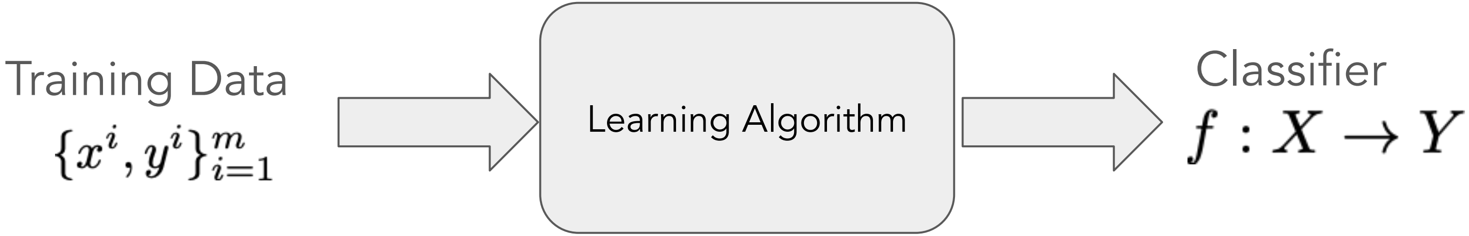
Georgia Tech College of Computing

CX4240 Spring 2026

Discriminative vs. Generative Classifier

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

Naive Bayes Classifier



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

Gaussian Naive Bayes Pipeline

1. Build probabilistic models:
   Gaussian Likelihood + Categorical Prior
2. Derive loss function: MLE or MAP
3. Select optimizer: Necessary Condition

Deeper Connection

Binary
Logistic Regression

$$p(y=1|x,\theta) = \frac{1}{1+\exp(-\theta^\top x)}$$

$$p(y=0|x,\theta) = 1 - \frac{1}{1+\exp(-\theta^\top x)} = \frac{\exp(-\theta^\top x)}{1+\exp(-\theta^\top x)}$$

Gaussian
Naive Bayes Classifier

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x,y)}{\sum_y P(x,y)}$$

$$= \frac{P(y)\mathcal{N}(x|\mu_y,\Sigma_y)}{\sum_y P(y)\mathcal{N}(x|\mu_y,\Sigma_y)}$$

Posterior of Gaussian Naive Bayes Classifier

$$P(y=1|x) = \frac{p(x,y=1)}{p(x,y=0)+p(x,y=1)} = \frac{\pi_1 \mathcal{N}(x|\mu_1,\Sigma)}{\pi_0 \mathcal{N}(x|\mu_0,\Sigma)+\pi_1 \mathcal{N}(x|\mu_1,\Sigma)}$$

$$= \left\{1 + \frac{\pi_0}{\pi_1}\exp\left[-\frac{1}{2}(x-\mu_0)^\top\Sigma^{-1}(x-\mu_0) + \frac{1}{2}(x-\mu_1)^\top\Sigma^{-1}(x-\mu_1)\right]\right\}^{-1}$$

$$= \left\{1 + \exp\left[\log\frac{\pi_0}{\pi_1} + (\mu_0-\mu_1)^\top\Sigma^{-1}x + \frac{1}{2}(\mu_0^\top\Sigma^{-1}\mu_0 - \mu_1^\top\Sigma^{-1}\mu_1)\right]\right\}^{-1}$$

$$= \frac{1}{1+\exp(-\theta^\top x - b)}$$

Decision Boundary Gaussian Naive Bayes Classifier

$$p(x, y = 0) = p(x, y = 1)$$

Decision Boundary Gaussian Naive Bayes Classifier

$$p(x, y = 0) = p(x, y = 1)$$

$$\log \pi_1 - \frac{1}{2}(x - \mu_1)^\top \Sigma_1^{-1}(x - \mu_1) = \log \pi_0 - \frac{1}{2}(x - \mu_0)^\top \Sigma_0^{-1}(x - \mu_0)$$

Decision Boundary Gaussian Naive Bayes Classifier

$$p(x, y=0) = p(x, y=1)$$

$$\log \pi_1 - \frac{1}{2}(x-\mu_1)^\top \Sigma_1^{-1}(x-\mu_1) = \log \pi_0 - \frac{1}{2}(x-\mu_0)^\top \Sigma_0^{-1}(x-\mu_0)$$

$$x^\top(\Sigma_1^{-1} - \Sigma_0^{-1})x - 2\left(\mu_1^\top \Sigma_1^{-1} - \mu_0^\top \Sigma_0^{-1}\right)x + \left(\mu_0^\top \Sigma_0^{-1}\mu_0 - \mu_1^\top \Sigma_1^{-1}\mu_1\right) = C$$

$$\Rightarrow x^\top Q x - 2b^\top x + c = 0$$

Decision Boundary Gaussian Naive Bayes Classifier

$$p(x, y=0) = p(x, y=1)$$

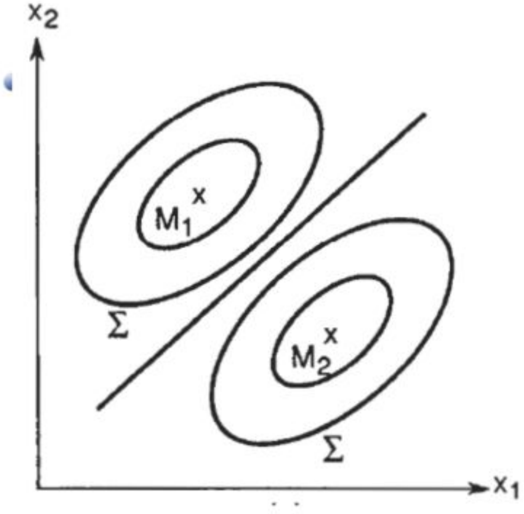
$$\log \pi_1 - \frac{1}{2}(x-\mu_1)^\top \Sigma_1^{-1}(x-\mu_1) = \log \pi_0 - \frac{1}{2}(x-\mu_0)^\top \Sigma_0^{-1}(x-\mu_0)$$

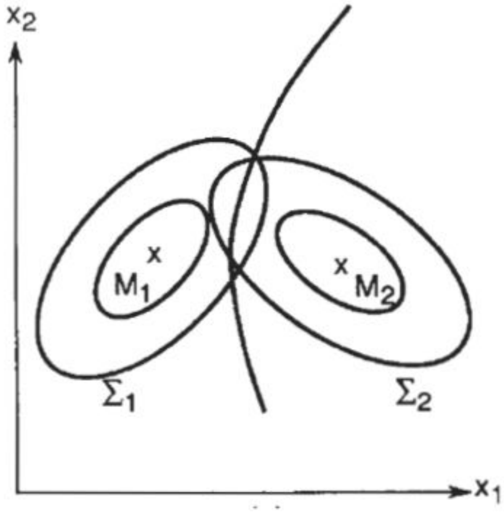
$$x^\top(\Sigma_1^{-1} - \Sigma_0^{-1})x - 2\left(\mu_1^\top \Sigma_1^{-1} - \mu_0^\top \Sigma_0^{-1}\right)x + \left(\mu_0^\top \Sigma_0^{-1}\mu_0 - \mu_1^\top \Sigma_1^{-1}\mu_1\right) = C$$

$$\Rightarrow x^\top Q x - 2b^\top x + c = 0$$

The decision boundary is a quadratic function. In 2-d case, it looks like an ellipse, or a parabola, or a hyperbola.

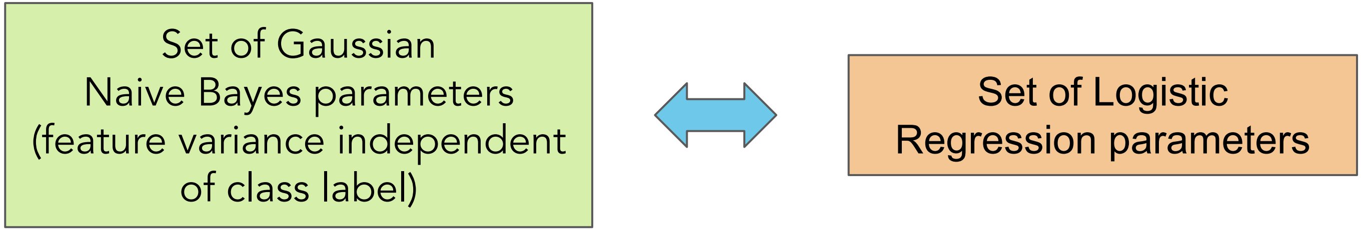
- Depending on the Gaussian distributions, the decision boundary can be very different





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

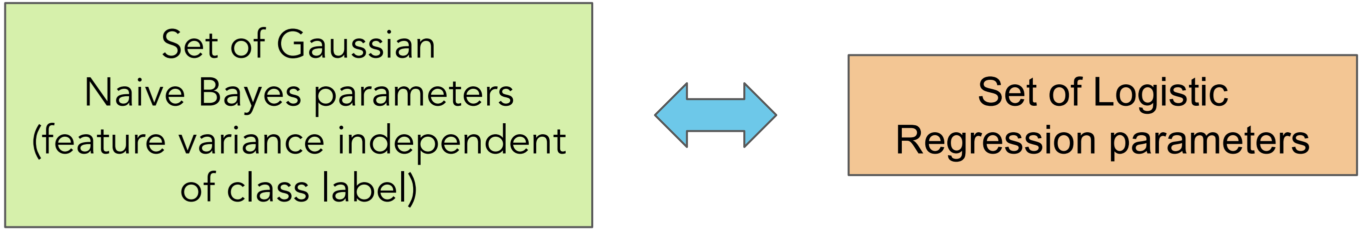
Gaussian Naive Bayes vs. Logistic Regression



Number of parameters

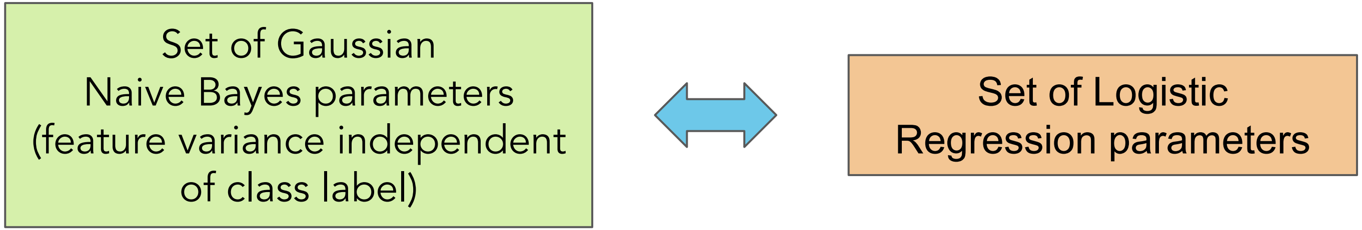
- Naive Bayes: $4D + 1$
  - When all random variables are binary
  - $4D + 1$ for Gaussians: $2D$ mean, $2D$ variance, and 1 for prior
- Logistic Regression: D
  - $\theta_1, \theta_2, ..., \theta_D$
- where D represents the number of features in the input data.

Gaussian Naive Bayes vs. Logistic Regression



- Estimation method:
    - Naive Bayes parameter estimates are decoupled (closed-form, easy)
    - Logistic regression parameter estimates are coupled (SGD, not easy)

Gaussian Naive Bayes vs. Logistic Regression



- Representation equivalence (both yield linear decision boundaries)
  - But only in special case (GNB with class-independent variances)
  - LR makes no assumptions about $P(\mathbf{X}|Y)$ in learning
  - Optimize different functions -> Obtain different solutions

Gaussian Naive Bayes vs. Logistic Regression

- Asymptotic comparison (# training examples → infinity)

- When model assumptions correct
  - Naive Bayes, logistic regression produce identical classifiers
  - Naive Bayes converges faster

- When model assumptions incorrect
  - logistic regression is less biased - does not assume conditional independence
  - logistic regression has fewer parameters
  - therefore expected to outperform Naive Bayes

Ng, Andrew, and Michael Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." *Advances in neural information processing systems* 14 (2001).

Gaussian Naive Bayes vs. Logistic Regression

Exploration Unlabeled Data

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

$$P(x) = \sum_y P(x|y)P(y)$$

Gaussian Naive Bayes vs. Logistic Regression

Exploration Unlabeled Data

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

$$P(x) = \sum_y P(x|y)P(y)$$

MLE

$$\max_\theta \log P_\theta(x) = \log \sum_y P(x|y)P(y)$$

Gaussian Naive Bayes vs. Logistic Regression

Exploration Unlabeled Data

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

$$P(x) = \sum_y P(x|y)P(y) \; = \; \textstyle\sum_y P(y)\mathcal{N}(x|\mu_y, \Sigma_y)$$

MLE

$$\max_\theta \log P_\theta(x) = \log \sum_y P(x|y)P(y)$$

Gaussian Mixture Model!

CS4641 Spring 2025
Neural Networks

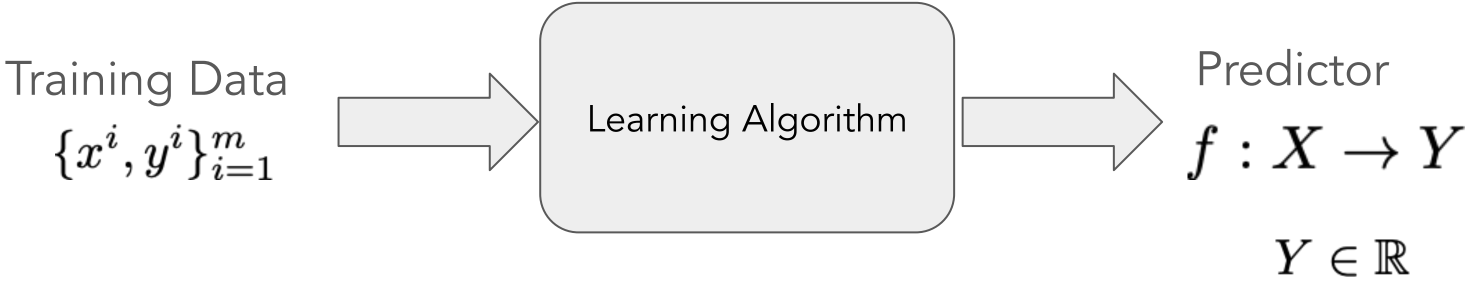
Bo Dai
School of CSE, Georgia Tech
bodai@cc.gatech.edu

ML Algorithm Pipeline

General ML Algorithm Pipeline

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

Regression Algorithms



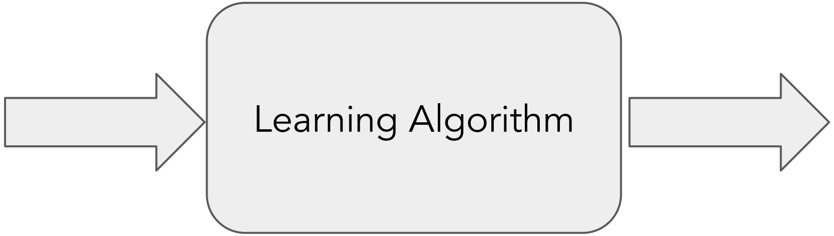
Linear Regression Pipeline

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

Training Data

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



Predictor

$f : X \to Y$

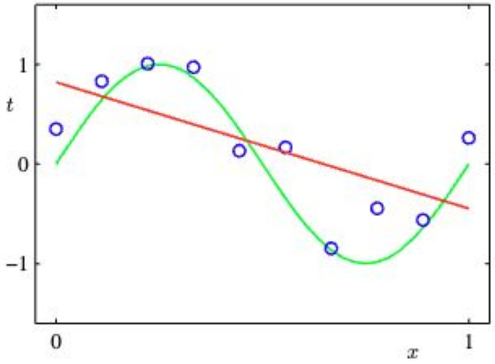
$Y \in \mathbb{R}$

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

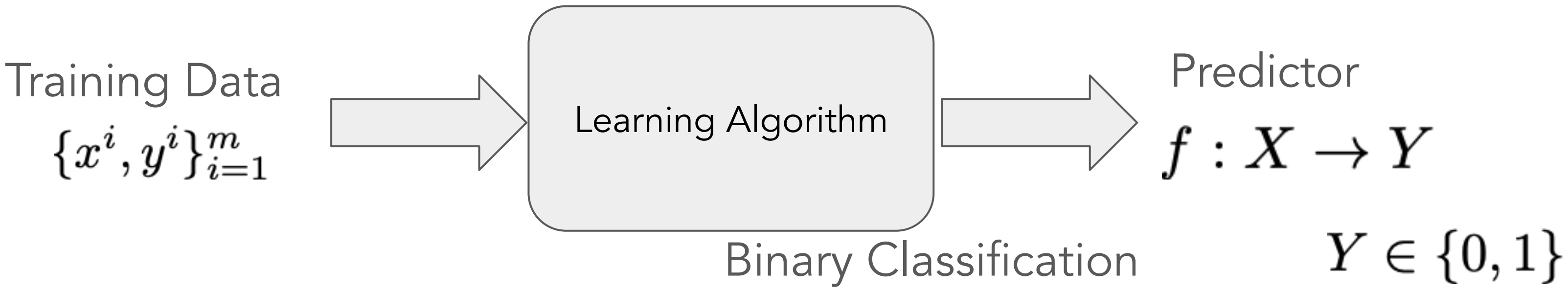
$$y = \theta^\top x + b$$

Linear Predictor



d=1

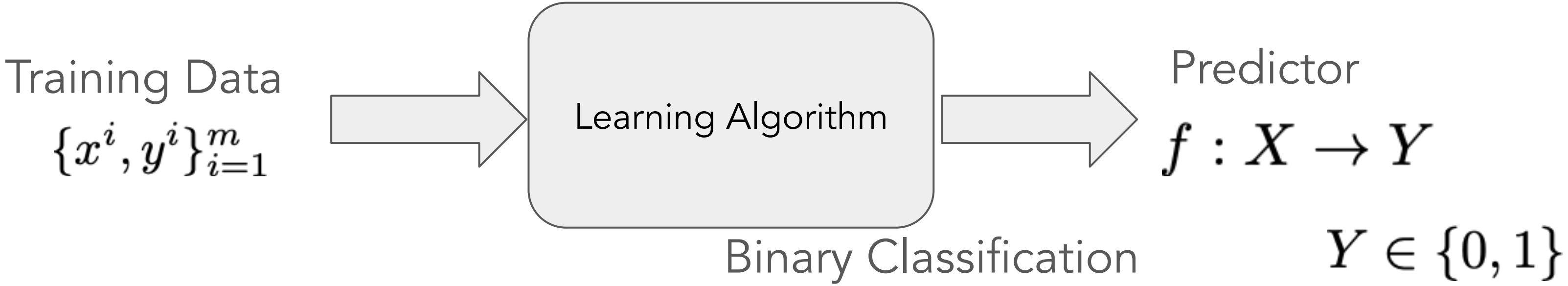
Binary Classification Algorithms



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

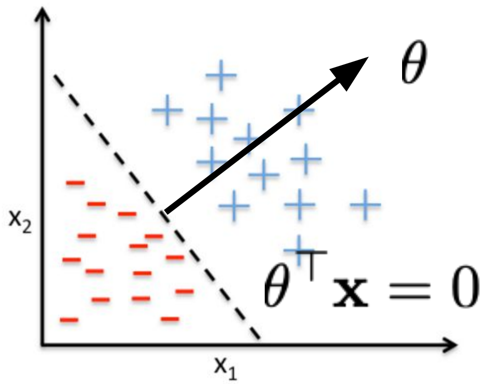
- Decision boundaries for Logistic Regression?
  - At the decision boundary, label 1/0 are equiprobable.

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

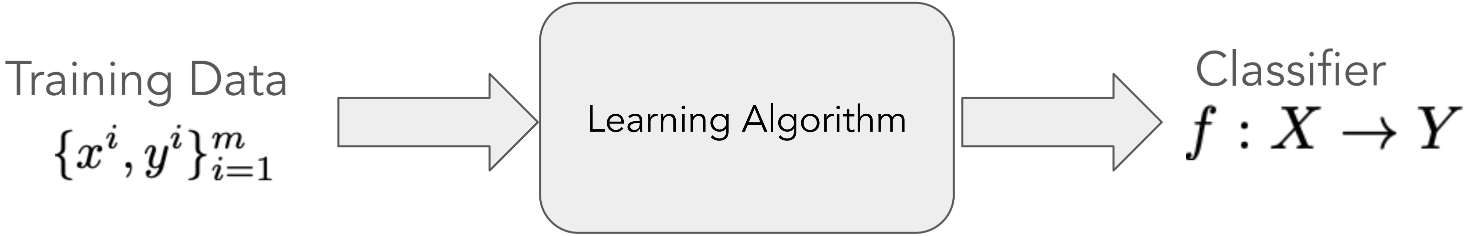
to be equal: $e^{-\theta^\top \mathbf{x}} = e^{\theta^\top \mathbf{x}}$, whose only solution is $\theta^\top \mathbf{x} = 0$.

✓ ⇒ Decision boundary is linear.

✓ ⇒ Logistic regression is a probabilistic linear classifier.



Multiclass Logistic Regression Algorithms

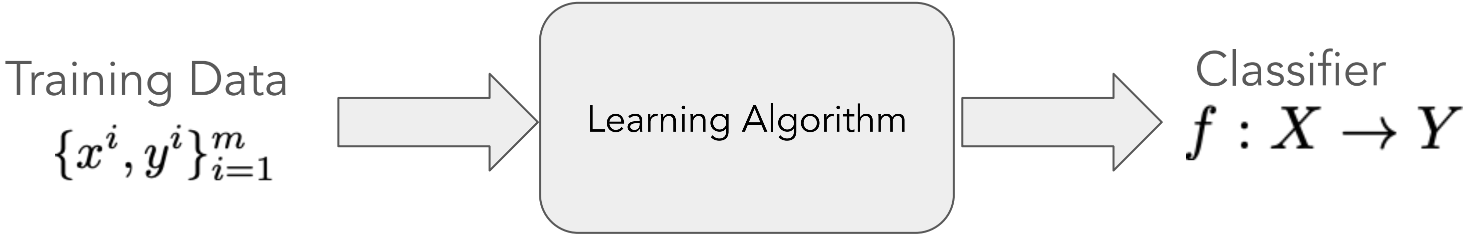


Multiclass Classification $Y \in \{0, 1, \ldots, k\}$

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



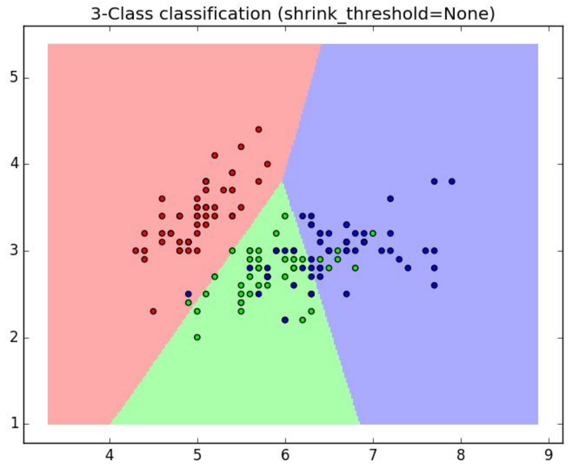
Multiclass Classification $Y \in \{0, 1, \ldots, k\}$

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

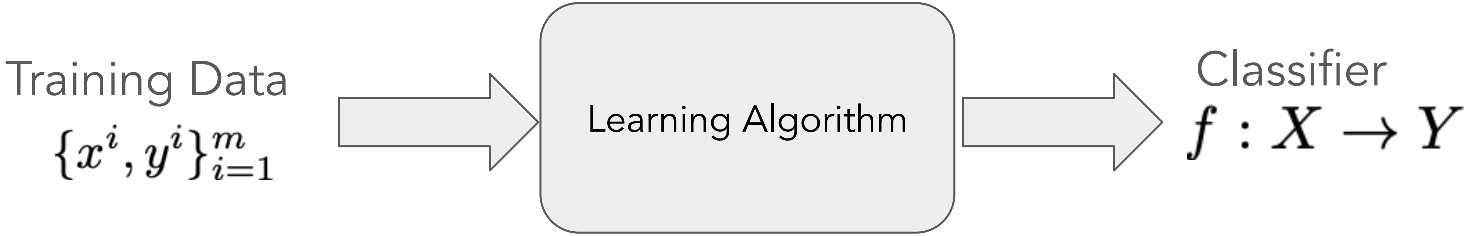
- Decision boundaries for Multiclass Logistic Regression?



✓ ⇒ Decision boundary is linear.

✓ ⇒ Multiclass Logistic regression is a probabilistic linear classifier.

Naive Bayes Classification

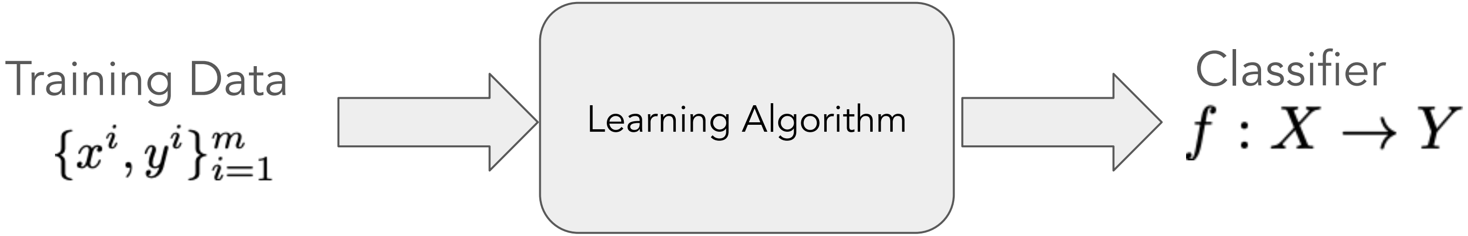


Multiclass Classification $Y \in \{0, 1, \ldots, k\}$

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

Naive Bayes Classification

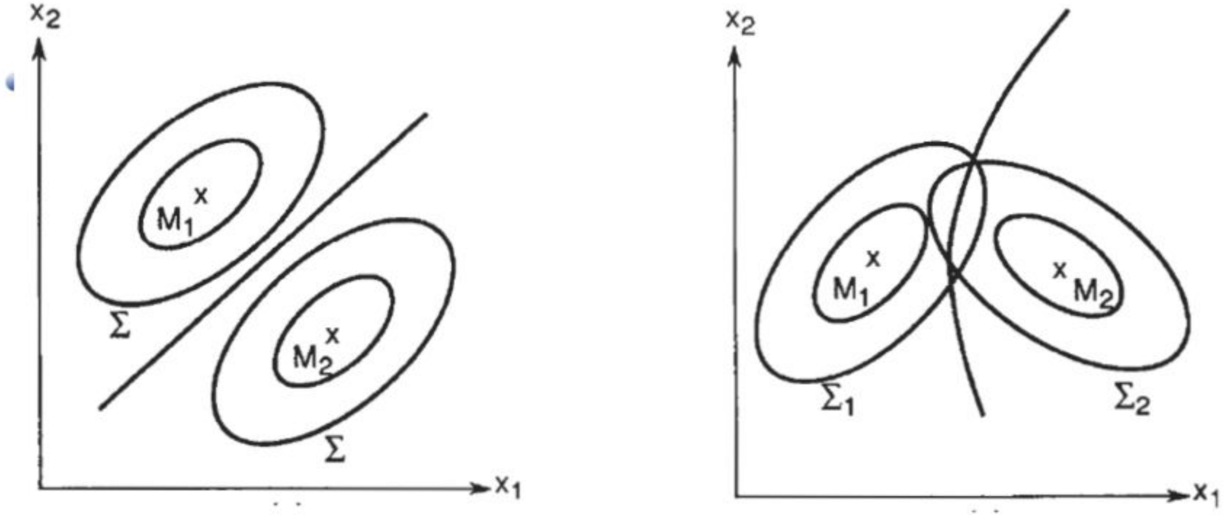


Multiclass Classification $Y \in \{0, 1, \ldots, k\}$

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

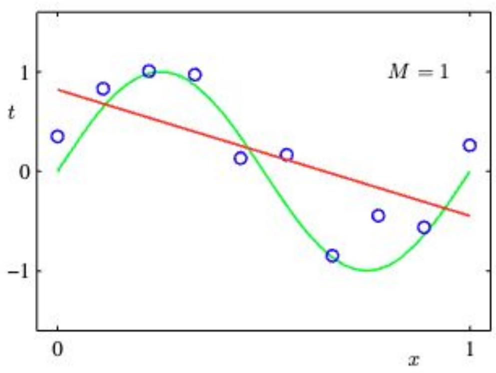
- Depending on the Gaussian distributions, the decision boundary can be very different



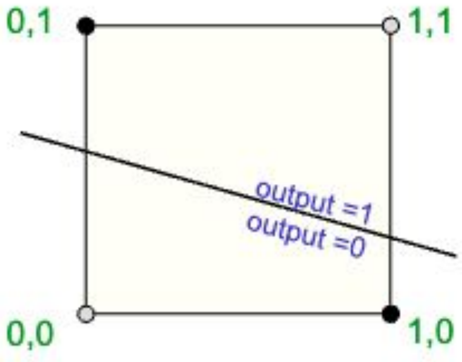
- Decision boundary: $h(\mathbf{x}) = -\ln \frac{q_i(x)}{q_j(x)} = 0$

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

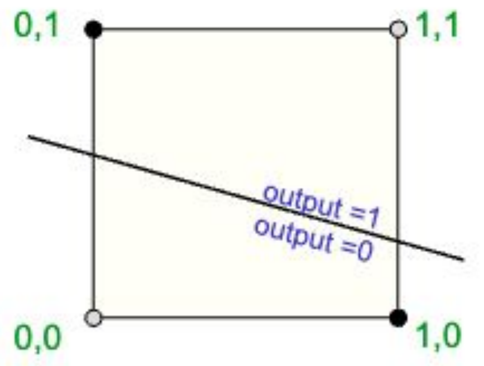


d=1



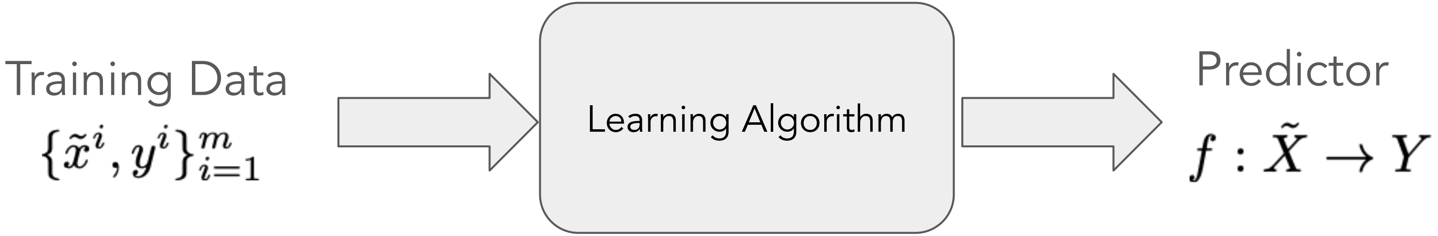
Limitations of Linear Classifier

- 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

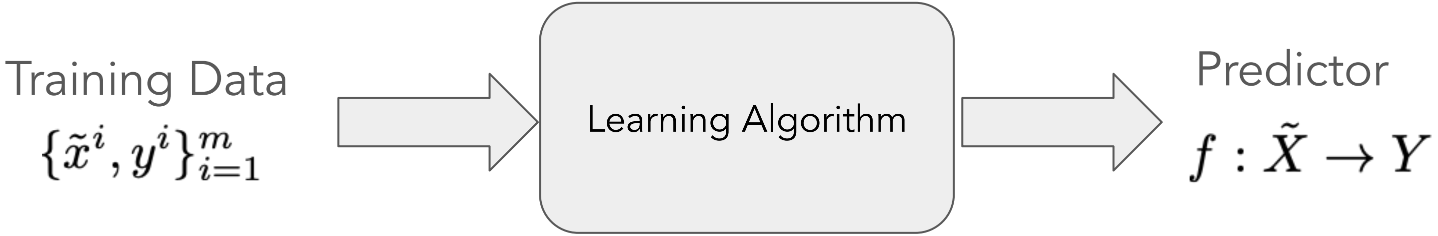
Nonlinear Parametrization: Polynomial Regression



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

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

Nonlinear Parametrization: Polynomial Regression



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

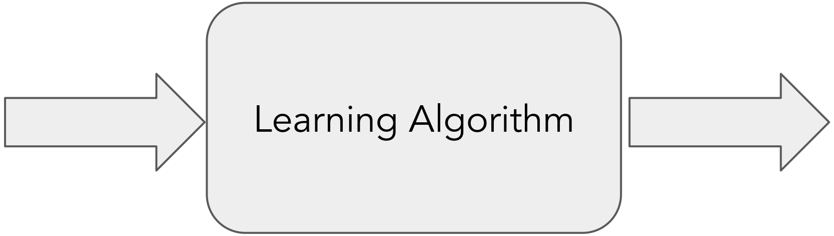
$$\begin{aligned} y = \theta_0 &+ \theta_1^1 x_1 + \theta_1^2 (x_1)^2 + \theta_1^3 (x_1)^3 + \ldots + \theta_1^d (x_1)^d \\ &+ \theta_2^1 x_2 + \theta_2^2 (x_2)^2 + \theta_2^3 (x_2)^3 + \ldots + \theta_2^d (x_2)^d \\ &+ \ldots \\ &+ \theta_n^1 x_n + \theta_n^2 (x_n)^2 + \theta_n^3 (x_n)^3 + \ldots +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

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

Nonlinear Parametrization: Polynomial Regression

Training Data

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



Predictor

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

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

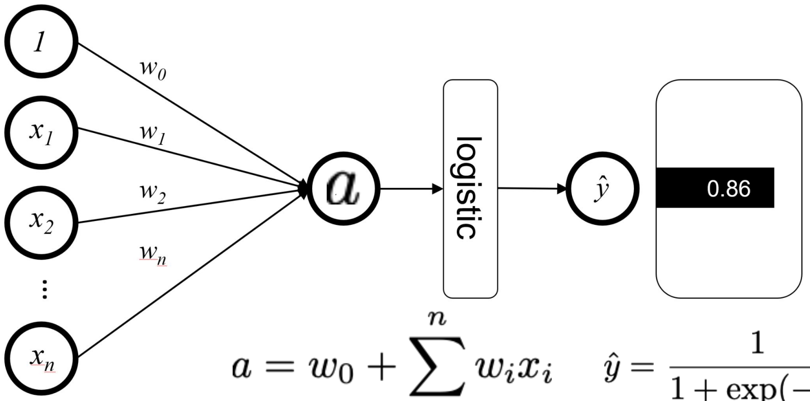
$$\begin{aligned} y = \theta_0 &+ \theta_1^1 x_1 + \theta_1^2 (x_1)^2 + \theta_1^3 (x_1)^3 + \ldots + \theta_1^d (x_1)^d \\ &+ \theta_2^1 x_2 + \theta_2^2 (x_2)^2 + \theta_2^3 (x_2)^3 + \ldots + \theta_2^d (x_2)^d \\ &+ \ldots \\ &+ \theta_n^1 x_n + \theta_n^2 (x_n)^2 + \theta_n^3 (x_n)^3 + \ldots +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

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

Combinatorial Parametrization

Better Nonlinear Parametrization: Neural Network

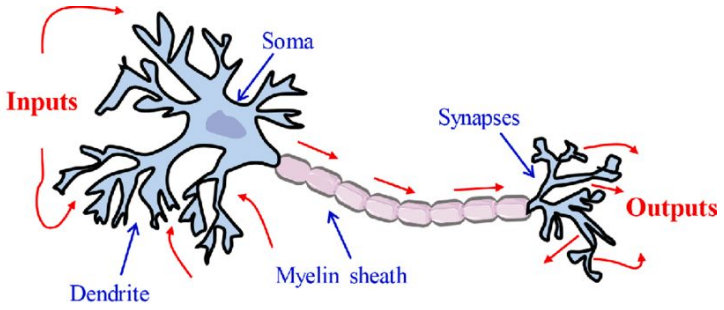
Logistic Regression Revisit

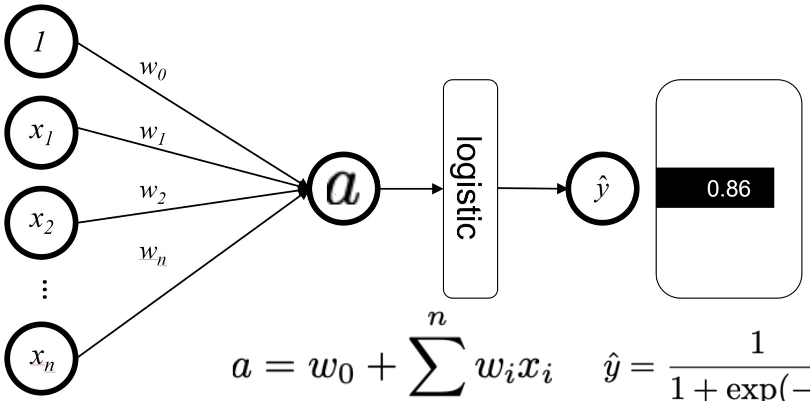


$$a = w_0 + \sum_{i=1}^{n} w_i x_i \qquad \hat{y} = \frac{1}{1 + \exp(-a)}$$

Better Nonlinear Parametrization: Neural Network

Neuron ⟺ Logistic Regression

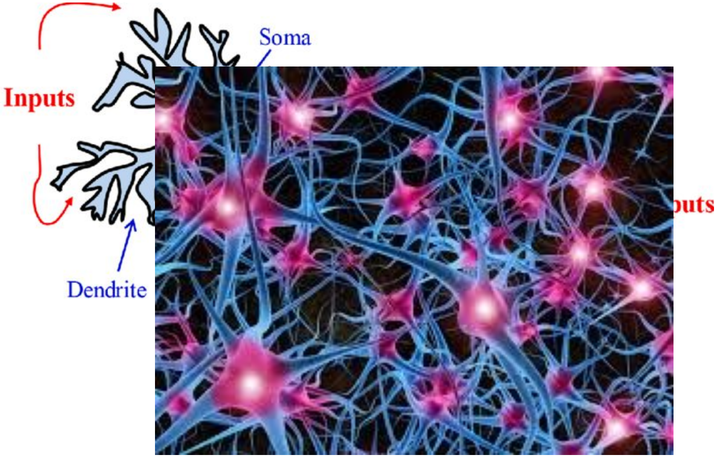


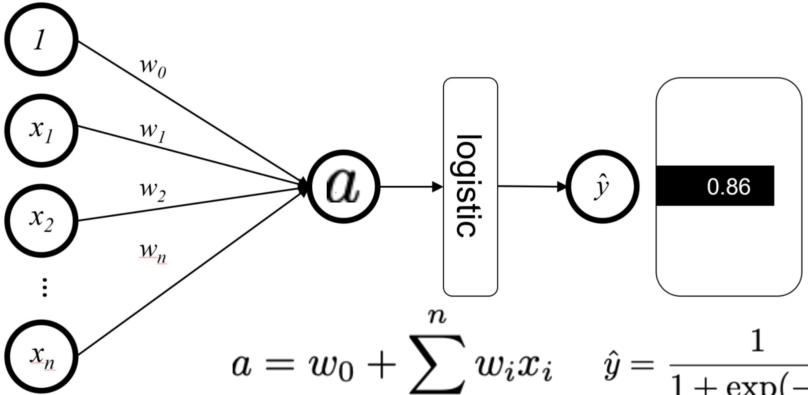


$$a = w_0 + \sum_{i=1}^{n} w_i x_i \qquad \hat{y} = \frac{1}{1 + \exp(-a)}$$

Better Nonlinear Parametrization: Neural Network

Neural Network ⟺ Composition of Neurons

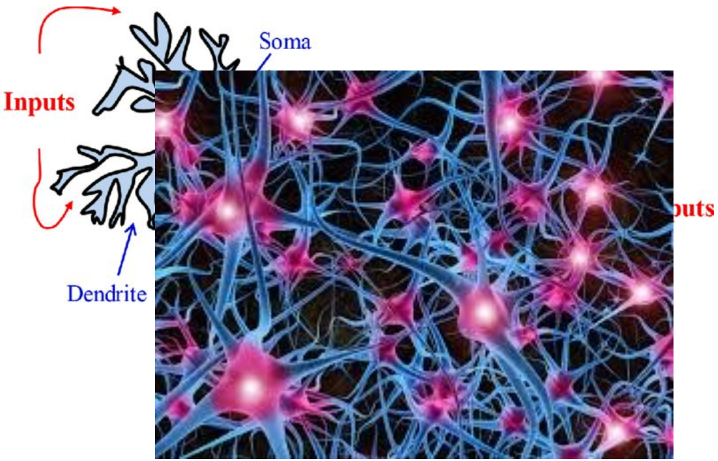


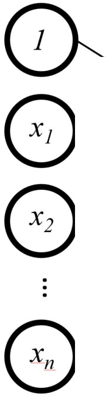


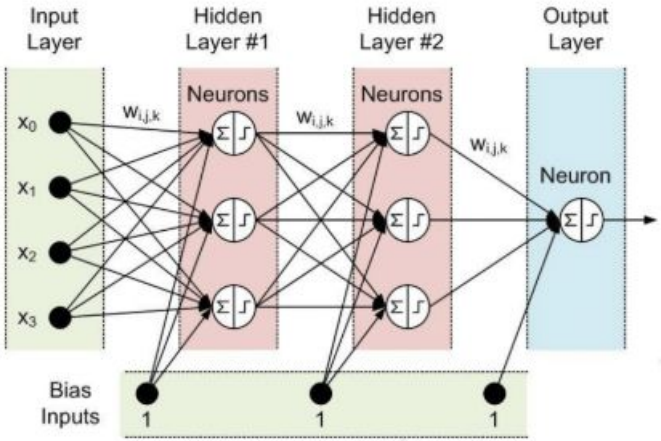
$$a = w_0 + \sum_{i=1}^{n} w_i x_i \qquad \hat{y} = \frac{1}{1 + \exp(-a)}$$

Better Nonlinear Parametrization: Neural Network

Neural Network $\Longleftrightarrow$ Composition of Neurons



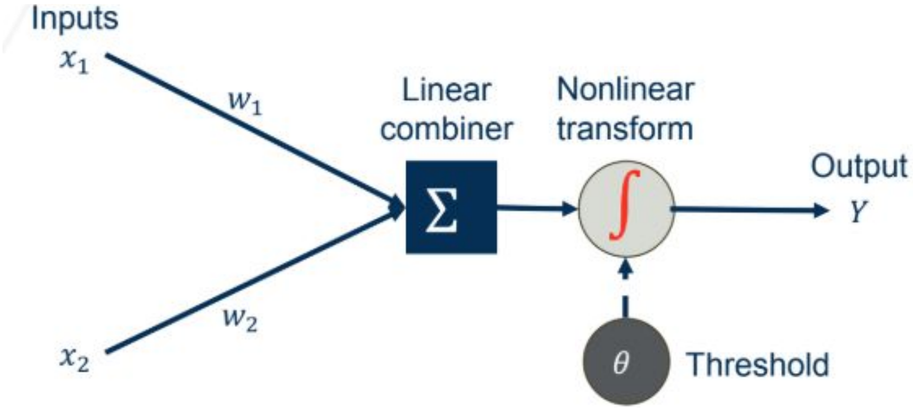




Alternative Neurons

- Use different nonlinear transformations $f(u)$

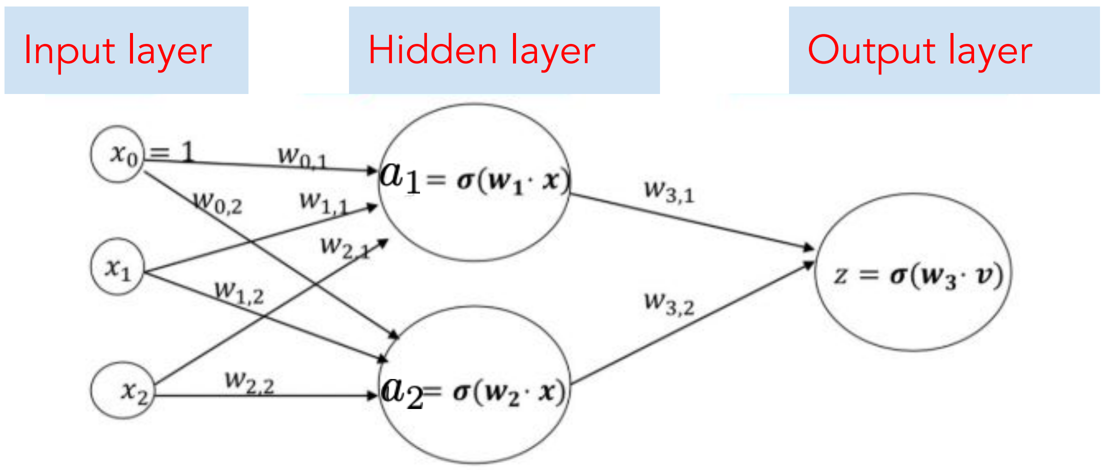
- Before that, perform weighted combination of inputs $u = w^T x$



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

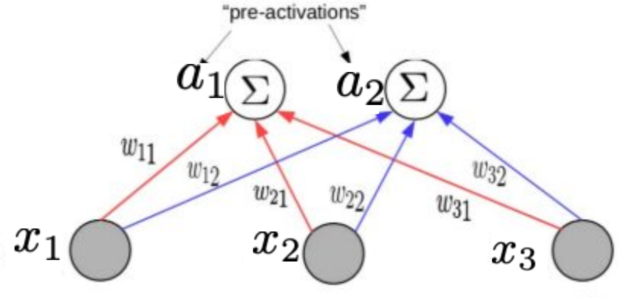
- The classifier/regressor is a multilayer network of units
- Each unit takes some inputs and produces one output. Output of one unit can be the input of another.
  - Advantage: Can produce highly non-linear decision boundaries!
  - Sigmoid is differentiable, so can use gradient descent



Forward Pass in MLP

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

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



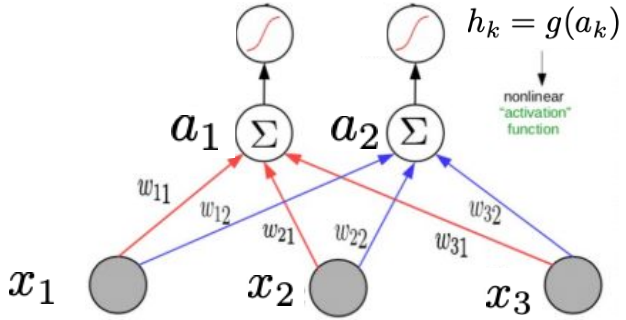
Forward Pass in MLP

- 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

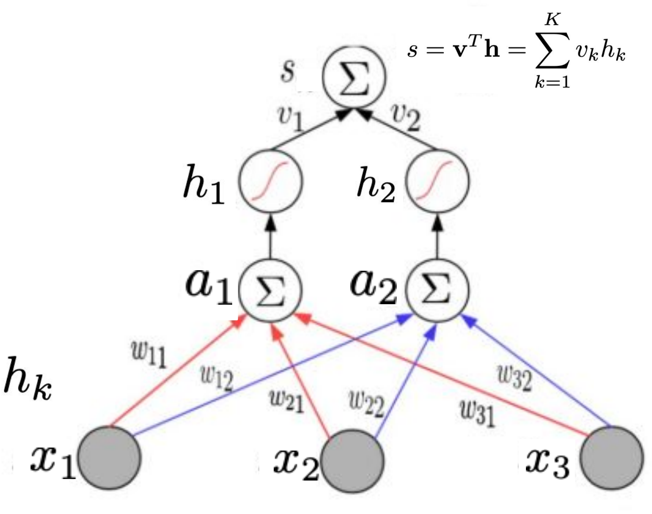
$$a_k = w_k^\top x = \sum_{i=1}^{n} w_{ki} x_i$$

- 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

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

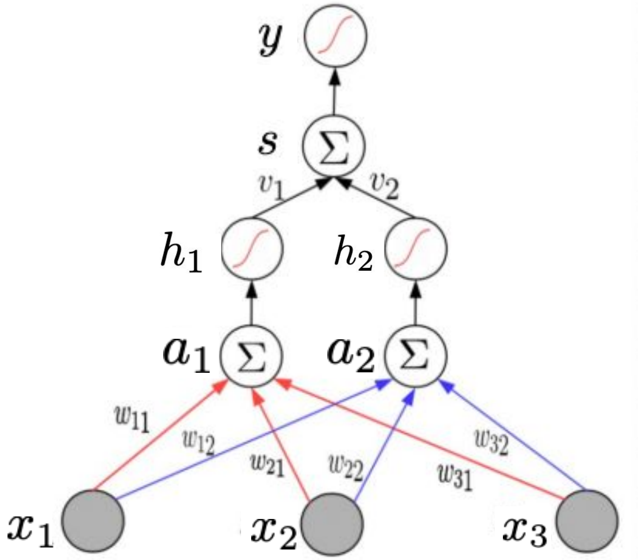
- Nonlinear activation applied on each pre-activation

$$h_k = g(a_k)$$

- 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

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

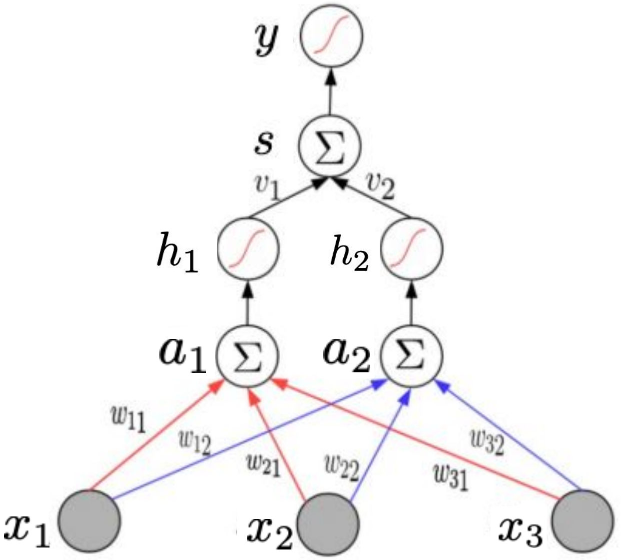
- Nonlinear activation applied on each pre-activation

$$h_k = g(a_k)$$

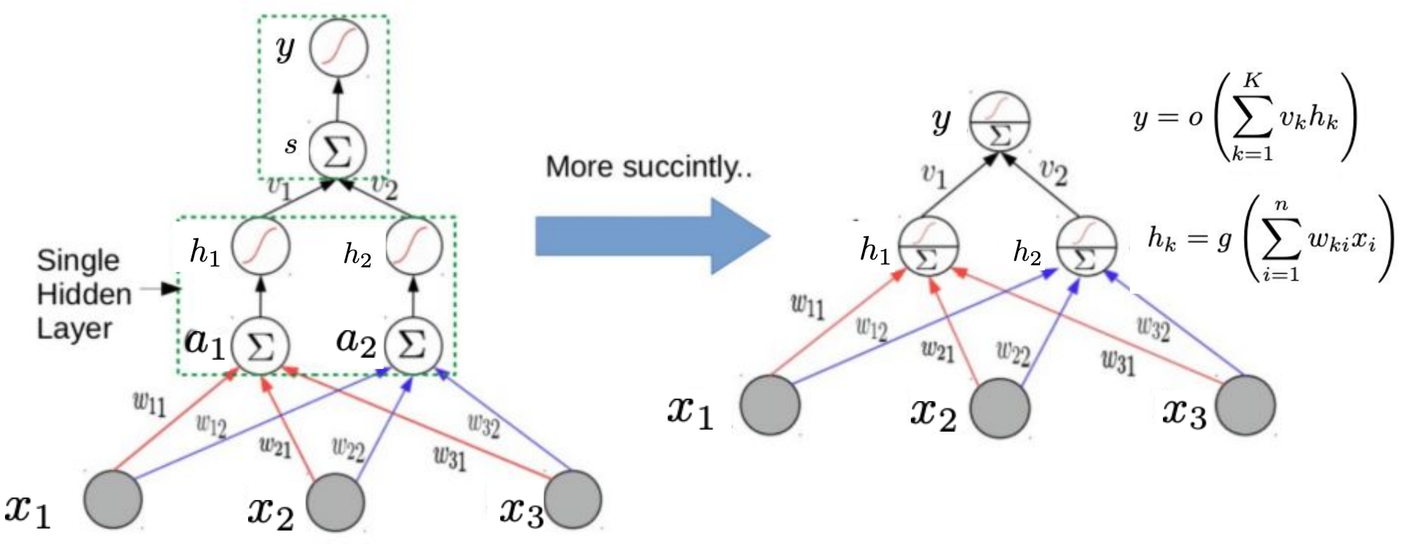
- A linear model applied on the new "features" h_k

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

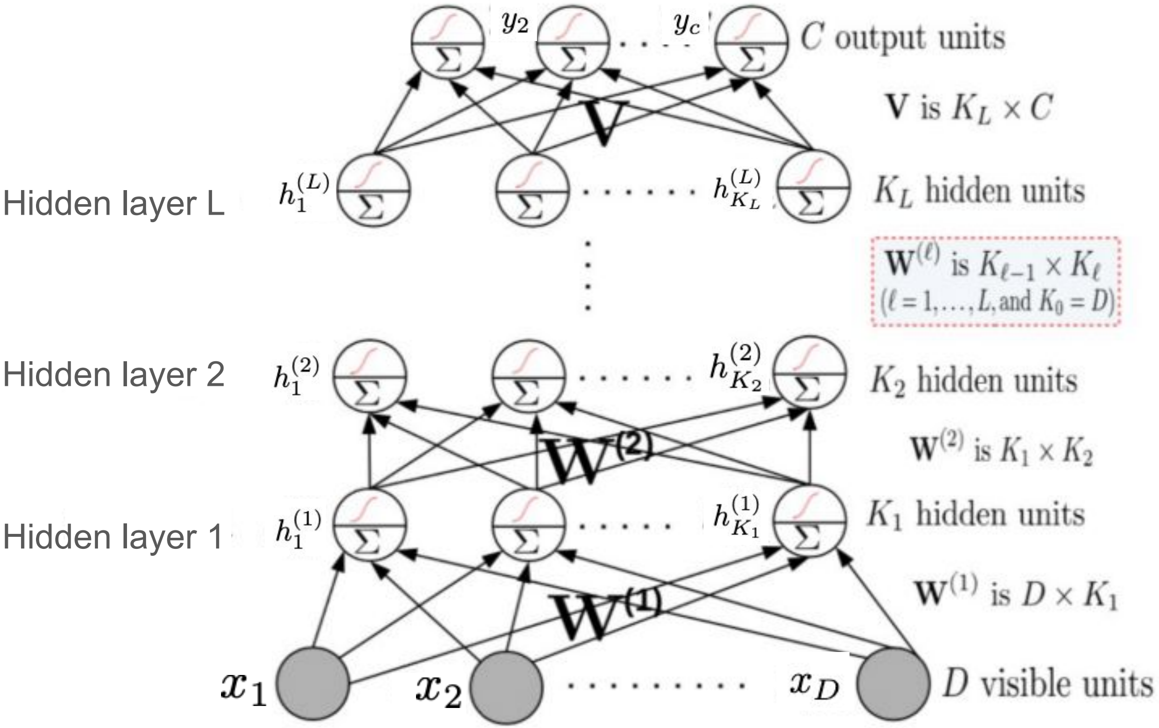
- Finally, the output is produced as $y = o(s)$
- Unknowns of the model $\mathbf{w_1}, \ldots, \mathbf{w_k}$ and $\mathbf{v}$ learned by minimizing a loss $\mathcal{L}(\mathbf{W}, \mathbf{v}) = \sum_{n=1}^{N} \ell(y_n, o(s_n))$, e.g, squared, logistic, softmax, etc (depending on the output)



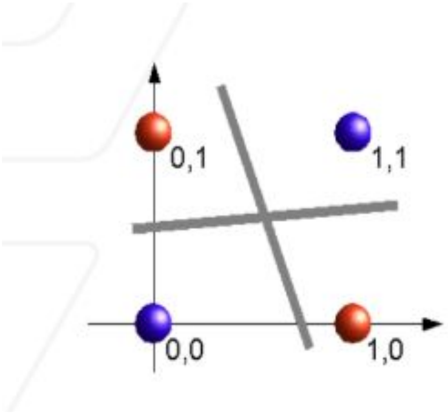
Compact Illustration



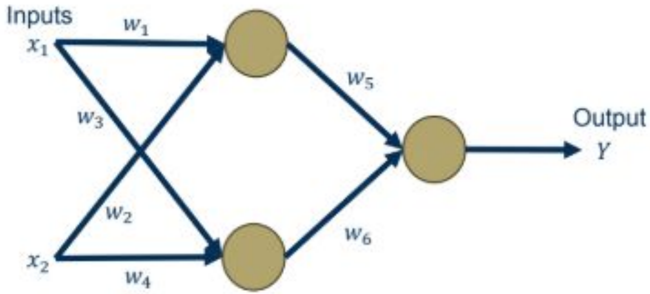
Multi-Layer, Multi-Hidden Units and Multi-Outputs Extension



Multi-layer Perception for XOR problem



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



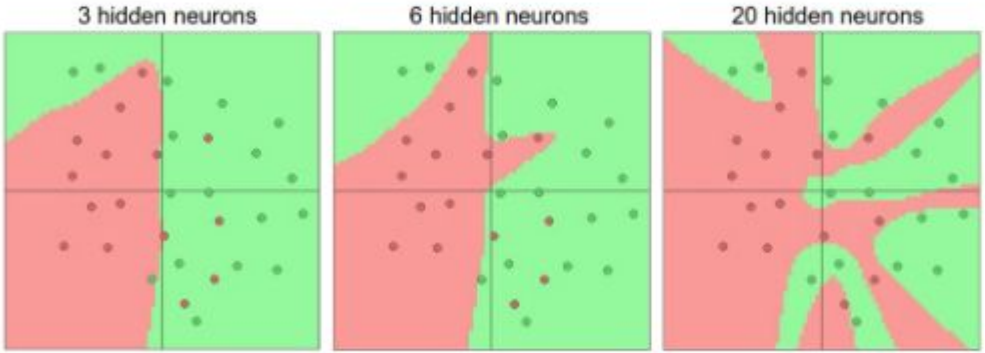
A possible set of weight:

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

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

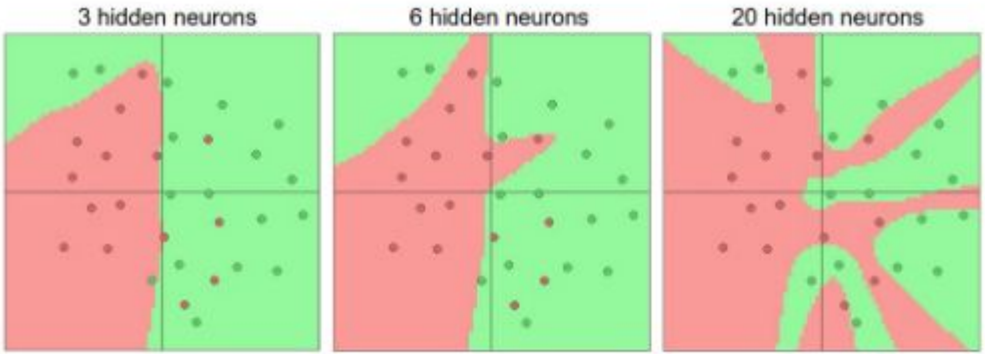


- The capacity of the network increases with more hidden units and more hidden layers

Representational Power

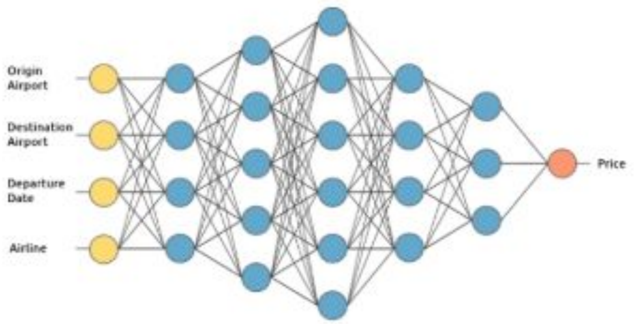
- Neural network with at least one hidden layer is a universal approximator (can represent any function).

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



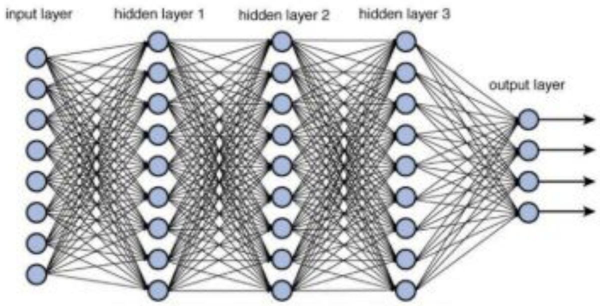
- The capacity of the network increases with more hidden units and more hidden layers (Depth vs. Width)

Neural Network Architecture



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

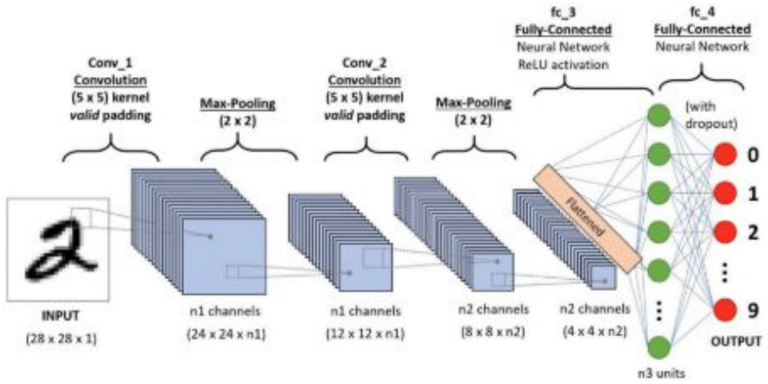
FF with Fully connected (*dense*) layers: each unit of layer *i* is fully connected with the units of the previous layer



Deep network (>3 hidden layers?)

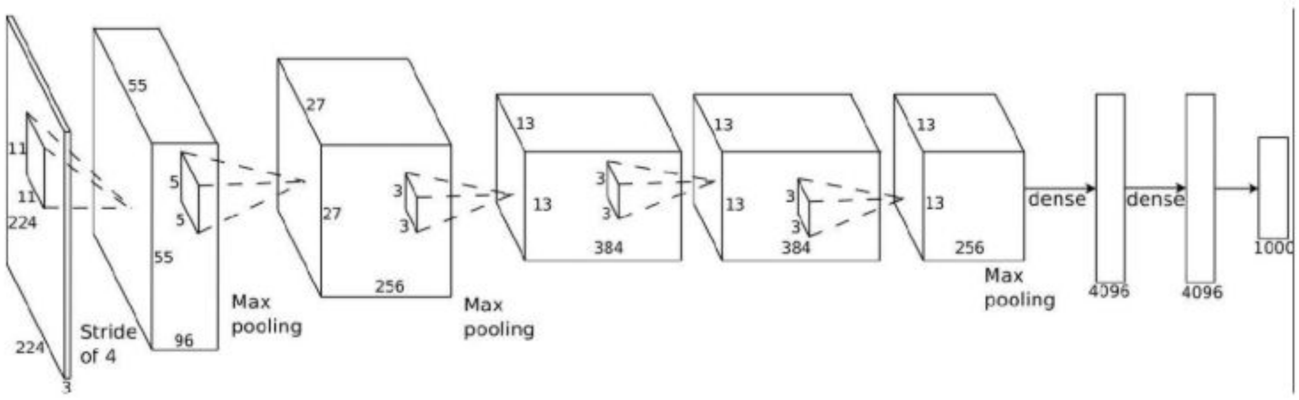
Convolutional Neural Network (CNN):

Not (all) fully connected layers

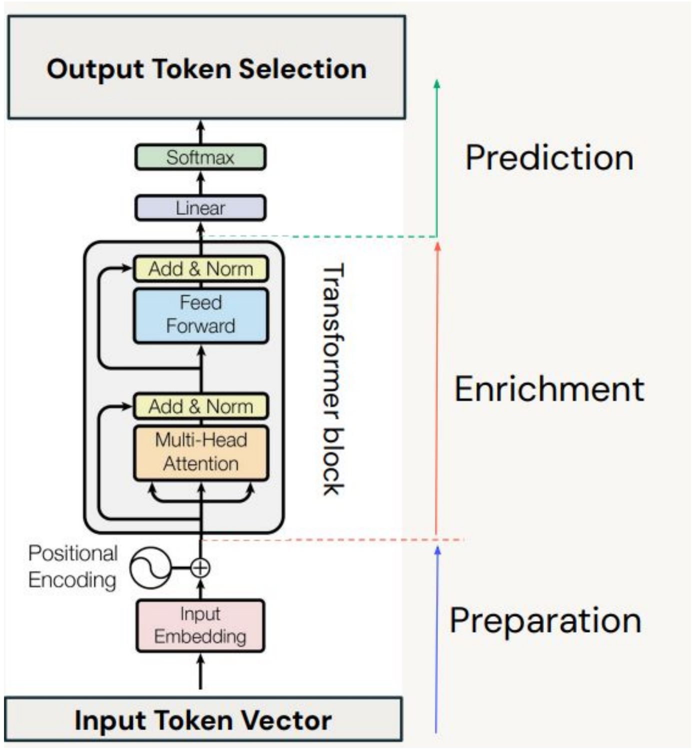


AlexNet

- 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