Georgia Tech College of Computing

CS4641 Spring 2025

Neural Networks

Bo Dai
School of CSE, Georgia Tech
bodai@cc.gatech.edu

ML Algorithm Pipeline

General ML Algorithm Pipeline

1. Build probabilistic models
2. Derive loss function (by MLE or MAP….)
3. Select optimizer

Regression Algorithms

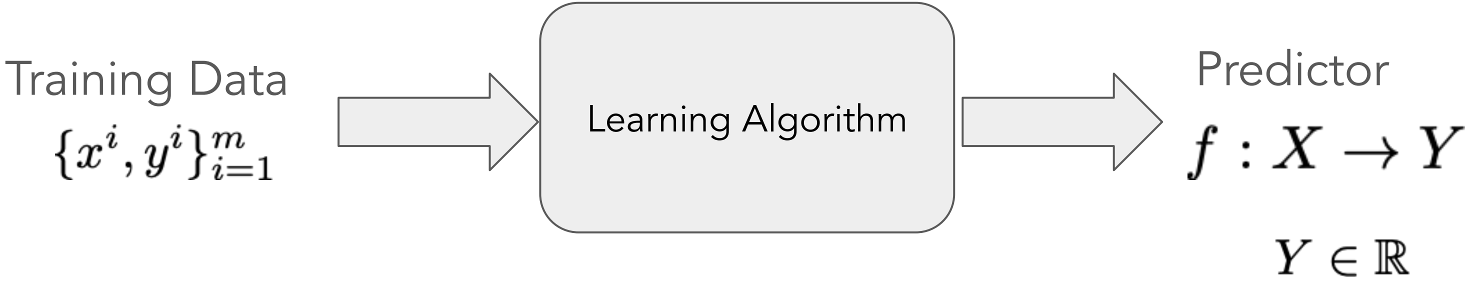
Training Data $\{x^i, y^i\}_{i=1}^m$ → Learning Algorithm → Predictor $f : X \to Y$

$Y \in \mathbb{R}$

Linear Regression Pipeline

1. Build probabilistic models:
   Gaussian Distribution + Linear Model
2. Derive loss function: MLE and MAP
3. Select optimizer
   Necessary Condition vs. (Stochastic) GD

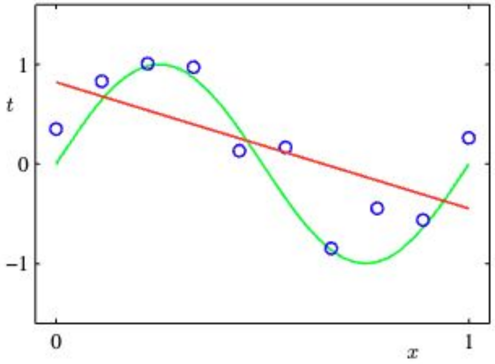
Regression Algorithms



Linear Regression Pipeline

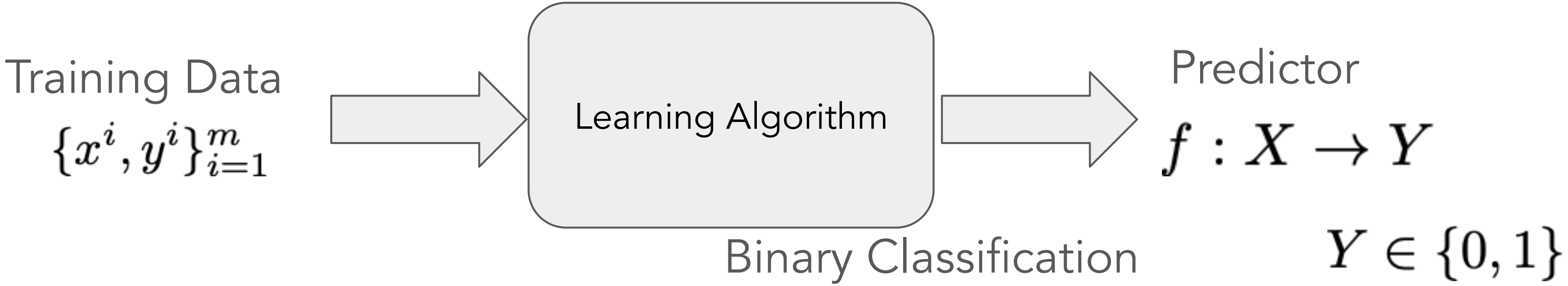
1. Build probabilistic models:
   Gaussian Distribution + Linear Model $\quad y = \theta^\top x + b$
2. Derive loss function: MLE and MAP
3. Select optimizer
   Necessary Condition vs. (Stochastic) GD

Linear Predictor



d=1

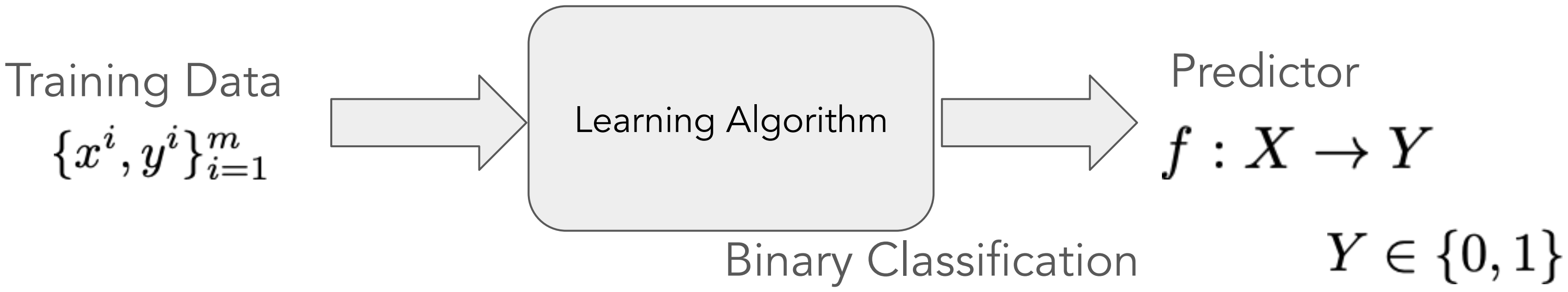
Binary Classification Algorithms



Binary Logistic Regression Pipeline

1. Build probabilistic models:
   Bernoulli Distribution + Linear Model
2. Derive loss function: MLE and MAP
3. Select optimizer: (Stochastic) Gradient Descent

Binary Classification Algorithms



Binary Logistic Regression Pipeline

1. Build probabilistic models:
   Bernoulli Distribution + Linear Model
2. Derive loss function: MLE and MAP
3. Select optimizer: (Stochastic) Gradient Descent

Logistic Regression is a Linear Classifier

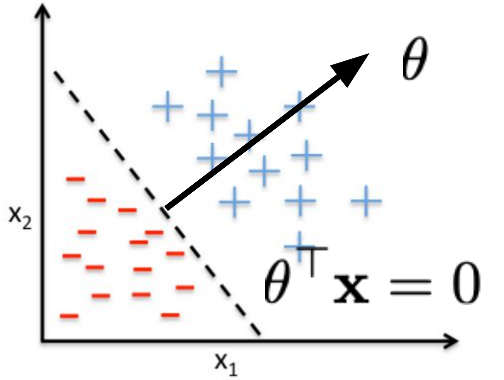
- Decision boundaries for Logistic Regression?
  - At the decision boundary, label 1/0 are equiprobable.

$$P(y=1|\mathbf{x},\theta) = \frac{1}{1+e^{-\theta^\top \mathbf{x}}}, \qquad P(y=0|\mathbf{x},\theta) = \frac{1}{1+e^{\theta^\top \mathbf{x}}}$$

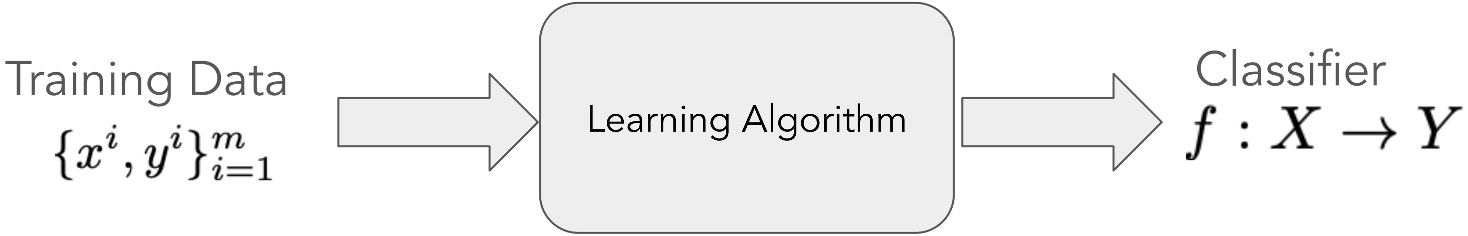
to be equal: $e^{-\theta^\top \mathbf{x}} = e^{\theta^\top \mathbf{x}}$, whose only solution is $\theta^\top \mathbf{x} = 0$.

✓ ⇒ Decision boundary is linear.

✓ ⇒ Logistic regression is a probabilistic linear classifier.



Multiclass Logistic Regression Algorithms

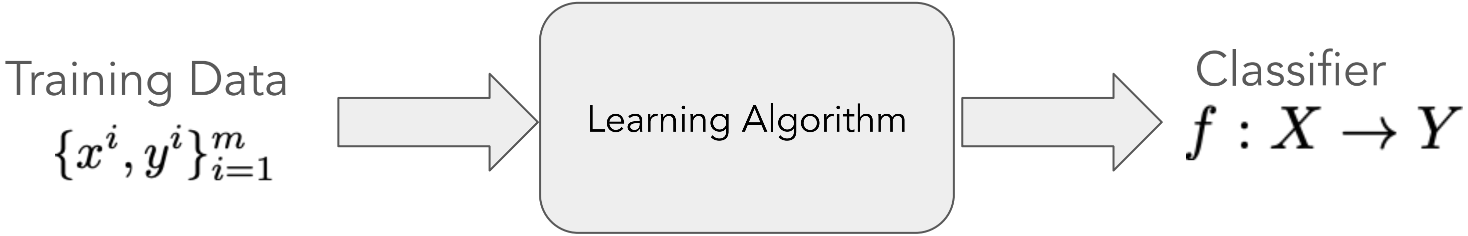


Multiclass Classification $Y \in \{0, 1, \ldots, k\}$

Multiclass Logistic Regression Pipeline

1. Build probabilistic models:
   Categorical Distribution + Linear Model
2. Derive loss function: MLE and MAP
3. Select optimizer: (Stochastic) Gradient Descent

Multiclass Logistic Regression Algorithms



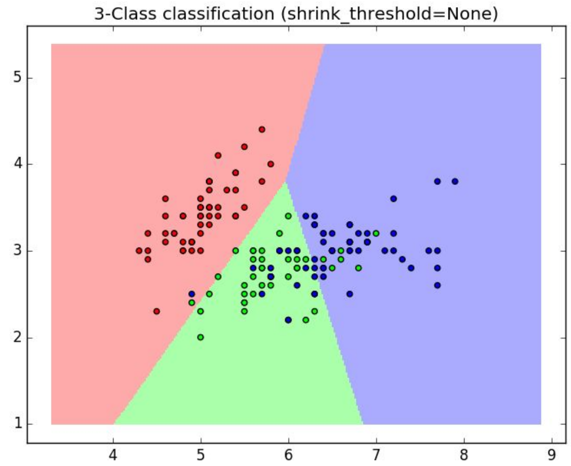
Multiclass Classification $Y \in \{0, 1, \ldots, k\}$

Multiclass Logistic Regression Pipeline

1. Build probabilistic models:
   Categorical Distribution + Linear Model
2. Derive loss function: MLE and MAP
3. Select optimizer: (Stochastic) Gradient Descent

Multiclass Logistic Regression is a Linear Classifier

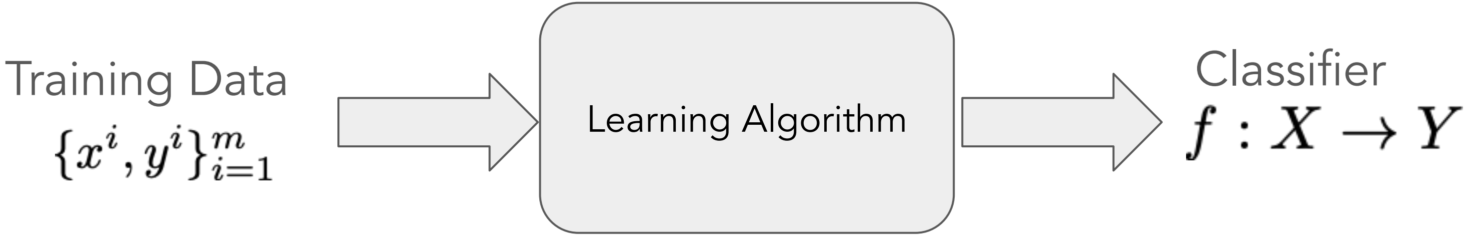
- Decision boundaries for Multiclass Logistic Regression?



✓ ⇒ Decision boundary is linear.

✓ ⇒ Multiclass Logistic regression is a probabilistic linear classifier.

Naive Bayes Classification



Multiclass Classification $Y \in \{0, 1, \dots, k\}$

Gaussian Naive Bayes Pipeline

1. Build probabilistic models
   Multinomial + Gaussian Likelihood => Quadratic/Linear
2. Derive loss function (by MLE or MAP)
3. Select optimizer
   Closed-form from Necessary Condition

Naive Bayes Classification

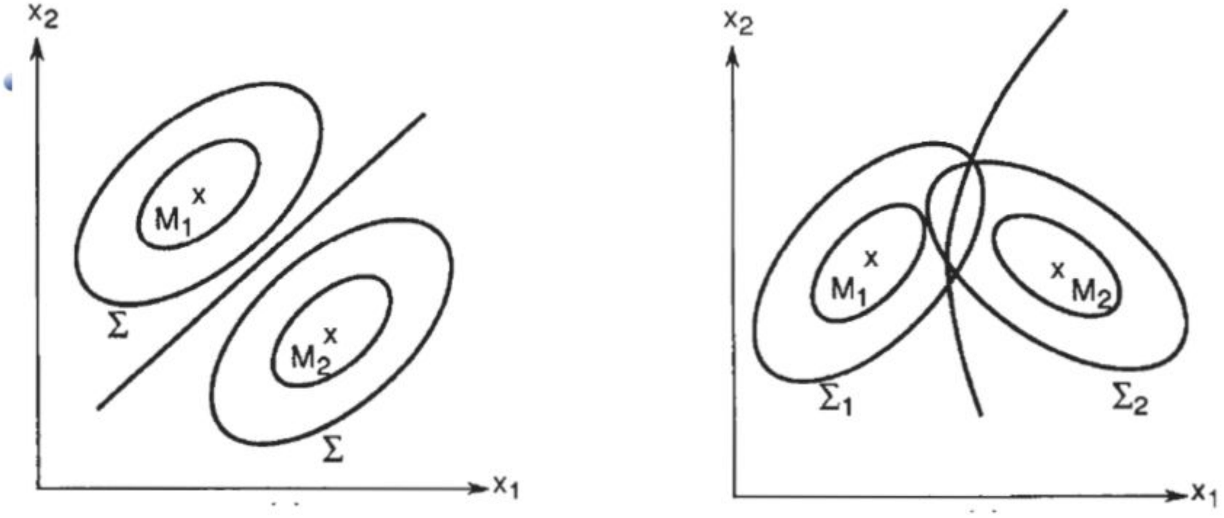
Training Data $\{x^i, y^i\}_{i=1}^m$ → Learning Algorithm → Classifier $f : X \rightarrow Y$

Multiclass Classification $Y \in \{0, 1, \ldots, k\}$

Gaussian Naive Bayes Pipeline

1. Build probabilistic models
   Multinomial + Gaussian Likelihood =>
   Quadratic/Linear
2. Derive loss function (by MLE or MAP)
3. Select optimizer
   Closed-form from Necessary Condition

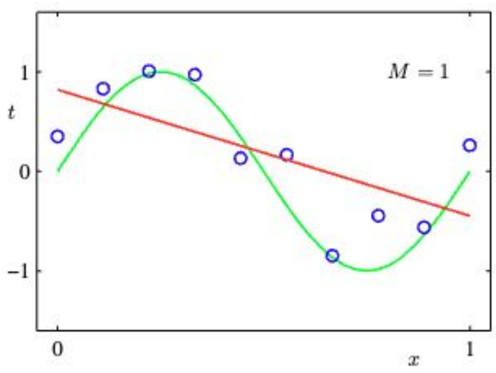
- Depending on the Gaussian distributions, the decision boundary can be very different



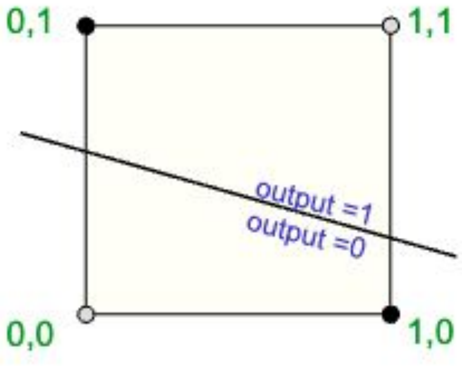
- Decision boundary: $h(\mathbf{x}) = -\ln \frac{q_i(x)}{q_j(x)} = 0$

Limitations of Linear Predictor/Classifier

- Linear predictor/classifiers (e.g., logistic regression) classify inputs based on linear combinations of features x_i
- Many decisions involve non-linear functions of the input

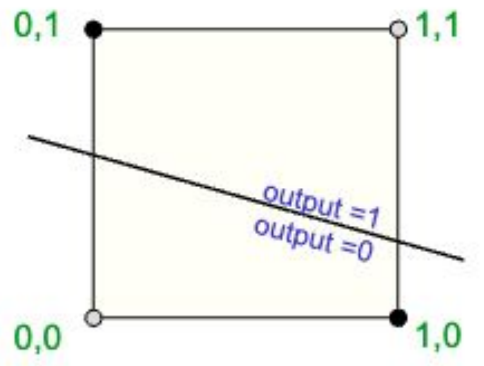


d=1



Limitations of Linear Classifier

- Linear classifiers (e.g., logistic regression) classify inputs based on linear combinations of features x_i
- Many decisions involve non-linear functions of the input



- The positive and negative cases cannot be separated by a plane

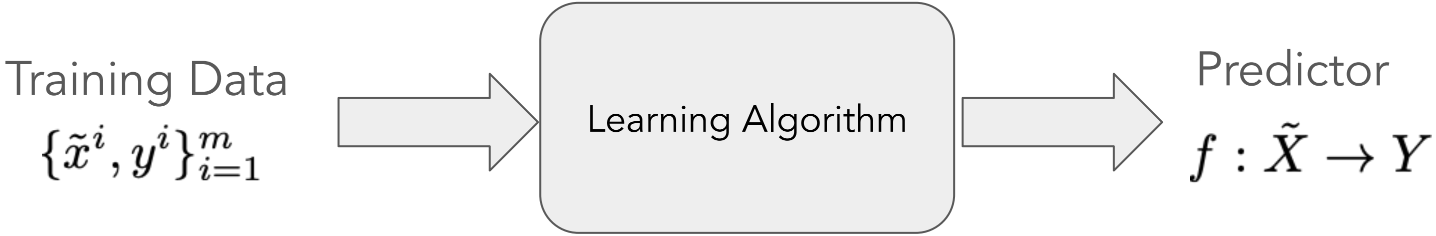
Nonlinear Parametrization: Polynomial Regression

Training Data $\{\tilde{x}^i, y^i\}_{i=1}^m$ → Learning Algorithm → Predictor $f : \tilde{X} \to Y$

$$x = [x_1, x_2, \ldots, x_d]$$

$$\begin{aligned} y = \theta_0 &+ \theta_1^1 x_1 + \theta_1^2 (x_1)^2 + \theta_1^3 (x_1)^3 + \ldots + \theta_1^d (x_1)^d \\ &+ \theta_2^1 x_2 + \theta_2^2 (x_2)^2 + \theta_2^3 (x_2)^3 + \ldots + \theta_2^d (x_2)^d \\ &+ \ldots \\ &+ \theta_n^1 x_n + \theta_n^2 (x_n)^2 + \theta_n^3 (x_n)^3 + \ldots + \theta_n^d (x_n)^d \end{aligned}$$

Nonlinear Parametrization: Polynomial Regression



$$x = [x_1, x_2, \dots, x_d]$$

$$\begin{aligned} y = \theta_0 &+ \theta_1^1 x_1 + \theta_1^2 (x_1)^2 + \theta_1^3 (x_1)^3 + \dots + \theta_1^d (x_1)^d \\ &+ \theta_2^1 x_2 + \theta_2^2 (x_2)^2 + \theta_2^3 (x_2)^3 + \dots + \theta_2^d (x_2)^d \\ &+ \dots \\ &+ \theta_n^1 x_n + \theta_n^2 (x_n)^2 + \theta_n^3 (x_n)^3 + \dots + \theta_n^d (x_n)^d \end{aligned}$$

$$x_1 \cdot x_2, \dots, x_i \cdot x_j, \dots$$

Nonlinear Parametrization: Polynomial Regression

Training Data

$\{\tilde{x}^i, y^i\}_{i=1}^m$

Learning Algorithm

Predictor

$f : \tilde{X} \rightarrow Y$

$$x = [x_1, x_2, \ldots, x_d]$$

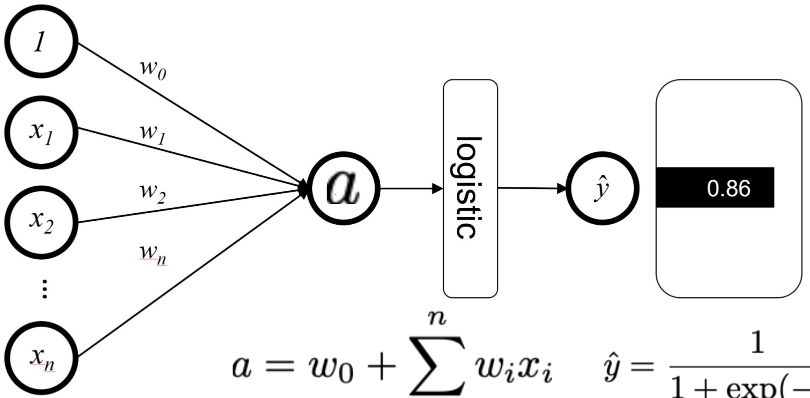
$$\begin{aligned} y = \theta_0 &+ \theta_1^1 x_1 + \theta_1^2 (x_1)^2 + \theta_1^3 (x_1)^3 + \ldots + \theta_1^d (x_1)^d \\ &+ \theta_2^1 x_2 + \theta_2^2 (x_2)^2 + \theta_2^3 (x_2)^3 + \ldots + \theta_2^d (x_2)^d \\ &+ \ldots \\ &+ \theta_n^1 x_n + \theta_n^2 (x_n)^2 + \theta_n^3 (x_n)^3 + \ldots + \theta_n^d (x_n)^d \end{aligned}$$

$x_1 \cdot x_2, \ldots, x_i \cdot x_j, \ldots$

Combinatorial Parametrization

Better Nonlinear Parametrization: Neural Network

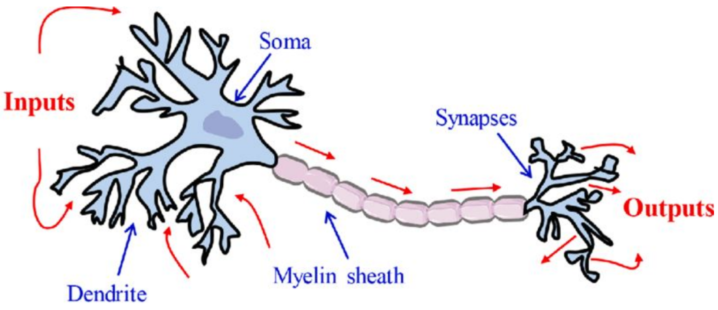
Logistic Regression Revisit

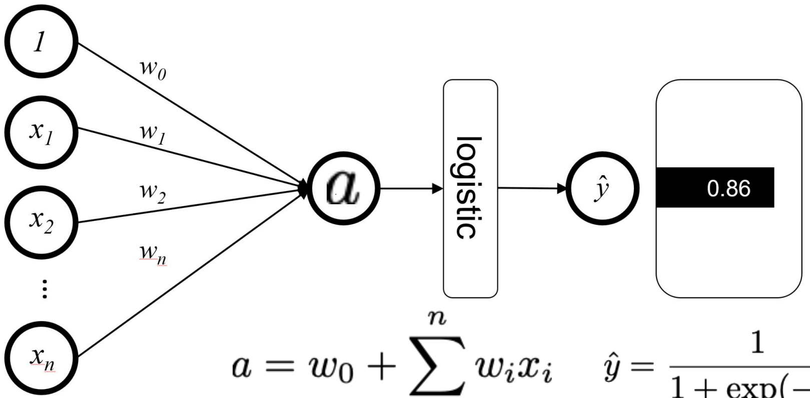


$$a = w_0 + \sum_{i=1}^{n} w_i x_i \qquad \hat{y} = \frac{1}{1 + \exp(-a)}$$

Better Nonlinear Parametrization: Neural Network

Neuron $\Longleftrightarrow$ Logistic Regression

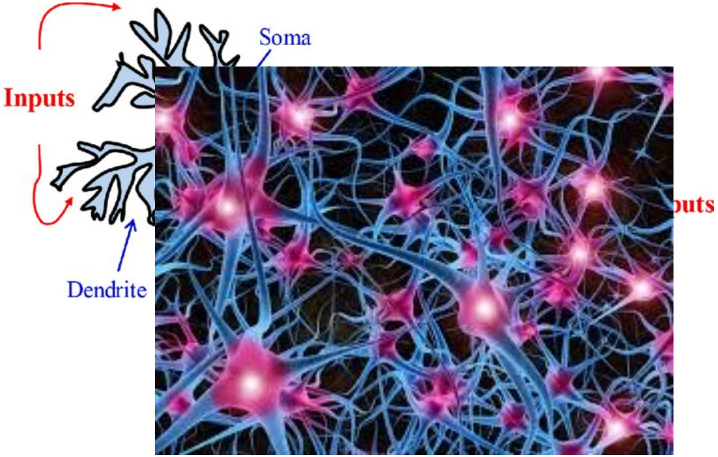


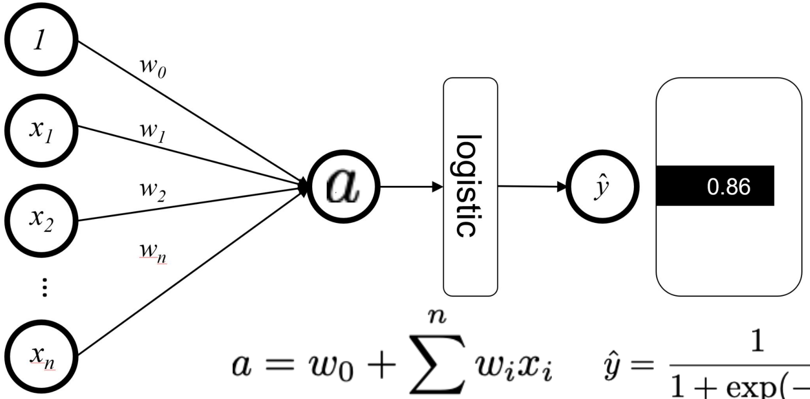


$$a = w_0 + \sum_{i=1}^{n} w_i x_i \qquad \hat{y} = \frac{1}{1 + \exp(-a)}$$

Better Nonlinear Parametrization: Neural Network

Neural Network $\Longleftrightarrow$ Composition of Neurons

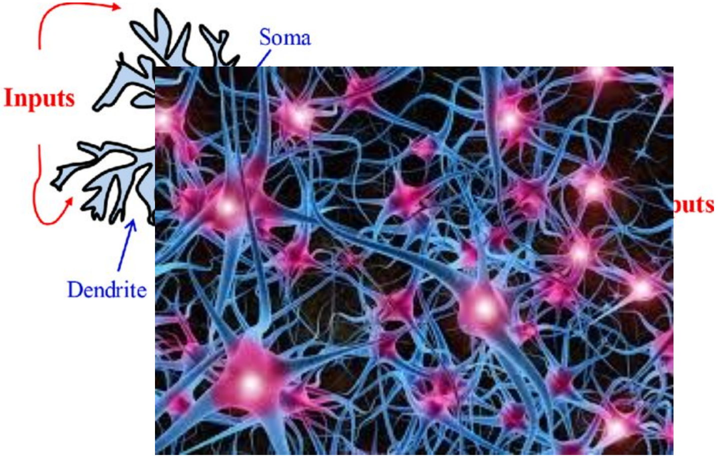


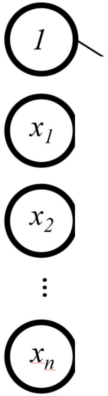


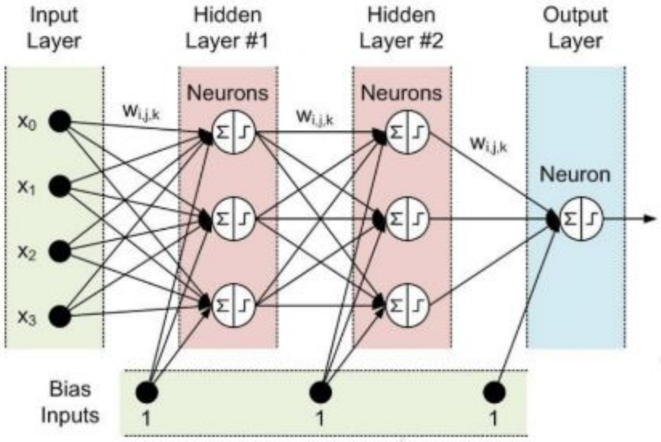
$$a = w_0 + \sum_{i=1}^{n} w_i x_i \qquad \hat{y} = \frac{1}{1 + \exp(-a)}$$

Better Nonlinear Parametrization: Neural Network

Neural Network $\Longleftrightarrow$ Composition of Neurons

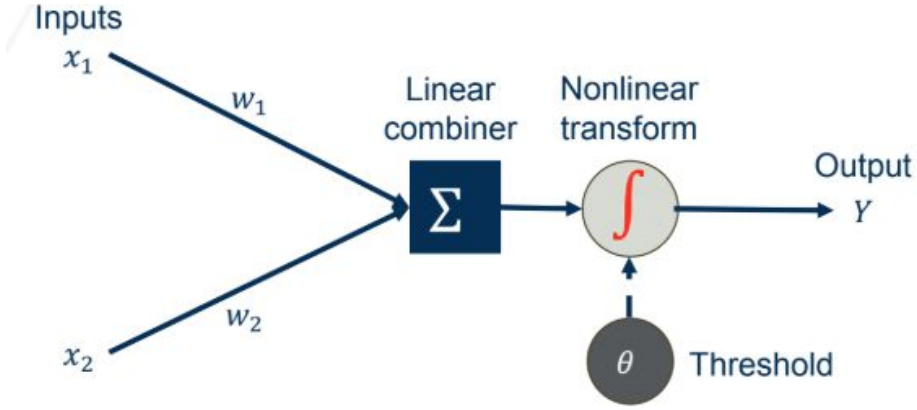






Alternative Neurons

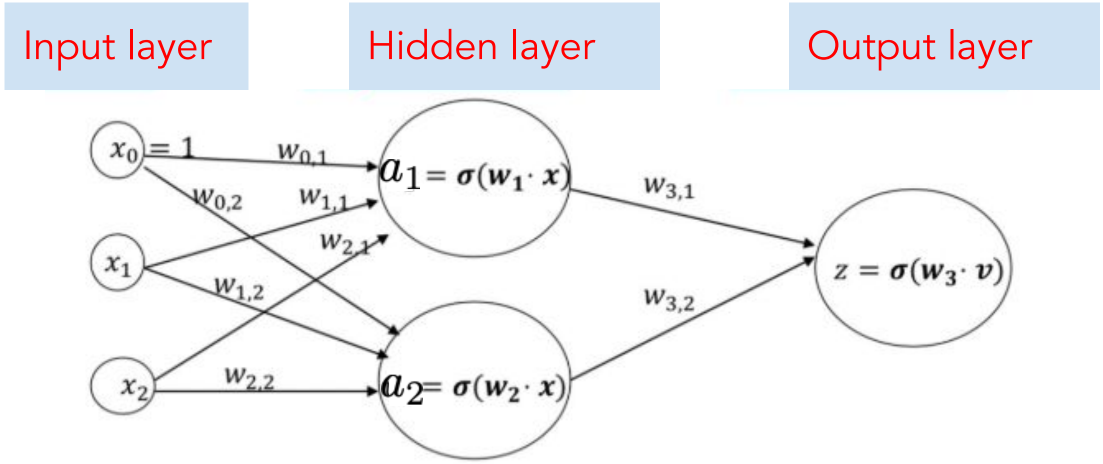
- Use different nonlinear transformations $f(u)$
- Before that, perform weighted combination of inputs $u = w^T x$



Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)[13]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU)[5]		$\frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = x\Phi(x)$ where erf is the gaussian error function.
Softplus[14]		$\ln(1 + e^x)$
Exponential linear unit (ELU)[15]		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α

Multi-Layer Perception: Composition of Neurons

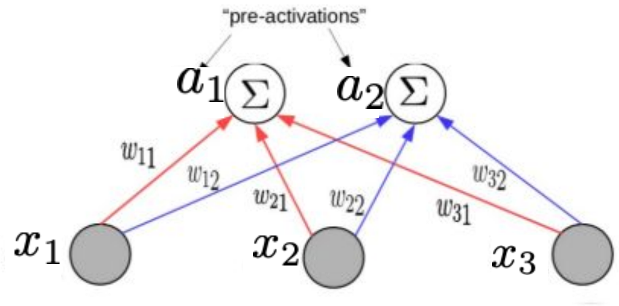
- The classifier/regressor is a multilayer network of units
- Each unit takes some inputs and produces one output. Output of one unit can be the input of another.
  - Advantage: Can produce highly non-linear decision boundaries!
  - Sigmoid is differentiable, so can use gradient descent



Forward Pass in MLP

- Each input x_n transformed into several "pre-activations" using linear models

$$a_k = w_k^\top x = \sum_{i=1}^{n} w_{ki} x_i$$



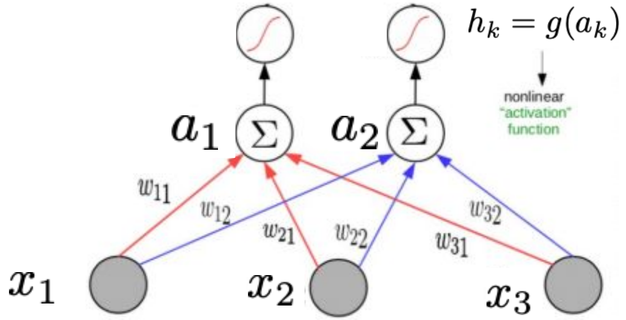
Forward Pass in MLP

- Each input x_n transformed into several "pre-activations" using linear models

$$a_k = w_k^\top x = \sum_{i=1}^{n} w_{ki} x_i$$

- Nonlinear activation applied on each pre-activation

$$h_k = g(a_k)$$



Forward Pass in MLP

- Each input x_n transformed into several "pre-activations" using linear models

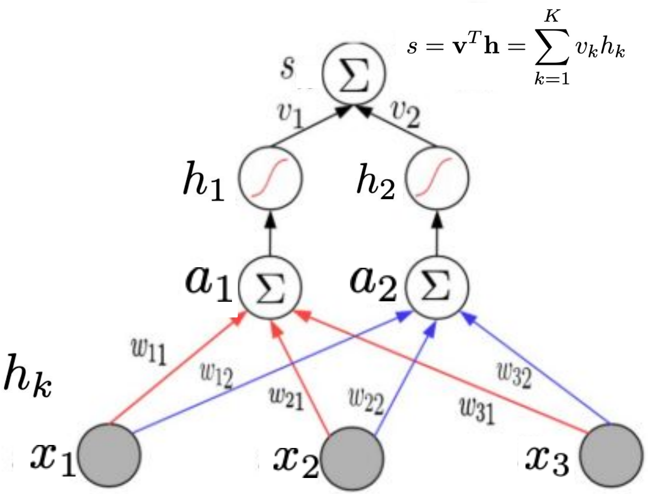
$$a_k = w_k^\top x = \sum_{i=1}^{n} w_{ki} x_i$$

- Nonlinear activation applied on each pre-activation

$$h_k = g(a_k)$$

- A linear model applied on the new "features" h_k

$$s = \mathbf{v}^T \mathbf{h} = \sum_{k=1}^{K} v_k h_k$$



Forward Pass in MLP

- Each input x_n transformed into several "pre-activations" using linear models

$$a_k = w_k^\top x = \sum_{i=1}^{n} w_{ki} x_i$$

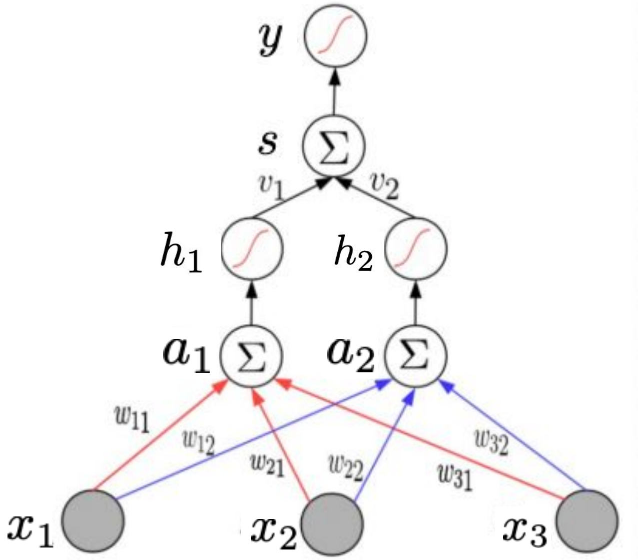
- Nonlinear activation applied on each pre-activation

$$h_k = g(a_k)$$

- A linear model applied on the new "features" h_k

$$s = \mathbf{v}^T \mathbf{h} = \sum_{k=1}^{K} v_k h_k$$

- Finally, the output is produced as $y = o(s)$



Forward Pass in MLP

- Each input x_n transformed into several "pre-activations" using linear models

$$a_k = w_k^\top x = \sum_{i=1}^{n} w_{ki} x_i$$

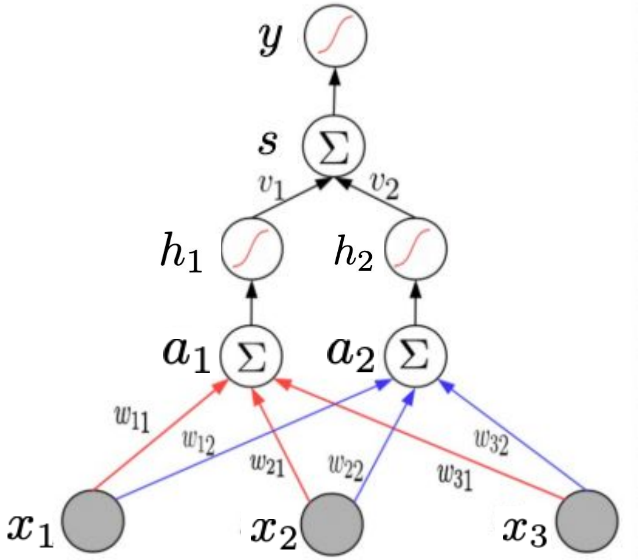
- Nonlinear activation applied on each pre-activation

$$h_k = g(a_k)$$

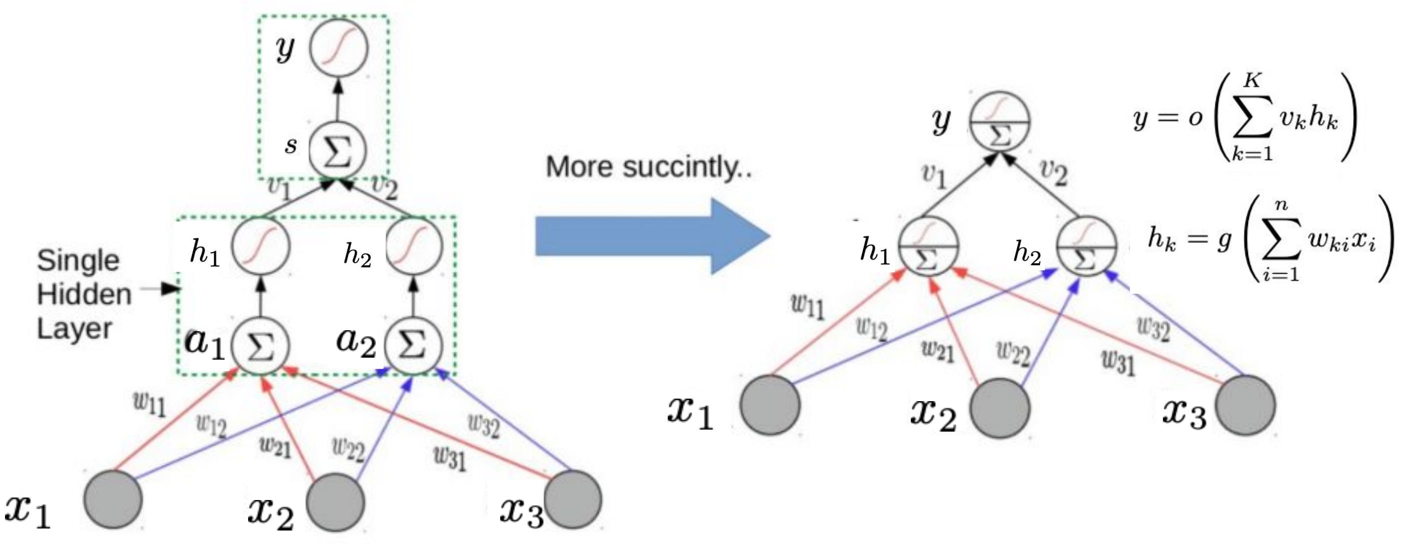
- A linear model applied on the new "features" h_k

$$s = \mathbf{v}^T \mathbf{h} = \sum_{k=1}^{K} v_k h_k$$

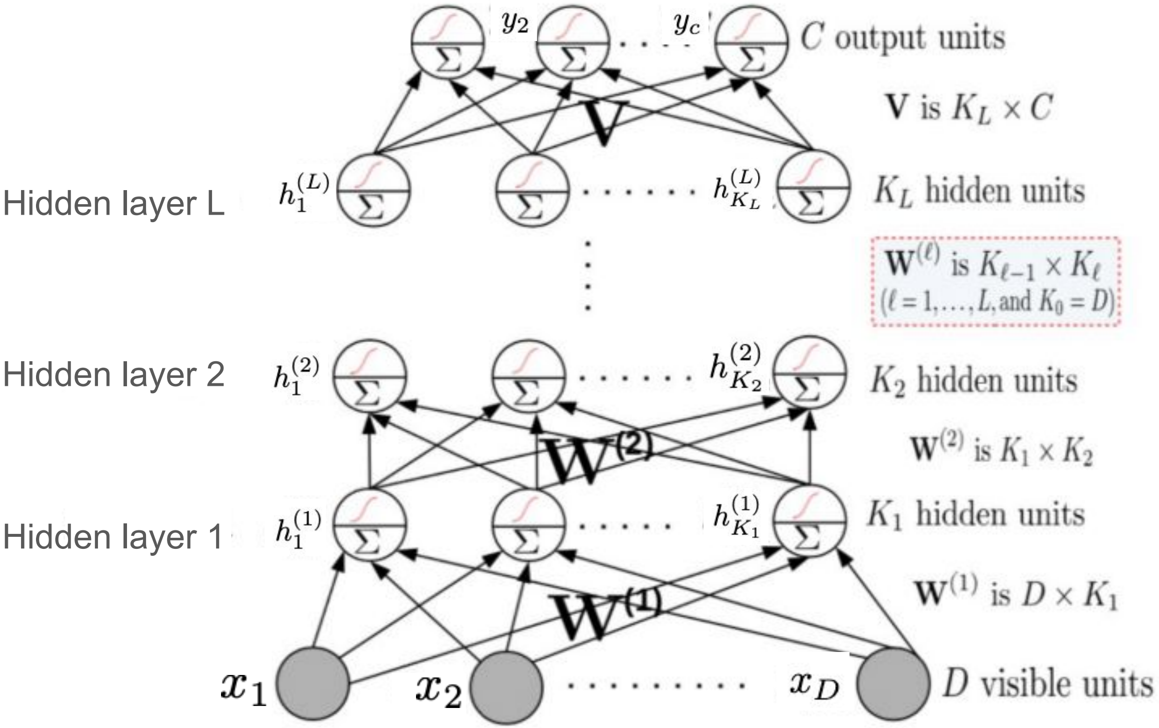
- Finally, the output is produced as $y = o(s)$
- Unknowns of the model $\mathbf{w_1}, \ldots, \mathbf{w_K}$ and $\mathbf{v}$ learned by minimizing a loss $\mathcal{L}(\mathbf{W}, \mathbf{v}) = \sum_{n=1}^{N} \ell(y_n, o(s_n))$, e.g, squared, logistic, softmax, etc (depending on the output)



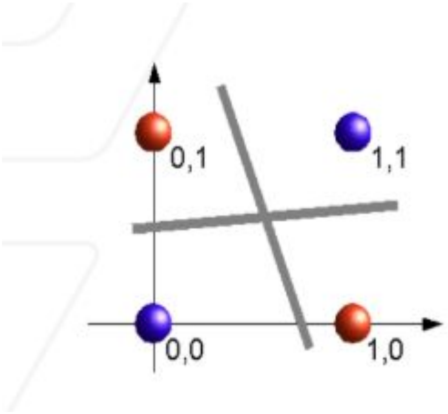
Compact Illustration



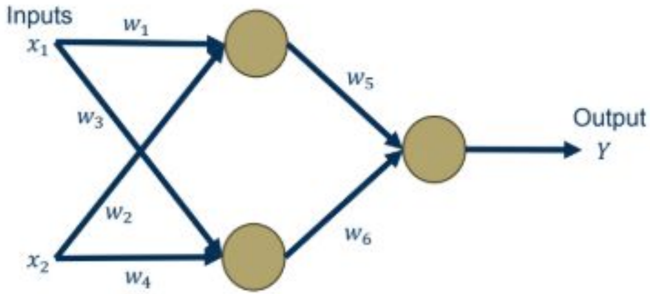
Multi-Layer, Multi-Hidden Units and Multi-Outputs Extension



Multi-layer Perception for XOR problem



x_1	x_2	y (color)
0	0	1
0	1	0
1	0	0
1	1	1



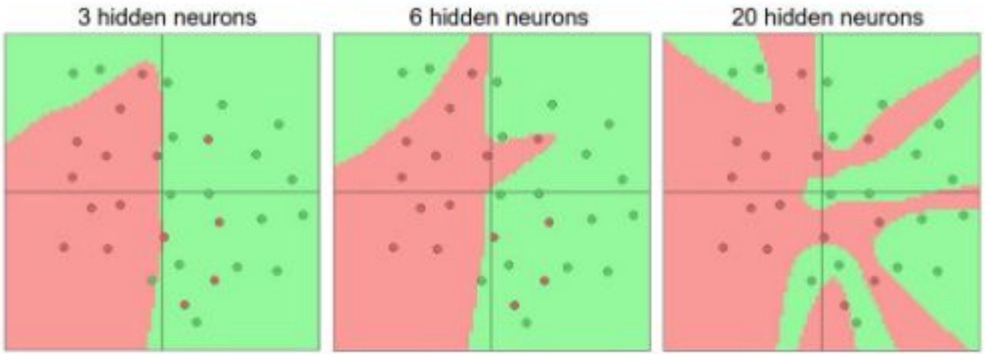
A possible set of weight:

$$(w_1, w_2, w_3, w_4, w_5, w_6) = (0.6, -0.6, -0.7, 0.8, 1, 1)$$

Representational Power

- Neural network with at least one hidden layer is a universal approximator (can represent any function).

  Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko, paper

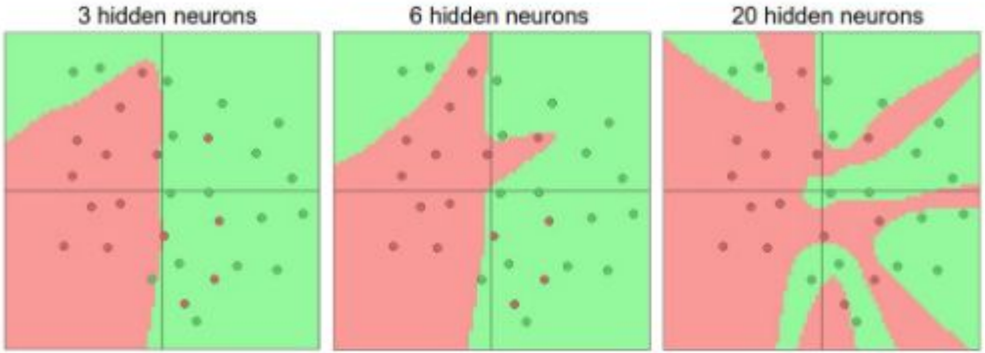


- The capacity of the network increases with more hidden units and more hidden layers

Representational Power

- Neural network with at least one hidden layer is a universal approximator (can represent any function).

  Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko, paper

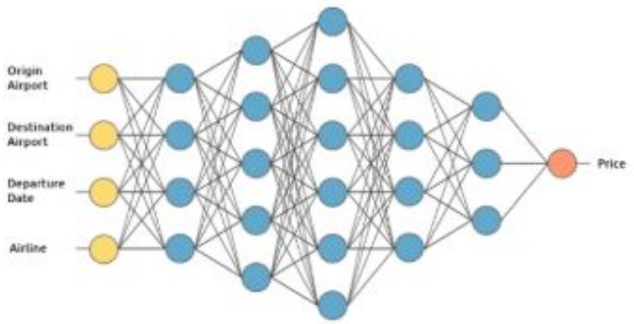


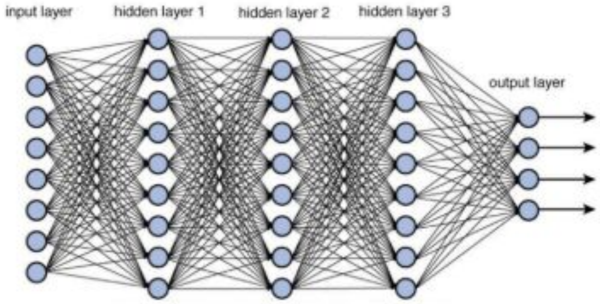
- The capacity of the network increases with more hidden units and more hidden layers (Depth vs. Width)

Neural Network Architecture

Multi-Layer Perceptron (MLP, 60's -):

FF with Fully connected (*dense*) layers: each unit of layer *i* is fully connected with the units of the previous layer

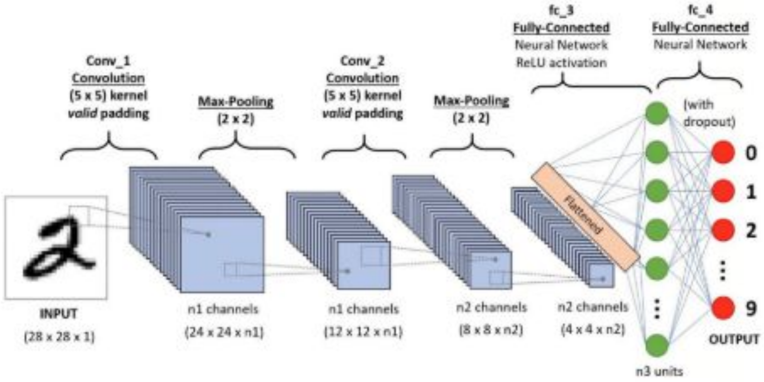




Deep network (>3 hidden layers?)

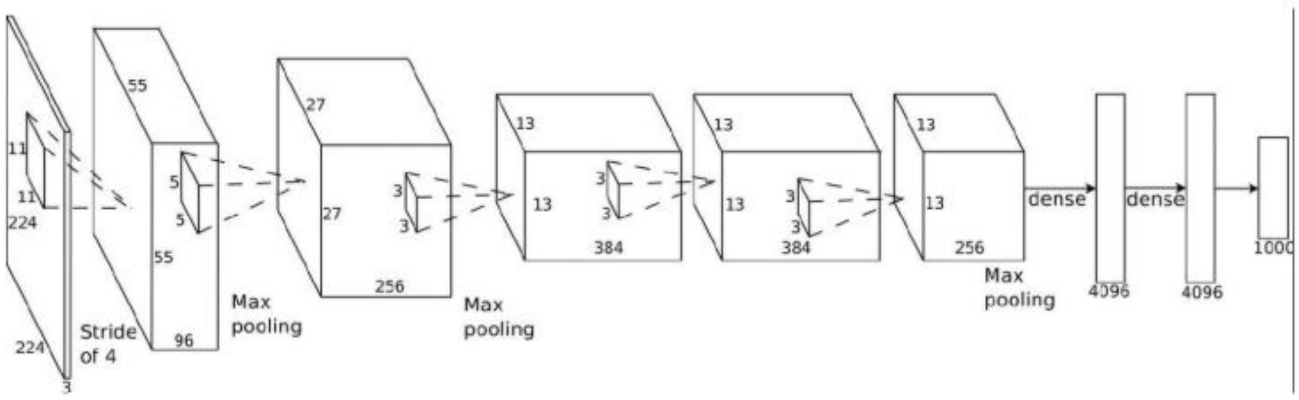
Convolutional Neural Network (CNN):

Not (all) fully connected layers

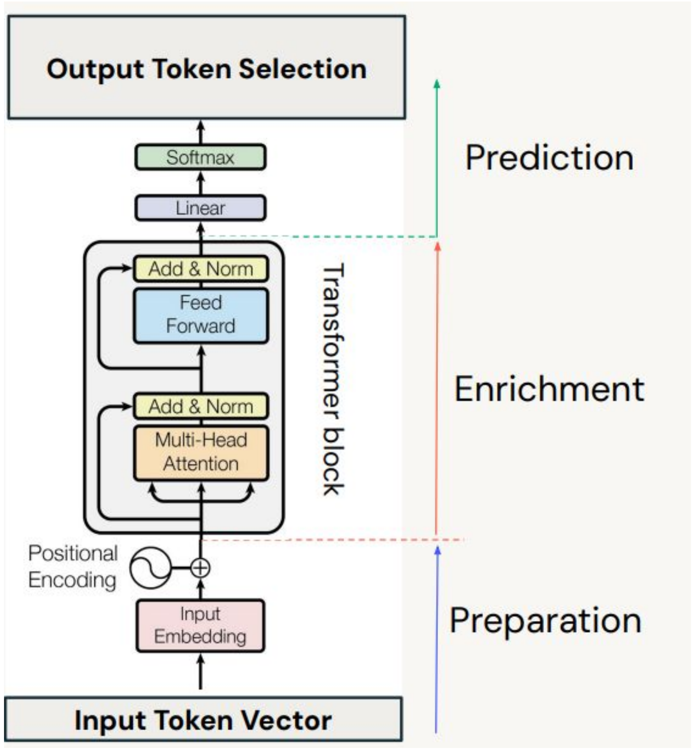


AlexNet

- 8 layer convolution neural network [Krizhevsky et al. 2012] achieved the state-of-the-art result (beating the second place by 10%).
  - Fist 5 layers: convolution + max pooling
  - Next 2 layers: fully connected nonlinear neurons
  - Last layer: multiclass logistic regression



Generative Pretrained Transformer (GPT)



Q&A