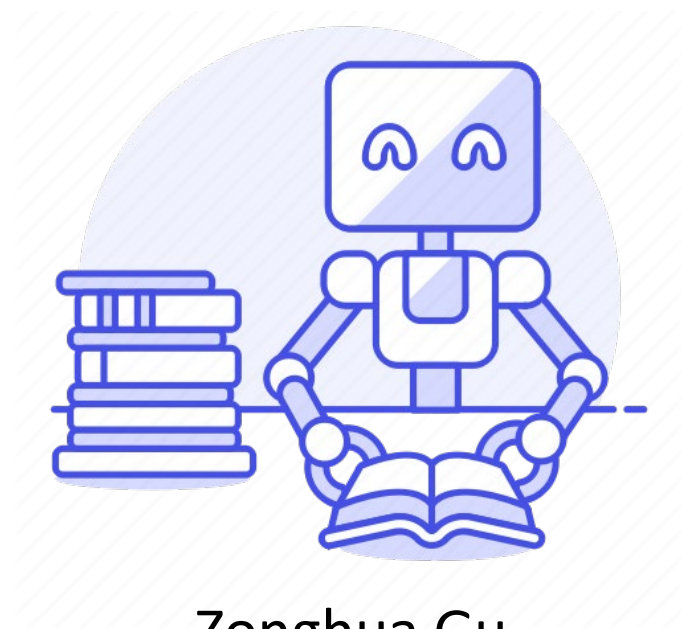


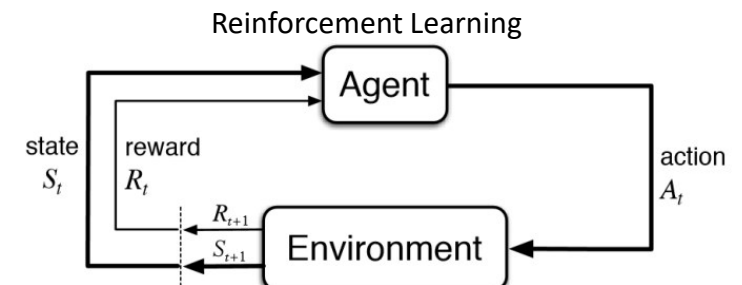
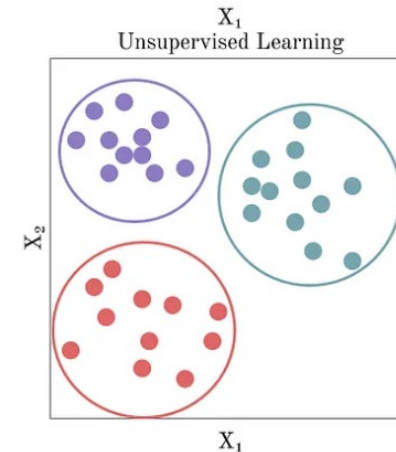
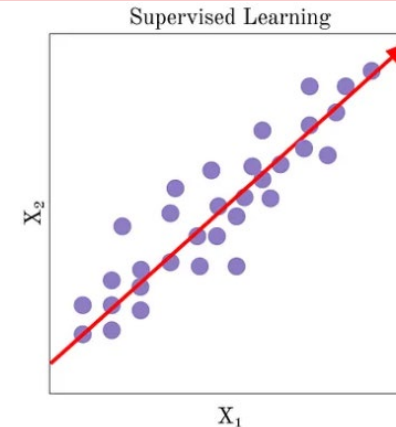
L2.1 Introduction to Machine Learning



Zonghua Gu
Fall 2026

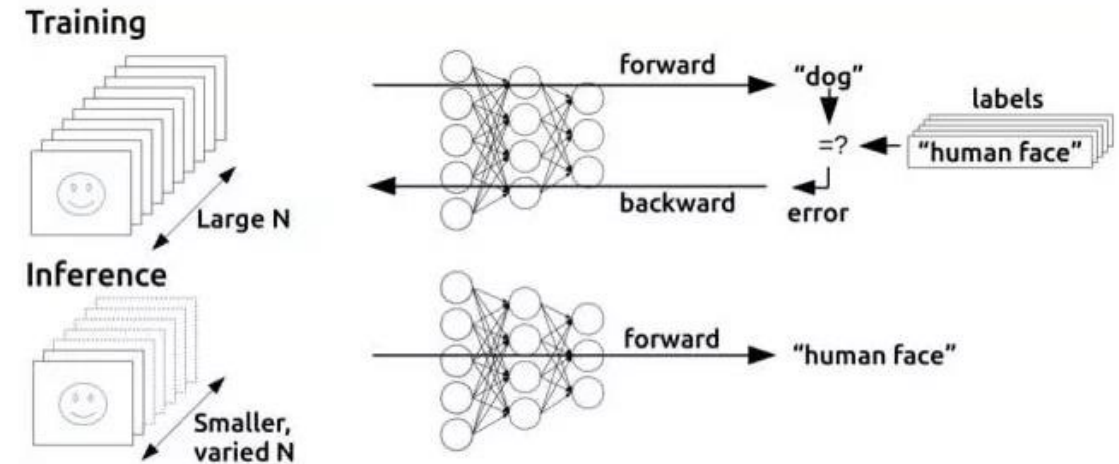
ML Taxonomy

- 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)
- 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.
 - Applications: Game playing (AlphaGo); Robotics; AD...



Training vs. Inference

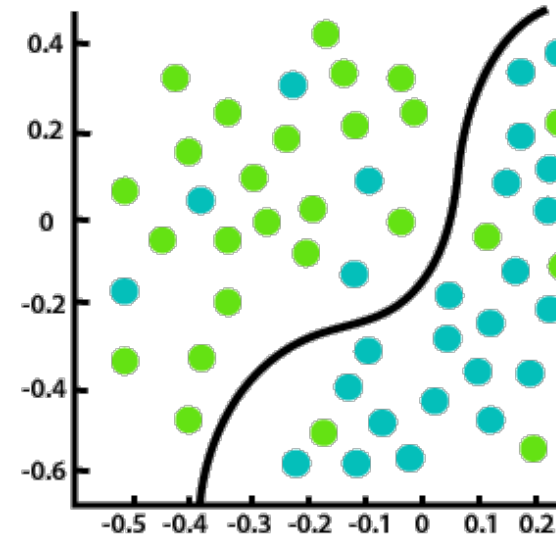
- Training: millions of iterations of forward pass + back propagation to adjust model params (e.g., NN weights); requires large CPU/GPU clusters and days/weeks of training time
- Inference (also called prediction): a single forward pass; can be run on edge devices



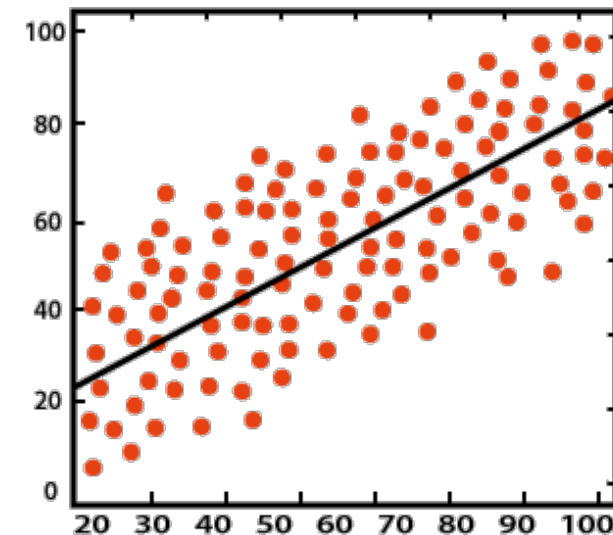
```
def train(train_images, train_labels):  
    # build a model for images -> labels...  
    return model  
  
def predict(model, test_images):  
    # predict test_labels using the model...  
    return test_labels
```

Supervised Learning: Classification and Regression

- Both are Supervised Learning algorithms that require ground-truth values as labels.
 - Classification is used to predict/classify discrete labels such as Male or Female, True or False, Spam or Not Spam, etc.
 - Regression is used to predict continuous values such as price, salary, age, etc.
- Loss functions measure how the predicted value differs from ground-truth value.



Classification

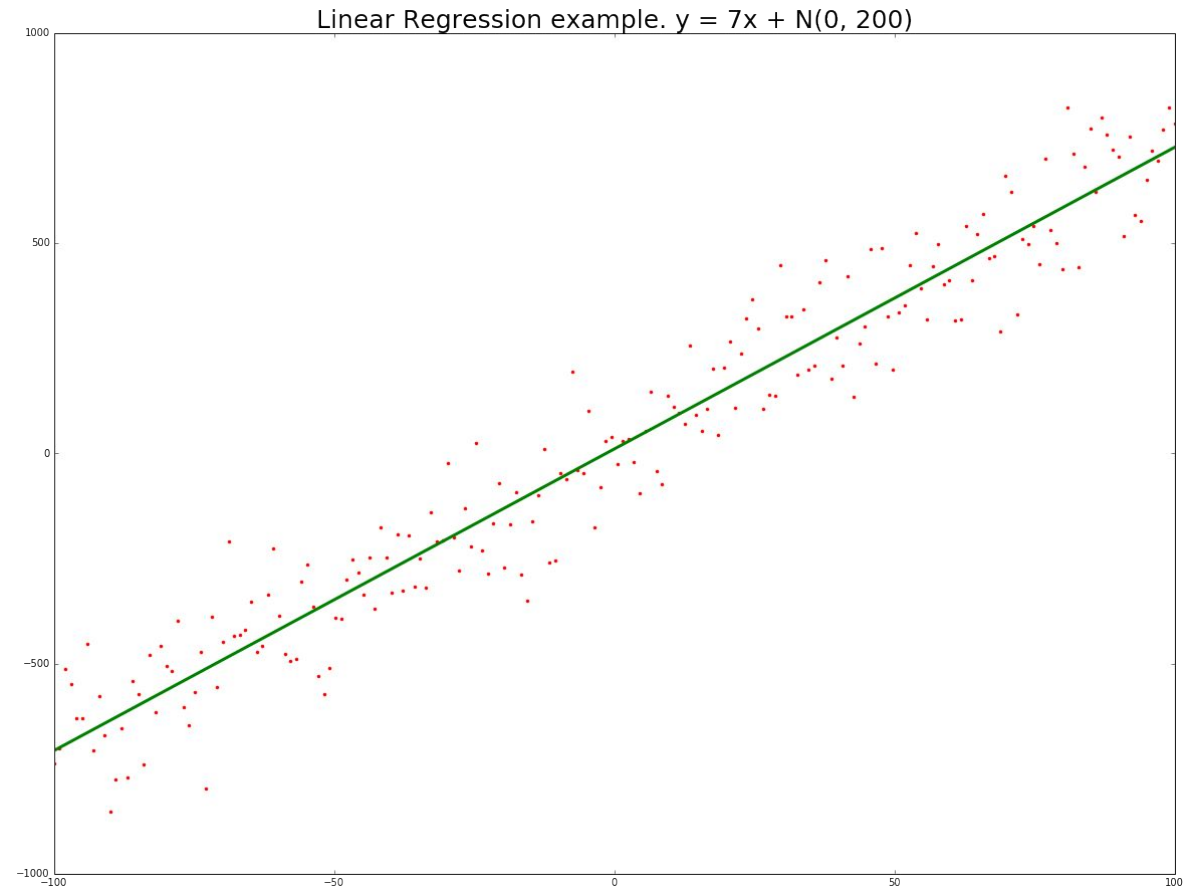


Regression

<https://www.javatpoint.com/regression-vs-classification-in-machine-learning>

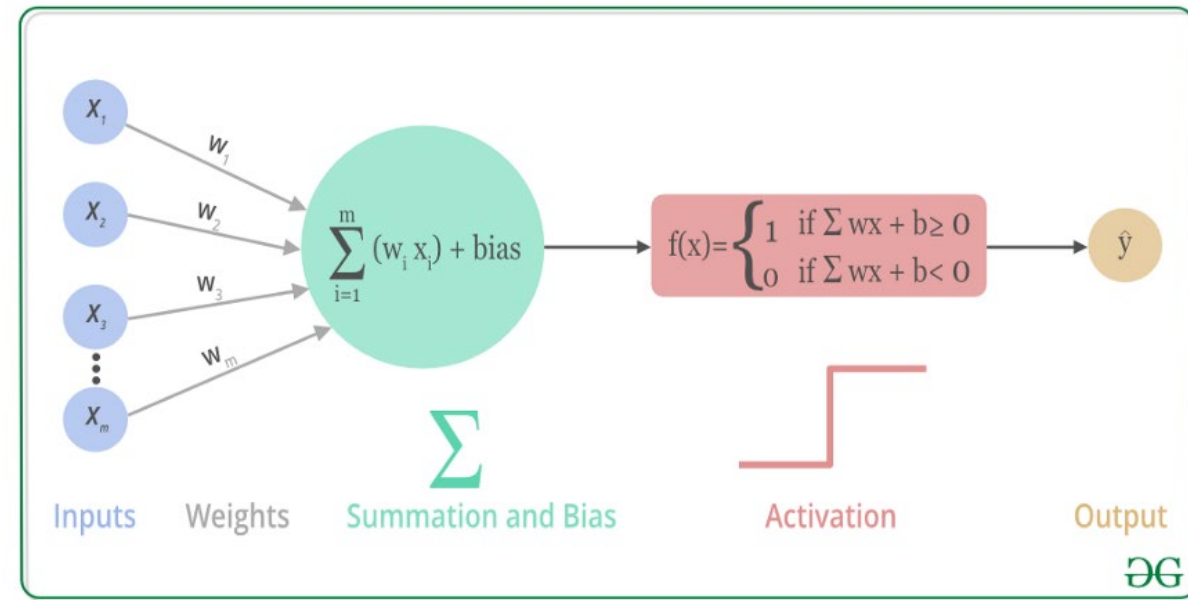
Linear Regression

- 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 $\mathbf{x} = [x_1 \ x_2 \ x_3]^T$, 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



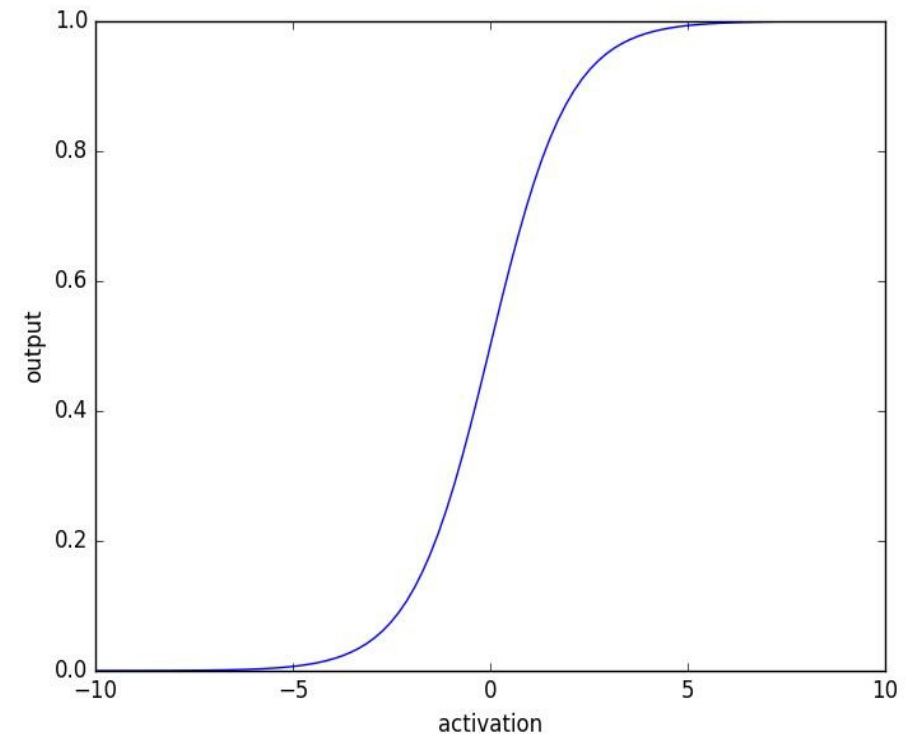
A Neuron and its Activation Function

- Activation function is a nonlinear monotonic function that acts like a “gate”: output is larger for larger input activation
 - Perceptron $y = \sigma(z) = \text{step}(wx + b)$ (activation function $f = \text{step}$ function, shown below)
 - Linear Regression if $y = z = wx + b$ (activation function $f = \text{identity}$ function)
 - Logistic Regression if $y = \sigma(z) = \sigma(wx + b)$ (activation function $f = \text{sigmoid}$ function)



Logistic Regression for Binary Classification

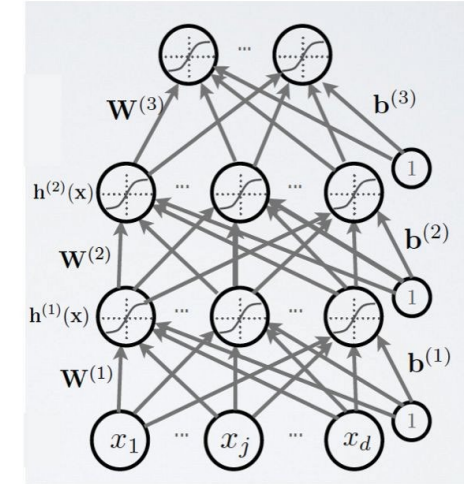
- Consider a binary classification problem: an input image x may be classified as a dog with probability $P(y = \text{dog}|x)$, a cat with probability $P(y = \text{cat}|x)$, with $P(y = \text{dog}|x) + P(y = \text{cat}|x) = 1.0$
- Logistic Regression: use sigmoid function $\sigma(z_i) = \frac{1}{1+e^{-z_i}}$ to map from the activation (also called the logit) to the output probability
- In addition to binary classification at the output layer, sigmoid may also be used as the non-linear activation function in the hidden layers of a NN



Sigmoid function

Deep Neural Networks and Activation Functions

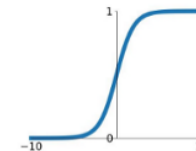
- We can stack many hidden layers to form a DNN if we have enough data and computing power to train it
- The high model capacity of DNN comes from non-linear activation functions in hidden layers
 - Without non-linear activation functions, a DNN with many layers can be collapsed into an equivalent single-layer NN
- Fully-Connected NNs
 - Number of params at the i-th layer is $(N_{i-1} + 1) * N_i$, where N_i is the number of neurons at the i-th layer. Can grow very large
 - Convolutional NNs have much fewer params



Slide Credit: Hugo Laroché NN course

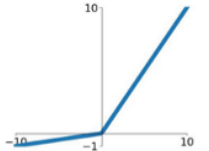
Sigmoid

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



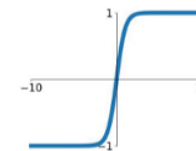
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

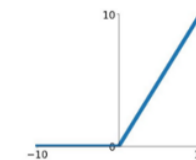


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

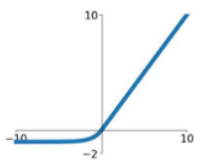
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

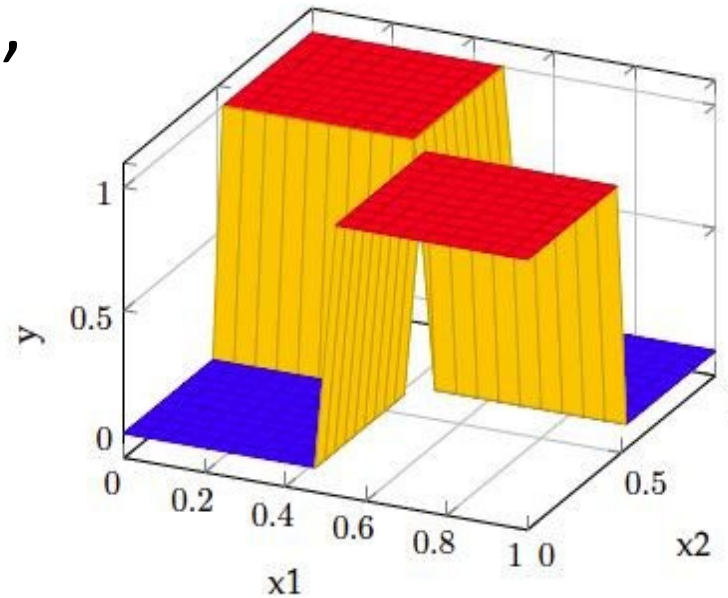


Common Activation Functions

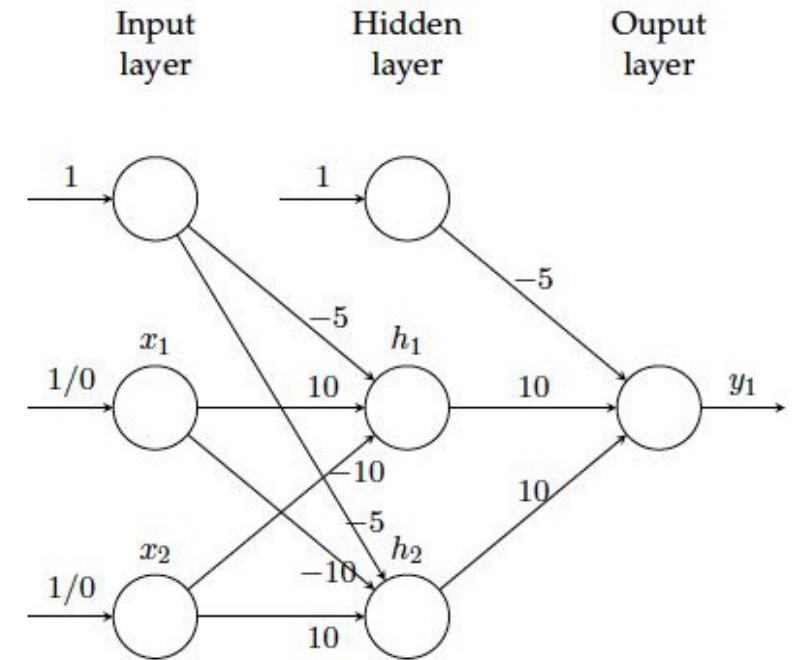
<https://laptrinhx.com/complete-guide-of-activation-functions-574622854/>

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

- Consists of one input, one hidden, and one output layer, with sigmoid activations

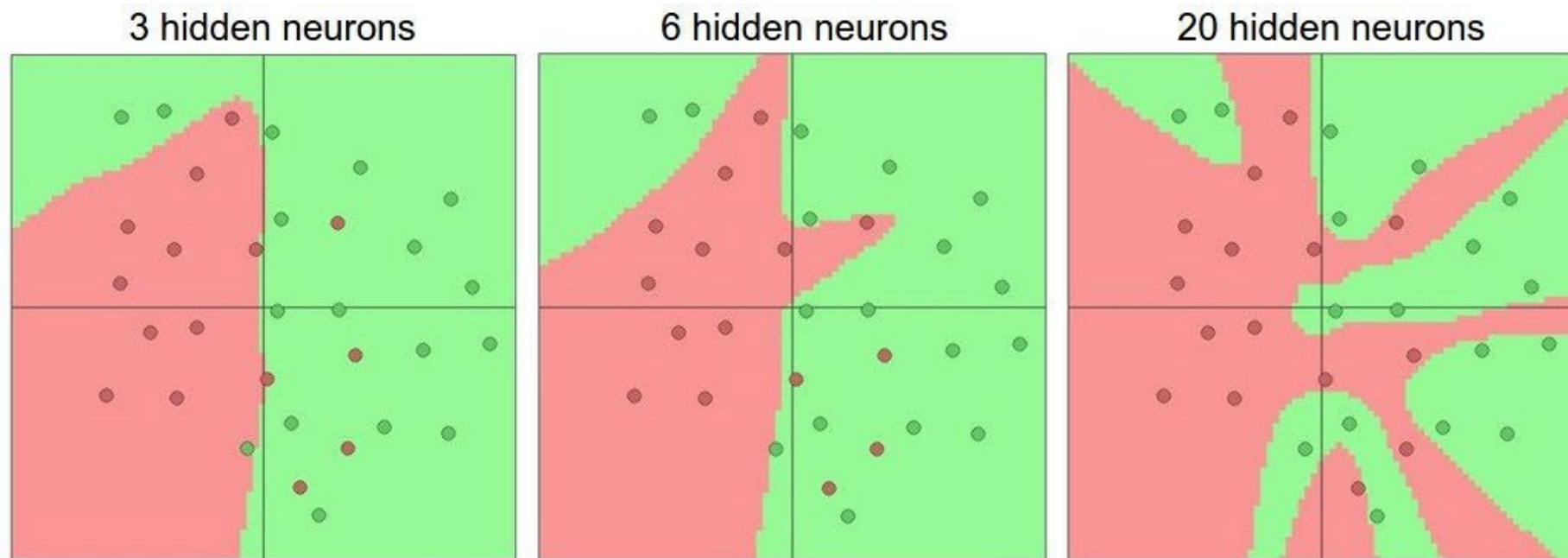


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



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 (good performance on training set, poor performance on test set)



Loss Functions

- Classification
 - Cross-Entropy Loss, Log Loss, Focal Loss, Exponential Loss, Hinge Loss...
- Regression
 - MSE (Mean Squared Error)/L2 Loss/Quadratic Loss, MAE (Mean Absolute Error)/L1 Loss, Huber Loss, Log Cosh Loss, Quantile Loss...

NN for Multi-Class Classification

- Consider a NN defining the model $h_{\theta}: \mathcal{X} \rightarrow \mathbb{R}^k$, 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
 - (Here h_{θ} does not include the last SoftMax layer that outputs the prediction scores)
- 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 $\sigma: \mathbb{R}^k \rightarrow \mathbb{R}^k$ computes a vector of predicted probabilities $\sigma(z): \mathbb{R}^k$ that sum to 1.0, from a vector of logits $z: \mathbb{R}^k$ in the last hidden layer (the penultimate layer), 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 :

$$\text{Loss}(x, y; \theta) = -\log \sigma(h_\theta(x))_y = -$$

$$\log \left(\frac{\exp(h_\theta(x)_y)}{\sum_{j=1}^k \exp(h_\theta(x)_j)} \right) =$$

$$\log(\sum_{j=1}^k \exp(h_\theta(x)_j)) - h_\theta(x)_y$$

- Minimizing $\text{Loss}(h_\theta(x), y)$ amounts to maximizing the logit $(h_\theta(x))_y$ corresponding to the correct label y

True distribution:

0%	0%	0%	0%	100%	0%	0%
Cat	Dog	Fox	Cow	Red Panda	Bear	Dolphin

Predicted distribution:

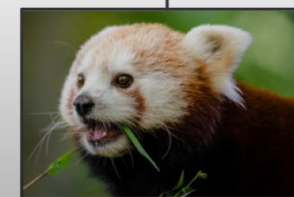
2%	30%	45%	0%	25%	5%	0%
Cat	Dog	Fox	Cow	Red Panda	Bear	Dolphin

Classifier

Cross-Entropy Loss:

$$H(\mathbf{p}, \mathbf{q}) = -\sum_i p_i \log(q_i) \\ = -\log(0.25) = 1.386$$

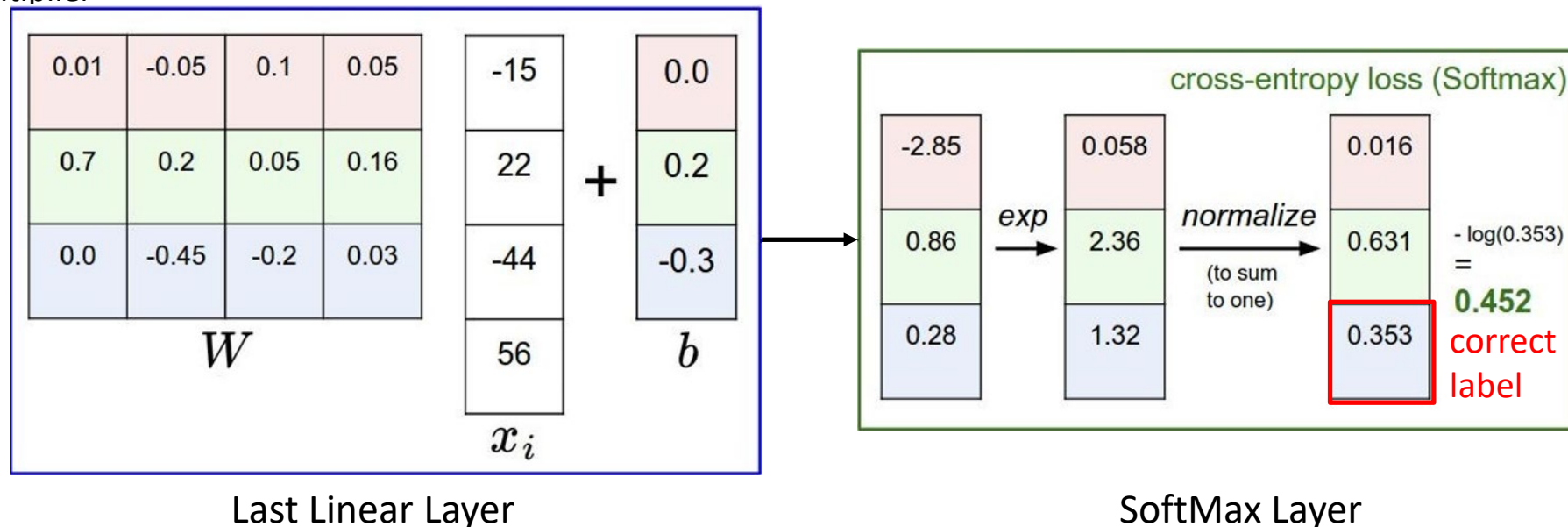
$$\log_2(x) = \log(x) / \log(2)$$



Aurélien Géron A Short Introduction to Entropy, Cross-Entropy and KL-Divergence (YouTube)

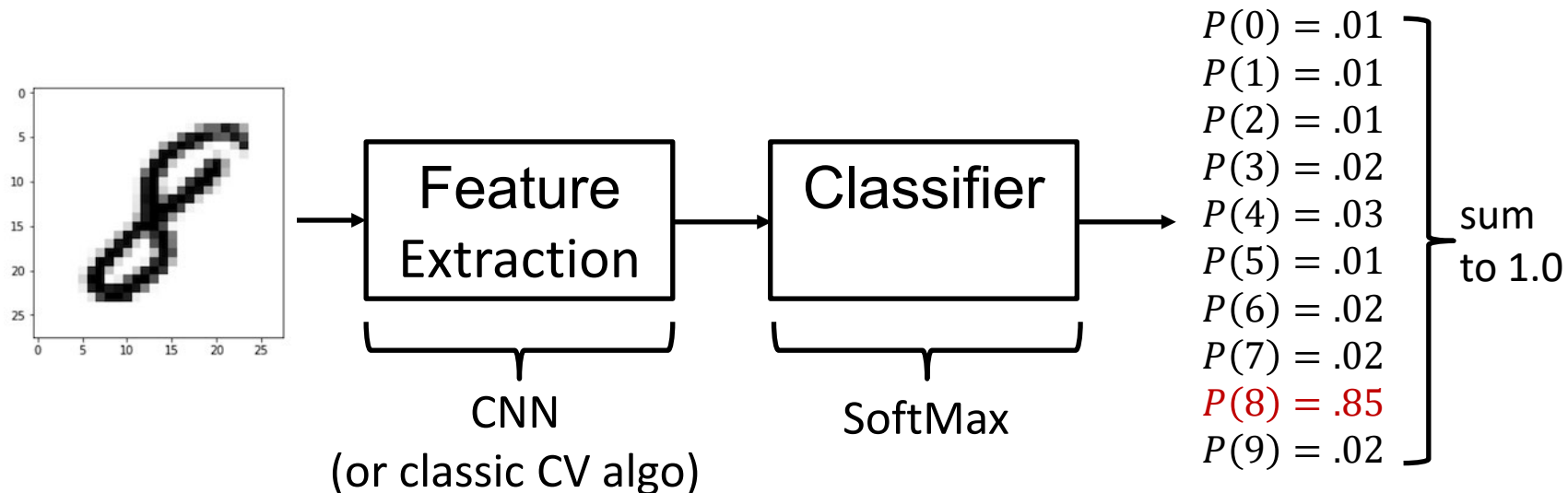
CE Loss Example

- 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 $[1 \quad 2 \quad 3]^T$. Loss = $-\log_{10} 0.353 = 0.452$, assuming correct label $y_i = 3$
 - Logits: $[e^{-2.85}, e^{.86}, e^{.28}] = [.058, 2.36, 1.32]$
 - Normalize by $e^{-2.85} + e^{.86} + e^{.28} = 3.738$ to get SoftMax scores $[\frac{.058}{3.738}, \frac{2.36}{3.738}, \frac{1.32}{3.738}] = [.016, .631, .353]$
 - We can set loss to be either $-\log_{10} 0.353$ or $-\log_2 0.353$, the log base of 2 or 10 do not matter as they differ by a constant multiplier



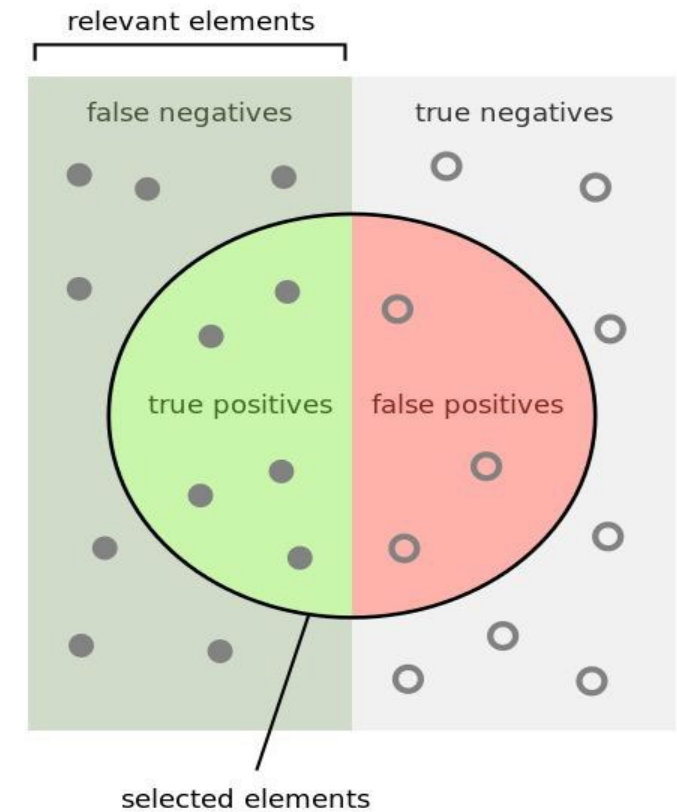
Multi-Class Image Classification

- 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



Binary Classification Metrics

- The relevant class is considered “Positive” in a binary classifier
- e.g., consider a medical test that aims to diagnose people with a certain disease. “Positive” denotes sick (has disease), and “Negative” denotes healthy (no disease)
 - True Positive (TP): a sick person is diagnosed as sick
 - True Negative (TN): a healthy person is diagnosed as healthy
 - False Positive (FP): a healthy person is misdiagnosed as sick
 - False Negative (FN): a sick person is misdiagnosed as healthy
- Never Forget Again! // Precision vs Recall with a Clear Example of Precision and Recall by Kimberly Fessel
 - <https://www.youtube.com/watch?v=qWfzIYCvBqo>

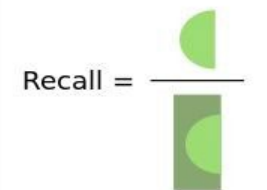


How many selected items are relevant?



Precision =

How many relevant items are selected?



Recall =

$$\text{Precision} = \frac{TP}{TP+FP}, \text{ Recall (TPR)} = \frac{TP}{TP+FN}$$

Example Confusion Matrix 1

- Precision = $\frac{TP}{TP+FP} = \frac{1}{1+7} = .125$
 - When the classifier predicts positive, it is correct 12.5% of the time
- Recall (True Positive Rate, TPR) = $\frac{TP}{TP+FN} = \frac{1}{1+2} \approx .333$
 - Among all the positive cases, the classifier correctly classifies 33.3% of them as positive
- $F1 = 2 * \frac{\text{Precision} * \text{Recall}}{(\text{Precision} + \text{Recall})} = 2 * \frac{.333 * .125}{.333 + .125} = .182$
- False Positive Rate (FPR) = $\frac{FP}{FP+TN} = \frac{7}{7+90} \approx .072$
 - Among all the negative cases, the classifier misclassifies 7.2% of them as positive
- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN} = \frac{1+90}{1+90+7+2} = .91$
 - The classifier makes the correct prediction 91% of the time
- Positive correlation between TPR vs. FPR
- In general, negative correlation between precision and recall (but not strictly monotonic)

		Ground Truth	
		Positive	Negative
Predicted	Neg	False Negative (FN)=2	True Negative (TN)=90
	Pos	True Positive (TP)=1	False Positive (FP)=7

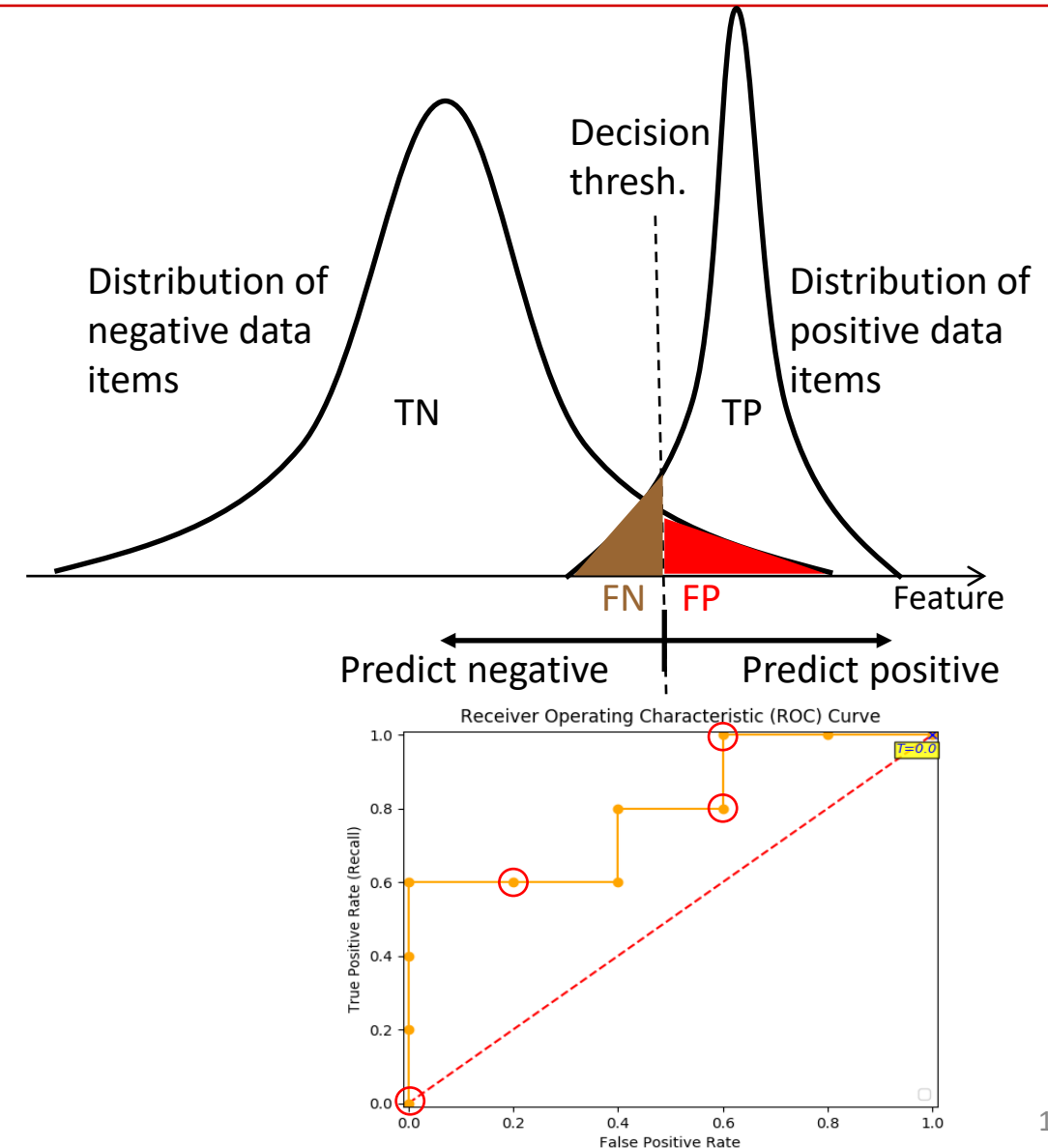
Example Confusion Matrix 2

- Precision = $\frac{TP}{TP+FP} = \frac{0}{0+0}$ (ill-defined)
 - When the classifier predicts positive, it is correct ?% of the time (since it never predicts positive, the question is ill-defined)
- Recall (TPR) = $\frac{TP}{TP+FN} = \frac{0}{0+3} = 0$
 - Among all the positive cases, the classifier correctly classifies 0% of them as positive
- FPR = $\frac{FP}{FP+TN} = \frac{0}{0+97} = 0$
 - Among all the negative cases, the classifier misclassifies 0% of them as positive
- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN} = \frac{0+97}{0+97+0+3} = .97$
 - The classifier makes the correct prediction 97% of the time
- A medical test that never makes any positive diagnoses is highly accurate for a rare disease (diagnose everyone to be healthy), but completely useless

		Ground Truth	
		Positive	Negative
Predicted	Neg	False Negative (FN)=3	True Negative (TN)=97
	Pos	True Positive (TP)=0	False Positive (FP)=0

ROC and AUC

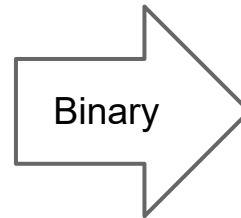
- Binary classification is typically based on a decision threshold parameter. Moving the decision threshold will cause FPR and TPR to move in the same direction
 - Lower threshold for positive prediction leads to higher FPR and higher TPR, and vice versa
- Receiver Operating Characteristic (ROC) Curve plots FPR (x-axis) vs. TPR (y-axis); Area Under the Curve (AUC) is the area under ROC ($.5 \leq AUC \leq 1$, since $FPR \leq TPR$)
 - Fig shows an example with 4 points (FPR, TPR) highlighted: (0,0), (.2, .6), (.6, .8), (.6,1.0)
 - The ideal ROC curve: $FPR \equiv 0, TPR \equiv 1, AUC = 1$, with $FP = FN = 0$,
 - The worst ROC curve; $FPR \equiv TPR, AUC = .5$ (dotted line)



Confusion Matrix

- Binary classification is a special case of multi-class classification

$$Accuracy = \frac{\sum_{i=1}^3 x(i, i)}{\sum_{i=1}^3 \sum_{j=1}^3 x(i, j)}$$

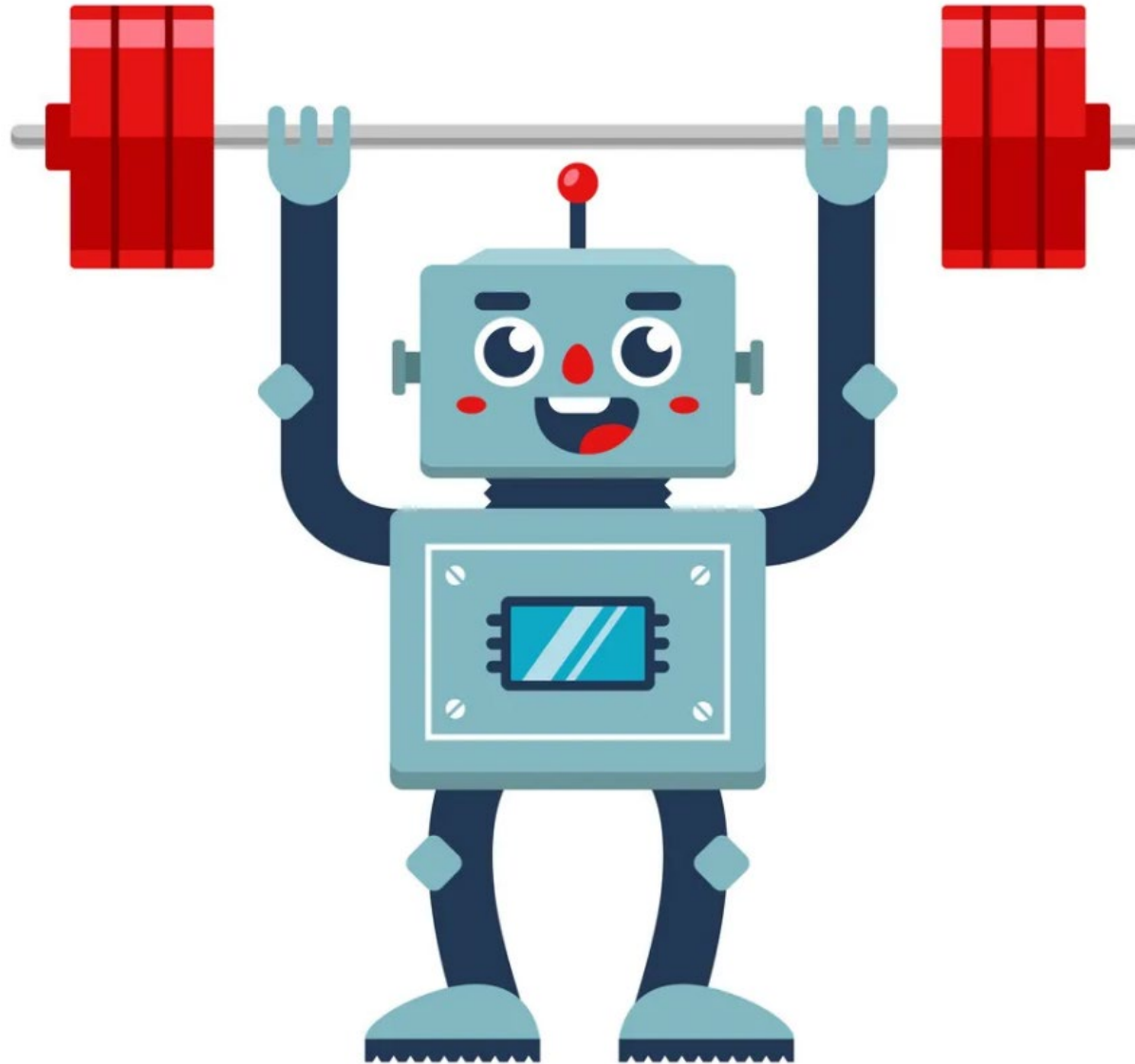


$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

		Ground Truth		
		Cls1	Cls2	Cls3
Pred.	Cls3			
	Cls2			
	Cls1			

		Ground Truth	
		Pos	Neg
Pred.	Neg	FN	TN
	Pos	TP	FP

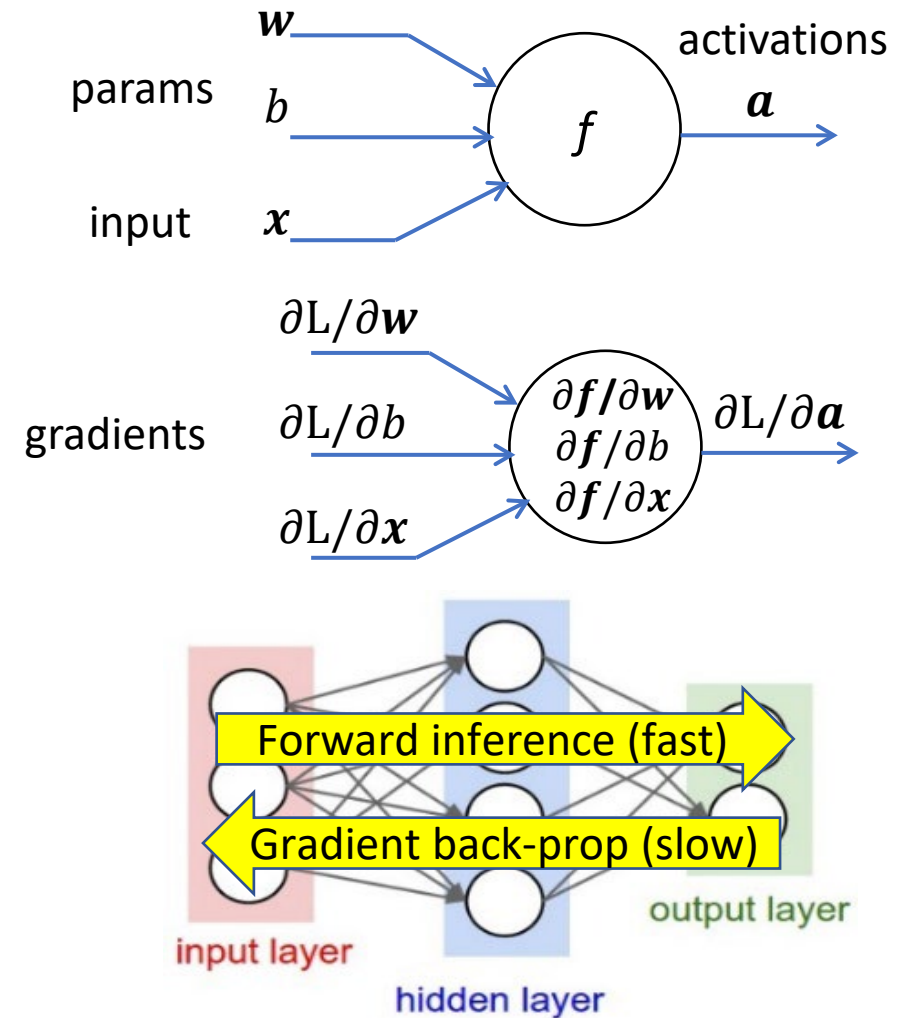
Training Neural Networks



Back-Propagation for NN Training

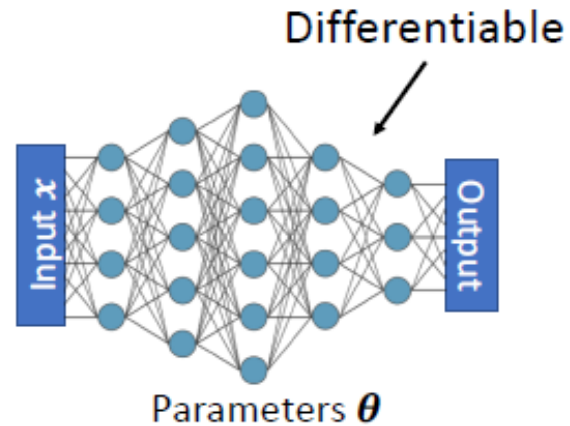
- Consider a single neuron denoted as a function f
- Forward inference: computing its output activation $\mathbf{a}$ based on its input $\mathbf{x}$, weight $\mathbf{w}$, and bias $\mathbf{b}$: $\mathbf{a} = \mathbf{w}\mathbf{x} + \mathbf{b}$
- Back-propagation: computing gradients with chain rule, and use gradient descent to update weights and biases:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{f}}{\partial \mathbf{w}}, \frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{f}}{\partial \mathbf{b}}, \frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$$

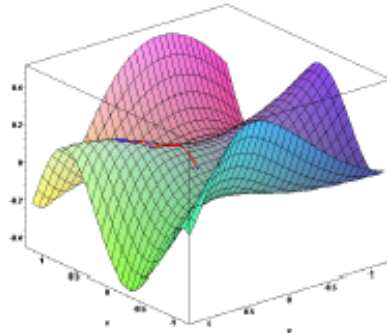


Gradient Descent

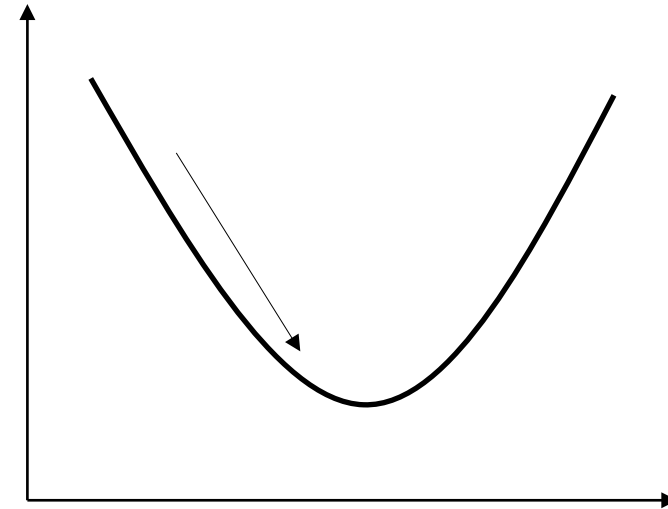
- To minimize the expected loss over training dataset D , use gradient descent $\theta \leftarrow \theta - \alpha \nabla_{\theta} \text{Loss}(x, y; \theta)$
- Loss surface of a DNN is highly non-convex; can only hope to find “reasonably good” local minima



Can use gradient descent method to find good θ

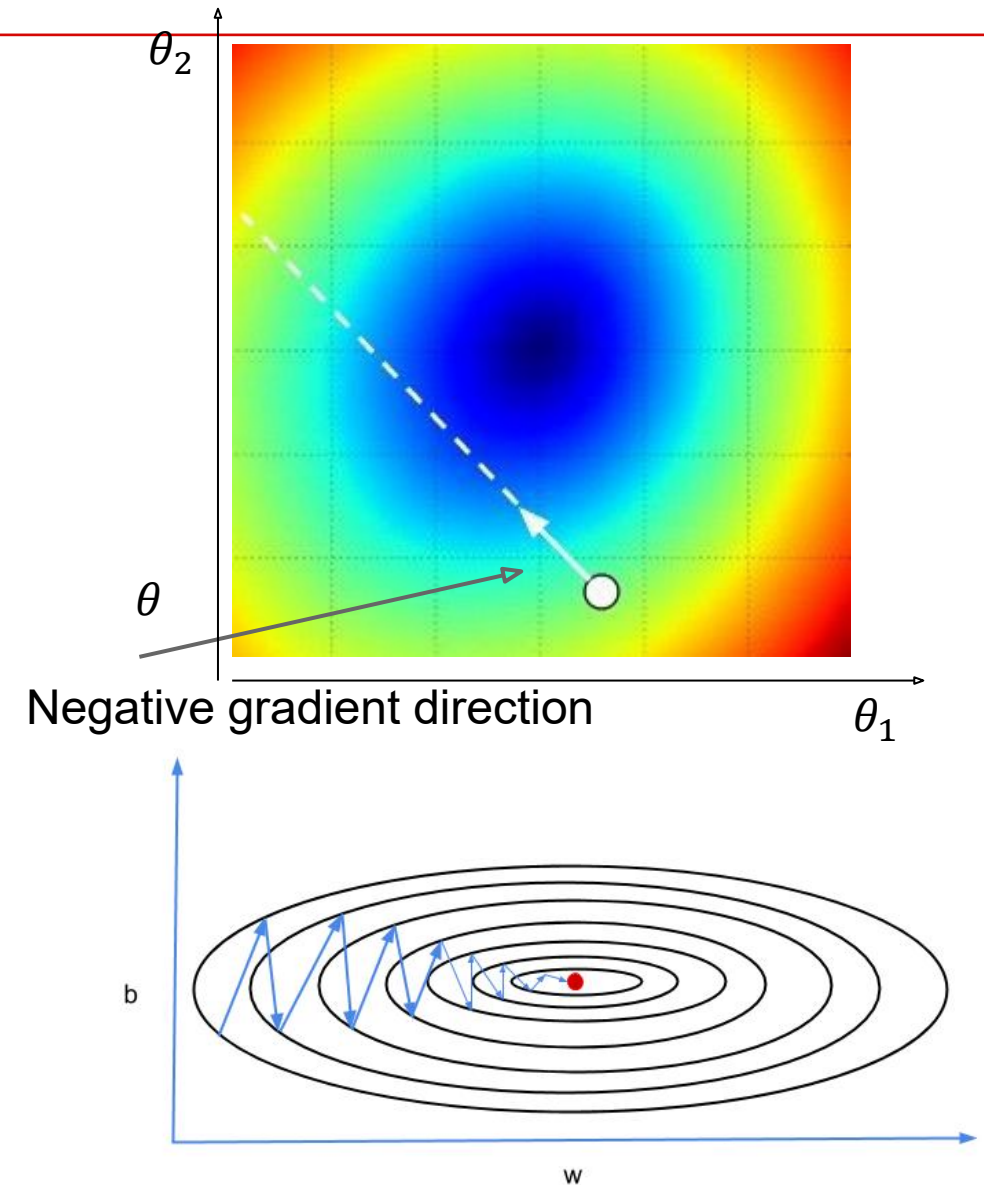


$$\mathbb{E}_{(x,y) \sim D} \text{Loss}(x, y; \theta)$$



Gradient Descent

- Steepest descent may result in inefficient zig-zag path
 - More advanced GD methods exploit momentum and/or adaptive learning rates, e.g., Nesterov, AdaGrad, RMSProp, Adam...
- 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: $\mathcal{B} = \{x_1 \dots x_m\}$;

Parameters to be learned: γ, β

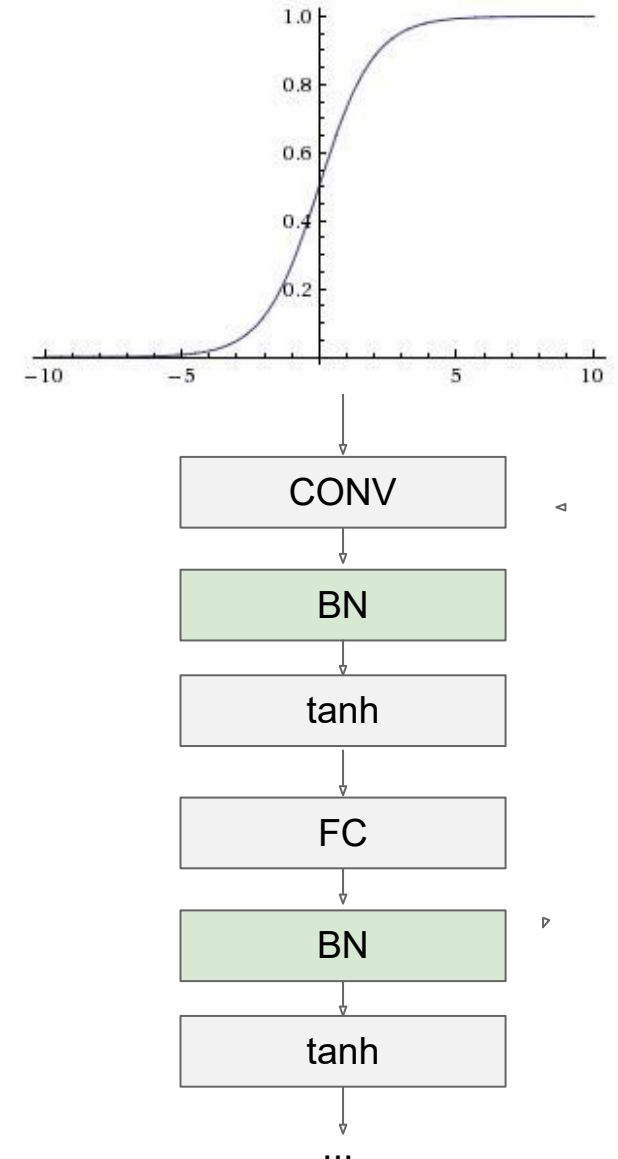
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

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

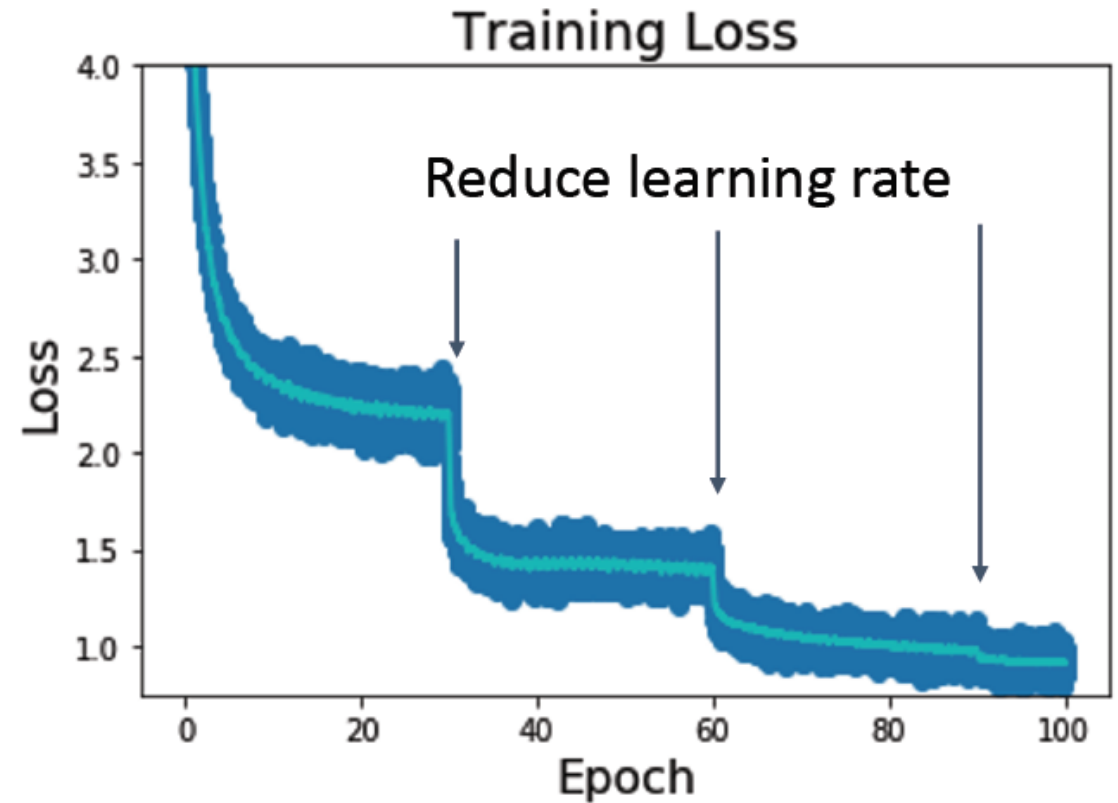
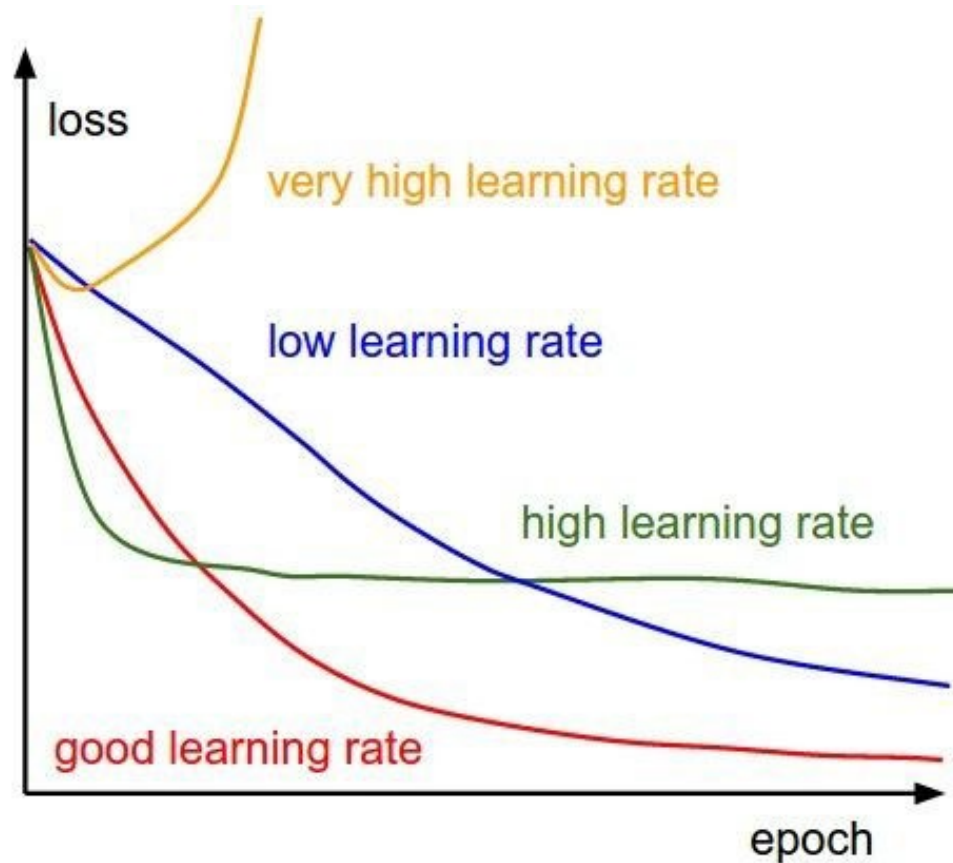
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

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

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



Learning Rate Schedule during Training



Hyperparameter Optimization

- Example hyperparams
 - Network architecture
 - Learning rate, its decay schedule, update type
 - Regularization (L2/Dropout strength)
- Grid search vs. random search
 - If a function f of two variables can be approximated by another function of one variable ($f(x_1, x_2) \approx g(x_1)$), then we say that f has a low effective dimension. Fig. 1 illustrates how point grids and uniformly random point sets differ in how they cope with low effective dimensionality
- Bergstra et al argued for random search for hyper-parameter optimization instead of grid search

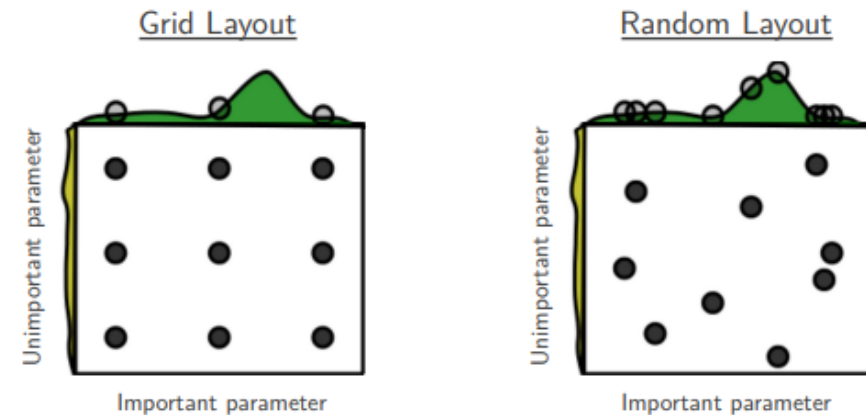
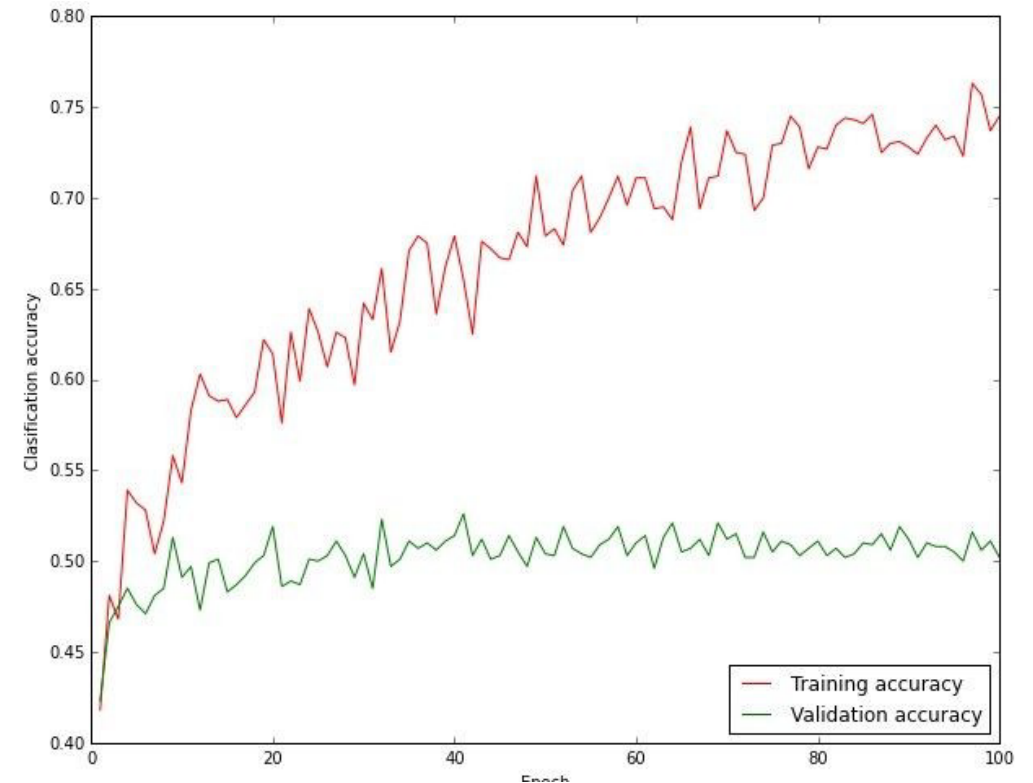


Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g . This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." *Journal of machine learning research* 13.2 (2012).

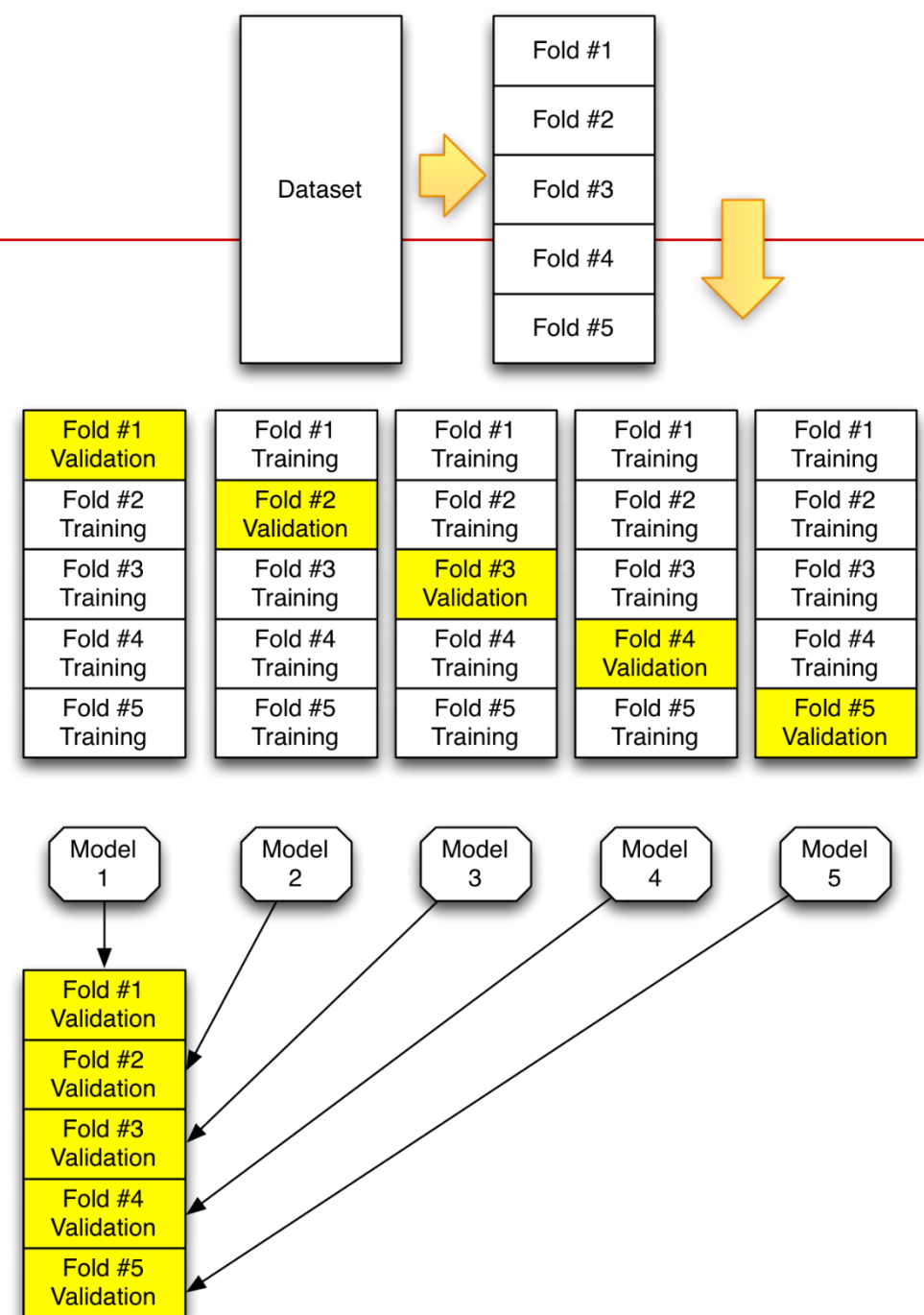
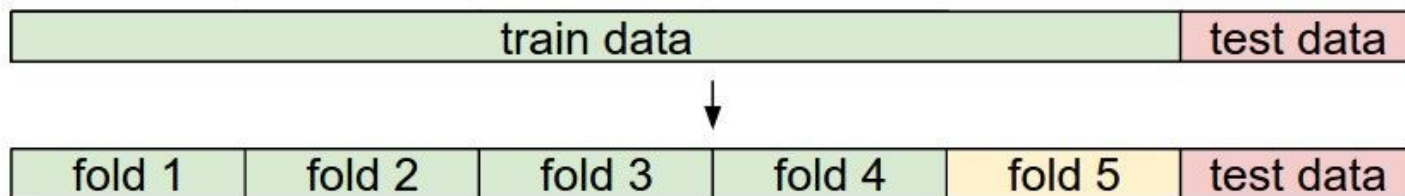
Classification Accuracy

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



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



Data Augmentation for Enlarging Training Dataset

- Mirroring, random cropping, color shifting, rotation, shearing, local warping...

Mirroring



Random Cropping



Color Shifting

