Introduction to Big Data Analysis
Classification : Part 2

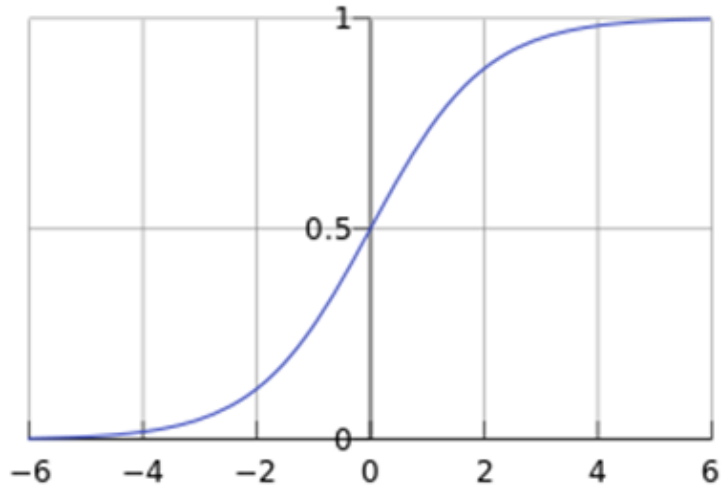
Zhen Zhang

Southern University of Science and Technology

Outlines

Logistic Regression

- Not regression, but a classification method
- Connection with linear regression : $y = w_0 + w_1 x + \epsilon$, y is binary (0 or 1) ; then $E(y|x) = P(y = 1|x) = w_0 + w_1 x$; but $w_0 + w_1 x$ may not be a probability
- Find a function to map it back to $[0, 1]$: Sigmoid function $g(z) = \frac{1}{1+e^{-z}}$ with $z = w_0 + w_1 x_1 + \ldots + w_d x_d$



- Equivalently, $\log \frac{P(y=1|x)}{1-P(y=1|x)} = w_0 + w_1 x_1 + \ldots + w_d x_d$, logit transform $logit(z) = \log \frac{z}{1-z}$

MLE for Logistic Regression

- The prob. distribution for two-class logistic regression model is

$$Pr(y = 1|\mathbf{X} = \mathbf{x}) = \frac{\exp(\mathbf{w}^T\mathbf{x})}{1 + \exp(\mathbf{w}^T\mathbf{x})},$$
$$Pr(y = 0|\mathbf{X} = \mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^T\mathbf{x})}.$$

- Let $P(y = k|\mathbf{X} = \mathbf{x}) = p_k(\mathbf{x}; \mathbf{w})$, $k = 0$ or 1. The likelihood function is defined by $L(\mathbf{w}) = \prod_{i=1}^{n} p_{y_i}(\mathbf{x}_i; \mathbf{w})$
- MLE estimate of $\mathbf{w}$: $\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} L(\mathbf{w})$
- Solve $\nabla_{\mathbf{w}} \log L(\mathbf{w}) = 0$ by Newton-Raphson method

Outlines

Linear Discriminant Analysis (LDA)

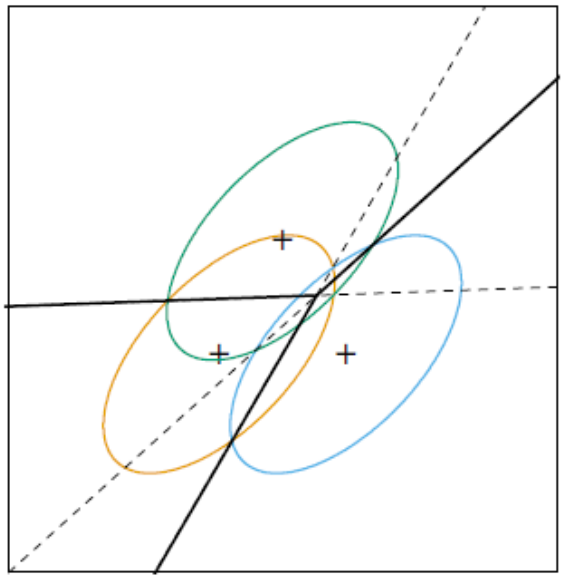
- Bayes Classifier amounts to know the class posteriors $\mathrm{P}(Y|\mathbf{X})$ for optimal classification : $k^* = \arg\max_k \mathrm{P}(Y = k|\mathbf{X})$
- Let $\pi_k = \mathrm{P}(Y = k)$ be the prior probability, $f_k(\mathbf{x}) = \mathrm{P}(\mathbf{X} = \mathbf{x}|Y = k)$ be the density function of samples in each class $Y = k$
- By Bayes theorem, $\mathrm{P}(Y|\mathbf{X} = \mathbf{x}) \propto f_k(\mathbf{x})\pi_k$ (Recall naive Bayes)
- Assume $f_k(x)$ is multivariate Gaussian : $f_k(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\mu_k)}$, with a common covariance matrix $\mathbf{\Sigma}_k = \mathbf{\Sigma}$, sufficient to look at the log-ratio

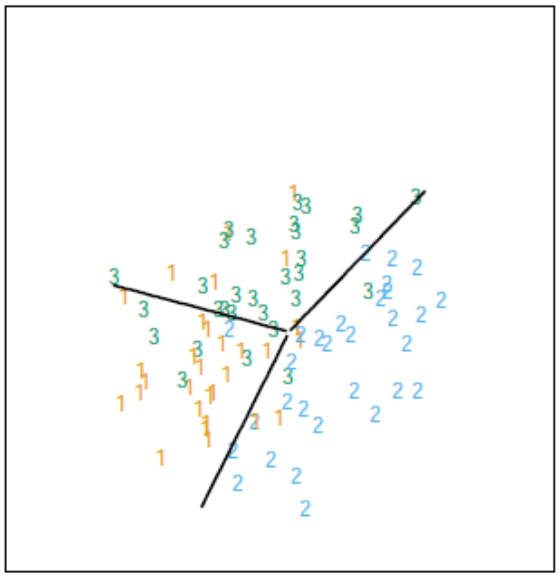
$$\log\frac{\mathrm{P}(Y = k|\mathbf{X} = \mathbf{x})}{\mathrm{P}(Y = l|\mathbf{X} = \mathbf{x})} = \log\frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k + \mu_l)^T\mathbf{\Sigma}^{-1}(\mu_k - \mu_l) + \mathbf{x}^T\mathbf{\Sigma}^{-1}(\mu_k - \mu_l)$$

for the decision boundary between class k and l

Discriminant Rule

- Linear discriminant functions :
  $\delta_k(\mathbf{x}) = \mathbf{x}^T \mathbf{\Sigma}^{-1} \mu_k - \frac{1}{2} \mu_k^T \mathbf{\Sigma}^{-1} \mu_k + \log \pi_k$
- Then $\log \frac{\mathrm{P}(Y=k|\mathbf{X}=\mathbf{x})}{\mathrm{P}(Y=l|\mathbf{X}=\mathbf{x})} = \delta_k(\mathbf{x}) - \delta_l(\mathbf{x})$
- Decision rule : $k^* = \arg\max_k \delta_k(\mathbf{x})$
- Sample estimate of unknowns : $\hat{\pi}_k = N_k/N$, where
  $N = \sum_{k=1}^K N_k$, $\hat{\mu}_k = \frac{1}{N_k} \sum_{y_i=k} \mathbf{x}_i$,
  $\hat{\Sigma} = \frac{1}{N-K} \sum_{k=1}^K \sum_{y_i=k} (\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^T$



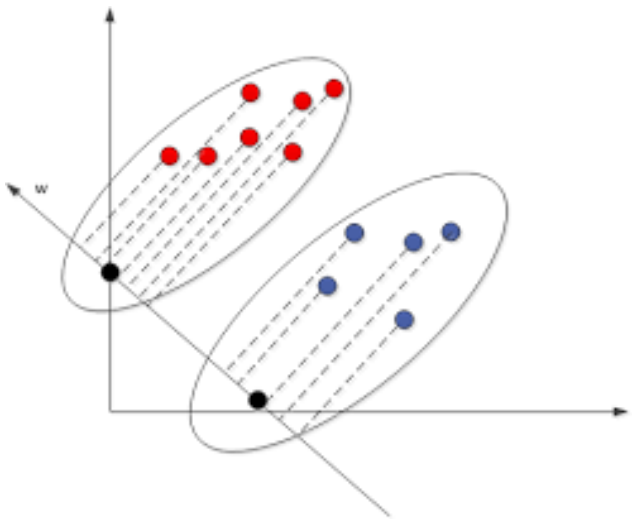


Two-class LDA

- LDA rule classifies to class 2 if

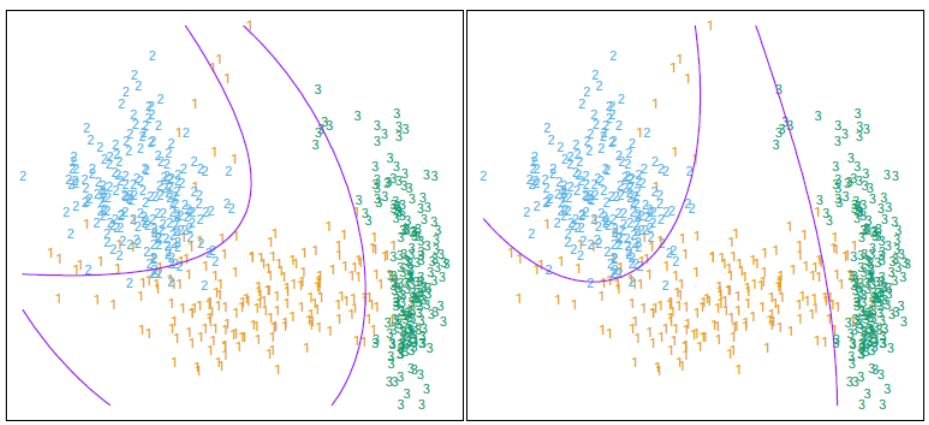
$$(\mathbf{x} - \frac{\hat{\mu}_1 + \hat{\mu}_2}{2})^T \mathbf{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) + \log \frac{\hat{\pi}_2}{\hat{\pi}_1} > 0$$

- Discriminant direction : $\beta = \mathbf{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$
- Bayes misclassfication rate $= 1 - \Phi(\beta^T(\mu_2 - \mu_1)/(\beta^T \mathbf{\Sigma} \beta)^{\frac{1}{2}})$, where $\Phi(x)$ is the Gaussian distribution function



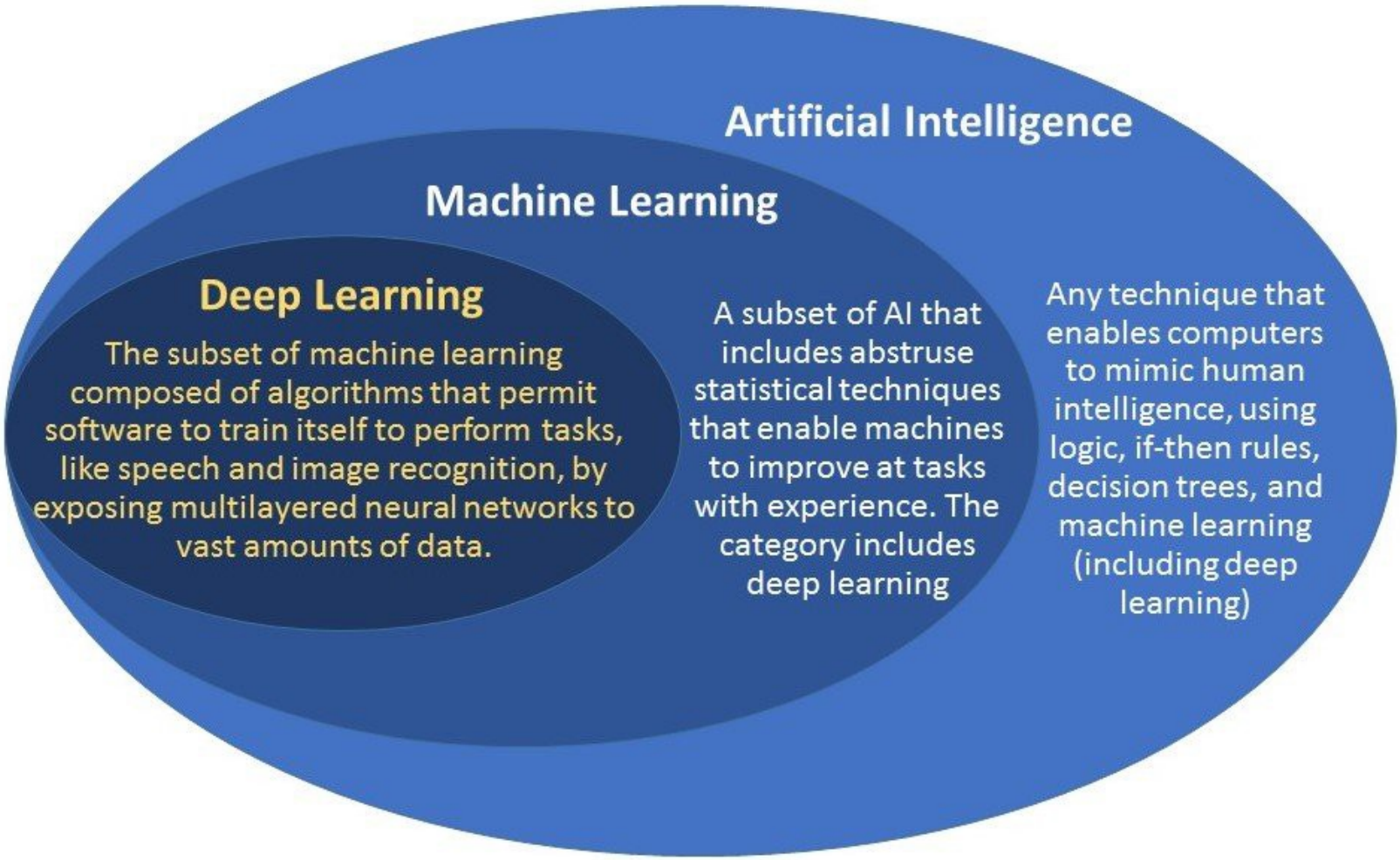
Other Variants

- Quadratic discriminant analysis (QDA) :
  $\delta_k(\mathbf{x}) = -\frac{1}{2}\log|\mathbf{\Sigma}_k| - \frac{1}{2}(\mathbf{x}-\mu_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\mu_k) + \log\pi_k$
- Regularized discriminant analysis : $\hat{\mathbf{\Sigma}}_k(\alpha) = \alpha\hat{\mathbf{\Sigma}}_k + (1-\alpha)\hat{\mathbf{\Sigma}}$
- Computations for LDA :
  1. Sphere the data with respect to $\hat{\mathbf{\Sigma}} = \mathbf{U}\mathbf{D}\mathbf{U}^T$: $\mathbf{X}^* = \mathbf{D}^{-\frac{1}{2}}\mathbf{U}^T\mathbf{X}$. Then the common covariance estimate of $\mathbf{X}^*$ is $\mathbf{I}_p$
  2. Classsify to the closest class centroid in the transformed space, taking into account of the class prior probabilities π_k's
- Reduced-Rank LDA : see dimensionality reduction



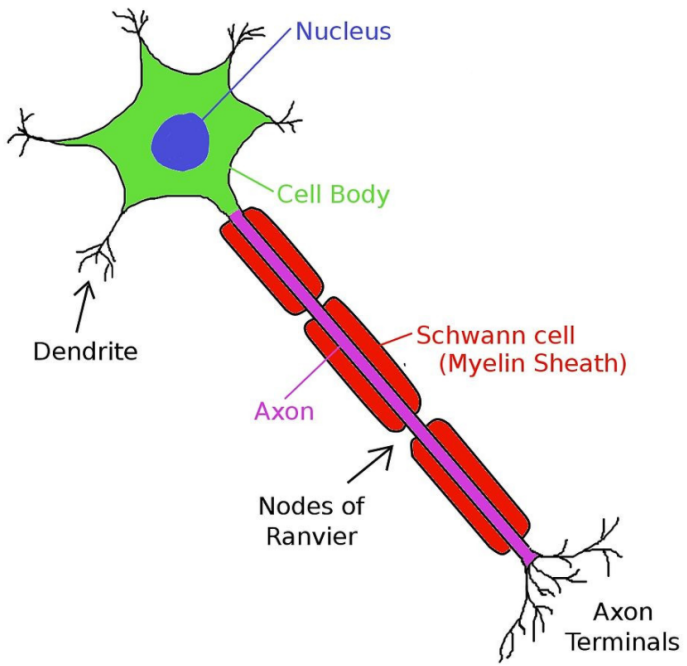
Outlines

What is Deep Learning ?



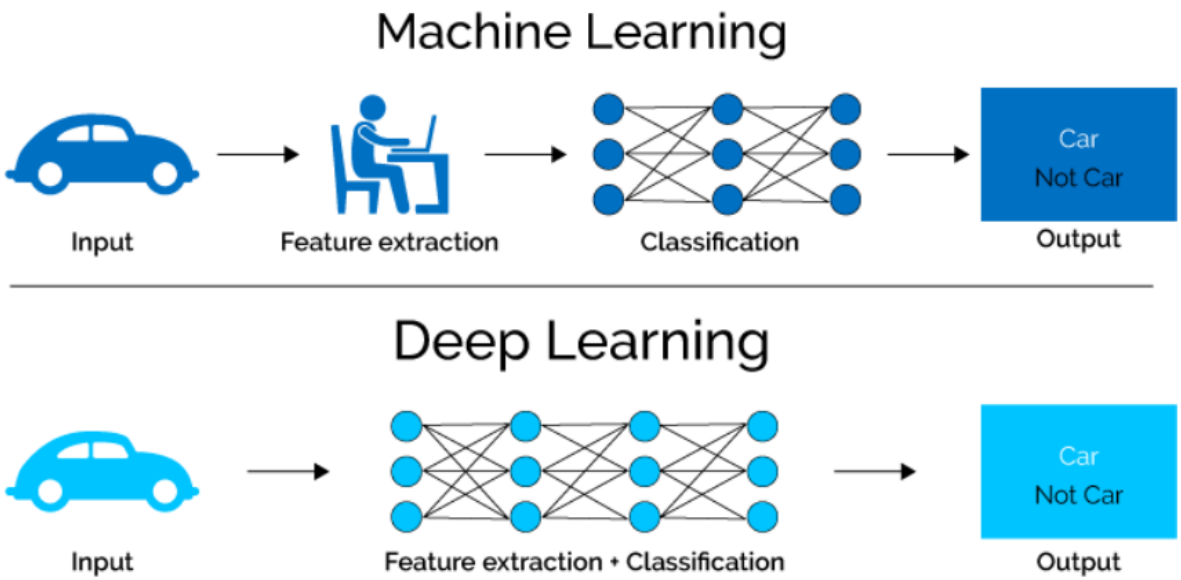
Deep Learning

- Deep learning is a sub field of Machine Learning that very closely tries to mimic human brain's working using neurons.
- These techniques focus on building Artificial Neural Networks (ANN) using several hidden layers.
- There are a variety of deep learning networks such as Multilayer Perceptron (MLP), Autoencoders (AE), Convolution Neural Network (CNN), Recurrent Neural Network (RNN).



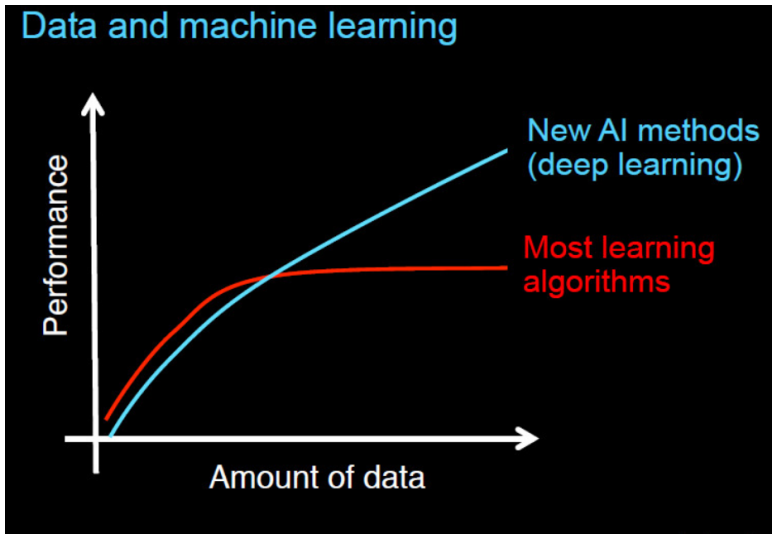
ML vs DL

- In Machine Learning, the features need to be identified by an domain expert.
- In Deep Learning, the features are learned by the neural network.

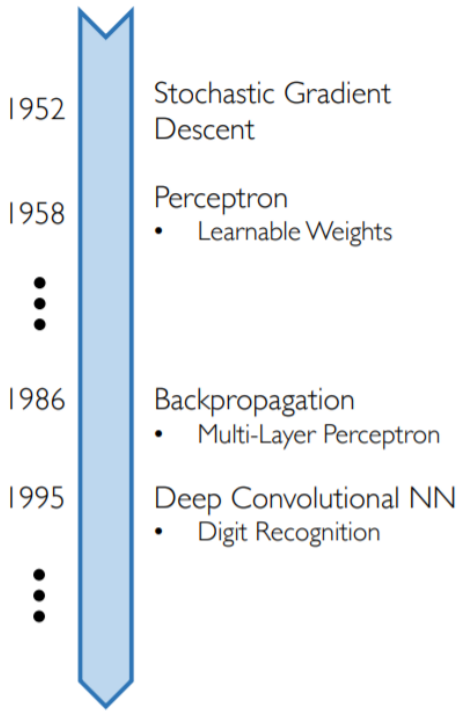


Why Deep Learning is Growing ?

- Processing power needed for Deep learning is readily becoming available using GPUs, Distributed Computing and powerful CPUs.
- Moreover, as the data amount grows, Deep Learning models seem to outperform Machine Learning models.
- Explosion of features and datasets.
- Focus on customization and real time decisioning.



Why Now ?



Neural Networks date back decades, so why the resurgence?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

2. Hardware

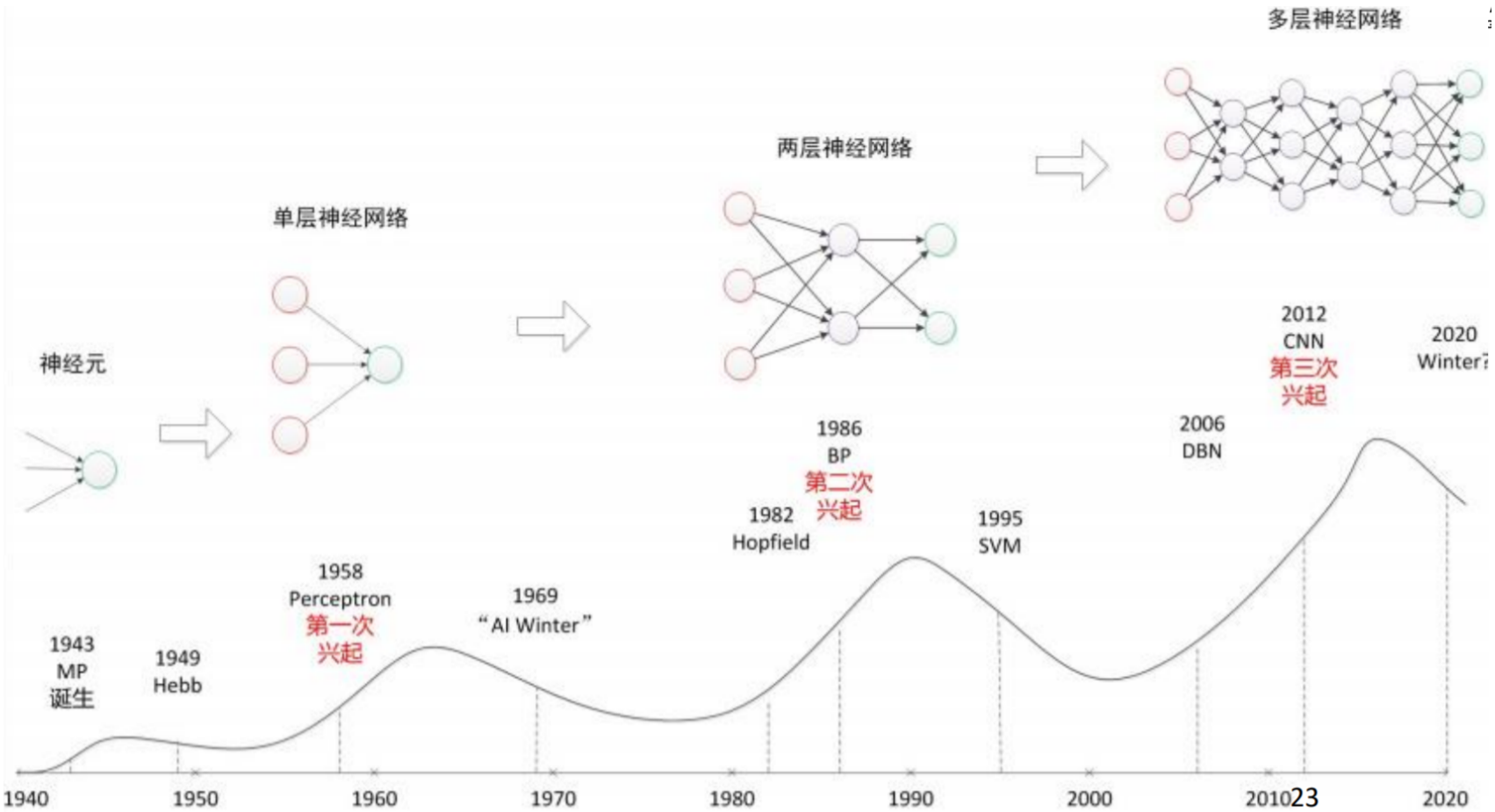
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software

- Improved Techniques
- New Models
- Toolboxes

TensorFlow

History

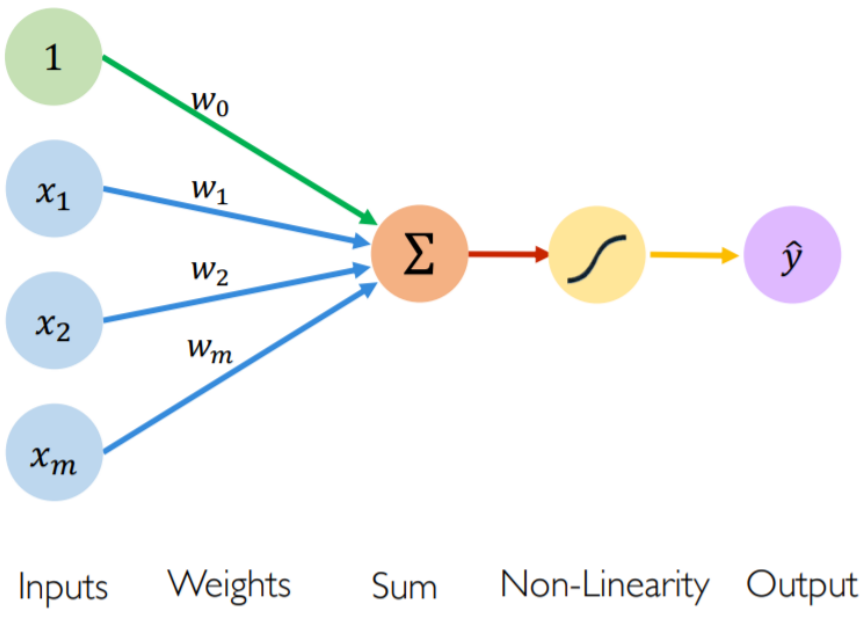

多层神经网络
两层神经网络
单层神经网络
神经元
2012
CNN
第三次
兴起
2020
Winter?
1986
BP
第二次
兴起
2006
DBN
1982
Hopfield
1995
SVM
1958
Perceptron
第一次
兴起
1969
"AI Winter"
1943
MP
诞生
1949
Hebb
1940
1950
1960
1970
1980
1990
2000
2010
2020

Big Guys


COMPANIES & PEOPLE
Taken in NIPS2014, from left: Yann LeCun (Facbook, NYU), Geoffrey Hinton (Google, U of Toronto), Yoshua Bengio (U of Montreal), Andrew Ng (Baidu)
Google
facebook
Google DeepMind
Microsoft
Baidu 百度
Alibaba Group 阿里巴巴集团
Disney Research
TWO SIGMA
amazon.com
NVIDIA
DiDi
BIDR
北京大数据研究院

Outlines

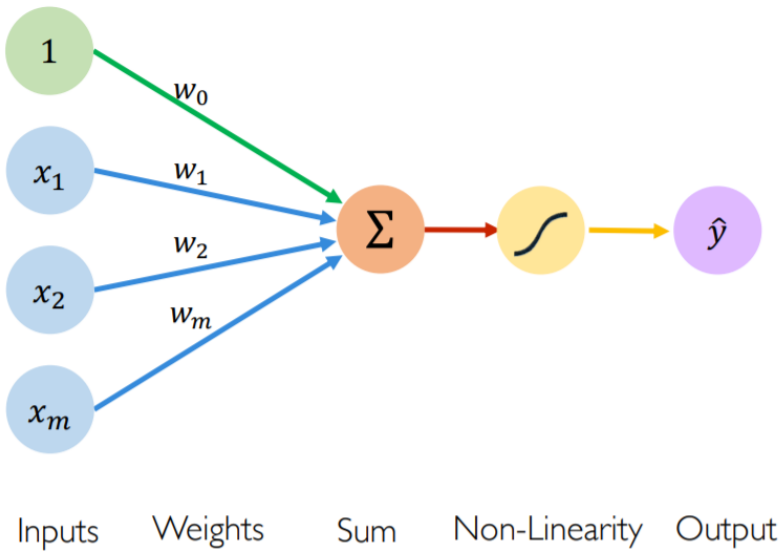
The Perceptron : Forward Propagation



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right),$$

- $\hat{y}$ is the Output,
- g is a Non-linear activation function,
- w_0 is the Bias.

The Perceptron : Forward Propagation

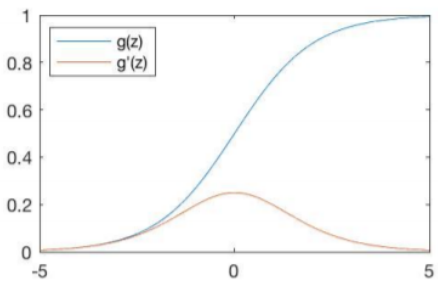


$$\hat{y} = g\left(w_0 + \mathbf{X}^T\mathbf{W}\right), \textit{where}$$

$$\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}.$$

Common Activation Functions

Sigmoid Function

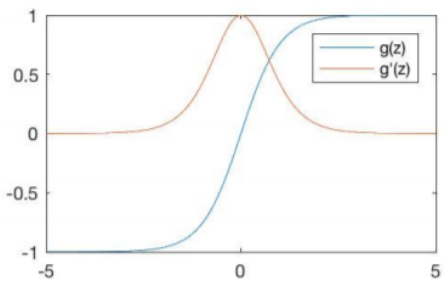


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

tf.nn.sigmoid(z)

Hyperbolic Tangent

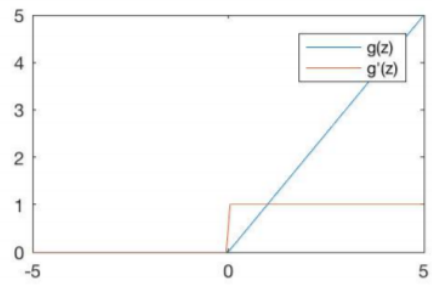


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

tf.nn.tanh(z)

Rectified Linear Unit (ReLU)



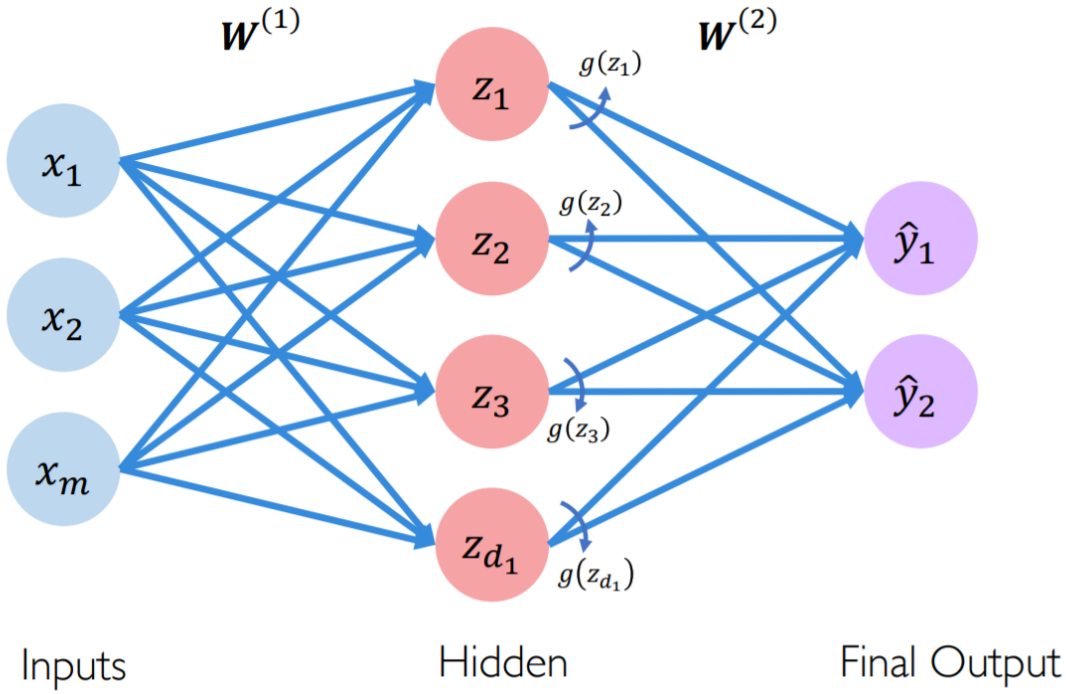
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

tf.nn.relu(z)

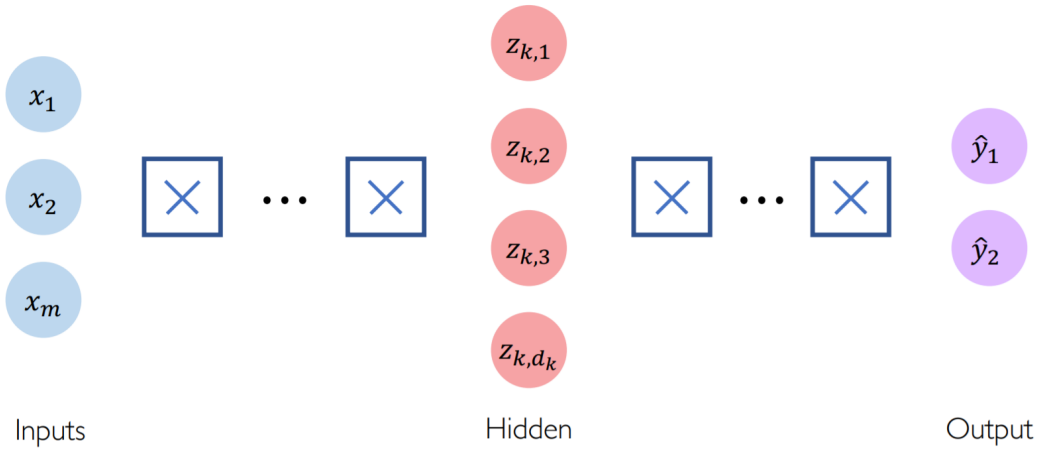
Note all activation functions are nonlinear.

Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)}, \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)}\right).$$

Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g\left(z_{k-1,j}\right) w_{j,i}^{(k)}.$$

Theorem (Universal approximation theorem (Cybenko 1980, 1989))

1. *Any function can be approximated by a three-layer neural network within sufficiently high accuracy.*
2. *Any bounded continuous function can be approximated by a two-layer neural network within sufficiently high accuracy.*

Loss Optimization

We want to find the network weights that achieve the lowest loss

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \mathbf{W}\right), y^{(i)}\right),$$

where $\mathcal{L}\left(f\left(x^{(i)}; \mathbf{W}\right), y^{(i)}\right)$ is the loss function we defined according to the specific problem to measure the differences between output state $f\left(x^{(i)}; \mathbf{W}\right)$ and reference state $y^{(i)}$. It also can be written as

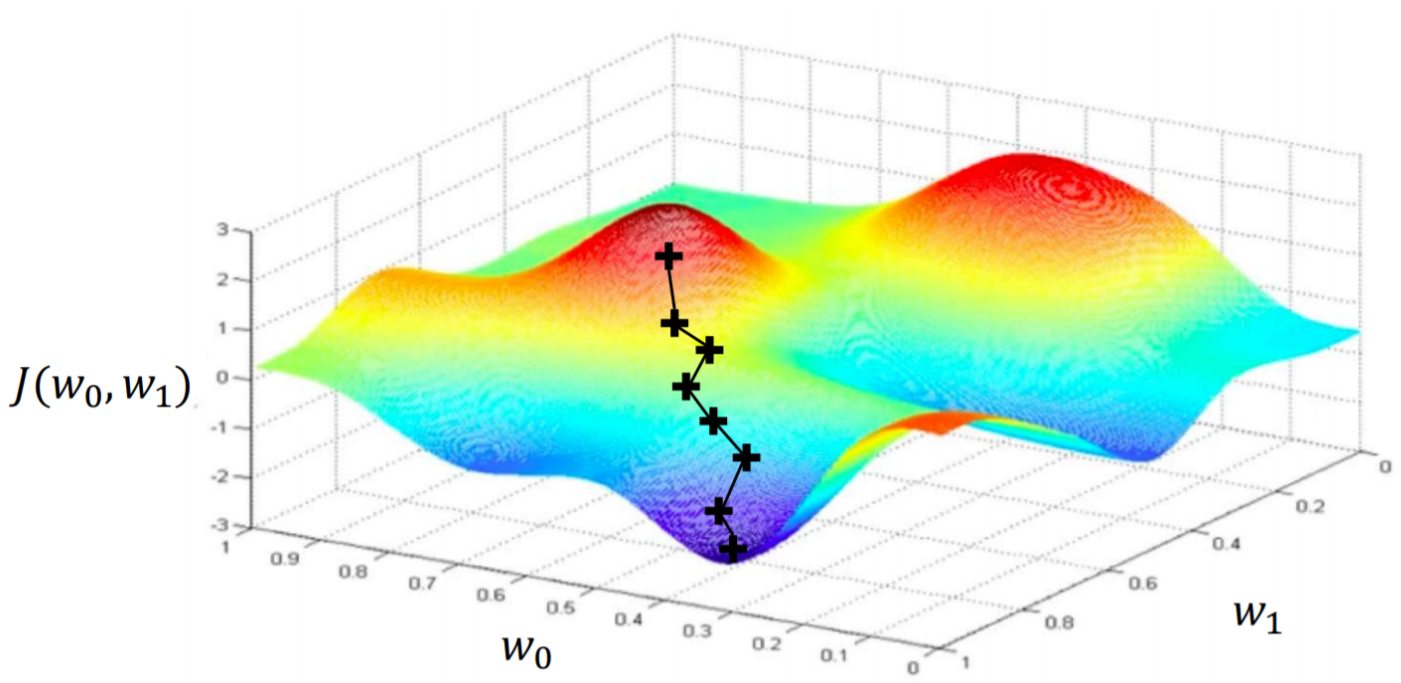
$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} C(\mathbf{W}).$$

Remember

$$\mathbf{W} = \left\{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots\right\}.$$

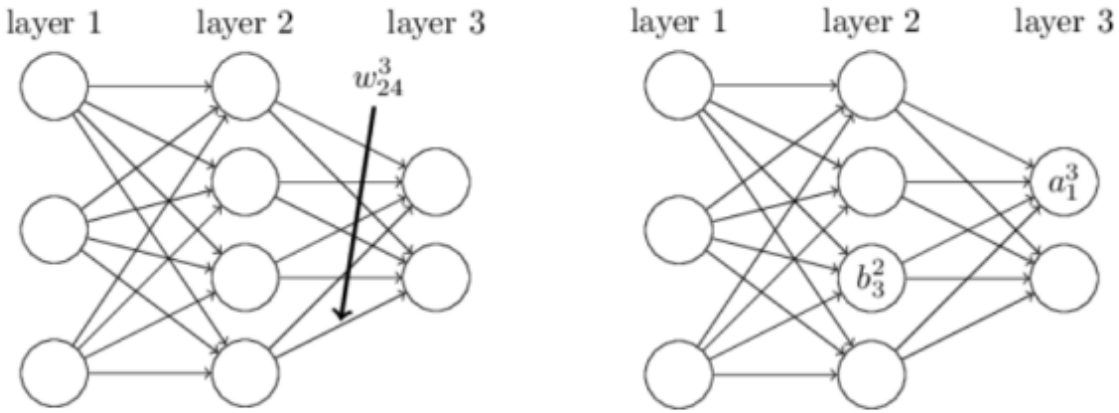
Gradient Decent

We can use Gradient Decent algorithm to find the optimal parameter **W**.



Note that we should calculate $\frac{\partial C}{\partial \mathbf{W}}$ to update **W**.

Notations



- w_{jk}^l is the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.
- for brevity $b_j^l = w_{j0}^l$ is the bias of the j^{th} neuron in the l^{th} layer.
- a_j^l for the activation of the j^{th} neuron in the l^{th} layer z_j^l.

$$a_j^l = g(z_j^l) = g\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

Four fundamental equations

We first define the error δ_j^l of neuron j in layer by

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l},$$

and we give the four fundamental equations of back propagation :

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right) \qquad (BP1)$$

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \qquad (BP2)$$

$$\frac{\partial C}{\partial b_j^L} = \delta_j^l \qquad (BP3)$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \qquad (BP4)$$

An equation for the error in the output layer (BP1)

The components of δ^L are given by

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right) \qquad (BP1)$$

Démonstration.

$$\begin{aligned}
\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\
&= \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial \sigma\left(z_j^L\right)}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right)
\end{aligned}$$

□

An equation for the error in the hidden layer (BP2)

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \qquad (BP2)$$

Démonstration.

$$\begin{cases} \delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ z_k^{l+1} = \left(\sum_i w_{ki}^{l+1} a_i^l\right) + b_k^{l+1} = \left(\sum_i w_{ki}^{l+1} \sigma\left(z_i^l\right)\right) + b_k^{l+1} \end{cases}$$

$$\Rightarrow \begin{cases} \delta_j^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ \frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'\left(z_j^l\right) \end{cases} \Rightarrow \delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'\left(z_j^l\right) \qquad \square$$

The change of the cost with respect to any bias (BP3)

$$\frac{\partial C}{\partial b_j^L} = \delta_j^l \qquad (BP3)$$

Démonstration.

$$\begin{cases} \frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l}\frac{\partial z_k^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \\ z_j^l = \left(\sum_k w_{jk}^l d_k^{l-1}\right) + b_j^l \Rightarrow \frac{\partial z_k^l}{\partial b_j^l} = 1 \end{cases} \Rightarrow \frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot 1 = \delta_j^l. \quad \square$$

The change of the cost with respect to any weight (BP4)

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \qquad (BP4)$$

Démonstration.

$$\begin{cases} \frac{\partial C}{\partial w^l_{jk}} = \sum_i \frac{\partial C}{\partial z^l_i} \frac{\partial z^l_i}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{jk}} \\ z^l_j = \left(\sum_m w^l_{jm} a^{l-1}_m \right) + b^l_j \Rightarrow \frac{\partial z^l_j}{\partial w^l_{jk}} = a^{l-1}_k \end{cases}$$

$$\Rightarrow \frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{jk}} = \delta^l_j a^{l-1}_k$$

.

□

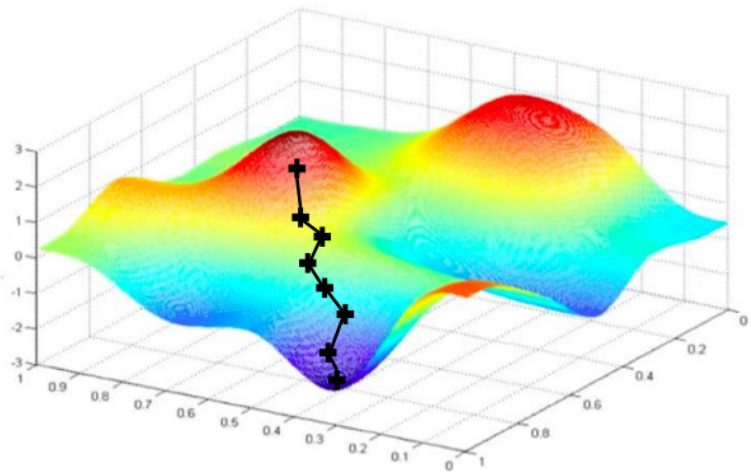
Back Propagation procedure

- 1. Input x : Set the corresponding activation a^1 for the input layer.
- 2. Feedforward : For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma\left(z^l\right)$.
- 3. Output error δ^L : Compute the vector $\delta^L = \nabla_a C \odot \sigma'\left(z^L\right)$.
- 4. Backpropagate the error : For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)$.
- 5. Output : The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

Gradient Descent

Algorithm

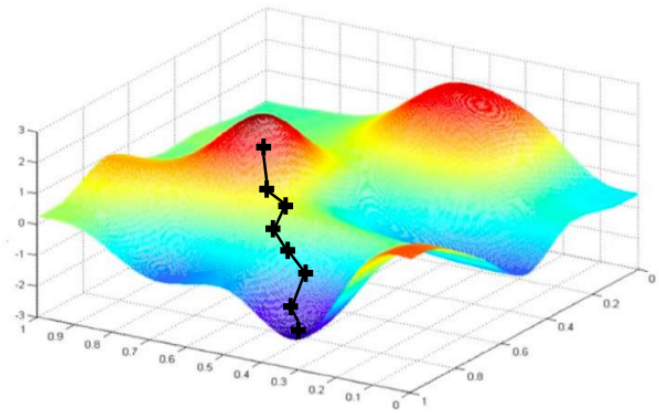
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

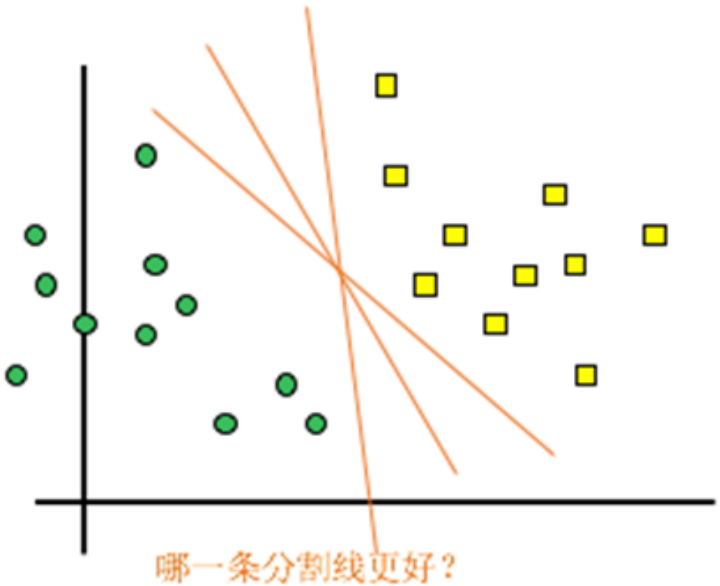


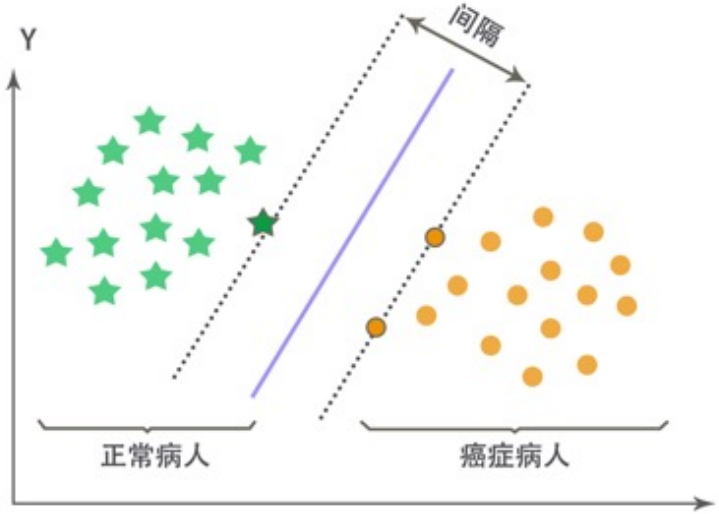
- Mini-batches lead to fast training!
- Can parallelize computation + achieve significant speed increases on GPUs.

Outlines

Support Vector Machine (SVM)

- Use hyperplane to separate data : maximize margin
- Can deal with low-dimensional data that are not linearly separated by using kernel functions
- Decision boundary only depends on some samples (support vectors)





Linear SVM

- Training data : $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $y_i \in \{-1, +1\}$
- Hyperplane : $S = \mathbf{w}^T\mathbf{x} + b$; decision function : $f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$

$$\left.\begin{array}{l} f(\mathbf{x}_i) > 0 \Leftrightarrow y_i = 1 \\ f(\mathbf{x}_i) < 0 \Leftrightarrow y_i = -1 \end{array}\right\} \Rightarrow y_i f(\mathbf{x}_i) > 0$$

- Geometric margin between a point and hyperplane : $r_i = \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|_2}$
- Margin between dataset and hyperplane : $\min_i r_i$
- Maximize margin : $\max_{\mathbf{w},b} \min_i \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|_2}$

Formulation as Constrained Optimization

- Without loss of generality, let $\min_i y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$ (multiply $\mathbf{w}$ and b by the same proper constant)
- Maximize margin is equivalent to

$$\max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|_2}, \quad \text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1, i = 1, \ldots, n$$

- Further reduce to

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2, \quad \text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1, i = 1, \ldots, n$$

- This is primal problem : quadratical programming with linear constraints, computational complexity is $O(p^3)$ where p is dimension

Method of Lagrange Multipliers

- Introduce $\alpha_i \geqslant 0$ as Lagrange multiplier of constraint $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1$
- Lagrange function :

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_{i=1}^{n} \alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

- Since

$$\max_{\alpha} L(\mathbf{w}, b, \alpha) = \begin{cases} \frac{1}{2}\|\mathbf{w}\|_2^2, & y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geqslant 0, \forall i \\ +\infty, & y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 < 0, \exists i \end{cases}$$

- Primal problem is equivalent to the minimax problem

$$\min_{\mathbf{w},b} \max_{\alpha} L(\mathbf{w}, b, \alpha)$$

Dual problem

- When slater condition is satisfied, $\min\max \Leftrightarrow \max\min$
- Dual problem : $\max_{\alpha} \min_{\mathbf{w},b} L(\mathbf{w}, b, \alpha)$
- Solve for inner minimization problem :

$$\nabla_{\mathbf{w}} L = 0 \Longrightarrow \mathbf{w}^* = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = 0 \Longrightarrow \sum_i \alpha_i y_i = 0$$

- Plug into L : $L(\mathbf{w}^*, b^*, \alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_i y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$
- Dual optimization :

$$\min_{\alpha} \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j) - \sum_i \alpha_i,$$

$$\text{s.t. } \alpha_i \geqslant 0, i = 1, \ldots, n, \sum_i \alpha_i y_i = 0$$

KKT conditions

- Three more conditions from the equivalence of primal and minimax problems

$$\begin{cases} \alpha_i^* \geqslant 0, \\ y_i((\mathbf{w}^*)^T \mathbf{x}_i + b^*) - 1 \geqslant 0, \\ \alpha_i^*[y_i((\mathbf{w}^*)^T \mathbf{x}_i + b^*) - 1] = 0. \end{cases}$$

- These together with two zero derivative conditions form KKT conditions
- $\alpha_i > 0 \Rightarrow y_i(\mathbf{w}^T \mathbf{x}_i + b^*) = 1$
- Index set of support vectors $S = \{i | \alpha_i > 0\}$
- $b = y_s - \mathbf{w}^T \mathbf{x}_s = y_s - \sum_{i \in S} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_s$
- More stable solution : $b = \frac{1}{|S|} \sum_{s \in S} \left(y_s - \sum_{i \in S} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_s \right)$

Sequential Minimal Optimization (SMO) Algorithm

- Invented by John C. Platt (1998)
- Coordinately optimize dual problem, select two variables and fix others, then dual problem reduces to one variable quadratic programming with positivity constraint
  1. Initially, choose α_i and α_j
  2. Fix other variables, solve for α_i and α_j
  3. Update α_i and α_j, redo step 1 iteratively
  4. Stop until convergence
- How to choose α_i and α_j ? choose the pair far from KKT conditions the most
- Computational complexity $O(n^3)$
- Easy to generalize to high dimensional problem with kernel functions

Soft Margin

- When data are not linear separable, introduce slack variables (tolerance control of fault) $\xi_i \geqslant 0$
- Relax constraint to $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1 - \xi_i$
- Primal problem :

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1 - \xi_i, \xi_i \geqslant 0, i = 1, \ldots, n$$

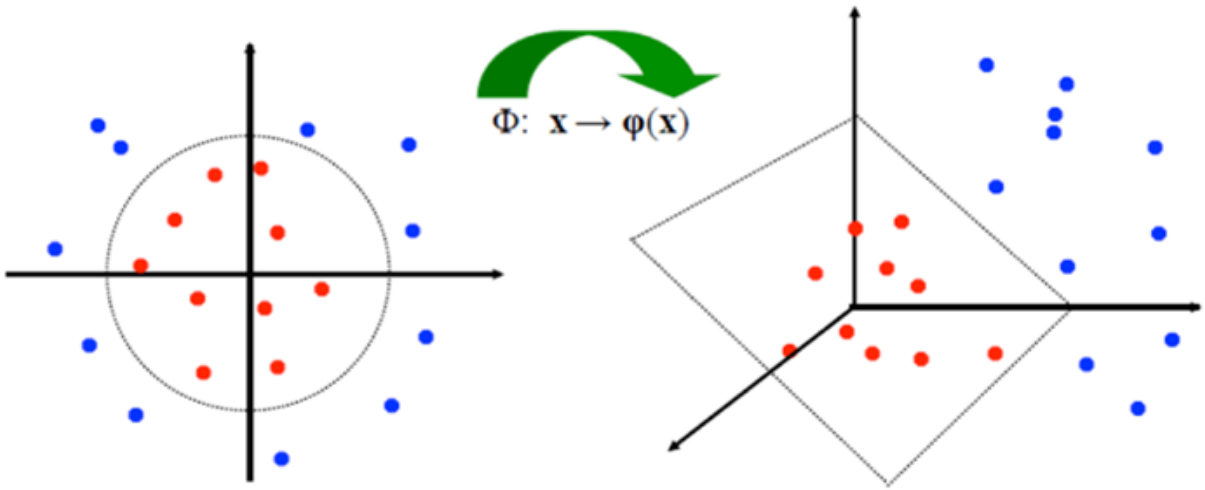
- Similar derivation to dual problem :

$$\min_{\alpha} \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j(\mathbf{x}_i^T\mathbf{x}_j) - \sum_i \alpha_i,$$

$$\text{s.t. } 0 \leqslant \alpha_i \leqslant C, i = 1, \ldots, n, \sum_i \alpha_i y_i = 0$$

Nonlinear SVM

- Nonlinear decision boundary could be mapped to linear boundary in high-dimensional space
- Modify objective function in dual problem :
  $\frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j (\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)) - \sum_i \alpha_i$
- Kernel function as inner product : $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$



Kernel Methods

- Reduce effect of curse of dimensionality
- Different kernels lead to different decision boundaries
- Popular kernels :

Kernel	Definition	Parameters
Polynomial	$(\mathbf{x}_1^T \mathbf{x}_2 + 1)^d$	d is positive integer
Gaussian	$e^{-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\delta^2}}$	$\delta > 0$
Laplacian	$e^{-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|}{\delta^2}}$	$\delta > 0$
Fisher	$\tanh(\beta \mathbf{x}_1^T \mathbf{x}_2 + \theta)$	$\beta > 0, \theta < 0$

Pros and Cons

- Where it is good
  - Applications in pattern recognition : text classification, face recognition
  - Easy to deal with high-dimensional data with kernels
  - Robust (only depends on support vectors), and easy to generalize to new dataset
- Disadvantage
  - Poor for ultra high dimensional data
  - Low computational efficiency for nonlinear SVM when sample size is large
  - Poor interpretability without probability

Outlines

References

- 数据分析导论，博雅大数据学院
- 周志华，机器学习，2016
- T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning : Data mining, Inference, and Prediction, 2nd Edition, 2009