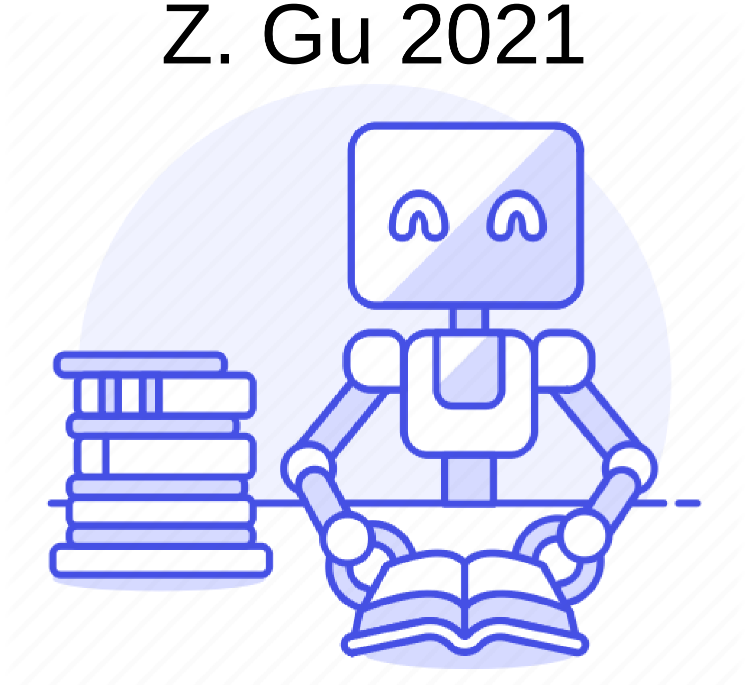
L3 Introduction to Machine Learning

Z. Gu 2021

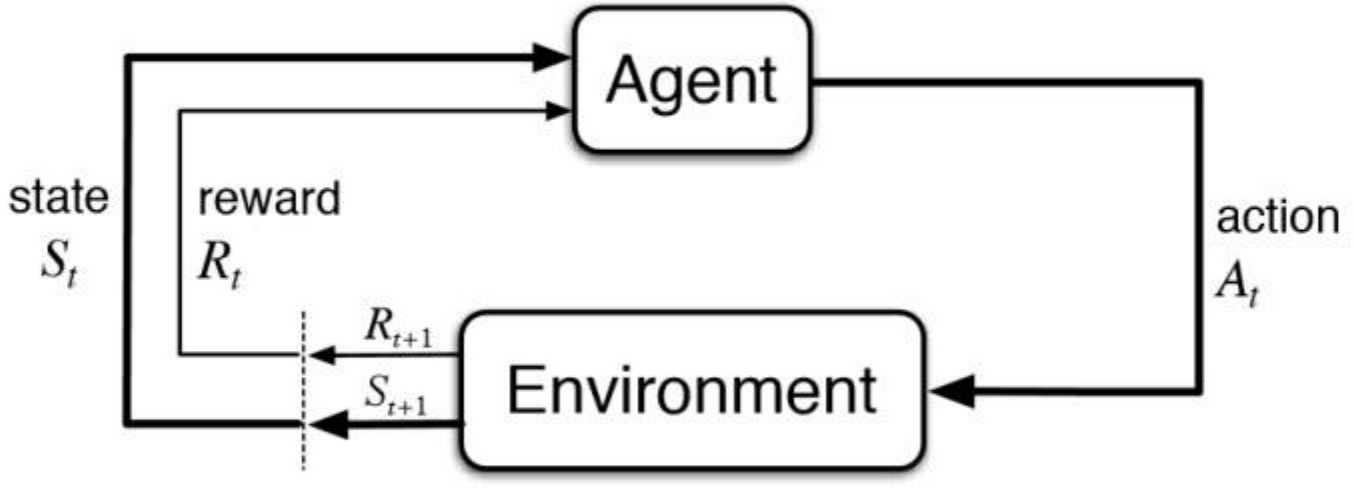


ML Taxonomy

- Supervised Learning:
  - The system is presented with example inputs and their desired outputs, given by a “teacher”, and the goal is to learn a general rule that maps inputs to outputs.
    - Classification (cat or dog?)
    - Regression (housing price next year?)
- Unsupervised Learning:
  - No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).
    - Parametric UL (e.g., Gaussian Mixture Models)
    - Non-parametric UL

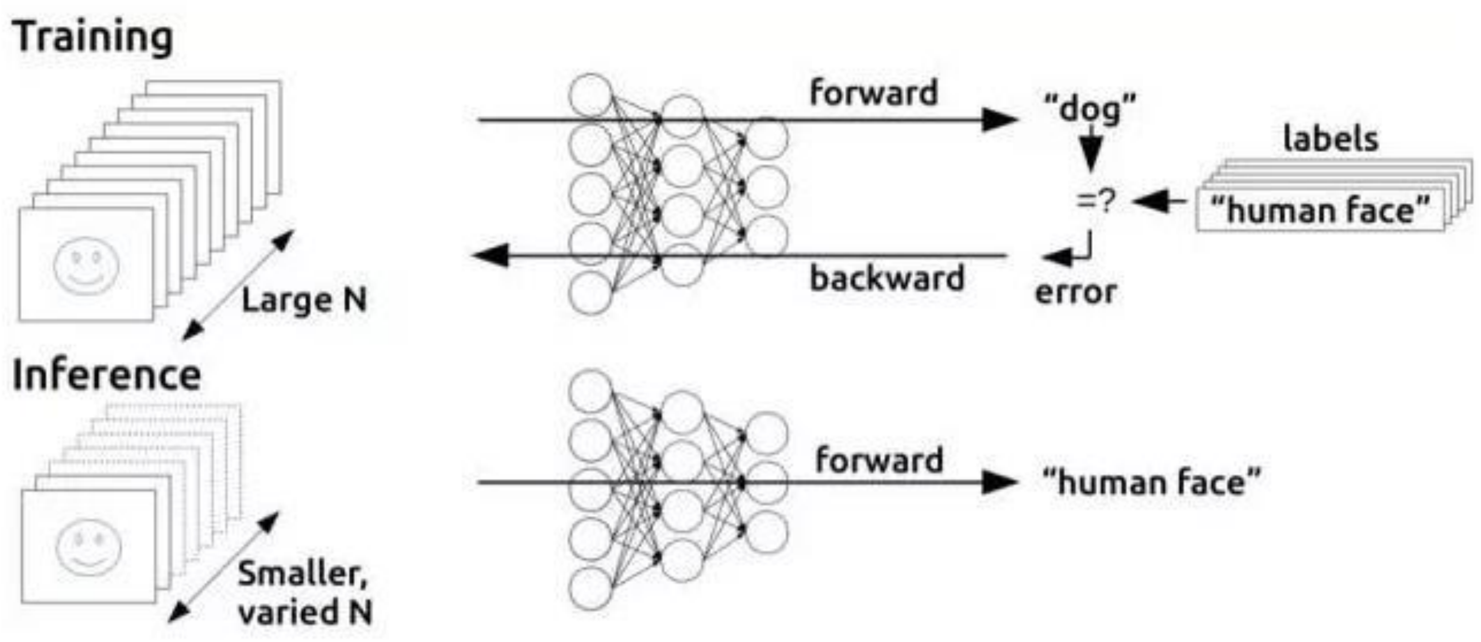
ML Taxonomy

- Reinforcement Learning:
  - An agent interacts with a dynamic environment in which it must perform a certain goal. The agent is provided feedback in terms of rewards and it tries to learn an optimal policy that maximizes its cumulative rewards.
  - Algorithms: Model-based; Model-free (Value-based, Policy-based)
  - Applications: Game playing (AlphaGo); Robotics; AD…



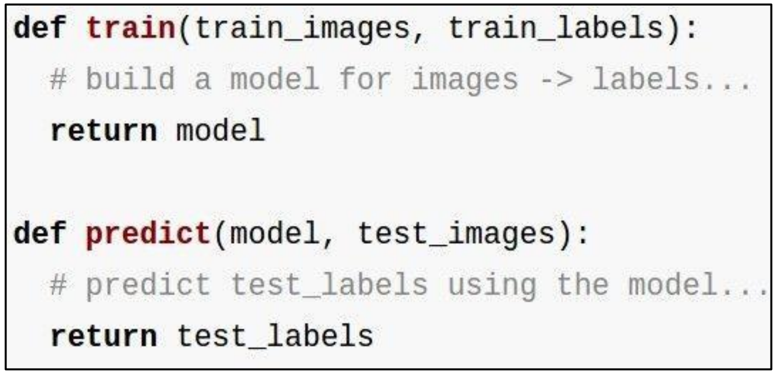
Training vs. Inference

- Training: millions of iterations of forward pass + back propagation to adjust model params (e.g., NN weights); requires large CPU/GPU clusters and days/weeks of training time
- Inference (also called prediction): a single forward pass; can be run on edge devices



Supervised Learning for Image Classification

- Collect a dataset of images and labels
- Use Machine Learning to train an image classifier
- Evaluate the classifier on test images



```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set

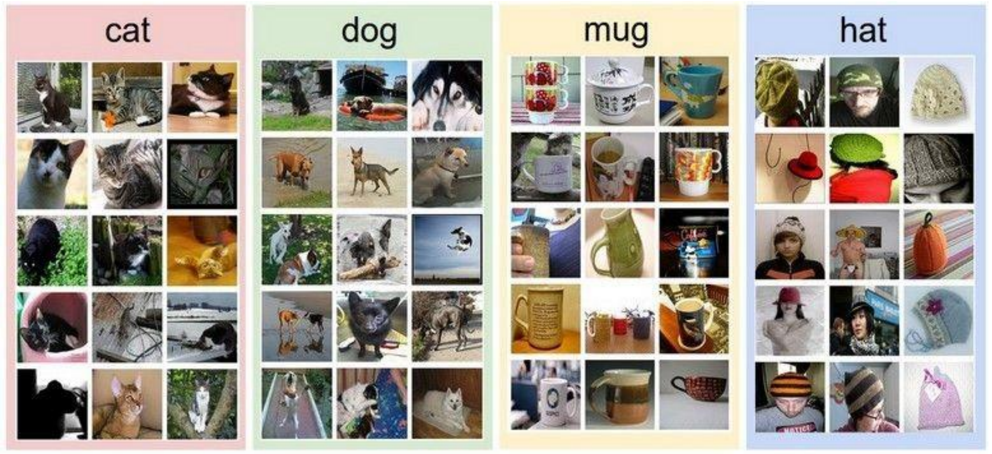
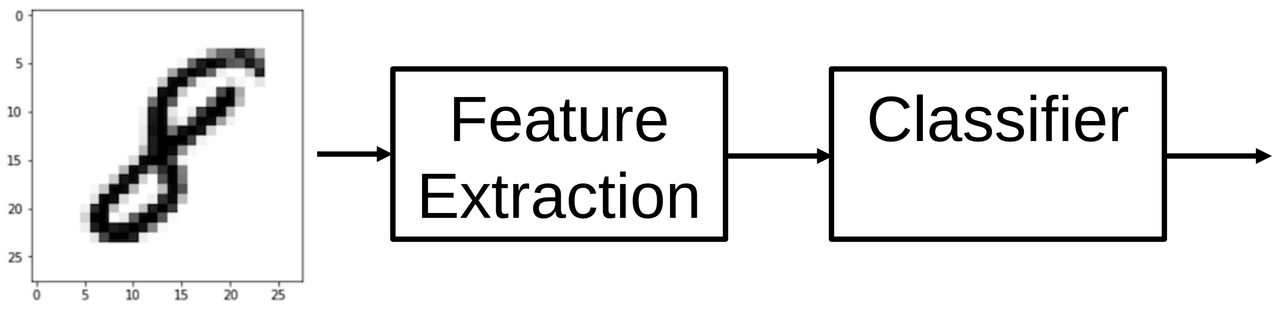


Image Classification

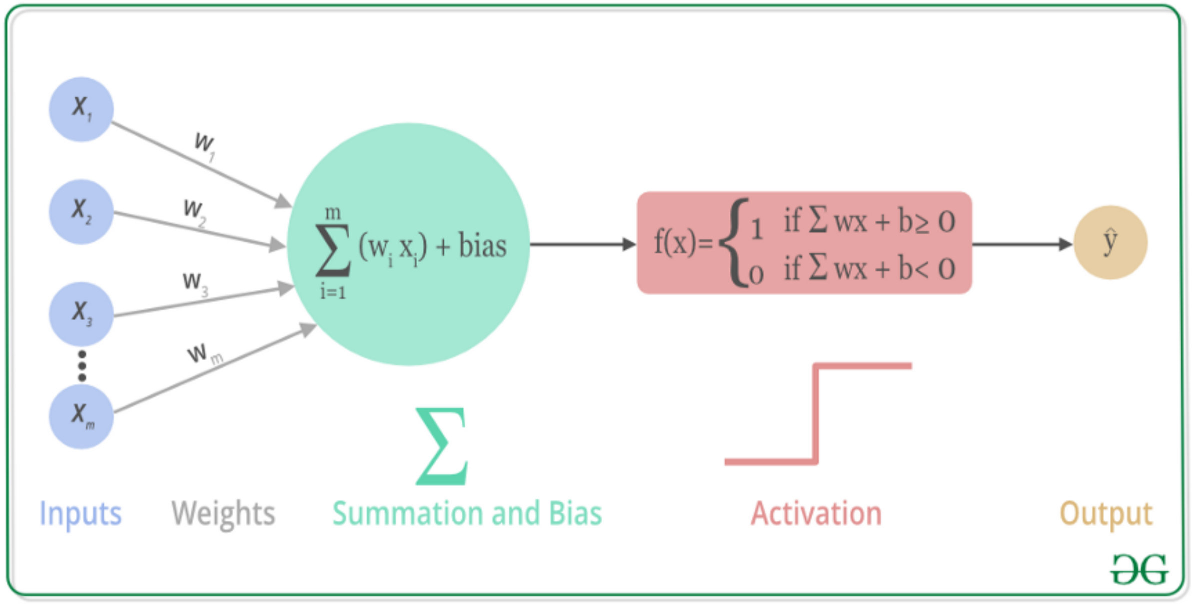
- Input encoding:
  - A color image is represented as a 3D tensor, with integers between [0, 255] denoting pixel intensities, e.g. 300x100x3 (3 color channels RGB)
  - A greyscale image is represented as a 2D tensor, e.g. 300x100x1
- Two stages: feature extraction from input, and classification based on extracted features
- Classifier returns output as a list of probabilities with size equal to the number of classes, but it may also return the top-1 or top-5 results with highest probability ranking



$$\left.\begin{array}{l}P(0) = .01\\ P(1) = .01\\ P(2) = .01\\ P(3) = .02\\ P(4) = .03\\ P(5) = .01\\ P(6) = .02\\ P(7) = .02\\ P(8) = .85\\ P(9) = .02\end{array}\right\}\text{sum to 1.0}$$

A Neuron and its Activation Function

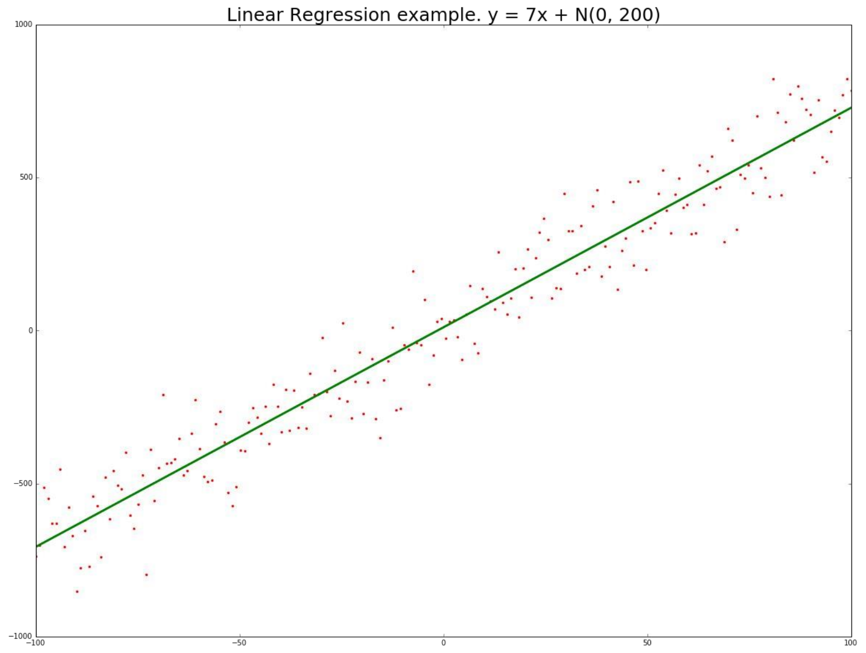
- A neuron may perform:
  - Linear Regression if $y = a = wx + b$ (activation function f = identity function)
  - Logistic Regression if $y = \sigma(a) = \sigma(wx + b)$ (activation function f = sigmoid function, used for binary classification)
  - Perceptron classification $y = \sigma(a) = step(wx + b)$ (activation function f = step function, shown below)
- The activation function is a nonlinear monotonic function that acts like a "gate": the output is larger for larger input activation
  - Many variants: step, sigmoid, tanh, ReLU, leaky-ReLU, PReLU, SoftPlus, Swish…



x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + \text{bias}$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
Σ
Inputs
Weights
Summation and Bias
Activation
Output

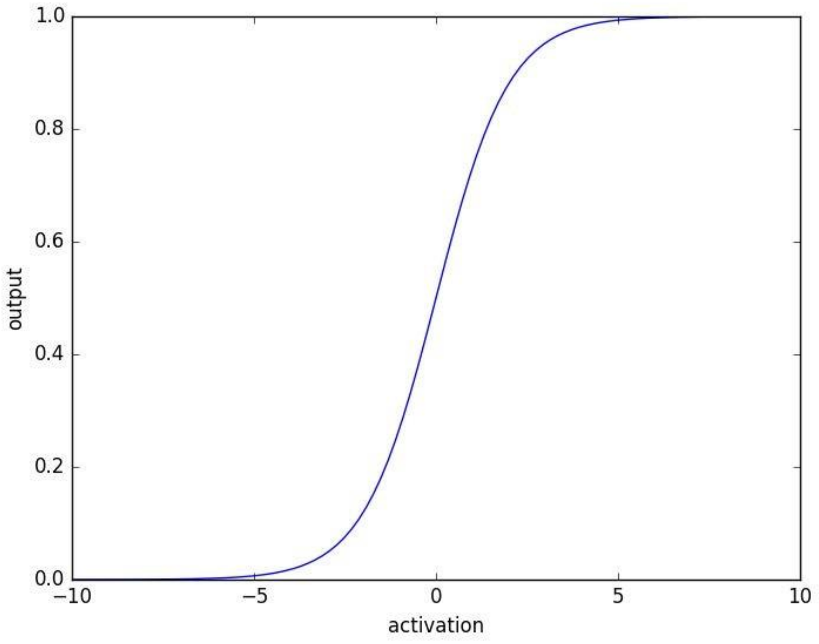
Linear Regression for Regression

- Function approximation $y = wx + b$, with learnable parameters $\theta = \{w, b\}$, where x, y, b are vectors, and w is a weight matrix
  - e.g., we want to predict price of a house based on its feature vector $\mathbf{x} = [x_1\ x_2\ x_3]^\mathbf{T}$, where $x_1$is area in square meters (sqm), x_2 is location ranking (loc), x_3 is year of construction (yoc)
  - Predicted price $y = wx + b = w_1x_1 + w_2x_2 + w_3x_3 + b$
  - Fig shows an example for scalar x and y

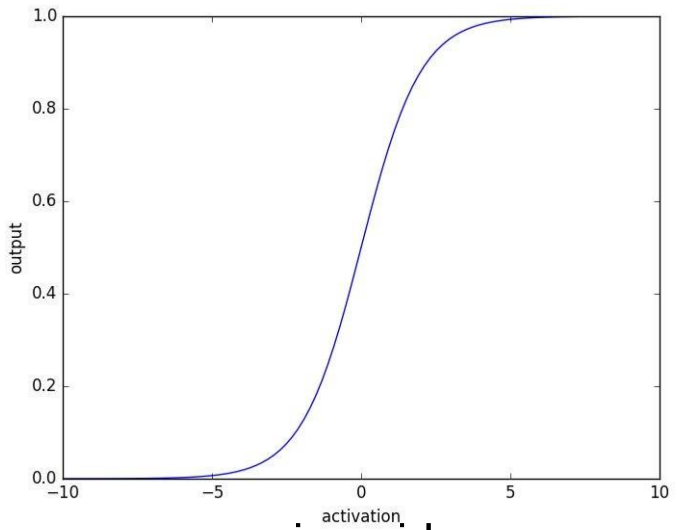


Logistic Regression for Binary Classification

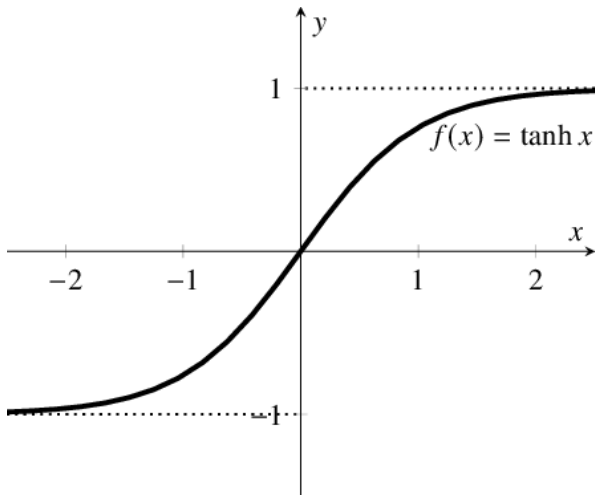
- Consider a binary classification problem: an input image x may be classified as a dog with probability $P(y = dog|x)$, a cat with probability $P(y = cat|x)$, with $P(y = dog|x) + P(y = cat|x) = 1.0$
- Logistic Regression: use sigmoid function $\sigma(a) = \frac{1}{1+e^{-a}}$ to map from activation (also called the logit) to output probabilities



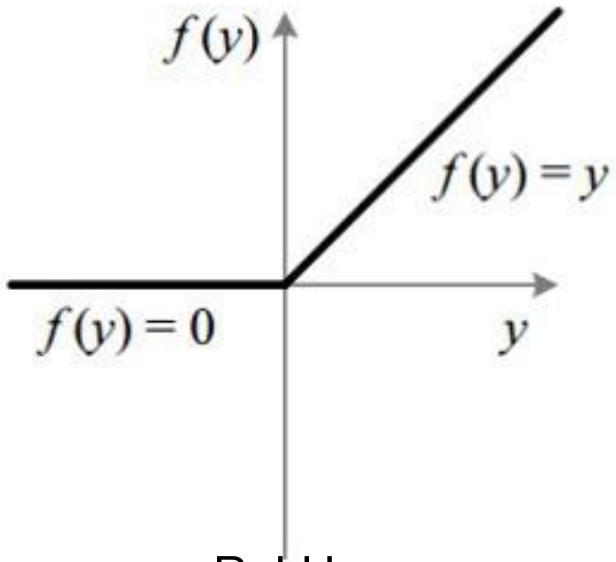
Common Activation Functions used in DL



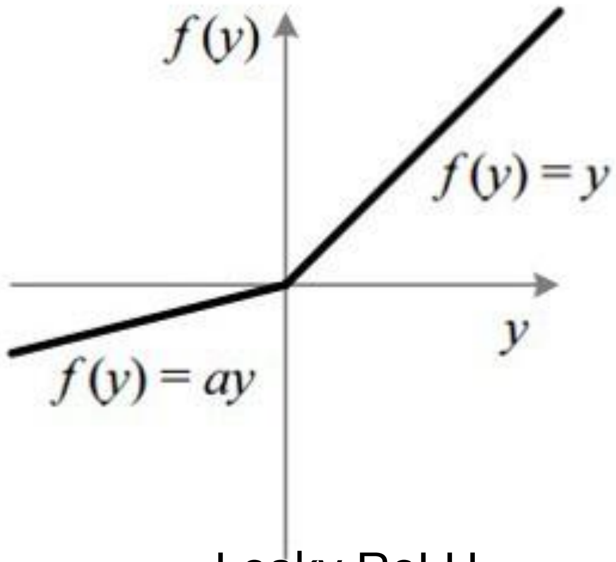
sigmoid



tanh



ReLU



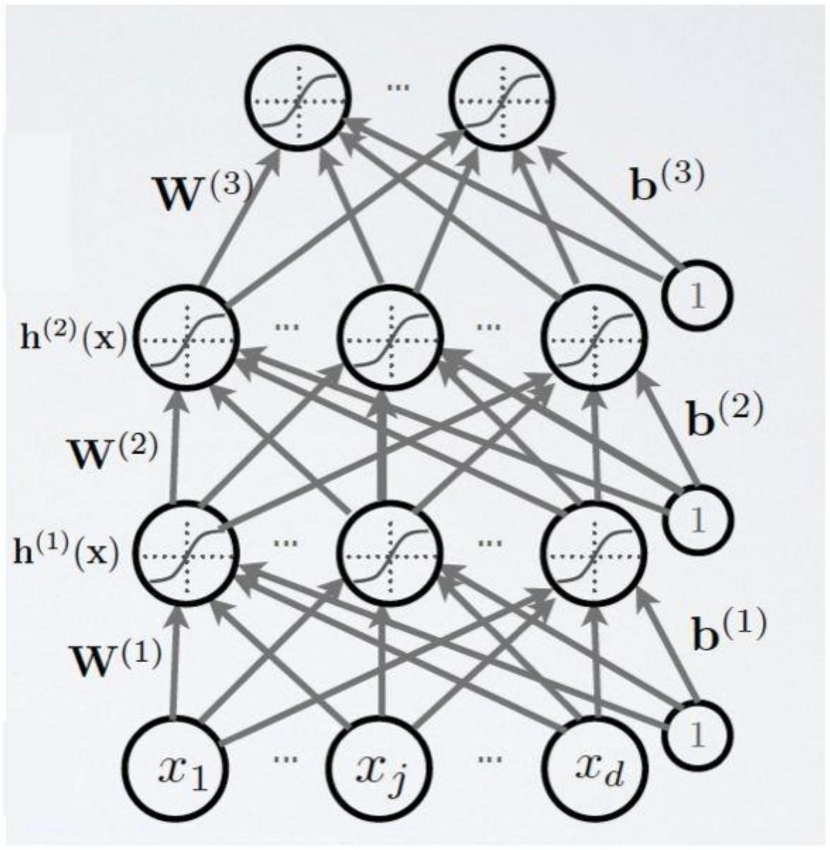
Leaky ReLU

Deep Neural Networks

- We can stack many hidden layers to form a DNN if we have enough data and computing power to train it
- The high model capacity of DNN comes from non-linear mappings: hidden units must be followed by a non-linear activation function
  - Without non-linear activation functions, a DNN with many layers can be collapsed into an equivalent single-layer NN

Fully-Connected NNs

- Number of params to learn at i-th layer is $(N_{i-1} + 1) * N_i$, where N_i is the number of neurons at i-th layer. Can grow very large
  - (We will discuss CNNs in the next lecture with much fewer params)

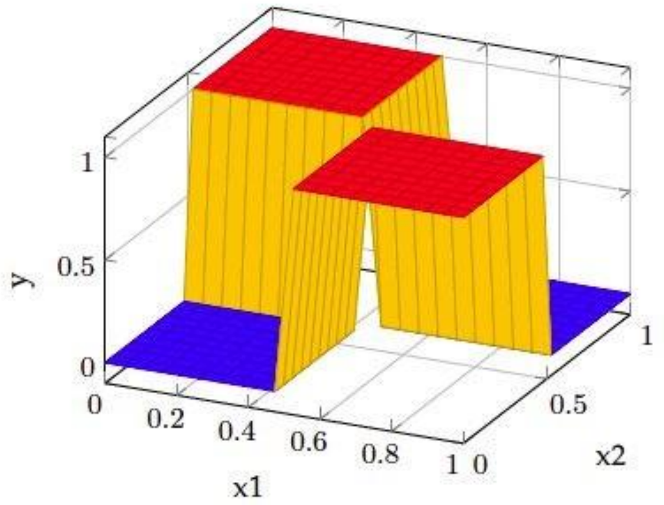


Slide Credit: Hugo Laroche NN course

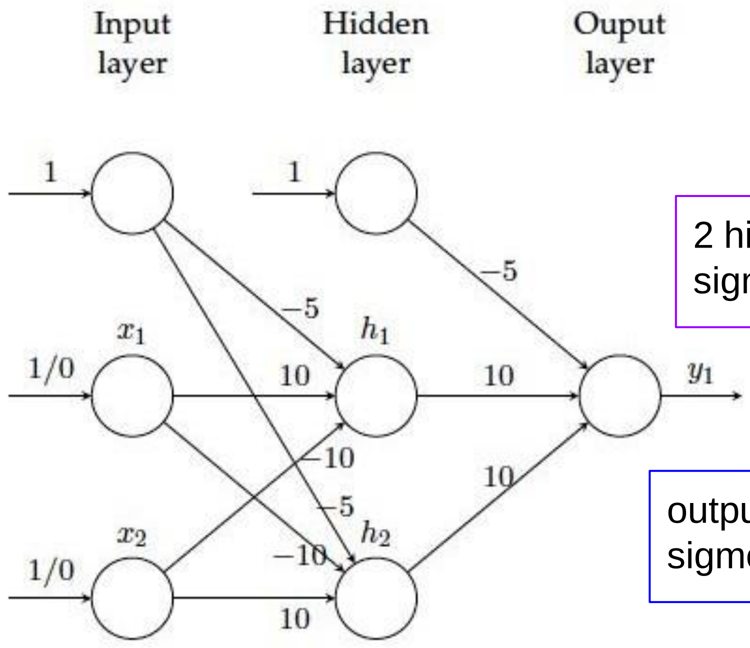
A 3-layer NN

Example: Two-Layer Fully-Connected NN for Solving XOR

- The NN consists of one input, one hidden, and one output layer, with sigmoid activations



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

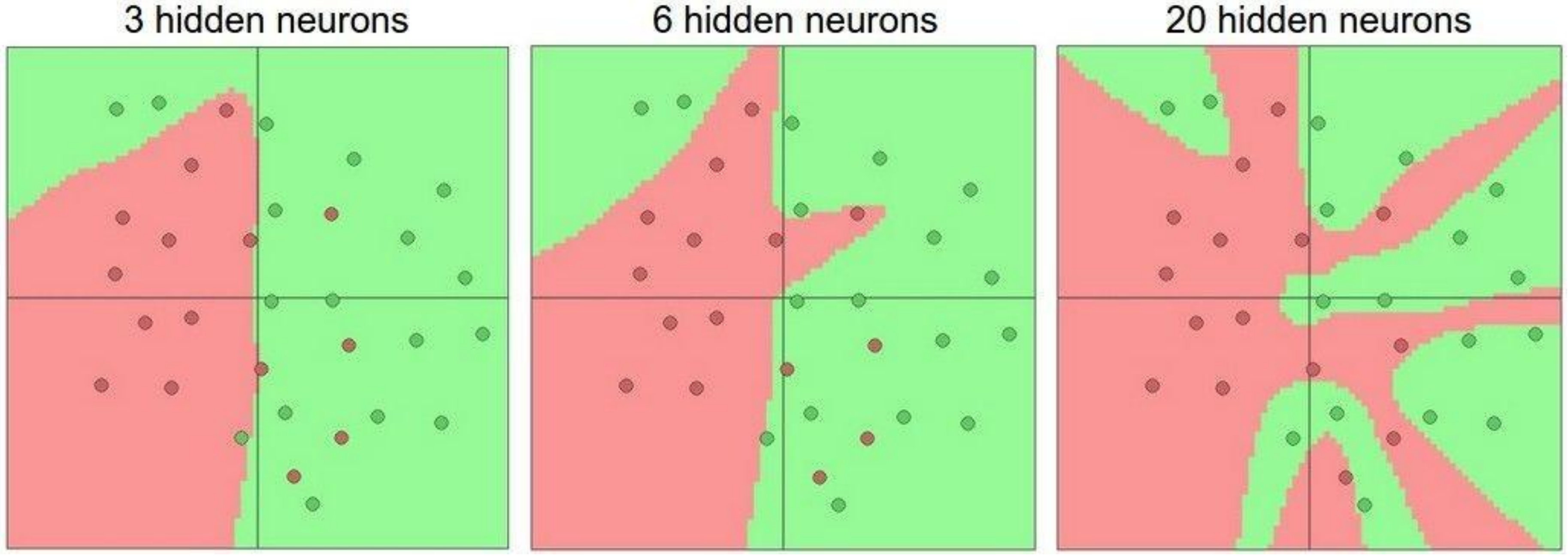


2 hidden units with sigmoid activations

output unit with sigmoid activation

Setting # Layers and Their Sizes

- An example illustrating adding more hidden neurons increases model capacity and reduces training error
- But too many layers and neurons may lead to overfitting



NN for Multi-Class Classification

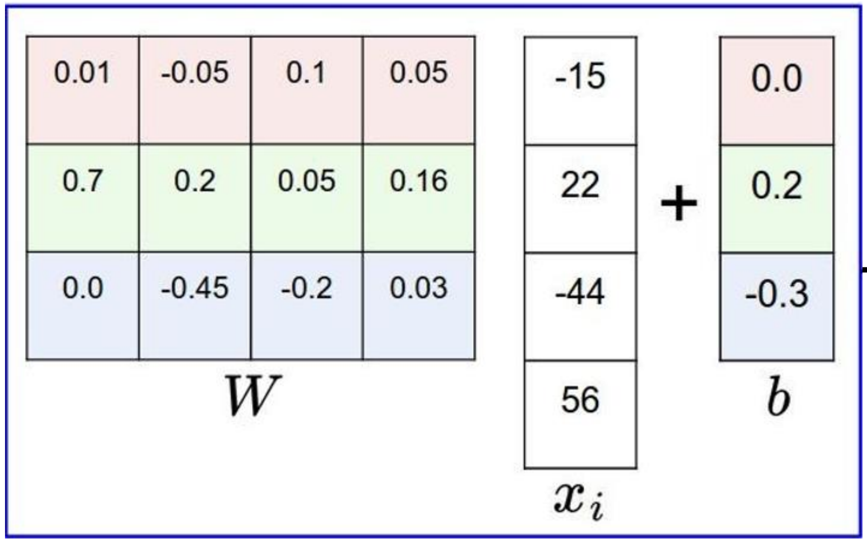
- Consider a NN defining the model, $h_\theta : \mathcal{X} \to \mathbb{R}^k$, as the mapping from input x to output $h_\theta(x)$, a k-dim vector of logits, where k is the number of classes
  - θ is the set of params (weights and biases)
  - y is the correct label for input x
  - Note that h_θ does not include the last SoftMax layer
- e.g., a 3-layer NN consisting of 2 layers with ReLU activation functions and a last linear layer is
  - $h_\theta(x) = W_3 \max(0, W_2 \max(0, W_1 x + b_1) + b_2) + b_3$

Cross-Entropy Loss for Multi-Class Classification

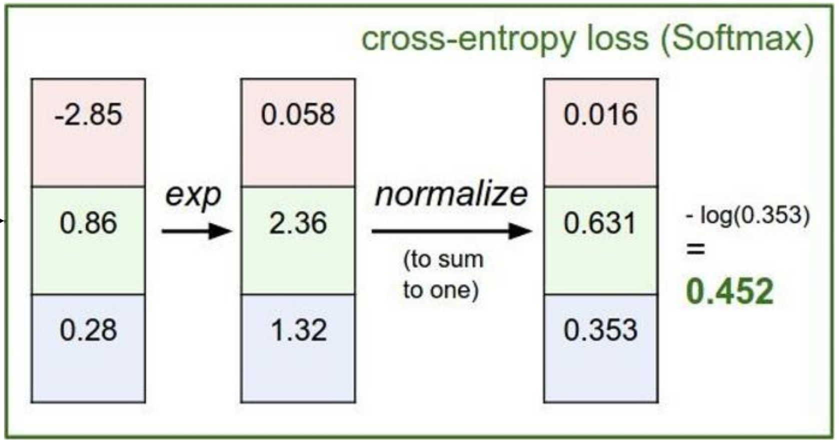
- The SoftMax operator $\sigma: \mathbb{R}^k \to \mathbb{R}^k$ computes a vector of predicted probabilities $\sigma(z): \mathbb{R}^k$ from a vector of logits $z: \mathbb{R}^k$, where k is the number of classes:
  - $\sigma(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)}$
- The loss function is defined as the negative log likelihood of the predicted probability corresponding to the correct label y:
  - $\mathrm{Loss}(h_\theta(x), y) = -\log \sigma\big(h_\theta(x)\big)_y = -\log\left(\frac{\exp\left(h_\theta(x)_y\right)}{\sum_{j=1}^{k} \exp\left(h_\theta(x)_j\right)}\right) = \log\left(\sum_{j=1}^{k} \exp\left(h_\theta(x)_j\right)\right) - h_\theta(x)_y$
- Minimizing $\mathrm{Loss}(h_\theta(x), y)$ amounts to maximizing the logit $\big(h_\theta(x)\big)_y$ corresponding to the correct label y

Cross-Entropy Loss Example

- Consider a NN for 3-class classification. Fig shows the last linear layer and the SoftMax layer
- The last linear layer computes the vector of logits $h_\theta(x) = Wx_i + b = [-2.85 \quad .86 \quad .28]^T$($x$ is the input image to the NN, x_i is the intermediate input to the last layer)
- The SoftMax layer computes the vector of predicted probabilities $[.016 \quad .631 \quad .353]^T$ for labels $[1 \quad 2 \quad 3]^T$, and the loss $-\log .353$, assuming correct label $y_i = 3$



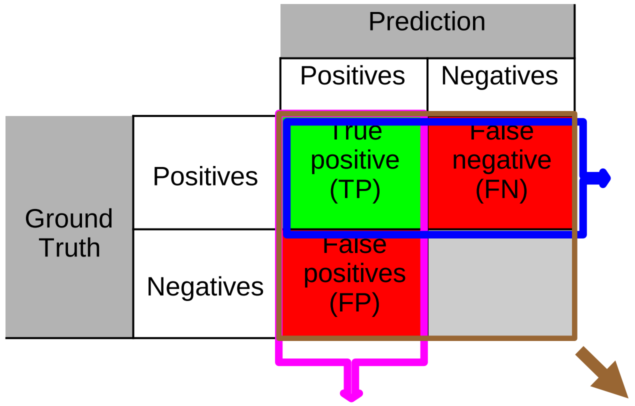
Last Linear Layer



SoftMax Layer

Binary Classification Metrics

- The relevant class is considered “positive” in a binary classifier
- e.g., for a medical test that aims to diagnose people with a certain disease. “Positive” denotes sick (has disease), and “negative” denotes healthy (no disease)
  - TP: a sick person is diagnosed as sick
  - TN: a healthy person is diagnosed as healthy
  - FP: a healthy person is misdiagnosed as sick
  - FN: a sick person is misdiagnosed as healthy

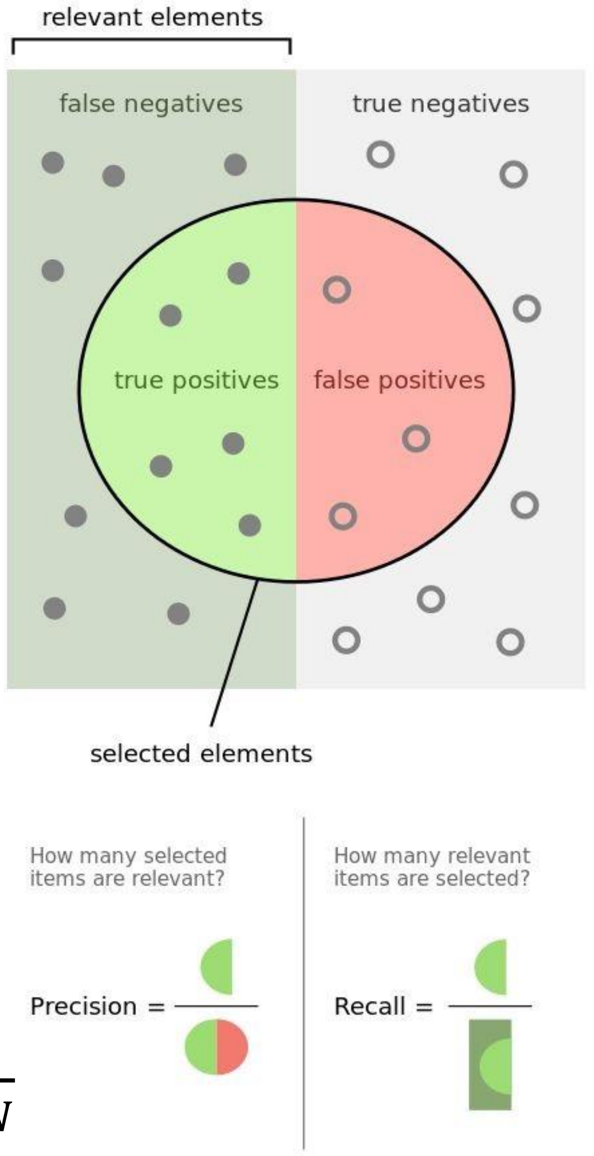


		Prediction	
		Positives	Negatives
Ground Truth	Positives	True positive (TP)	False negative (FN)
	Negatives	False positives (FP)	

$$\text{Recall} = \frac{TP}{TP+FN}$$

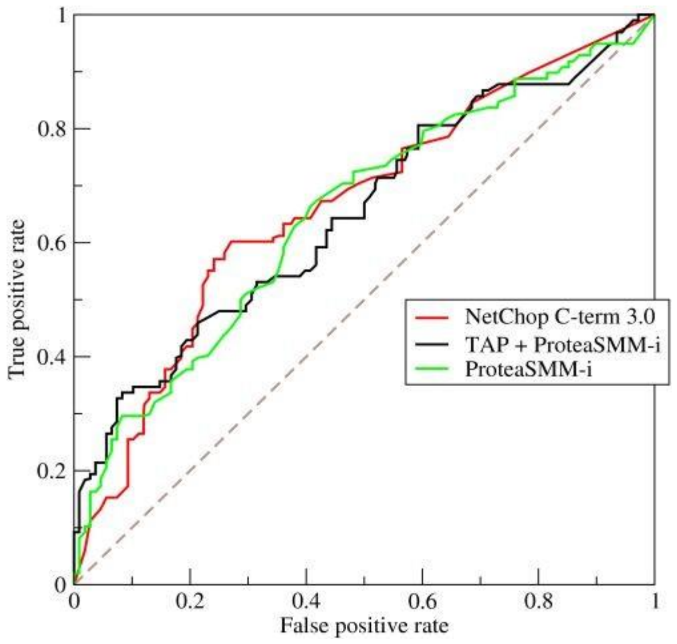
$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$



ROC Curve

- Binary classification results typically depend on a parameter (eg. decision threshold) whose value impacts precision and recall.
- Receiver Operating Curve (ROC curve):
  - x-axis: False Positive Rate $FPR = \frac{FP}{FP+TN}$ = recall
  - y-axis: True Positive Rate $TPR = \frac{TP}{TP+FN}$
  - The ideal ROC curve: $FPR \equiv 0, TPR \equiv 1$, with $FP = FN = 0$
- Moving the decision threshold will cause FPR and TPR to move in the same direction
  - e.g., a medical test that sets a high threshold for positive diagnosis will have both low FPR and low TPR, and vice versa



Confusion Matrix for Multi-Class Classification

- Binary classification as a special case of multi-class classification:

$$Accuracy = \frac{\boxed{\sum_{i=1}^{3} x(i,i)}}{\boxed{\sum_{i=1}^{3}\sum_{j=1}^{3} x(i,j)}} \xrightarrow{\text{Binary}} Accuracy = \frac{\boxed{TP+TN}}{\boxed{TP+TN+FP+FN}}$$

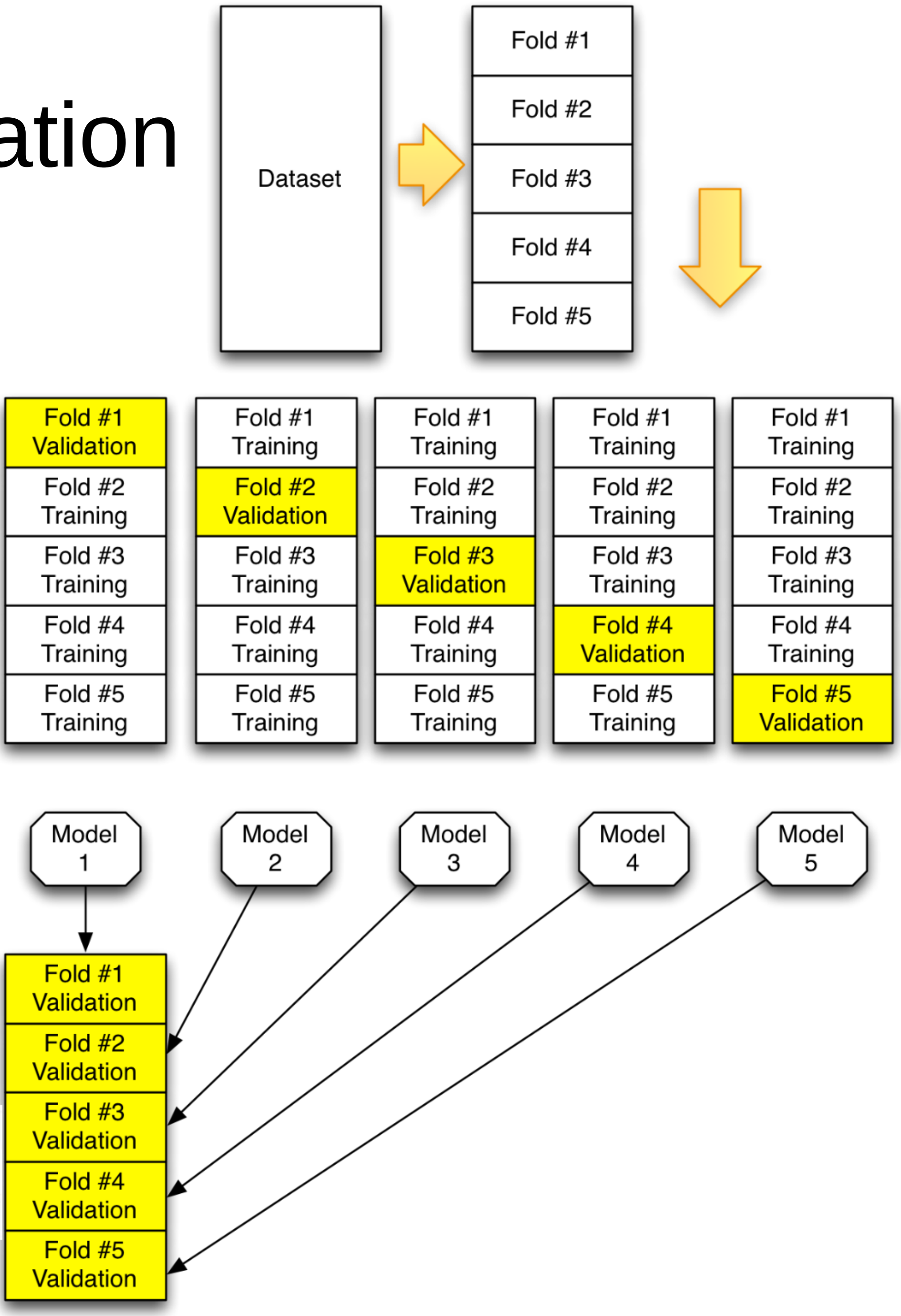
		Prediction		
		Class 1	Class 2	Class 3
Ground Truth	Class 1	x(1,1)	x(1,2)	x(1,3)
	Class 2	x(2,1)	x(2,2)	x(2,3)
	Class 3	x(3,1)	x(3,2)	x(3,3)

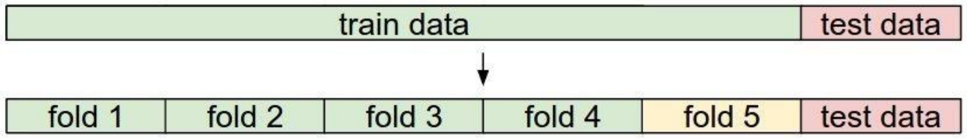
		Prediction	
		Positives	negative
Ground Truth	Positives	True positive (TP)	False negative (FN)
	Negative	False positives (FP)	True negative (TN)

Training Neural Networks

K-Fold Cross-Validation

- Divide data into train data and test data
- Since we cannot peek at the test data during training time, we use part of the train data for Cross-Validation:
- e.g., Divide training data into K=5 parts (folds). Use each fold as validation data, and the other 4 folds as training data. Cycle through the choice of which fold used for validation and average results.

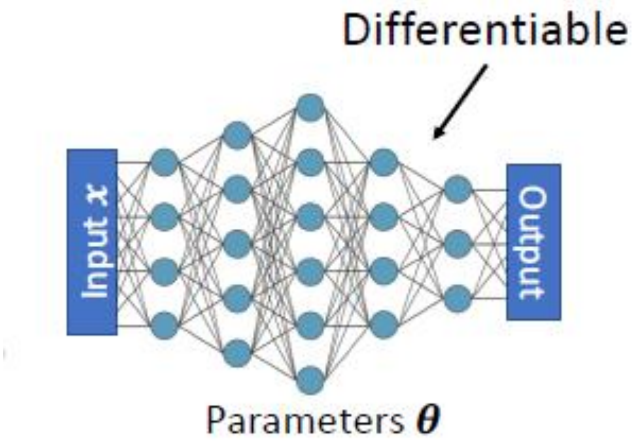




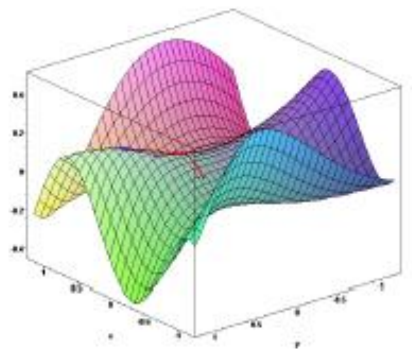
https://github.com/jeffheaton/t81_558_deep_learning/blob/master/t81_558_class_05_2_kfold.ipynb

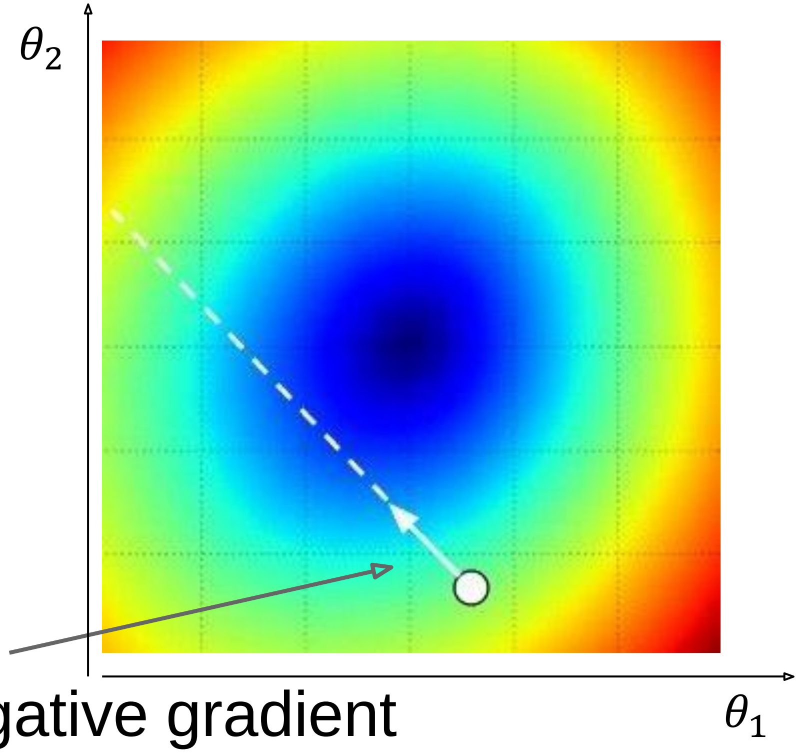
Gradient Descent

- Gradient descent $\theta \leftarrow \theta - \alpha \nabla_\theta \mathrm{Loss}(x, y; \theta)$



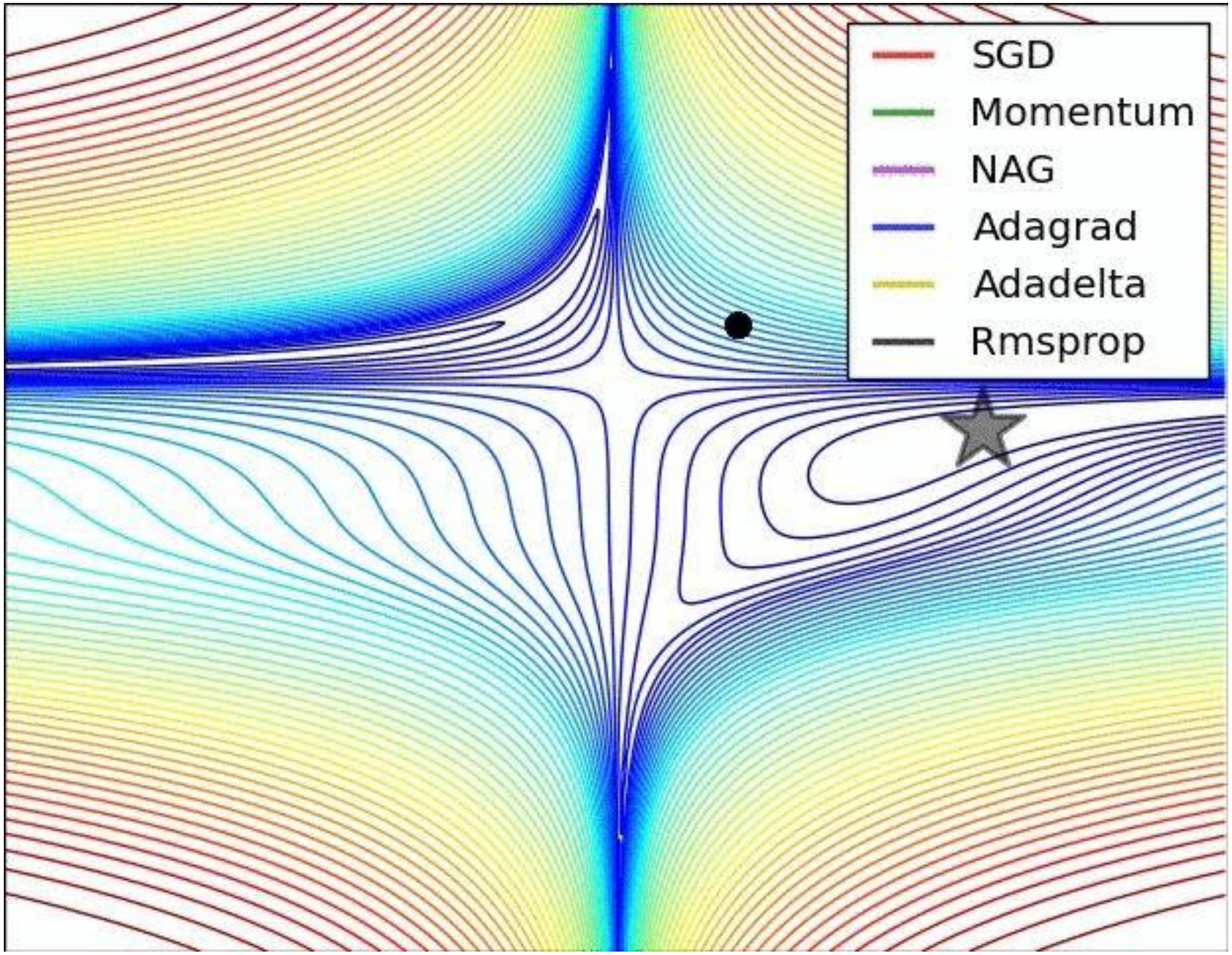
Can use gradient descent method to find good θ





Negative gradient direction

Different GD Update Formulas



Mini-batch Stochastic Gradient Descent

- Only use a small portion (a mini-batch) of the training data to compute the gradient
- Common mini-batch sizes are 32/64/128 examples
- Loop:
  – Sample a mini-batch of data
  – Forward prop it through the graph, get loss
  – Backprop to calculate the gradients
  – Update the parameters using gradient descent

Batch Normalization

- For each mini-batch:
  - 1. Compute the empirical mean and variance independently for each dimension $i = 1, \dots m$
  - 2. Normalize to a unit Gaussian with 0 mean and unit variance
- BN layers inserted before nonlinear activation function, and it keeps x's average value around 0 for maximum gradient during learning
- Scale and shift params γ, β gives more flexibility during training
- Benefits:
  - Improves gradient flow through the network; Allows higher learning rates; Reduces the strong dependence on initialization; Acts as a form of regularization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

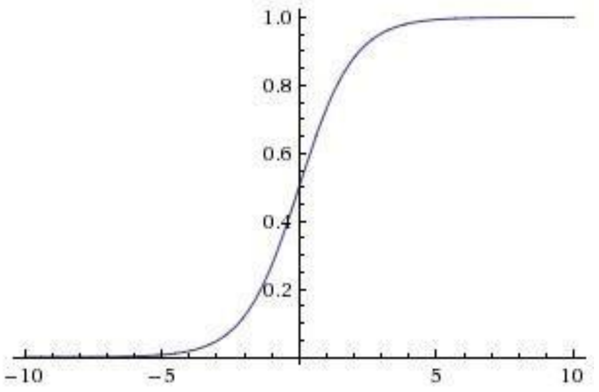
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

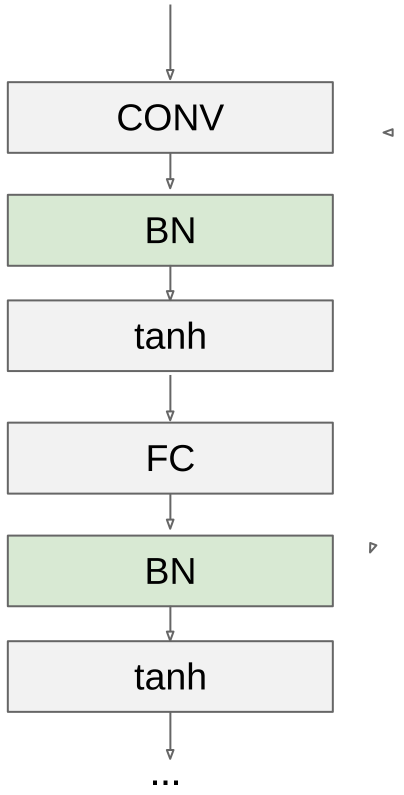
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

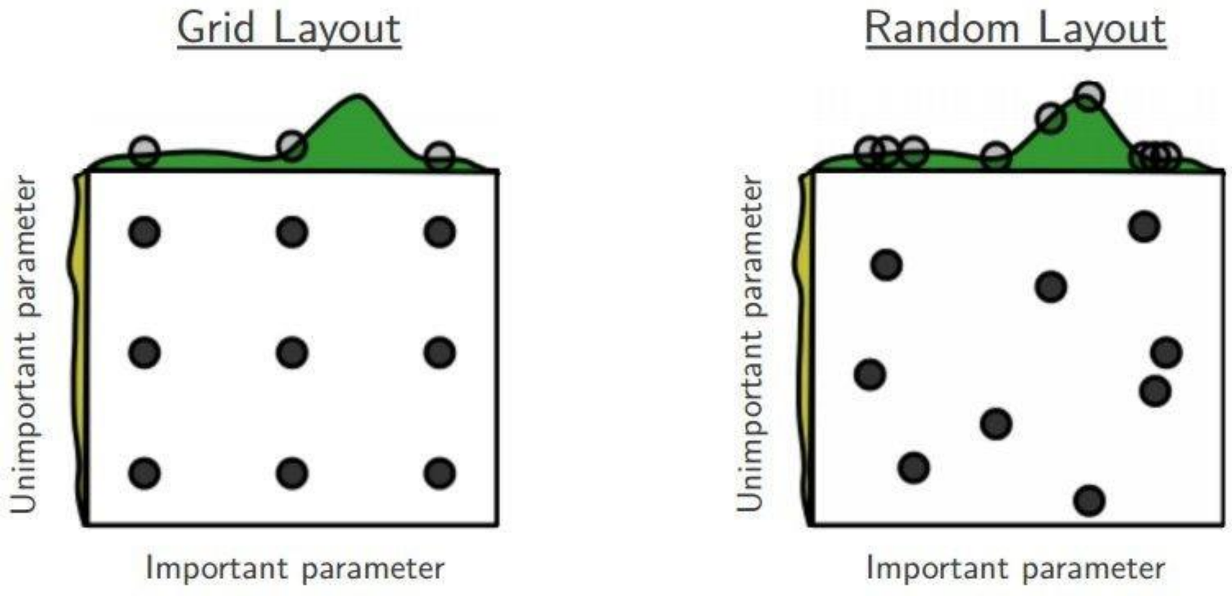
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



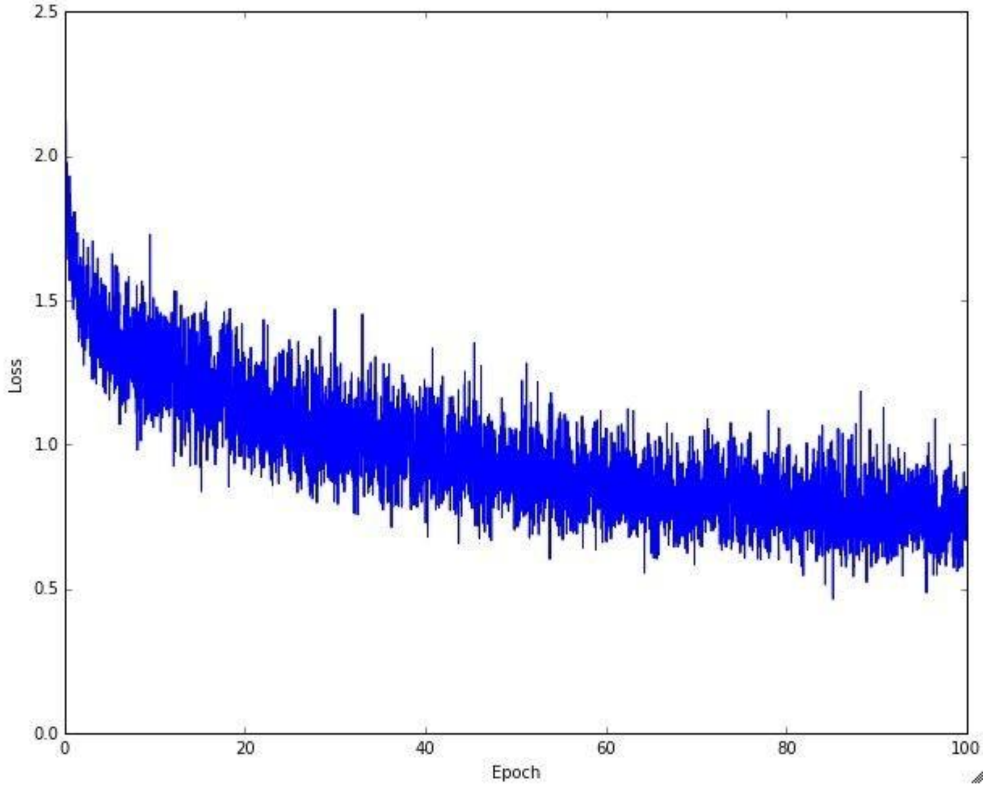


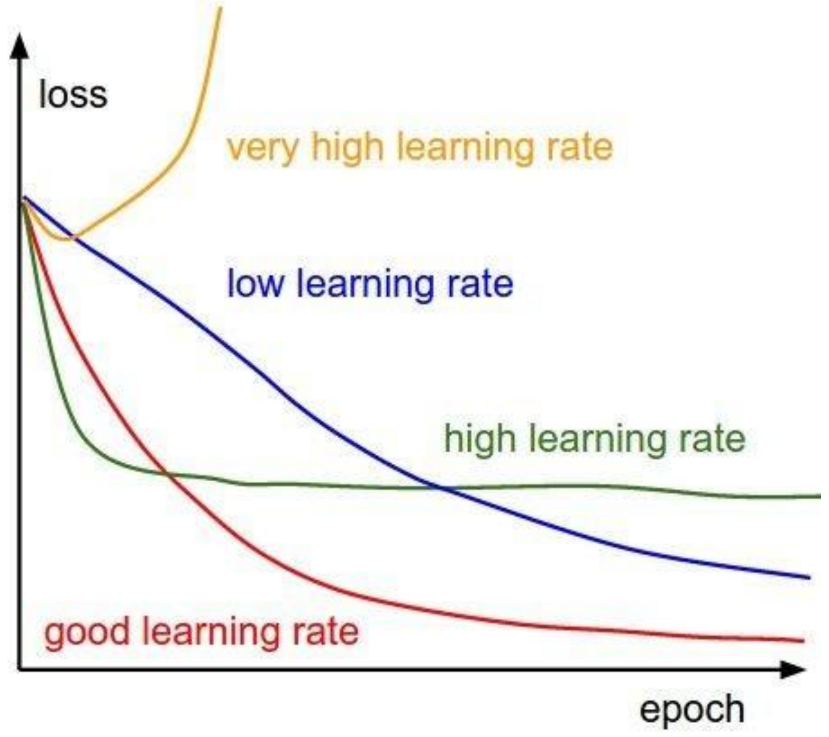
Hyperparameter Optimization

- Example hyperparams
  - Network architecture
  - Learning rate, its decay schedule, update type
  - Regularization (L2/Dropout strength)
- Grid search vs. random search



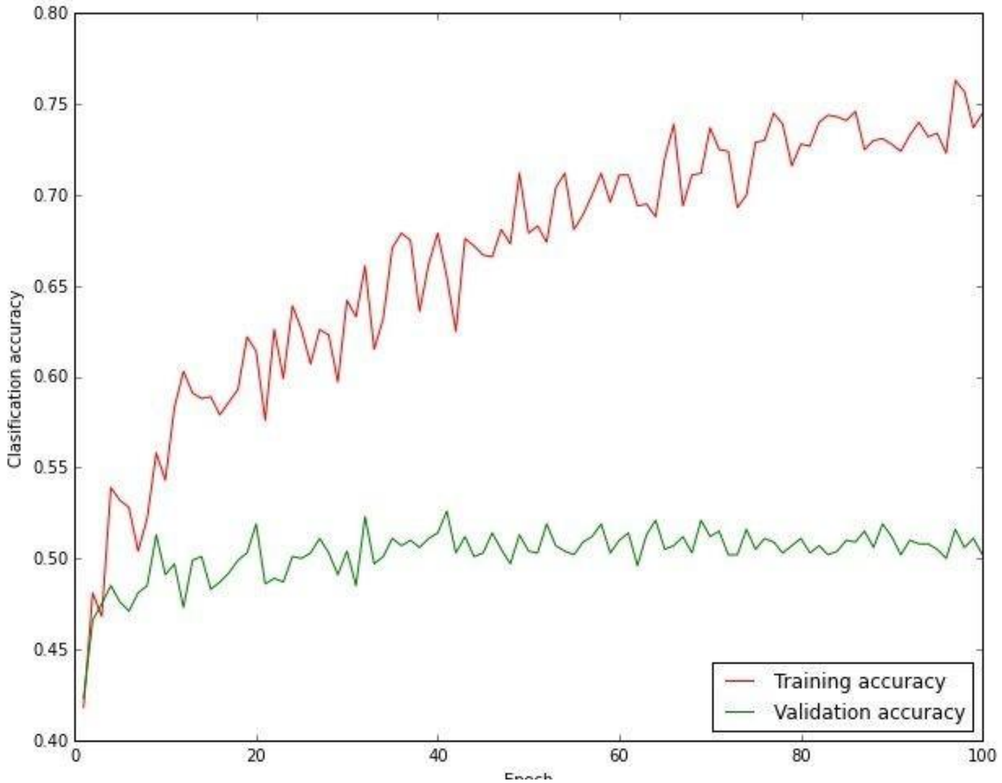
The Loss Curve during Training





Classification Accuracy

- Big gap between training accuracy and validation accuracy may imply overfitting => decrease model capacity?
- No gap may imply underfitting => increase model capacity?


Clasification accuracy
Epoch
Training accuracy
Validation accuracy