Data Analytics
EEE 4774 & 6777

Module 4 - Classification

Perceptron, Multilayer Perceptron (MLP) / Artificial Neural Network

Spring 2022

$\nabla_{\vec{w}} E_P(w) = \begin{bmatrix} \frac{\partial E_P}{\partial w_1} \\ \vdots \\ \frac{\partial E_P}{\partial w_d} \end{bmatrix} = \begin{bmatrix} -\sum_{n\in\mathcal{M}} \phi_n(1) t_n \\ \vdots \\ -\sum_{n\in\mathcal{M}} \phi_n(d) t_n \end{bmatrix} = -\sum_{n\in\mathcal{M}} \vec{\phi}_n t_n$

Perceptron

(x_n, t_n) $\quad w^T \phi_n = \sum_{i=1}^{d} w_i \phi_n(i)$ $\quad \frac{\partial}{\partial w_i} w^T \phi_n = \phi_n(i)$

$x \in \mathbb{R}^d$ or $\phi(x) \in \mathbb{R}^d$

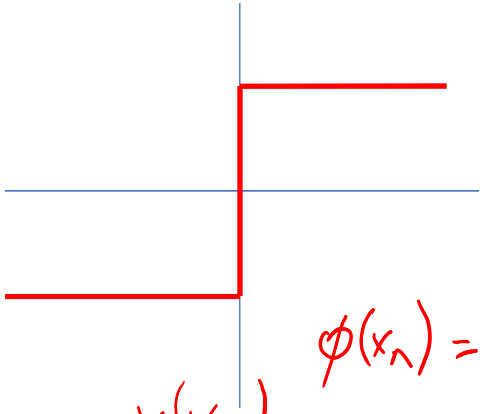
$w \in \mathbb{R}^d$

- 2-class model, $\quad t_n \in \{-1, +1\}$

true label (ground truth) $\quad w^T x$

$$y(\boldsymbol{x}) = f(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})) = \begin{cases} +1, & \boxed{\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})} \geq 0 \\ -1, & \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) < 0 \end{cases}$$

predicted label $\quad$ parameter vector



$\phi(x_n) = \phi_n$

misclassified instances $\quad y(x_n) = y_n$

Loss/ Cost/Error function: (***Perceptron criterion***) $\quad \boxed{E_P(\boldsymbol{w}) = -\sum_{n\in\mathcal{M}} \boldsymbol{w}^T \boldsymbol{\phi}_n t_n}$

set of misclassified instances

Correctly classified: $y_n t_n = 1 > 0$

Misclassified: $y_n t_n = -1 < 0$

$\mathcal{M}$: set of misclassified instances

$\nabla_w E_P(w) = -\sum_{n\in\mathcal{M}} \phi_n t_n \overset{\text{iteration } i}{\cong} -\phi_i t_i \;, i \in \mathcal{M}$

Stochastic gradient descent: $\quad \underbrace{\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} - \overset{\text{step size}}{\eta} \nabla E_P(\boldsymbol{w})}_{\text{Gradient descent}} \cong \underbrace{\boxed{\boldsymbol{w}^{(i)} + \eta \boldsymbol{\phi}_n t_n}}_{\text{Stochastic Gradient descent}}$

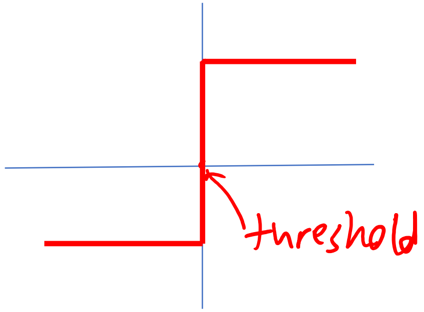
feature vector i.e. data point

- When $\boldsymbol{w}$ is multiplied by a constant, $y(\boldsymbol{x})$ does not change, hence w.l.o.g. $\underbrace{\eta}_{\text{learning rate}} = 1$

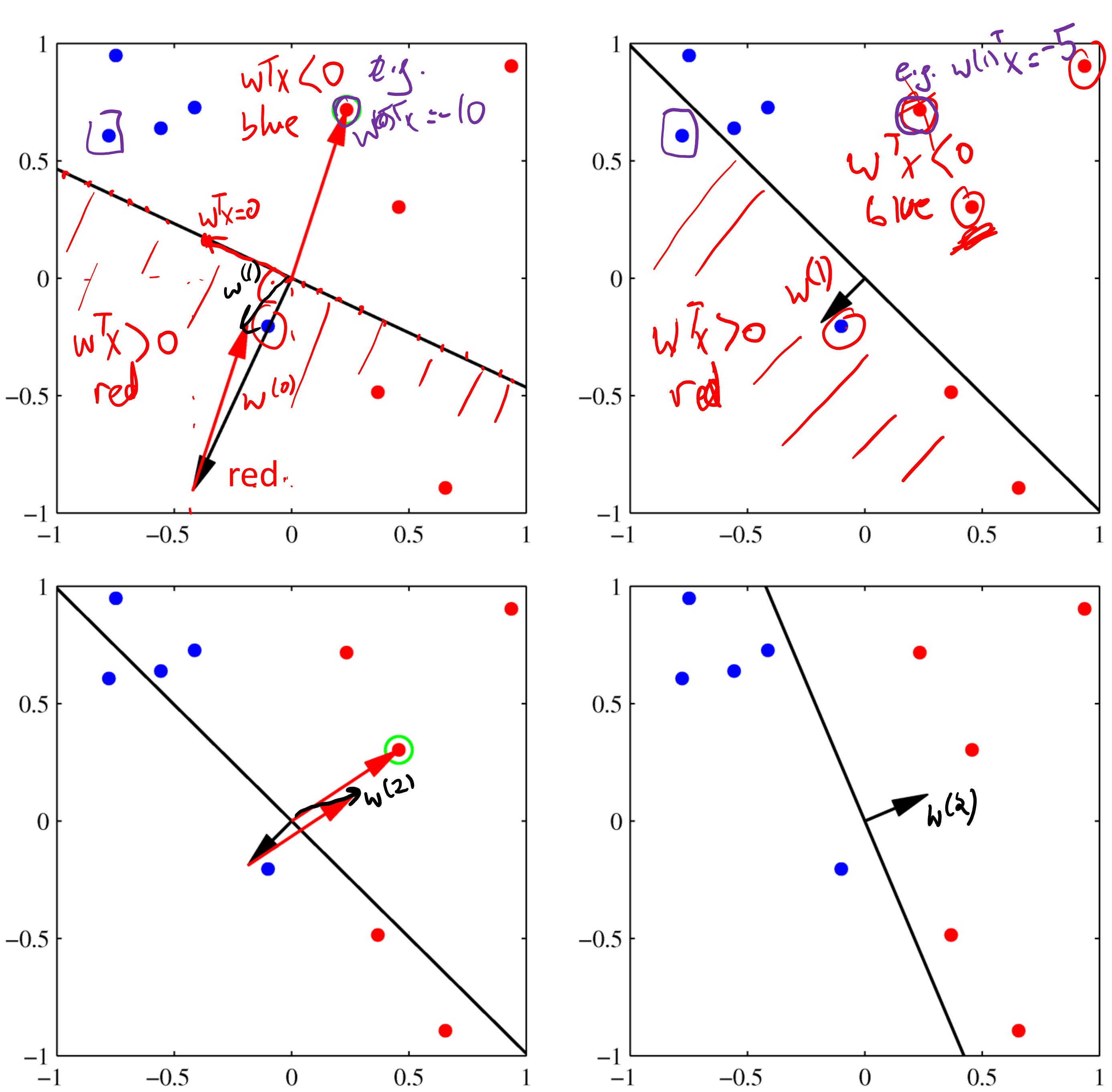
$t_n = \begin{cases} +1, & \text{red} \\ -1, & \text{blue} \end{cases}$

Perceptron procedure: For each instance $\boldsymbol{x_n}$

- Initialize w → e.g. $w^{(0)} = \begin{bmatrix} -0.4 \\ -0.8 \end{bmatrix}$
- Compute $f(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x_n}))$
- Decide



- If incorrectly classified
  - For class 1 (red), add $\boldsymbol{\phi}(\boldsymbol{x_n})$ to the $\boldsymbol{w}$ estimate
  - For class 2 (blue), subtract $\boldsymbol{\phi}(\boldsymbol{x_n})$ from the $\boldsymbol{w}$ estimate



- At each step error for the considered point decreases

$$-\underbrace{\boldsymbol{w}^{(i+1)T}}_{[w^{(i)}+\phi_n t_n]^T}(\phi_n t_n) = -\underbrace{\boldsymbol{w}^{(i)T}\phi_n t_n}_{\text{error at iter } i \text{ for nth pt.}} - \underbrace{(\phi_n t_n)^T\phi_n t_n}_{>0} < -\underbrace{\boldsymbol{w}^{(i)T}\phi_n t_n}_{\text{error at iter. } i+1 \text{ for nth pt.}}$$

- No guarantee that the total error decreases,
  e.g., a previously correctly classified point may be misclassified after the update

Perceptron convergence theorem:
If training data is linearly separable, the perceptron algorithm is guaranteed to find an exact solution in a finite number of steps

- Number of steps can be substantial
- Solution found depend on initialization (many solutions possible)
- Never converge if not linearly separable
- Not possible to distinguish between a linearly nonseparable problem and slow convergence
- Does not generalize readily to multi-class
- Fixed basis functions, not adaptive to the input

$x \in \mathbb{R}^{400 \times 1}$ $\phi(x)$ $\vec{w}$

Figure 4.8 Illustration of the Mark 1 perceptron hardware. The photograph on the left shows how the inputs were obtained using a simple camera system in which an input scene, in this case a printed character, was illuminated by powerful lights, and an image focussed onto a 20×20 array of cadmium sulphide photocells, giving a primitive 400 pixel image. The perceptron also had a patch board, shown in the middle photograph, which allowed different configurations of input features to be tried. Often these were wired up at random to demonstrate the ability of the perceptron to learn without the need for precise wiring, in contrast to a modern digital computer. The photograph on the right shows one of the racks of adaptive weights. Each weight was implemented using a rotary variable resistor, also called a potentiometer, driven by an electric motor thereby allowing the value of the weight to be adjusted automatically by the learning algorithm.

Artificial Neural Network (ANN) / Multilayer Perceptron (MLP)

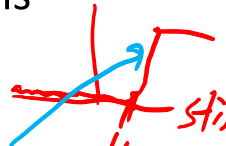
Biological Neural System

- Origins in attempts to find mathematical representations of information processing in biological systems

- Also known as ***Multilayer Perceptron*** / Can be also regarded as ***Multilayer Logistic Regression***

- Finds basis functions $\boldsymbol{\phi}(\boldsymbol{x})$ adaptive to the training data

Neuron output



Neuron firing

$\vec{w}^{(1)}_m = \begin{bmatrix} w^{(1)}_{m1} \\ w^{(1)}_{m2} \\ \vdots \\ w^{(1)}_{md} \end{bmatrix} \in \mathbb{R}^D$ $\quad h(\vec{w}^{(1)T}_m \vec{x} + b) = h(w^{(1)}_{m1} x_1 + \cdots + w^{(1)}_{mD} x_D + b)$

output (e.g. class label prediction)

$$y_k(\boldsymbol{x}) = f\left(\boldsymbol{w}_{2k}^T \boldsymbol{\phi}(\boldsymbol{x}) + b_{2k}\right) = \sigma\left(\boldsymbol{w}_k^{(2)\,T} \underbrace{\boldsymbol{\phi}(\boldsymbol{x})}_{M\times 1} + b_{2k}\right), k = 1, \dots, K$$

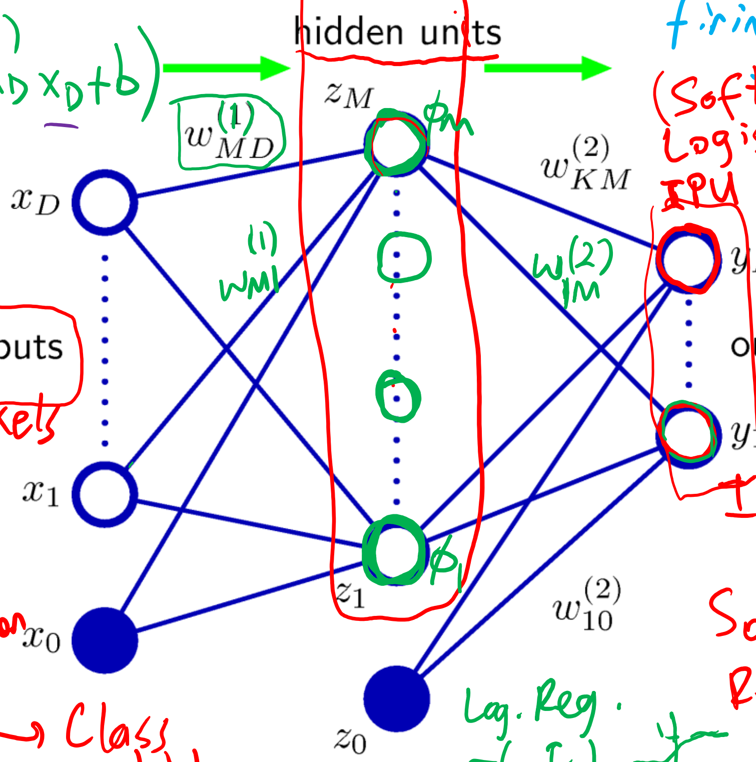
neuron index, layer index, offset parameter

$$\phi_m(\boldsymbol{x}) = h\left(\boldsymbol{w}_{1m}^T \boldsymbol{x} + b_{1m}\right) = \sigma\left(\boldsymbol{w}_m^{(1)\,T} \boldsymbol{x} + b_{1m}\right), \ m = 1, \dots, M$$

nonlinear activation func. — parameter vector — input vector — slice 1 … slice K

tanh (hyperbolic tangent), another sigmoid function, ReLU, GELU, etc. can be used instead of logistic sigmoid function.

hidden layer



e.g. image pixels

(Softmax) Logistic Reg. IPU for class K

class label

$y_1 = f(w_1^{(2)} \ldots$

IPU for class 1

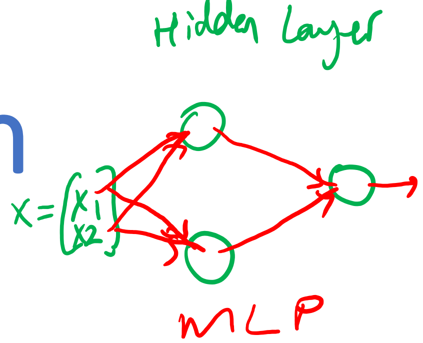
Perceptron : One info. processing unit (IPU)

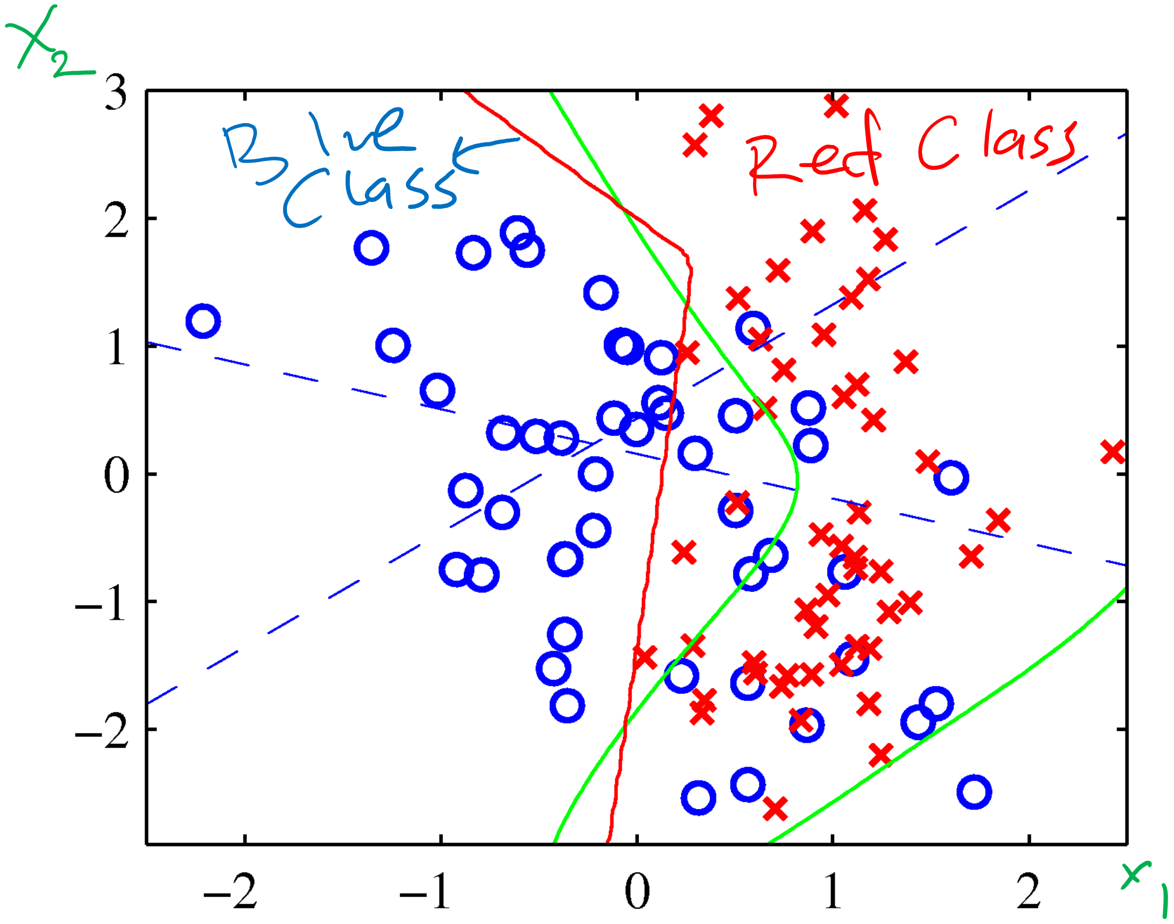
Perceptron $\vec{w}$: $\phi(x) \rightarrow$ [IPU] $\rightarrow$ Class prediction

Log. Reg. $\sigma(w^T x)$

Softmax Regression

Neural Network for Classification

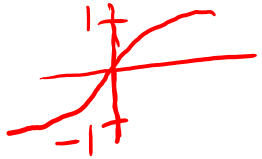




$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Two hidden units with activation functions

$$h(\boldsymbol{w}_{1i}^T\boldsymbol{x}) = \tanh(\boldsymbol{w}_{1i}^T\boldsymbol{x})$$

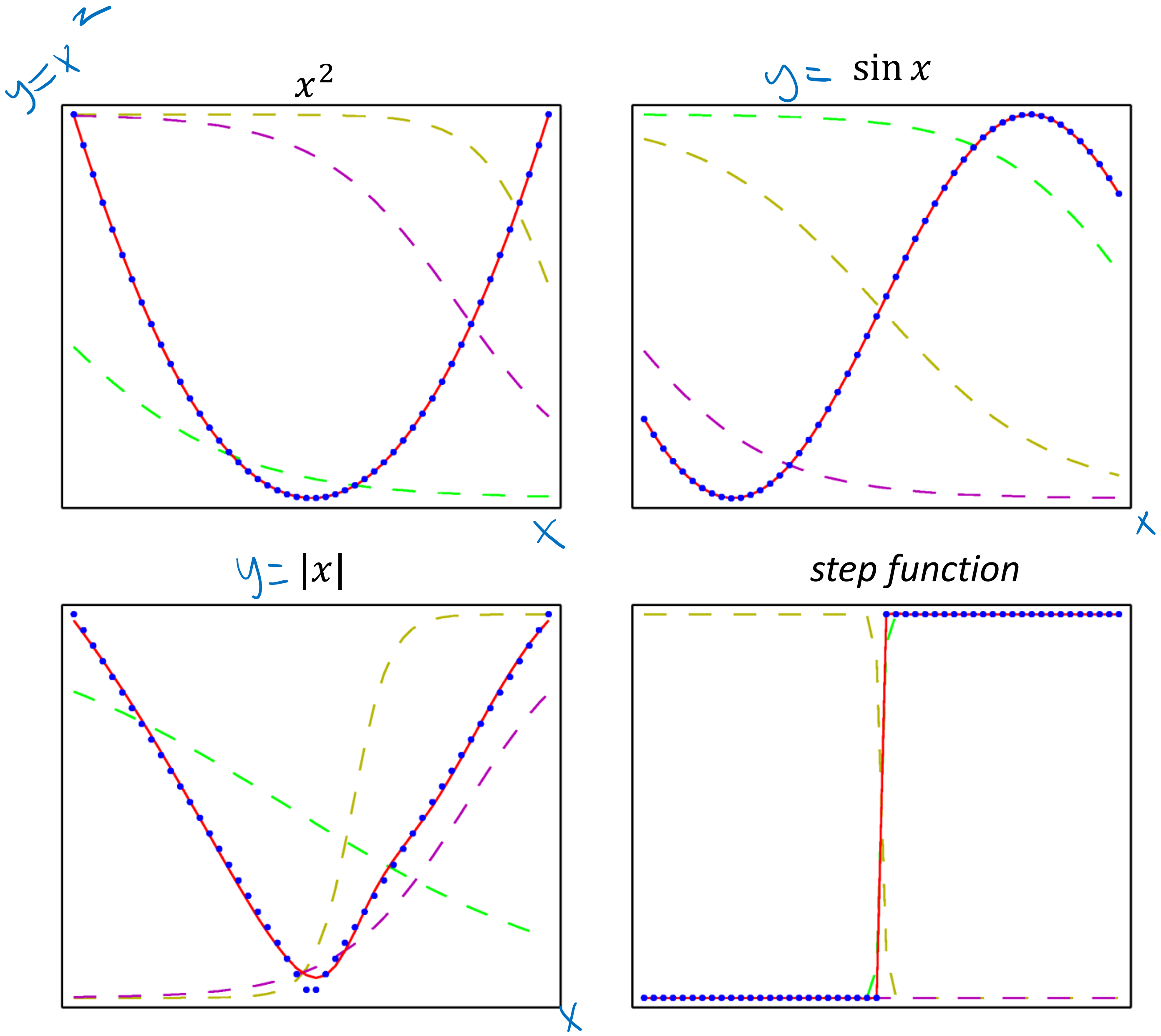


- Single output with activation function

$$f\left(\boldsymbol{w}_2^T\boldsymbol{h}(\boldsymbol{x})\right) = \sigma\left(\boldsymbol{w}_2^T\boldsymbol{h}(\boldsymbol{x})\right)$$

- Blue dashed lines show $h = 0$ for hidden units
- Red line shows $f = 0.5$ decision boundary
- Green line shows the optimum decision boundary

Neural Network for Regression



- $N = 50$ data points
- 2-layer network having 3 hidden units with $tanh$ activation function

1 hidden layer MLP

- Single output unit with linear activation function
- Dashed curves show hidden units
- Red curves show output functions

Adjust parameters $\{w_1^{(1)}, w_2^{(1)}, w_3^{(1)}, \vec{w}^{(2)}, b_1^{(1)}, b_2^{(1)}, b_3^{(1)}, b^{(2)}\}$ in training

Artificial Neural Network

An ANN is typically defined by three types of parameters:

σ, tanh, ReLU, ...

Fixed (Not updated in training)

1. $\boldsymbol{f}_j$: The activation function that converts a neuron's weighted input to its output activation.

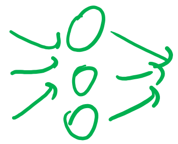
2. The interconnection pattern between the different layers of neurons

Fully connected (MLP), Convolutional (CNN)

3. $\boldsymbol{w}_j, b_j$: The weights of the interconnections and offset, which are updated in the learning process.

Supervised Learning:

Training

x_i  $y_i = P(\text{class } 1 \mid x_i) = \sigma$

$$w^T x + b = (w_1 \cdots w_D)\begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix} + b = \sum_{i=1}^{D} w_i x_i + b \cdot 1 = \sum_{i=0}^{D} \tilde{w}_i \tilde{x}_i$$

$$\tilde{w} = \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_D \end{pmatrix} \quad \tilde{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{pmatrix}$$

a) Feed the input $\boldsymbol{x_i}$, and obtain the output y_i

b) Compare the output y_i with the ground truth t_i, resulting in the error $y_i - t_i$

c) Adjust the weights $\boldsymbol{w}_j$ to minimize a function of the error, e.g.,

$t_i \log y_i + (1 - t_i)\log(1 - y_i)$ for classification or $(y_i - t_i)^2$ for regression

cross entropy loss

squared error loss → min MSE

Learning ANN coefficients

Testing

$w_1^{(i)}, \ldots, w_D^{(i)}, b^{(i)}$

fixed after training

Use input and trained network params. to predict output

- Mean squared error (MSE): $$E(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}[\, t_i - y_i(\boldsymbol{w}, \boldsymbol{x}_i)\,]^2$$
- MAE, Huber loss, etc. Cross-Entropy
- **Gradient Descent** based on **Backpropagation**
  - Efficient technique for computing the gradient of error function $E(\boldsymbol{w})$.
  - Local message passing scheme in which information is sent alternately forward and backward through the network.
  - Given input x, the information flows forward to produce prediction y and scalar cost $E(\boldsymbol{w})$.
  - In computing the derivatives, errors are propagated backwards through the network using the chain rule of differentiation, hence the name.
  - Finally, an optimization algorithm such as Stochastic Gradient Descent (SGD), Adam, RMSProp, etc. updates the weights using the computed gradient.

$w_1, \ldots, w_D, b$

Backpropagation Algorithm

Error / Loss Func.

$$E(t, f_L(W_L f_{L-1}(W_{L-1} \ldots f_2(W_2 f_1(W_1 x)))))$$

$$\frac{dE}{dx} = \frac{dE}{da_L} \circ \frac{da_L}{dz_L} \frac{dz_L}{da_{L-1}} \circ \frac{da_{L-1}}{dz_{L-1}} \frac{dz_{L-1}}{da_{L-2}} \ldots \frac{da_1}{dz_1} \frac{dz_1}{dx}$$

$$\frac{dE}{da_L} \circ f_L' \, W_L \circ f_{L-1}' \, W_{L-1} \ldots f_1' \, W_1$$

$$\nabla_x E = \underbrace{W_1^T f_1' \ldots \circ \underbrace{W_{L-1}^T f_{L-1}' \circ W_L^T f_L' \circ \frac{dE}{da_L}}_{\delta_{L-1}}}_{\delta_1}$$

δ_{L-1}: error at level L-1

$$\nabla_{W_k} E = \delta_k a_{k-1}^T$$

$$\boxed{\delta_k = f_k' \circ W_{k+1}^T \delta_{k+1}}$$

Computing δ_k in terms δ_{k+1} avoids duplicate operations.