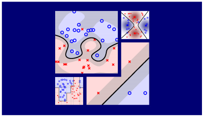
Machine Learning Techniques
(機器學習技法)



Lecture 12: Neural Network

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models

Lecture 11: Gradient Boosted Decision Tree

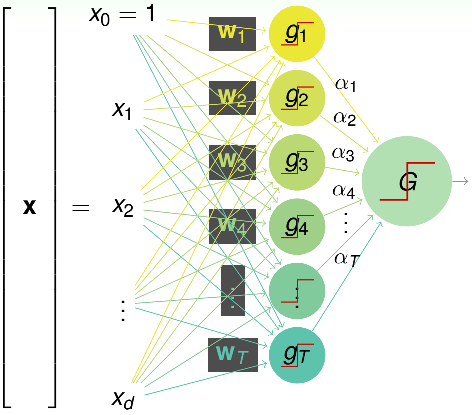
aggregating trees from **functional gradient** and **steepest descent** subject to **any error measure**

3. Distilling Implicit Features: Extraction Models

Lecture 12: Neural Network

- Motivation
- Neural Network Hypothesis
- Neural Network Learning
- Optimization and Regularization

Linear Aggregation of Perceptrons: Pictorial View

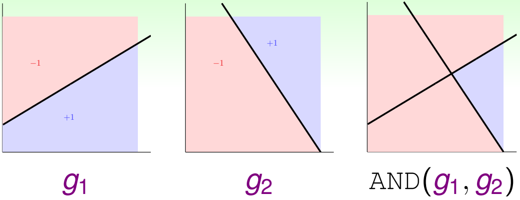


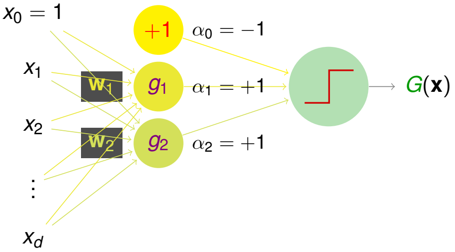
$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \underbrace{\text{sign}\left(\mathbf{w}_t^T \mathbf{x}\right)}_{g_t(\mathbf{x})}\right)$$

- two layers of weights: $\mathbf{w}_t$ and $\boldsymbol{\alpha}$
- two layers of sign functions: in g_t and in G

what boundary can G **implement**?

Logic Operations with Aggregation



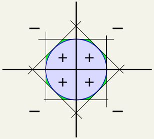


$G(\mathbf{x}) = \mathrm{sign}\left(-1 + g_1(\mathbf{x}) + g_2(\mathbf{x})\right)$

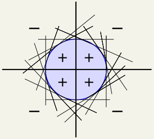
- $g_1(\mathbf{x}) = g_2(\mathbf{x}) = +1$ (TRUE): $G(\mathbf{x}) = +1$ (TRUE)
- otherwise: $G(\mathbf{x}) = -1$ (FALSE)
- $G \equiv \mathrm{AND}(g_1, g_2)$

OR, NOT can be **similarly implemented**

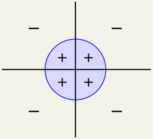
Powerfulness and Limitation



8 perceptrons

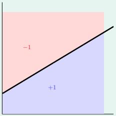


16 perceptrons

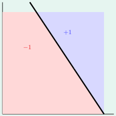


target boundary

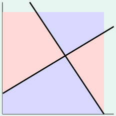
- 'convex set' hypotheses implemented: $d_{\text{VC}} \to \infty$, **remember? :-)**
- powerfulness: enough perceptrons $\approx$ **smooth boundary**



g_1



g_2



$\text{XOR}(g_1, g_2)$

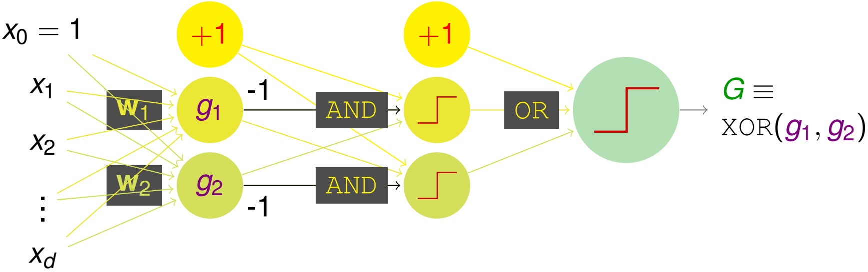
- limitation: XOR **not 'linear separable'** under $\phi(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}))$

how to implement $\text{XOR}(g_1, g_2)$?

Multi-Layer Perceptrons: Basic Neural Network

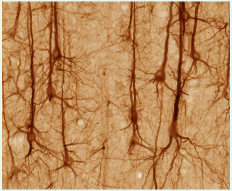
- non-separable data: can use more **transform**
- how about one more layer of **AND transform**?

$$\mathrm{XOR}(g_1, g_2) = \mathrm{OR}(\mathrm{AND}(-g_1, g_2), \mathrm{AND}(g_1, -g_2))$$



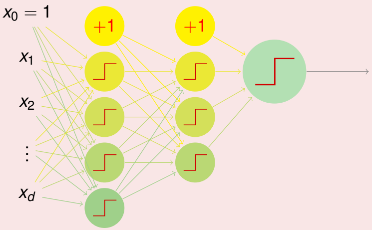
perceptron (simple)
$\Longrightarrow$ aggregation of perceptrons (powerful)
$\Longrightarrow$ **multi-layer perceptrons** (**more powerful**)

Connection to Biological Neurons



by UC Regents Davis campus-`brainmaps.org`.

Licensed under CC BY 3.0 via Wikimedia Commons



by Lauris Rubenis. Licensed under CC BY 2.0 via `https://flic.kr/p/fkVuZX`

by Pedro Ribeiro Simões. Licensed under CC BY 2.0 via `https://flic.kr/p/adiV7b`

neural network: bio-inspired model

Fun Time

Let $g_0(\mathbf{x}) = +1$. Which of the following $(\alpha_0, \alpha_1, \alpha_2)$ allows $G(\mathbf{x}) = \text{sign}\left(\sum_{t=0}^{2} \alpha_t g_t(\mathbf{x})\right)$ to implement $\mathrm{OR}(g_1, g_2)$?

1. $(-3, +1, +1)$
2. $(-1, +1, +1)$
3. $(+1, +1, +1)$
4. $(+3, +1, +1)$

Fun Time

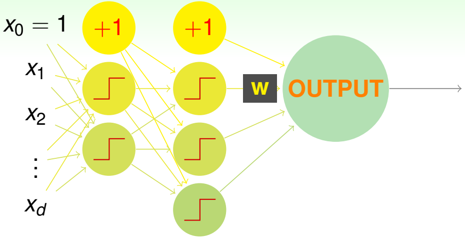
Let $g_0(\mathbf{x}) = +1$. Which of the following $(\alpha_0, \alpha_1, \alpha_2)$ allows $G(\mathbf{x}) = \text{sign}\left(\sum_{t=0}^{2} \alpha_t g_t(\mathbf{x})\right)$ to implement $\text{OR}(g_1, g_2)$?

1. $(-3, +1, +1)$
2. $(-1, +1, +1)$
3. $(+1, +1, +1)$
4. $(+3, +1, +1)$

Reference Answer: 3

You can easily verify with all four possibilities of $(g_1(\mathbf{x}), g_2(\mathbf{x}))$.

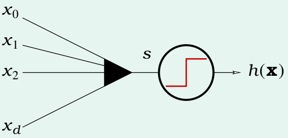
Neural Network Hypothesis: Output



- **OUTPUT**: simply a **linear model** with $s = \mathbf{w}^T \phi^{(2)}(\phi^{(1)}(\mathbf{x}))$
- any linear model can be used—**remember? :-)**

linear classification

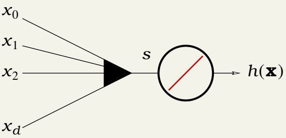
$h(\mathbf{x}) = \text{sign}(s)$



err = 0/1

linear regression

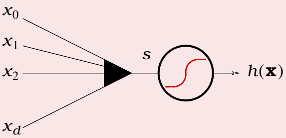
$h(\mathbf{x}) = s$



err = squared

logistic regression

$h(\mathbf{x}) = \theta(s)$

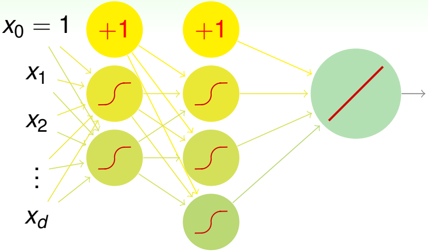


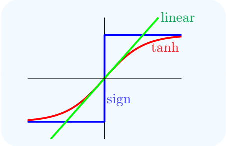
err = cross-entropy

will discuss **'regression' with squared error**

Neural Network Hypothesis: Transformation

- : **transformation** function of score (signal) s
- any transformation?
  - : whole network linear & thus **less useful**
  - : discrete & thus **hard to optimize** for $\mathbf{w}$
- popular choice of transformation: $= \tanh(s)$
  - 'analog' approximation of : **easier to optimize**
  - somewhat **closer to biological** neuron
  - **not that new! :-)**

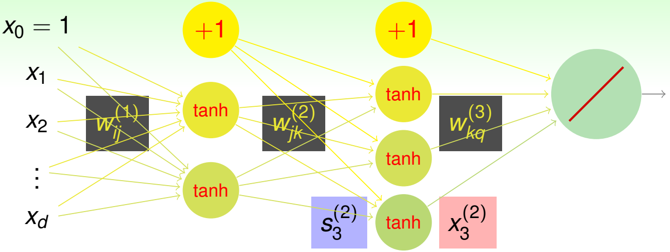




$$
\begin{aligned}
\tanh(s) &= \frac{\exp(s) - \exp(-s)}{\exp(s) + \exp(-s)} \\
&= 2\theta(2s) - 1
\end{aligned}
$$

will discuss with tanh as transformation function

Neural Network Hypothesis



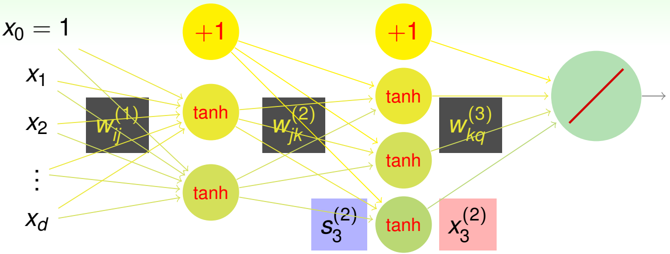
$d^{(0)}$-$d^{(1)}$-$d^{(2)}$-$\cdots$-$d^{(L)}$ Neural Network (NNet)

$$w_{ij}^{(\ell)}: \begin{cases} 1 \le \ell \le L & \text{layers} \\ 0 \le i \le d^{(\ell-1)} & \text{inputs} \\ 1 \le j \le d^{(\ell)} & \text{outputs} \end{cases}, \text{ score } s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} x_i^{(\ell-1)},$$

$$\text{transformed } x_j^{(\ell)} = \begin{cases} \tanh\left(s_j^{(\ell)}\right) & \text{if } \ell < L \\ s_j^{(\ell)} & \text{if } \ell = L \end{cases}$$

apply $\mathbf{x}$ as input layer $\mathbf{x}^{(0)}$, go through hidden layers to get $\mathbf{x}^{(\ell)}$, predict at output layer $x_1^{(L)}$

Physical Interpretation



- each layer: **transformation** to be **learned** from data
- $\phi^{(\ell)}(\mathbf{x}) = \tanh\left(\begin{bmatrix} \sum_{i=0}^{d^{(\ell-1)}} w_{i1}^{(\ell)} x_i^{(\ell-1)} \\ \vdots \end{bmatrix}\right)$

  —whether $\mathbf{x}$ 'matches' weight vectors in pattern

NNet: **pattern extraction** with
layers of connection weights

Fun Time

How many weights $\{w_{ij}^{(\ell)}\}$ are there in a 3-5-1 NNet?

1. 9
2. 15
3. 20
4. 26

Fun Time

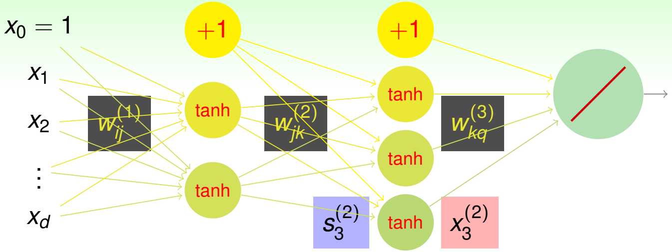
How many weights $\{w_{ij}^{(\ell)}\}$ are there in a 3-5-1 NNet?

1. 9
2. 15
3. 20
4. 26

Reference Answer: 4

There are $(3+1) \times 5$ weights in $w_{ij}^{(1)}$, and $(5+1) \times 1$ weights in $w_{jk}^{(2)}$.

How to Learn the Weights?



- goal: learning all $\{w_{ij}^{(\ell)}\}$ to **minimize** $E_{\text{in}}\left(\{w_{ij}^{(\ell)}\}\right)$
- one hidden layer: simply **aggregation of perceptrons** —**gradient boosting** to determine hidden neuron one by one
- multiple hidden layers? **not easy**
- let $e_n = (y_n - \text{NNet}(\mathbf{x}_n))^2$: can apply **(stochastic) GD** after computing $\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$!

next: efficient computation of $\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$

Computing $\frac{\partial e_n}{\partial w_{i1}^{(L)}}$ (Output Layer)

$$e_n = (y_n - \text{NNet}(\mathbf{x}_n))^2 = \left(y_n - s_1^{(L)}\right)^2 = \left(y_n - \sum_{i=0}^{d^{(L-1)}} w_{i1}^{(L)} x_i^{(L-1)}\right)^2$$

specially (output layer) $(0 \le i \le d^{(L-1)})$

$$\begin{aligned} & \frac{\partial e_n}{\partial w_{i1}^{(L)}} \\ = \; & \frac{\partial e_n}{\partial s_1^{(L)}} \cdot \frac{\partial s_1^{(L)}}{\partial w_{i1}^{(L)}} \\ = \; & -2\left(y_n - s_1^{(L)}\right) \cdot \left(x_i^{(L-1)}\right) \end{aligned}$$

generally $(1 \le \ell < L)$ $(0 \le i \le d^{(\ell-1)}; 1 \le j \le d^{(\ell)})$

$$\begin{aligned} & \frac{\partial e_n}{\partial w_{ij}^{(\ell)}} \\ = \; & \frac{\partial e_n}{\partial s_j^{(\ell)}} \cdot \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} \\ = \; & \delta_j^{(\ell)} \cdot \left(x_i^{(\ell-1)}\right) \end{aligned}$$

$\delta_1^{(L)} = -2\left(y_n - s_1^{(L)}\right)$, how about **others**?

Computing $\delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}}$

$$s_j^{(\ell)} \overset{\tanh}{\Longrightarrow} x_j^{(\ell)} \overset{w_{jk}^{(\ell+1)}}{\Longrightarrow} \begin{bmatrix} s_1^{(\ell+1)} \\ \vdots \\ s_k^{(\ell+1)} \\ \vdots \end{bmatrix} \Longrightarrow \cdots \Longrightarrow e_n$$

$$\begin{aligned} \delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}} &= \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} \\ &= \sum_k \left(\delta_k^{(\ell+1)}\right) \left(w_{jk}^{(\ell+1)}\right) \left(\tanh'\left(s_j^{(\ell)}\right)\right) \end{aligned}$$

$\delta_j^{(\ell)}$ can be computed **backwards** from $\delta_k^{(\ell+1)}$

Backpropagation (Backprop) Algorithm

Backprop on NNet

initialize all weights $w_{ij}^{(\ell)}$
for $t = 0, 1, \ldots, T$

1. stochastic: randomly pick $n \in \{1, 2, \cdots, N\}$
2. forward: compute all $x_i^{(\ell)}$ with $\mathbf{x}^{(0)} = \mathbf{x}_n$
3. backward: compute all $\delta_j^{(\ell)}$ subject to $\mathbf{x}^{(0)} = \mathbf{x}_n$
4. gradient descent: $w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} - \eta x_i^{(\ell-1)} \delta_j^{(\ell)}$

return $g_{\text{NNET}}(\mathbf{x}) = \left(\cdots \tanh \left(\sum_j w_{jk}^{(2)} \cdot \tanh \left(\sum_i w_{ij}^{(1)} x_i \right) \right) \right)$

sometimes 1 to 3 is (parallelly) done many times and average$(x_i^{(\ell-1)} \delta_j^{(\ell)})$ taken for update in 4, called **mini-batch**

basic NNet algorithm: backprop to compute the gradient **efficiently**

Fun Time

According to $\frac{\partial e_n}{\partial w_{i1}^{(L)}} = -2\left(y_n - s_1^{(L)}\right) \cdot \left(x_i^{(L-1)}\right)$ when would $\frac{\partial e_n}{\partial w_{i1}^{(L)}} = 0$?

1. $y_n = s_1^{(L)}$
2. $x_i^{(L-1)} = 0$
3. $s_i^{(L-1)} = 0$
4. all of the above

Fun Time

According to $\frac{\partial e_n}{\partial w_{i1}^{(L)}} = -2\left(y_n - s_1^{(L)}\right) \cdot \left(x_i^{(L-1)}\right)$ when would $\frac{\partial e_n}{\partial w_{i1}^{(L)}} = 0$?

1. $y_n = s_1^{(L)}$
2. $x_i^{(L-1)} = 0$
3. $s_i^{(L-1)} = 0$
4. all of the above

Reference Answer: 4

Note that $x_i^{(L-1)} = \tanh(s_i^{(L-1)}) = 0$ if and only if $s_i^{(L-1)} = 0$.

Neural Network Optimization

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \mathrm{err}\left(\left(\cdots \tanh\left(\sum_j w_{jk}^{(2)} \cdot \tanh\left(\sum_i w_{ij}^{(1)} x_{n,i}\right)\right)\right), y_n\right)$$

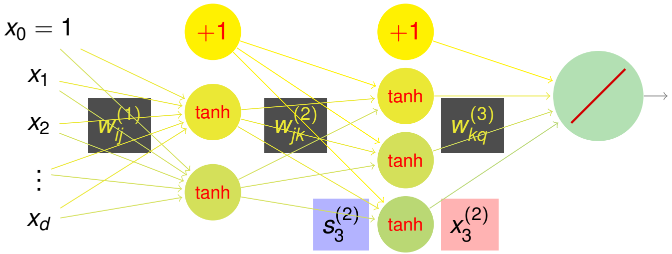
- generally **non-convex** when multiple hidden layers
  - not easy to reach **global minimum**
  - GD/SGD with **backprop** only gives **local minimum**
- different initial $w_{ij}^{(\ell)} \Longrightarrow$ different **local minimum**
  - somewhat '**sensitive**' to initial weights
  - **large weights** $\Longrightarrow$ **saturate** (small gradient)
  - advice: try **some random** & **small** ones

NNet: **difficult to optimize**,
but **practically works**

VC Dimension of Neural Network Model

roughly, with tanh-**like transfer functions**:

$d_{\mathrm{VC}} = O(VD)$ where V = # of neurons, D = # of weights



- pros: can **approximate 'anything'** if enough neurons (V large)
- cons: can **overfit** if too many neurons

NNet: **watch out for overfitting!**

Regularization for Neural Network

basic choice:

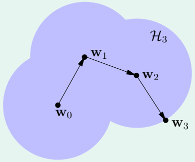
old friend weight-decay (L2) regularizer $\Omega(\mathbf{w}) = \sum \left(w_{ij}^{(\ell)}\right)^2$

- 'shrink' weights:
  large weight $\rightarrow$ large shrink; small weight $\rightarrow$ small shrink
- want $w_{ij}^{(\ell)} = 0$ (sparse) to effectively **decrease** d_{VC}
  - L1 regularizer: $\sum \left|w_{ij}^{(\ell)}\right|$, but **not differentiable**
  - weight-elimination ('scaled' L2) regularizer:
    large weight $\rightarrow$ median shrink; small weight $\rightarrow$ median shrink

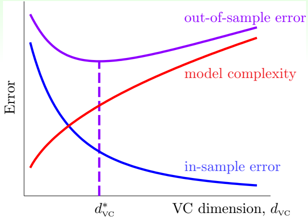
weight-elimination regularizer: $\sum \frac{\left(w_{ij}^{(\ell)}\right)^2}{1+\left(w_{ij}^{(\ell)}\right)^2}$

Yet Another Regularization: Early Stopping

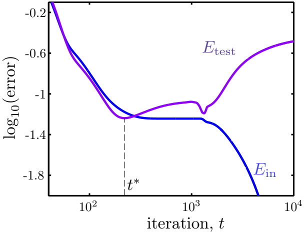
- **GD/SGD (backprop)** visits more weight combinations as t increases



- smaller t effectively decrease d_{VC}
- better 'stop in middle': **early stopping**



(**d^*_{VC} in middle, remember? :-)**)



when to stop? **validation!**

Fun Time

For the weight elimination regularizer $\sum \frac{\left(w_{ij}^{(\ell)}\right)^2}{1+\left(w_{ij}^{(\ell)}\right)^2}$, what is $\frac{\partial \text{regularizer}}{\partial w_{ij}^{(\ell)}}$?

1. $2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^1$
2. $2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^2$
3. $2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^3$
4. $2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^4$

Fun Time

For the weight elimination regularizer $\sum \frac{\left(w_{ij}^{(\ell)}\right)^2}{1+\left(w_{ij}^{(\ell)}\right)^2}$, what is $\frac{\partial \text{regularizer}}{\partial w_{ij}^{(\ell)}}$?

1. $2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^1$
2. $2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^2$
3. $2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^3$
4. $2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^4$

Reference Answer: 2

Too much calculus in this class, huh? :-)

Summary

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 12: Neural Network

- Motivation

  multi-layer for power with biological inspirations
- Neural Network Hypothesis

  layered pattern extraction until linear hypothesis
- Neural Network Learning

  backprop to compute gradient efficiently
- Optimization and Regularization

  tricks on initialization, regularizer, early stopping

- **next: making neural network 'deeper'**