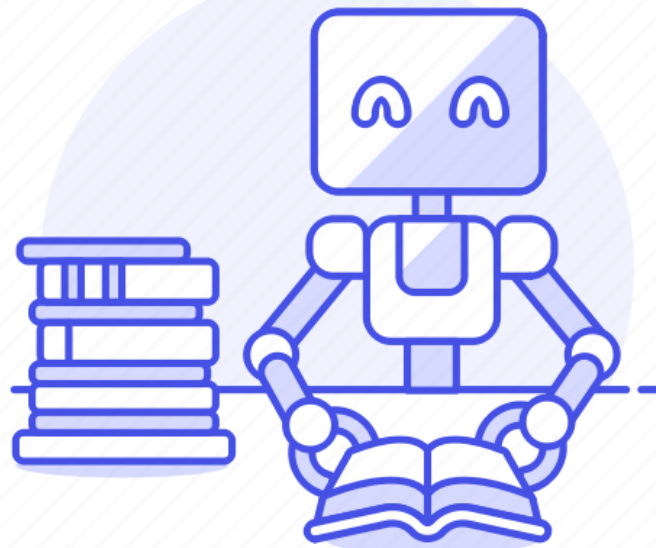


L3 Introduction to Machine Learning

Z. Gu 2021

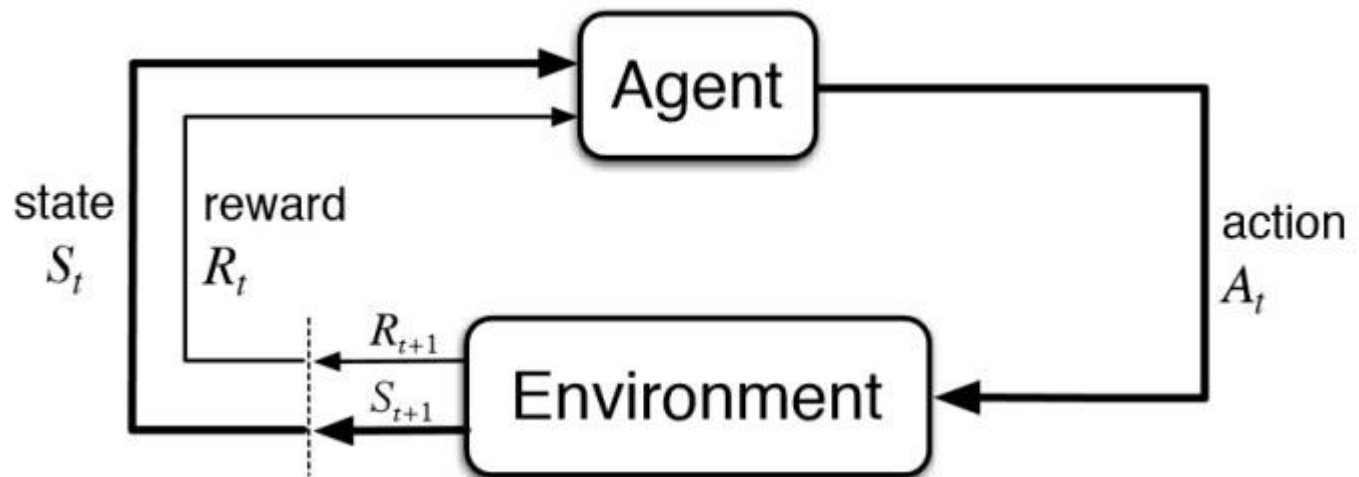


ML Taxonomy

- 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

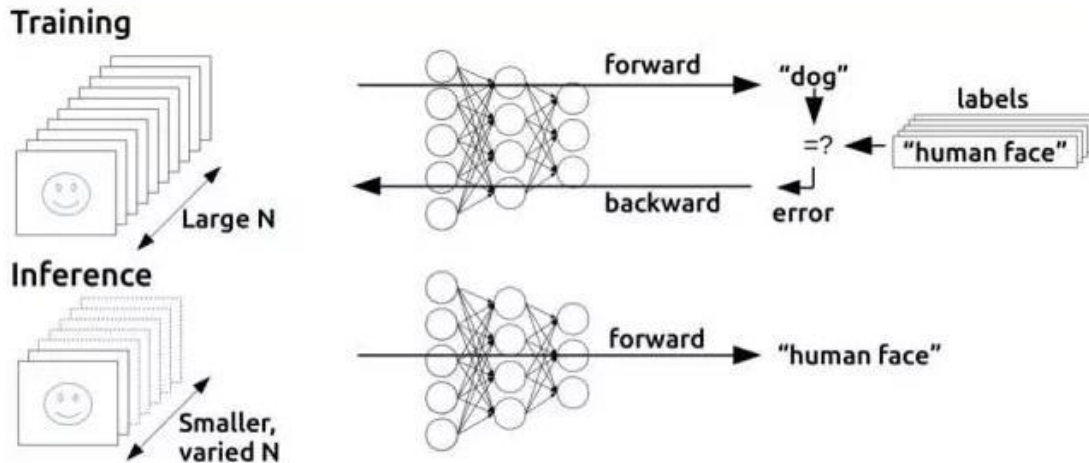
ML Taxonomy

- 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

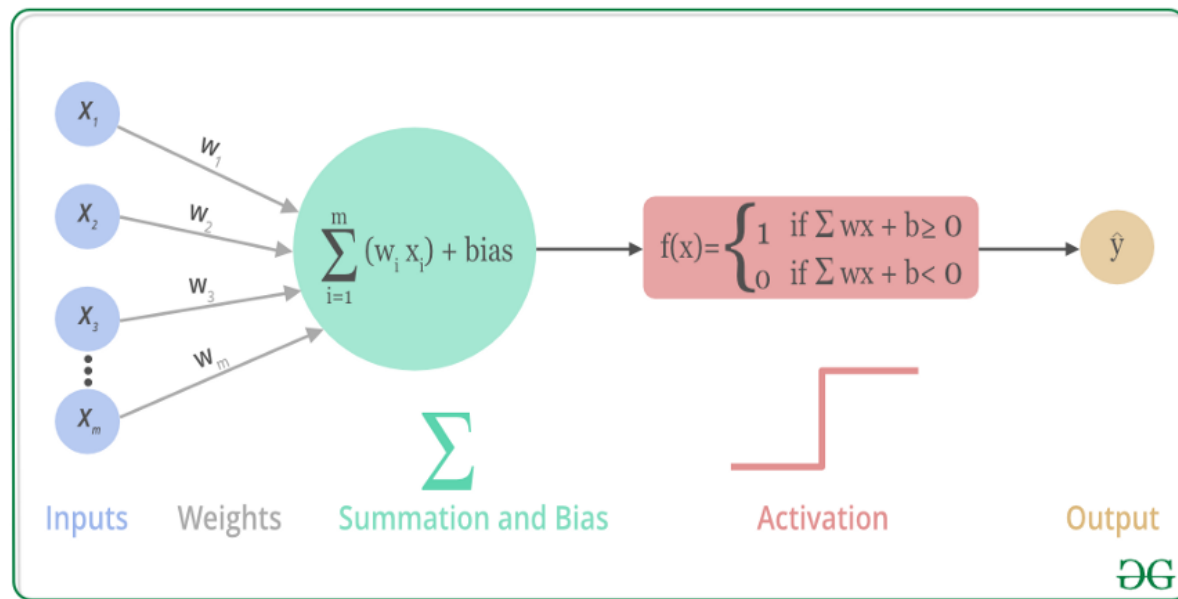
- Training: millions of iterations of forward pass + back propagation to adjust model params (e.g., NN weights); requires large CPU/GPU clusters and days/weeks of training time
- Inference (also called prediction): a single forward pass; can be run on edge devices



```
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
```

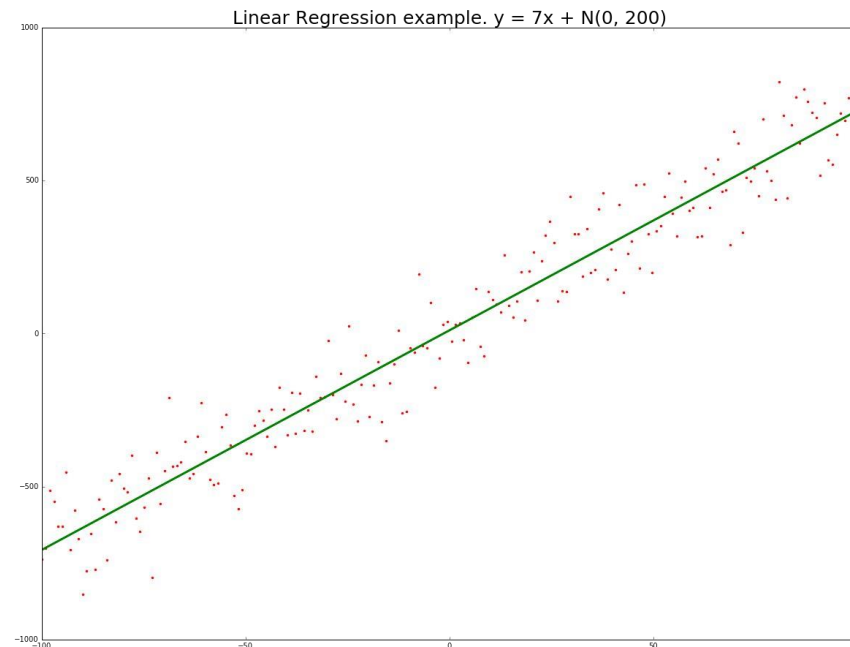
A Neuron and its Activation Function

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



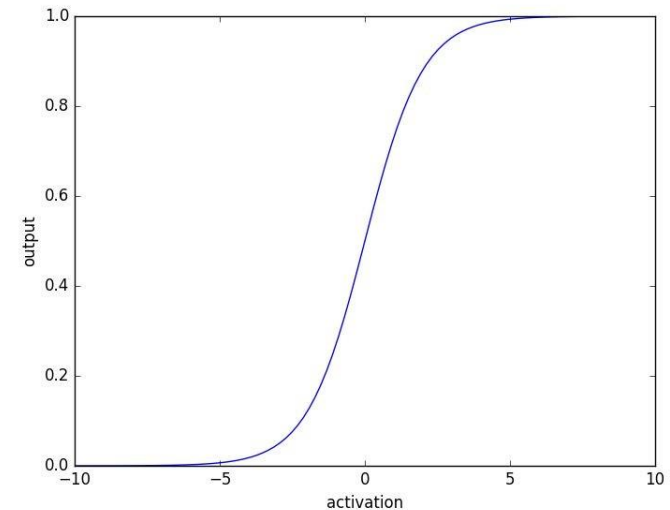
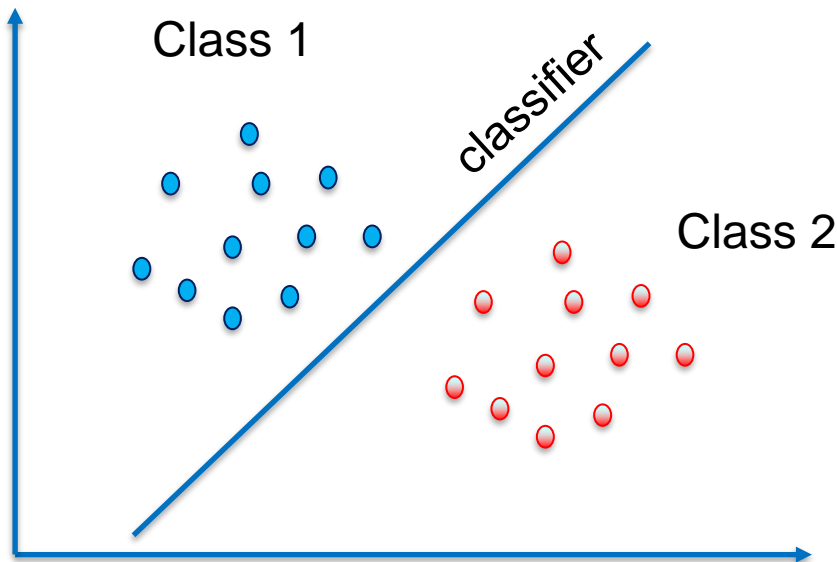
Linear Regression for Regression

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

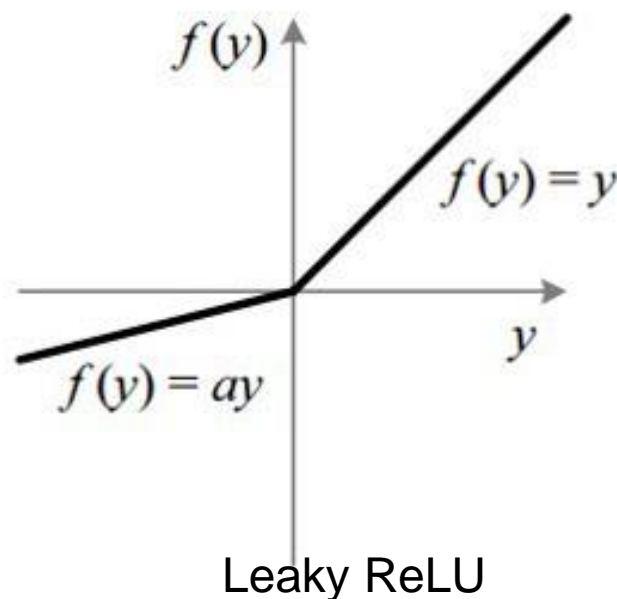
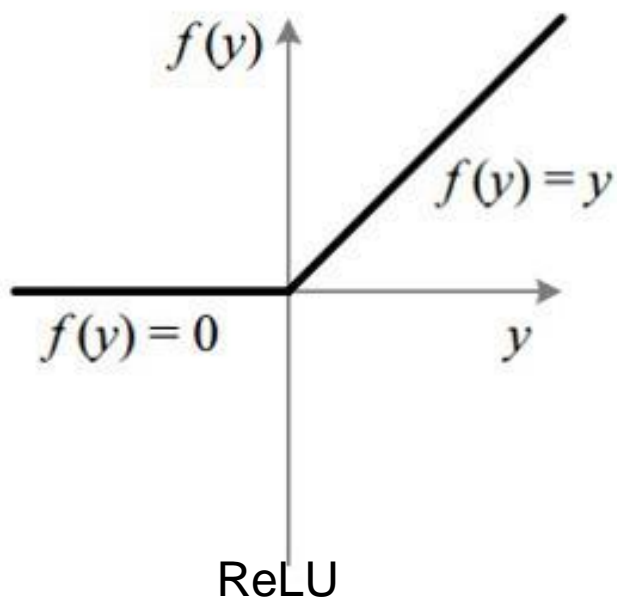
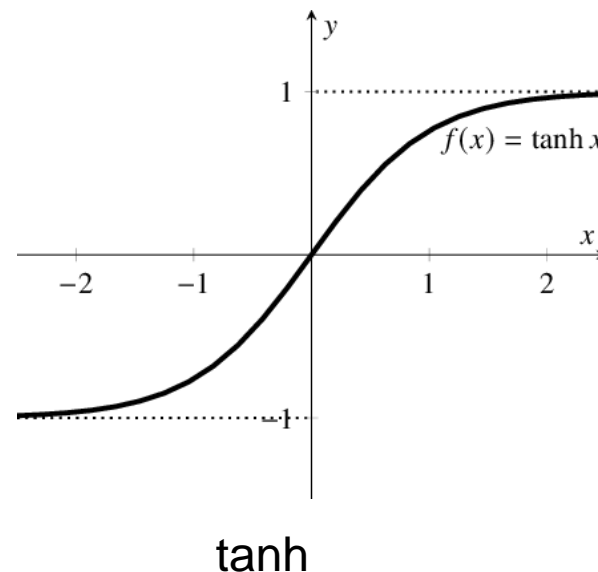
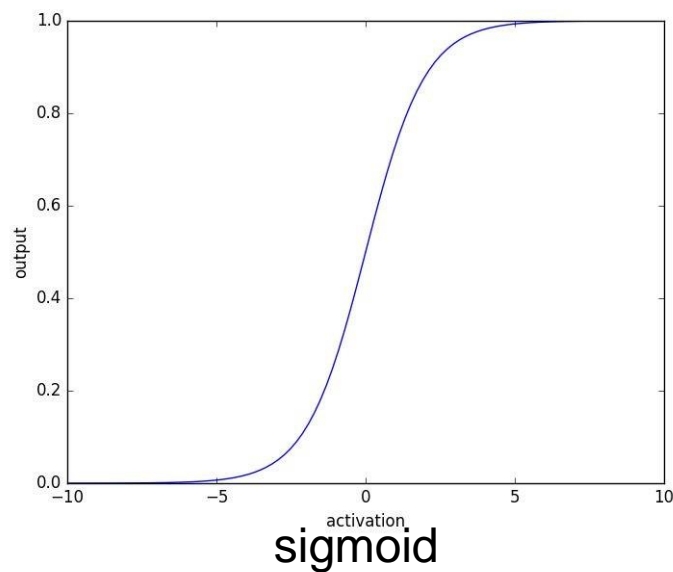


Logistic Regression for Binary Classification

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



Common Activation Functions used in DL

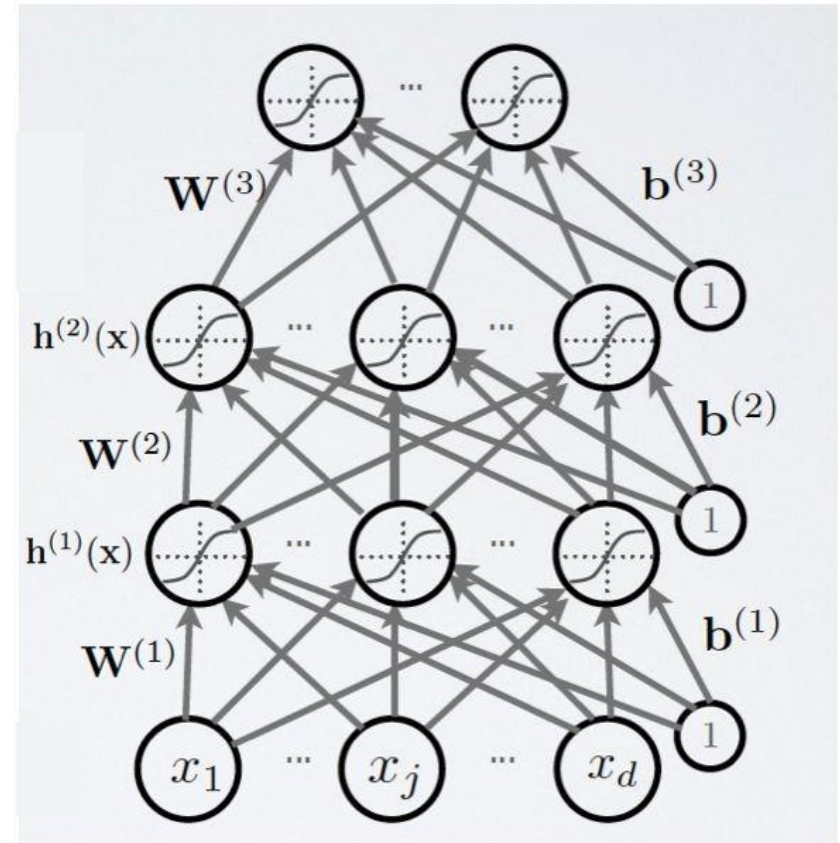


Deep Neural Networks

- 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 with much fewer params in the next lecture)

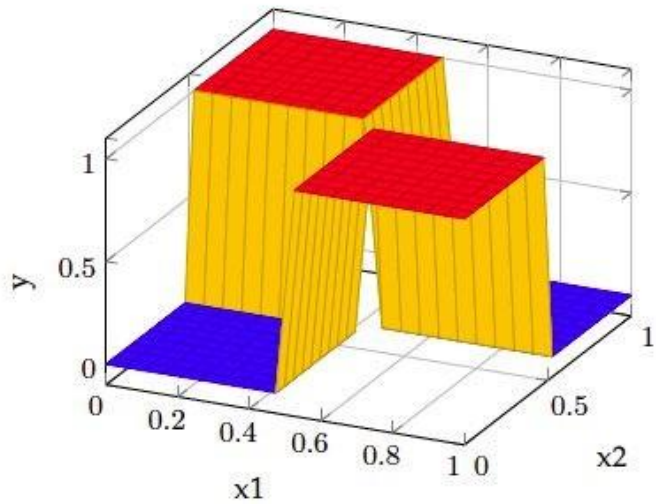


Slide Credit: Hugo Laroché NN course

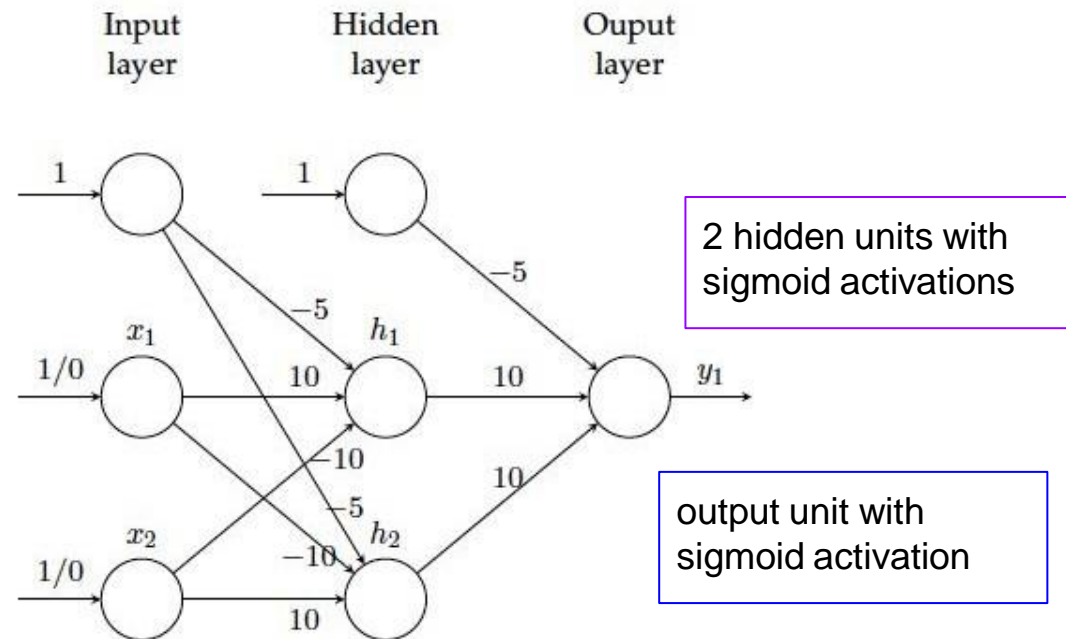
A 3-layer NN

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

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

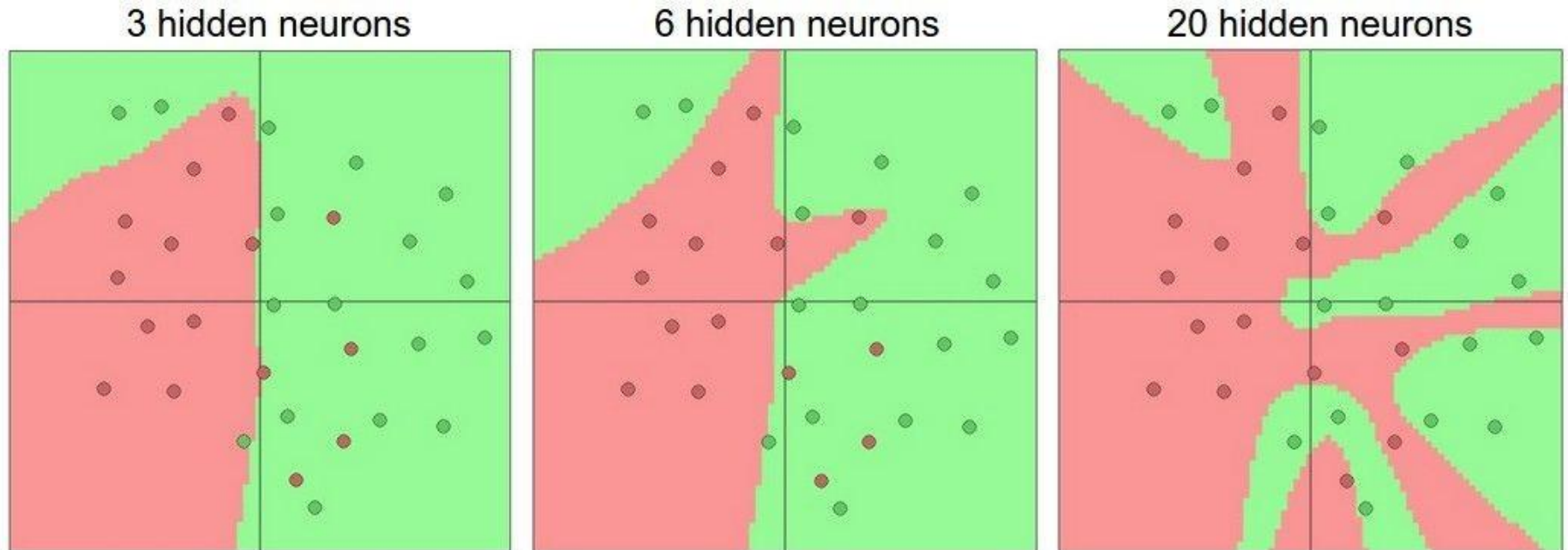


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



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



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
 - 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 $\sigma: \mathbb{R}^k \rightarrow \mathbb{R}^k$ 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 :

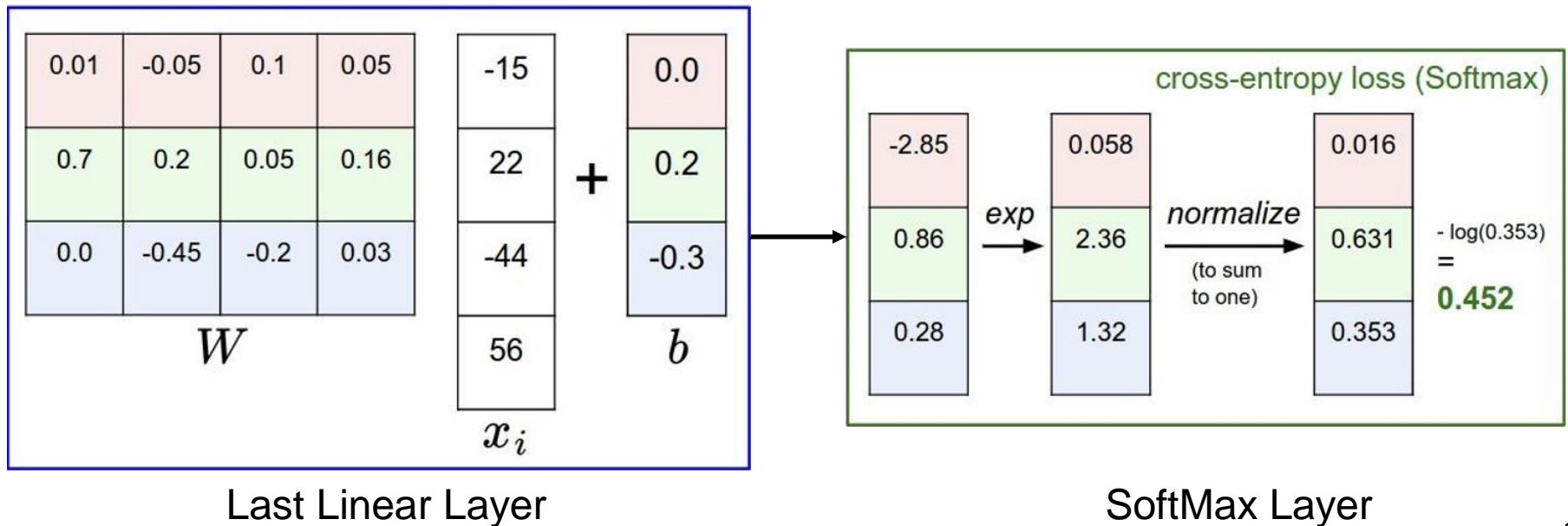
- $\text{Loss}(h_\theta(x), y) = -\log \sigma(h_\theta(x))_y = -$

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

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

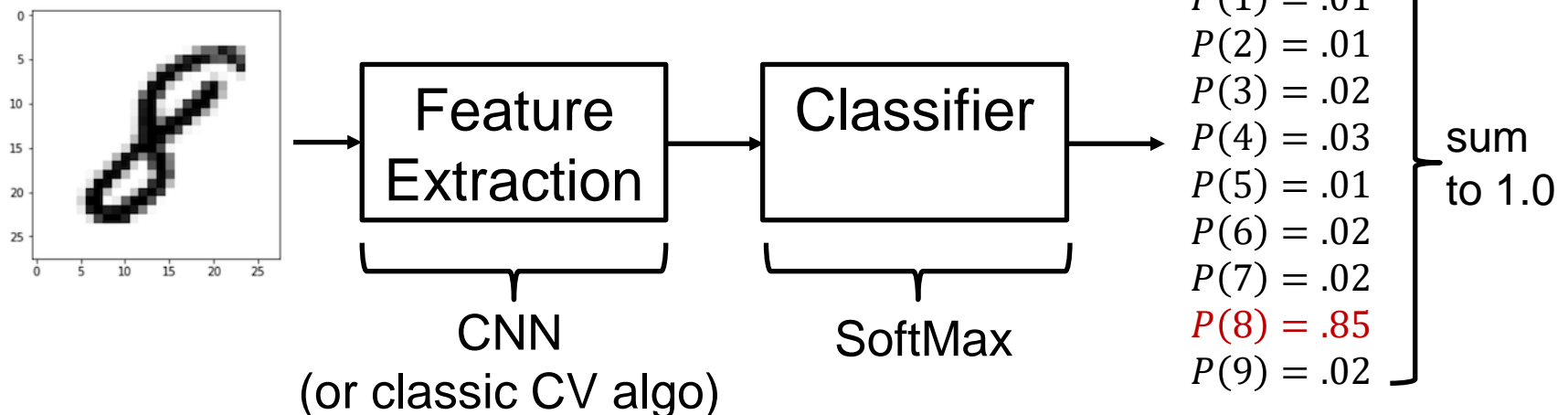
Cross-Entropy Loss Example

- 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 $y_i = 3$



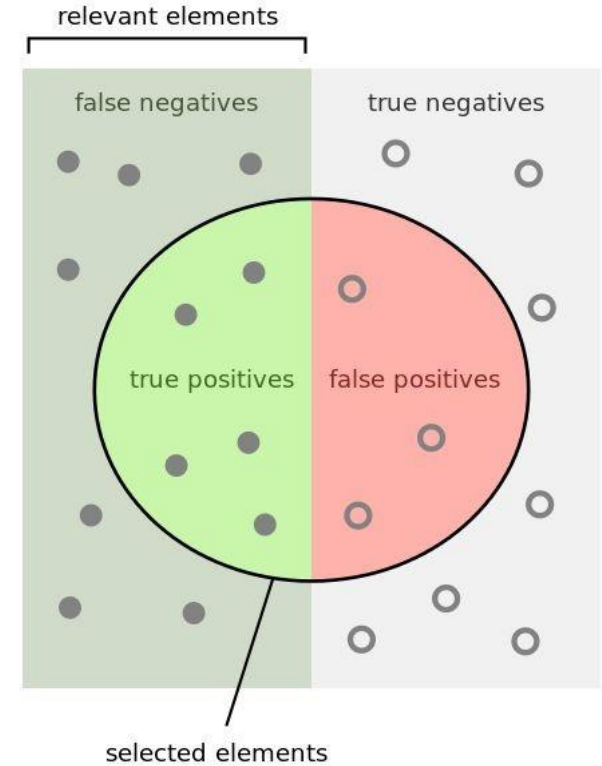
Example CV Task: Multi-Class Image Classification

- Input encoding:
 - A color image is a 3D tensor encoded as a vector of integers between [0, 255] (assuming 8-bit pixel depth) denoting pixel intensities for 3 RGB channels, e.g. a 128x128 pixel color image is a 128x128x3 tensor, encoded as a vector of size 128*128*3=49152
 - A greyscale image is a 128x128x1 tensor
- 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., 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



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Example Confusion Matrix 1

- Precision = $\frac{TP}{TP+FP} = \frac{100}{100+700} = .125$
 - When the classifier predicts positive, it is correct 12.5% of the time
- Recall (TPR) = $\frac{TP}{TP+FN} = \frac{100}{100+200} \approx .333$
 - The classifier catches 33.3% of all the positive cases.
- $F1 = 2 * \frac{\text{Recall} * \text{Precision}}{(\text{Recall} + \text{Precision})} = 2 * \frac{.333 * .125}{.333 + .125} = .182$
- False Positive Rate (FPR) = $\frac{FP}{FP+TN} = \frac{700}{700+9000} \approx .072$
 - The classifier misclassifies 7.2% of the negative cases as positive
- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN} = \frac{100+9000}{100+9000+700+200} = .91$
 - The classifier makes the correct prediction 91% percent of the time
- Positive correlation between TPR vs. FPR; Unpredictable correlation between precision vs. recall

		Predicted / Classified	
		- (Negative)	+ (Positive)
True	- (Negative)	True Negative (TN) 9,000	False Positive (FP) 700
	+ (Positive)	False Negative (FN) 200	True Positive (TP) 100

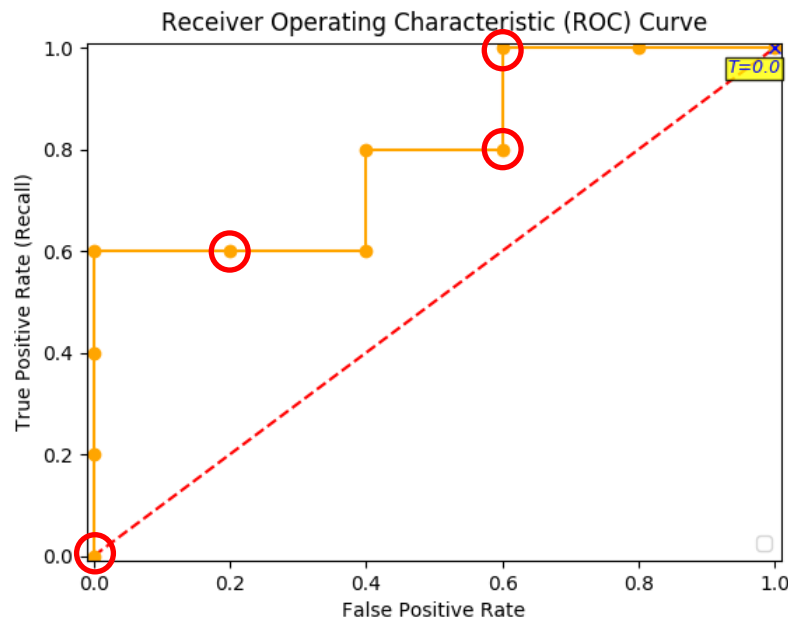
Example Confusion Matrix 2

- 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) = $\frac{TP}{TP+FN} = \frac{0}{0+300} = 0$
 - The classifier catches 0% of all the positive cases.
- False Positive Rate (FPR) = $\frac{FP}{FP+TN} = \frac{0}{0+9700} = 0$
 - The classifier misclassifies 0% of the negative cases as positive
- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN} = \frac{0+9700}{0+9700+0+300} = .97$
 - The classifier makes the correct prediction 97% percent of the time
- A medical test that never makes any positive diagnoses is very accurate for a rare disease (declare everyone to be healthy), but not very useful

		Predicted / Classified	
		- (Negative)	+ (Positive)
True	- (Negative)	True Negative (TN) 9,700	False Positive (FP) 0
	+ (Positive)	False Negative (FN) 300	True Positive (TP) 0

ROC Curve

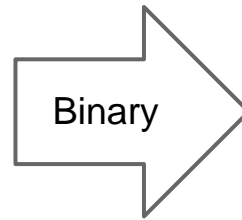
- Binary classification results typically depend on a decision threshold parameter. Moving the decision threshold will cause FPR and TPR to move in the same direction
 - e.g., a medical test that sets a high threshold for positive diagnosis will have both low FPR and low TPR, and vice versa
- Receiver Operating Characteristic (ROC) Curve plots FPR (x-axis) vs. TPR (y-axis):
 - Fig shows an example with 4 points (FPR, TPR) highlighted: (0,0), (.2, .6), (.6, .8), (.6, 1.0)
 - The ideal ROC curve: $FPR \equiv 0, TPR \equiv 1$, with $FP = FN = 0$



Confusion Matrix for Multi-Class Classification

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

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



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

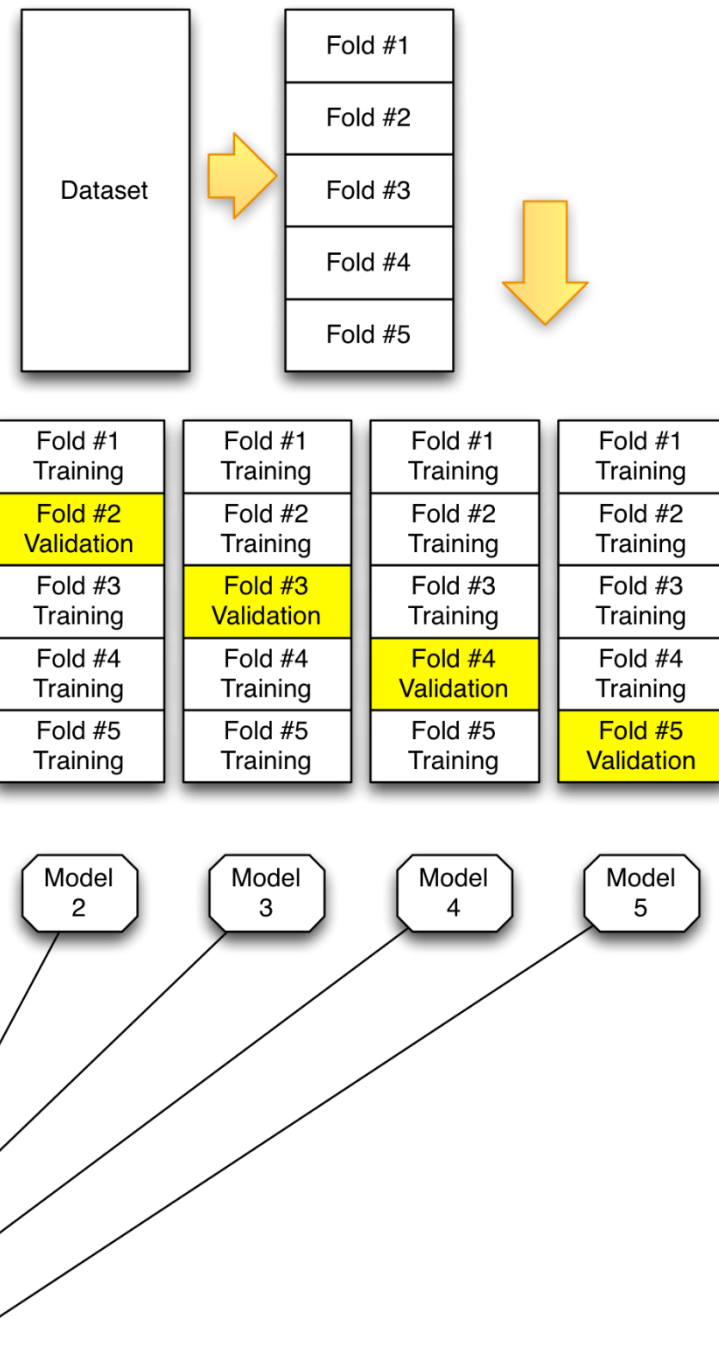
		Prediction		
		Class 1	Class 2	Class 3
Ground Truth	Class 1	x(1,1)	x(1,2)	x(1,3)
	Class 2	x(2,1)	x(2,2)	x(2,3)
	Class 3	x(3,1)	x(3,2)	x(3,3)

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

Training Neural Networks

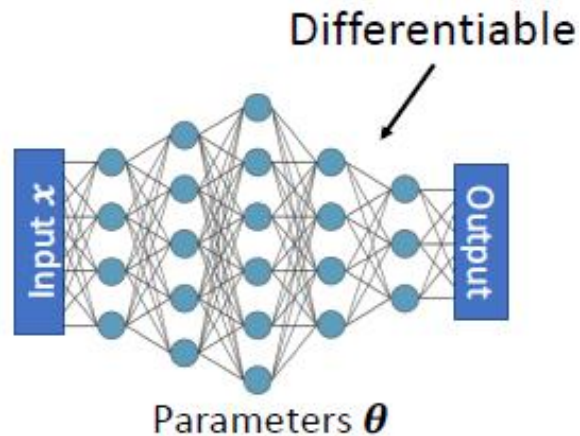
K-Fold Cross-Validation

- 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.

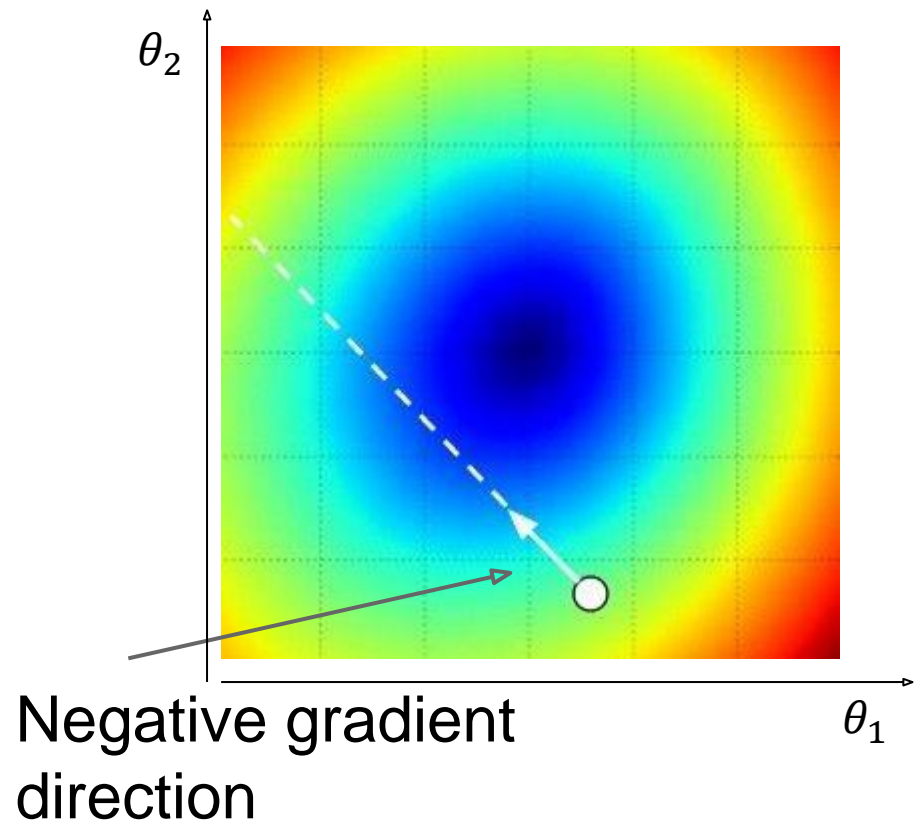
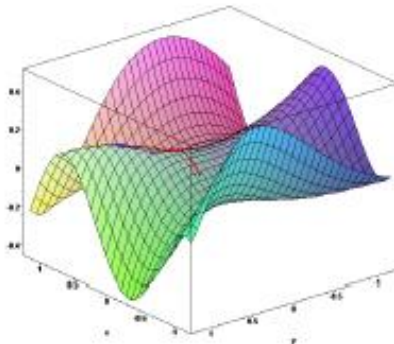


Gradient Descent

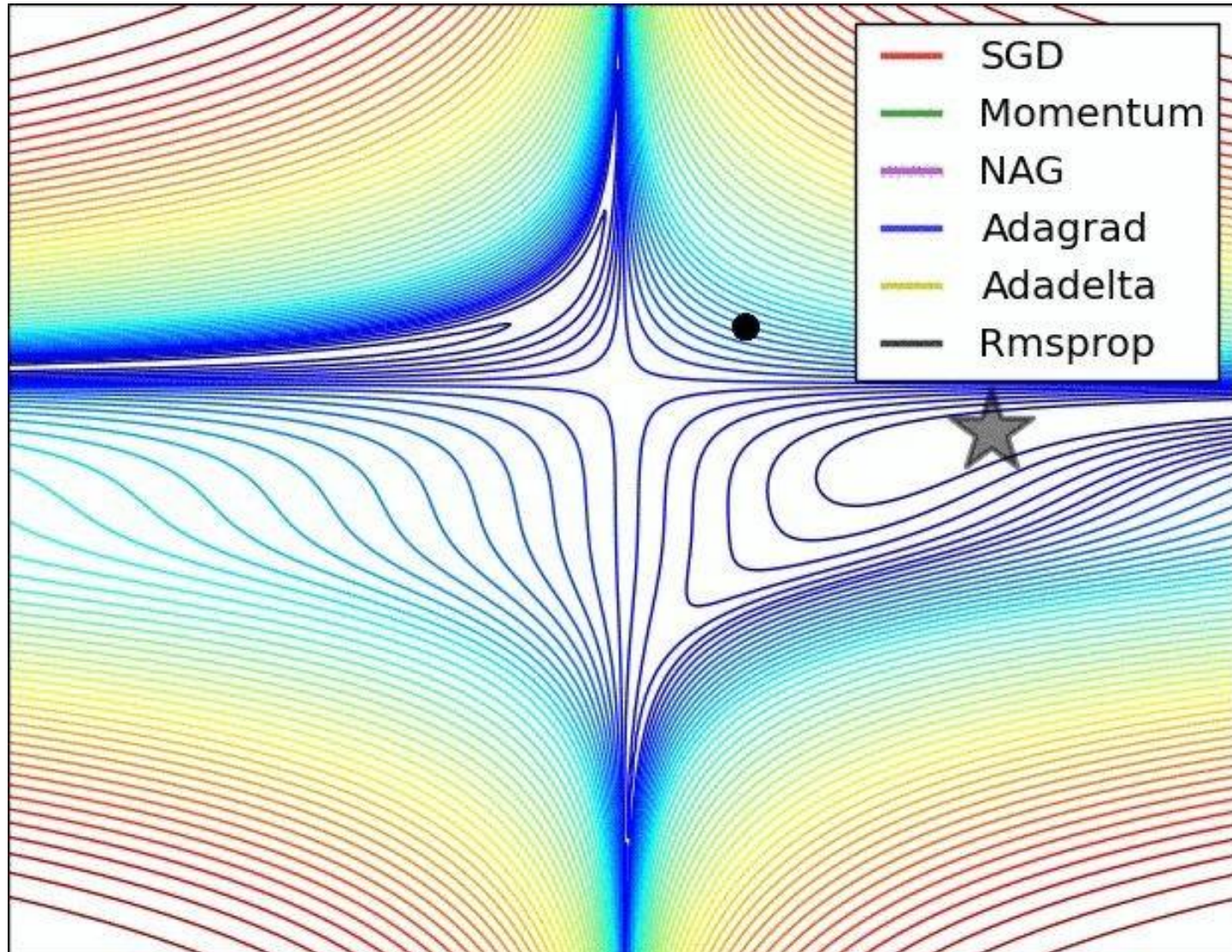
- Gradient descent $\theta \leftarrow \theta - \alpha \nabla_{\theta} \text{Loss}(x, y; \theta)$



Can use gradient descent method to find good θ



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: $\mathcal{B} = \{x_1, \dots, x_m\}$;

Parameters to be learned: γ, β

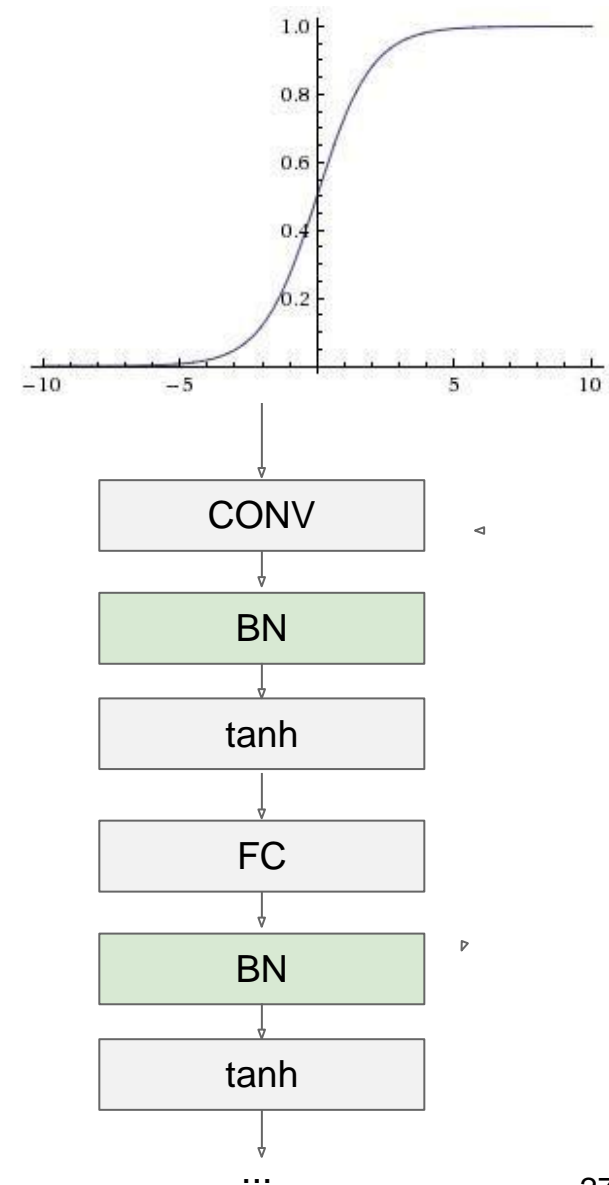
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

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

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

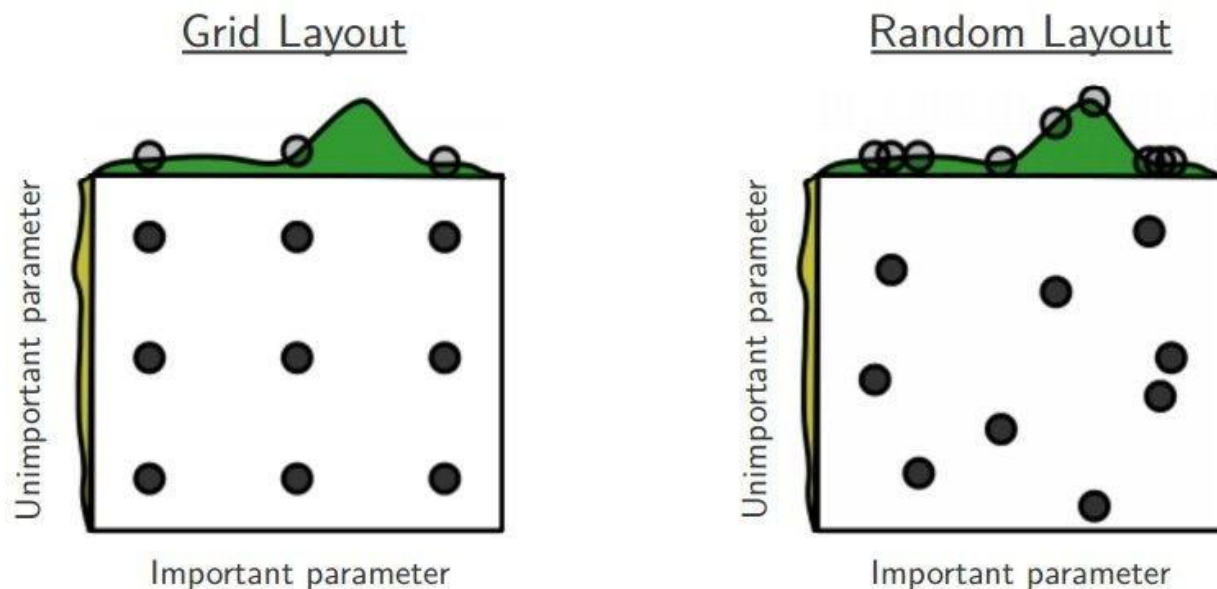
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

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

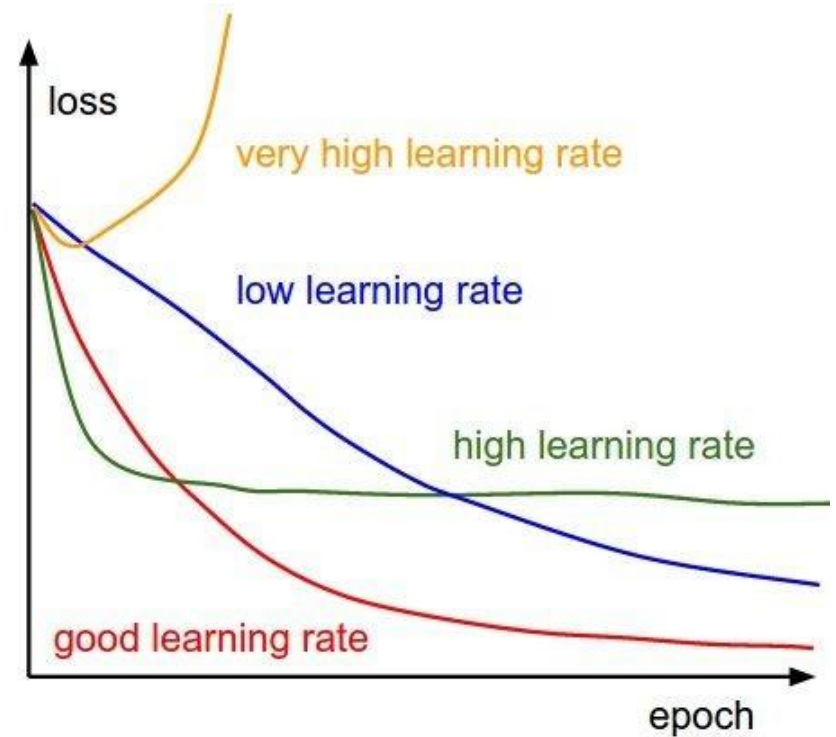
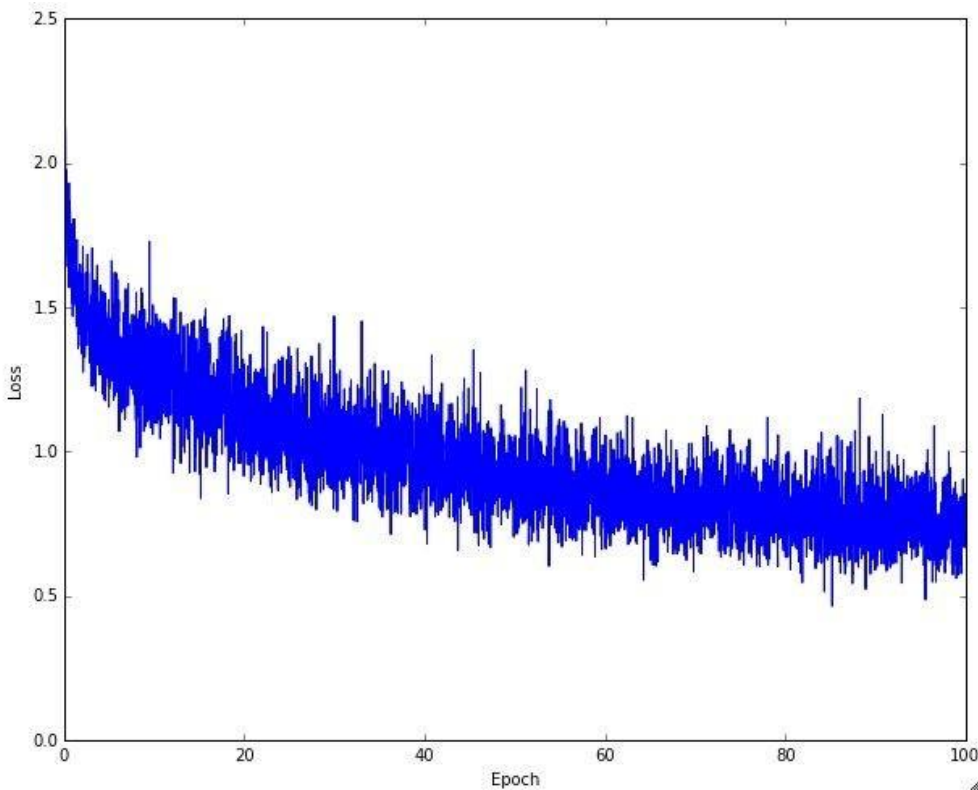


Hyperparameter Optimization

- 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?

