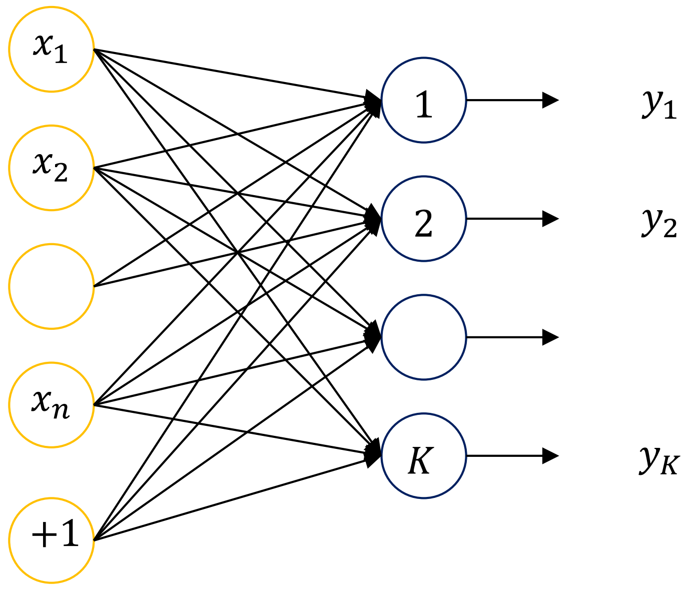
Chapter 3

Neuron Layers

Neural networks and deep learning

Weight matrix of a layer

Consider a layer of K neurons.



Let $\boldsymbol{w}_k$ and b_k denote the weight vector and bias of k th neuron. Weights connected to a neuron layer is given by a weight matrix:

$$\boldsymbol{W} = (\boldsymbol{w}_1 \quad \boldsymbol{w}_2 \quad \cdots \quad \boldsymbol{w}_K)$$

where columns are given by weight vectors of individual neurons.

And a bias vector $\boldsymbol{b}$ where each element corresponds to a bias of a neuron:

$$\boldsymbol{b} = (b_1, b_2, \cdots b_K)^T$$

Synaptic input at a layer for single input

Given an input pattern $\boldsymbol{x} \in \boldsymbol{R}^n$ to a layer of K neurons.

Synaptic input u_k to kth neuron:

$$u_k = \boldsymbol{w}_k{}^T \boldsymbol{x} + b_k$$

$\boldsymbol{w_k}$ and b_k denote the weight vector and bias of kth neuron.

Synaptic input vector $\boldsymbol{u}$ to the layer :

$$\boldsymbol{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_K \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_1^T \boldsymbol{x} + b_1 \\ \boldsymbol{w}_2^T \boldsymbol{x} + b_2 \\ \vdots \\ \boldsymbol{w}_K^T \boldsymbol{x} + b_k \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_K^T \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{pmatrix} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{u} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$$

where $\boldsymbol{W}$ is the weight matrix and $\boldsymbol{b}$ is the bias vector of the layer.

Synaptic input to a layer for batch input

Given a set $\{\boldsymbol{x}_p\}_{p=1}^{P}$ input patterns to a layer of K neurons where $\boldsymbol{x}_p \in \boldsymbol{R}^n$.

Synaptic input $\boldsymbol{u}_p$ to the layer for an input pattern $\boldsymbol{x}_p$:

$$\boldsymbol{u}_p = \boldsymbol{W}^T \boldsymbol{x}_p + \boldsymbol{b}$$

The synaptic input matrix $\boldsymbol{U}$ to the layer for P patterns:

$$(\boldsymbol{AB})^T = \boldsymbol{B}^T \boldsymbol{A}^T$$

$$\boldsymbol{U} = \begin{pmatrix} \boldsymbol{u}_1^T \\ \boldsymbol{u}_2^T \\ \vdots \\ \boldsymbol{u}_P^T \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_1{}^T \boldsymbol{W} + \boldsymbol{b}^T \\ \boldsymbol{x}_2{}^T \boldsymbol{W} + \boldsymbol{b}^T \\ \vdots \\ \boldsymbol{x}_P{}^T \boldsymbol{W} + \boldsymbol{b}^T \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_P^T \end{pmatrix} \boldsymbol{W} + \begin{pmatrix} \boldsymbol{b}^T \\ \boldsymbol{b}^T \\ \vdots \\ \boldsymbol{b}^T \end{pmatrix} = \boldsymbol{XW} + \boldsymbol{B}$$

where rows of $\boldsymbol{U}$ are synaptic inputs corresponding to individual input patterns.

The matrix $\boldsymbol{B} = \begin{pmatrix} \boldsymbol{b}^T \\ \boldsymbol{b}^T \\ \vdots \\ \boldsymbol{b}^T \end{pmatrix}$ has bias vector propagated as rows.

Activation at a layer for batch input

The synaptic input to the layer due to a batch of patterns:

$$\boldsymbol{U} = \boldsymbol{X}\,\boldsymbol{W} + \boldsymbol{B}$$

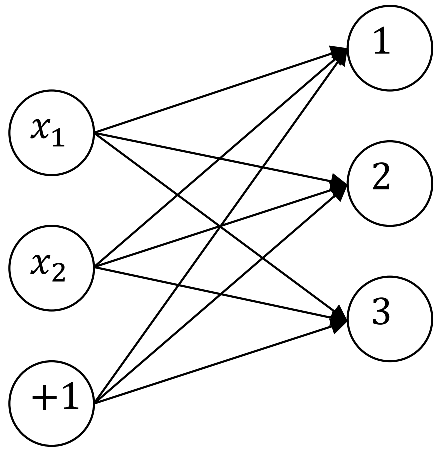
where rows of $\boldsymbol{U}$ corresponds to synaptic inputs of the layer, corresponding to individual input patterns:

Activation of the layer:

$$f(\boldsymbol{U}) = \begin{pmatrix} f(\boldsymbol{u}_1^T) \\ f(\boldsymbol{u}_2^T) \\ \vdots \\ f(\boldsymbol{u}_P^T) \end{pmatrix} = \begin{pmatrix} f(\boldsymbol{u}_1)^T \\ f(\boldsymbol{u}_2)^T \\ \vdots \\ f(\boldsymbol{u}_P)^T \end{pmatrix}$$

where activations due to individual patterns are written as rows.

Example 1: activations and outputs of a perceptron layer



A perceptron layer of 3 neurons shown in the figure receives 2-dimensional inputs $(x_1, x_2)^T$, and has a weight matrix $\boldsymbol{W}$ and a bias vector $\boldsymbol{b}$ given by

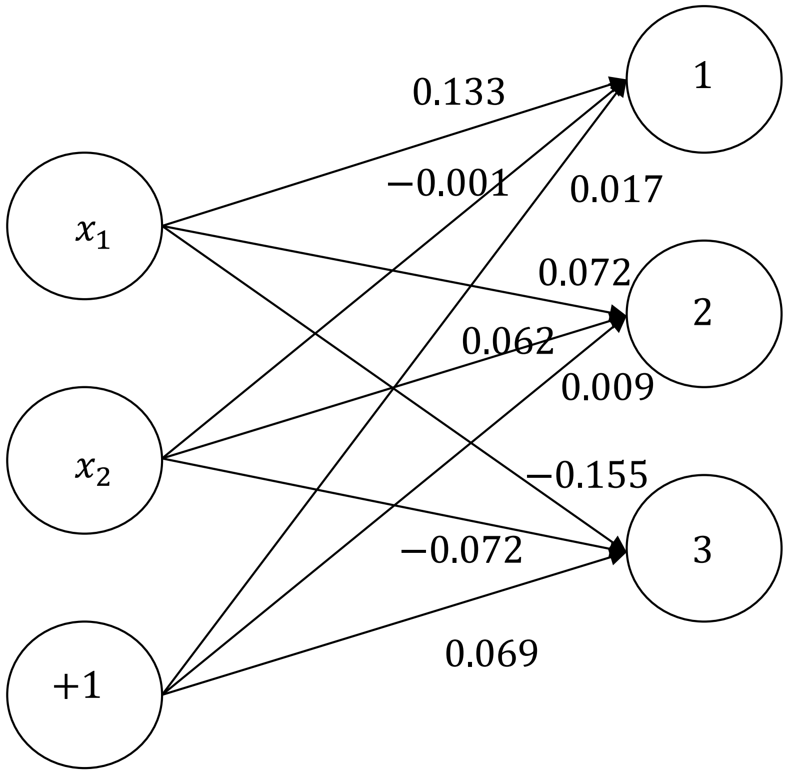
$$\boldsymbol{W} = \begin{pmatrix} 0.133 & 0.072 & -0.155 \\ -0.001 & 0.062 & -0.072 \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} 0.017 \\ 0.009 \\ 0.069 \end{pmatrix}$$

Using batch processing, find the output for input patterns:

$\begin{pmatrix} 0.5 \\ -1.66 \end{pmatrix}$, $\begin{pmatrix} -1.0 \\ -0.51 \end{pmatrix}$, $\begin{pmatrix} 0.78 \\ -0.65 \end{pmatrix}$, and $\begin{pmatrix} 0.04 \\ -0.2 \end{pmatrix}$.

Example 1

$$\boldsymbol{W} = \begin{pmatrix} 0.133 & 0.072 & -0.155 \\ -0.001 & 0.062 & -0.072 \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} 0.017 \\ 0.009 \\ 0.069 \end{pmatrix}.$$



Example 1

$$\boldsymbol{W} = \begin{pmatrix} 0.133 & 0.072 & -0.155 \\ -0.001 & 0.062 & -0.072 \end{pmatrix} \text{ and } \boldsymbol{B} = \begin{pmatrix} 0.017 & 0.009 & 0.069 \\ 0.017 & 0.009 & 0.069 \\ 0.017 & 0.009 & 0.069 \\ 0.017 & 0.009 & 0.069 \end{pmatrix}.$$

Input as a batch of four patterns:

$$\boldsymbol{X} = \begin{pmatrix} 0.5 & -1.66 \\ -1.0 & -0.51 \\ 0.78 & -0.65 \\ 0.04 & -0.2 \end{pmatrix}$$

The synaptic input to the layer:

$$\begin{aligned} \boldsymbol{U} &= \boldsymbol{XW} + \boldsymbol{B} \\ &= \begin{pmatrix} 0.5 & -1.66 \\ -1.0 & -0.51 \\ 0.78 & -0.65 \\ 0.04 & -0.2 \end{pmatrix} \begin{pmatrix} 0.133 & 0.072 & -0.155 \\ -0.001 & 0.062 & -0.072 \end{pmatrix} + \begin{pmatrix} 0.017 & 0.009 & 0.069 \\ 0.017 & 0.009 & 0.069 \\ 0.017 & 0.009 & 0.069 \\ 0.017 & 0.009 & 0.069 \end{pmatrix} \\ &= \begin{pmatrix} 0.085 & -0.059 & 0.111 \\ -0.115 & 0.094 & 0.26 \\ 0.121 & 0.024 & -0.005 \\ 0.022 & -0.001 & 0.077 \end{pmatrix} \end{aligned}$$

Example 1

$$\boldsymbol{U} = \begin{pmatrix} 0.085 & -0.059 & 0.111 \\ -0.115 & 0.094 & 0.26 \\ 0.121 & 0.024 & -0.005 \\ 0.022 & -0.001 & 0.077 \end{pmatrix}$$

For a perceptron layer

$$\boldsymbol{Y} = f(\boldsymbol{U}) = \frac{1}{1+e^{-\boldsymbol{U}}} = \begin{pmatrix} 0.521 & 0.485 & 0.527 \\ 0.471 & 0.476 & 0.565 \\ 0.530 & \bigcirc\!\!\!\!\!\!0.506 \xrightarrow{} & 0.499 \\ 0.506 & 0.500 & 0.519 \end{pmatrix}$$

2nd neuron output for 3rd input pattern

For example, third row corresponding to 3rd input:

$$\boldsymbol{x} = \begin{pmatrix} 0.78 \\ -0.65 \end{pmatrix}$$

And the corresponding output

$$\boldsymbol{y} = \begin{pmatrix} 0.530 \\ 0.506 \\ 0.499 \end{pmatrix}$$

SGD for single layer

Computational graph processing input $(\boldsymbol{x}, \boldsymbol{d})$:

$$\begin{tikzcd} \boldsymbol{x} \arrow[r, "\boldsymbol{W}{,}\,\boldsymbol{b}"] & \boldsymbol{u} \arrow[r, "f"] & \boldsymbol{y} \arrow[r] & J & \boldsymbol{d} \arrow[l] \end{tikzcd}$$

J denotes the cost function.

Need to compute gradients $\nabla_{\boldsymbol{W}} J$ and $\nabla_{\boldsymbol{b}} J$ to learn weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{b}$.

SGD for single layer

Consider k th neuron at the layer:

$$u_k = \boldsymbol{x}^T \boldsymbol{w}_k + b_k$$

And

$$\frac{\partial u_k}{\partial \boldsymbol{w}_k} = \boldsymbol{x}$$

The gradient of the cost with respect to the weight connected to kth neuron:

$$\nabla_{\boldsymbol{w}_k} J = \frac{\partial J}{\partial u_k} \frac{\partial u_k}{\partial \boldsymbol{w}_k} = \boldsymbol{x} \frac{\partial J}{\partial u_k} \qquad \text{(A)}$$

$$\nabla_{b_k} J = \frac{\partial J}{\partial u_k} \frac{\partial u_k}{\partial b_k} = \frac{\partial J}{\partial u_k} \qquad \text{(B)}$$

SGD for single layer

Gradient of J with respect to $\boldsymbol{W} = (\boldsymbol{w}_1 \quad \boldsymbol{w}_2 \quad \cdots \quad \boldsymbol{w}_K)$:

$$\begin{aligned}
\nabla_{\boldsymbol{W}} J &= (\nabla_{\boldsymbol{w}_1} J \quad \nabla_{\boldsymbol{w}_2} J \quad \cdots \quad \nabla_{\boldsymbol{w}_K} J) \\
&= \left(\boldsymbol{x}\frac{\partial J}{\partial u_1} \quad \boldsymbol{x}\frac{\partial J}{\partial u_2} \quad \cdots \quad \boldsymbol{x}\frac{\partial J}{\partial u_K}\right) \qquad \text{From (A)} \\
&= \boldsymbol{x}\left(\frac{\partial J}{\partial u_1} \quad \frac{\partial J}{\partial u_2} \quad \cdots \quad \frac{\partial J}{\partial u_K}\right) \\
&= \boldsymbol{x}(\nabla_{\boldsymbol{u}} J)^T
\end{aligned}$$

where

$$\nabla_{\boldsymbol{u}} J = \frac{\partial J}{\partial \boldsymbol{u}} = \begin{pmatrix} \frac{\partial J}{\partial u_1} \\ \frac{\partial J}{\partial u_2} \\ \vdots \\ \frac{\partial J}{\partial u_K} \end{pmatrix}$$

That is, $\nabla_{\boldsymbol{W}} J = \boldsymbol{x}(\nabla_{\boldsymbol{u}} J)^T$ (C)

SGD for single layer

Similarly, by substituting $\frac{\partial J}{\partial b_k} = \frac{\partial J}{\partial u_k}$ from (B):

$$\nabla_{\boldsymbol{b}} J = \begin{pmatrix} \frac{\partial J}{\partial b_1} \\ \frac{\partial J}{\partial b_2} \\ \vdots \\ \frac{\partial J}{\partial b_K} \end{pmatrix} = \begin{pmatrix} \frac{\partial J}{\partial u_1} \\ \frac{\partial J}{\partial u_2} \\ \vdots \\ \frac{\partial J}{\partial u_K} \end{pmatrix} = \nabla_{\boldsymbol{u}} J \qquad \text{(D)}$$

$$\nabla_{\boldsymbol{b}} J = \nabla_{\boldsymbol{u}} J$$

SGD for single layer

From (C) and (D),

$$\nabla_{\boldsymbol{W}} J = \boldsymbol{x}(\nabla_{\boldsymbol{u}} J)^T$$
$$\nabla_{\boldsymbol{b}} J = \nabla_{\boldsymbol{u}} J$$

That is, by computing gradient $\nabla_{\boldsymbol{u}} J$ with respect to synaptic input $\boldsymbol{u}$, the gradient of cost J with respect to the weights and biases is obtained.

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{x}(\nabla_{\boldsymbol{u}} J)^T$$
$$\boldsymbol{b} = \boldsymbol{b} - \alpha \nabla_{\boldsymbol{u}} J$$

GD for single layer

Given a set of patterns $\{(\boldsymbol{x}_p, \boldsymbol{d}_p)\}_{p=1}^{P}$ where $\boldsymbol{x}_p \in \boldsymbol{R}^n$ and $\boldsymbol{d}_p \in \boldsymbol{R}^K$ for regression and $d_p \in \{1, 2, \cdots K\}$ for classification.

The cost J is given by the sum of cost due to individual patterns:

$$J = \sum_{p=1}^{P} J_p$$

Where Then,

$$\nabla_{\boldsymbol{w}} J = \sum_{p=1}^{P} \nabla_{\boldsymbol{w}} J_p$$

GD for single layer

Substituting $\nabla_{\boldsymbol{W}} J_p = \boldsymbol{x}_p \left(\nabla_{\boldsymbol{u}_p} J_p\right)^T$ from (C) :

$$\nabla_{\boldsymbol{W}} J = \sum_{p=1}^{P} \boldsymbol{x}_p \left(\nabla_{\boldsymbol{u}_p} J_p\right)^T$$

$$= \textstyle\sum_{p=1}^{P} \boldsymbol{x}_p \left(\nabla_{\boldsymbol{u}_p} J\right)^T \qquad \text{since } \nabla_{\boldsymbol{u}_p} J = \nabla_{\boldsymbol{u}_p} J_p.$$

$$= \boldsymbol{x}_1 (\nabla_{\boldsymbol{u}_1} J)^T + \quad \boldsymbol{x}_2 (\nabla_{\boldsymbol{u}_2} J)^T \quad + \cdots \quad + \boldsymbol{x}_P (\nabla_{\boldsymbol{u}_P} J)^T$$

$$= \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_P \end{pmatrix} \begin{pmatrix} (\nabla_{\boldsymbol{u}_1} J)^T \\ (\nabla_{\boldsymbol{u}_2} J)^T \\ \vdots \\ (\nabla_{\boldsymbol{u}_P} J)^T \end{pmatrix}$$

$$= \boldsymbol{X}^T \, \nabla_{\boldsymbol{U}} J \qquad \text{(E)}$$

Note that $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1{}^T \\ \boldsymbol{x}_2{}^T \\ \vdots \\ \boldsymbol{x}_P{}^T \end{pmatrix}$ and $\boldsymbol{U} = \begin{pmatrix} \boldsymbol{u}_1{}^T \\ \boldsymbol{u}_2{}^T \\ \vdots \\ \boldsymbol{u}_P{}^T \end{pmatrix}$

GD for single layer

$$J = \sum_{p=1}^{P} J_p$$

$$\nabla_{\boldsymbol{b}} J = \sum_{p=1}^{P} \nabla_{\boldsymbol{b}} J_p$$

$$= \sum_{p=1}^{P} \nabla_{\boldsymbol{u}_p} J_p$$ Substituting from (D)

$$= \sum_{p=1}^{P} \nabla_{\boldsymbol{u}_p} J$$ Since $\nabla_{\boldsymbol{u}_p} J = \nabla_{\boldsymbol{u}_p} J_p$.

$$= \nabla_{\boldsymbol{u}_1} J + \nabla_{\boldsymbol{u}_2} J + \cdots + \nabla_{\boldsymbol{u}_P} J$$

$$= \begin{pmatrix} \nabla_{\boldsymbol{u}_1} J & \nabla_{\boldsymbol{u}_2} J & \cdots & \nabla_{\boldsymbol{u}_P} J \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

$$= (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P \qquad \text{(F)}$$

where $\mathbf{1}_P = (1,1,\cdots 1)^T$ is a vector of P ones.

GD for single layer

From (E) and (F):

$$\nabla_{\boldsymbol{W}} J = \boldsymbol{X}^T \nabla_{\boldsymbol{U}} J$$
$$\nabla_{\boldsymbol{b}} J = (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P$$

That is, by computing gradient $\nabla_{\boldsymbol{U}} J$ with respect to synaptic input, the weights and biases can be updated.

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{U}} J$$
$$\boldsymbol{b} = \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P$$

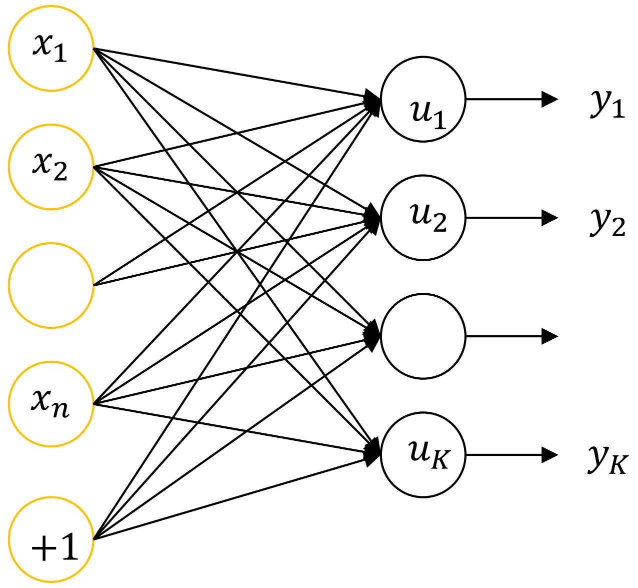
Learning a single layer

Learning a layer of neurons	
SGD	$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{x}(\nabla_{\boldsymbol{u}} J)^T$ $\boldsymbol{b} = \boldsymbol{b} - \alpha \nabla_{\boldsymbol{u}} J$
GD	$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{U}} J$ $\boldsymbol{b} = \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{U}} J)^T \boldsymbol{1}_P$

To learn a given layer, we need to compute
$\nabla_{\boldsymbol{u}} J$ for SGD and $\nabla_{\boldsymbol{U}} J$ for GD.

Those gradients with respect to synaptic inputs are dependent on the types of neurons in the layer.

Perceptron layer



$$y_k = f(u_k) = \frac{1}{1 + e^{-u_k}}$$

A layer of perceptrons performs **multidimensional non-linear regression** and learns a multidimensional non-linear mapping:

$$\phi : \boldsymbol{R}^n \rightarrow \boldsymbol{R}^K$$

SGD for perceptron layer

Given a training pattern $(\boldsymbol{x}, \boldsymbol{d})$
Note $\boldsymbol{x} = (x_1, x_2, \cdots x_n)^{\boldsymbol{T}} \in \boldsymbol{R}^n$ and $\boldsymbol{d} = (d_1, d_2, \cdots d_K)^{\boldsymbol{T}} \in \boldsymbol{R}^K$.

The square-error cost function:

$$J = \frac{1}{2}\sum_{k=1}^{K}(d_k - y_k)^2$$

where $y_k = f(u_k) = \frac{1}{1+e^{-u_k}}$ and $u_k = \boldsymbol{x}^T\boldsymbol{w}_k + b_k$.

Gradient of J with respect to u_k:

$$\frac{\partial J}{\partial u_k} = \frac{\partial J}{\partial y_k}\frac{\partial y_k}{\partial u_k} = -(d_k - y_k)\frac{\partial y_k}{\partial u_k} = -(d_k - y_k)f'(u_k) \qquad \text{(G)}$$

SGD for perceptron layer

Substituting$\nabla_{u_k} J = \frac{\partial J}{\partial u_k} = -(d_k - y_k) f'(u_k)$ from (G):

$$\nabla_{\boldsymbol{u}} J = \begin{pmatrix} \nabla_{u_1} J \\ \nabla_{u_2} J \\ \vdots \\ \nabla_{u_K} J \end{pmatrix} = -\begin{pmatrix} (d_1 - y_1) f'(u_1) \\ (d_2 - y_2) f'(u_2) \\ \vdots \\ (d_K - y_K) f'(u_K) \end{pmatrix} = -(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u}) \qquad \text{(H)}$$

and ‘$\cdot$’ denotes element-wise multiplication.

$$\nabla_{\boldsymbol{u}} J = -(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u})$$

SGD for perceptron layer

```
Given a training dataset $\{(\boldsymbol{x}, \boldsymbol{d})\}$
Set learning parameter α
Initialize $\boldsymbol{W}$ and $\boldsymbol{b}$
Repeat until convergence:
        For every pattern $(\boldsymbol{x}, \boldsymbol{d})$:
                $\boldsymbol{u} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$
                $\boldsymbol{y} = f(\boldsymbol{u}) = \frac{1}{1+e^{-\boldsymbol{u}}}$
                $\nabla_{\boldsymbol{u}} J = -(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u})$
                $\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{x} (\nabla_{\boldsymbol{u}} J)^T$
                $\boldsymbol{b} = \boldsymbol{b} - \alpha \nabla_{\boldsymbol{u}} J$
```

GD for perceptron layer

Given a training dataset $\{(\boldsymbol{x}_p, \boldsymbol{d}_p)\}_{p=1}^{P}$

Note $\boldsymbol{x}_p = (x_{p1}, x_{p2}, \cdots x_{pn})^T \in \boldsymbol{R}^n$ and $\boldsymbol{d}_p = (d_{p1}, d_{p2}, \cdots d_{pK})^{\boldsymbol{T}} \in \boldsymbol{R}^K$.

The cost function J is given by the sum of square errors (s.s.e.):

$$J = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}(d_{pk} - y_{pk})^2$$

J can be written as the sum of cost due to individual patterns:

$$J = \sum_{p=1}^{P} J_p$$

where $J_p = \frac{1}{2}\sum_{k=1}^{K}(d_{pk} - y_{pk})^2$ is the square error for the p th pattern.

GD for perceptron layer

$$\boldsymbol{U}=\begin{pmatrix}\boldsymbol{u}_1{}^T\\ \boldsymbol{u}_2{}^T\\ \vdots\\ \boldsymbol{u}_P{}^T\end{pmatrix}\rightarrow \nabla_{\boldsymbol{U}}J=\begin{pmatrix}(\nabla_{\boldsymbol{u}_1}J)^T\\ (\nabla_{\boldsymbol{u}_2}J)^T\\ \vdots\\ (\nabla_{\boldsymbol{u}_P}J)^T\end{pmatrix}=\begin{pmatrix}(\nabla_{\boldsymbol{u}_1}J_1)^T\\ (\nabla_{\boldsymbol{u}_2}J_2)^T\\ \vdots\\ (\nabla_{\boldsymbol{u}_P}J_P)^T\end{pmatrix}$$

From (H), substitute $\nabla_{\boldsymbol{u}}J=-(\boldsymbol{d}-\boldsymbol{y})\cdot f'(\boldsymbol{u})$:

$$\nabla_{\boldsymbol{U}}\,J=-\begin{pmatrix}\big((\boldsymbol{d}_1-\boldsymbol{y}_1)\cdot f'(\boldsymbol{u}_1)\big)^T\\ \big((\boldsymbol{d}_2-\boldsymbol{y}_2)\cdot f'(\boldsymbol{u}_2)\big)^T\\ \vdots\\ \big((\boldsymbol{d}_P-\boldsymbol{y}_P)\cdot f'(\boldsymbol{u}_P)\big)^T\end{pmatrix}=-\begin{pmatrix}\big(\boldsymbol{d}_1{}^T-\boldsymbol{y}_1{}^T\big)\cdot f'(\boldsymbol{u}_1{}^T)\\ \big(\boldsymbol{d}_2{}^T-\boldsymbol{y}_2{}^T\big)\cdot f'(\boldsymbol{u}_2{}^T)\\ \vdots\\ \big(\boldsymbol{d}_P{}^T-\boldsymbol{y}_P{}^T\big)\cdot f'(\boldsymbol{u}_P{}^T)\end{pmatrix}$$

$$\nabla_{\boldsymbol{U}}\,J=-(\boldsymbol{D}-\boldsymbol{Y})\cdot f'(\boldsymbol{U})$$

where $\boldsymbol{D}=\begin{pmatrix}\boldsymbol{d}_1^T\\ \boldsymbol{d}_2^T\\ \vdots\\ \boldsymbol{d}_P^T\end{pmatrix}$, $\boldsymbol{Y}=\begin{pmatrix}\boldsymbol{y}_1^T\\ \boldsymbol{y}_2^T\\ \vdots\\ \boldsymbol{y}_P^T\end{pmatrix}$, and $\boldsymbol{U}=\begin{pmatrix}\boldsymbol{u}_1^T\\ \boldsymbol{u}_2^T\\ \vdots\\ \boldsymbol{u}_P^T\end{pmatrix}$

GD for perceptron layer

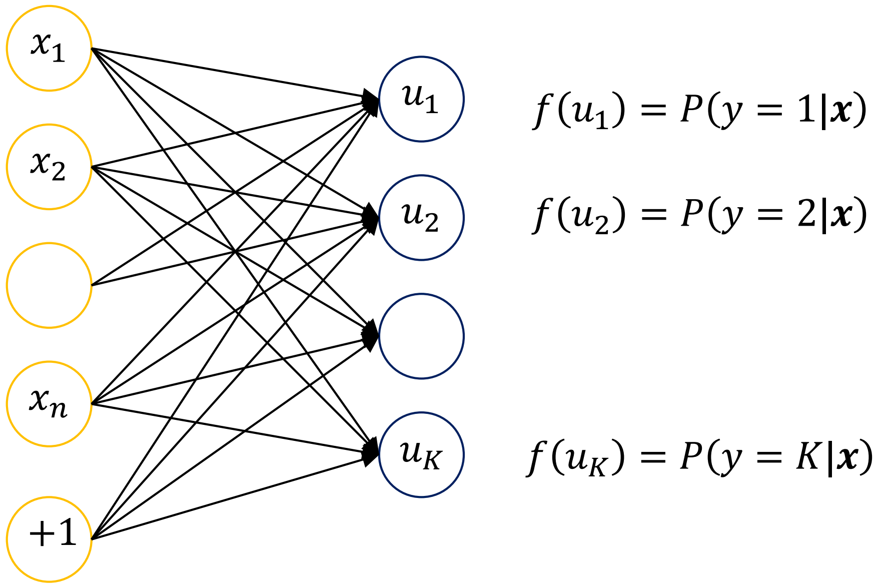
```
Given a training dataset $(\boldsymbol{X}, \boldsymbol{D})$
Set learning parameter $\alpha$
Initialize $\boldsymbol{W}$ and $\boldsymbol{b}$
Repeat until convergence:
        $\boldsymbol{U} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{B}$
        $\boldsymbol{Y} = f(\boldsymbol{U}) = \frac{1}{1+\boldsymbol{e}^{-\boldsymbol{U}}}$
        $\nabla_{\boldsymbol{U}} J = -(\boldsymbol{D} - \boldsymbol{Y}) \cdot f'(\boldsymbol{U})$
        $\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{U}} J$
        $\boldsymbol{b} = \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P$
```

Learning a perceptron layer

GD	SGD
$(\boldsymbol{X}, \boldsymbol{D})$	$(\boldsymbol{x}, \boldsymbol{d})$
$\boldsymbol{U} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{B}$	$\boldsymbol{u} = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}$
$\boldsymbol{Y} = f(\boldsymbol{U})$	$\boldsymbol{y} = f(\boldsymbol{u})$
$\nabla_U J = -(\boldsymbol{D} - \boldsymbol{Y}) \cdot f'(\boldsymbol{U})$	$\nabla_{\boldsymbol{u}} J = -(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u})$
$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_U J$	$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{x}(\nabla_{\boldsymbol{u}} J)^T$
$\boldsymbol{b} = \boldsymbol{b} - \alpha(\nabla_U J)^T \mathbf{1}_P$	$\boldsymbol{b} = \boldsymbol{b} - \alpha \nabla_{\boldsymbol{u}} J$

Softmax layer

Softmax layer is the extension of logistic regression to **multiclass classification** problem, which is also known as *multinomial logistic regression*.



x_1
x_2
x_n
$+1$
u_1
u_2
u_K
$f(u_1) = P(y = 1|\boldsymbol{x})$
$f(u_2) = P(y = 2|\boldsymbol{x})$
$f(u_K) = P(y = K|\boldsymbol{x})$

Each neuron in the softmax layer corresponds to one class label. The activation of a neuron gives the probability of the input belonging to that class label. Output is the class label having the maximum probability.

Softmax layer

The K neurons in the softmax layer performs K class classification and represent K classes.

The activation of each neuron k estimates the probability $P(y = k|x)$ that the input $\boldsymbol{x}$ belongs to the class k:

$$P(y = k|\boldsymbol{x}) = f(u_k) = \frac{e^{u_k}}{\sum_{k'=1}^{K} e^{u_{k'}}}$$

where $u_k = \boldsymbol{w}_k{}^T\boldsymbol{x} + b_k$, and $\boldsymbol{w}_k$ is weight vector and b_k is bias of neuron k.

The above activation function f is known as **softmax activation function**.

Softmax layer

The output y denotes the class label of the input pattern, which is given by

$$y = \underset{k}{\operatorname{argmax}}\, P(y = k|\boldsymbol{x}) = \underset{k}{\operatorname{argmax}}\, f(u_k)$$

That is, the class label is assigned to the class with the maximum activation.

SGD for softmax layer

Given a training pattern $(\boldsymbol{x}, d)$ where $\boldsymbol{x} \in \boldsymbol{R}^n$ and $d \in \{1,2, \cdots K\}$.

The cost function for learning is by the *multiclass cross-entropy:*

$$J = -\sum_{k=1}^{K} 1(d = k) log\big(f(u_k)\big)$$

where u_k is the synaptic input to the k the neuron.

The cost function can also be written as

$$J = -log\big(f(u_d)\big)$$

where d is the target label of input $\boldsymbol{x}$.

Note that the logarithm here is natural: $log = log_e$

SGD for softmax layer

$$J = -log\left(f(u_d)\right)$$

The gradient with respect to u_k is given by

$$\frac{\partial J}{\partial u_k} = -\frac{1}{f(u_d)}\frac{\partial f(u_d)}{\partial u_k} \qquad \text{(I)}$$

where

$$\frac{\partial f(u_d)}{\partial u_k} = \frac{\partial}{\partial u_k}\left(\frac{e^{u_d}}{\sum_{k'=1}^{K} e^{u_{k'}}}\right)$$

The above differentiation need to be considered separately for $k = d$ and for $k \neq d$.

SGD for softmax layer

If $k = d$:

$$\frac{\partial f(u_d)}{\partial u_k} = \frac{\partial}{\partial u_k}\left(\frac{e^{u_k}}{\sum_{k'=1}^{K} e^{u_{k'}}}\right)$$

$$= \frac{\left(\sum_{k'=1}^{K} e^{u_{k'}}\right) e^{u_k - e^{u_k} e^{u_k}}}{\left(\sum_{k'=1}^{K} e^{u_{k'}}\right)^2}$$

$$\frac{\partial\left(\sum_{k'=1}^{K} e^{u_{k'}}\right)}{\partial u_k} = e^{u_k}$$

$$= \frac{e^{u_k}}{\sum_{k'=1}^{K} e^{u_{k'}}}\left(1 - \frac{e^{u_k}}{\sum_{k'=1}^{K} e^{u_{k'}}}\right)$$

$$= f(u_k)\big(1 - f(u_k)\big)$$

$$= f(u_d)\big(1(k = d) - f(u_k)\big)$$

If $k \neq d$:

$$\frac{\partial f(u_d)}{\partial u_k} = \frac{\partial}{\partial u_k}\left(\frac{e^{u_d}}{\sum_{k'=1}^{K} e^{u_{k'}}}\right)$$

$$= -\frac{e^{u_d} e^{u_k}}{\left(\sum_{k'=1}^{K} e^{u_{k'}}\right)^2}$$

$$= -f(u_d) f(u_k)$$

$$= f(u_d)\big(1(k = d) - f(u_k)\big)$$

$$1(k = d) = 0$$

SGD for softmax layer

$$\frac{\partial f(u_d)}{\partial u_k} = f(u_d)\big(1(d=k) - f(u_k)\big)$$

Substituting in (I):

$$\nabla_{u_k} J = \frac{\partial J}{\partial u_k} = -\frac{1}{f(u_d)}\frac{\partial f(u_d)}{\partial u_k} = -\big(1(d=k) - f(u_k)\big)$$

Gradient J with respect to $\boldsymbol{u}$:

$$\nabla_{\boldsymbol{u}} J = \begin{pmatrix} \nabla_{u_1} J \\ \nabla_{u_2} J \\ \vdots \\ \nabla_{u_K} J \end{pmatrix} = -\begin{pmatrix} 1(d=1) - f(u_1) \\ 1(d=2) - f(u_2) \\ \vdots \\ 1(d=K) - f(u_K) \end{pmatrix} = -\big(1(\boldsymbol{k}=d) - f(\boldsymbol{u})\big) \qquad \text{(J)}$$

where $\boldsymbol{k} = (1 \quad 2 \quad \cdots \quad K)^T$

SGD for softmax layer

For a softmax layer:

$$\nabla_{\boldsymbol{u}} J = -\left(1(\boldsymbol{k} = d) - f(\boldsymbol{u})\right)$$

where:

$$1(\boldsymbol{k} = d) = \begin{pmatrix} 1(d = 1) \\ 1(d = 2) \\ \vdots \\ 1(d = K) \end{pmatrix} \text{ and } f(\boldsymbol{u}) = \begin{pmatrix} f(u_1) \\ f(u_2) \\ \vdots \\ f(u_K) \end{pmatrix}$$

Note that $1(\boldsymbol{k} = d)$ is a one-hot vector where the element corresponding to the target label d is '1' and elsewhere is '0'.

GD for softmax layer

Given a set of patterns $\{(\boldsymbol{x}_p, d_{\boldsymbol{p}})\}_{p=1}^{P}$ where $\boldsymbol{x}_p \in \boldsymbol{R}^n$ and $d_p \in \{1,2,\cdots K\}$.

The cost function of the *softmax layer* is given by the *multiclass cross-entropy*:

$$J = -\sum_{p=1}^{P}\left(\sum_{k=1}^{K} 1(d_p = k)\log\left(f(u_{pk})\right)\right)$$

where u_{pk} is the synaptic input to the k the neuron for input $\boldsymbol{x}_p$.

The cost function J can also be written as

$$J = -\sum_{p=1}^{P} log\left(f\left(u_{pd_p}\right)\right)$$

GD for softmax layer

J can be written as the sum of cost due to individual patterns:

$$J = \sum_{p=1}^{P} J_p$$

where $J_p = -log\left(f\left(u_{pd_p}\right)\right)$ is the cross-entropy for the p th pattern.

GD for softmax layer

$$\nabla_{\boldsymbol{U}} J = \begin{pmatrix} (\nabla_{\boldsymbol{u}_1} J)^T \\ (\nabla_{\boldsymbol{u}_2} J)^T \\ \vdots \\ (\nabla_{\boldsymbol{u}_P} J)^T \end{pmatrix} = \begin{pmatrix} (\nabla_{\boldsymbol{u}_1} J_1)^T \\ (\nabla_{\boldsymbol{u}_2} J_2)^T \\ \vdots \\ (\nabla_{\boldsymbol{u}_P} J_P)^T \end{pmatrix}$$

Substituting $\nabla_{\boldsymbol{u}} J = -\left(1(\boldsymbol{k} = d) - f(\boldsymbol{u})\right)$ from (J):

$$\nabla_{\boldsymbol{U}} J = -\begin{pmatrix} \left(1(\boldsymbol{k} = d_1) - f(\boldsymbol{u}_1)\right)^T \\ \left(1(\boldsymbol{k} = d_2) - f(\boldsymbol{u}_2)\right)^T \\ \vdots \\ \left(1(\boldsymbol{k} = d_P) - f(\boldsymbol{u}_P)\right)^T \end{pmatrix}$$

$$\nabla_{\boldsymbol{U}} J = -\left(\boldsymbol{K} - f(\boldsymbol{U})\right)$$

where $\boldsymbol{K} = \begin{pmatrix} 1(\boldsymbol{k} = d_1)^T \\ 1(\boldsymbol{k} = d_2)^T \\ \vdots \\ 1(\boldsymbol{k} = d_P)^T \end{pmatrix}$ is a matrix with every row is a one-hot vector.

Learning a softmax layer

GD	SGD
$(\boldsymbol{X}, \boldsymbol{D})$	$(\boldsymbol{x}, d)$
$\boldsymbol{U} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{B}$	$\boldsymbol{u} = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}$
$f(\boldsymbol{U}) = \frac{e^{\boldsymbol{U}}}{\sum_{k'=1}^{K} e^{\boldsymbol{U}_{k'}}}$	$f(\boldsymbol{u}) = \frac{e^{u_k}}{\sum_{k'=1}^{K} e^{u_{k'}}}$
$\boldsymbol{y} = \underset{k}{\operatorname{argmax}} f(\boldsymbol{U})$	$y = \underset{k}{\operatorname{argmax}} f(\boldsymbol{u})$
$\nabla_{\boldsymbol{U}} J = -\left(\boldsymbol{K} - f(\boldsymbol{U})\right)$	$\nabla_{\boldsymbol{u}} J = -\left(1(\boldsymbol{k} = d) - f(\boldsymbol{u})\right)$
$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{U}} J$	$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{x} (\nabla_{\boldsymbol{u}} J)^T$
$\boldsymbol{b} = \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{U}} J)^T \boldsymbol{1}_P$	$\boldsymbol{b} = \boldsymbol{b} - \alpha \nabla_{\boldsymbol{u}} J$

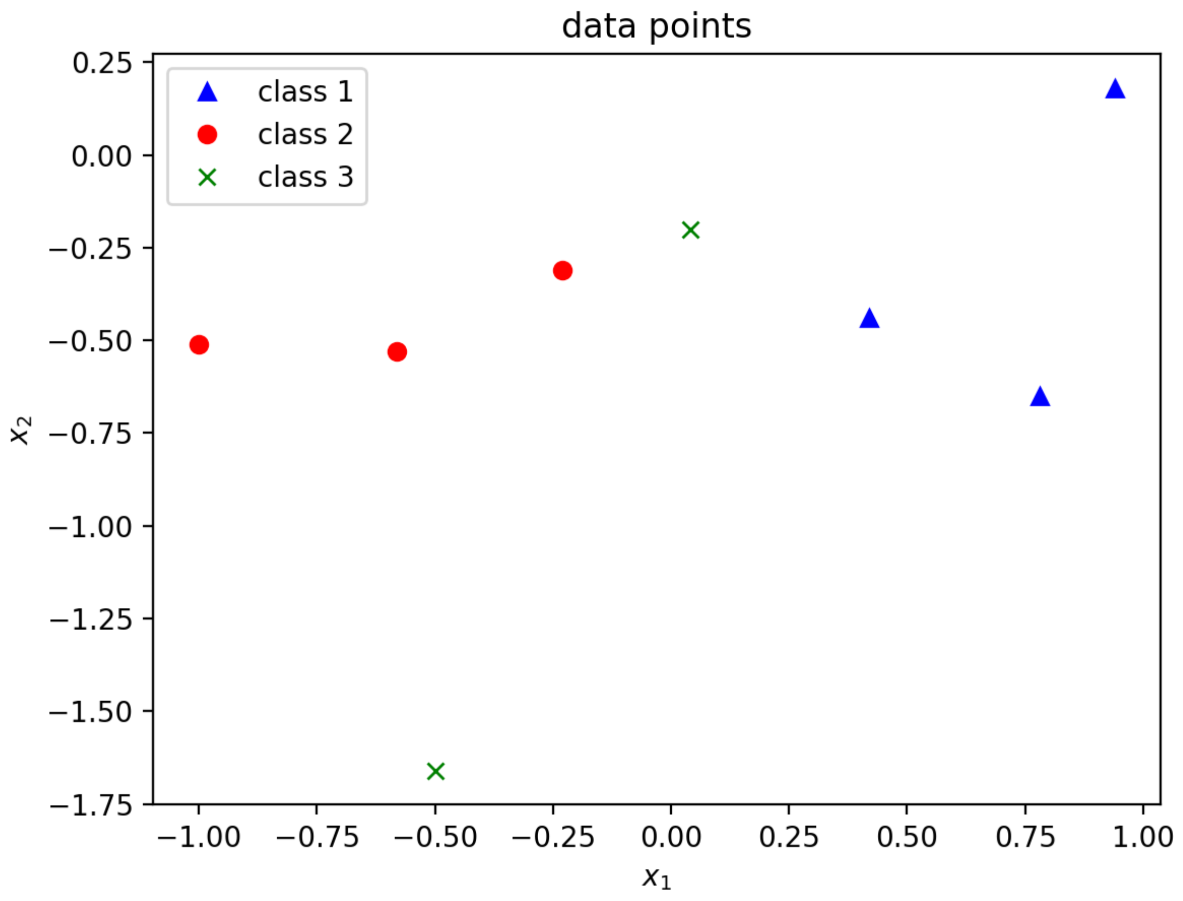
Example 2: GD of a softmax layer

Train a softmax regression layer of neurons to perform the following classification:

$$\begin{array}{rrl}
(0.94 & 0.18) & \rightarrow class\ A \\
(-0.58 & -0.53) & \rightarrow class\ B \\
(-0.23 & -0.31) & \rightarrow class\ B \\
(0.42 & -0.44) & \rightarrow class\ A \\
(0.5 & -1.66) & \rightarrow class\ C \\
(-1.0 & -0.51) & \rightarrow class\ B \\
(0.78 & -0.65) & \rightarrow class\ A \\
(0.04 & -0.20) & \rightarrow class\ C
\end{array}$$

Use a learning factor $\alpha = 0.05$.

Example 2

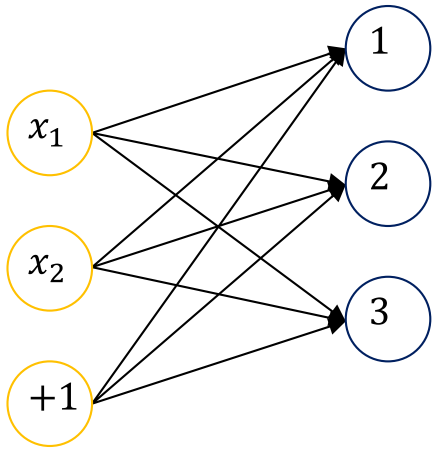

data points
▲ class 1
● class 2
× class 3
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
x_2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x_1

Example 2

$$\text{Let } y = \begin{cases} 1, \text{for } class\ A \\ 2, \text{for } class\ B \\ 3, \text{for } class\ C \end{cases}$$

$$\boldsymbol{X} = \begin{pmatrix} 0.94 & 0.18 \\ -0.58 & -0.53 \\ -0.23 & -0.31 \\ 0.42 & -0.44 \\ 0.5 & -1.66 \\ -1.0 & -0.51 \\ 0.78 & -0.65 \\ 0.04 & -0.2 \end{pmatrix}, \boldsymbol{d} = \begin{pmatrix} 1 \\ 2 \\ 2 \\ 1 \\ 3 \\ 2 \\ 1 \\ 3 \end{pmatrix}$$

$$\boldsymbol{K} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Example 2

Initialize $\boldsymbol{W} = \begin{pmatrix} 0.77 & 0.02 & 0.63 \\ 0.75 & 0.50 & 0.23 \end{pmatrix}$, $\boldsymbol{b} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$,

Then, $\boldsymbol{B} = \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$

1st epoch starts …

$$\boldsymbol{U} = \boldsymbol{XW} + \boldsymbol{B} = \begin{pmatrix} 0.94 & 0.18 \\ -0.58 & -0.53 \\ -0.23 & -0.31 \\ 0.42 & -0.44 \\ 0.5 & -1.66 \\ -1.0 & -0.51 \\ 0.78 & -0.65 \\ 0.04 & -0.2 \end{pmatrix} \begin{pmatrix} 0.77 & 0.02 & 0.63 \\ 0.75 & 0.50 & 0.23 \end{pmatrix} + \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Example 2

$$\boldsymbol{U} = \begin{pmatrix} 0.86 & 0.11 & 0.64 \\ -0.84 & -0.28 & -0.49 \\ -0.41 & -0.16 & -0.22 \\ -0.01 & -0.21 & 0.17 \\ -0.86 & -0.82 & -0.06 \\ -1.15 & -0.27 & -0.75 \\ 0.11 & -0.31 & 0.35 \\ -0.12 & -0.10 & -0.02 \end{pmatrix}$$

$$f(u_{12}) = \frac{e^{0.11}}{e^{0.86} + e^{0.11} + e^{0.64}}$$

$$f(\boldsymbol{U}) = \frac{e^{(\boldsymbol{U})}}{\sum_{k=1}^{K} e^{(\boldsymbol{U})}} = \begin{pmatrix} 0.44 & 0.21 & 0.35 \\ 0.24 & 0.42 & 0.34 \\ 0.29 & 0.37 & 0.35 \\ 0.33 & 0.27 & 0.40 \\ 0.23 & 0.24 & 0.52 \\ 0.20 & 0.49 & 0.31 \\ 0.34 & 0.22 & 0.43 \\ 0.32 & 0.33 & 0.35 \end{pmatrix}$$

Example 2

$$\boldsymbol{y} = \underset{k}{\text{argmax}}\{f(\boldsymbol{U})\} = \underset{k}{\text{argmax}}\left\{\begin{pmatrix} 0.44 & 0.21 & 0.35 \\ 0.24 & 0.42 & 0.34 \\ 0.29 & 0.37 & 0.35 \\ 0.33 & 0.27 & 0.40 \\ 0.23 & 0.24 & 0.52 \\ 0.20 & 0.49 & 0.31 \\ 0.34 & 0.22 & 0.43 \\ 0.32 & 0.33 & 0.35 \end{pmatrix}\right\} = \begin{pmatrix} 1 \\ 2 \\ 2 \\ 3 \\ 3 \\ 2 \\ 3 \\ 3 \end{pmatrix}$$

$$\boldsymbol{d} = \begin{pmatrix} 1 \\ 2 \\ 2 \\ 1 \\ 3 \\ 2 \\ 1 \\ 3 \end{pmatrix}$$

$$\text{Errors} = \sum_{p=1}^{8} 1\left(d_p \neq y_p\right) = 2$$

Entropy, $J = -\sum_{p=1}^{8} log\left(f\left(u_{pd_p}\right)\right)$

$$= -log(0.44) - log(0.42) - \cdots - log(0.35)$$

$$= 7.26$$

Example 2

$$\nabla_U J = -\left(\boldsymbol{K} - f(\boldsymbol{U})\right)$$

$$= -\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0.44 & 0.21 & 0.35 \\ 0.24 & 0.42 & 0.34 \\ 0.29 & 0.37 & 0.35 \\ 0.33 & 0.27 & 0.40 \\ 0.23 & 0.24 & 0.52 \\ 0.20 & 0.49 & 0.31 \\ 0.34 & 0.22 & 0.43 \\ 0.32 & 0.33 & 0.35 \end{pmatrix}\right)$$

$$= \begin{pmatrix} -0.56 & 0.21 & 0.35 \\ 0.24 & -0.58 & 0.34 \\ 0.29 & -0.63 & 0.35 \\ -0.67 & 0.27 & 0.40 \\ 0.23 & 0.24 & -0.48 \\ 0.20 & -0.51 & 0.31 \\ -0.65 & 0.22 & 0.43 \\ 0.32 & 0.33 & -0.65 \end{pmatrix}$$

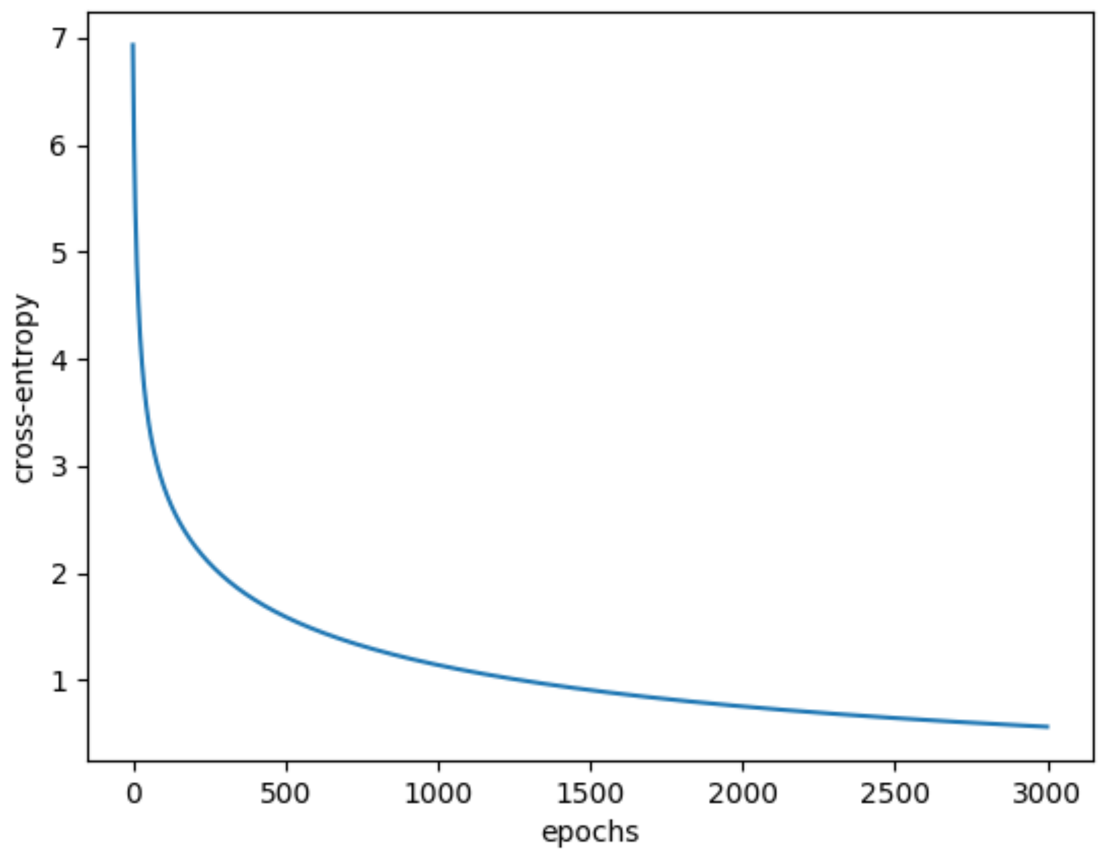
$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T \, \nabla_{\boldsymbol{U}} J$$

$$= \begin{pmatrix} 0.77 & 0.02 & 0.63 \\ 0.75 & 0.50 & 0.23 \end{pmatrix} - 0.05 \begin{pmatrix} 0.94 & 0.18 \\ -0.58 & -0.53 \\ -0.23 & -0.31 \\ 0.42 & -0.44 \\ 0.5 & -1.66 \\ -1.0 & -0.51 \\ 0.78 & -0.65 \\ 0.04 & -0.2 \end{pmatrix}^T \begin{pmatrix} -0.56 & 0.21 & 0.35 \\ 0.24 & -0.58 & 0.34 \\ 0.29 & -0.63 & 0.35 \\ -0.67 & 0.27 & 0.40 \\ 0.23 & 0.24 & -0.48 \\ 0.20 & -0.51 & 0.31 \\ -0.65 & 0.22 & 0.43 \\ 0.32 & 0.33 & -0.65 \end{pmatrix}$$

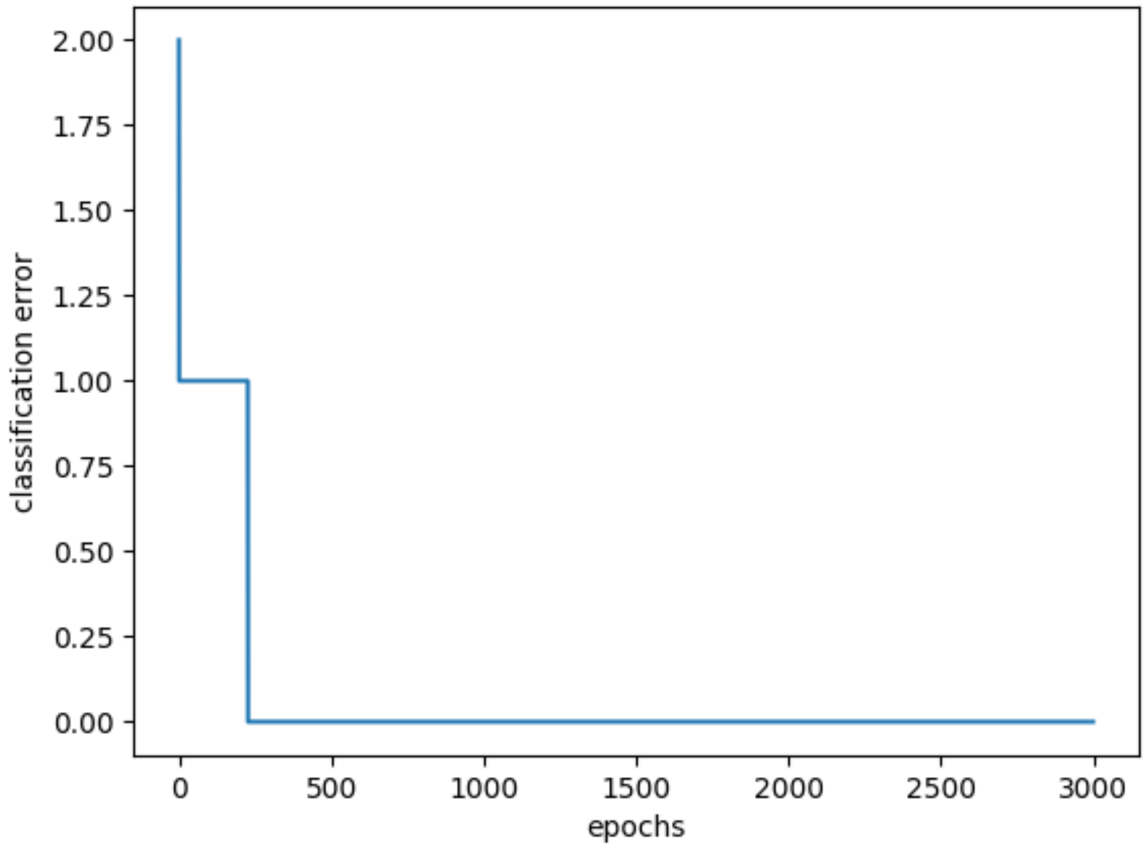
$$= \begin{pmatrix} 0.85 & -0.06 & 0.63 \\ 0.76 & 0.50 & 0.22 \end{pmatrix}$$

$$\boldsymbol{b} = \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix} - 0.05 \begin{pmatrix} -0.56 & 0.21 & 0.35 \\ 0.24 & -0.58 & 0.34 \\ 0.29 & -0.63 & 0.35 \\ -0.67 & 0.27 & 0.40 \\ 0.23 & 0.24 & -0.48 \\ 0.20 & -0.51 & 0.31 \\ -0.65 & 0.22 & 0.43 \\ 0.32 & 0.33 & -0.65 \end{pmatrix}^T \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.03 \\ 0.02 \\ -0.05 \end{pmatrix}$$

Example 2



Example 2



2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
classification error
0
500
1000
1500
2000
2500
3000
epochs

Example 2

At convergence at 3000 iterations:

$$\boldsymbol{W} = \begin{pmatrix} 14.22 & -13.04 & 0.00 \\ 4.47 & -2.05 & -0.95 \end{pmatrix}$$

$$\boldsymbol{b} = \begin{pmatrix} -0.53 \\ -0.47 \\ 1.00 \end{pmatrix}$$

Entropy = 0.562

Errors = 0

Initialization of weights

Random initialization is inefficient

At initialization, it is desirable that weights are small and near zero
- to operate in the linear region of the activation function
- to preserve the variance of activations and gradients.

Two methods:
- Using a unform distribution within specified limits
- Using a truncated normal distribution

Initialization from a uniform distribution (Xavier/Glorat uniform Initialization)

$Uniformly$ draws weight samples $w \sim U(-a,+a)$:

where

$$a = gain \times \sqrt{\frac{6}{n_{in}+n_{out}}}$$

n_{in} is the number of input nodes and n_{out} is the number of neurons in the layer.

activation	linear	sigmoid	Tanh	ReLU	Leaky ReLU
gain	1	1	5/3	$\sqrt{2}$	$\sqrt{\frac{1}{1+slope^2}}$

Initialization from a truncated normal distribution

$$w \sim truncated\_normal\left[mean = 0, std = \frac{gain}{\sqrt{n_{in}}}\right]$$

In the truncated normal, the samples that are two s.d. away from the center are discarded and resampled again.

This is also known as **Kaiming normal initialization**.

Iris dataset

Iris dataset:
https://archive.ics.uci.edu/ml/datasets/Iris

Three classes of iris flower:

Setosa

Versicolour

Virginica

Four features:
Sepal length, sepal width, petal length, petal width

Iris dataset

150 data points, 50 for each class

Features:

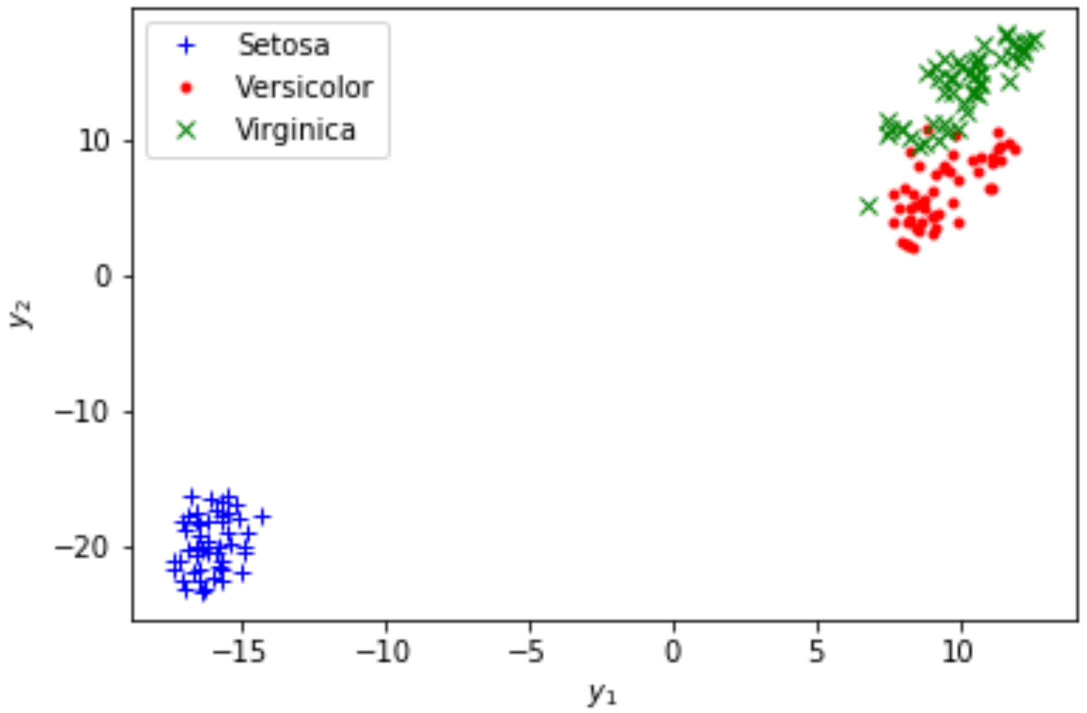
```
[ -7.43333333e-01   4.46000000e-01  -2.35866667e+00  -9.98666667e-01]
 [ -9.43333333e-01  -5.40000000e-02  -2.35866667e+00  -9.98666667e-01]
 [ -1.14333333e+00   1.46000000e-01  -2.45866667e+00  -9.98666667e-01]
```

Labels:

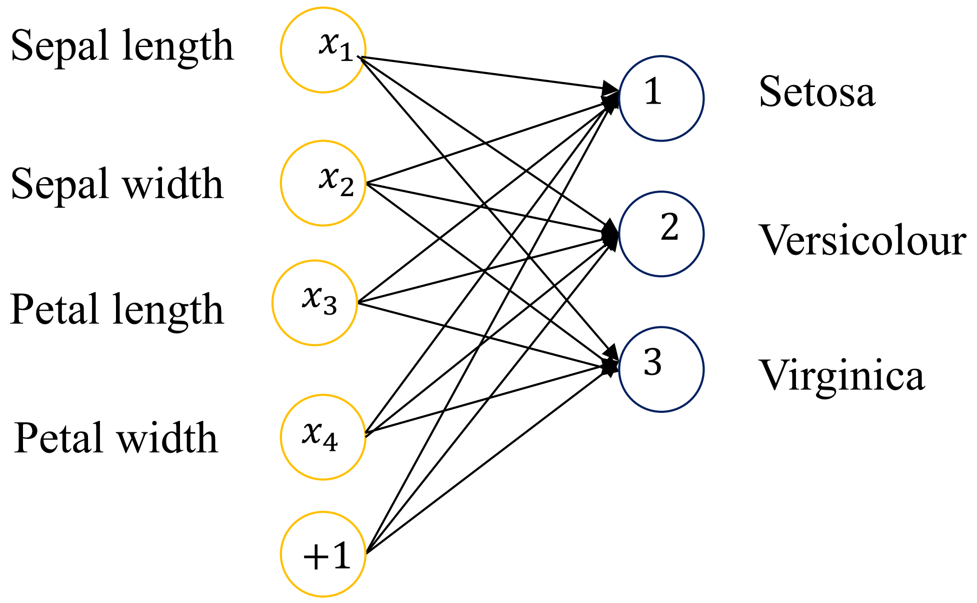
```
[0 0 0 0 0 0 0 0 0 0 ..1 1 1 1 1 1 1 1 1 1….. 2 2 2 2 2 2 2 2 2]
```

Iris data

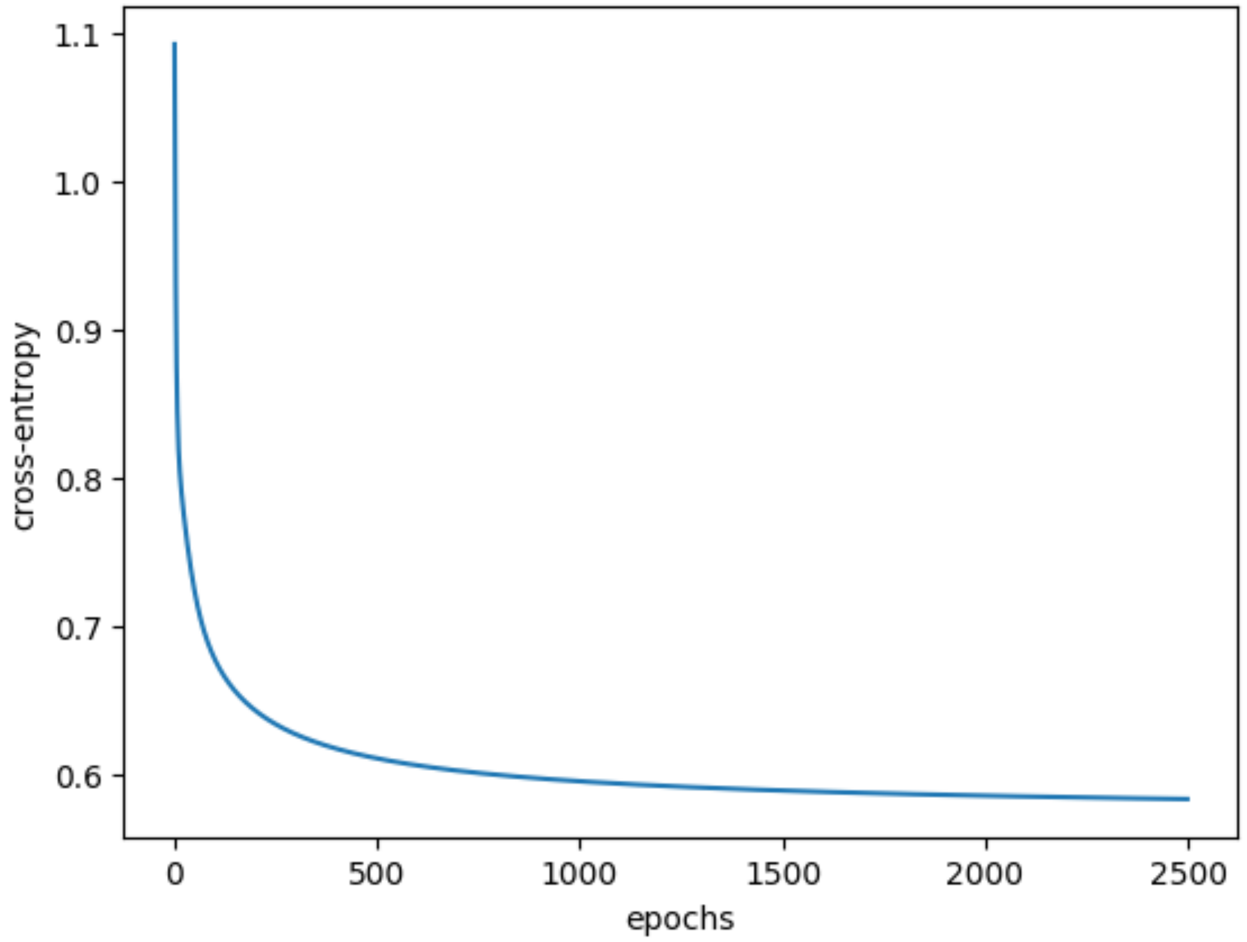
Display of data points after dimensionality reduction (from Four dimensions to Two) by t-SNE.



Example 3: Softmax classification of iris data

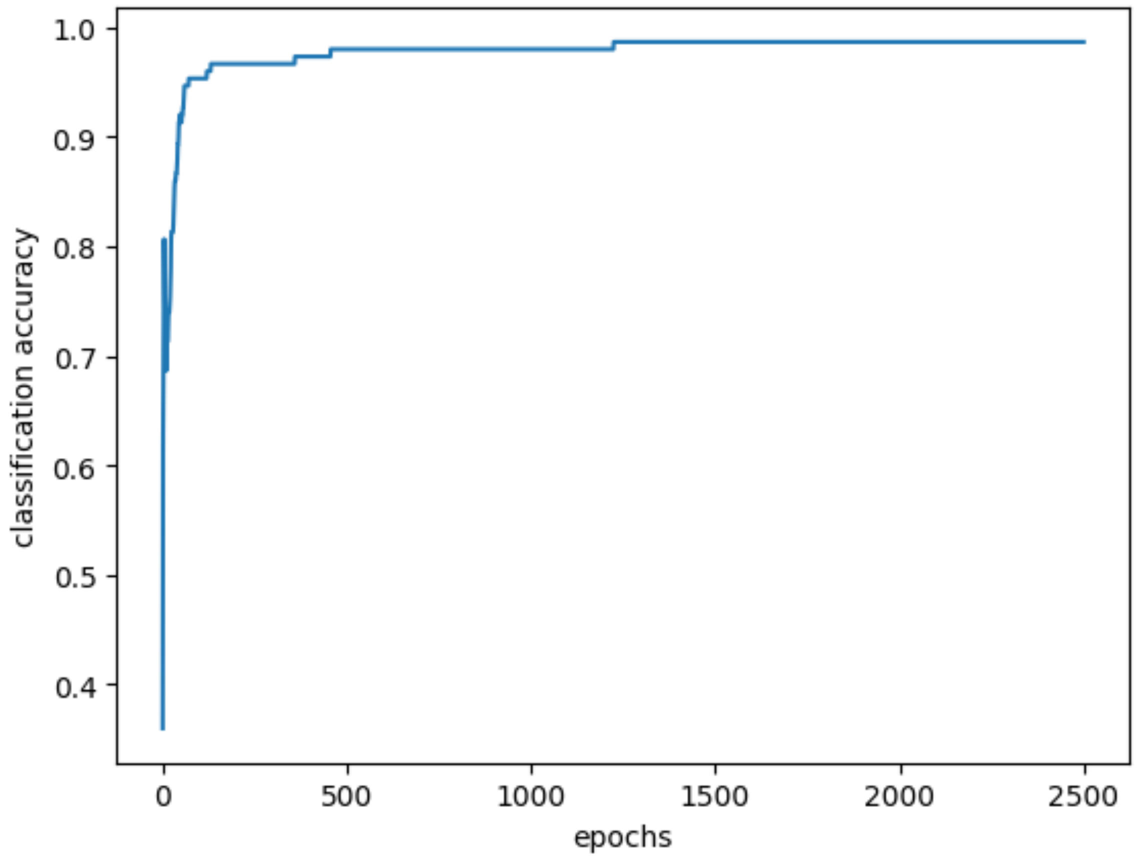


Example 3



1.1
1.0
0.9
0.8
0.7
0.6
cross-entropy
0
500
1000
1500
2000
2500
epochs

Example 3



Final classification error = 5

Revision: neurons and layers

Classification	**Logistic neurons**
Two-class	Logistic regression neuron
Multiclass	Softmax layer

Regression	**Linear**	**Non-linear**
One dimensional	Linear neuron	Perceptron
Multi-dimensional	Linear neuron layer	Perceptron layer

Summary: GD for layers

$$(\boldsymbol{X}, \boldsymbol{D})$$
$$\boldsymbol{U} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{B}$$
$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T (\boldsymbol{\nabla}_{\boldsymbol{U}} J)$$
$$\boldsymbol{b} = \boldsymbol{b} - \alpha (\boldsymbol{\nabla}_{\boldsymbol{U}} J)^T \boldsymbol{1}_P$$

layer	$f(U), Y$	$\boldsymbol{\nabla}_U J$
Linear neuron layer	$\boldsymbol{Y} = f(\boldsymbol{U}) = \boldsymbol{U}$	$-(\boldsymbol{D} - \boldsymbol{Y})$
Perceptron layer	$\boldsymbol{Y} = f(\boldsymbol{U}) = \dfrac{1}{1 + e^{-\boldsymbol{U}}}$	$-(\boldsymbol{D} - \boldsymbol{Y}) \cdot f'(\boldsymbol{U})$
Softmax layer	$f(\boldsymbol{U}) = \dfrac{e^{\boldsymbol{U}}}{\sum_{k=1}^{K} e^{\boldsymbol{U}_k}}$ $\boldsymbol{y} = \underset{k}{\operatorname{argmax}}\, f(\boldsymbol{U})$	$-(\boldsymbol{K} - f(\boldsymbol{U}))$

Summary: SGD for layers

$$(\boldsymbol{x}, \boldsymbol{d})$$
$$\boldsymbol{u} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{x} (\boldsymbol{\nabla}_{\boldsymbol{u}} J)^T$$
$$\boldsymbol{b} = \boldsymbol{b} - \alpha (\boldsymbol{\nabla}_{\boldsymbol{u}} J)$$

layer	$f(\boldsymbol{u}), \boldsymbol{y}$	$\boldsymbol{\nabla}_{\boldsymbol{u}} J$
Linear neuron layer	$\boldsymbol{y} = f(\boldsymbol{u}) = \boldsymbol{u}$	$-(\boldsymbol{d} - \boldsymbol{y})$
Perceptron layer	$\boldsymbol{y} = f(\boldsymbol{u}) = \dfrac{1}{1 + e^{-\boldsymbol{u}}}$	$-(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u})$
Softmax layer	$f(\boldsymbol{u}) = \dfrac{e^{u}}{\sum_{k'=1}^{K} e^{k'}}$ $\boldsymbol{y} = \underset{k}{\operatorname{argmax}} f(\boldsymbol{u})$	$-\big(1(\boldsymbol{k} = d) - f(\boldsymbol{u})\big)$

Thank you.

Example 4: GD of a perceptron layer

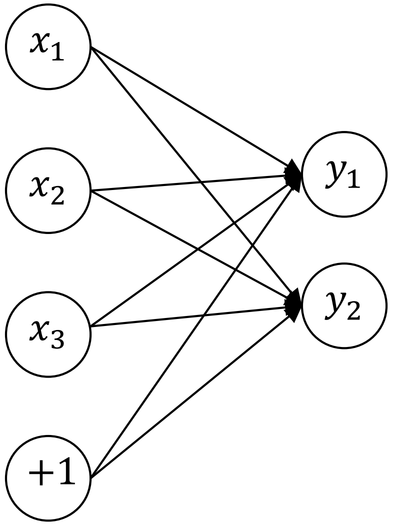
Design a perceptron layer to perform the following mapping using GD learning:

$\boldsymbol{x} = (x_1, x_2, x_3)$	$\boldsymbol{d} = (d_1, d_2\ )$
$(0.77, 0.02, 0.63)$	$(0.37, 0.47)$
$(0.75, 0.50, 0.22)$	$(0.36, 0.38)$
$(0.20, 0.76, 0.17)$	$(0.35, 0.25)$
$(0.09, 0.69, 0.95)$	$(0.48\ 0.42)$
$(0.00, 0.51, 0.81)$	$(0.36, 0.29)$
$(0.61, 0.72, 0.29)$	$(0.44\ 0.52)$
$(0.92, 0.71, 0.54)$	$(0.60, 0.52)$
$(0.14, 0.37, 0.67)$	$(0.28, 0.37)$

Use $\alpha = 0.1$.

Example 4

$$\boldsymbol{X} = \begin{pmatrix} 0.77 & 0.02 & 0.63 \\ 0.75 & 0.50 & 0.22 \\ 0.20 & 0.76 & 0.17 \\ 0.09 & 0.69 & 0.95 \\ 0.00 & 0.51 & 0.81 \\ 0.61 & 0.72 & 0.29 \\ 0.92 & 0.71 & 0.54 \\ 0.14 & 0.37 & 0.67 \end{pmatrix} \text{ and } \boldsymbol{D} = \begin{pmatrix} 0.37 & 0.47 \\ 0.36 & 0.38 \\ 0.35 & 0.25 \\ 0.48 & 0.42 \\ 0.36 & 0.29 \\ 0.44 & 0.52 \\ 0.60 & 0.52 \\ 0.28 & 0.37 \end{pmatrix}$$



Output $y_1, y_2 \in [0, 1]$

So, activation function for both neurons:

$$f(u) = \frac{1}{1 + e^{-u}}$$
$$f'(u) = y(1 - y)$$

Learning factor $\alpha = 0.1$.

Weights and biases are initialized:

$$\boldsymbol{W} = \begin{pmatrix} 0.03 & 0.04 \\ 0.01 & 0.04 \\ 0.02 & 0.04 \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$$

Example 4

Epoch 1:

$\boldsymbol{U} = \boldsymbol{XW} + \boldsymbol{B}$

$$\boldsymbol{U} = \begin{pmatrix} 0.77 & 0.02 & 0.63 \\ 0.75 & 0.50 & 0.22 \\ 0.20 & 0.76 & 0.17 \\ 0.09 & 0.69 & 0.95 \\ 0.00 & 0.51 & 0.81 \\ 0.61 & 0.72 & 0.29 \\ 0.92 & 0.71 & 0.54 \\ 0.14 & 0.37 & 0.67 \end{pmatrix} \begin{pmatrix} 0.03 & 0.04 \\ 0.01 & 0.04 \\ 0.02 & 0.04 \end{pmatrix} + \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix} = \begin{pmatrix} 0.03 & 0.06 \\ 0.03 & 0.06 \\ 0.02 & 0.05 \\ 0.03 & 0.07 \\ 0.02 & 0.05 \\ 0.03 & 0.07 \\ 0.04 & 0.09 \\ 0.02 & 0.05 \end{pmatrix}$$

$$\boldsymbol{Y} = f(\boldsymbol{U}) = \frac{1}{1+e^{-\boldsymbol{U}}} = \begin{pmatrix} 0.51 & 0.51 \\ 0.51 & 0.52 \\ 0.50 & 0.51 \\ 0.51 & 0.52 \\ 0.50 & 0.51 \\ 0.51 & 0.52 \\ 0.51 & 0.52 \\ 0.50 & 0.51 \end{pmatrix}$$

Mean square error $=\frac{1}{8}\sum_{p=1}^{8}\sum_{k=1}^{2}\left(d_{pk}-y_{pk}\right)^2 = \frac{1}{8}\sum_{p=1}^{8}\left(d_{p1}-y_{p1}\right)^2 + \left(d_{p2}-y_{p2}\right)^2 = 0.04$

Example 4

$$f'(\boldsymbol{U}) = \boldsymbol{Y} \cdot (1 - \boldsymbol{Y}) = \begin{pmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \end{pmatrix}$$

$$\nabla_{\boldsymbol{U}} J = -(\boldsymbol{D} - \boldsymbol{Y}) \cdot f'(\boldsymbol{U})$$

$$= -\left(\begin{pmatrix} 0.37 & 0.47 \\ 0.36 & 0.38 \\ 0.35 & 0.25 \\ 0.48 & 0.42 \\ 0.36 & 0.29 \\ 0.44 & 0.52 \\ 0.60 & 0.52 \\ 0.28 & 0.37 \end{pmatrix} - \begin{pmatrix} 0.51 & 0.51 \\ 0.51 & 0.52 \\ 0.50 & 0.51 \\ 0.51 & 0.52 \\ 0.50 & 0.51 \\ 0.51 & 0.52 \\ 0.51 & 0.52 \\ 0.50 & 0.51 \end{pmatrix} \right) \cdot \begin{pmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \end{pmatrix} = \begin{pmatrix} 0.04 & 0.01 \\ 0.04 & 0.03 \\ 0.04 & 0.07 \\ 0.01 & 0.03 \\ 0.04 & 0.06 \\ 0.02 & 0.00 \\ -0.02 & 0.00 \\ 0.06 & 0.04 \end{pmatrix}$$

Example 4

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_U J$$

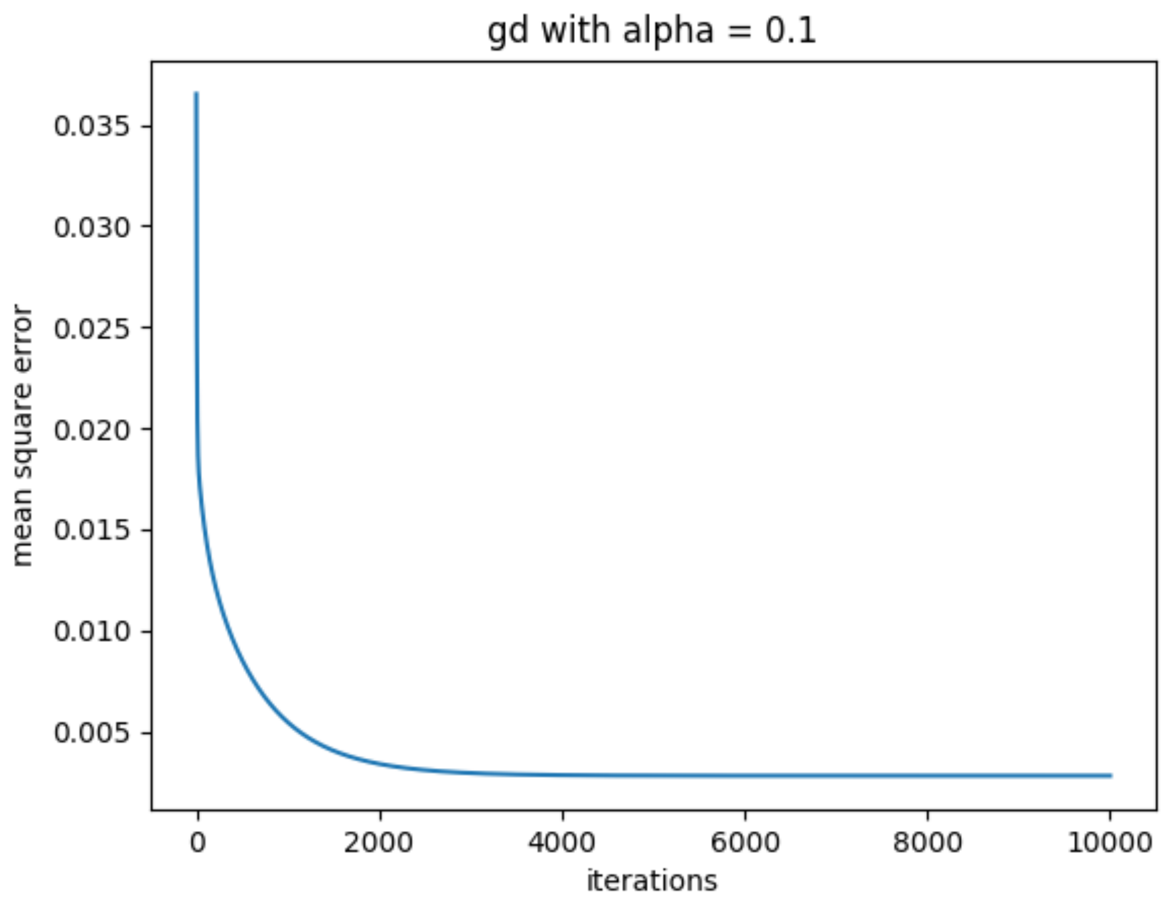
$$= \begin{pmatrix} 0.03 & 0.04 \\ 0.01 & 0.04 \\ 0.02 & 0.04 \end{pmatrix} - 0.1 \begin{pmatrix} 0.77 & 0.02 & 0.63 \\ 0.75 & 0.50 & 0.22 \\ 0.20 & 0.76 & 0.17 \\ 0.09 & 0.69 & 0.95 \\ 0.00 & 0.51 & 0.81 \\ 0.61 & 0.72 & 0.29 \\ 0.92 & 0.71 & 0.54 \\ 0.14 & 0.37 & 0.67 \end{pmatrix}^T \begin{pmatrix} 0.04 & 0.01 \\ 0.04 & 0.03 \\ 0.04 & 0.07 \\ 0.01 & 0.03 \\ 0.04 & 0.06 \\ 0.02 & 0.00 \\ -0.02 & 0.00 \\ 0.06 & 0.04 \end{pmatrix} = \begin{pmatrix} 0.02 & 0.04 \\ 0.00 & 0.03 \\ 0.01 & 0.03 \end{pmatrix}$$

$$\boldsymbol{b} = \boldsymbol{b} - \alpha (\nabla_U J)^T \boldsymbol{1}_P = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} - 0.1 \begin{pmatrix} 0.03 & 0.01 \\ 0.03 & 0.03 \\ 0.04 & 0.06 \\ 0.00 & 0.02 \\ 0.03 & 0.05 \\ 0.01 & 0.00 \\ -0.02 & 0.00 \\ 0.05 & 0.03 \end{pmatrix}^T \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.02 \\ -0.02 \end{pmatrix}$$

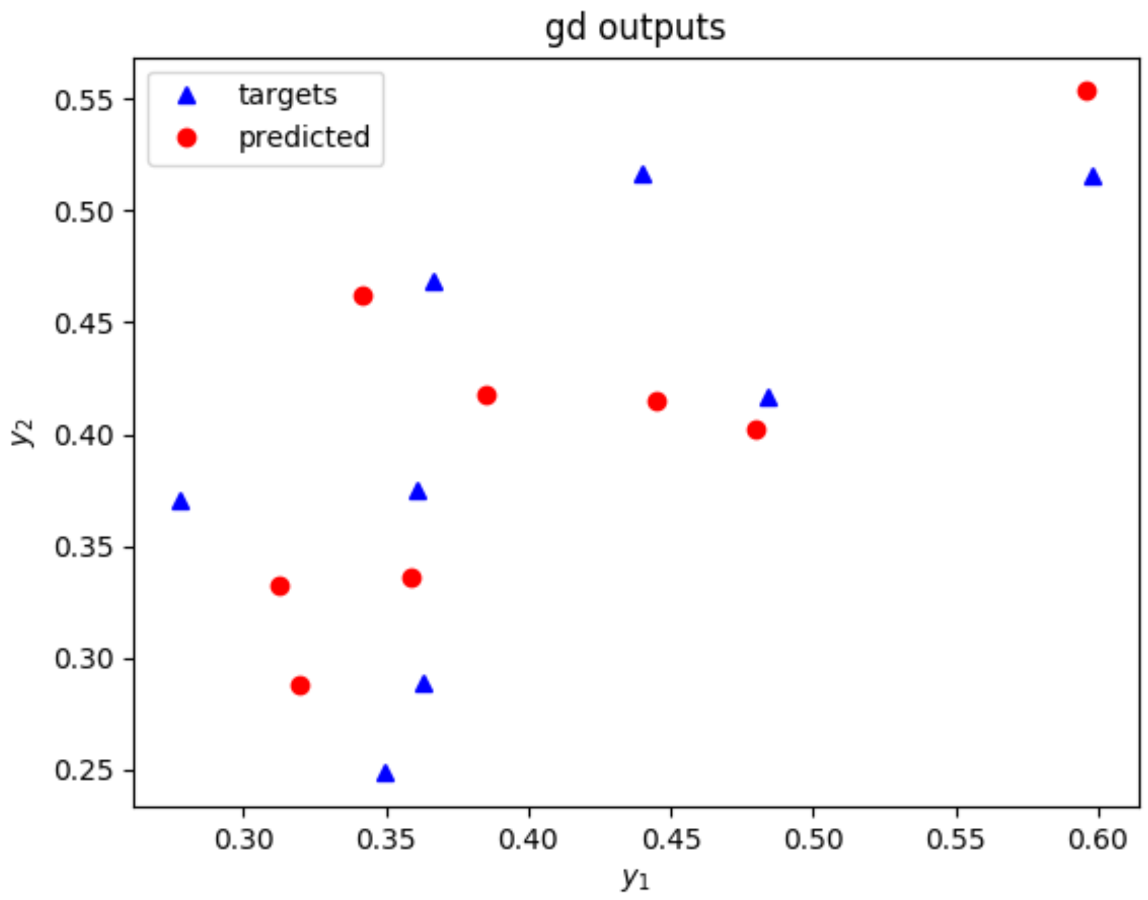
Example 4

Epoch	Y	mse	W	b
2	$\begin{pmatrix} 0.50 & 0.51 \\ 0.50 & 0.51 \\ 0.50 & 0.50 \\ 0.50 & 0.51 \\ 0.50 & 0.50 \\ 0.50 & 0.51 \\ 0.50 & 0.51 \\ 0.50 & 0.50 \end{pmatrix}$	0.036	$\begin{pmatrix} 0.02 & 0.03 \\ -0.01 & 0.02 \\ 0.00 & 0.01 \end{pmatrix}$	$\begin{pmatrix} -0.04 \\ -0.04 \end{pmatrix}$
10000	$\begin{pmatrix} 0.34 & 0.46 \\ 0.39 & 0.42 \\ 0.32 & 0.29 \\ 0.48 & 0.40 \\ 0.36 & 0.34 \\ 0.45 & 0.42 \\ 0.59 & 0.55 \\ 0.31 & 0.33 \end{pmatrix}$	0.003	$\begin{pmatrix} 1.08 & 1.14 \\ 1.42 & 0.40 \\ 1.14 & 0.84 \end{pmatrix}$	$\begin{pmatrix} -2.24 \\ -1.57 \end{pmatrix}$

Example 4


gd with alpha = 0.1
mean square error
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0
2000
4000
6000
8000
10000
iterations

Example 4


gd outputs
targets
predicted
y_2
y_1
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60