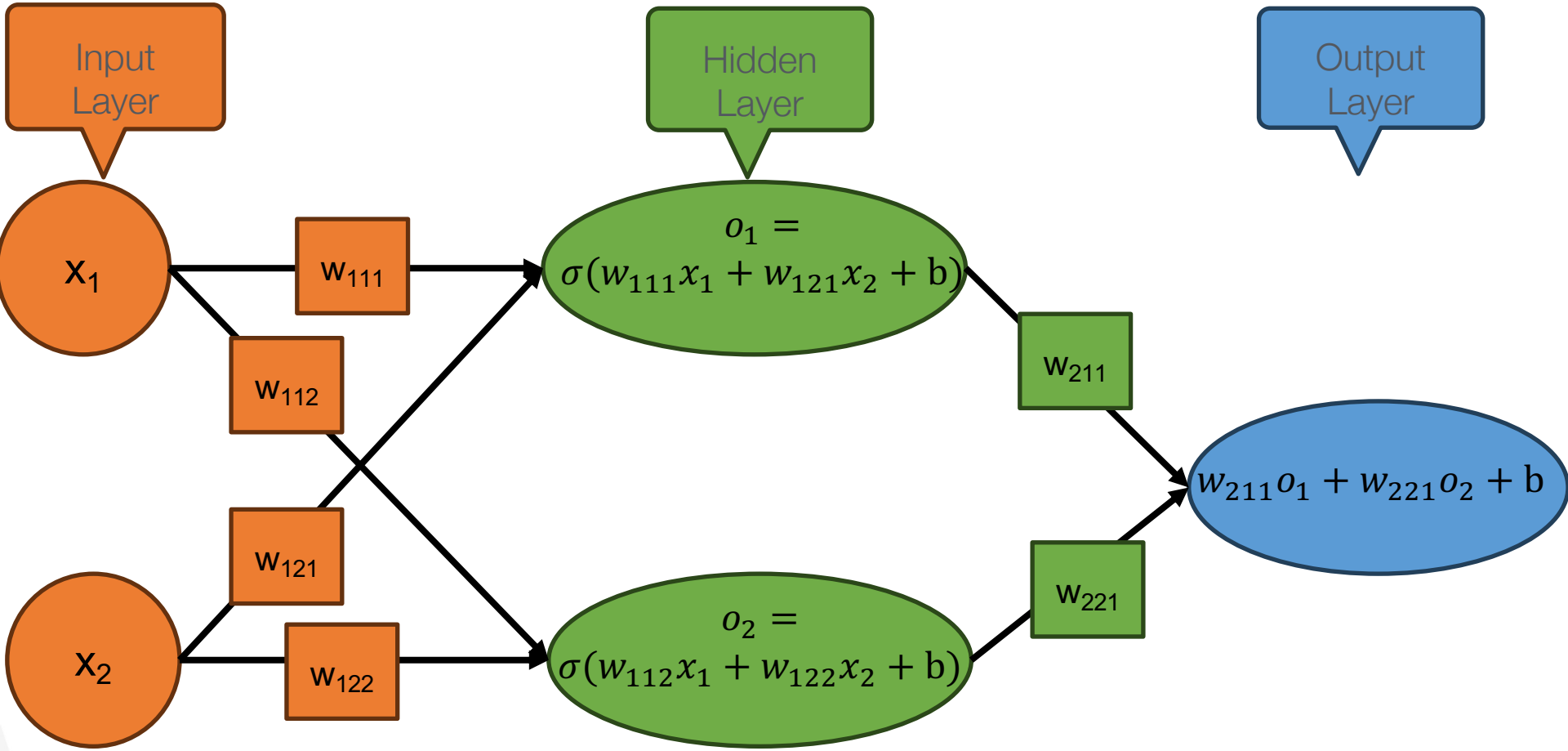
- MLPs can be expressed as a directed graph
- What does this graph represent? How is connected to building supervised ML models?
- Let's explain this by visualizing Linear and Logisitic Regression

Linear/Logistic Regression to Neural Networks



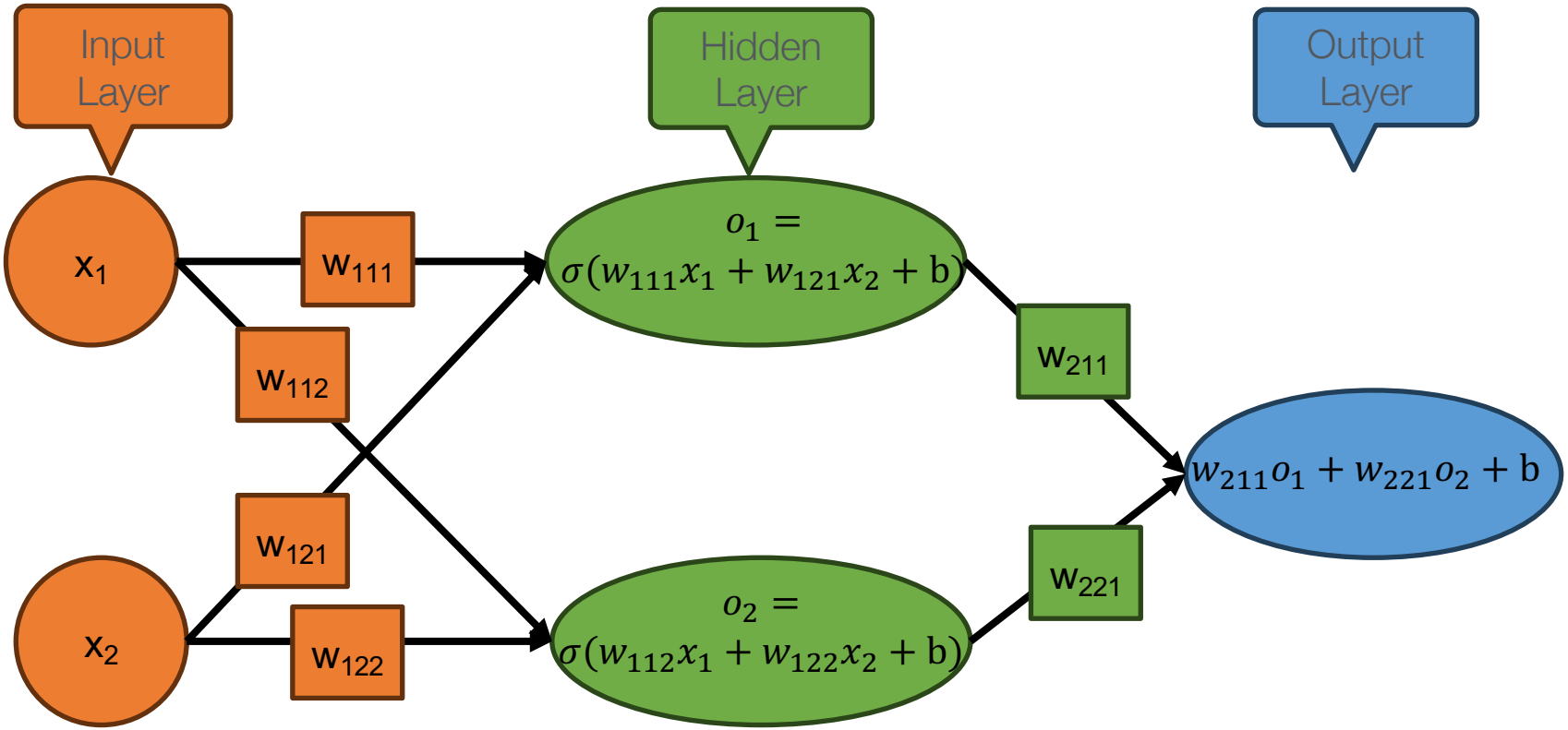
Neural networks were inspired by the complex way neurons, a fundamental building block of your nervous system, communicate in the human brain.

Neural Networks: 1-Hidden Layer with 2 Neurons



Three numbers represent: input layer, input node, output node

Neural Networks: 1-Hidden Layer with 2 Neurons

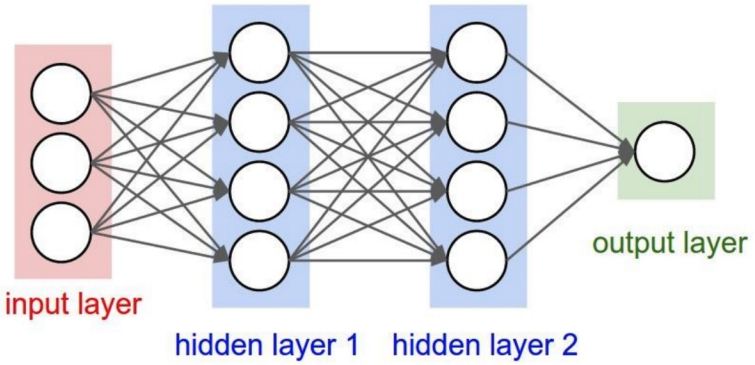


$$x = [x_1, x_2] \qquad w^{(1)} = \begin{bmatrix} w_{111} & w_{112} \\ w_{121} & w_{122} \end{bmatrix} \qquad o^{(1)} = \sigma\left(\left(w^{(1)}\right)^T x\right) \qquad o^{(2)} = \left(w^{(2)}\right)^T o^{(1)}$$

Multi-Layer Perceptron

- Layers in MLP are called **fully connected layer**.
  - Neurons in each layer are fully connected to the neurons in its following layer.

A 3-layer neural network:



$w^{(i)}$: weight matrix of layer i
$o^{(i)}$: output of layer i
x: input vector(s), y: output scalar
σ: activation function (e.g. sigmoid, ReLU)

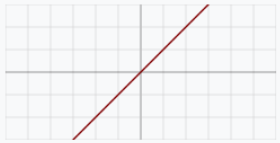
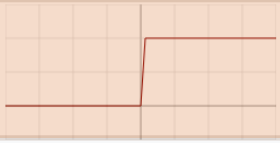
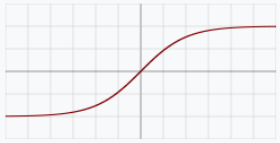
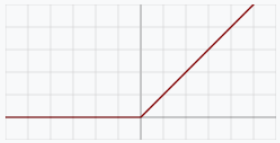
$$o^{(1)} = \sigma\left(\left(w^{(1)}\right)^T x\right)$$

$$o^{(2)} = \sigma\left(\left(w^{(2)}\right)^T o^{(1)}\right)$$

$$y = \left(w^{(3)}\right)^T o^{(2)}$$

Image credit: Stanford CS231n

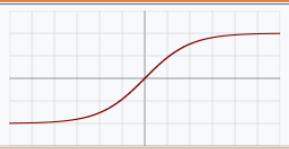
Activation Functions

Name	Plot	Function, $f(x)$
Identity		x
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Log sign soft		
tanh		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)[11]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\}$

When the binary step activation is used a neuron is a perceptron.

https://training.galaxyproject.org/training-material/topics/statistics/tutorials/FNN/tutorial.html

Activation Functions

Name	Plot	Function, $f(x)$
tanh		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)[11]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\}$

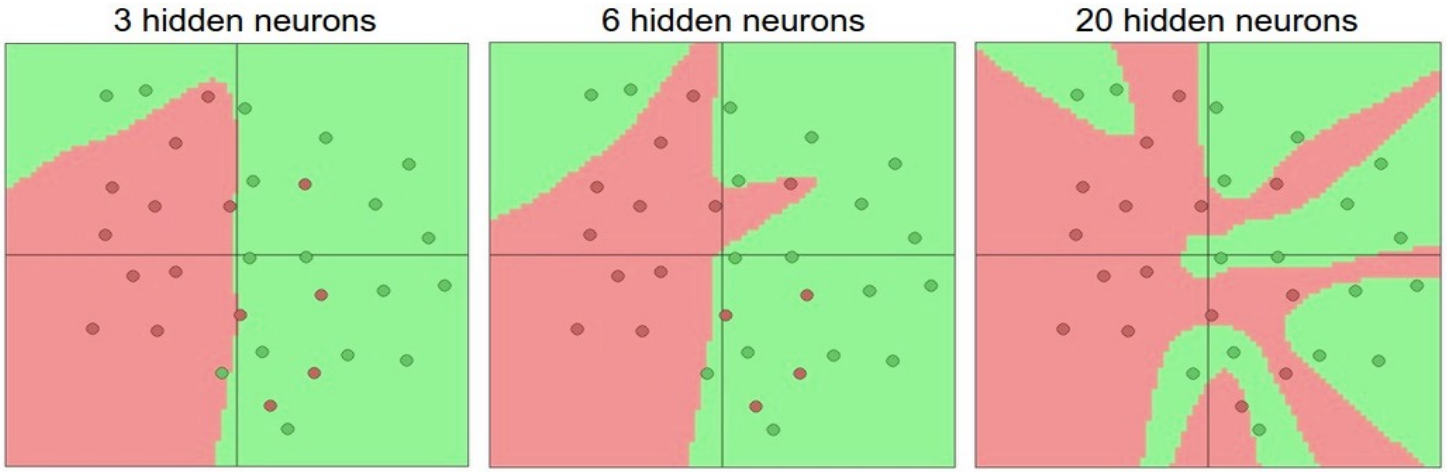
ReLu is one of the most popular activation functions today. Why?

- Training is fast
- Less expensive operations
  - compared to sigmoid/tanh (exponentials etc.)
  - implemented by simply thresholding a matrix at zero

https://training.galaxyproject.org/training-material/topics/statistics/tutorials/FNN/tutorial.html

Why Activation Functions?

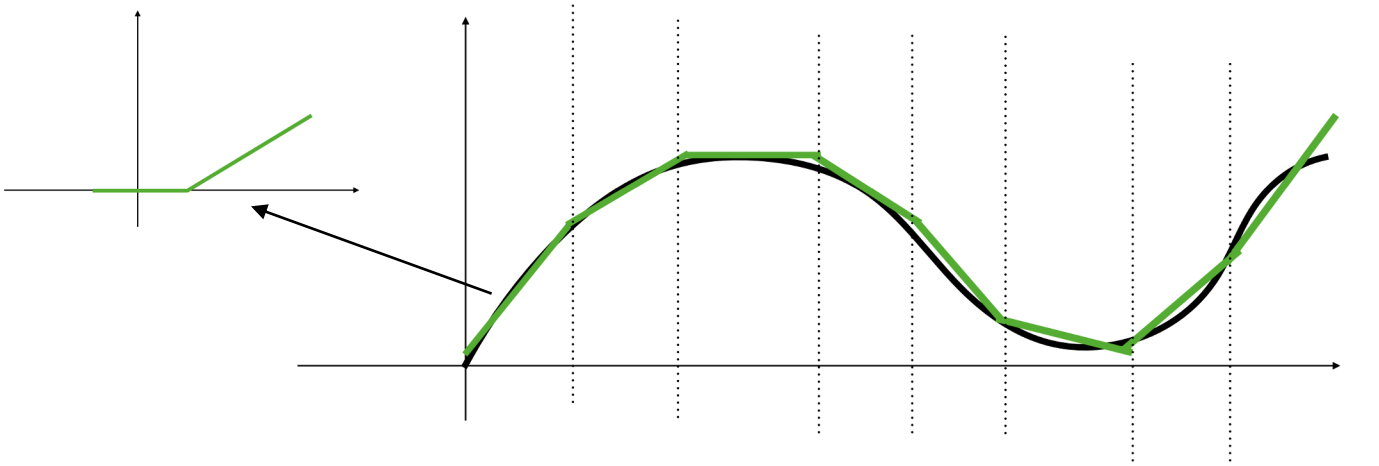
- Activation Functions allow us to find non-linear relationships
- More layers and neurons can approximate more complex functions



http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

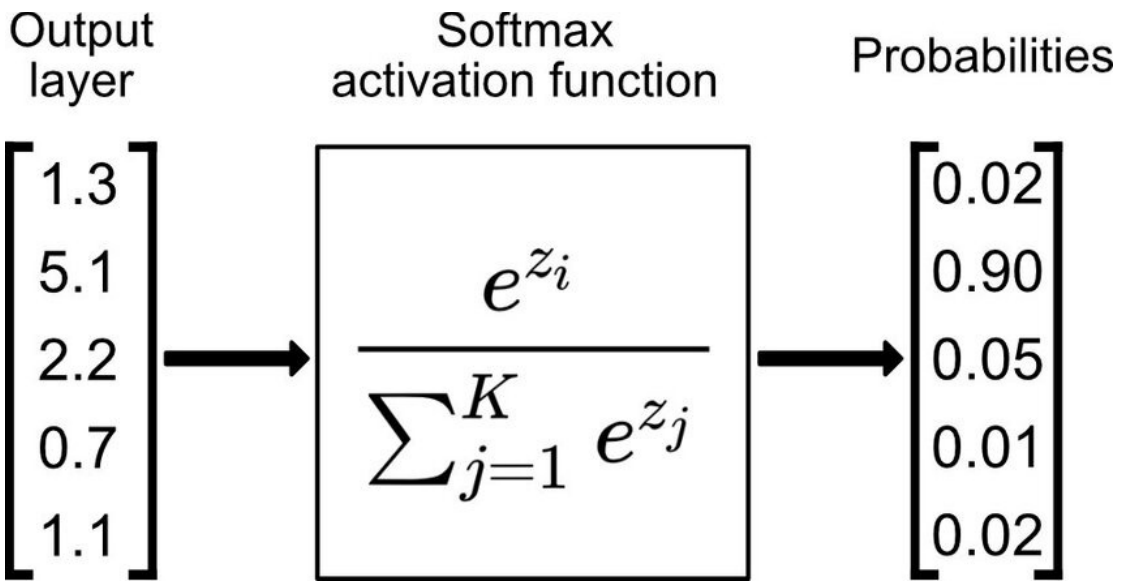
Nonlinear Relationships with ReLu Activation

- When MLP is applied to a linear regression problem, it can be considered as building a piecewise linear function to simulate the target curve.



Activation Functions for Multiclass Classification: Softmax

- Multiclass Classification (> 2 targets; e.g. cat, dog, bunny)
- Softmax Activation function is used in the final layer for Multiclass Classification models.
- Converts linear layer to probabilities



Deep Learning Terminology

- Multilayer Perceptron (MLP)
  - A type of neural network that consists of fully connected neurons with non linear activations functions
- Layers
  - A set of neurons in a neural network
  - Three common types are:
    - input (features),
    - hidden (finds nonlinear relationships between feature and label),
    - output(the predictions for the target)
- Neurons
  - A distinct unit in a neural network
  - Performs a two step action
    - Weighted sum of inputs from previous layer
    - Passes weighted sum into an activation function
- Activation Functions
  - A function that enables neural networks to learn non-linear relationships

Morning Outline

- Introduction on basic ML/DL models
  - Introduction Deep Learning
  - **Review of Supervised Learning**
  - **Multi-layer Perceptron (MLP)**
    - Architecture
    - **Training**
  - Convolutional Neural Network (CNN)
    - Residual Networks
    - Transfer Learning
- Case study: Natural Hazard Detection
  - Hands-on session

Loss Functions

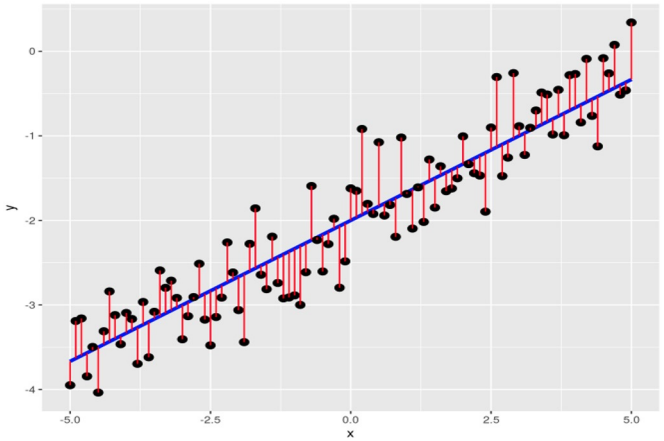
Common Loss Function for Regression

Least absolute deviations

$$\frac{1}{N}\sum_{i=1}^{N}|f(x_i,\theta)-y_i|$$

Ordinary Least Squares

$$\frac{1}{N}\sum_{i=1}^{N}(f(x_i,\theta)-y_i)^2$$



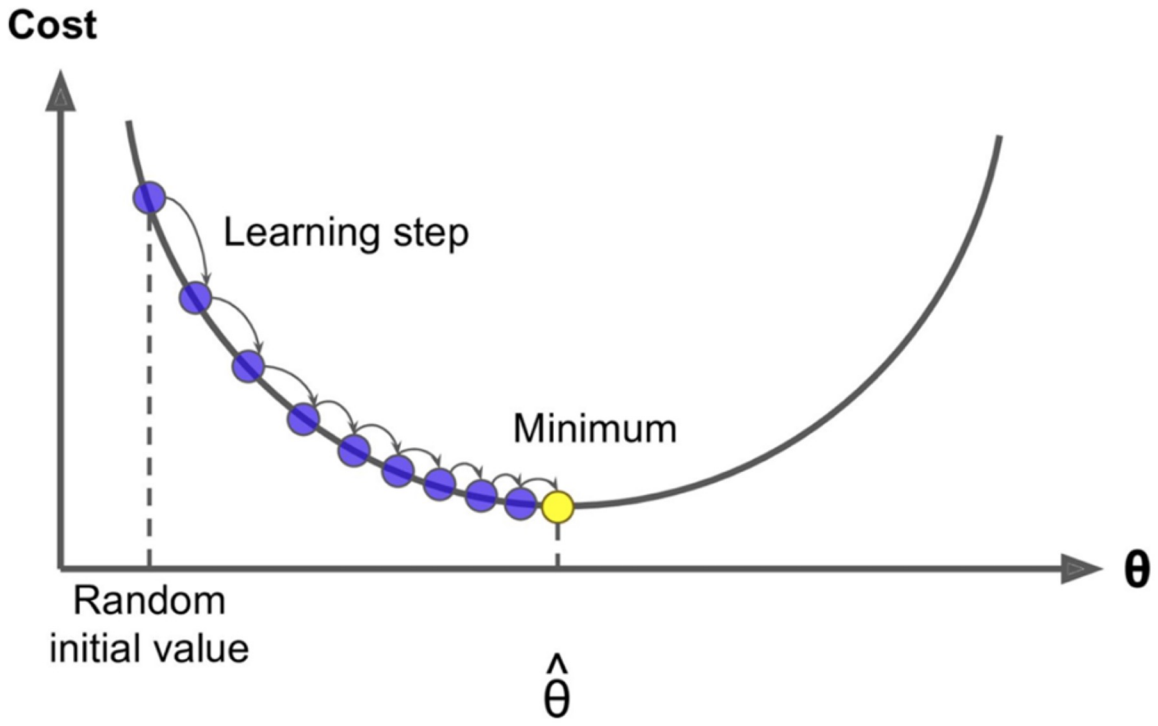
Common Loss Function for Binary Classification

$$\frac{1}{N}\sum_{i=1}^{N}y_i\log(p_i)+(1-y_i)\log(1-p_i)$$

- We can think of a loss as an average per data point loss
- $L(x,y,\theta)=\frac{1}{N}\sum_{i=1}^{N}l(x,y,\theta)$
- Where
  - N is the total number of datapoints in the training set
  - l is a per datapoint loss

Training Models by Minimize “errors”

- A basic idea in many ML is to minimize loss function
- Numerical optimization methods like gradient descent are commonly used to find optimal values for θ



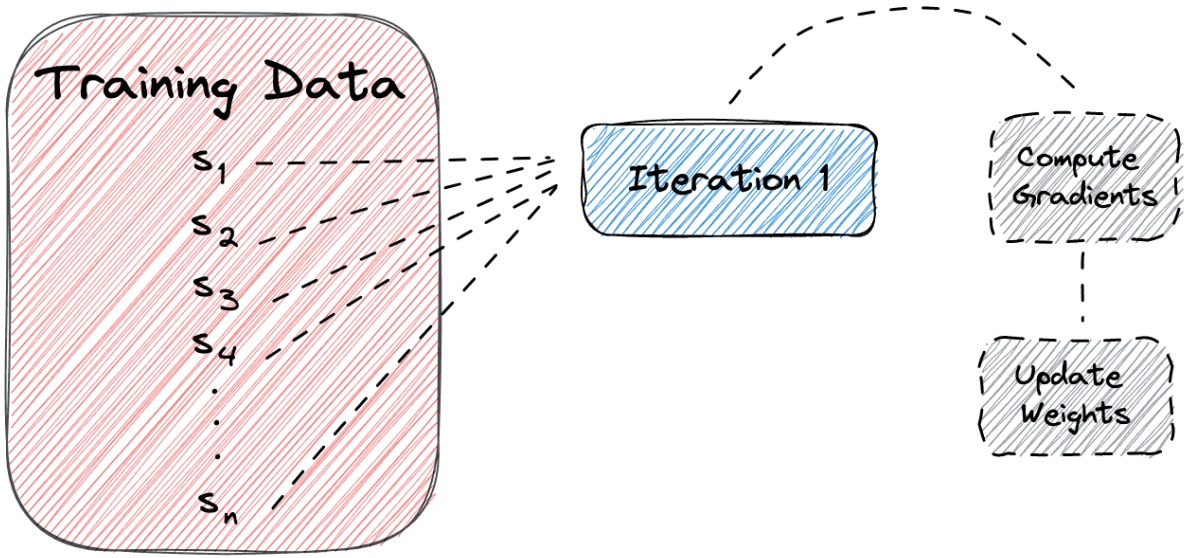
Training: Gradient Descent

Pseudocode for Gradient Descent

For number of iterations:

Compute Gradients: $\mathrm{g} = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta l(x, y, \theta)$

Update weights: $\theta_{i+1} = \theta_i - \alpha g$



Training: mini-batch GD

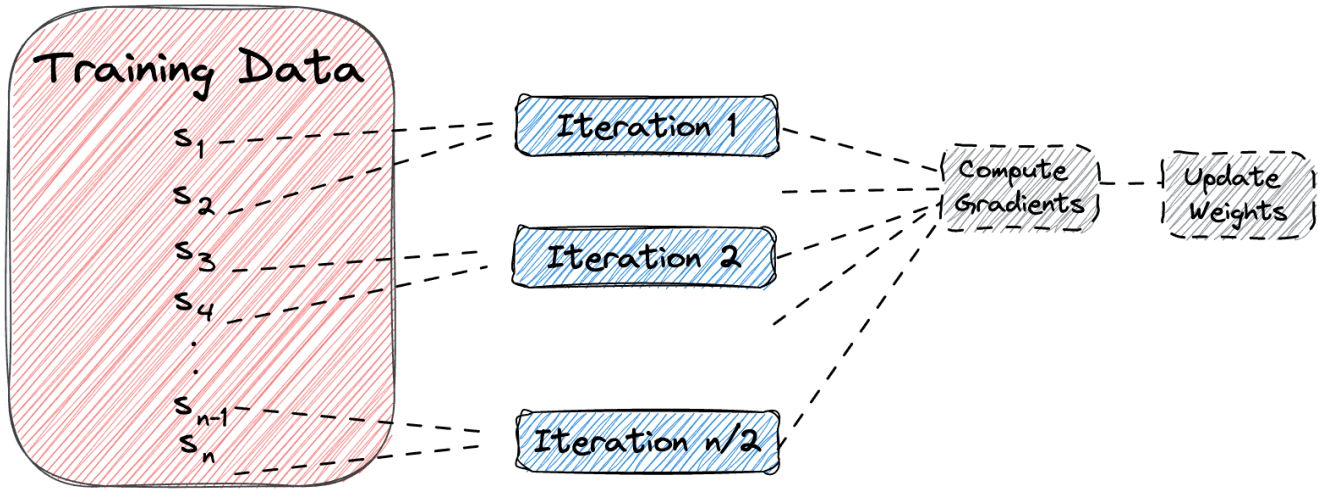
Pseudocode for mini-batch gradient descent

While stopping criteria has not met:

Compute Gradients: $g = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta l(x, y, \theta)$

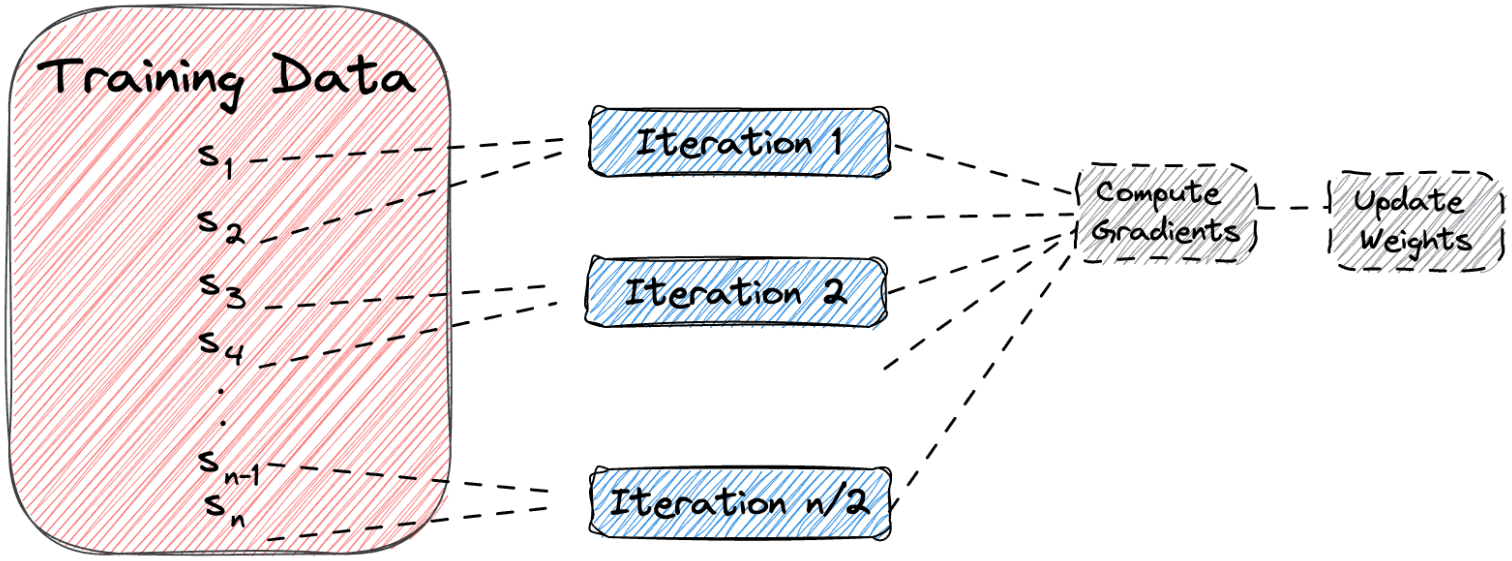
Update weights: $\theta_{i+1} = \theta_i - \alpha g$

- *n is a number smaller than N*
- *We approximate gradients with a smaller sample to make the per iteration cost cheaper*



Batch Size and Epochs

- When using mini batch gradient descent, we need to specify the **batch size** or**,** the number of data points used to approximate gradients
- The number of mini batch iterations needed to sample the entire training set is an **Epoch**
- So for 1,200,000 images with a 512 mini-batch size, an epoch roughly take 2,400 iterations



Training: Gradient Descent

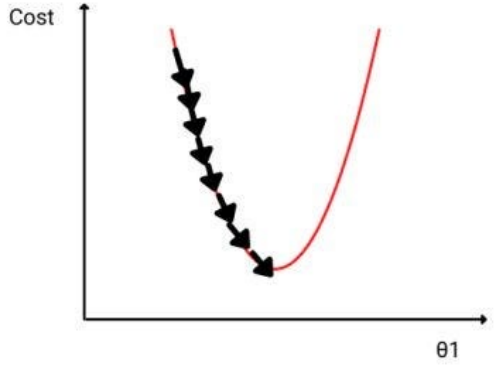
Pseudocode for Gradient Descent

While stopping criteria has not met:

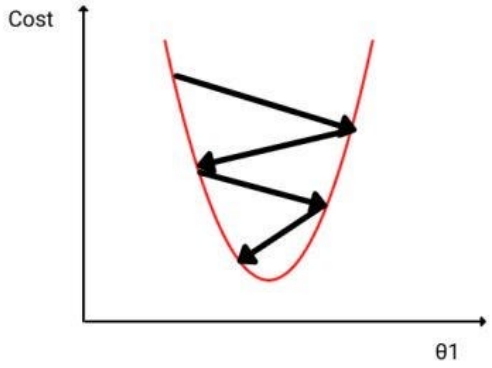
Compute Gradients: $g = \sum_{i=1}^{N} \nabla_\theta l(x, y, \theta)$

Update weights: $\theta_{i+1} = \theta_i - \alpha g$

α is the learning rate



For very small "α" gradient descent will be very slow in achieving the minimum



For very large "α" gradient descent may fail to reach the minima

Backpropagation

- Fortunately, manual derivation for gradients of different neural networks is not needed.
- All ML/DL frameworks (e.g. PyTorch and Tensorflow) have auto-derivation engine, which is usually implemented based on computation graph and backprop.

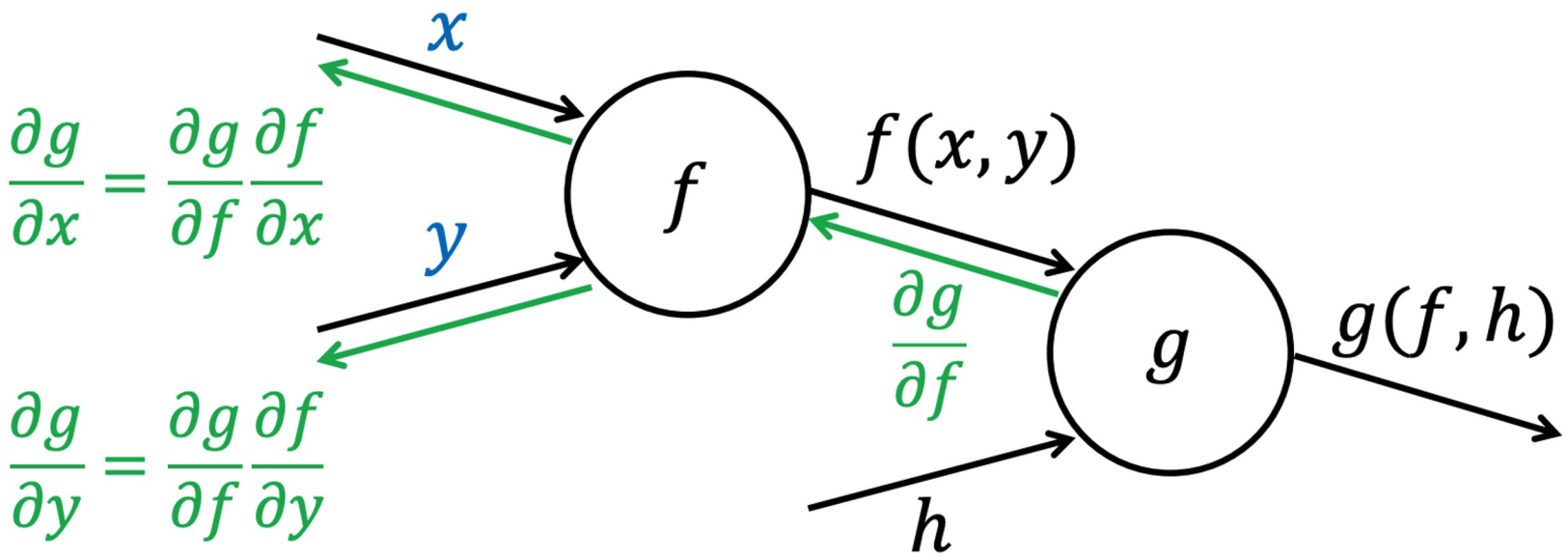


Image credit: Cornell ECE 5775 lecture14

Deep Learning Terminology

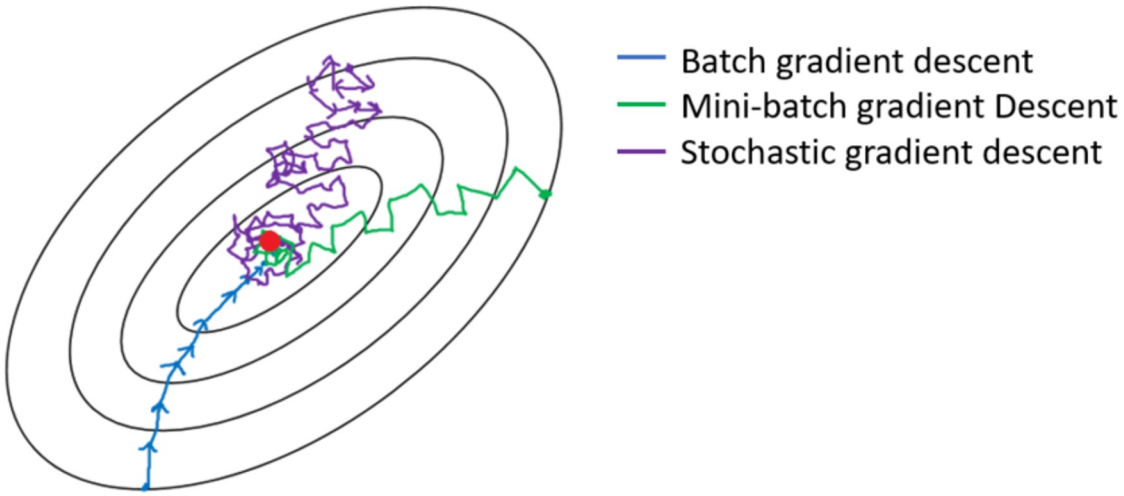
- Gradient Descent
  - Optimization algorithm used for minimizing loss functions in neural networks
  - Mini batch gradient descent uses an approximated gradient with a small subset of your data
- Backpropogation
  - A technique to compute the gradients of neural network in a feedforward network
- Learning Rate
  - A hyperparameter used in training neural networks that controls how weights are updated in gradient descent
  - If too high, gradient descent may not converge. If too low, convergence will be slow
- Batch Size
  - The number of datapoints used to approximate the gradient in one interation of mini batch gradient descent.
- Epoch
  - A full training pass over the entire dataset

A few other comments on training deep learning models…

- Learning Rate Decay
- Other Optimization Algorithms
- Regularization
- Batch Normalization

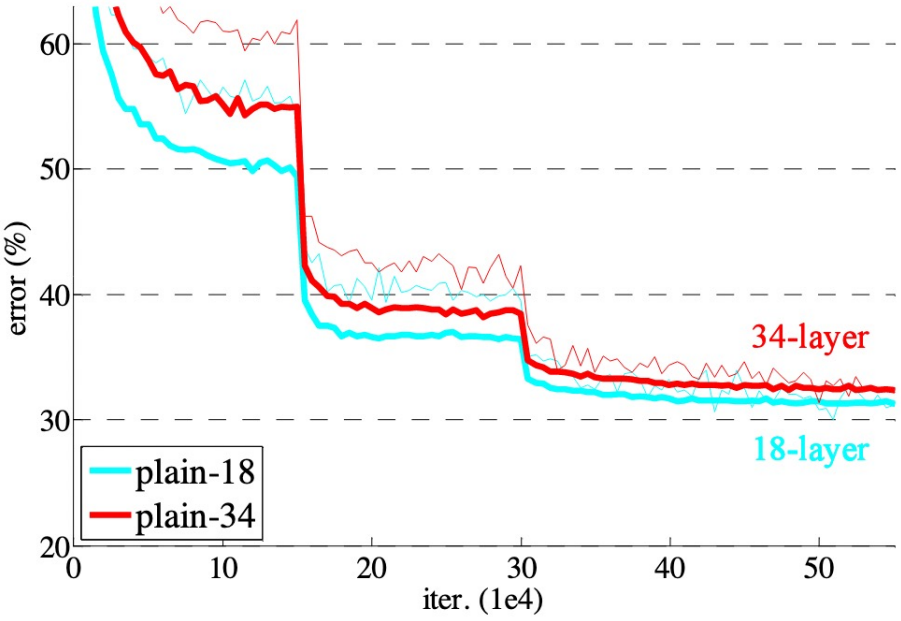
Learning Rate Decay

- When using mini-batch gradient and stochastic gradient descent (batch size=1), more iterations are required because of noisy gradient
- When we get close to a minima, the gradients are so noisy that the algorithm will fail to reach the minima
- **Solution**: make your learning rate smaller



Learning Rate Decay

- **Solution**: make your learning rate smaller
  - Identify when learning as plateaued (i.e. test accuracy is not decreasing)
  - Lower learning rate by ~ an order of magnitude

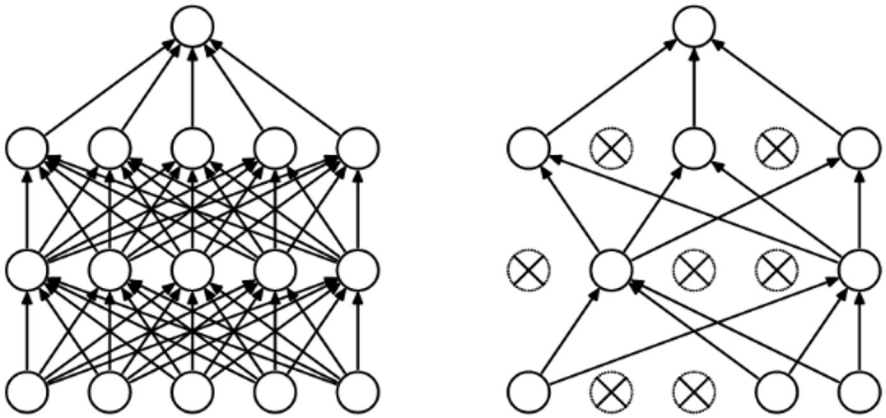


Alternative Optimizers

- Mini batch gradient descent is a good optimizer for training deep learning models but there are other optimization algorithms you could choose
  - Adam
  - RMSprop
  - Second Order Methods
  - Review paper: https://arxiv.org/abs/2211.15596
- All terminology introduced (batch size, epoch, learning rate, etc.) apply to these optimizers

Dropout Layer

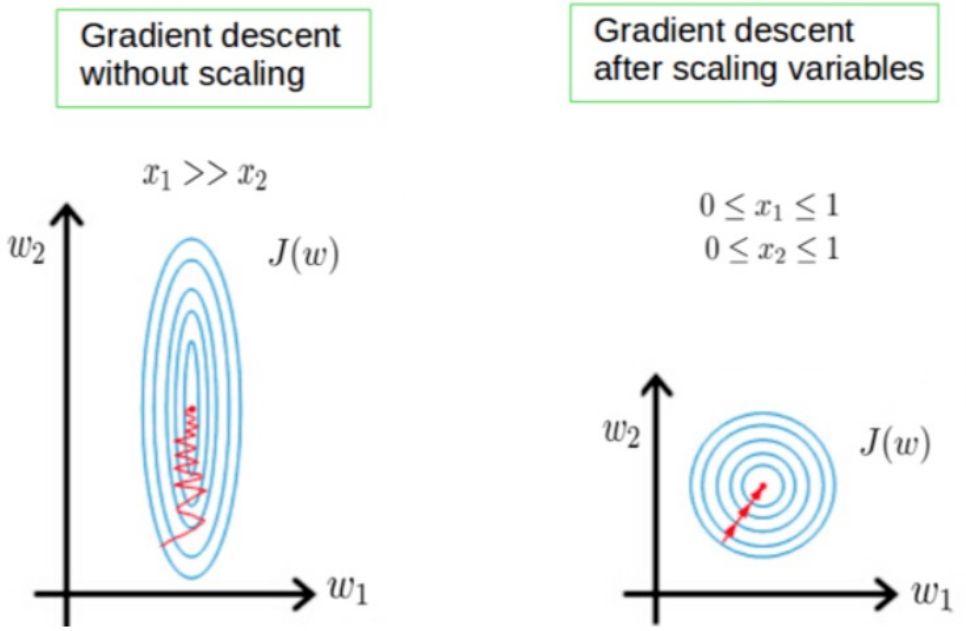
- Dropout is a popular regularization technique used to avoid overfitting
- Given a probability of dropout, it randomly drops neurons in a fully connected layer
- When we train the model, if a neuron is selected by the dropout layer, it will not contribute to the following layers, its gradient will not be computed, and its parameter will not be updated.
- Tends to make more generalized predictions because the model needs to adapt to missing data



(a) Standard Neural Net (b) After applying dropout.

Batch Normalization

- Deep MLP can be difficult to train.
- Parameters at different dimension can have very different gradient.
- This can be improved by scaling/normalizing the data



Batch Normalization

- An extension of the idea of normalizing data to increase training speed, but applied to deep learning models
- As data propogates through the network, the activations in hidden layers will no longer be normalized.
- **Batch Normalization** aims to correct for this by normalizing inputs of each layer.
- See supplemental slides for more details

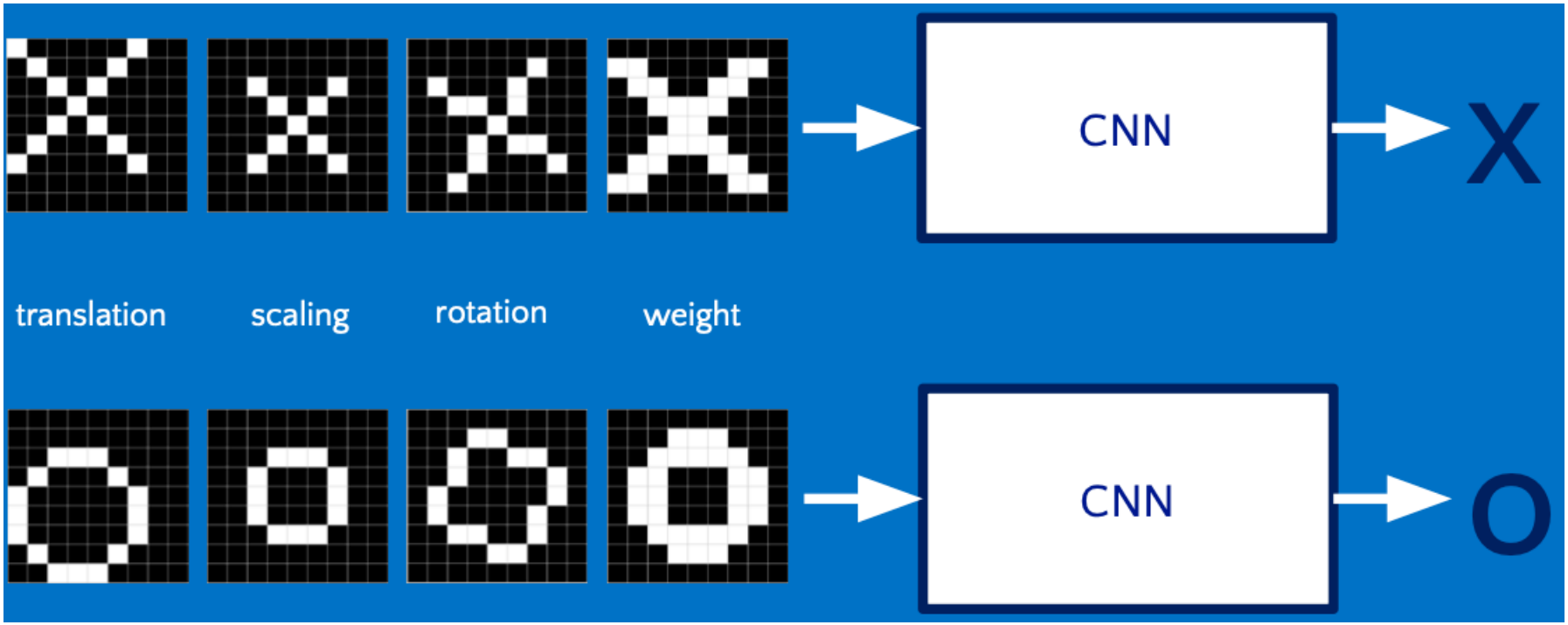
Morning Outline

- Introduction on basic ML/DL models
  - Introduction Deep Learning
  - Review of Supervised Learning
  - Multi-layer Perceptron (MLP)
    - Architecture
    - Training
    - Advanced Topics
  - **Convolutional Neural Network (CNN)**
    - **Fundamentals**
    - Residual Networks
    - Transfer Learning
- Case study: Natural Hazard Detection
  - Hands-on session

Convolutional Neural Network (CNN)

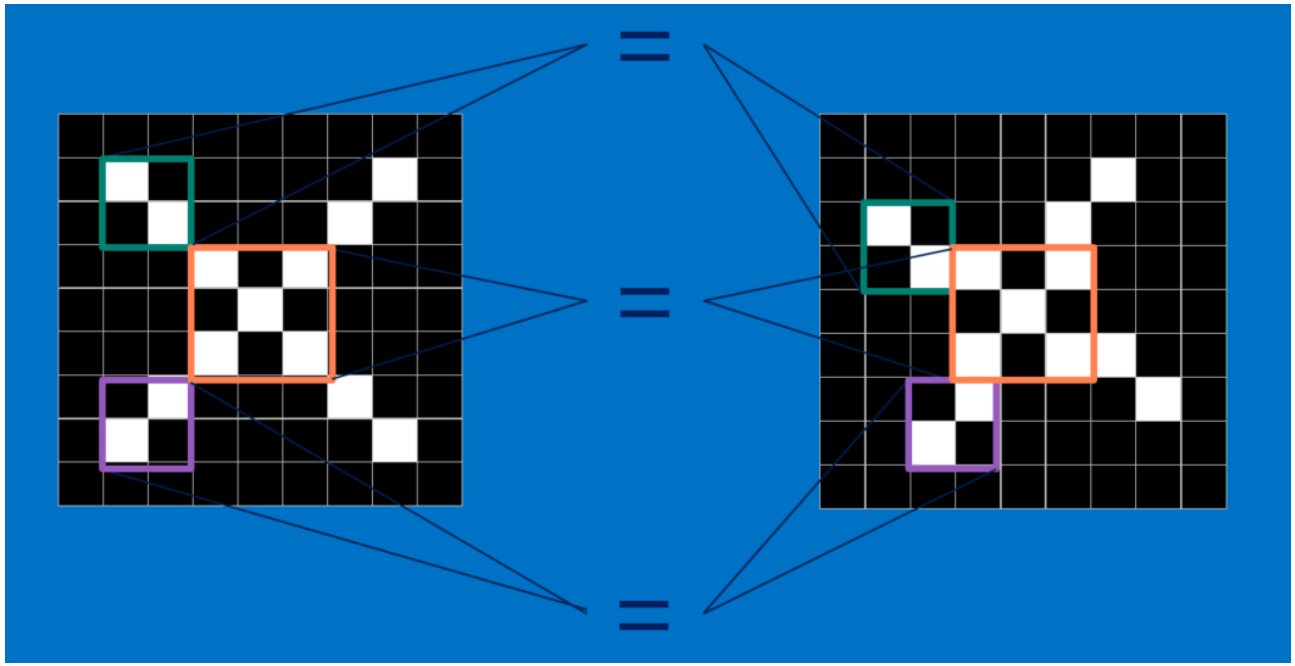
From MLP to CNN

- MLPs fail for complex image recognition tasks.
- Why?



https://brohrer.github.io/how_convolutional_neural_networks_work.html

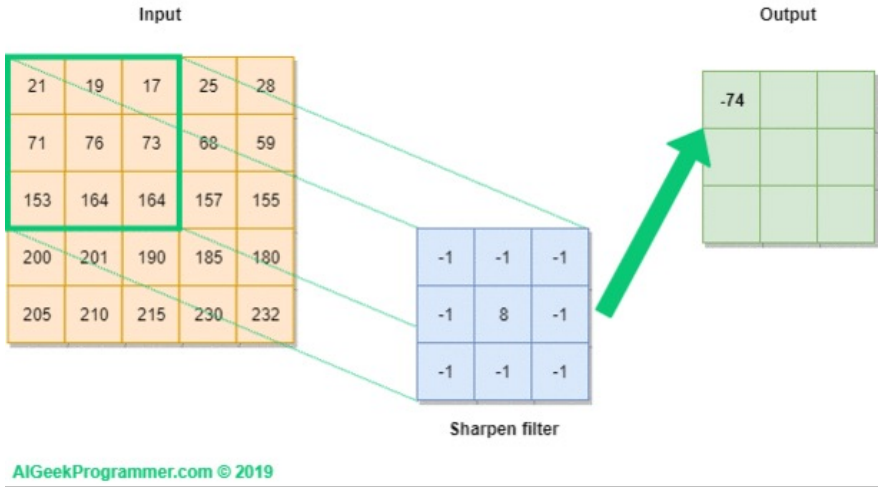
CNN match portions of an Image



https://brohrer.github.io/how_convolutional_neural_networks_work.html

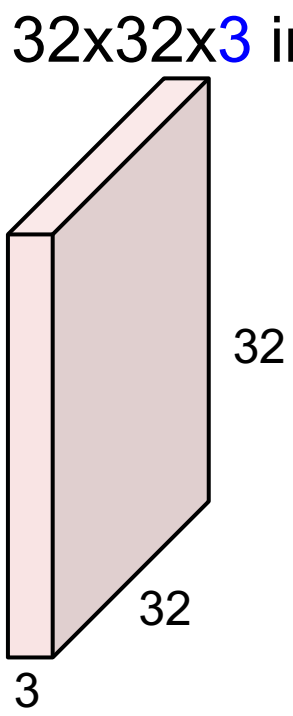
Filters for Edge Detection

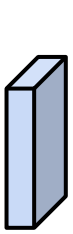
- Filters are applied to images to pick up on patterns that could exist any where on the image
- Filters do two things
  - They convolve over the entire image
  - At each location on the image they take a dot product



Convolution Layer

Filters always extend the full depth of the input volume



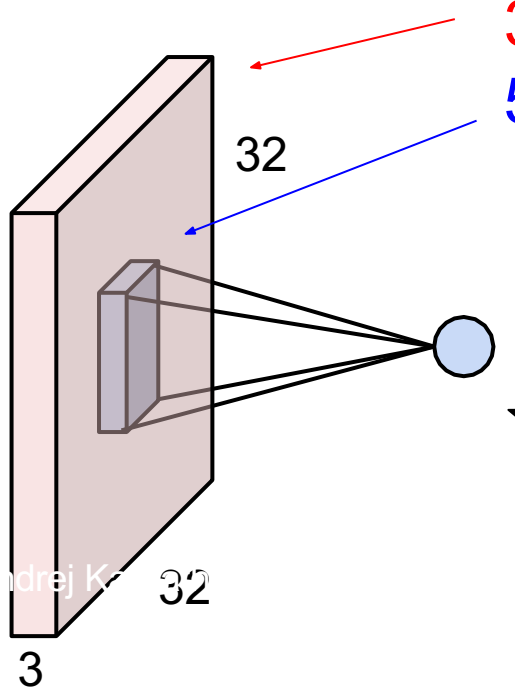


Convolve the filter with the image

i.e. “slide over the image spatially, computing dot products”

http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

Convolution Layer

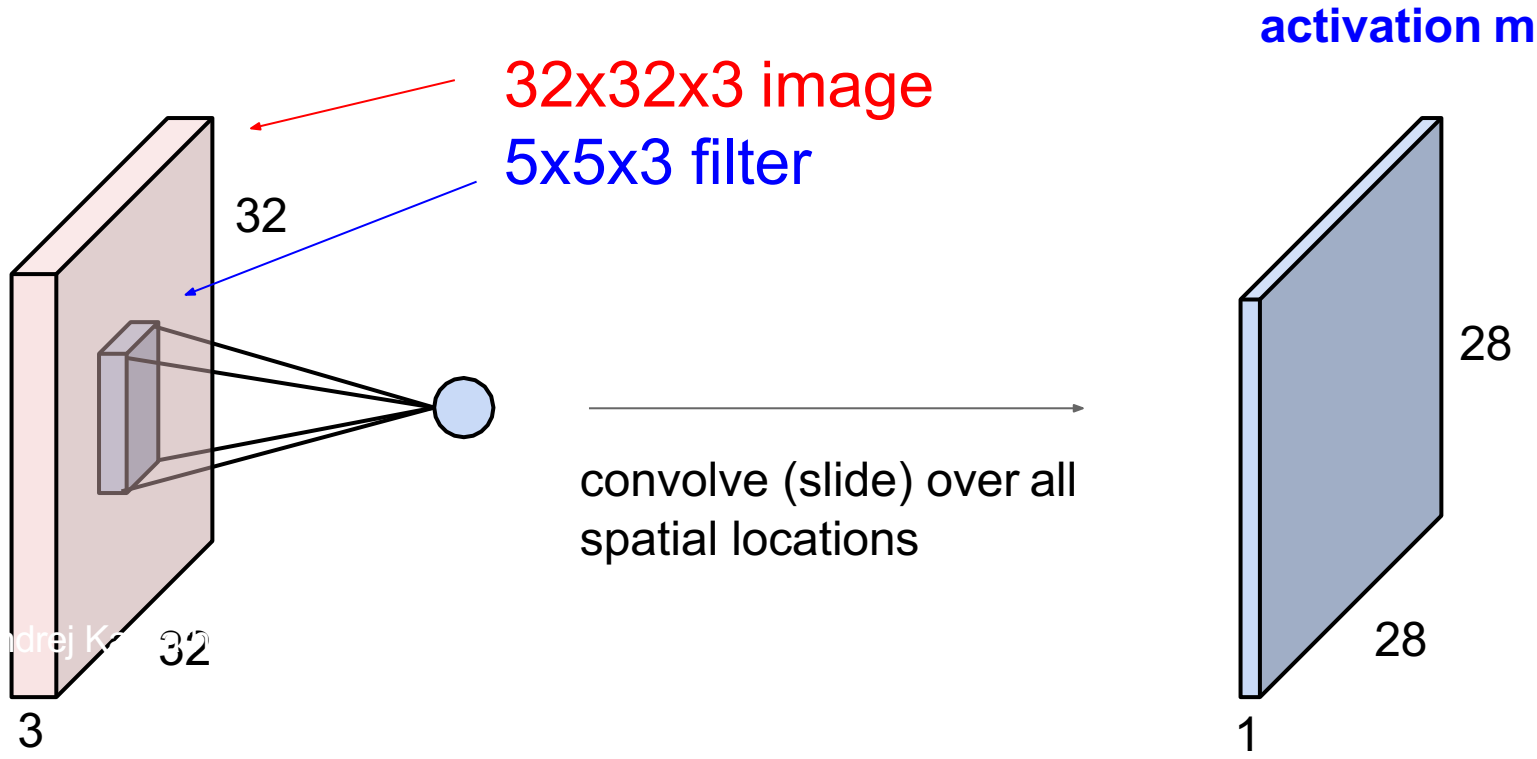


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

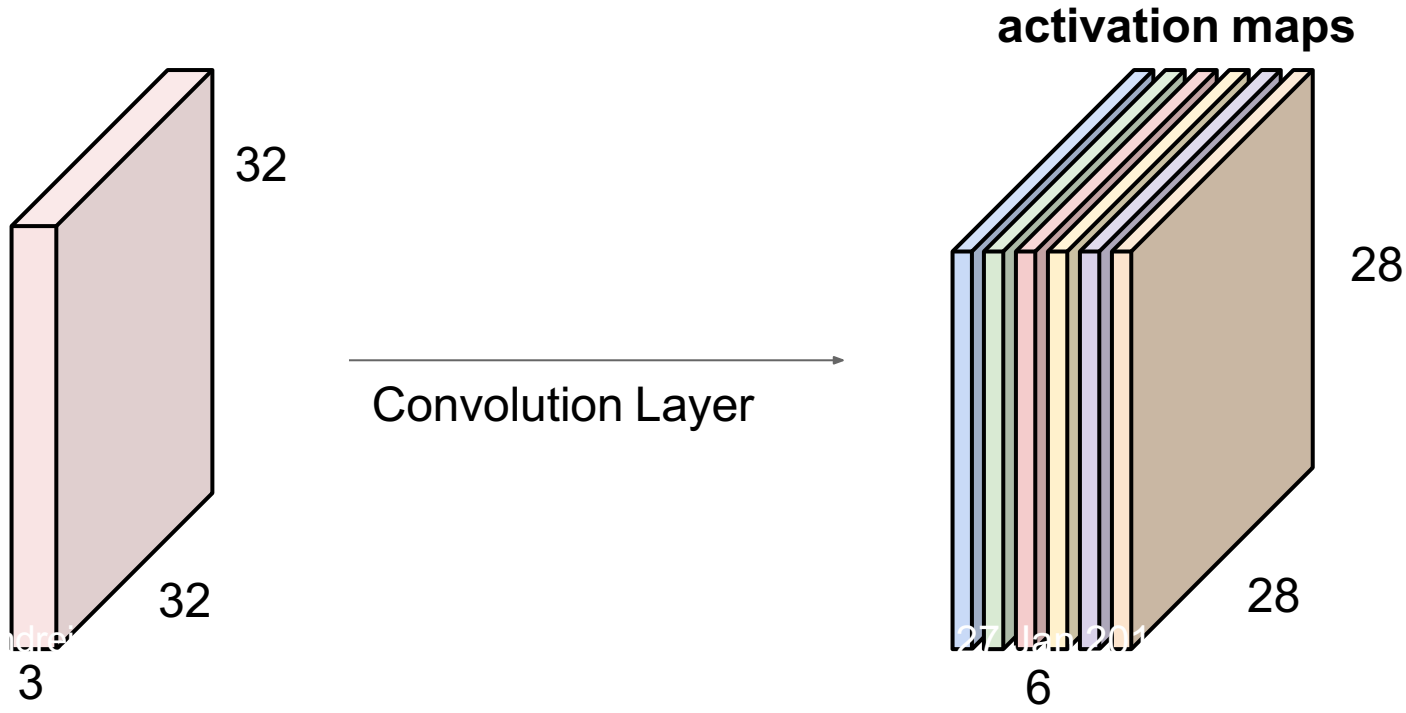
Convolution Layer



Convolution Layer

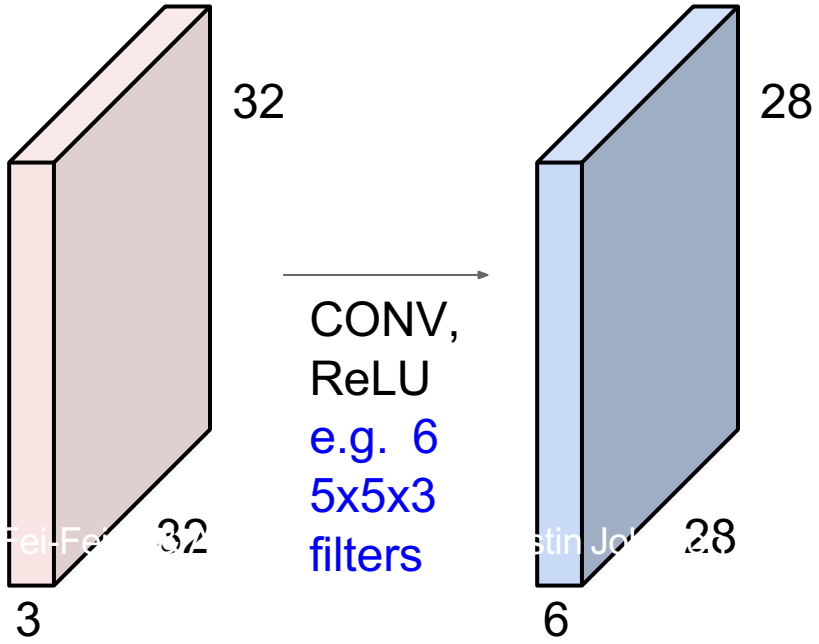
consider a second, green filter

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



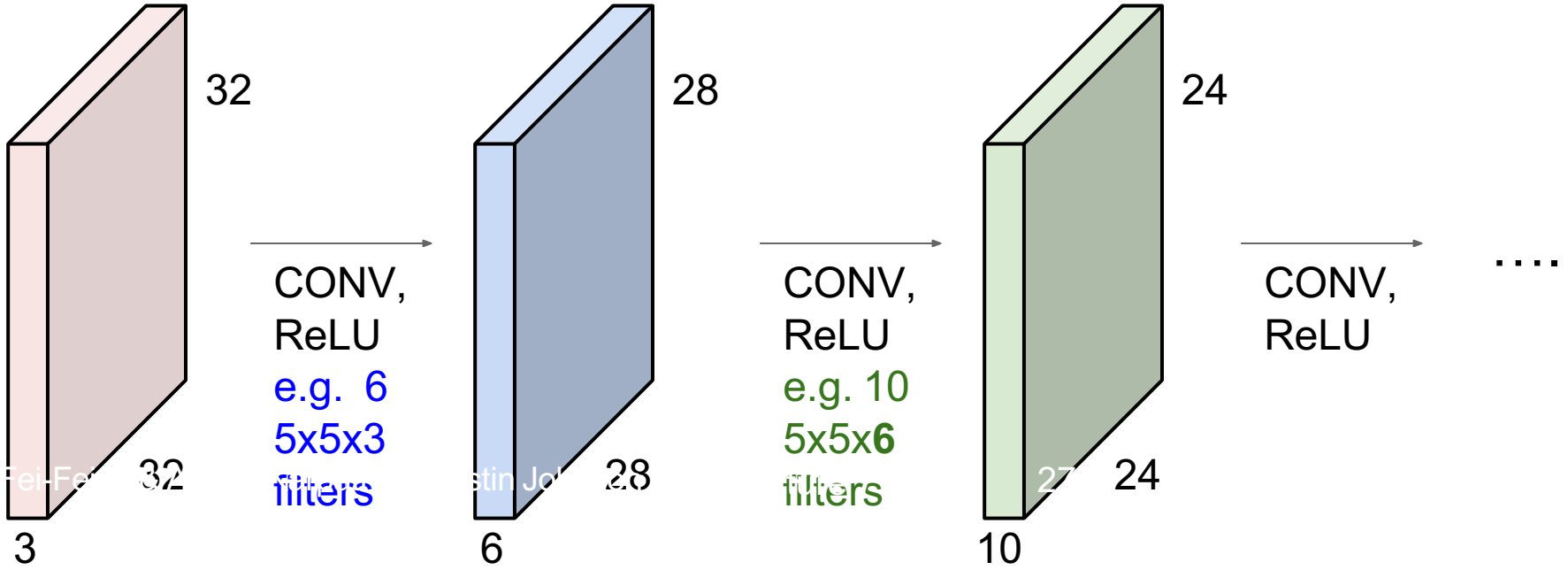
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

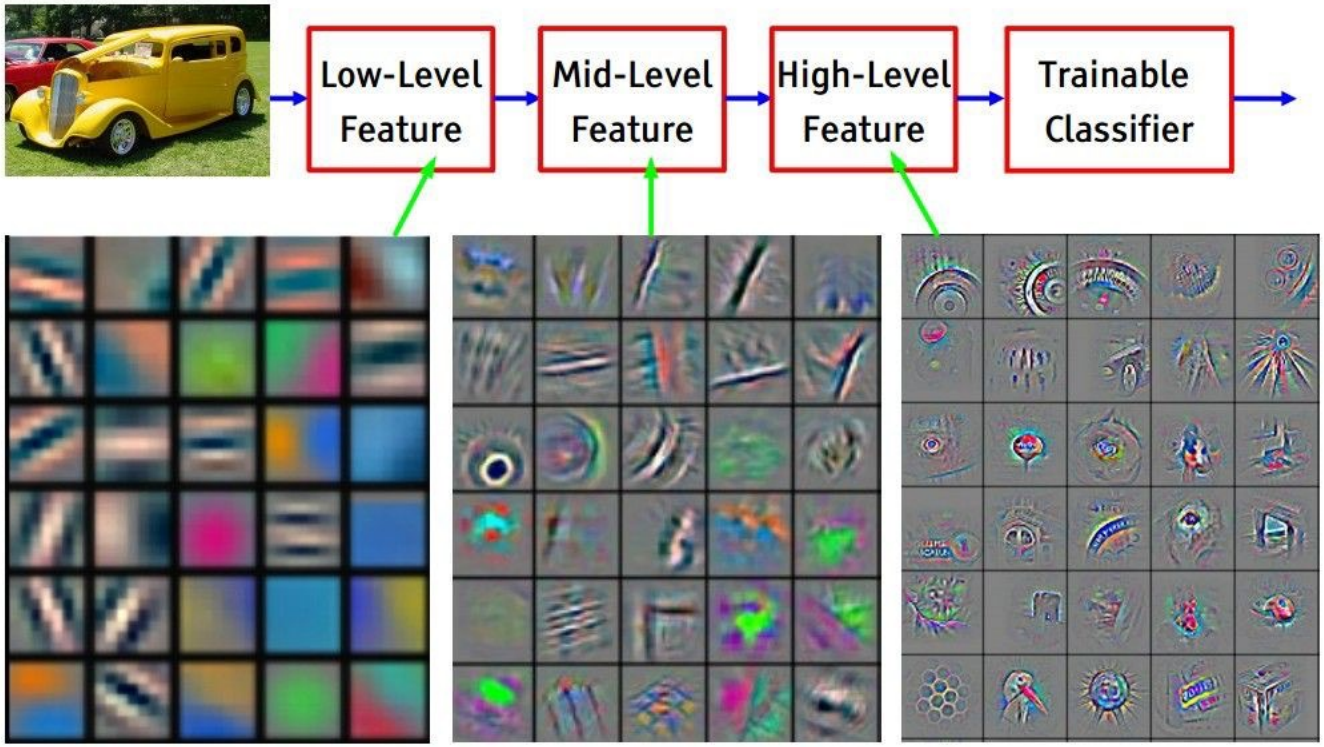


http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

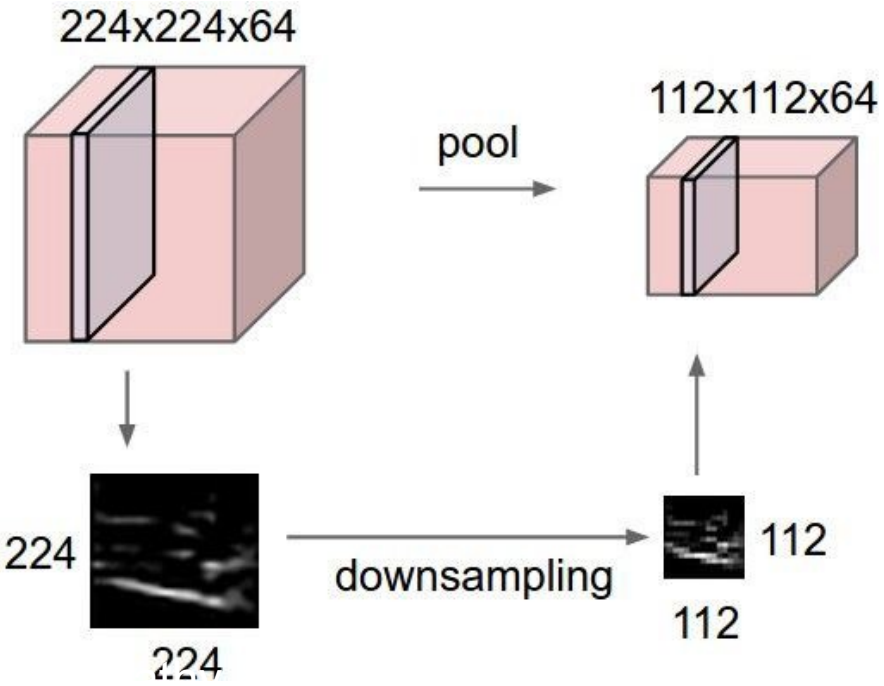


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

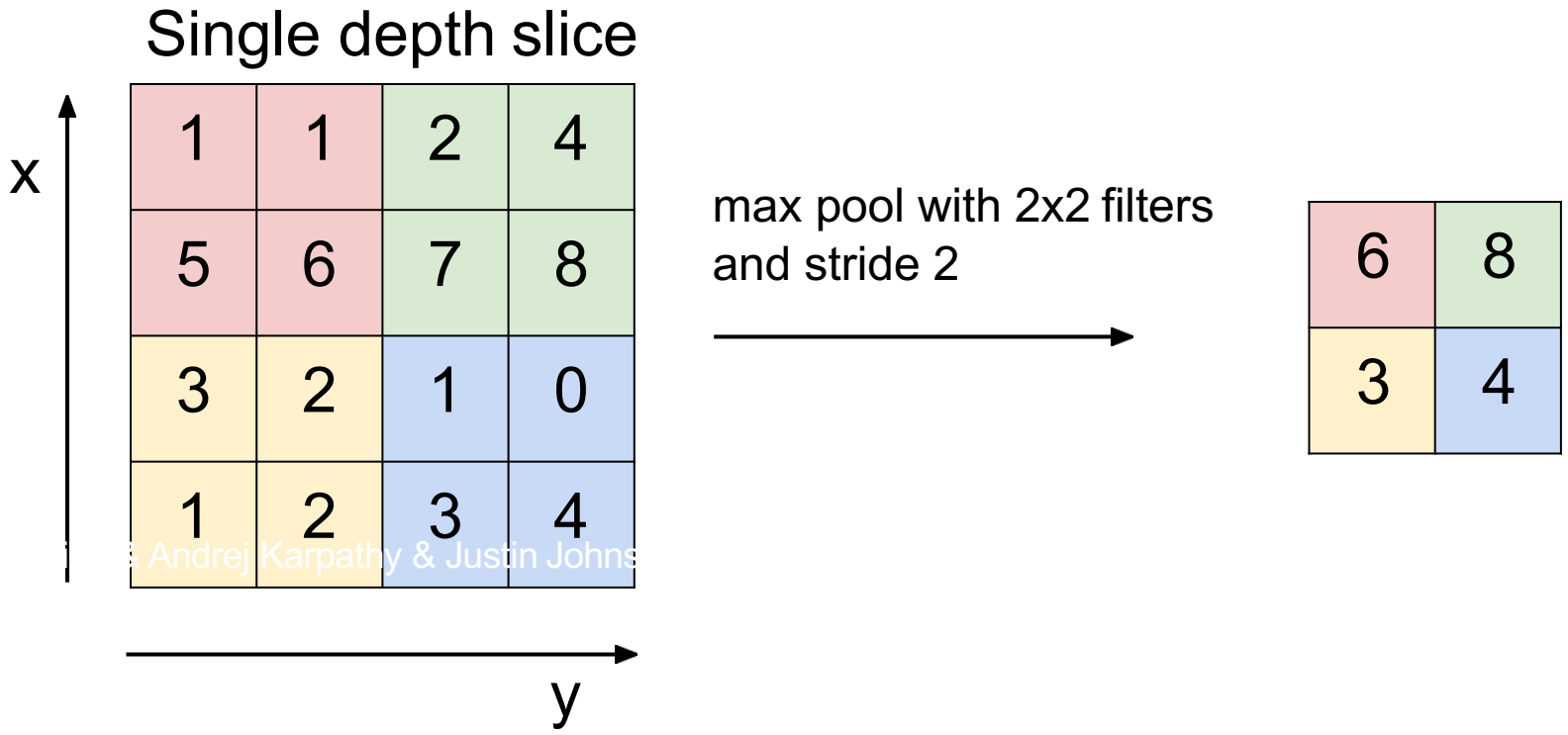
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

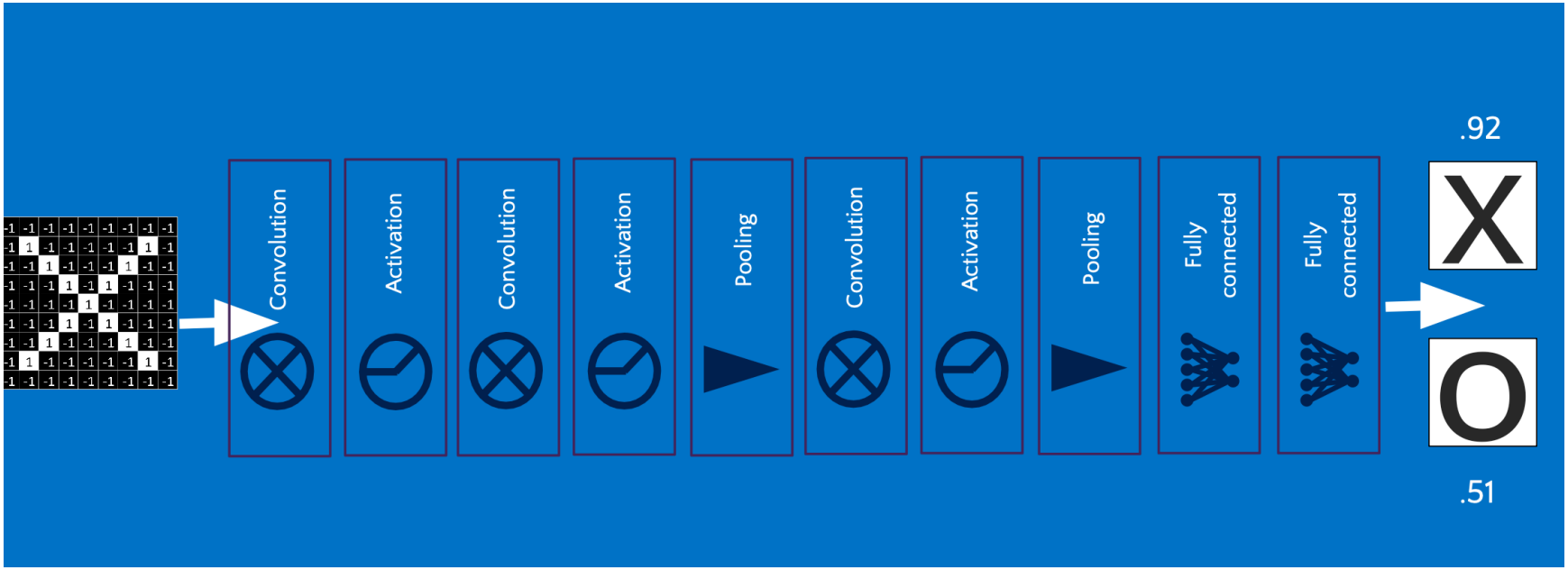


http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

MAX POOLING



CNN



Feature Engineering

Classifying newly engineered images

https://brohrer.github.io/how_convolutional_neural_networks_work.html

Deep Learning Terminology

- Convolutional Neural Network
  - A neural network with at least one convolutional layer
  - Typically consists of mulitple convolutional, pooling, and a fully connected layers
- Convolutional Layer
  - A layer in a convolutional neural network in which a filter is convolved around almost every possible location of an input matrix
- Filters
  - A matrix of weights that have the same depth as an image, but smaller dimension
- Pooling Layer
  - Operation that reduces matrix by performing some aggregate operation (min,max,etc.) over a pooled area

Issues with CNNs

- Deeper models needed to improve performances of CNNs
- Performance of deeper networks get worse
- This phenomena is NOT due to overfitting

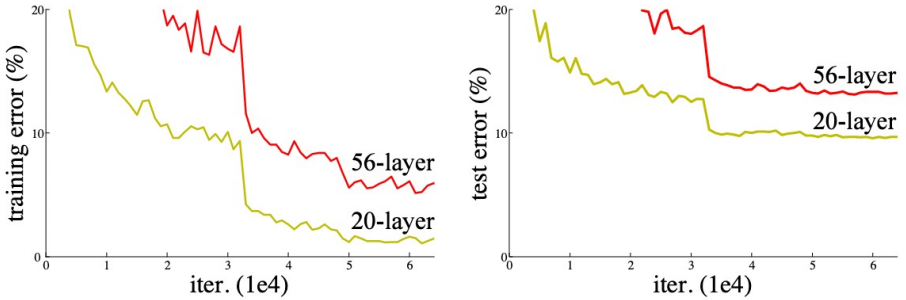


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

- Why? More challenging optimization problem

https://ieeexplore.ieee.org/document/7780459

Residual Learning

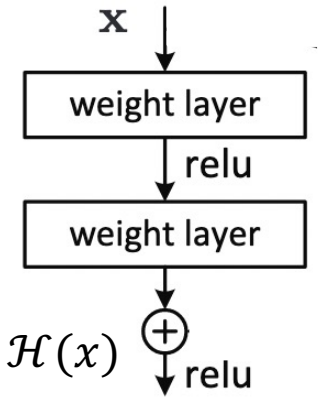


Figure 2. Residual learning: a building block.

Assume Dimensionality of x and H(x) are the same

https://ieeexplore.ieee.org/document/7780459

Residual Learning

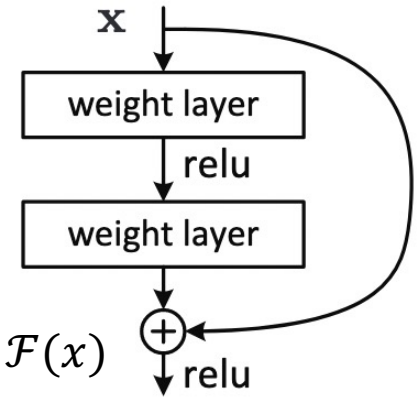


Figure 2. Residual learning: a building block.

Assume Dimensionality of x and H(x) are the same, have weight layers learn F(x) instead:

$$\mathcal{F}(x) \coloneqq \mathcal{H}(x) - x$$

https://ieeexplore.ieee.org/document/7780459

Residual Learning

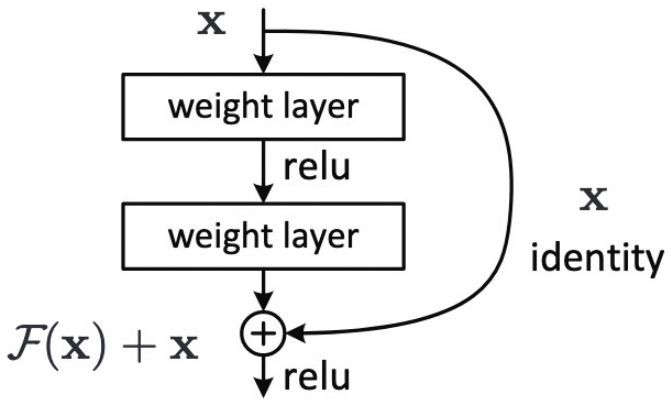


Figure 2. Residual learning: a building block.

Assume Dimensionality of x and H(x) are the same, have weight layers learn F(x) instead:

$$\mathcal{F}(x) \coloneqq \mathcal{H}(x) - x$$

Then pass x via a *shortcut connection or identity mapping* to the end of the residual layer to recreate $\mathcal{H}(\mathrm{x})$

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

X: Input of residual block
Y: Output of residual block
F: residual mapping to be learned

https://ieeexplore.ieee.org/document/7780459

Results: ImageNet

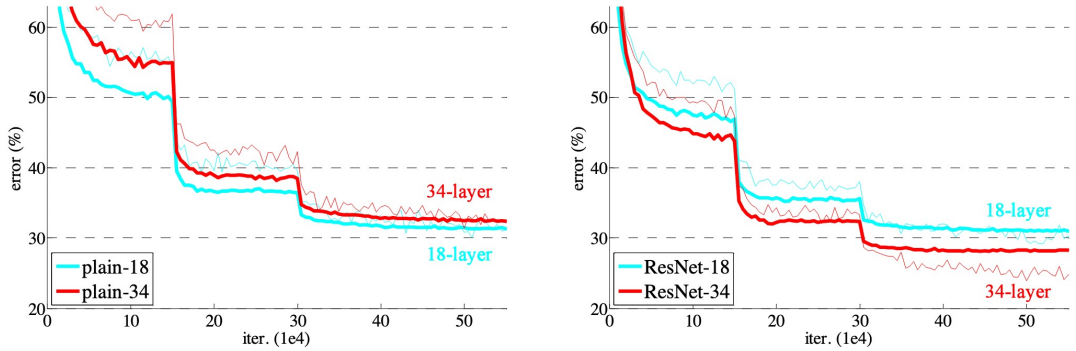
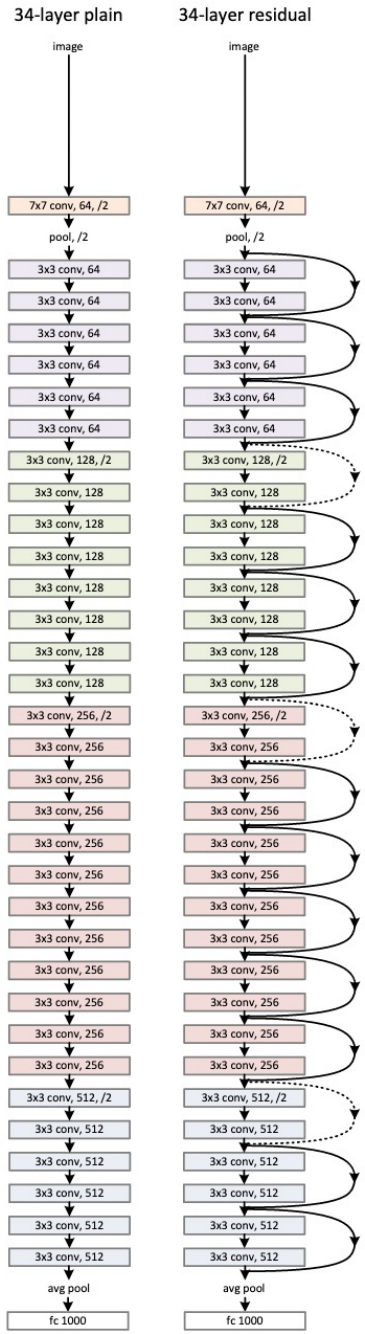


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.



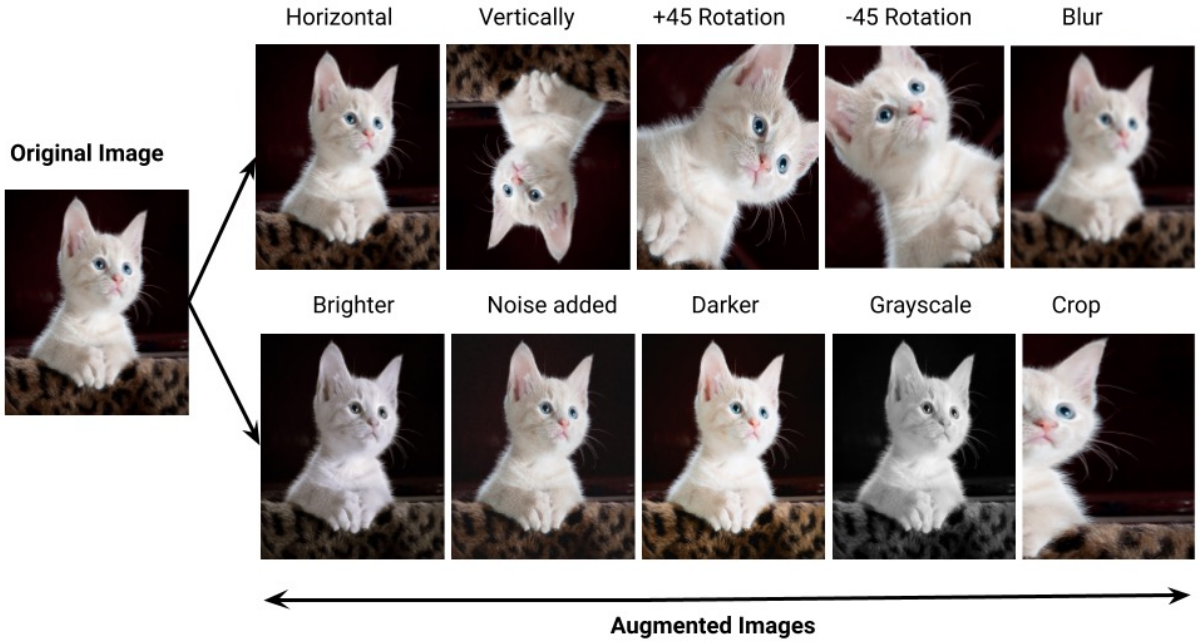
https://ieeexplore.ieee.org/document/7780459

A few other comments on CNN…

- Data Augmentation
- Transfer Learning

Data augmentation

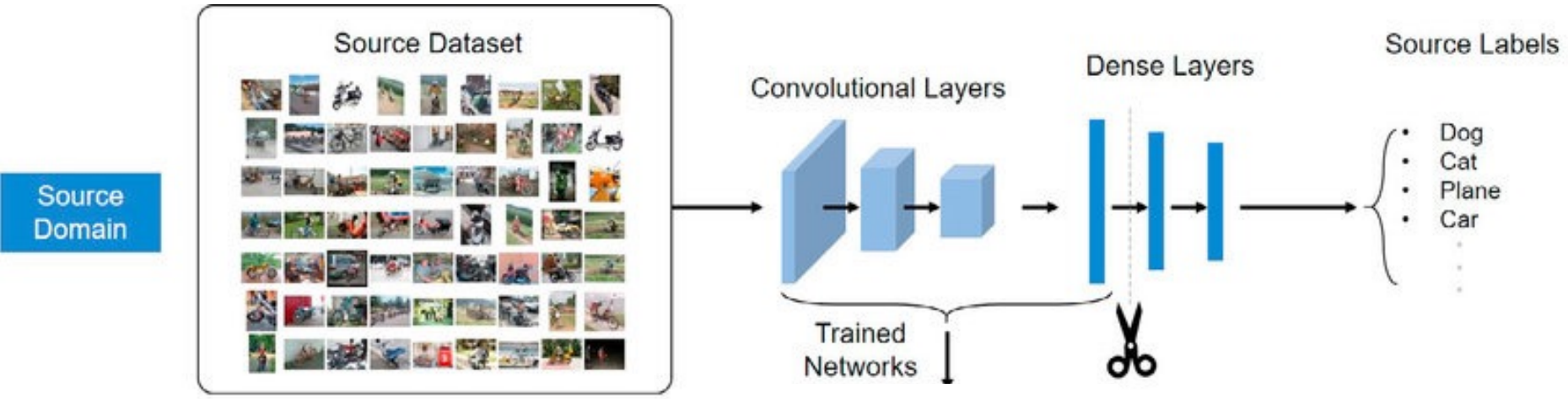
- Often times we do not have enough data to train CNN models, which can lead to overfitting a small dataset
- A simple solution is to just “make up” more data.
- Every time we sample an image from the dataset, we randomly perform some operations on them.
  - For instance, shift, rotate, horizontal and vertical flip, clip..



Transfer learning

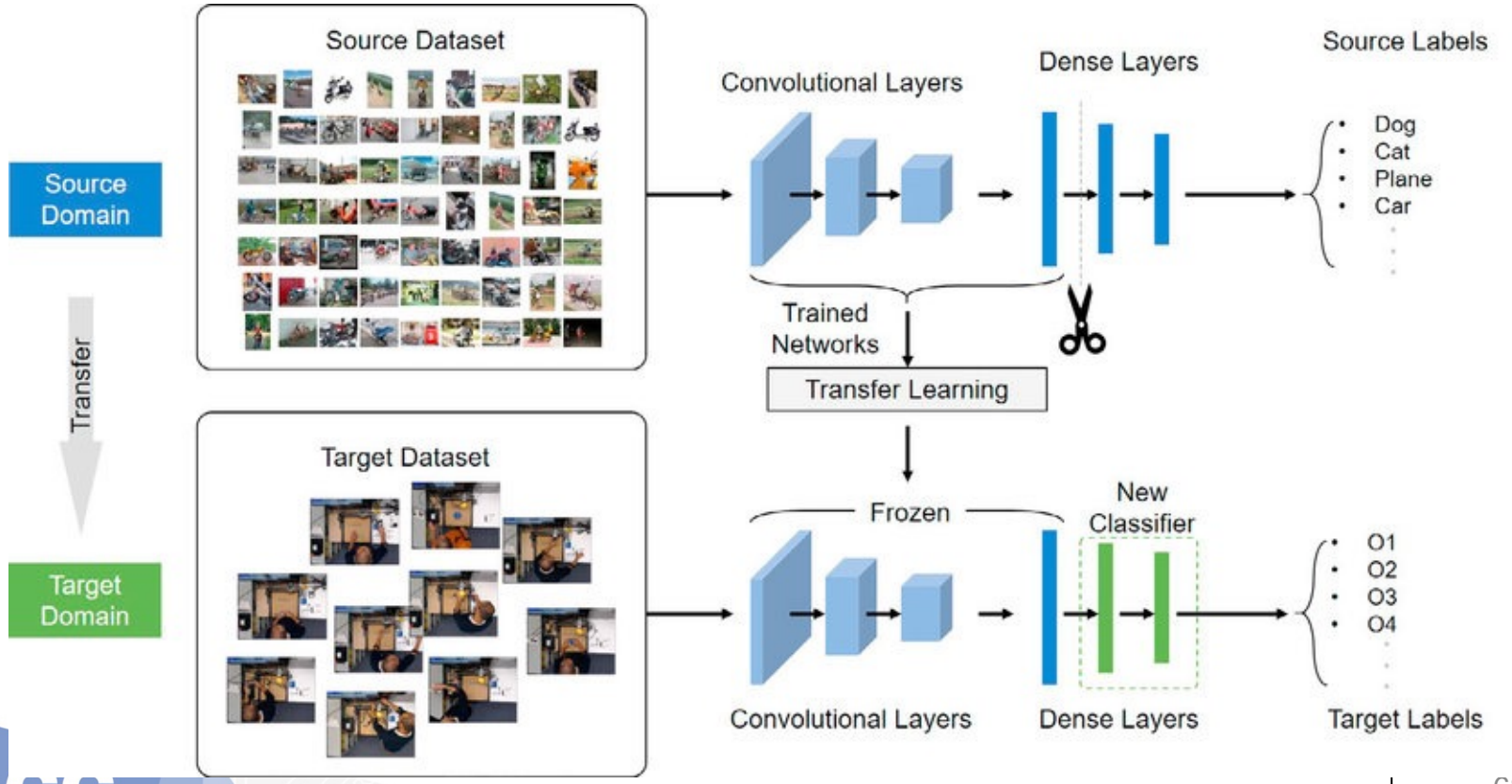
- Larger models tend to have better performance (assuming no overfitting)
- To avoid overfitting, large dataset and substantial computing time is needed to fit large CNN models
- For specific tasks, like natural hazard detection, we have much smaller datasets. This makes our models prone to overfitting.
- A work around is to use transfer learning.

Toy Model trained with ImageNet-1k



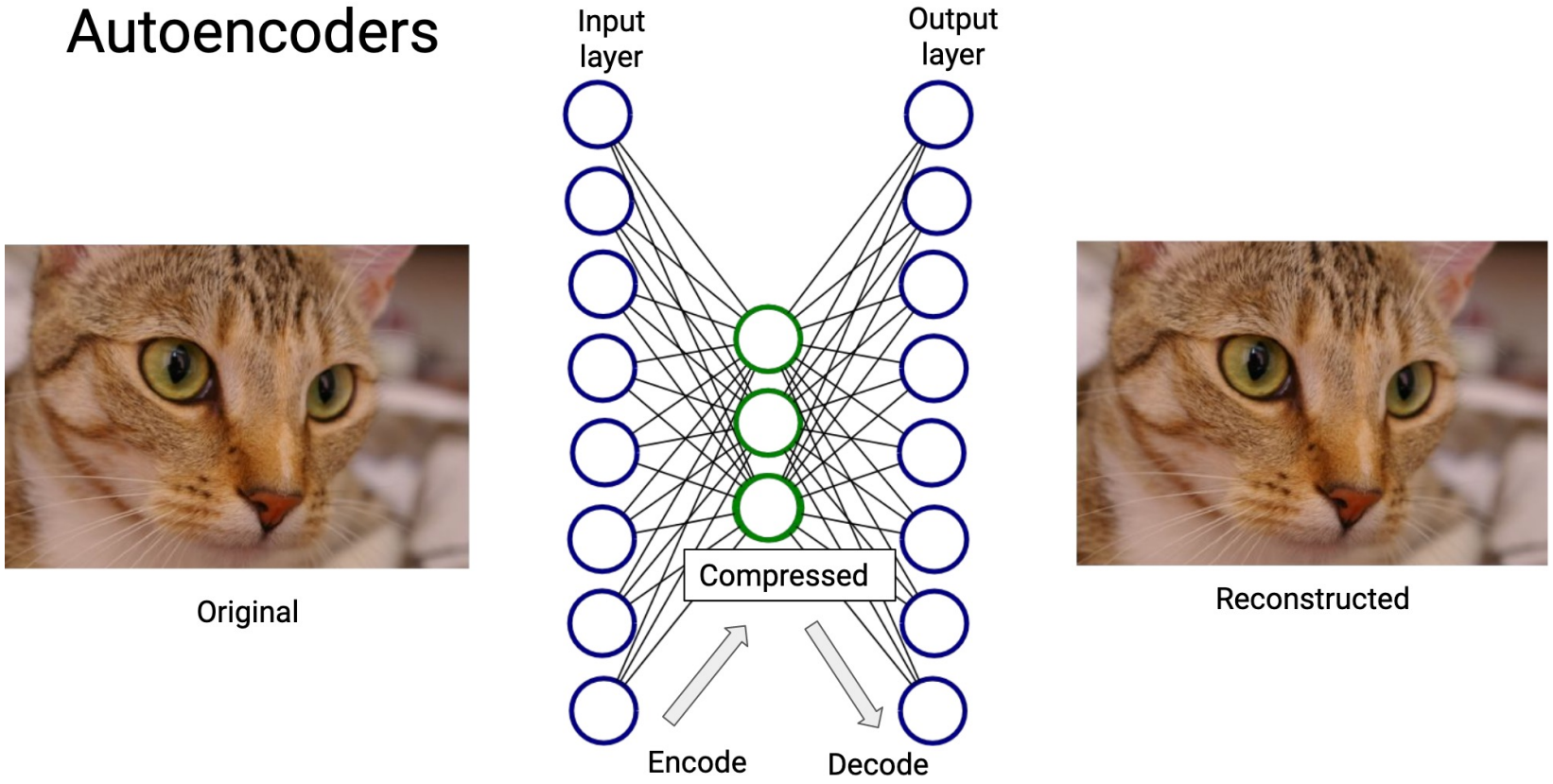
Transfer learning

- We can use large model and train it again on our small dataset.
- To adapt large model to our small dataset with fewer number of classes, modifications on the “classification” block is usually needed.



Extra Deep Learning Material…

Unsupervised Learning: AutoEncoders



With a standard autoencoder, we try to find compression and decompression models which simply result in the original image -- i.e. minimizing a loss function like mean squared error of the reconstructed image.

Hands On Exercises

Pytorch

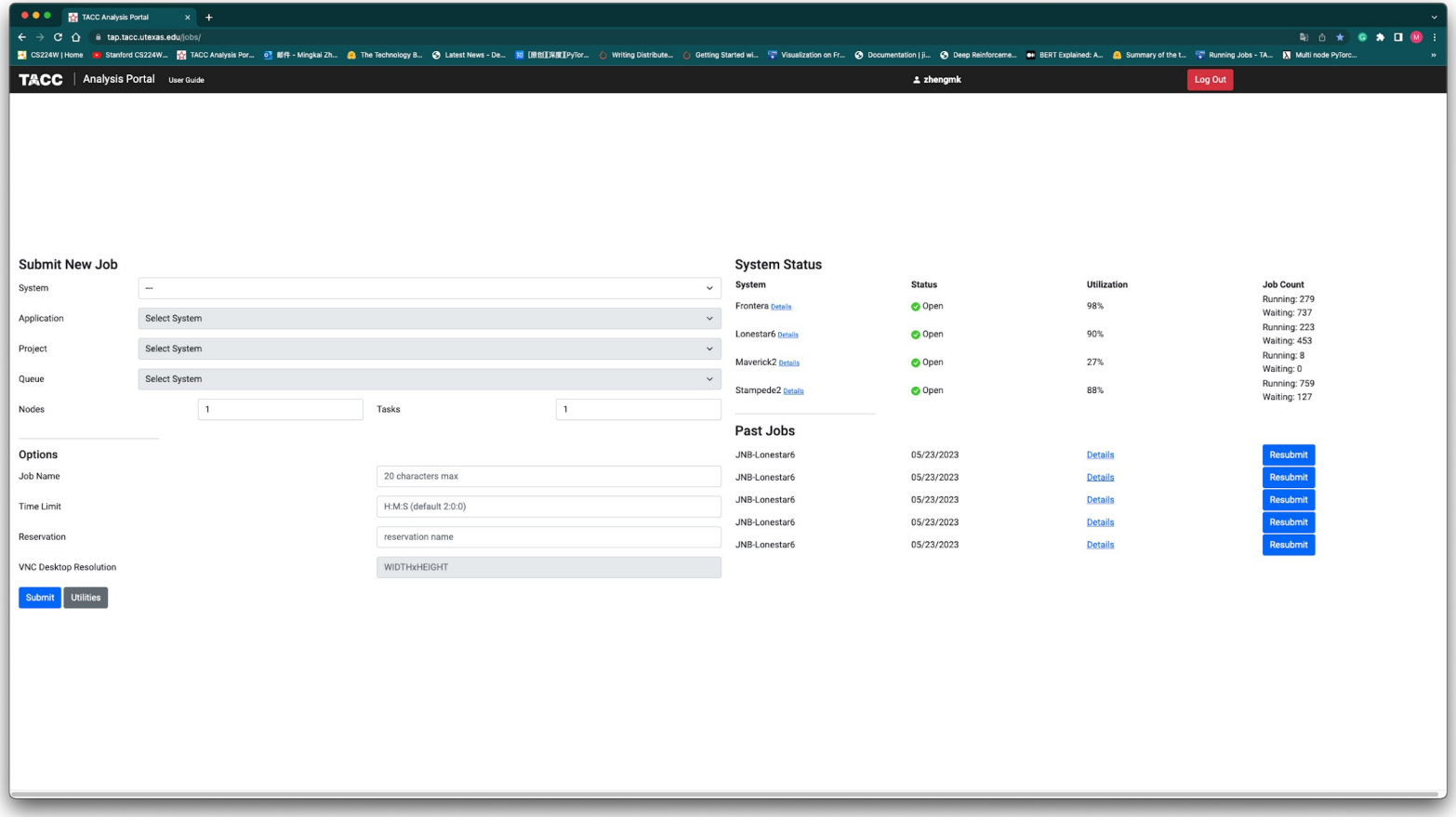
- Free and open source deep learning library
- Developed by Meta/Facebook
- Optimized to run on GPUs

Case Study: Natural Hazard Detection

How to acquire TACC resources

Use Analysis Portal

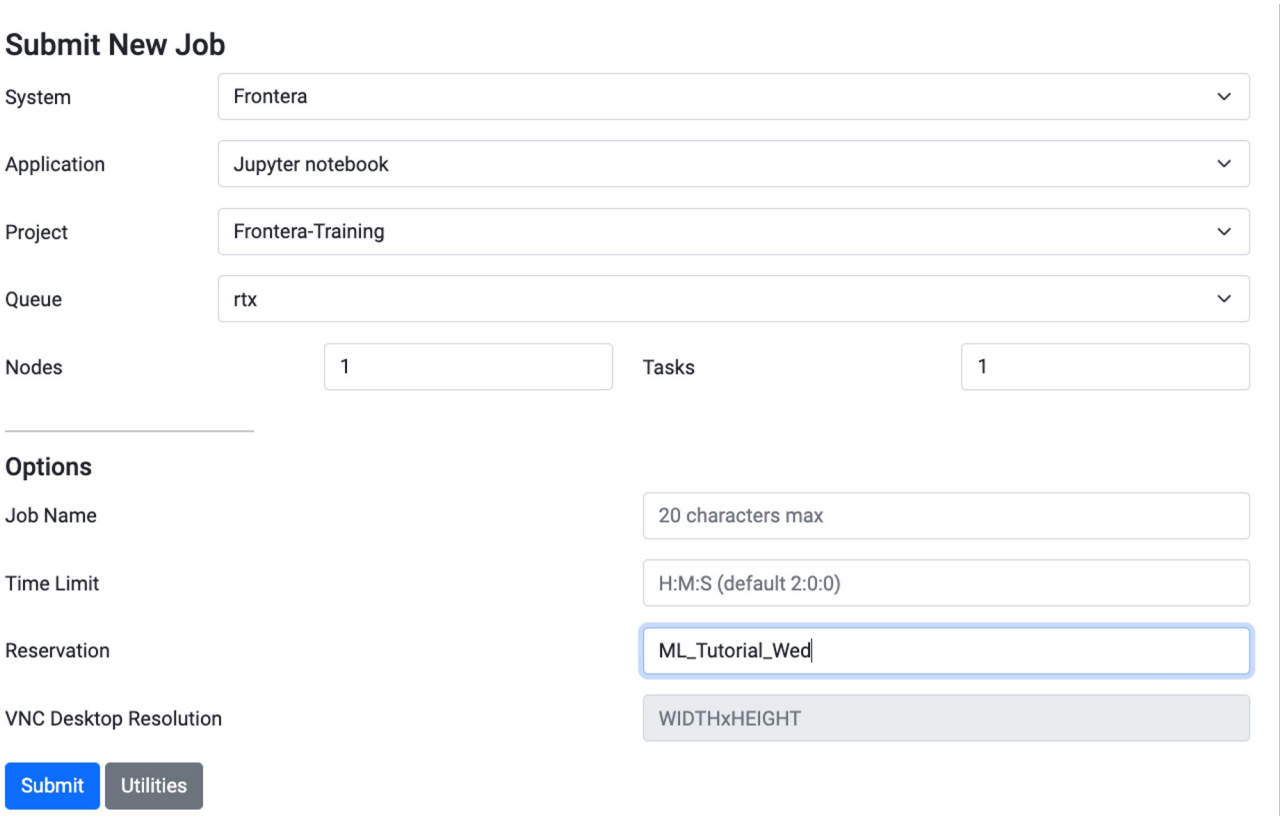
Login with your training account



How to acquire TACC resources

Use Analysis Portal

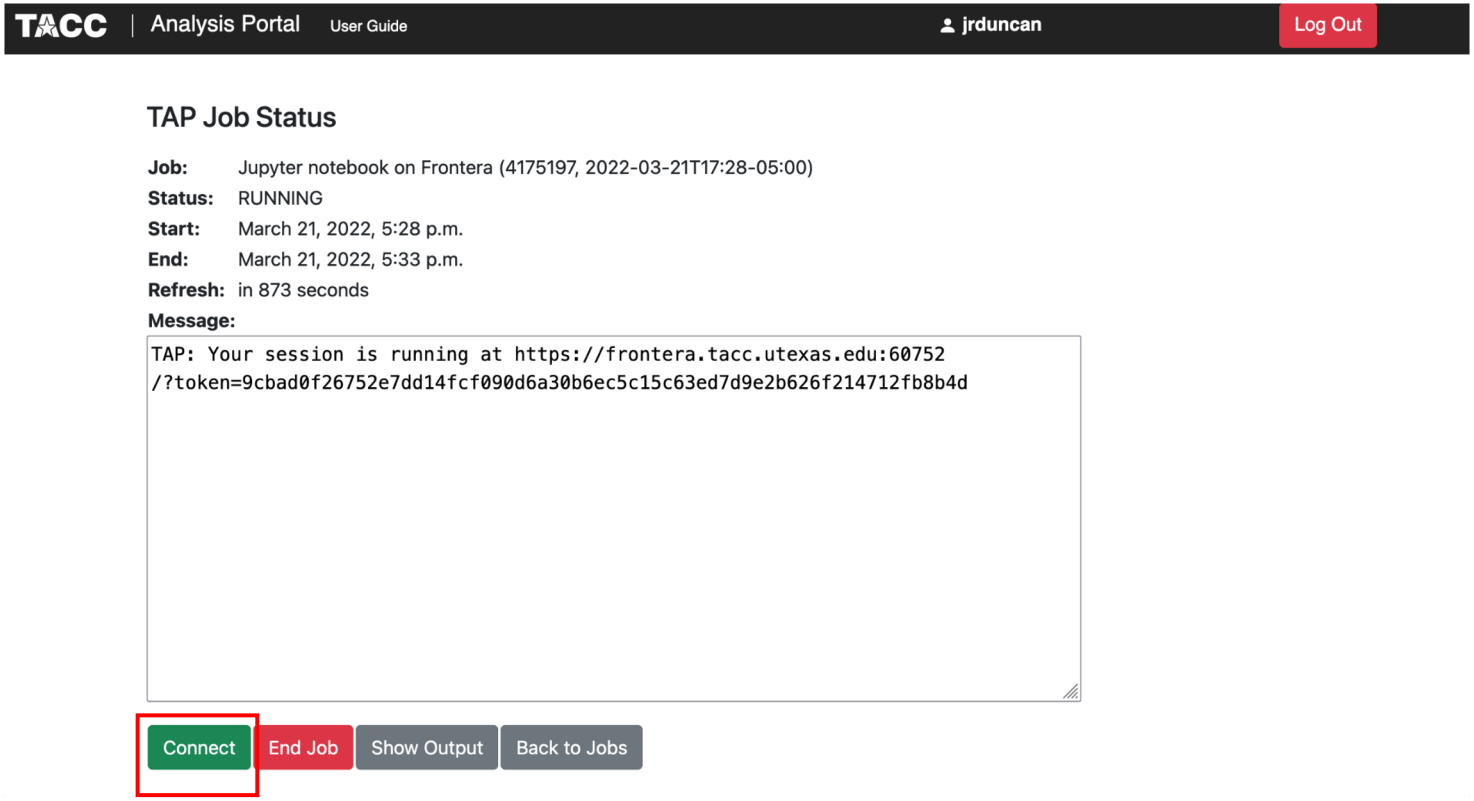
Select the following options



How to acquire TACC resources

Use Analysis Portal

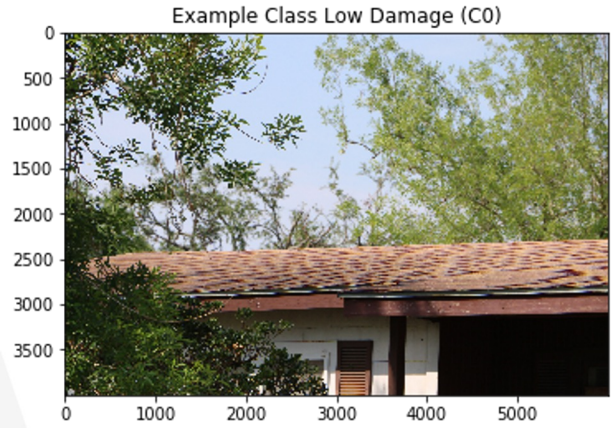
Connect to the compute node(s)

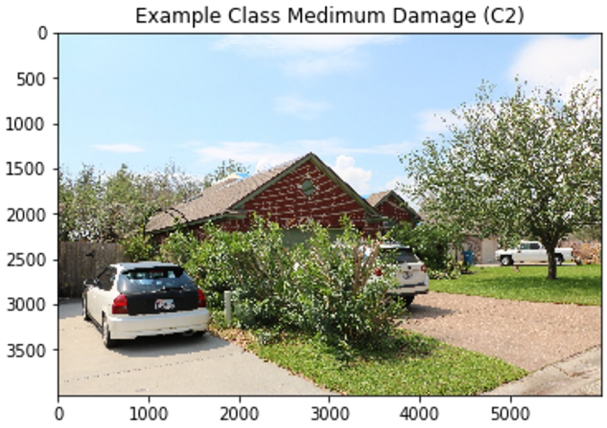


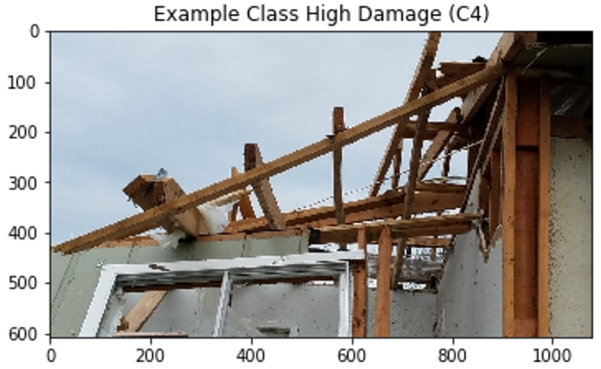
Exercise

DESIGNSAFE-CI
A NATURAL HAZARDS ENGINEERING COMMUNITY

Image Classification with Hurricane Harvey Dataset







Training process

Image processing

Vectorized Data

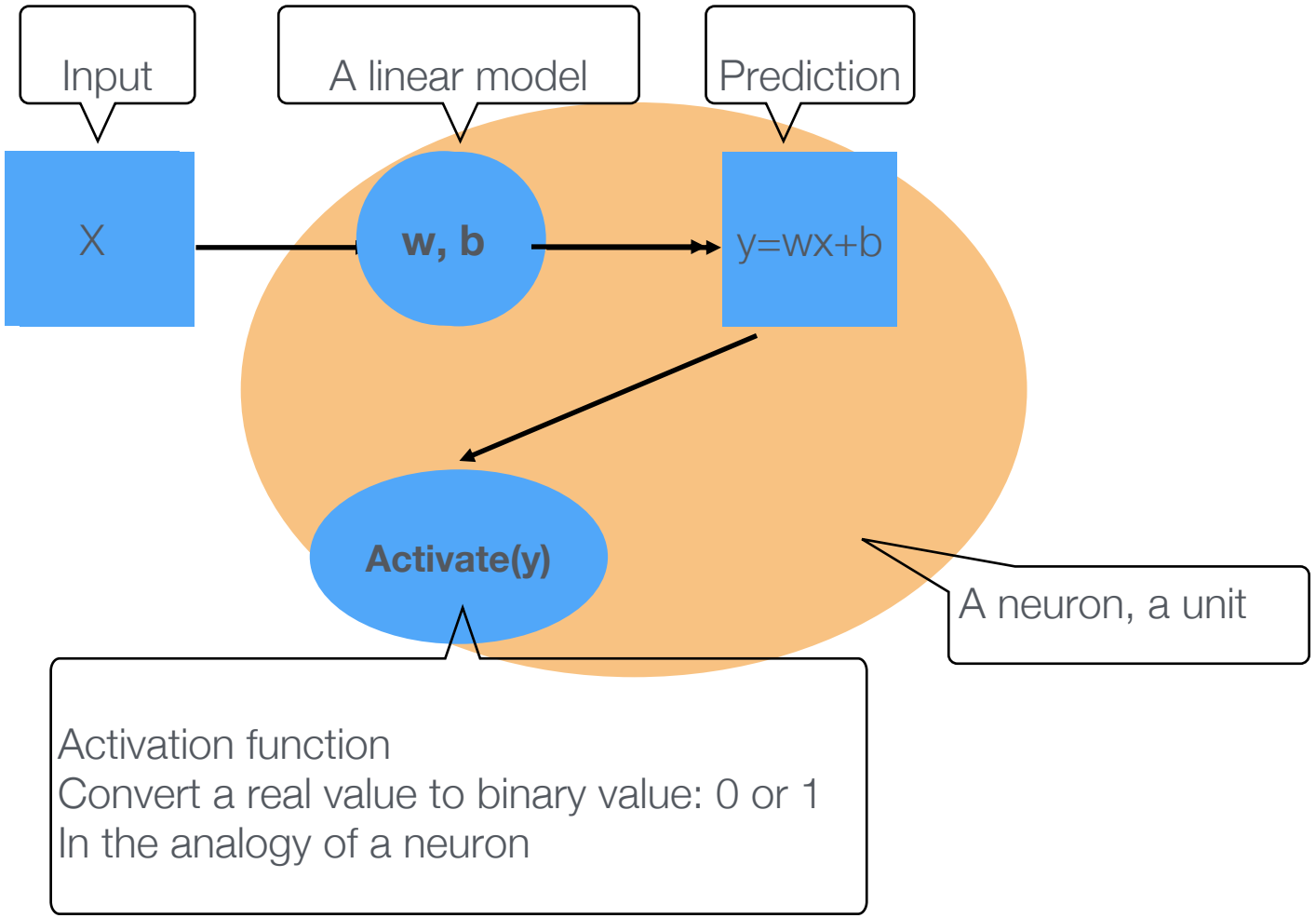
									low
			pixels						medium
									high

Train Model

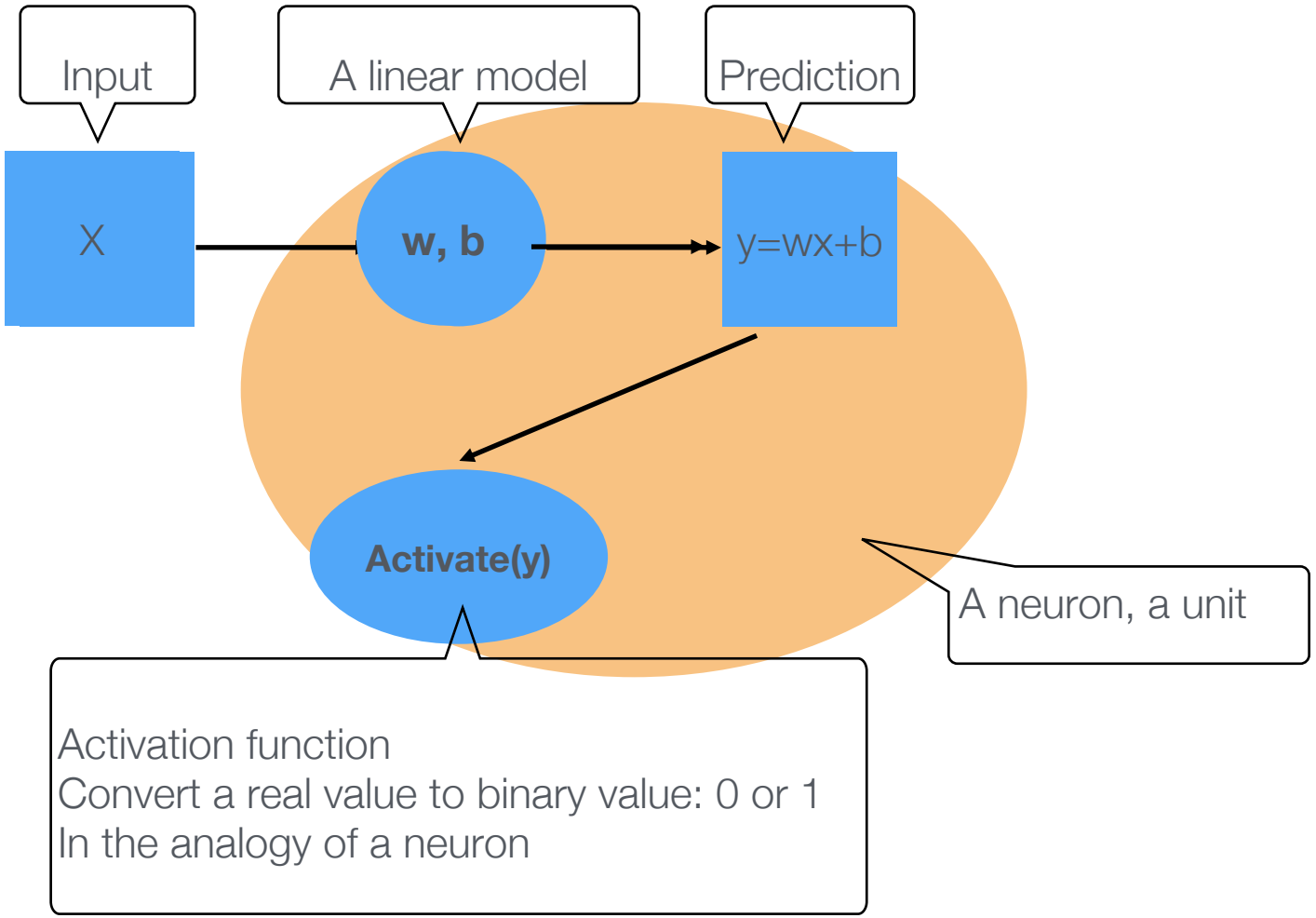
Evaluate Model

Extra Slides

From Linear Regression to Neural Network

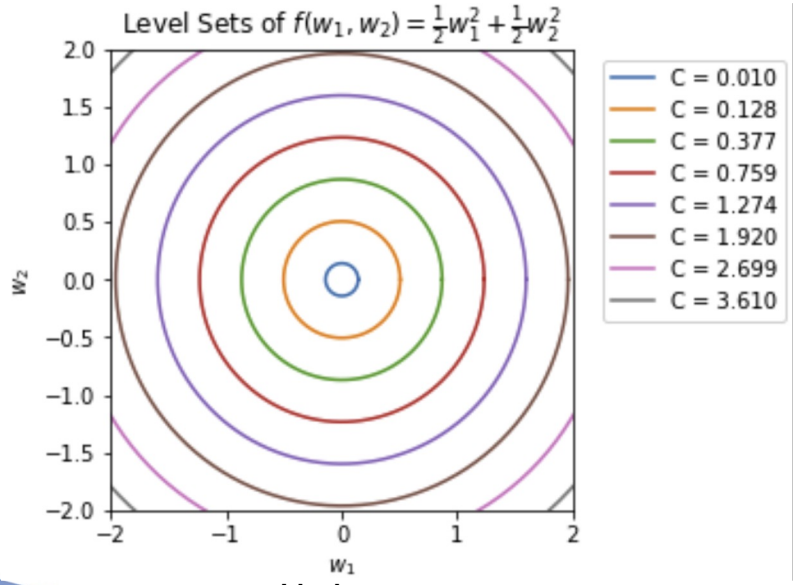


From Linear Regression to Neural Network

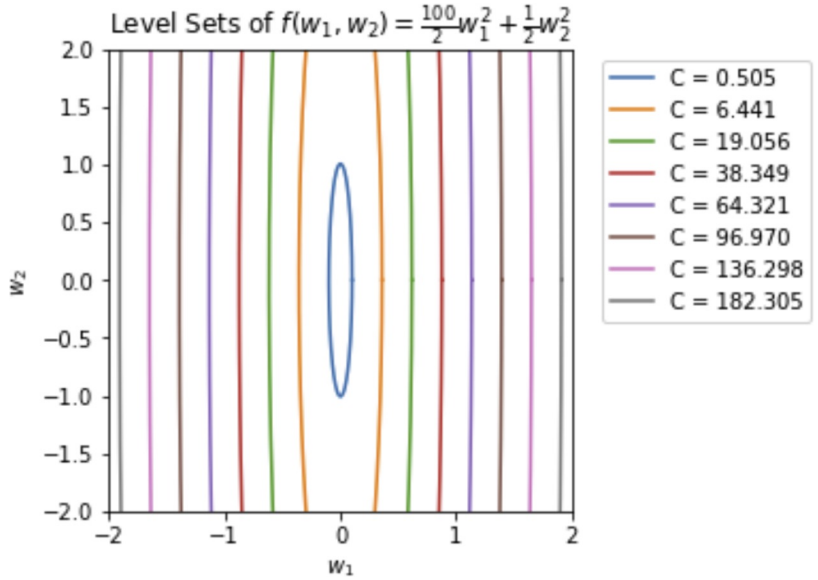


Batch normalization

- Deep MLP can be difficult to train.
- Parameters at different dimension can have very different gradient. However, we use the same learning rate for all dimensions.



Ideal case



Practical case

Image credit: http://www.opennn.net/

Batch normalization

- One direct consequence is that model learns fast in one direction but learns slow in another direction, elongating the whole training process.

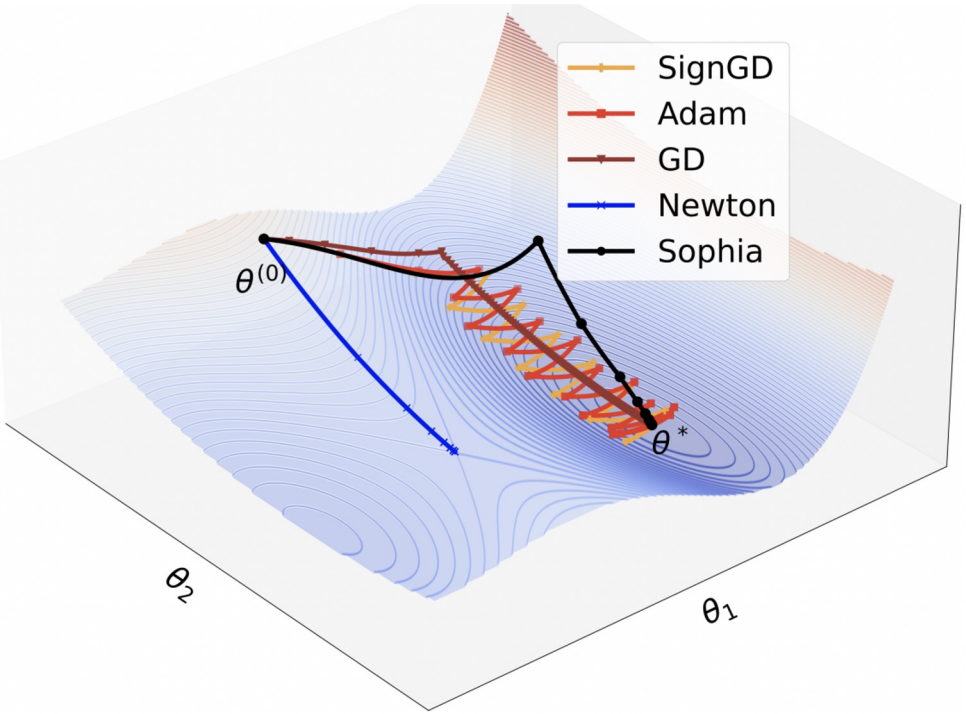


Image credit: https://arxiv.org/pdf/2305.14342.pdf

Batch normalization

- Let $u \in \mathbb{R}^B$ denote a vector of activation of the original NN.
- Batch normalization replaces u with

$$\mathrm{BN}(u) = \gamma \frac{u - \mu}{\sigma} + \beta,$$

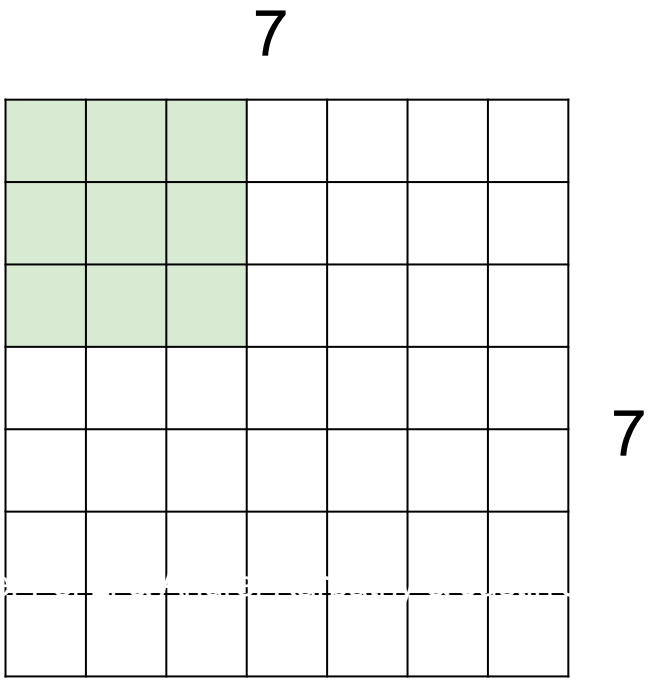
where at training time

$$\mu = \frac{1}{B}\sum_{b=1}^{B} u_b \text{ and } \sigma = \sqrt{\frac{1}{B}\sum_{b=1}^{B}(u_b - \mu)^2}$$

parameter γ and β for u are learned during the training process.

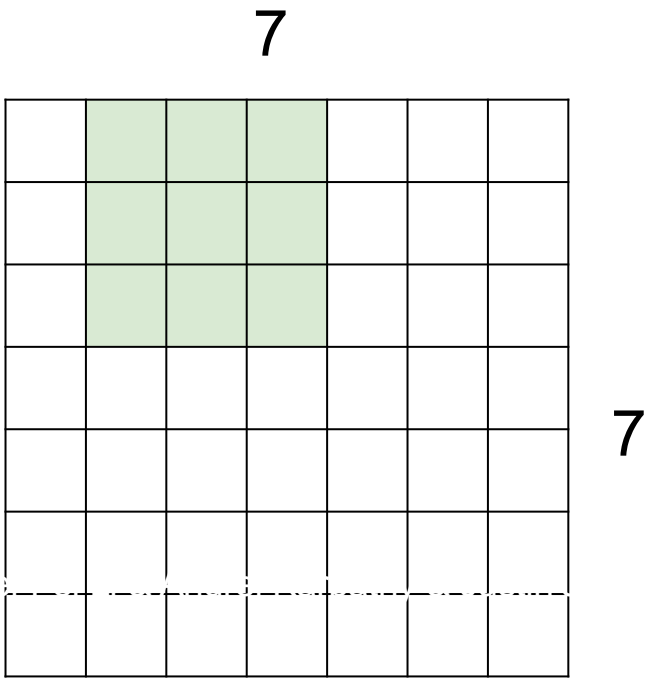
https://www.geeksforgeeks.org/what-is-batch-normalization-in-deep-learning/

A closer look at spatial dimensions:



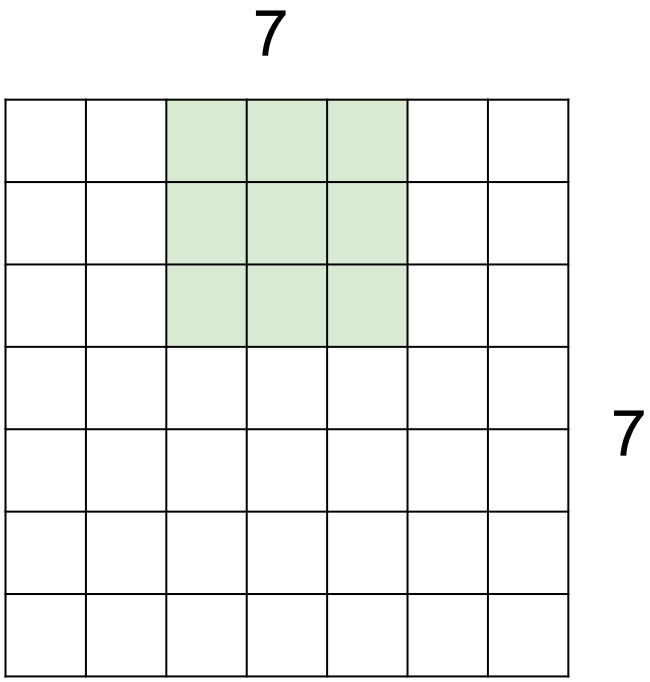
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



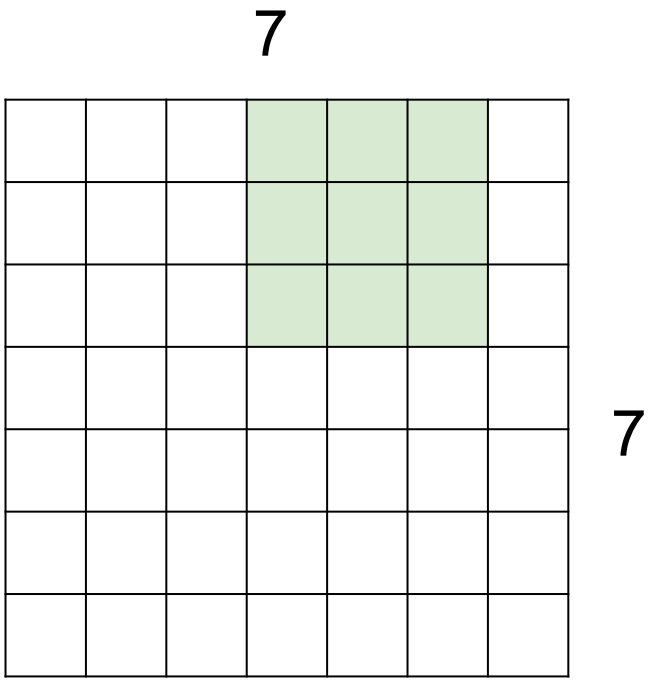
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



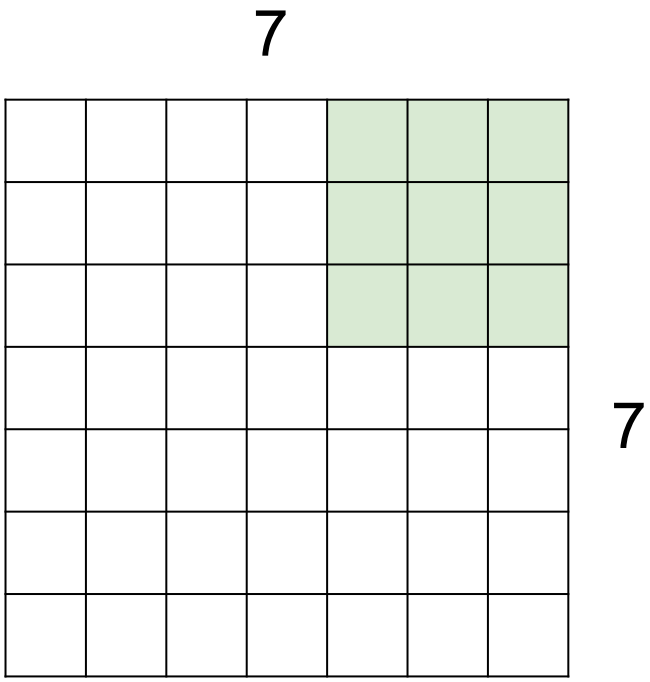
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

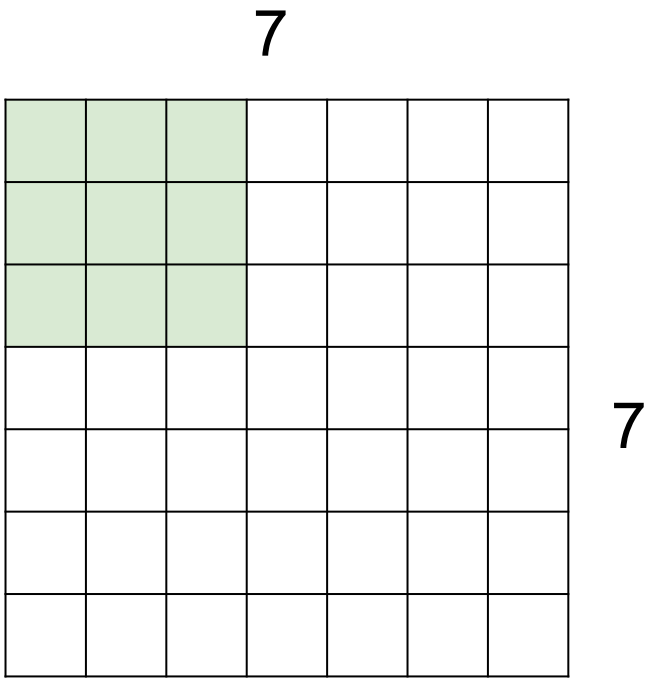
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

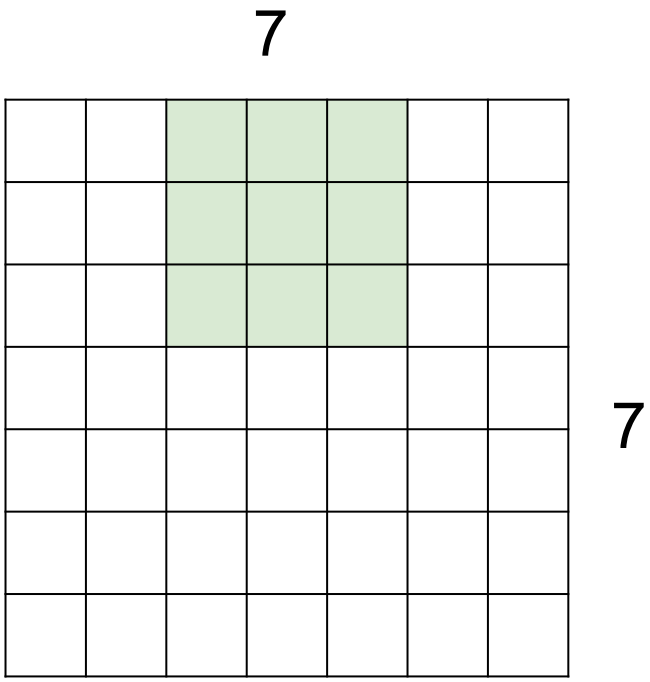
=> 5x5 output

A closer look at spatial dimensions:



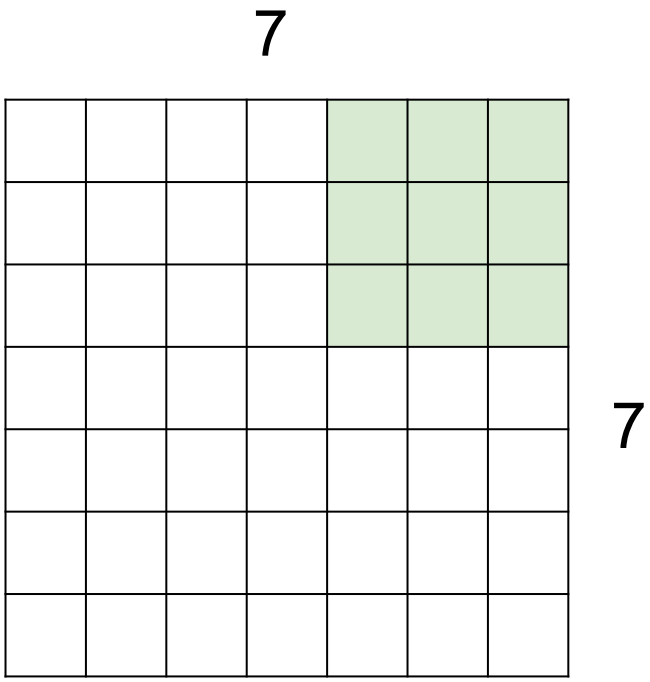
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



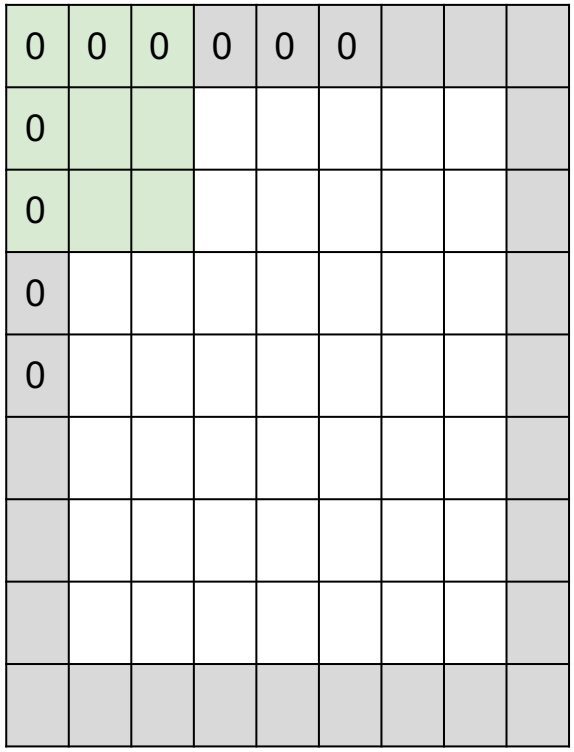
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



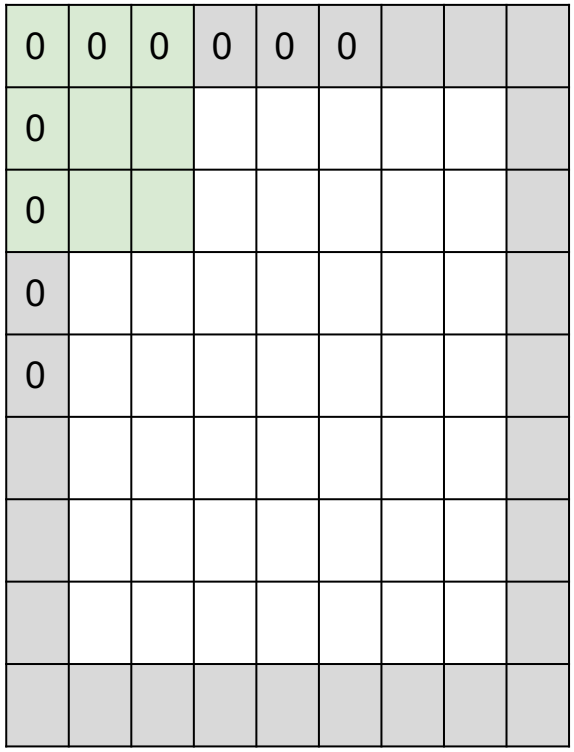
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

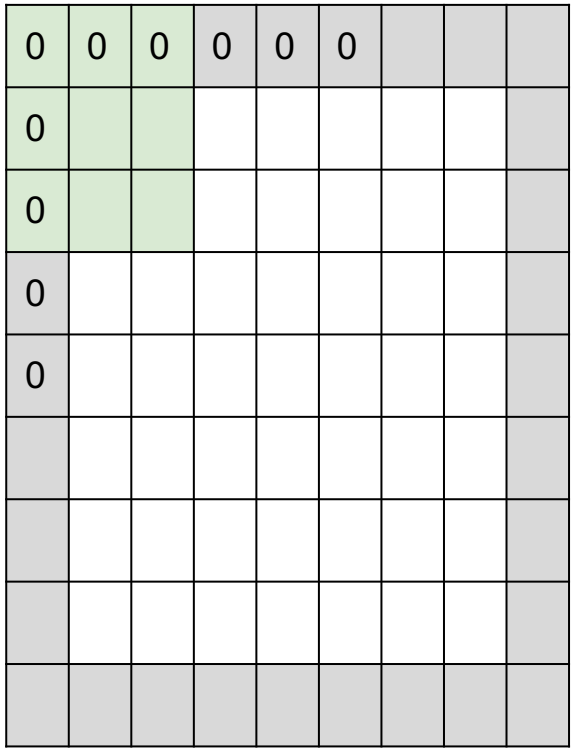
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3