TUT 1

1) $w_1 = \begin{pmatrix} 1 \\ -0.5 \\ -1 \end{pmatrix}$ $w_2 = \begin{pmatrix} 0 \\ \frac{1}{2} \\ 0.6 \end{pmatrix}$ $w_3 = \begin{pmatrix} -0.5 \\ 0.6 \end{pmatrix}$

a) biases

$b_1 = 0$ $b_2 = 0.5$ $b_3 = 0.05$ (see +1)

$w = \begin{pmatrix} 1 & 0 \\ -0.5 & \frac{1}{2} \\ -1 & 0.6 \end{pmatrix}$ $b = \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$

b) $u_1 = 0.25 = x_1^T w_1 + b_1$

$g(u_1)$ $y_1 = 0.531$

$u_2 = x_1^T w_2 + b_2 = 0.1$

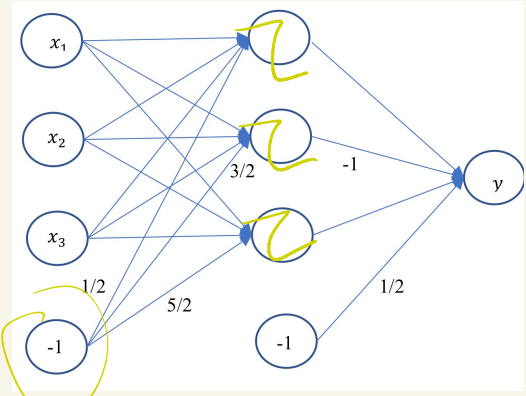
$g(u_2)$ $y_2 = 0.512$

$z = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ $u_3 = z^T w_3 + b_3 = 0.092$

$y_3 = f(u_3) = \max(0, y_3) = 0.092$

Q2) $(-0.2, 0.2)$ yes

Q3) logic function



-0.5

need to list out the truth table!!!

TUT 1

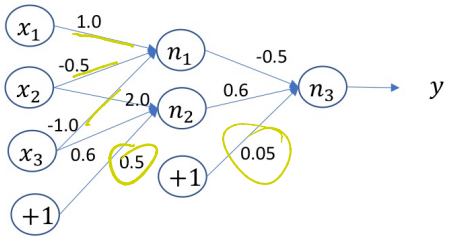


Figure 1

1. Figure 1 shows a two-layer feedforward neural network receiving 3-dimensional inputs $(x_1, x_2, x_3) \in \boldsymbol{R}^3$. The connection weights and biases of the neurons n_1, n_2, and n_3 are as indicated in the figure. The hidden-layer neurons have activation functions given by $q(u) = \frac{1.0}{1+e^{-0.5u}}$ where u denotes the synaptic input to the neuron. The activation function $f(u)$ of the output neuron is a ReLU function: $f(u) = \max\{0, u\}$.

   a. Write weight vectors and biases connected to individual neurons, and the weight matrix and bias vector connected to the hidden layer.

   b. Find the synaptic inputs and activations of the neurons for the following input signals:

   (i) (1.0, –0.5, 1.0) (ii) (-1.0, 0.0, –2.0) (iii) (2.0, 0.5, –1.0).

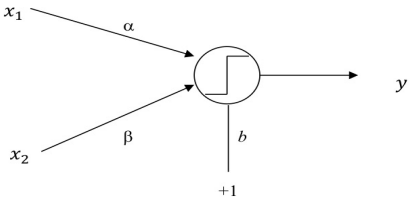


Figure 2

2. Two input binary neuron shown in figure 2 has a unit step activation function with a bias $b = 0.5$ and receives two-dimensional input $(x_1, x_2) \in \boldsymbol{R}^2$.

(a) Find the space of possible values of weights (α, β) if the neuron is
   (i) ON for input (1.0, 1.0)
   (ii) ON for input (0.5, -1.0)
   (iii) OFF for input (2.0, -0.5).

(b) Indicate the weight space in 2-D α-β plot and show that (-0.2, 0.2) is in this space.

3. The network shown in figure 3 consists of neurons having threshold activation functions and receives three-bit binary patterns $(x_1, x_2, x_3) \in \{0,1\}^3$. By analyzing the outputs for all possible three-bit input patterns, determine the logic function that the network implements. All unlabeled weights shown in figure 3 are of unity weight.

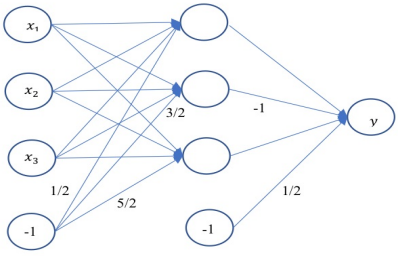


Figure 3

1) $u = w^T x + b$ / $w^T x - \theta$

$g(u) = \frac{1}{1+e^{-0.5u}}$ Recall $f(u) = \max\{0, u\}$

$u > 0$ $f(u) = u$

$u < 0$ $f(u) = 0$

For hidden layer !! For outer layer!

a) weight^T $= \begin{pmatrix} 1 & -0.5 & -1 \\ 0 & 2 & 0.6 \end{pmatrix}$ $\begin{matrix} \leftarrow n_1 \\ \leftarrow n_2 \end{matrix}$ bias $\begin{pmatrix} 0.5 \\ 0.05 \end{pmatrix}$

weight vector n_1, n_2 $= \begin{pmatrix} 1 & 0 \\ -0.5 & 2 \\ -1 & 0.6 \end{pmatrix}$ (n_1, n_2) weight vector n_3 $= \begin{pmatrix} -0.5 \\ 0.6 \end{pmatrix}$ (n_3)

bias vector for $n_1 = 0$, $n_2 = 0.5$, $n_3 = 0.05$

weight vector $\begin{pmatrix} 1 & 0 \\ -0.5 & 2 \\ -1 & 0.6 \end{pmatrix}$ $\begin{pmatrix} -0.5 \\ 0.6 \end{pmatrix}$

bias vector $\begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$ 0.05

b) Synaptic inputs (u)

activation of neurons (y)

for input

i) $x = \begin{pmatrix} 1 \\ -0.5 \end{pmatrix}$ $\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = w^T x + b$

$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 & -0.5 & -1 \\ 0 & 2 & 0.6 \end{pmatrix} \begin{pmatrix} 1 \\ -0.5 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$

$= \begin{pmatrix} 1 + 0.25 - 1 \\ 0 - 1 + 0.6 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$

$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 0.25 \\ -0.4 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0.1 \end{pmatrix}$ $g_1(u) = 0.531$, $g_2(u) = 0.512$

$u_3 = w_3^T x_3 + b$

$= (-0.5 \ \ 0.6) \begin{pmatrix} 0.531 \\ 0.512 \end{pmatrix} + (0.05) = 0.0417 + 0.05 = 0.0917$

$u_3 > 0$, $f(u_3) = u_3$

ii) $\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 & -0.5 & -1 \\ 0 & 2 & 0.6 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ -2 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$

$= \begin{pmatrix} -1 + 0 + 2 \\ 0 + 0 - 1.2 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 1 \\ -0.7 \end{pmatrix}$

$\begin{pmatrix} g(u_1) \\ g(u_2) \end{pmatrix} = \begin{pmatrix} \frac{1}{1+e^{-0.5}} \\ \frac{1}{1+e^{-0.5 \cdot -0.7}} \end{pmatrix} = \begin{pmatrix} 0.622 \\ 0.413 \end{pmatrix}$

$u_3 = (-0.5 \ \ 0.6) \begin{pmatrix} 0.622 \\ 0.413 \end{pmatrix} + 0.05 = -0.013$

$u_3 < 0$

$g(u_2) = 0$

iii) $\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 & -0.5 & -1 \\ 0 & 2 & 0.6 \end{pmatrix} \begin{pmatrix} 2 \\ 0.5 \\ -1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$

$= \begin{pmatrix} 2 - 0.25 + 1 \\ 0 + 1 - 0.6 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 2.75 \\ 0.9 \end{pmatrix}$ $-0.399 + 0.366 + 0.05$

$\begin{pmatrix} g(u_1) \\ g(u_2) \end{pmatrix} = \begin{pmatrix} 0.798 \\ 0.711 \end{pmatrix}$ $u_3 = (-0.5 \ \ 0.6) \begin{pmatrix} 0.798 \\ 0.711 \end{pmatrix} + 0.05 = 0.0176$, $u_3 > 0$, $f(u_3) = u_3 = 0.017$

2) unit step activation

$I(x) = \begin{cases} 1 \\ 0 \end{cases}$ $\quad w = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$

$b = 0.5$

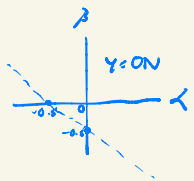
a) $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ neuron $Y = 1$ (assume $Y=1 \equiv$ ON) if $y = 1$

$y_1 = w^T x + b$

$= (\alpha \ \beta)\begin{pmatrix} 1 \\ 1 \end{pmatrix} + 0.5$

$y = \alpha + \beta + 0.5$

if $u = 0$ $\quad \alpha + \beta = -0.5$



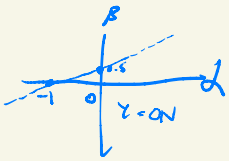
$y_2 = (\alpha \beta)\begin{pmatrix} 0.5 \\ -1 \end{pmatrix} + 0.5$

$y_1 = 0.5\alpha - \beta + 0.5$

if $y_2 = 0$ $\quad 0.5\alpha - \beta = -0.5$

if $\alpha = 0$ $\ \beta = 0.5$

$\beta = 0$ $\ \alpha = -1$



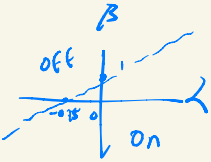
$y_3 = (\alpha \beta)\begin{pmatrix} 2 \\ -0.5 \end{pmatrix} + 0.5$

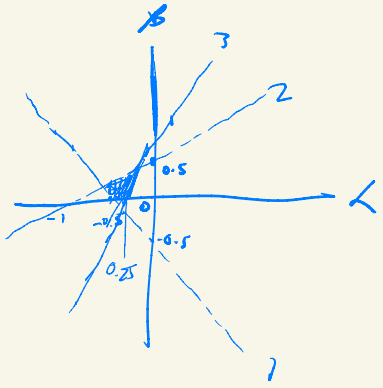
$y_3 = 2\alpha - 0.5\beta + 0.5$

if $y_3 = 0$ $\quad 2\alpha - 0.5\beta = -0.5$

if $\alpha = 0$ $\quad \beta = 1$

if $\beta = 0$ $\quad \alpha = -0.25$





b) $(-0.2 \quad 0.2)$

(from picture yes

or

use 2 and 3 line

$y_3 = 2(-0.2) - 0.5(0.2) + 0.5 = -0.4 - 0.1 + 0.5 = 0.8 \rightarrow$ on ✓

$y_2 = 0.5(-0.2) - 0.2 + 0.5 = -0.1 - 0.2 + 0.5 = 0.2 \rightarrow$ on

3) weight vector = $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ hidden layer $\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$ output layer

$\theta = 1/2 \quad 3/2 \quad 5/2 \qquad 1/2$

$u = w^T x - \theta$

$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - \begin{matrix} 1/2 \\ 3/2 \\ 5/2 \end{matrix} = \begin{matrix} x_1 + x_2 + x_3 - 1/2 \\ x_1 + x_2 + x_3 - 3/2 \\ x_1 + x_2 + x_3 - 5/2 \end{matrix}$

$u = w^T x - \theta = (1 \; -1 \; 1) \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} - 1/2 = u_1 - u_2 + u_3 - 1/2$

x_1	x_2	x_3	u_1	u_2	u_3	y_1	y_2	y_3	u_{final}	y_3
0	0	0	$-1/2$	$-3/2$	$-5/2$	0	0	0	-0.5	0
0	0	1	$1/2$	$-1/2$	$-3/2$	1	0	0	0.5	1
0	1	0	$1/2$	$-1/2$	$-3/2$	1	0	0	0.5	1
0	1	1	1.5	$1/2$	$-1/2$	1	1	0	-0.5	0
1	0	0	$1/2$	$-1/2$	$-3/2$	1	0	0	0.5	1
1	0	1	1.5	$1/2$	$-1/2$	1	1	0	-0.5	0
1	1	0	1.5	$1/2$	$-1/2$	1	1	0	-0.5	0
1	1	1	2.5	1.5	0.5	1	1	1	0.5	1

$y_1 \bar{y}_2 \bar{y}_3 + y_1 y_2 y_3$

$y = \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 x_3$

$= \bar{x}_1 (\bar{x}_2 x_3 + x_2 \bar{x}_3) + x_1 (\bar{x}_2 \bar{x}_3 + x_2 x_3)$

$= \bar{x}_1 (x_2 \oplus x_3) + x_1 (\overline{x_2 \oplus x_3})$

$= \bar{x}_1 \oplus (x_2 \oplus x_3)$

$= x_1 \oplus x_2 \oplus x_3$

TUT2

$x = (x_1, x_2, x_3)$			y
(0.09	−0.44	−0.15)	−2.57
(0.69	−0.99	−0.76)	−2.97
(0.34	0.65	−0.73)	0.96
(0.15	0.78	−0.58)	1.04
(−0.63	−0.78	−0.56)	−3.21
(0.96	0.62	−0.66)	1.05
(0.63	−0.45	−0.14)	−2.39
(0.88	0.64	−0.33)	0.66

1) linear neuron

$p = 8$

a) SGD

$w_0 = \begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix}$ $b = 0$ $\alpha = 0.01$

Shuffle

$p = 1, 2, 3, 4, 5, 6, 7, 8$

epoch 1

Shuffle the patterns

$p = 1$ $x_p^T = (0.09 \ -0.44 \ -0.15)$

$y_p = w^T x_p + b$

$w \leftarrow w + \alpha (d_p - y_p) x_p$

$b \leftarrow b + \alpha (d_p - y_p)$

$Se \leftarrow (d_p - y_p)^2$

$y_p = (0.77, 0.02, 0.63) \begin{pmatrix} 0.09 \\ -0.44 \\ -0.15 \end{pmatrix} + 0 = -0.034$

$w \leftarrow \begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix} + 0.01 \left(-2.57 - (-0.034)\right) \begin{pmatrix} 0.09 \\ -0.44 \\ -0.15 \end{pmatrix} = \begin{pmatrix} 0.768 \\ 0.031 \\ 0.634 \end{pmatrix}$

$b \leftarrow 0 + -0.02536 = -0.0254$

$p = 1$ choose 3

$x^T = (0.34 \ 0.65 \ -0.73)$ $d = 0.96$

$y_p = (0.77 \ 0.02 \ 0.63) \begin{pmatrix} 0.34 \\ 0.65 \\ -0.73 \end{pmatrix} + 0 = -0.1851$

$w \leftarrow \begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix} + 0.01 \left(0.96 - (-0.1851)\right) \begin{pmatrix} 0.34 \\ 0.65 \\ -0.73 \end{pmatrix} = \begin{pmatrix} 0.774 \\ 0.028 \\ 0.622 \end{pmatrix}$

$Se = 1.316$

$b \leftarrow 0 + 0.011451 = 0.011451$

$p = 2$ choose 4

$x^T = (0.15 \ 0.78 \ -0.58)$ $d = 1.04$

$y_p = (0.774 \ 0.028 \ 0.622) \begin{pmatrix} 0.15 \\ 0.78 \\ -0.58 \end{pmatrix} + 0.011451 = -0.21161$

$w \leftarrow \begin{pmatrix} 0.774 \\ 0.028 \\ 0.622 \end{pmatrix} + 0.01 \left(1.04 - (-0.21161)\right) \begin{pmatrix} 0.15 \\ 0.78 \\ -0.58 \end{pmatrix} = \begin{pmatrix} 0.776 \\ 0.037 \\ 0.614 \end{pmatrix}$

$Se = 1.564$

$b \leftarrow 0.011451 + 0.012516099 = 0.024$

$p = 3$ choose 7

$x^T = (0.63 \ -0.45 \ -0.14)$ $d = -2.39$

$y_p = (0.776 \ 0.037 \ 0.614) \begin{pmatrix} 0.63 \\ -0.45 \\ -0.14 \end{pmatrix} + 0.024 = 0.40994$

$w \leftarrow \begin{pmatrix} 0.776 \\ 0.037 \\ 0.614 \end{pmatrix} + 0.01 \left(-2.39 - (0.40994)\right) \begin{pmatrix} 0.63 \\ -0.45 \\ -0.14 \end{pmatrix} = \begin{pmatrix} 0.758 \\ 0.050 \\ 0.618 \end{pmatrix}$

$Se = 7.84$

$b \leftarrow 0.024 + (-0.027999467) = -0.004$

$p = 4$ choose 8

$x^T = (0.88 \ 0.64 \ -0.33)$ $d = 0.66$

$y_p = (0.758 \ 0.05 \ 0.618) \begin{pmatrix} 0.88 \\ 0.64 \\ -0.33 \end{pmatrix} + -0.004 = 0.49095$

$w \leftarrow \begin{pmatrix} 0.758 \\ 0.05 \\ 0.618 \end{pmatrix} + 0.01 \left(0.66 - (0.49095)\right) \begin{pmatrix} 0.88 \\ 0.64 \\ -0.33 \end{pmatrix} = \begin{pmatrix} 0.76 \\ 0.051 \\ 0.618 \end{pmatrix}$

$Se = 0.0286$

$b \leftarrow -0.004 + \ldots = -0.00234$

P = 5 choose 1

$x^T = (0.09 \ -0.44 \ -0.15)$ $d = -2.57$

$y_p = \begin{pmatrix} 0.76 \\ 0.051 \\ 0.618 \end{pmatrix} \begin{pmatrix} 0.09 \\ -0.44 \\ -0.15 \end{pmatrix} - 0.00234 = -0.049$ $SC = 6.35$

$w \leftarrow \begin{pmatrix} 0.76 \\ 0.051 \\ 0.618 \end{pmatrix} + 0.01\left(-2.57 - (-0.049)\right)\begin{pmatrix} 0.09 \\ -0.44 \\ -0.15 \end{pmatrix} = \begin{pmatrix} 0.751 \\ 0.062 \\ 0.622 \end{pmatrix}$

$b \leftarrow -0.00234 + -0.0252 = -0.0276$

P = 6 choose 5

$(-0.63 \ -0.78 \ -0.56)$ $d = -3.21$

$y_p = \begin{pmatrix} 0.751 \\ 0.062 \\ 0.622 \end{pmatrix} \begin{pmatrix} -0.63 \\ -0.78 \\ -0.56 \end{pmatrix} + b = -0.9012$

$w \leftarrow \begin{pmatrix} 0.751 \\ 0.062 \\ 0.622 \end{pmatrix} + 0.01\left(-3.21 - (-0.9012)\right)\begin{pmatrix} -0.63 \\ -0.78 \\ -0.56 \end{pmatrix} = \begin{pmatrix} 0.772 \\ 0.08 \\ 0.6345 \end{pmatrix}$

$b \leftarrow -0.0276 + -0.02309 = -0.0506$

$Se \leftarrow 5.3311$

P = 7 choose 6

$x^T = 0.96 \ \ 0.62 \ \ -0.66$ $d = 1.05$

$y_p = w^T x + b = \begin{pmatrix} 0.772 \\ 0.08 \\ 0.6345 \end{pmatrix}\begin{pmatrix} 0.96 \\ 0.62 \\ -0.66 \end{pmatrix} + (-0.0506) = 0.321$

$w \leftarrow \begin{pmatrix} 0.772 \\ 0.08 \\ 0.6345 \end{pmatrix} + 0.01\left(1.05 - y\right)x = \begin{pmatrix} 0.779 \\ 0.085 \\ 0.63 \end{pmatrix}$

$b \leftarrow -0.0506 + 0.00729 = -0.0434$

$Se \leftarrow 0.5311$

p = 8 choose 2

$x = \begin{pmatrix} 0.69 \\ -0.99 \\ -0.76 \end{pmatrix}$ $d = -2.97$ $\begin{pmatrix} 0.779 \\ 0.085 \\ 0.63 \end{pmatrix}$ -0.0434

$y_p = w^T x + b = (0.779\ 0.085\ 0.63) \begin{pmatrix} 0.69 \\ -0.99 \\ -0.76 \end{pmatrix} + b = -0.068$

$w \leftarrow w + 0.01(-2.97 - y)x = \begin{pmatrix} 0.759 \\ 0.113 \\ 0.652 \end{pmatrix}$

$b \leftarrow b + -0.029$ -0.0724

$se \leftarrow 8.421 = (d - y_p)^2$

ii) GD

$y = u = Xw + b\,1_p$

$w \leftarrow w + \alpha X^T (d - y)$

$b \leftarrow b + \alpha\, 1_p^T (d - y)$

$X = \begin{pmatrix} 0.09 & -0.44 & -0.15 \\ 0.69 & -0.99 & -0.76 \\ 0.34 & 0.65 & -0.73 \\ 0.15 & 0.78 & -0.58 \\ -0.63 & -0.78 & -0.56 \\ 0.96 & 0.62 & -0.66 \\ 0.63 & -0.45 & -0.14 \\ 0.88 & 0.64 & -0.33 \end{pmatrix}$ $d = \begin{pmatrix} -2.97 \\ -2.97 \\ 0.96 \\ 1.04 \\ -3.21 \\ 1.05 \\ -2.79 \\ 0.66 \end{pmatrix}$

$w = \begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix}$ $b = 0$ $\alpha = 0.01$

$y = u = \begin{pmatrix} 0.09 & -0.44 & -0.15 \\ 0.69 & -0.99 & -0.76 \\ 0.34 & 0.65 & -0.73 \\ 0.15 & 0.78 & -0.58 \\ -0.63 & -0.78 & -0.56 \\ 0.96 & 0.62 & -0.66 \\ 0.63 & -0.45 & -0.14 \\ 0.88 & 0.64 & -0.33 \end{pmatrix} \begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix} + 0 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.034 \\ 0.0327 \\ -0.185 \\ -0.234 \\ -0.854 \\ 0.336 \\ 0.388 \\ 0.483 \end{pmatrix}$

$w = \begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix} + 0.01 \begin{pmatrix} 0.09 & -0.44 & -0.15 \\ 0.69 & -0.99 & -0.76 \\ 0.34 & 0.65 & -0.73 \\ 0.15 & 0.78 & -0.58 \\ -0.63 & -0.78 & -0.56 \\ 0.96 & 0.62 & -0.66 \\ 0.63 & -0.45 & -0.14 \\ 0.88 & 0.64 & -0.33 \end{pmatrix}^T (d - y) = \begin{pmatrix} 0.759 \\ 0.115 \\ 0.653 \end{pmatrix}$

$b = 0 + 0.01 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T (d - y) = -0.013621$

60

Output $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + b\mathbf{1}_P =$

$$\begin{pmatrix} 0.09 & -0.44 & -0.15 \\ 0.69 & -0.99 & -0.76 \\ 0.34 & 0.65 & -0.73 \\ 0.15 & 0.78 & -0.58 \\ -0.63 & -0.78 & -0.56 \\ 0.96 & 0.62 & -0.66 \\ 0.63 & -0.45 & -0.14 \\ 0.88 & 0.64 & -0.33 \end{pmatrix} \begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix} + 0.0 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.03 \\ 0.03 \\ -0.18 \\ -0.24 \\ -0.85 \\ 0.34 \\ 0.39 \\ 0.48 \end{pmatrix}$$

8×3, 3×1 → 8×1, 8×1, 8×1

$$\boldsymbol{d} = \begin{pmatrix} -2.57 \\ -2.97 \\ 0.96 \\ 1.04 \\ -3.21 \\ 1.05 \\ -2.39 \\ 0.66 \end{pmatrix}$$

m.s.e. $= \frac{1}{8}\sum_{p=1}^{8}(d_p - y_p)^2$

$= \frac{1}{8}((-2.57 + 0.03)^2 + (-2.97 - 0.03)^2 + \cdots\cdots + (0.66 - 0.48)^2)$

$= 4.02$

epoch	y	mse	w	b
2	$\begin{pmatrix} -0.15 \\ -0.16 \\ -0.22 \\ -0.25 \\ -1.01 \\ 0.29 \\ 0.26 \\ 0.45 \end{pmatrix}$	3.66	$\begin{pmatrix} 0.75 \\ 0.21 \\ 0.67 \end{pmatrix}$	-0.14
200	$\begin{pmatrix} -2.23 \\ -3.29 \\ 0.78 \\ 1.01 \\ -3.28 \\ 0.92 \\ -2.06 \\ 0.87 \end{pmatrix}$	0.05	$\begin{pmatrix} 0.368 \\ 2.56 \\ -0.21 \end{pmatrix}$	-1.164

$\boldsymbol{w} = \boldsymbol{w} + \alpha \boldsymbol{X}^T(\boldsymbol{d} - \boldsymbol{y})$

8×3, 3×1

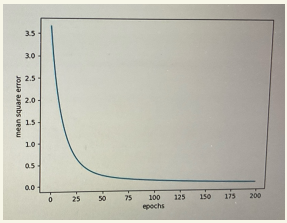
$$= \begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix} + 0.01 \times \begin{pmatrix} 0.09 & 0.69 & 0.34 & 0.15 & -0.63 & 0.96 & 0.63 & 0.88 \\ -0.44 & -0.99 & 0.65 & 0.78 & -0.78 & 0.62 & -0.45 & 0.64 \\ -0.15 & -0.76 & -0.73 & -0.58 & -0.56 & -0.66 & -0.14 & -0.33 \end{pmatrix} \left(\begin{pmatrix} -2.57 \\ -2.97 \\ 0.96 \\ 1.04 \\ -3.21 \\ 1.05 \\ -2.39 \\ 0.66 \end{pmatrix} - \begin{pmatrix} -0.03 \\ 0.03 \\ -0.18 \\ -0.24 \\ -0.85 \\ 0.34 \\ 0.39 \\ 0.48 \end{pmatrix} \right)$$

3×8, 8×1

$$= \begin{pmatrix} 0.76 \\ 0.11 \\ 0.65 \end{pmatrix}$$

$$b = b + \alpha \mathbf{1}_P^T(\boldsymbol{d} - \boldsymbol{y}) = 0.0 + 0.01 \times (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1) \left(\begin{pmatrix} -2.57 \\ -2.97 \\ 0.96 \\ 1.04 \\ -3.21 \\ 1.05 \\ -2.39 \\ 0.66 \end{pmatrix} - \begin{pmatrix} -0.03 \\ 0.03 \\ -0.18 \\ -0.24 \\ -0.85 \\ 0.34 \\ 0.39 \\ 0.48 \end{pmatrix} \right) = -0.07$$

1×8, 8×1



At convergence:

$\boldsymbol{w} = \begin{pmatrix} 0.37 \\ 2.57 \\ -0.21 \end{pmatrix}, b = -1.16$

mse = 0.054

If $\boldsymbol{x} = (x_1, x_2, x_3)^T$, the learned function by the linear neuron:

$$y = \boldsymbol{w}^T\boldsymbol{x} + b$$

$$y = (0.37 \quad 2.57 \quad -0.21)\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - 1.16$$

$$y = 0.37x_1 + 2.57x_2 - 0.21x_3 - 1.16$$

After learning ...

x	d	y (SGD)	y (GD)
(0.09, −0.44, −0.15)	−2.57	−2.23	−2.23
(0.69, −0.99, −0.76)	−2.97	−3.29	−3.29
(0.34, 0.65, −0.73)	0.96	0.78	0.78
(0.15, , 0.78, −0.58)	1.04	1.01	1.01
(−0.63, −0.78, −0.56)	−3.21	−3.28	−3.28
(0.96, 0.62, −0.66)	1.05	0.92	0.92
(0.63, −0.45, −0.14)	−2.39	−2.06	−2.06
(0.88, 0.64, −0.33)	0.66	0.87	0.88
m.s.e.		0.055	0.054
$\boldsymbol{w}$		$\begin{pmatrix} 0.369 \\ 2.566 \\ -0.212 \end{pmatrix}$ ⟺	$\begin{pmatrix} 0.368 \\ 2.567 \\ -0.207 \end{pmatrix}$
b		−1.165 ⟺	-1.163
y		$0.37x_1 + 2.57x_2 - 0.21x_3 - 1.1$	$0.37x_1 + 2.57x_2 - 0.21x_3 - 1.16$

1b) $w = \begin{pmatrix} 0.36871063 \\ 2.56531648 \\ -0.22584826 \end{pmatrix}$ ↓

$Y = 0.37x_1 + 2.57x_2 - 0.22x_3 - 1.17$

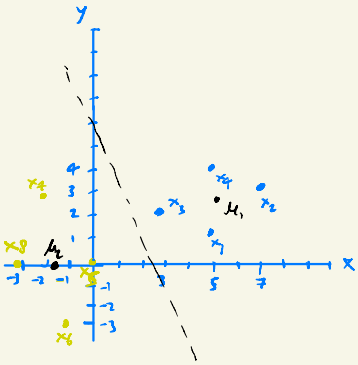
$b = -1.16421293402482$

mse : 0.0845471

1c) $Y_p = w^T x + b$

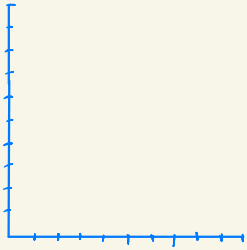
	Y
1	−2.23
2	−3.29
3	0.782
4	1.013
5	−3.281
6	0.914
7	−2.059
8	0.87

2)



class 1
class 2

a) yes they are linearly seperable



b) $6.5x_1 + 2.5x_2 - 14.5 = 0$ decision boundry?

take class 1 = 0
class 2 = 1

converse $w = \begin{pmatrix} 6.5 \\ 2.5 \end{pmatrix}$ $b = -14.5$

$$X = \begin{pmatrix} x_1 & x_2 \\ 5 & 1 \\ 4 & 3 \\ 3 & 2 \\ 5 & 4 \\ 0 & 0 \\ -1 & -3 \\ -2 & 3 \\ -3 & 0 \end{pmatrix} \quad d = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

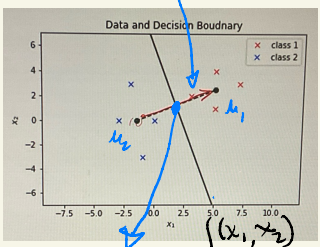
Steps :

(1) Find 2 Centroids

Center of class 1 $(\mu_1) = \frac{1}{4}(x_1 + x_2 + x_3 + x_4) = \frac{1}{4}\begin{pmatrix} x_1 & x_2 \\ 20 & 10 \end{pmatrix} = (5 \;\; 2.5)$

Center of class 2 $(\mu_2) = \frac{1}{4}(x_5 + x_6 + x_7 + x_8) = \frac{1}{4}\begin{pmatrix} -6 & 0 \end{pmatrix} = (-1.5 \;\; 0)$

(2) Find line joining 2 centroids



$$\mu_1 - \mu_2 = \begin{pmatrix} 5 \\ 2.5 \end{pmatrix} - \begin{pmatrix} -1.5 \\ 0 \end{pmatrix} = \begin{pmatrix} 6.5 \\ 2.5 \end{pmatrix}$$

(3) middle point connecting 2 Centroid $= \frac{1}{2}(\mu_1 + \mu_2)$

$$= \frac{1}{2}\left(\begin{pmatrix} 5 \\ 2.5 \end{pmatrix} + \begin{pmatrix} -1.5 \\ 0 \end{pmatrix}\right) = \begin{pmatrix} 1.75 \\ 1.25 \end{pmatrix}$$

(4) line passing perpendicularly through middle point

$\begin{pmatrix} x_1 - 1.75 \\ x_2 - 1.25 \end{pmatrix}$

↳ inner product $= 0$

$\begin{pmatrix} x_1 - 1.75 \\ x_2 - 1.25 \end{pmatrix}$ vector $\mu_1 - \mu_2 = 0$

$\begin{pmatrix} x_1 - 1.75 \\ x_2 - 1.25 \end{pmatrix} \cdot \begin{pmatrix} 6.5 \\ 2.5 \end{pmatrix} = 0$

$6.5x_1 - 11.375 + 2.5x_2 - 3.125 = 0$

$6.5x_1 + 2.5x_2 - 14.5 = 0$

2c) use $u = 6.5x_1 + 2.5x_2 - 14.5$ if $u > 0$ class 1
$u \leq 0$ class 2

2d) ↳ use $x_1 \sim x_8$ to check (training)

test the 3 infer → result class 1, 2, 1

2a)

x_1	20.5	} Class 1
x_2	38.5	
x_3	10	
x_4	28	
x_5	-14.5	} Class 2
x_6	-28.5	
x_7	-20	
x_8	-34	

if $u > 0$ → Class 1
$u \leq 0$ → Class 2

2d) $\begin{pmatrix} 4 \\ 2 \end{pmatrix}$ $u = 16.5$ → Class 1

$\begin{pmatrix} 0 \\ 5 \end{pmatrix}$ $u = -2$ → Class 2

$\begin{pmatrix} 36/12 \\ 0 \end{pmatrix}$ $u = 3.5$ → Class 1

TUT 3

1) a) GD $\alpha = 0.05$ $w = \begin{pmatrix} 0.88 & 0.08 & -0.34 \\ 0.68 & -0.39 & -0.19 \end{pmatrix}$ $b = 0$

$k = 2$ Neuron = 9

take class A = 0
class B = 1
class C = 2

$X = \begin{pmatrix} 0 & 4 \\ -1 & 3 \\ 2 & 3 \\ -2 & 2 \\ 0 & 2 \\ 1 & 2 \\ -1 & 2 \\ -3 & 1 \\ -1 & 1 \end{pmatrix}$ $d = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

$X = \begin{pmatrix} 2 & 1 \\ 4 & 1 \\ -2 & 0 \\ 1 & 0 \\ 3 & 0 \\ -3 & -1 \\ -2 & -1 \\ 2 & -1 \\ 4 & -1 \end{pmatrix}$ $d = \begin{pmatrix} 2 \\ 2 \\ 1 \\ 2 \\ 2 \\ 1 \\ 1 \\ 2 \\ 2 \end{pmatrix}$

$y = \underset{k}{\arg\max}\, f(u)$ $f(u) = \frac{e^u}{\sum_{k=1}^{2} e^{u_k}}$ $u = xw + B$

$w = w - \alpha x^T \left(-(k - f(u))\right)$

$b = b - \alpha \left(-(k - f(u))\right)^T 1_p$

1st epoch

$k = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ (0 1 2)

$u = xw + B = \underset{N \times 2}{\begin{pmatrix} 0 & 4 \\ -1 & 3 \\ 2 & 3 \\ -2 & 2 \\ 0 & 2 \\ 1 & 2 \\ -1 & 2 \\ -3 & 1 \\ -1 & 1 \end{pmatrix}} \underset{2 \times 3}{\begin{pmatrix} 0.88 & 0.08 & -0.34 \\ 0.68 & -0.39 & -0.19 \end{pmatrix}} + 0 = \begin{matrix} \text{result} \\ \begin{pmatrix} 2.72 & -1.56 & -0.76 \\ 1.16 & -1.25 & -0.23 \\ 2.84 & 0.71 & -1.17 \\ 18.4 & 19 & \end{pmatrix} \end{matrix}$ 1 error

$f(u) = \frac{e^u}{\sum_{k=1}^{2} e^{u_k}} = \begin{pmatrix} 0 & 1 & 2 \\ 0.957 & 0.013 & 0.0295 \end{pmatrix}$

$y = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$ 1 y's

cal $\nabla_w J$, w, and b

b, c, d

TUT4

1) $x_1 = \begin{pmatrix}1\\3\end{pmatrix}$ $d_1 = \begin{pmatrix}0\\1\end{pmatrix}$ $x_2 = \begin{pmatrix}-2\\-2\end{pmatrix}$ $d_2 = \begin{pmatrix}1\\0\end{pmatrix}$ GD of 2 patterns Single feedforward and feedback

a) matrix of W hidden layer V output layer } weight n biases

$w = \begin{pmatrix}1 & 2\\-2 & 0\end{pmatrix}$ (hidden layer) $V = \begin{pmatrix}1 & 1\\0 & -2\end{pmatrix}$ (output layer) $d = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$

$b_w = (3\ -1)$ $b_v = (-2\ \ 3)$

b) z n h of hidden layer
u n y of output layer

$z = xw + B = \begin{pmatrix}n_1 & n_2\\1 & 3\\-2 & -2\end{pmatrix}\begin{pmatrix}1 & 2\\-2 & 0\end{pmatrix} + \begin{pmatrix}3 & -1\\-2 & 3\end{pmatrix} = \begin{pmatrix}-5 & 2\\2 & -4\end{pmatrix} + \begin{pmatrix}3 & -1\\3 & -1\end{pmatrix} = \begin{pmatrix}-2 & 1\\5 & -5\end{pmatrix}$

$h = g(z) = \frac{1}{1+e^{-z}} = \begin{pmatrix}0.12 & 0.73\\0.99 & 0.01\end{pmatrix}$

$u = HV + C = \begin{pmatrix}0.12 & 0.73\\0.99 & 0.01\end{pmatrix}\begin{pmatrix}1 & 1\\0 & -2\end{pmatrix} + \begin{pmatrix}-2 & 3\\-2 & 3\end{pmatrix} = \begin{pmatrix}-1.88 & 1.66\\-1.01 & 1.97\end{pmatrix}$

$Y = f(u) = \frac{1}{1+e^{-u}} = \begin{pmatrix}0.13 & 0.84\\0.267 & 0.98\end{pmatrix}$

c) $m.s.e. = \frac{1}{2}\sum_{p=1}^{p}\sum_{k=1}^{k}(d_{pk}-y_{pk})^2 = \frac{1}{2}\left((0-0.13)^2 + (1-0.84)^2 + (1-0.27)^2 + (0-0.98)^2\right) = 0.77.$

d)

$f'(u) = y(1-y) = \begin{pmatrix}0.11 & 0.13\\0.2 & 0.02\end{pmatrix}$

! $\nabla_u J = -(D-Y)\overset{f'(u)}{\wedge} = -\left(\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix} - \begin{pmatrix}0.13 & 0.84\\0.27 & 0.99\end{pmatrix}\right) = \begin{pmatrix}0.13 & -0.16\\-0.73 & 0.99\end{pmatrix}$ $f'(u) = \begin{pmatrix}0.02 & -0.02\\-0.14 & 0.02\end{pmatrix}$

$g'(z) = h(1-h) = \begin{pmatrix}0.1 & 0.2\\0.01 & 0.01\end{pmatrix}$

$\nabla_z J = (\nabla_u J)V^T \cdot g'(z) = \nabla_u J\begin{pmatrix}1 & 0\\1 & -2\end{pmatrix} \cdot \begin{pmatrix}0.1 & 0.2\\0.01 & 0.01\end{pmatrix} = \begin{pmatrix}0 & 0.04\\-0.12 & -0.04\end{pmatrix}\begin{pmatrix}0.1 & 0.2\\0.01 & 0.01\end{pmatrix} = \begin{pmatrix}-0.001 & 0.01\\-0.001 & 0\end{pmatrix}$

e) $V \leftarrow V - \alpha H^T \nabla_u J$

$C \leftarrow C - \alpha(\nabla_u J)^T 1_p$

$w \leftarrow w - \alpha x^T \nabla_z J$

$b \leftarrow b - \alpha(\nabla_z J)^T 1_p$

2) $k = 1$

$$Z = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 3 & 4 \\ 2 & 2 \\ 2 & -2 \\ -2 & -3 \end{pmatrix} \begin{pmatrix} -0.1 & 0.97 & 0.18 \\ -0.7 & 0.38 & 0.93 \end{pmatrix} + B$$

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

$$B = \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \qquad 0$$

$H = g(z)$

$U = HV + C$

$f(u) =$

$y = \operatorname{argmax}_k f(u)$

Perceptron: $f(u) = \frac{1}{1+e^{-z}}$

Softmax: $f(u) = \frac{e^u}{\sum_{k=1}^{k} e^{u_k}}$

$y = \operatorname{argmax} f(u)$

do the rest!

$\text{ReLu} = \max\{0, u\}$

3)

0 0

0 0 0

0 0

hidden

linear norm $= u$

output

TUT 6

1) a)

$$I = \begin{pmatrix} 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

Padding = 0 strides = (1,1)

$u_1 = Conv(I, w_1) + b_1$

$u_1(1,1) = I = \begin{matrix} 0.7 & 0.1 & 0.2 \\ 0.8 & 0.1 & 0.3 \\ 1.0 & 0.2 & 0.0 \end{matrix}$ $w + b_1 = 9$

$u_2(1,2) \quad I = \begin{matrix} 0.1 & 0.2 & 0.3 \\ 0.1 & 0.3 & 0.5 \\ 0.2 & 0.0 & 0.3 \end{matrix}$ $w + b_1 = 6$ Continue

$u_2(2,1) \quad I = \begin{matrix} 0.8 & 0.1 & 0.3 \\ 1.0 & 0.2 & 0.0 \\ 0.8 & 0.1 & 0.5 \end{matrix}$ $w + b_1 = c$

$u_1 = \begin{pmatrix} a & b & \cdots \\ c & \cdots & \cdots \end{pmatrix}$

do for u_2 too $\quad u_2 = Conv(I, w_2) + b_2$

$y_1 = f(u_1) = \frac{1}{1+e^{-u_1}}$

$y_2 = f(u_2) = \frac{1}{1+e^{-u_2}}$ //

Output spatial size
= (Input size – Filter size +2(Padding))/Stride + 1

= (6 – 3)/1 + 1 = 4

For Padding = 1 Strides = (2,2)

↳ start

$u_1(1,1)$ Padding $u_1(1,2)$ $u_1(1,3)$

Padding

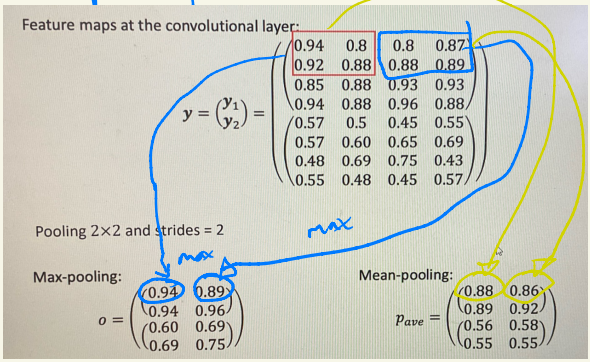
$$I = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0 & 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 0 & 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0 & 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ 0 & 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 0 & 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

$u_1(2,1)$

the rest Same

1b) Strides (2,2)

max pooling



TUT1

TUT2

TUT3

TUT4

TUT5

TUT6

TUT7

TUT8

TUT 10

1) 4 hidden neuron $\alpha < 0.1$

$w, b, f(u)$

$h = f(W^T x + b)$

$= f$

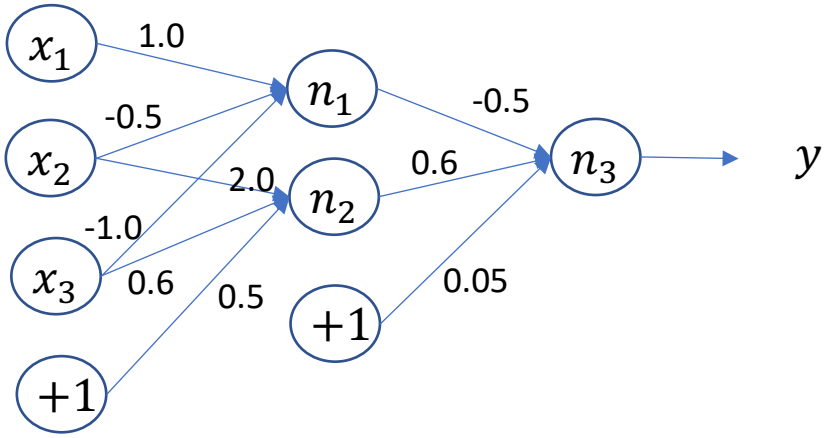


Figure 1

1. Figure 1 shows a two-layer feedforward neural network receiving 3-dimensional inputs $(x_1, x_2, x_3) \in \boldsymbol{R}^3$. The connection weights and biases of the neurons $n_1, n_2,$ and n_3 are as indicated in the figure. The hidden-layer neurons have activation functions given by $g(u) = \frac{1.0}{1+e^{-0.5u}}$ where u denotes the synaptic input to the neuron. The activation function $f(u)$ of the output neuron is a ReLU function: $f(u) = \max\{0,\ u\}$.

   a. Write weight vectors and biases connected to individual neurons, and the weight matrix and bias vector connected to the hidden layer.

   b. Find the synaptic inputs and activations of the neurons for the following input signals:

      (i) (1.0, –0.5, 1.0) (ii) (-1.0, 0.0, –2.0) (iii) (2.0, 0.5, –1.0).

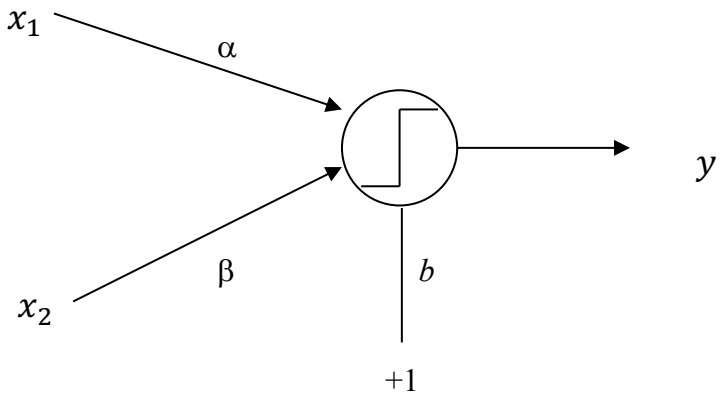


Figure 2

2. Two input binary neuron shown in figure 2 has a unit step activation function with a bias b = 0.5 and receives two-dimensional input $(x_1, x_2) \in \boldsymbol{R}^2$.

   (a) Find the space of possible values of weights (α, β) if the neuron is
   (i) ON for input (1.0, 1.0)
   (ii) ON for input (0.5, -1.0)
   (iii) OFF for input (2.0, -0.5).

   (b) Indicate the weight space in 2-D α-β plot and show that (-0.2, 0.2) is in this space.

3. The network shown in figure 3 consists of neurons having threshold activation functions and receives three-bit binary patterns $(x_1, x_2, x_3) \in \{0,1\}^3$. By analyzing the outputs for all possible three-bit input patterns, determine the logic function that the network implements. All unlabeled weights shown in figure 3 are of unity weight.

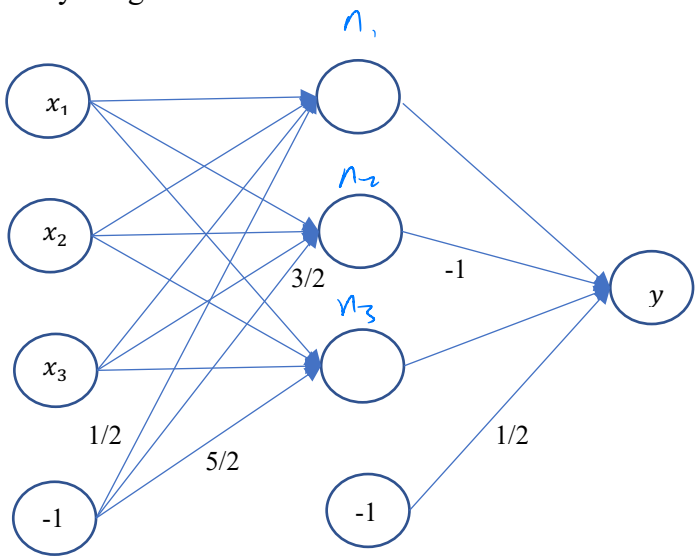


Figure 3

SC4001 Tutorial 2 **Regression and classification**

1. Train a linear neuron to learng the mapping from input $\boldsymbol{x} \in \boldsymbol{R}^3$ to output y from the following examples:

	$\boldsymbol{x} = (x_1, x_2, x_2)$	y
1	(0.09 −0.44 −0.15)	−2.57
2	(0.69 −0.99 −0.76)	−2.97
3	(0.34 0.65 −0.73)	0.96
4	(0.15 0.78 −0.58)	1.04
5	(−0.63 −0.78 −0.56)	−3.21
6	(0.96 0.62 −0.66)	1.05
7	(0.63 −0.45 −0.14)	−2.39
8	(0.88 0.64 −0.33)	0.66

(a) Show one iteration of learning of the neuron with
   i. Stochastic gradient descent learning
   ii. Gradient descent learning

Initialize the weights as $\begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix}$ and biases to 0.0, and use a leaning factor $\alpha = 0.01$.

(b) Plot the learning curves (mean square error vs. epochs) until convergence. Determine the learned weights and biases.

(c) Find the predicted values y of training inputs after the training.

2. Two-dimensional training patterns (inputs) to design a dichotomizer are given as:

$$\mathbf{X}_1 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}; \quad \mathbf{X}_2 = \begin{bmatrix} 7 \\ 3 \end{bmatrix}; \quad \mathbf{X}_3 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}; \quad \mathbf{X}_4 = \begin{bmatrix} 5 \\ 4 \end{bmatrix}; \qquad \textit{Class } 1$$

$$\mathbf{X}_5 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad \mathbf{X}_6 = \begin{bmatrix} -1 \\ -3 \end{bmatrix}; \quad \mathbf{X}_7 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}; \quad \mathbf{X}_8 = \begin{bmatrix} -3 \\ 0 \end{bmatrix}; \qquad \textit{Class } 2$$

(a) Determine whether the two classes of patterns are linearly separable.

(b) Find the center of gravity of patterns in each class. Show that a linear decision boundary passing perpendicularly through the middle point of the line joining the two centroids is given by:

$$6.5x_1 + 2.5x_2 - 14.5 = 0$$

(c) Design a discrete perceptron having the decision boundary as in part (b) for the classification and show that the perceptron separates the points perfectly.

(d) Determine the classes identified by the neuron for following input patterns:

$$\begin{pmatrix}4\\2\end{pmatrix}, \begin{pmatrix}0\\5\end{pmatrix}, \begin{pmatrix}36/13\\0\end{pmatrix}$$

3. Use 'make_blobs' function from **sklearn**.datasets to create 100 samples of two Gaussian distributed classes for 3-dimensional inputs:

   from **sklearn**.datasets import make_blobs

   Assume each class has a standard deviation = 5.0.

   Use torch.nn.BCELoss() and torch.autograd() functions to train a logistic neuron to separate the two classes.
   Find the classification error at convergence and plot the decision boundary.

4. Train a perceptron to learn the following function ϕ:

$$\phi(x, y) = 1.5 + 3.3x - 2.5y + 1.2xy$$

for $0 \leq x, y \leq 1$.

(a) Sample 100 data points randomly from the input space to create a training dataset.
(b) Use the gradient descent algorithm to train the perceptron
(c) Compute the training error and plot the function approximated by the perceptron.
(d) Show how a linear neuron can be used to predict the above function
(e) Compare the results of approximations above by the linear neuron and the perceptron

SC4001 Tutorial 3 **Neuron Layers**

1. Train a softmax layer of neurons to perform the following classification, given the inputs $\boldsymbol{x} = (x_1, x_2)$ and target class labels d:

(x_1, x_2)	(0 4)	(-1 3)	(2 3)	(-2 2)	(0 2)	(1 2)	(-1 2)	(-3 1)	(-1 1)
d	A	A	A	B	B	A	B	B	B

(x_1, x_2)	(2 1)	(4 1)	(-2 0)	(1 0)	(3 0)	(-3 -1)	(-2 -1)	(2 -1)	(4 -1)
d	C	C	B	C	C	B	B	C	C

   (a) Show one iteration of gradient descent learning at a learning factor 0.05.
   Initialize the weights to $\begin{pmatrix} 0.88 & 0.08 & -0.34 \\ 0.68 & -0.39 & -0.19 \end{pmatrix}$ and biases to zero
   (b) Find the weights and biases at the convergence of learning
   (c) Indicate the probabilities that the network predicts the classes of trained patterns.
   (d) Plot the decision boundaries separating the three classes.

2. Use mini-batch gradient decent learning to train a softmax layer to classify Iris dataset (https://archive.ics.uci.edu/ml/datasets/Iris). The dataset contains 150 data points. Use 90 data points for training the classifier and the remaining 60 data points for testing. Plot cross-entropies and classification accuracies against epochs for both train and test data. Set learning rate = 0.1, batch size = 16, and number of epochs = 1000.

   You can use the following python commands to load Iris data:

```
from sklearn import datasets
iris = datasets.load_iris()
```

   Repeat the classification with batch sizes = 2, 4, 8, 16, 24, 32, and 64, and compare the accuracies and the times taken for an epoch at different batch sizes.

3. Read the Linnerud dataset from sklearn.datasets package by using

    from sklearn.datasets i**mport** load_linnerud

    The Linnerud dataset is a multi-output regression dataset consisting of three exercise (data) and three physiological variables (targets) collected from twenty middle-aged men in a fitness club:
    physiological - Weight, Waist and Pulse
    exercise - Chins, Situps and Jumps.

    Divide the dataset into train and test partitions at 0.75:0.25 ratio and train a perceptron layer to predict physiological data from exercise variables by implementing with
    a) direct gradients
    b) 'autograd' functions available in pytorch

Draw learning curves and find mean square error and R^2 values of prediction. Which physiological variable can be better predicted by exercise data?

Compare the time-taken for a weight update and the number of epochs required for convergence for (a) and (b).

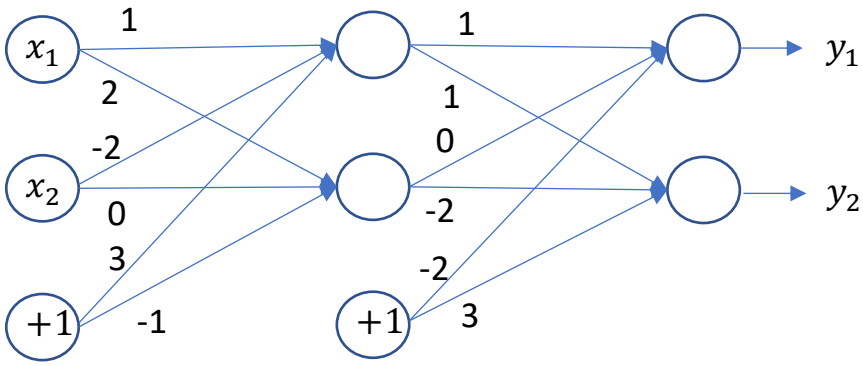


Figure 1

1. The two-layer feedforward perceptron network shown in figure 1 has weights and biases initialized as indicated and receives 2-dimensional inputs (x_1, x_2). The network is to respond with $\boldsymbol{d}_1 = \begin{pmatrix}0\\1\end{pmatrix}$ and $\boldsymbol{d}_2 = \begin{pmatrix}1\\0\end{pmatrix}$ for input patterns $\boldsymbol{x}_1 = \begin{pmatrix}1.0\\3.0\end{pmatrix}$ and $\boldsymbol{x}_2 = \begin{pmatrix}-2.0\\-2.0\end{pmatrix}$, respectively.

   Analyse a single feedforward and feedback step for gradient decent learning of the two patterns by doing the following:

   (a) Find the weight matrix $\boldsymbol{W}$ to the hidden-layer and weight matrix $\boldsymbol{V}$ to the output-layer, and the corresponding biases.
   (b) Calculate the synaptic input $\boldsymbol{z}$ and output $\boldsymbol{h}$ of the hidden-layer, and the synaptic input $\boldsymbol{u}$ and output $\boldsymbol{y} = (y_1, y_2)$ of the output layer.
   (c) Find the mean square error cost J between the outputs and targets.
   (d) Calculate the gradients $\boldsymbol{\nabla}_u J$ and $\boldsymbol{\nabla}_z J$ at the output-layer and the hidden-layer, respectively.
   (e) Compute the new weights and biases.
   (f) Write a program to continue iterations until convergence and find the final weights and biases.

   Assume a learning rate of 0.05.

   Repeat above (a) – (f) for stochastic gradient decent learning.

2. A feedforward neural network with one hidden layer is to perform the following classification:

class	inputs
A	$(1.0, 1.0), (0.0, 1.0)$
B	$(3.0, 4.0), (2.0, 2.0)$
C	$(2.0, -2.0), (-2.0, -3.0)$

The network has a hidden layer consisting of three perceptrons and a softmax output layer.

Initialize the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ to the hidden layer, and the weights $\boldsymbol{V}$ and biases $\boldsymbol{c}$ to the output layer as follows:

$$\boldsymbol{W} = \begin{pmatrix} -0.10 & 0.97 & 0.18 \\ -0.70 & 0.38 & 0.93 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

$$\boldsymbol{V} = \begin{pmatrix} 1.01 & 0.09 & -0.39 \\ 0.79 & -0.45 & -0.22 \\ 0.28 & 0.96 & -0.07 \end{pmatrix}, \boldsymbol{c} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

Show one iteration of gradient descent learning and plot the learning curves until convergence at a learning rate $\alpha = 0.1$.

Determine the weights and biases at convergence.

Find the class labels predicted by the trained network for patterns:
$\boldsymbol{x}_1 = \begin{pmatrix} 2.5 \\ 1.5 \end{pmatrix}$ and $\boldsymbol{x}_2 = \begin{pmatrix} -1.5 \\ 0.5 \end{pmatrix}$

3. Design a deep neural network consisting of two ReLU hidden layers to approximate the following function:

$$\phi(x, y) = 0.8x^2 - y^3 + 2.5xy$$

for $-1.0 \leq x, y \leq 1.0$.

Use 10 neurons and 5 neurons at first and second hidden layers, respectively, and a linear neuron at the output layer.

(a) Divide the input space equally into square regions of size 0.25×0.25 and use grid points as data points to learn the function ϕ.
(b) Train the network using gradient decent learning at learning rate $\alpha = 0.01$ and plot the learning curve (mean square error vs. iterations) and the predicted data points.
(c) Compare the behaviour of learning by plotting learning curves at rates $\alpha = 0.005, 0.001, 0.01, \text{and } 0.05$.

1. The first hidden layer of a convolution neural network (CNN) has a convolution layer consisting of two feature maps with filters $\boldsymbol{w_1}$and $\boldsymbol{w_2}$ and biases = 0.1, and neurons having sigmoid activation functions, and a pooling layer with a pooling windows of size 2x2:

$$\boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \text{ and } \boldsymbol{w}_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

The input layer is of 6x6 size and receives an input image $\boldsymbol{I}$:

$$\boldsymbol{I} = \begin{pmatrix} 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

   a. Find the outputs at the first convolution layer if
      i. padding = 0 and strides = $(1, 1)$.
      ii. padding = 1 and strides = $(2, 2)$.
   b. Find the outputs at the first pooling layer for Part (a), assuming strides of $(2, 2)$ and pooling is
      i. max pooling
      ii. mean pooling

2. Given '3wolfmoon.jpg' color image of size 639×516.
   a. Initialize weights and biases of a convolutional layer with two kernels of size 9×9. Note that the input image is in color and has three channels.
   b. Display the feature maps at the convolution layer, assuming sigmoid activation functions. Use VALID padding and strides = 1.
   c. Display the outputs of a mean pooling layer with a pooling window size 5×5 and strides = 5.

3. Design a CNN with one hidden layer to recognize digit images in MNIST database:

   http://yann.lecun.com/exdb/mnist/

   The convolution layer consists of 25 filters of dimensions 9x9 and the pooling layer has a pooling window size 4x4. Assume VALID padding and default strides for both convolution

and pooling layer. Train the network with mini batch gradient decent learning with learning factor $\alpha = 10^{-3}$ and batch size = 128.

Plot

a. The training and test errors against learning epochs.
b. Final filter weights
c. Feature maps at the convolution and pooling layers for a representative test pattern
d. Repeat training by introducing decay parameter $\beta = 10^{-6}$ and momentum term with $\gamma = 0.5$, and compare the learning curves

TUT 1

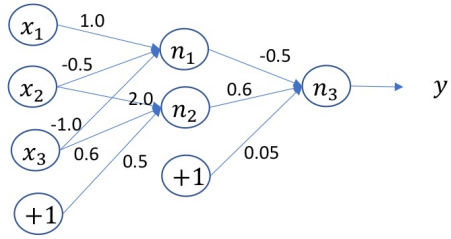


Figure 1

1. Figure 1 shows a two-layer feedforward neural network receiving 3-dimensional inputs $(x_1, x_2, x_3) \in \boldsymbol{R}^3$. The connection weights and biases of the neurons $n_1, n_2,$ and n_3 are as indicated in the figure. The hidden-layer neurons have activation functions given by $g(u) = \frac{1.0}{1+e^{-0.5u}}$ where u denotes the synaptic input to the neuron. The activation function $f(u)$ of the output neuron is a ReLU function: $f(u) = \max\{0,\ u\}$.

   a. Write weight vectors and biases connected to individual neurons, and the weight matrix and bias vector connected to the hidden layer.

   b. Find the synaptic inputs and activations of the neurons for the following input signals:

      (i) (1.0, –0.5, 1.0) (ii) (-1.0, 0.0, –2.0) (iii) (2.0, 0.5, –1.0).

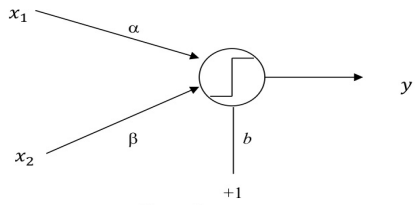


Figure 2

2. Two input binary neuron shown in figure 2 has a unit step activation function with a bias $b = 0.5$ and receives two-dimensional input $(x_1, x_2) \in \boldsymbol{R}^2$.

   (a) Find the space of possible values of weights (α, β) if the neuron is
   (i) ON for input (1.0, 1.0)
   (ii) ON for input (0.5, -1.0)
   (iii) OFF for input (2.0, -0.5).

   (b) Indicate the weight space in 2-D α-β plot and show that (-0.2, 0.2) is in this space.

3. The network shown in figure 3 consists of neurons having threshold activation functions and receives three-bit binary patterns $(x_1, x_2, x_3) \in \{0,1\}^3$. By analyzing the outputs for all possible three-bit input patterns, determine the logic function that the network implements. All unlabeled weights shown in figure 3 are of unity weight.

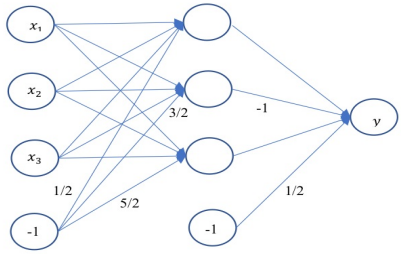


Figure 3

TUT 2

Jupyter 34781562

1. Train a linear neuron to learng the mapping from input $\boldsymbol{x} \in \boldsymbol{R}^3$ to output y from the following examples:

	$\boldsymbol{x} = (x_1, x_2, x_2)$	y
1	(0.09 −0.44 −0.15)	−2.57
2	(0.69 −0.99 −0.76)	−2.97
3	(0.34 0.65 −0.73)	0.96
4	(0.15 0.78 −0.58)	1.04
5	(−0.63 −0.78 −0.56)	−3.21
6	(0.96 0.62 −0.66)	1.05
7	(0.63 −0.45 −0.14)	−2.39
8	(0.88 0.64 −0.33)	0.66

(a) Show one iteration of learning of the neuron with
   i. Stochastic gradient descent learning
   ii. Gradient descent learning

Initialize the weights as $\begin{pmatrix} 0.77 \\ 0.02 \\ 0.63 \end{pmatrix}$ and biases to 0.0, and use a leaning factor $\alpha = 0.01$.

(b) Plot the learning curves (mean square error vs. epochs) until convergence. Determine the learned weights and biases.

(c) Find the predicted values y of training inputs after the training.

2. Two-dimensional training patterns (inputs) to design a dichotomizer are given as:

$$\mathbf{X}_1 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}; \quad \mathbf{X}_2 = \begin{bmatrix} 7 \\ 3 \end{bmatrix}; \quad \mathbf{X}_3 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}; \quad \mathbf{X}_4 = \begin{bmatrix} 5 \\ 4 \end{bmatrix}; \qquad \textit{Class 1}$$

$$\mathbf{X}_5 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad \mathbf{X}_6 = \begin{bmatrix} -1 \\ -3 \end{bmatrix}; \quad \mathbf{X}_7 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}; \quad \mathbf{X}_8 = \begin{bmatrix} -3 \\ 0 \end{bmatrix}; \qquad \textit{Class 2}$$

(a) Determine whether the two classes of patterns are linearly separable.

(b) Find the center of gravity of patterns in each class. Show that a linear decision boundary passing perpendicularly through the middle point of the line joining the two centroids is given by:

$$6.5x_1 + 2.5x_2 - 14.5 = 0$$

(c) Design a discrete perceptron having the decision boundary as in part (b) for the classification and show that the perceptron separates the points perfectly.

(d) Determine the classes identified by the neuron for following input patterns:

$$\binom{4}{2}, \binom{0}{5}, \binom{36/13}{0}$$

3. Use 'make_blobs' function from **sklearn**.datasets to create 100 samples of two Gaussian distributed classes for 3-dimensional inputs:

   from **sklearn**.datasets import make_blobs

   Assume each class has a standard deviation = 5.0.

   Use torch.nn.BCELoss() and torch.autograd() functions to train a logistic neuron to separate the two classes.
   Find the classification error at convergence and plot the decision boundary.

4. Train a perceptron to learn the following function ϕ:

$$\phi(x, y) = 1.5 + 3.3x - 2.5y + 1.2xy$$

for $0 \leq x, y \leq 1$.

(a) Sample 100 data points randomly from the input space to create a training dataset.
(b) Use the gradient descent algorithm to train the perceptron
(c) Compute the training error and plot the function approximated by the perceptron.
(d) Show how a linear neuron can be used to predict the above function
(e) Compare the results of approximations above by the linear neuron and the perceptron

1. Train a softmax layer of neurons to perform the following classification, given the inputs $\boldsymbol{x} = (x_1, x_2)$ and target class labels d:

(x_1, x_2)	(0 4)	(-1 3)	(2 3)	(-2 2)	(0 2)	(1 2)	(-1 2)	(-3 1)	(-1 1)
d	A	A	A	B	B	A	B	B	B

(x_1, x_2)	(2 1)	(4 1)	(-2 0)	(1 0)	(3 0)	(-3 -1)	(-2 -1)	(2 -1)	(4 -1)
d	C	C	B	C	C	B	B	C	C

   (a) Show one iteration of gradient descent learning at a learning factor 0.05.
   Initialize the weights to $\begin{pmatrix} 0.88 & 0.08 & -0.34 \\ 0.68 & -0.39 & -0.19 \end{pmatrix}$ and biases to zero
   (b) Find the weights and biases at the convergence of learning
   (c) Indicate the probabilities that the network predicts the classes of trained patterns.
   (d) Plot the decision boundaries separating the three classes.

2. Use mini-batch gradient decent learning to train a softmax layer to classify Iris dataset (https://archive.ics.uci.edu/ml/datasets/Iris). The dataset contains 150 data points. Use 90 data points for training the classifier and the remaining 60 data points for testing. Plot cross-entropies and classification accuracies against epochs for both train and test data. Set learning rate = 0.1, batch size = 16, and number of epochs = 1000.

   You can use the following python commands to load Iris data:

```
from sklearn import datasets
iris = datasets.load_iris()
```

   Repeat the classification with batch sizes = 2, 4, 8, 16, 24, 32, and 64, and compare the accuracies and the times taken for an epoch at different batch sizes.

3. Read the Linnerud dataset from sklearn.datasets package by using

```
from sklearn.datasets import load_linnerud
```

   The Linnerud dataset is a multi-output regression dataset consisting of three exercise (data) and three physiological variables (targets) collected from twenty middle-aged men in a fitness club:
   physiological - Weight, Waist and Pulse
   exercise - Chins, Situps and Jumps.

   Divide the dataset into train and test partitions at 0.75:0.25 ratio and train a perceptron layer to predict physiological data from exercise variables by implementing with
   a) direct gradients
   b) 'autograd' functions available in pytorch

Draw learning curves and find mean square error and R^2 values of prediction. Which physiological variable can be better predicted by exercise data?

Compare the time-taken for a weight update and the number of epochs required for convergence for (a) and (b).

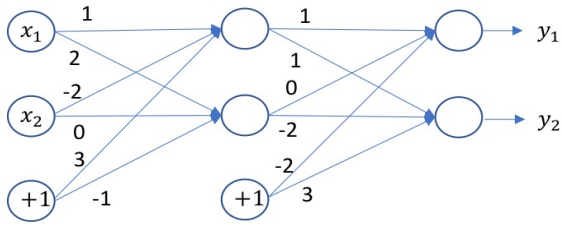


Figure 1

1. The two-layer feedforward perceptron network shown in figure 1 has weights and biases initialized as indicated and receives 2-dimensional inputs (x_1, x_2). The network is to respond with $\boldsymbol{d}_1 = \begin{pmatrix}0\\1\end{pmatrix}$ and $\boldsymbol{d}_2 = \begin{pmatrix}1\\0\end{pmatrix}$ for input patterns $\boldsymbol{x}_1 = \begin{pmatrix}1.0\\3.0\end{pmatrix}$ and $\boldsymbol{x}_2 = \begin{pmatrix}-2.0\\-2.0\end{pmatrix}$, respectively.

   Analyse a single feedforward and feedback step for gradient decent learning of the two patterns by doing the following:

   (a) Find the weight matrix $\boldsymbol{W}$ to the hidden-layer and weight matrix $\boldsymbol{V}$ to the output-layer, and the corresponding biases.
   (b) Calculate the synaptic input $\boldsymbol{z}$ and output $\boldsymbol{h}$ of the hidden-layer, and the synaptic input $\boldsymbol{u}$ and output $\boldsymbol{y} = (y_1, y_2)$ of the output layer.
   (c) Find the mean square error cost J between the outputs and targets.
   (d) Calculate the gradients $\boldsymbol{\nabla}_u J$ and $\boldsymbol{\nabla}_z J$ at the output-layer and the hidden-layer, respectively.
   (e) Compute the new weights and biases.
   (f) Write a program to continue iterations until convergence and find the final weights and biases.

   Assume a learning rate of 0.05.

   Repeat above (a) – (f) for stochastic gradient decent learning.

2. A feedforward neural network with one hidden layer is to perform the following classification:

class	inputs
A	$(1.0, 1.0), (0.0, 1.0)$
B	$(3.0, 4.0), (2.0, 2.0)$
C	$(2.0, -2.0), (-2.0, -3.0)$

The network has a hidden layer consisting of three perceptrons and a softmax output layer.

Initialize the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ to the hidden layer, and the weights $\boldsymbol{V}$ and biases $\boldsymbol{c}$ to the output layer as follows:

$$\boldsymbol{W} = \begin{pmatrix} -0.10 & 0.97 & 0.18 \\ -0.70 & 0.38 & 0.93 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

$$\boldsymbol{V} = \begin{pmatrix} 1.01 & 0.09 & -0.39 \\ 0.79 & -0.45 & -0.22 \\ 0.28 & 0.96 & -0.07 \end{pmatrix}, \boldsymbol{c} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

Show one iteration of gradient descent learning and plot the learning curves until convergence at a learning rate $\alpha = 0.1$.

Determine the weights and biases at convergence.

Find the class labels predicted by the trained network for patterns:

$\boldsymbol{x}_1 = \begin{pmatrix} 2.5 \\ 1.5 \end{pmatrix}$ and $\boldsymbol{x}_2 = \begin{pmatrix} -1.5 \\ 0.5 \end{pmatrix}$

3. Design a deep neural network consisting of two ReLU hidden layers to approximate the following function:

$$\phi(x, y) = 0.8x^2 - y^3 + 2.5xy$$

for $-1.0 \leq x, y \leq 1.0$.

Use 10 neurons and 5 neurons at first and second hidden layers, respectively, and a linear neuron at the output layer.

(a) Divide the input space equally into square regions of size 0.25×0.25 and use grid points as data points to learn the function ϕ.
(b) Train the network using gradient decent learning at learning rate $\alpha = 0.01$ and plot the learning curve (mean square error vs. iterations) and the predicted data points.
(c) Compare the behaviour of learning by plotting learning curves at rates $\alpha = 0.005, 0.001, 0.01, \text{and } 0.05$.

1. The first hidden layer of a convolution neural network (CNN) has a convolution layer consisting of two feature maps with filters $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$ and biases = 0.1, and neurons having sigmoid activation functions, and a pooling layer with a pooling windows of size 2x2:

$$\boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \text{ and } \boldsymbol{w}_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

The input layer is of 6x6 size and receives an input image $\boldsymbol{I}$:

$$\boldsymbol{I} = \begin{pmatrix} 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

   a. Find the outputs at the first convolution layer if
      i. padding = 0 and strides = $(1, 1)$.
      ii. padding = 1 and strides = $(2, 2)$.
   b. Find the outputs at the first pooling layer for Part (a), assuming strides of $(2, 2)$ and pooling is
      i. max pooling
      ii. mean pooling

2. Given '3wolfmoon.jpg' color image of size 639×516.
   a. Initialize weights and biases of a convolutional layer with two kernels of size 9×9. Note that the input image is in color and has three channels.
   b. Display the feature maps at the convolution layer, assuming sigmoid activation functions. Use VALID padding and strides = 1.
   c. Display the outputs of a mean pooling layer with a pooling window size 5×5 and strides = 5.

3. Design a CNN with one hidden layer to recognize digit images in MNIST database:

   http://yann.lecun.com/exdb/mnist/

   The convolution layer consists of 25 filters of dimensions 9x9 and the pooling layer has a pooling window size 4x4. Assume VALID padding and default strides for both convolution and pooling layer. Train the network with mini batch gradient decent learning with learning factor $\alpha = 10^{-3}$ and batch size = 128.

   Plot
   a. The training and test errors against learning epochs.
   b. Final filter weights
   c. Feature maps at the convolution and pooling layers for a representative test pattern
   d. Repeat training by introducing decay parameter $\beta = 10^{-6}$ and momentum term with $\gamma = 0.5$, and compare the learning curves

Neuron Layers

SC4001 – Tutorial 3

1. Design a softmax layer of neurons to perform the following classification, given the inputs $\boldsymbol{x} = (x_1, x_2)$ and target class labels d:

(x_1, x_2)	(0 4)	(-1 3)	(2 3)	(-2 2)	(0 2)	(1 2)	(-1 2)	(-3 1)	(-1 1)
d	A	A	A	B	B	A	B	B	B

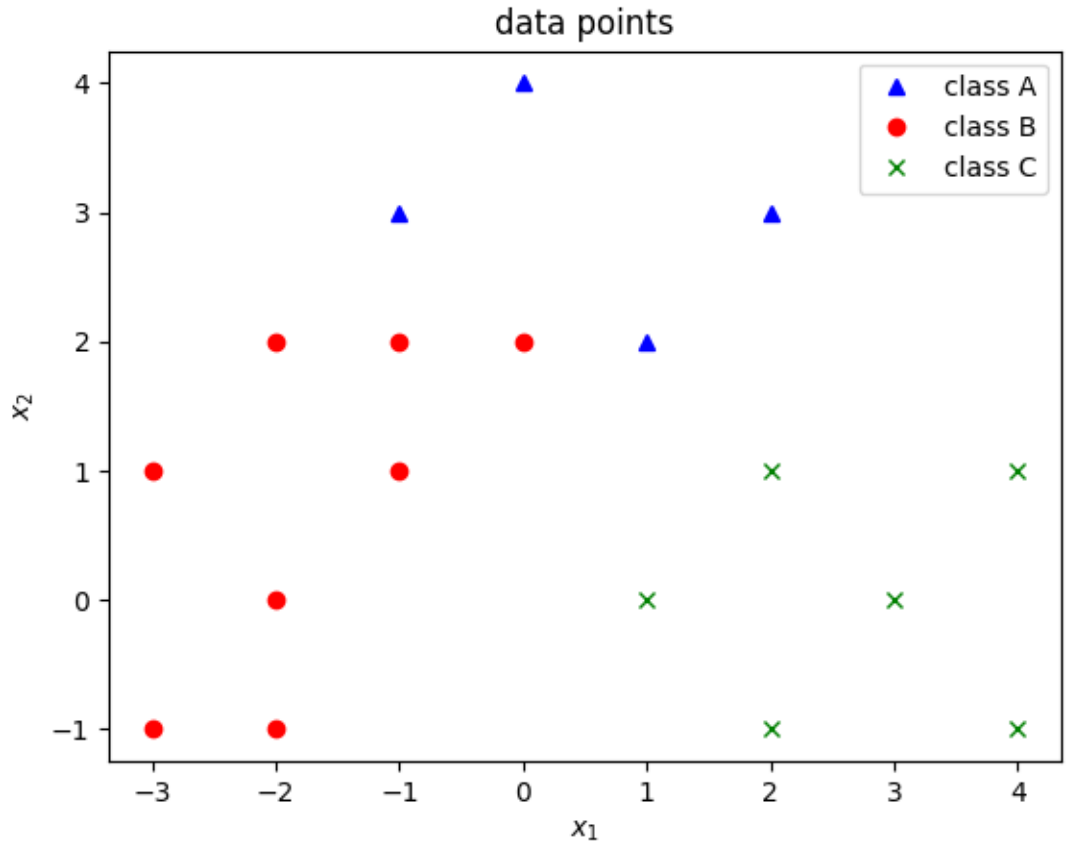
(x_1, x_2)	(2 1)	(4 1)	(-2 0)	(1 0)	(3 0)	(-3 -1)	(-2 -1)	(2 -1)	(4 -1)
d	C	C	B	C	C	B	B	C	C

(a) Show one iteration of gradient descent learning at a learning factor 0.05.
Initialize the weights to $\begin{pmatrix} 0.88 & 0.08 & -0.34 \\ 0.68 & -0.39 & -0.19 \end{pmatrix}$ and biases to zero

(b) Find the weights and biases at convergence of learning

(c) Indicate the probabilities that the network predicts the classes of trained patterns.

(d) Plot the decision boundaries separating the three classes.


data points
class A
class B
class C
4
3
2
1
0
−1
−3
−2
−1
0
1
2
3
4
x_2
x_1

GD for Softmax layer

Given training set $(\boldsymbol{X}, \boldsymbol{d})$

Set learning rate α

Initialize $\boldsymbol{W}$ and $\boldsymbol{b}$

Iterate until convergence:

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

$$f(\boldsymbol{U}) = \frac{e^{\boldsymbol{U}}}{\sum_{k=1}^{K} e^{\boldsymbol{U_k}}}$$

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

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

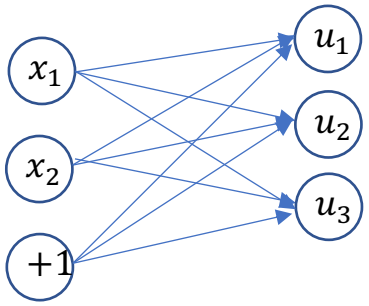
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P$$

Labels for classes:

$$Class\ A \rightarrow 1, Class\ B \rightarrow 2, Class\ C \rightarrow 3$$

The data matrix and target vector:

$$\boldsymbol{X} = \begin{pmatrix} 0 & 4 \\ -1 & 3 \\ 2 & 3 \\ -2 & 2 \\ 0 & 2 \\ 1 & 2 \\ -1 & 2 \\ \vdots & \vdots \\ 4 & -1 \end{pmatrix}, \quad \boldsymbol{d} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \\ \vdots \\ 3 \end{pmatrix}, \quad \boldsymbol{K} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix}$$



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

$$\boldsymbol{Y} = \underset{k}{\operatorname{argmax}}\, f(\boldsymbol{U})$$

Learning rate $\alpha = 0.05$.

Initialize weights and biases:

$$\boldsymbol{W} = \begin{pmatrix} 0.88 & 0.08 & -0.34 \\ 0.68 & -0.39 & -0.19 \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

Epoch 1:

$$\boldsymbol{U} = \boldsymbol{XW} + \boldsymbol{B} = \begin{pmatrix} 0 & 4 \\ -1 & 3 \\ 2 & 3 \\ -2 & 2 \\ \vdots & \vdots \\ 4 & -1 \end{pmatrix} \begin{pmatrix} 0.88 & 0.08 & -0.34 \\ 0.68 & -0.39 & -0.19 \end{pmatrix} + \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ \vdots & \vdots & \vdots \\ 0.0 & 0.0 & 0.0 \end{pmatrix} = \begin{pmatrix} 2.72 & -1.54 & -0.75 \\ 1.17 & -1.23 & -0.23 \\ 3.8 & -1.0 & -1.23 \\ -0.39 & -0.93 & 0.30 \\ \vdots & \vdots & \vdots \\ 2.82 & 0.71 & -1.16 \end{pmatrix}$$

$$\boldsymbol{U} = \begin{pmatrix} 2.72 & -1.54 & -0.75 \\ 1.17 & -1.23 & -0.23 \\ 3.8 & -1.0 & -1.23 \\ -0.39 & -0.93 & 0.30 \\ \vdots & \vdots & \vdots \\ 2.82 & 0.71 & -1.16 \end{pmatrix}$$

$$f(u_i) = \frac{e^{u_i}}{\sum_{k=1}^{K} e^{u_k}}$$

$$f(\boldsymbol{U}) == \begin{pmatrix} \frac{e^{2.72}}{e^{2.72} + e^{-1.54} + e^{-0.75}} & \frac{e^{-1.54}}{e^{2.72} + e^{-1.54} + e^{-0.75}} & \frac{e^{-0.75}}{e^{2.72} + e^{-1.54} + e^{-0.75}} \\ \frac{e^{1.17}}{e^{1.17} + e^{-1.23} + e^{-0.23}} & \frac{e^{-1.23}}{e^{1.17} + e^{-1.23} + e^{-0.23}} & \frac{e^{-0.23}}{e^{1.17} + e^{-1.23} + e^{-0.23}} \\ \vdots & \vdots & \vdots \\ \frac{e^{2.82}}{e^{2.82} + e^{0.71} + e^{-1.16}} & \frac{e^{0.71}}{e^{2.82} + e^{0.71} + e^{-1.16}} & \frac{e^{-1.16}}{e^{2.82} + e^{0.71} + e^{-1.16}} \end{pmatrix} = \begin{pmatrix} 0.96 & 0.01 & 0.03 \\ 0.75 & 0.07 & 0.18 \\ \vdots & \vdots & \vdots \\ 0.88 & 0.11 & 0.02 \end{pmatrix}$$

$$\boldsymbol{y} = \underset{k}{\operatorname{argmax}}\, f(\boldsymbol{U}) = \underset{k}{\operatorname{argmax}} \begin{pmatrix} 0.96 & 0.01 & 0.03 \\ 0.75 & 0.07 & 0.18 \\ 0.99 & 0.01 & 0.01 \\ 0.28 & 0.16 & 0.56 \\ \vdots & \vdots & \vdots \\ 0.88 & 0.11 & 0.02 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 3 \\ \vdots \\ 1 \end{pmatrix}$$

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

$$Classification\ error = \sum_{p=1}^{P} 1(\boldsymbol{y} \neq \boldsymbol{d}) = 14$$

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

$$= -\log(0.96) - \log(0.75) - \log(0.99) - \log(0.16) \cdots - \log(0.02)$$

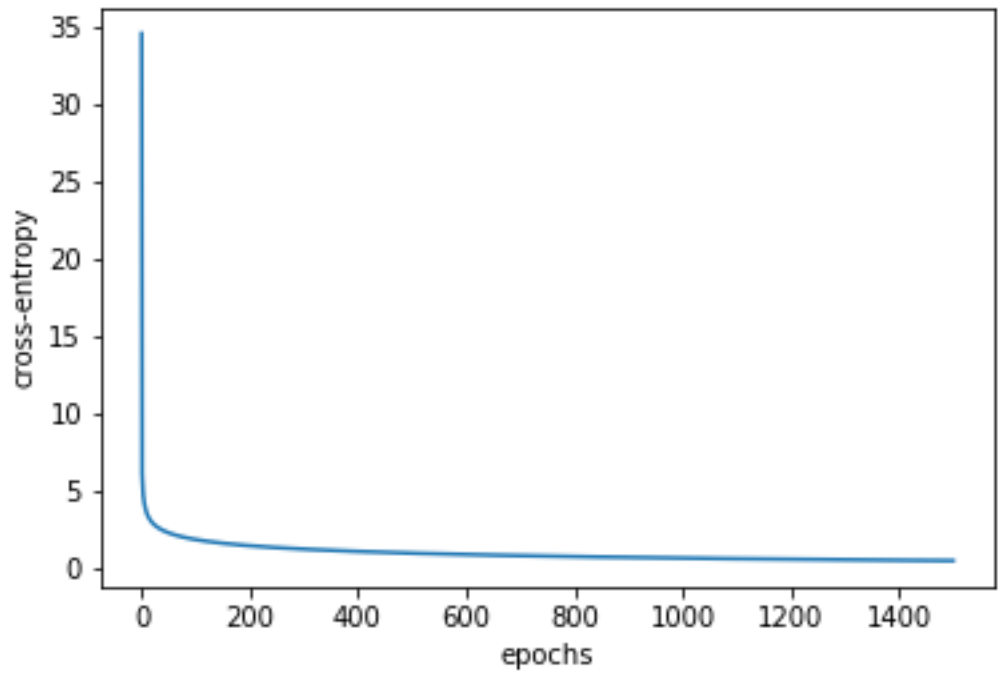
$$= 34.36$$

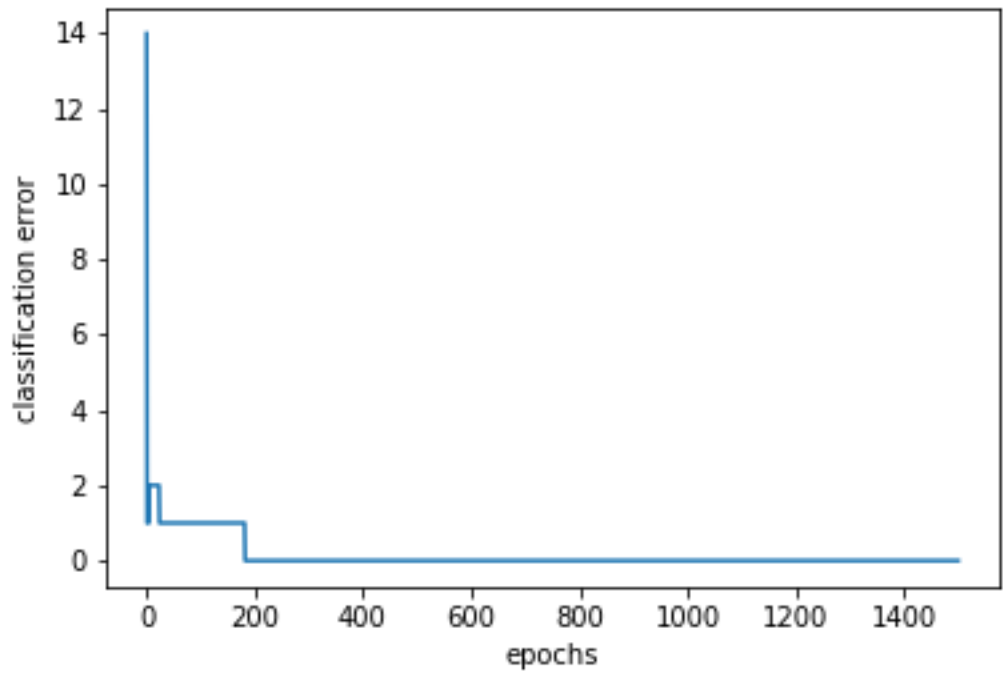
$$\nabla_U J = -\big(\boldsymbol{K} - f(\boldsymbol{U})\big) = -\left(\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0.96 & 0.01 & 0.03 \\ 0.75 & 0.07 & 0.18 \\ \vdots & \vdots & \vdots \\ 0.88 & 0.11 & 0.02 \end{pmatrix}\right) = \begin{pmatrix} -0.04 & 0.01 & 0.03 \\ -0.25 & 0.07 & 0.18 \\ \vdots & \vdots & \vdots \\ 0.88 & 0.11 & -0.98 \end{pmatrix}$$

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{U}} J = \begin{pmatrix} 0.88 & 0.08 & -0.34 \\ 0.68 & -0.39 & -0.19 \end{pmatrix} - 0.05 \begin{pmatrix} 0 & -1 & \cdots & 4 \\ 4 & 3 & \cdots & -1 \end{pmatrix} \begin{pmatrix} -0.04 & 0.01 & 0.03 \\ -0.25 & 0.07 & 0.18 \\ \vdots & \vdots & \vdots \\ 0.88 & 0.11 & -0.98 \end{pmatrix}$$

$$= \begin{pmatrix} 0.28 & -0.54 & 0.89 \\ 0.54 & -0.12 & -0.31 \end{pmatrix}$$

$$\boldsymbol{b} = \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix} + 0.05 \begin{pmatrix} -0.04 & 0.01 & 0.03 \\ -0.25 & 0.07 & 0.18 \\ \vdots & \vdots & \vdots \\ 0.88 & 0.11 & -0.98 \end{pmatrix}^T \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} -0.32 \\ 0.27 \\ 0.06 \end{pmatrix}$$


35
30
25
20
15
10
5
0
cross-entropy
0
200
400
600
800
1000
1200
1400
epochs


14
12
10
8
6
4
2
0
classification error
0
200
400
600
800
1000
1200
1400
epochs

At convergence:

$$\boldsymbol{W} = \begin{pmatrix} -0.15 & -3.41 & 4.18 \\ 5.27 & -1.02 & -4.15 \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} -7.82 \\ 5.81 \\ 2.02 \end{pmatrix}$$

Entropy = 0.58
Classification Error = 0

At convergence:

$$\boldsymbol{X} = \begin{pmatrix} -1 & 2 \\ 0 & 4 \\ -1 & 3 \\ 0 & 2 \\ 3 & 0 \\ -2 & -1 \\ 4 & 1 \\ 1 & 2 \\ 2 & -1 \\ 2 & 3 \\ 2 & 1 \\ -2 & 0 \\ -3 & -1 \\ 1 & 0 \\ -1 & 1 \\ 4 & -1 \\ -3 & 1 \\ -2 & 2 \end{pmatrix} \quad f(\boldsymbol{U}) = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.88 & 0.12 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.26 & 0.74 & 0.0 \\ 0.89 & 0.1 & 0.0 \\ 0.01 & 0.99 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.02 & 0.98 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}, \boldsymbol{Y} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \\ 2 \\ 2 \\ 3 \\ 3 \\ 2 \\ 3 \\ 3 \\ 2 \\ 2 \\ 3 \\ 3 \end{pmatrix}$$

Probabilities that input patterns belong to target classes are given in RED.

At convergence:

$$\boldsymbol{W} = \begin{pmatrix} -0.15 & -3.41 & 4.18 \\ 5.27 & -1.02 & -4.15 \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} -7.82 \\ 5.81 \\ 2.02 \end{pmatrix}$$

Synaptic inputs at the softmax layer for an input $\boldsymbol{x} = (x_1, x_2)$:

Neuron of class A, $u_1 = \boldsymbol{w}_1{}^T \boldsymbol{x} + b_1 = -0.15x_1 + 5.27x_2 - 7.82$
Neuron of class B, $u_2 = \boldsymbol{w}_2{}^T \boldsymbol{x} + b_2 = -3.41x_1 - 1.02x_2 + 5.81$
Neuron of class C, $u_3 = \boldsymbol{w}_3{}^T \boldsymbol{x} + b_3 = 4.18x_1 - 4.15x_2 + 2.02$

Decision boundaries:

Between class A and class B is given when $u_1 = u_2$

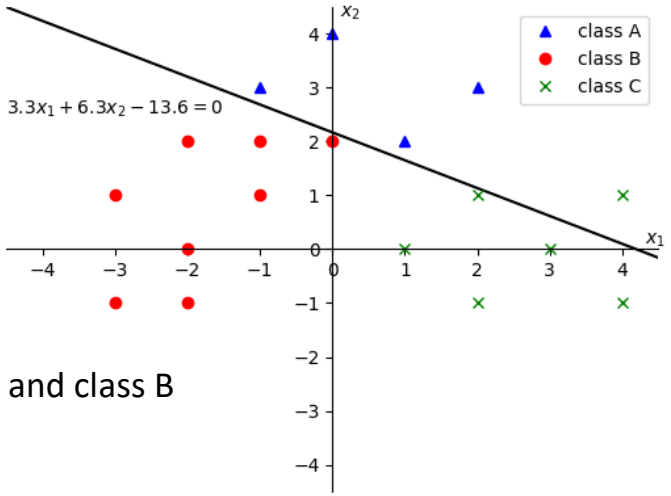
$$-0.15x_1 + 5.27x_2 - 7.82 = -3.41x_1 - 1.02x_2 + 5.81$$
$$3.25x_1 + 6.29x_2 - 13.63 = 0$$

Similarly, between class B and class C $u_2 = u_3$:

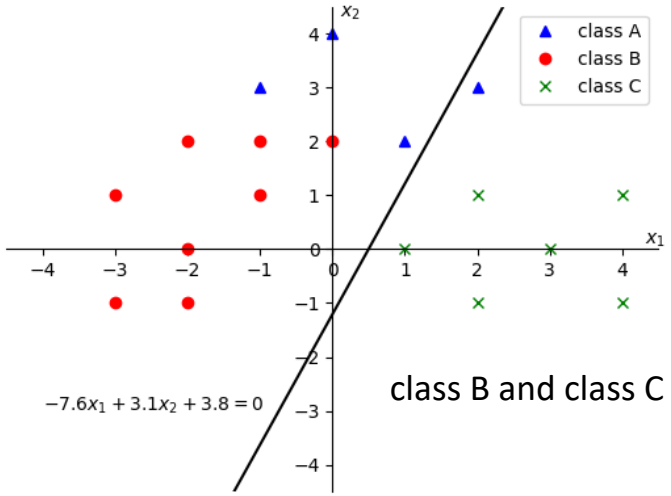
$$-7.59x_1 + 3.13x_2 + 3.79 = 0$$

between class A and class C $u_3 = u_1$:

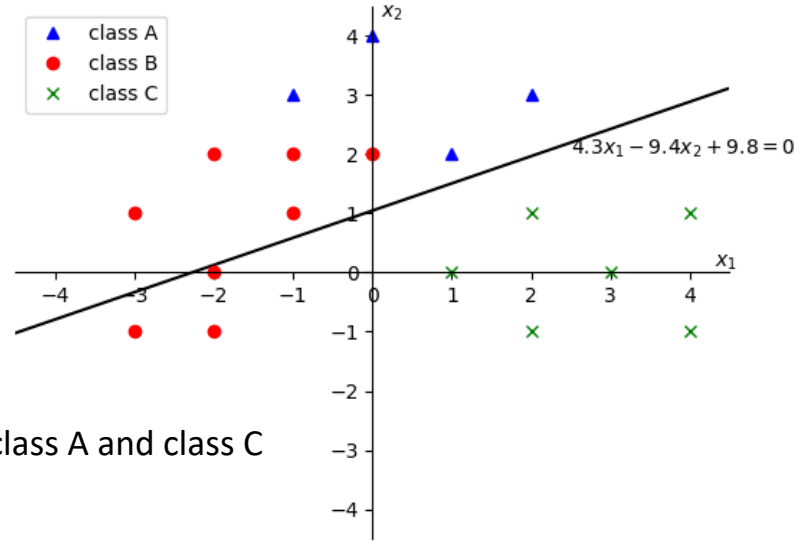
$$4.33x_1 - 9.42x_2 + 9.84 = 0$$



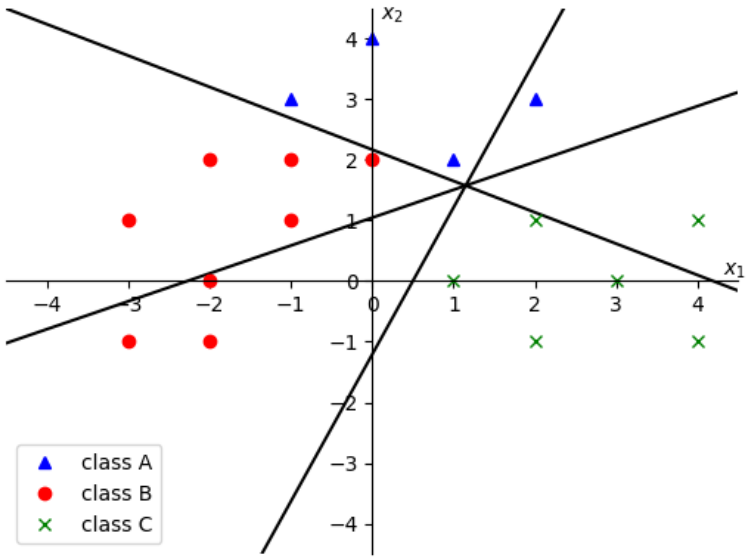
class A and class B



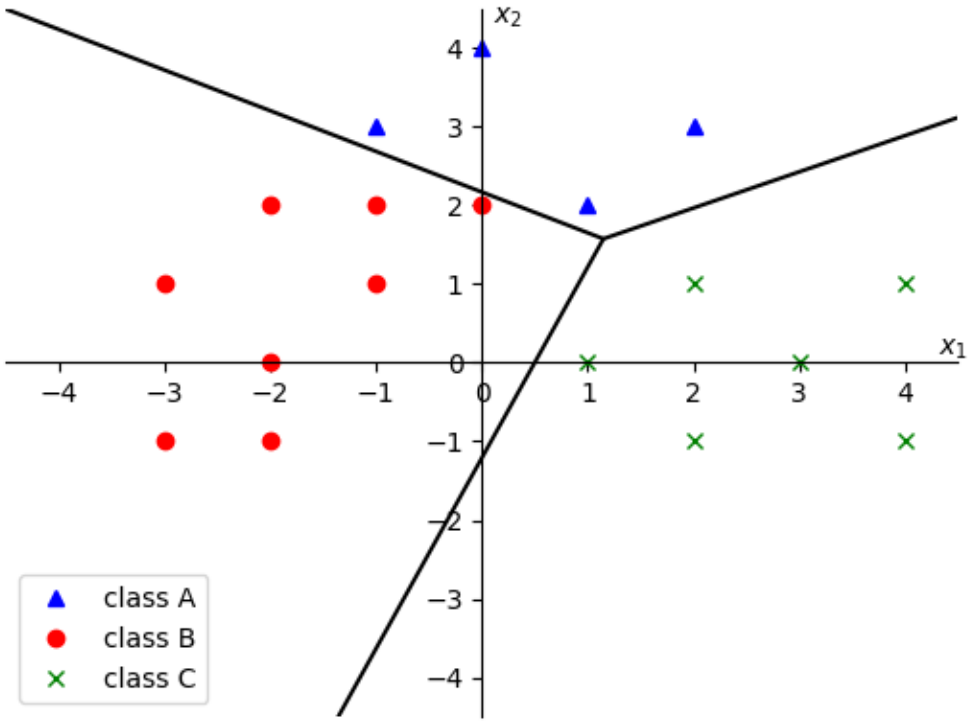
class B and class C



class A and class C



Decision boundaries leant by the softmax layer



3. Read the Linnerud dataset from sklearn.datasets package by using

```
from sklearn.datasets import load_linnerud
```

The Linnerud dataset is a multi-output regression dataset consisting of three exercise (data) and three physiological variables (targets) collected from twenty middle-aged men in a fitness club:

physiological - Weight, Waist and Pulse
exercise - Chins, Situps and Jumps.

Divide the dataset into train and test partitions at 0.75:0.25 ratio and train a perceptron layer to predict physiological data from exercise variables by implementing with

a) direct gradients
b) 'autograd' functions available in pytorch

Draw learning curves and find mean square error and R^2 values of prediction. Which physiological variable can be better predicted by exercise data?

Compare the time-taken for a weight update and the number of epochs required for convergence for (a) and (b).

```
from sklearn.datasets import load_linnerud

X, y = load_linnerud(return_X_y=True)
```

X

	chins	sit_ups	jumps
0	5.0	162.0	60.0
1	2.0	110.0	60.0
2	12.0	101.0	101.0
3	12.0	105.0	37.0
4	13.0	155.0	58.0

y

	weight	waist	pulse
0	191.0	36.0	50.0
1	189.0	37.0	52.0
2	193.0	38.0	58.0
3	162.0	35.0	62.0
4	189.0	35.0	46.0

Parameters of the data

targets **inputs**

	weight	waist	pulse	chins	sit_ups	jumps
count	20.000000	20.000000	20.00000	20.000000	20.000000	20.000000
mean	9.450000	145.550000	70.30000	178.600000	35.400000	56.100000
std	5.286278	62.566575	51.27747	24.690505	3.201973	7.210373
min	1.000000	50.000000	25.00000	138.000000	31.000000	46.000000
25%	4.750000	101.000000	39.50000	160.750000	33.000000	51.500000
50%	11.500000	122.500000	54.00000	176.000000	35.000000	55.000000
75%	13.250000	210.000000	85.25000	191.500000	37.000000	60.500000
max	17.000000	251.000000	250.00000	247.000000	46.000000	74.000000

Normalizing input variables such that $x' \sim N(0, 1)$.

```
# preprocess input and output data
from sklearn import preprocessing

# normal Gaussian scaling for inputs
X_scaler = preprocessing.StandardScaler().fit(X_train)
X_scaled = X_scaler.transform(X_train)
```

$$x' = \frac{x - \mu}{\sigma}$$

```
print(X_scaler.mean_)
print(X_scaler.var_)
```

```
[ 9.06666667 161.6 77.8 ]
[ 28.46222222 3795.17333333 2861.62666667]
```

Sclaing output variables such that $y' \sim [0, 1]$.

```
# linear scaling up to [0,1] for outputs
y_scaler = preprocessing.MinMaxScaler().fit(y_train)
y_scaled = y_scaler.transform(y_train)
```

$$y' = \underbrace{\left(\frac{1}{y_{max} - y_{min}}\right)}_{scale} y + \underbrace{\left(\frac{-y_{min}}{y_{max} - y_{min}}\right)}_{min}$$

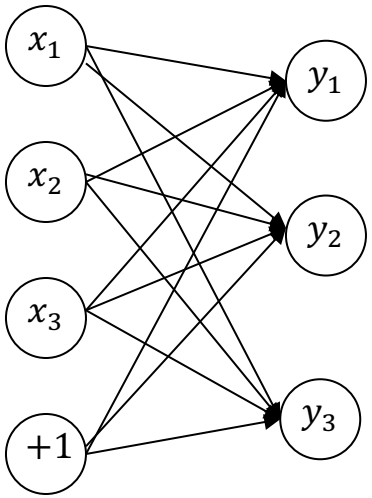
```
print(y_scaler.scale_)  #
print(y_scaler.min_)
```

```
[0.00917431 0.06666667 0.03571429]
[-1.26605505 -2.06666667 -1.64285714]
```

GD for a perceptron layer with direct gradients

Given a training dataset $(\boldsymbol{X}, \boldsymbol{D})$
Set learning parameter α
Initialize $\boldsymbol{W}$ and $\boldsymbol{b}$
Repeat until convergence:

$$\boldsymbol{U} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{B}$$
$$\boldsymbol{Y} = f(\boldsymbol{U})$$
$$\nabla_U J = -(\boldsymbol{D} - \boldsymbol{Y}) \cdot f'(\boldsymbol{U})$$
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{U}} J$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P$$



Implementing **Perceptron Layer** using pytorch libraries**:**

from torch **import** nn # torch nn API provides methods for build custom neural networks

nn.Module
- Base class for all neural network modules.
- Your models should also subclass this class.

nn.Linear
- applies a linear transformation $y = w^T x + b$

nn.Sigmoidal
- implements a perceptron layer

nn.Sequential
A sequential container. Modules will be added to it in the order they are passed in the constructor. The value a Sequential provides over manually calling a sequence of modules is that it allows treating the whole container as a single module

Implementing **MSE loss** and **SGD optim**izer

loss_fn = **nn.MSELoss**()
optimizer = **torch.optim.SGD**(model.parameters(), lr=lr)

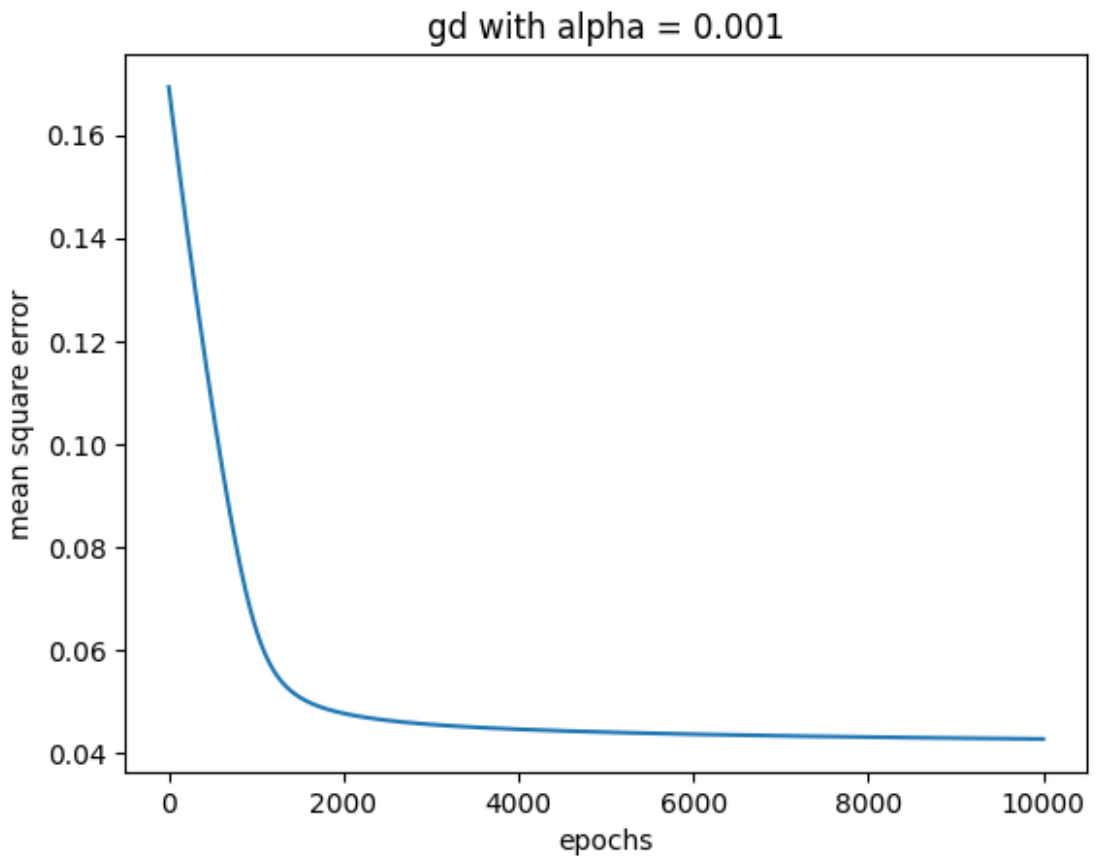
GD for a perceptron layer with PyTorch nn.Module class

```
# Perceptron layer class
class PerceptronLayer(nn.Module):
  def __init__(self, no_inputs, no_outputs):
        super().__init__()
        self.perceptron_layer = nn.Sequential(
            nn.Linear(no_inputs, no_outputs),
            nn.Sigmoid()
        )

  def forward(self, x):
        logits = self.perceptron_layer(x)
        return logits

# create an instance of the layer
no_inputs, no_outputs = 3, 3
model = PerceptronLayer(no_inputs, no_outputs)
```

GD for a perceptron layer with PyTorch autograd

```
loss_fn = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=.001)

no_epochs, lr = 50000, 0.001
for epoch in range(no_epochs):
    # Compute prediction and loss
    pred = model(torch.tensor(X_scaled, dtype=torch.float))
    loss = loss_fn(pred, torch.tensor(y_scaled, dtype=torch.float))

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

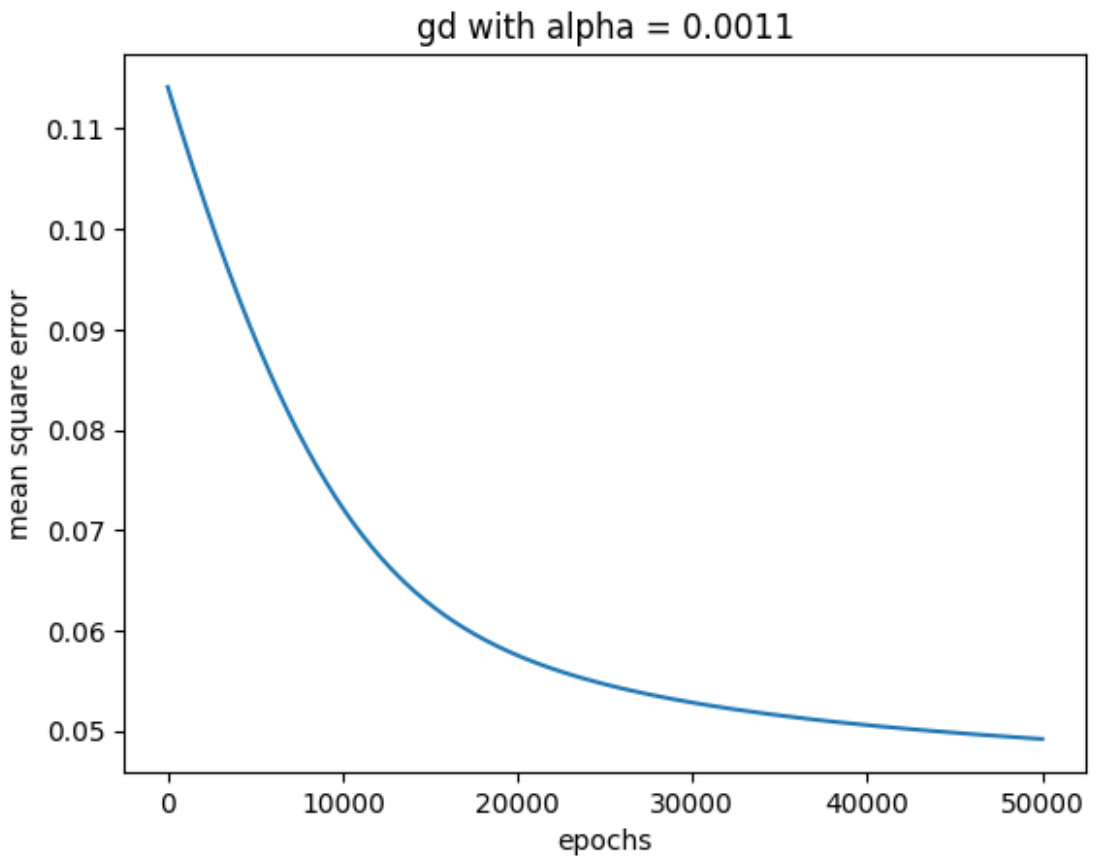
Direct gradients



Time for weight update = 0.074ms

Number of epochs = 10,000

Autograd



Time for weight update = 0.114ms

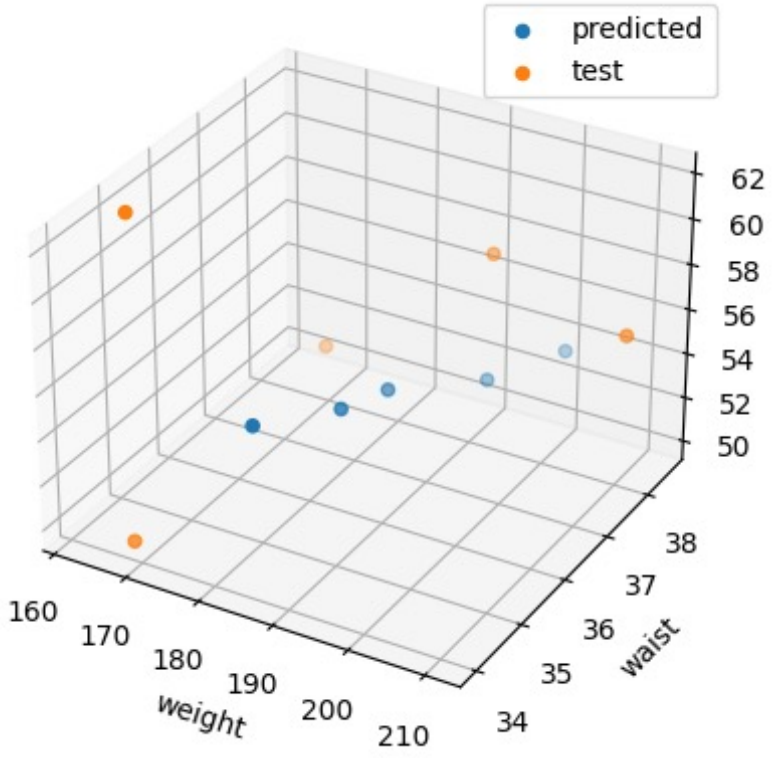
Number of epochs = 50,000

```
# scaling testing inputs
X_scaled = X_scaler.transform(X_test)

y_pred = model(X_scaled)

# scaling predicted outputs
y_scaled = y_scaler.inverse_transform(y_pred.detach().numpy())
```

```
# computing mean square error
from sklearn.metrics import mean_squared_error
rms = mean_squared_error(y_scaled, y_test, squared=True, multioutput='raw_values')
```

```
# computing R2
from sklearn.metrics import r2_score
r2 = r2_score(y_scaled, y_test,  multioutput='raw_values')
```

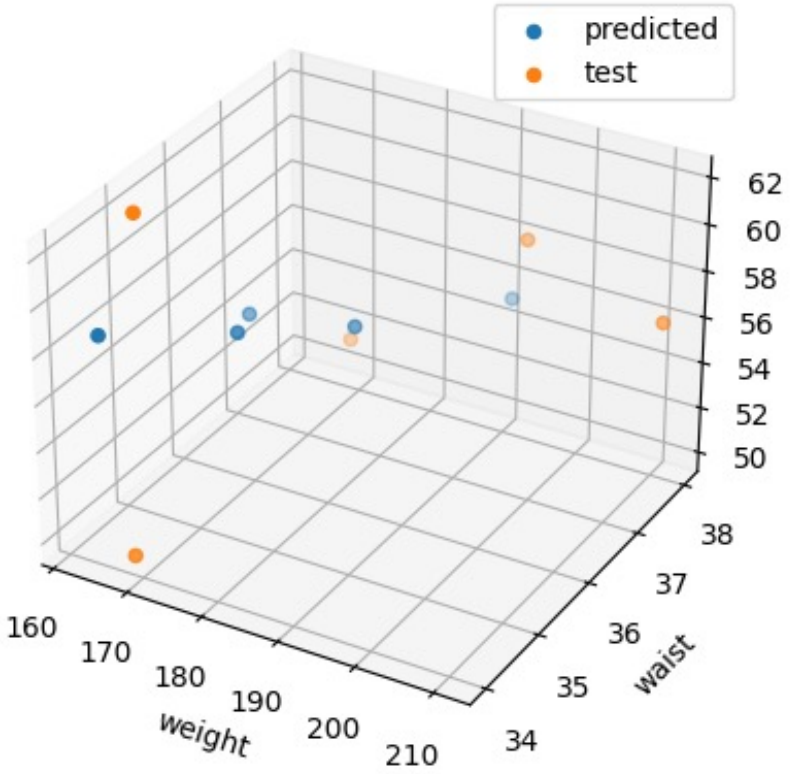
Direct gradients



MSE = [407.06558087, 1.21179368, 22.40310961]

R^2 = [-15.16134924, 0.53783585, -11.22642492]

Autograd



MSE = [356.95227178, 1.61016719, 21.11763412]

R^2 = [-3.18467906, 0.06741378, -11.90990362]

Exercise can best predict the waist!

R^2: coefficient of determination

Let $\hat{y}_i$ be the predicted value of data point y_i and the number of data points be n:

The **residual** sum of square error $SS_{res} = \sum_1^n (\hat{y}_i - y_i)^2$

If mean of data, $\bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$.

The **total** sum of squares (proportional to the variance), $SS_{tot} = \sum_1^n (\bar{y} - y_i)^2$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

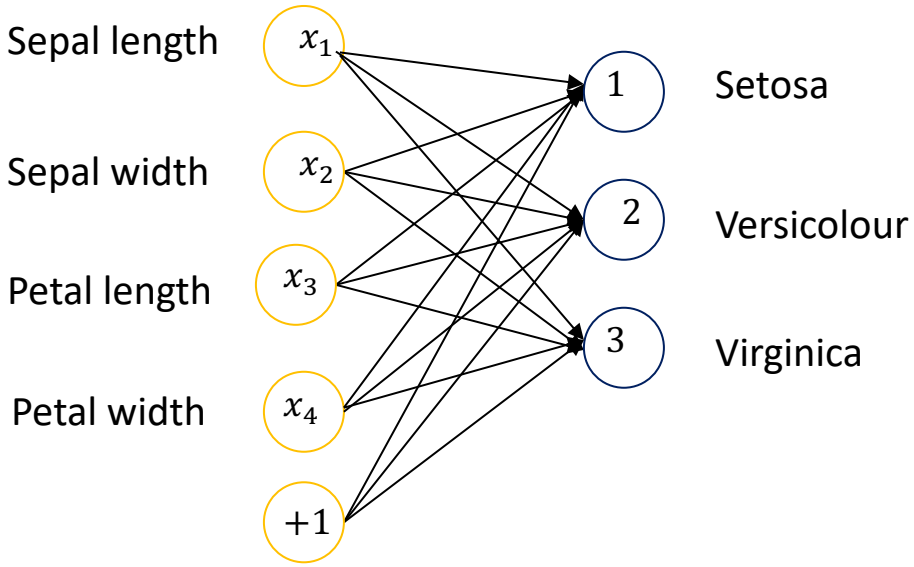
In the best case, the modelled value exactly match the observed value SS_{res}=0, then $R^2 = 1$.
The baseline model which predicts $\bar{y}$, then $R^2 = 0$.
The models that have worse prediction than baseline models have negative R^2.

2. Use mini-batch gradient decent learning to train a softmax layer to classify Iris dataset (https://archive.ics.uci.edu/ml/datasets/Iris). The dataset contains 150 data points. Use 90 data points for training the classifier and the remaining 60 data points for testing. Plot cross-entropies and classification accuracies against epochs for both train and test data. Set learning rate = 0.1, batch size = 16, and number of epochs = 1000.

   You can use the following python commands to load Iris data:

   ```
   from sklearn import datasets
   iris = datasets.load_iris()
   ```

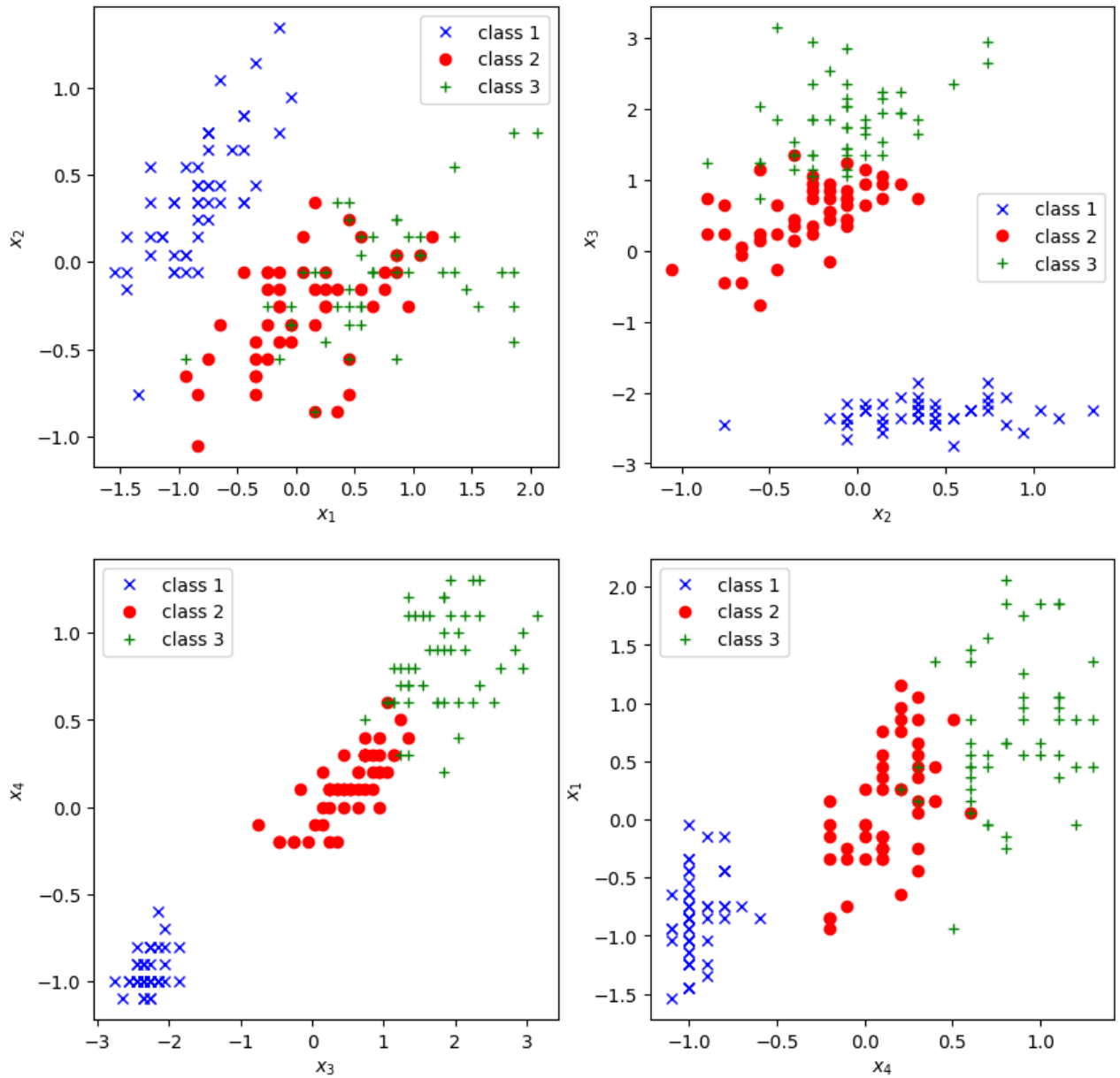
   Repeat the classification with batch sizes = 2, 4, 8, 16, 24, 32, and 64, and compare the accuracies and the times taken for an epoch at different batch sizes.



Four features and **three labels**

90 data points for training and **60** data points for testing

boundaries in 2-dimensional feature spaces



```
# datasets import
from sklearn import datasets
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
iris.data -= np.mean(iris.data, axis=0)  # mean correct data

# dataset split for train and test
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=2)
```

```
from torch import nn


class SoftmaxLayer(nn.Module):
  def __init__(self, no_inputs, no_outputs):
      super().__init__()
      self.softmax_layer = nn.Sequential(
        nn.Linear(no_inputs, no_outputs),     # applies a linear transformation y = w^T x + b
        nn.Softmax(dim=1)       # implements softmax; sum up across rows to 1.0
      )

  def forward(self, x):
      logits = self.softmax_layer(x)
      return logits
```

Mini-batch gradient descent
Batch size = 16, learning factor = 0.1

Torch **Dataset** and **Dataloader** utils provide easy function for mini-batch learning

```
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
```

We will be creating a subclass of **Dataset** class and using **Dataloaders** to implement **mini-batch gradient descent**

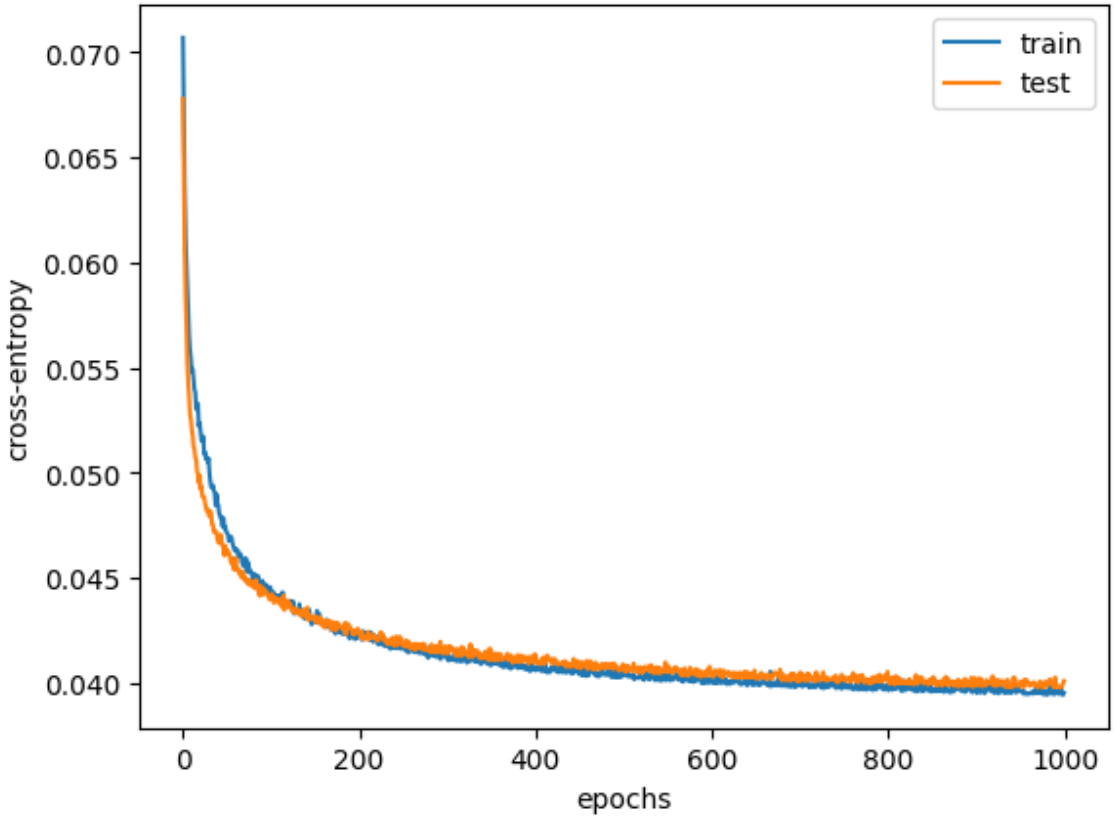
```
# create a Dataset class
# A custom Dataset class must implement three functions: __init__, __len__, and __getitem__.
class MyDataset(Dataset):
  def __init__(self, X, y):
    self.X =torch.tensor(X, dtype=torch.float)
    self.y =torch.tensor(y)

  def __len__(self):
    return len(self.y)

  def __getitem__(self,idx):
    return self.X[idx], self.y[idx]


# create Dataset objects for train and test data
train_data = MyDataset(x_train, y_train)
test_data = MyDataset(x_test, y_test)

# create DataLoader objects for train and test data
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
```

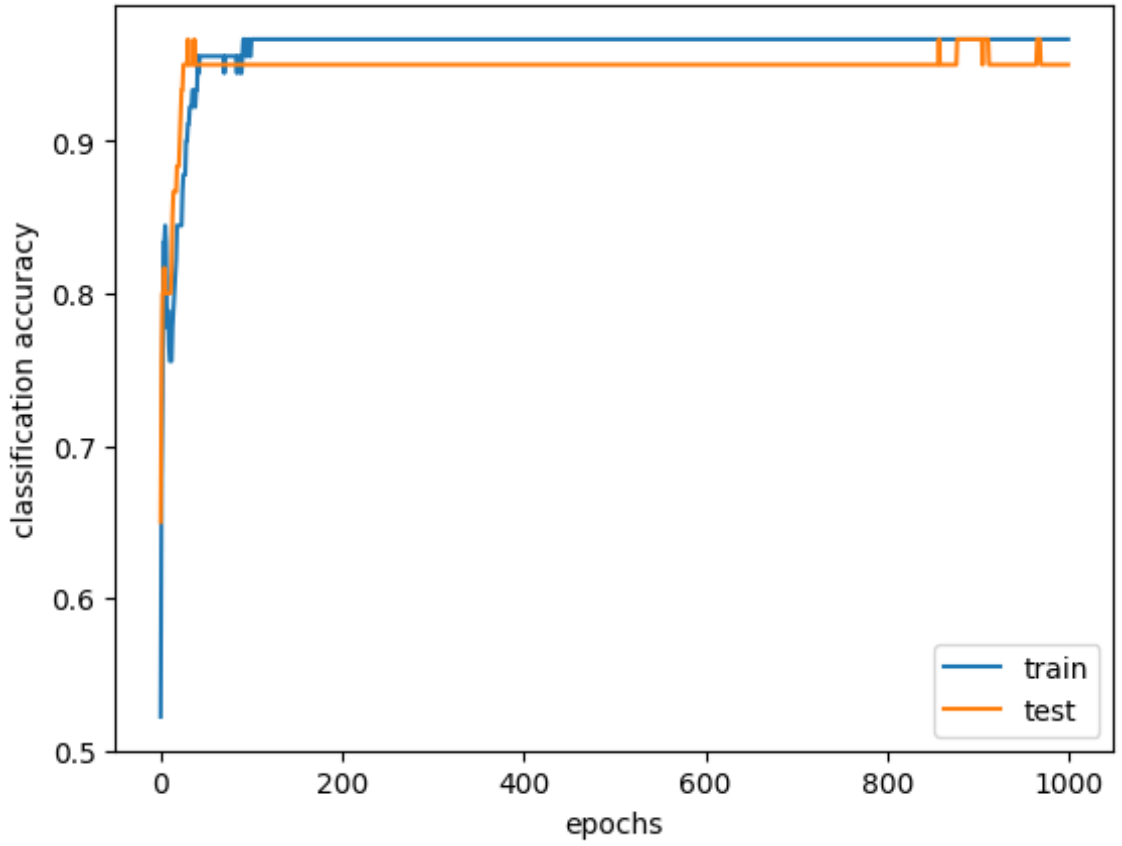
```
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)

    train_loss, correct = 0, 0
    for batch, (X, y) in enumerate(dataloader):

        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        optimizer.zero_grad().  #initialize gradient calculations
        loss.backward()          # compute gradients
        optimizer.step()         # execute one step of SGD

        train_loss += loss.item()
        correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    train_loss /= size
    correct /= size
    return train_loss, correct
```

```
def test_loop(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(dim=1) == y).type(torch.float).sum().item()

    test_loss /= size
    correct /= size

    return test_loss, correct
```

```
train_loss_, train_acc_, test_loss_, test_acc_ = [], [], [], []

for epoch in range(no_epochs):
    train_loss, train_acc = train_loop(train_dataloader, model, loss_fn, optimizer)
    test_loss, test_acc = test_loop(test_dataloader, model, loss_fn)

    train_loss_.append(train_loss), train_acc_.append(train_acc)
    test_loss_.append(test_loss), test_acc_.append(test_acc)
```

Learning curves at batch-size = 16





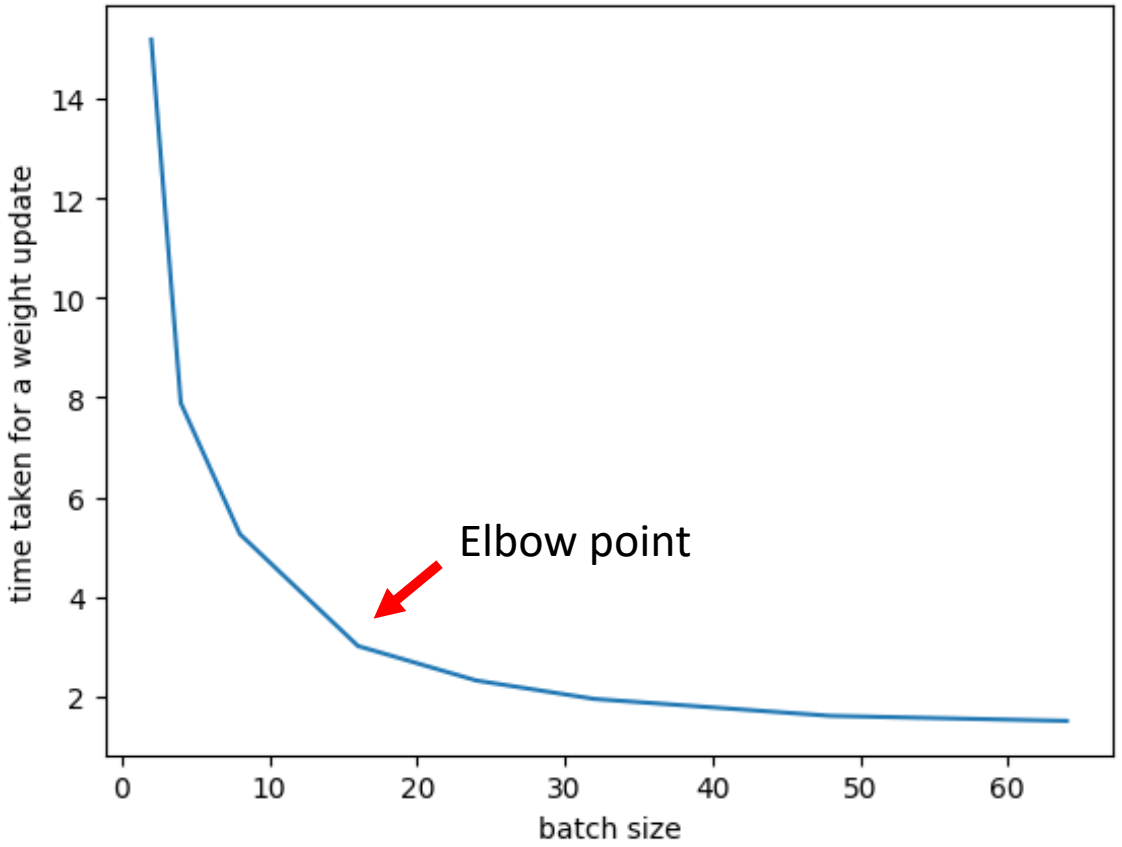
At convergence (1000 epochs):

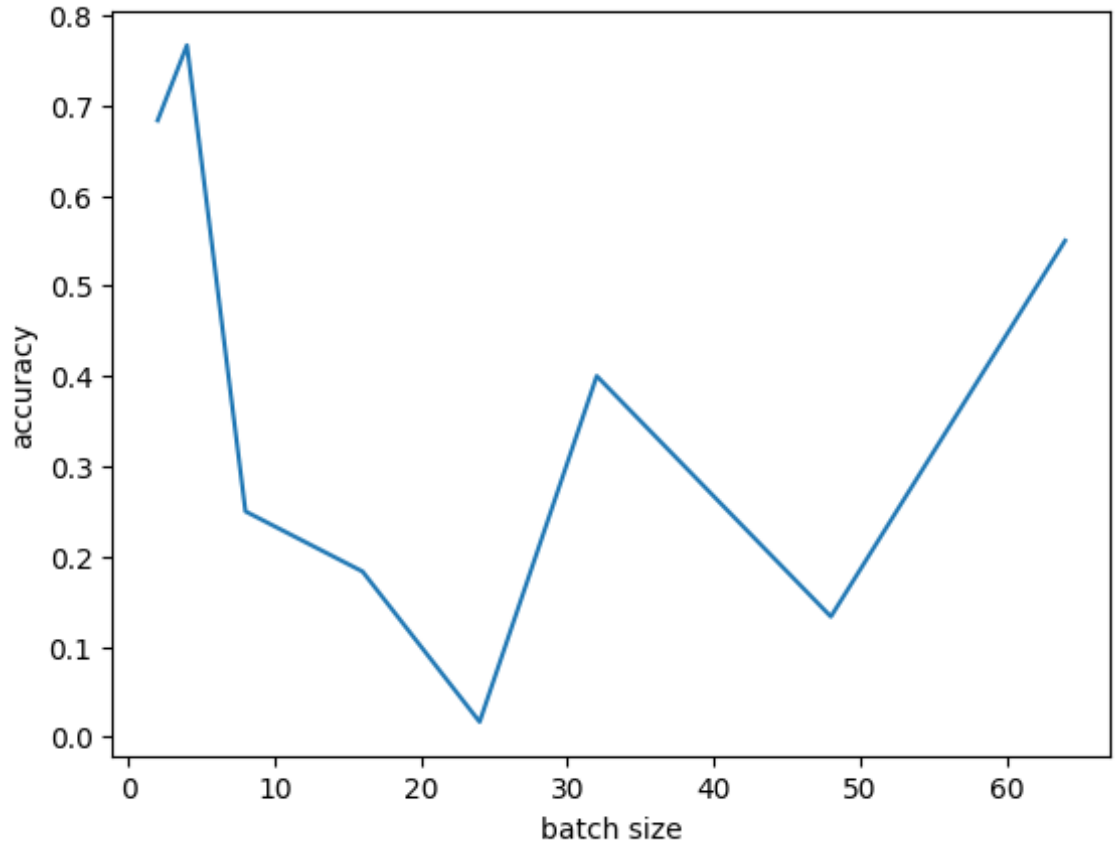
weight = [[-0.875575 1.9825934 -4.016873 -1.6209128]
[0.70730793 -1.0039392 -0.2982792 -2.5409048]
[-0.3355393 -0.79086286 3.4620962 3.9241226]]

bias = [-0.6561739 3.990793 -2.9169736]

train_loss = 0.039542
train_acc = 0.966667
test_loss = 0.040103,
test_acc = 0.950000

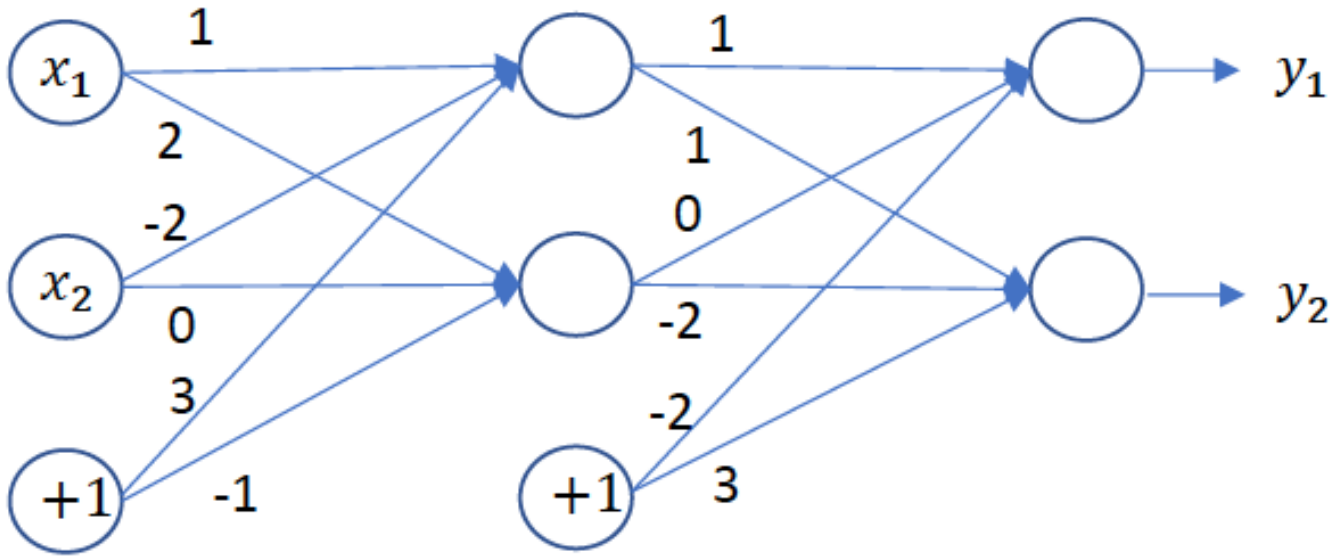
Accuracies and time-to-update weights against batch-size





Deep neural networks

SC4001 – Tutorial 4



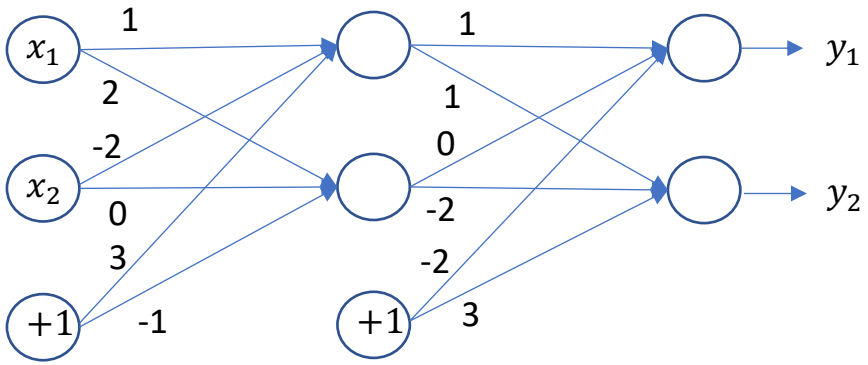
1. The two-layer feedforward perceptron network shown in figure 1 has weights and biases initialized as indicated and receives 2-dimensional inputs (x_1, x_2). The network is to respond with $\boldsymbol{d}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and $\boldsymbol{d}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ for input patterns $\boldsymbol{x}_1 = \begin{pmatrix} 1.0 \\ 3.0 \end{pmatrix}$ and $\boldsymbol{x}_2 = \begin{pmatrix} -2.0 \\ -2.0 \end{pmatrix}$, respectively.

Analyse a single feedforward and feedback step for gradient decent learning of the two patterns by doing the following:

(a) Find the weight matrix $\boldsymbol{W}$ to the hidden-layer and weight matrix $\boldsymbol{V}$ to the output-layer, and the corresponding biases.
(b) Calculate the synaptic input $\boldsymbol{z}$ and output $\boldsymbol{h}$ of the hidden-layer, and the synaptic input $\boldsymbol{u}$ and output $\boldsymbol{y} = (y_1, y_2)$ of the output layer.
(c) Find the mean square error cost J between the outputs and targets.
(d) Calculate the gradients $\boldsymbol{\nabla}_u J$ and $\boldsymbol{\nabla}_z J$ at the output-layer and the hidden-layer, respectively.
(e) Compute the new weights and biases.
(f) Write a program to continue iterations until convergence and find the final weights and biases.

Assume a learning rate of 0.05.

Repeat above (a) – (f) for stochastic gradient decent learning.



Weight matrix to the hidden layer, $\boldsymbol{W} = \begin{pmatrix} 1.0 & 2.0 \\ -2.0 & 0.0 \end{pmatrix}$

Bias vector to the hidden-layer $\boldsymbol{b} = \begin{pmatrix} 3.0 \\ -1.0 \end{pmatrix}$

Weight matrix to the output-layer, $\boldsymbol{V} = \begin{pmatrix} 1.0 & 1.0 \\ 0.0 & -2.0 \end{pmatrix}$

Bias vector to the output-layer $\mathbf{c} = \begin{pmatrix} -2.0 \\ 3.0 \end{pmatrix}$

GD for 2-layer perceptron network:

Given a training dataset $(\boldsymbol{X}, \boldsymbol{D})$
Set learning parameter α
Initialize $\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{V}, \boldsymbol{c}$
Repeat until convergence:

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{B}$$
$$\boldsymbol{H} = g(\boldsymbol{Z})$$
$$\boldsymbol{U} = \boldsymbol{H}\boldsymbol{V} + \boldsymbol{C}$$
$$\boldsymbol{Y} = f(\boldsymbol{U})$$

Forward propagation of activation

$$\nabla_{\boldsymbol{U}} J = -(\boldsymbol{D} - \boldsymbol{Y}) \cdot f'(\boldsymbol{U})$$
$$\nabla_{\boldsymbol{Z}} J = (\nabla_{\boldsymbol{U}} J)\boldsymbol{V}^T \cdot g'(\boldsymbol{Z})$$

Backward propagation of gradients

$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \alpha \boldsymbol{H}^T \nabla_{\boldsymbol{U}} J$$
$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P$$
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{Z}} J$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{Z}} J)^T \mathbf{1}_P$$

Weight updating

$\boldsymbol{x}_1 = \begin{pmatrix} 1.0 \\ 3.0 \end{pmatrix}$ and $\boldsymbol{d}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

$\boldsymbol{x}_2 = \begin{pmatrix} -2.0 \\ -2.0 \end{pmatrix}$ and $\boldsymbol{d}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$\boldsymbol{X} = \begin{pmatrix} 1.0 & 3.0 \\ -2.0 & -2.0 \end{pmatrix}$ and $\boldsymbol{D} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

Forward propagation:

Synaptic input to hidden-layer, $\boldsymbol{Z} = \boldsymbol{XW} + \boldsymbol{B}$

$$= \begin{pmatrix} 1.0 & 3.0 \\ -2.0 & -2.0 \end{pmatrix} \begin{pmatrix} 1.0 & 2.0 \\ -2.0 & 0.0 \end{pmatrix} + \begin{pmatrix} 3.0 & -1.0 \\ 3.0 & -1.0 \end{pmatrix}$$

$$= \begin{pmatrix} -2.0 & 1.0 \\ 5.0 & -5.0 \end{pmatrix}$$

Output of the hidden layer, $\boldsymbol{H} = g(\boldsymbol{Z}) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{Z}}} = \begin{pmatrix} 0.12 & 0.73 \\ 0.99 & 0.01 \end{pmatrix}$

Synaptic input to output-layer, $\boldsymbol{U} = \boldsymbol{HV} + \boldsymbol{C}$

$$= \begin{pmatrix} 0.12 & 0.73 \\ 0.99 & 0.01 \end{pmatrix} \begin{pmatrix} 1.0 & 1.0 \\ 0.0 & -2.0 \end{pmatrix} + \begin{pmatrix} -2.0 & 3.0 \\ -2.0 & 3.0 \end{pmatrix}$$

$$= \begin{pmatrix} -1.88 & 1.66 \\ -0.99 & 3.98 \end{pmatrix}$$

Output of the output layer, $\boldsymbol{Y} = f(\boldsymbol{U}) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{U}}} = \begin{pmatrix} 0.13 & 0.84 \\ 0.27 & 0.98 \end{pmatrix}$

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

$$m.s.e. = \frac{1}{2}\sum_{p=1}^{2}\sum_{k=1}^{2}\left(d_{pk} - y_{pk}\right)^2$$

$$= \frac{1}{2}\Big(\left((0-0.13)^2 + (1-0.84)^2\right) + \left((1-0.27)^2 + (0-0.98)^2\right)\Big)$$

$$= 0.77$$

Computing gradients (backward propagation):

$$f'(\boldsymbol{U}) = \boldsymbol{Y} \cdot (\boldsymbol{1} - \boldsymbol{Y}) = \begin{pmatrix} 0.13 & 0.84 \\ 0.27 & 0.98 \end{pmatrix} \cdot \left(\begin{pmatrix} 1.0 & 1.0 \\ 1.0 & 1.0 \end{pmatrix} - \begin{pmatrix} 0.13 & 0.84 \\ 0.27 & 0.98 \end{pmatrix} \right) = \begin{pmatrix} 0.11 & 0.13 \\ 0.20 & 0.02 \end{pmatrix}$$

$$\nabla_{\boldsymbol{U}} J = -(\boldsymbol{D} - \boldsymbol{Y}) \cdot f'(\boldsymbol{U}) = -\left(\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} - \begin{pmatrix} 0.13 & 0.84 \\ 0.27 & 0.98 \end{pmatrix} \right) \begin{pmatrix} 0.12 & 0.13 \\ 0.20 & 0.02 \end{pmatrix} = \begin{pmatrix} 0.02 & -0.02 \\ -0.14 & 0.02 \end{pmatrix}$$

$$g'(\boldsymbol{Z}) = \boldsymbol{H} \cdot (1 - \boldsymbol{H}) = \begin{pmatrix} 0.12 & 0.73 \\ 0.99 & 0.01 \end{pmatrix} \cdot \left(\begin{pmatrix} 1.0 & 1.0 \\ 1.0 & 1.0 \end{pmatrix} - \begin{pmatrix} 0.12 & 0.73 \\ 0.99 & 0.01 \end{pmatrix} \right) = \begin{pmatrix} 0.10 & 0.2 \\ 0.01 & 0.01 \end{pmatrix}$$

$$\nabla_{\boldsymbol{Z}} J = (\nabla_{\boldsymbol{U}} J)\boldsymbol{V}^T \cdot g'(\boldsymbol{Z}) = \begin{pmatrix} 0.02 & -0.02 \\ -0.14 & 0.02 \end{pmatrix} \begin{pmatrix} 1.0 & 0.0 \\ 1.0 & -2.0 \end{pmatrix} \cdot \begin{pmatrix} 0.10 & 0.2 \\ 0.01 & 0.01 \end{pmatrix} = \begin{pmatrix} -0.001 & 0.01 \\ -0.001 & 0.00 \end{pmatrix}$$

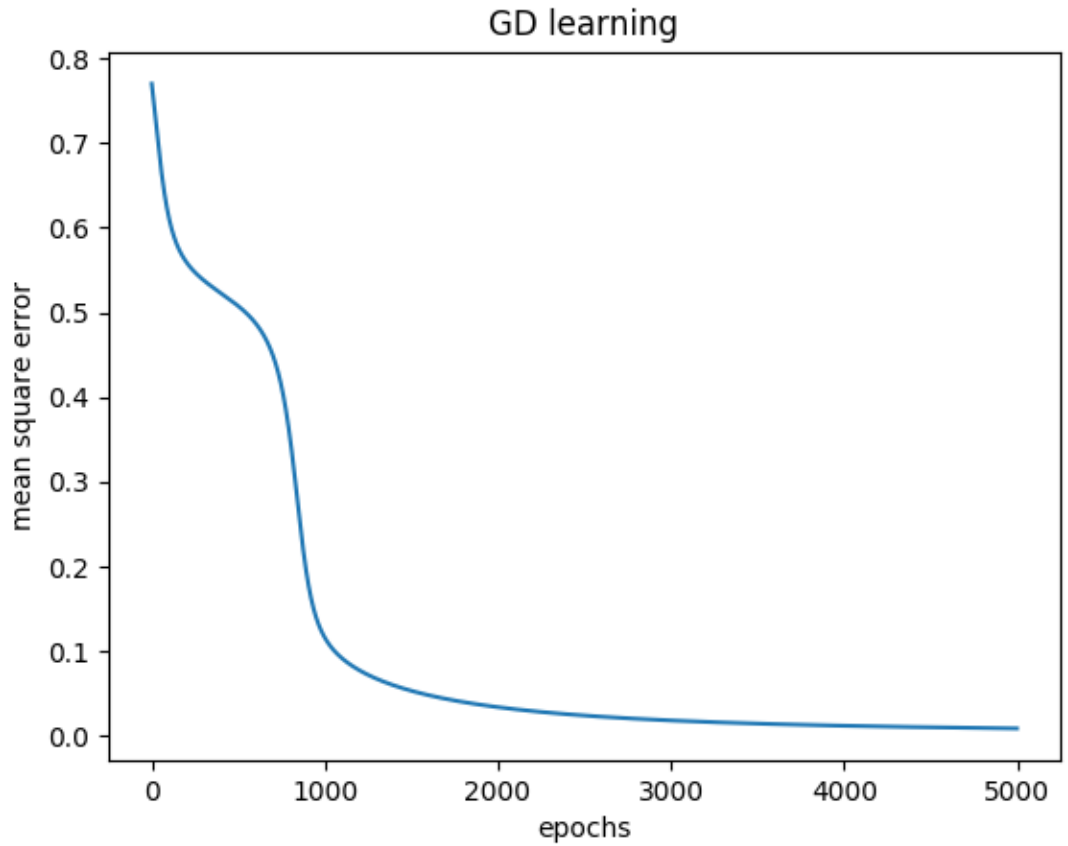
Updating weights:

$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \alpha \boldsymbol{H}^T \nabla_{\boldsymbol{U}} J = \begin{pmatrix} 1.01 & 1.0 \\ 0.0 & -2.0 \end{pmatrix}$$

$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P = \begin{pmatrix} -1.99 \\ 3.00 \end{pmatrix}$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{Z}} J = \begin{pmatrix} 1.0 & 2.0 \\ -2.0 & 0.0 \end{pmatrix}$$

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{Z}} J)^T \mathbf{1}_P = \begin{pmatrix} 3.0 \\ -1.0 \end{pmatrix}$$


GD learning
mean square error
epochs
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1000
2000
3000
4000
5000

At convergence:

$$\boldsymbol{W} = \begin{pmatrix} 0.63 & 0.60 \\ -3.0 & -2.0 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 2.72 \\ -0.74 \end{pmatrix}$$

$$\boldsymbol{V} = \begin{pmatrix} 4.97 & -3.46 \\ 0.25 & -2.37 \end{pmatrix}, \mathbf{c} = \begin{pmatrix} -2.42 \\ 2.56 \end{pmatrix}$$

Predicted values:

$$\boldsymbol{y}_1 = \begin{pmatrix} 0.08 \\ 0.93 \end{pmatrix} \text{ and } \boldsymbol{y}_2 = \begin{pmatrix} 0.94 \\ 0.05 \end{pmatrix}$$

m.s.e. = 0.004

SGD learning for 2-layer perceptron network:

Given a training dataset $\{(\boldsymbol{x}, \boldsymbol{d})\}$
Set learning parameter α
Initialize $\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{V}, \boldsymbol{c}$
Repeat until convergence:
For every pattern $(\boldsymbol{x}, \boldsymbol{d})$:

$$\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{h} = g(\boldsymbol{z})$$
$$\boldsymbol{u} = \boldsymbol{V}^T \boldsymbol{h} + \boldsymbol{c}$$
$$\boldsymbol{y} = f(\boldsymbol{u})$$

Forward propagation of activation

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

Backward propagation of gradients

$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \alpha \boldsymbol{h} (\nabla_{\boldsymbol{u}} J)^T$$
$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha \nabla_{\boldsymbol{u}} J$$
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{x} (\nabla_{\boldsymbol{z}} J)^T$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \nabla_{\boldsymbol{z}} J$$

Weight updating

Epoch 1:

Apply **first** pattern $\boldsymbol{x} = \begin{pmatrix} 1.0 \\ 3.0 \end{pmatrix}$ and $\boldsymbol{d} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$:

Synaptic input to the hidden-layer

$$\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b} = \begin{pmatrix} 1.0 & -2.0 \\ 2.0 & 0.0 \end{pmatrix} \begin{pmatrix} 1.0 \\ 3.0 \end{pmatrix} + \begin{pmatrix} 3.0 \\ -1.0 \end{pmatrix} = \begin{pmatrix} -2.0 \\ 1.0 \end{pmatrix}$$

Output of the hidden-layer $\boldsymbol{h} = g(\boldsymbol{z}) = \frac{1}{1+e^{-\boldsymbol{z}}} = \begin{pmatrix} 0.12 \\ 0.73 \end{pmatrix}$

Synaptic input to output-layer

$$\boldsymbol{u} = \boldsymbol{V}^T \boldsymbol{h} + \boldsymbol{c} = \begin{pmatrix} -1.88 \\ 1.66 \end{pmatrix}$$

Output of the output-layer $\boldsymbol{y} = f(\boldsymbol{u}) = \frac{1}{1+e^{-\boldsymbol{u}}} = \begin{pmatrix} 0.13 \\ 0.84 \end{pmatrix}$

$s.e. = (d_1 - y_1)^2 + (d_2 - y_2)^2 = 0.043$

Computing gradients:

$$f'(\boldsymbol{u}) = \boldsymbol{y} \cdot (1 - \boldsymbol{y}) = \begin{pmatrix} 0.13 \\ 0.84 \end{pmatrix} \cdot \left(\begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix} - \begin{pmatrix} 0.13 \\ 0.84 \end{pmatrix} \right) = \begin{pmatrix} 0.11 \\ 0.13 \end{pmatrix}$$

$$\nabla_{\boldsymbol{u}} J = -(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u}) = -\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 0.13 \\ 0.84 \end{pmatrix} \right) \cdot \begin{pmatrix} 0.12 \\ 0.14 \end{pmatrix} = \begin{pmatrix} 0.02 \\ -0.02 \end{pmatrix}$$

$$g'(\boldsymbol{z}) = \boldsymbol{h} \cdot (1 - \boldsymbol{h}) = \begin{pmatrix} 0.12 \\ 0.73 \end{pmatrix} \cdot \left(\begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix} - \begin{pmatrix} 0.12 \\ 0.73 \end{pmatrix} \right) = \begin{pmatrix} 0.10 \\ 0.20 \end{pmatrix}$$

$$\nabla_{\boldsymbol{z}} J = \boldsymbol{V} \nabla_{\boldsymbol{u}} J \cdot g'(\boldsymbol{z}) = \begin{pmatrix} 1.0 & 1.0 \\ 0.0 & -2.0 \end{pmatrix} \begin{pmatrix} 0.02 \\ -0.02 \end{pmatrix} \cdot \begin{pmatrix} 0.11 \\ 0.20 \end{pmatrix} = \begin{pmatrix} -0.001 \\ 0.008 \end{pmatrix}$$

Updating weights:

$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \alpha \boldsymbol{h} (\nabla_{\boldsymbol{u}} J)^T = \begin{pmatrix} 1.0 & 1.0 \\ 0.0 & -2.0 \end{pmatrix} - 0.2 \begin{pmatrix} 0.12 \\ 0.73 \end{pmatrix} (-0.02 \quad 0.022) = \begin{pmatrix} 1.0 & 1.0001 \\ 0.00 & -2.0 \end{pmatrix}$$

$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha \nabla_{\boldsymbol{u}} J = \begin{pmatrix} -2.0 \\ 3.0 \end{pmatrix} + 0.2 \begin{pmatrix} 0.02 \\ -0.02 \end{pmatrix} = \begin{pmatrix} -2.00 \\ 3.001 \end{pmatrix}$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{x} (\nabla_{\boldsymbol{z}} J)^T = \begin{pmatrix} 1.0 & 2.0 \\ -2.00 & -0.001 \end{pmatrix}$$

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \nabla_{\boldsymbol{z}} J = \begin{pmatrix} 3.00 \\ -1.00 \end{pmatrix}$$

Apply **second** pattern $\boldsymbol{x} = \begin{pmatrix} -2.0 \\ -2.0 \end{pmatrix}$ and $\boldsymbol{d} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$:

Synaptic input to the hidden-layer

$$\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b} = \begin{pmatrix} 5.0 \\ -5.0 \end{pmatrix}$$

Output of the hidden-layer $\boldsymbol{h} = g(\boldsymbol{z}) = \frac{1}{1+e^{-\boldsymbol{z}}} = \begin{pmatrix} 1.0 \\ 0.007 \end{pmatrix}$

Synaptic input to output-layer

$$\boldsymbol{u} = \boldsymbol{V}^T \boldsymbol{h} + \boldsymbol{c} = \begin{pmatrix} -0.99 \\ 3.98 \end{pmatrix}$$

Output of the output-layer $\boldsymbol{y} = f(\boldsymbol{u}) = \frac{1}{1+e^{-\boldsymbol{u}}} = \begin{pmatrix} 0.27 \\ 0.98 \end{pmatrix}$

$s.e. = (d_1 - y_1)^2 + (d_2 - y_2)^2 = 1.5$

Computing gradients:

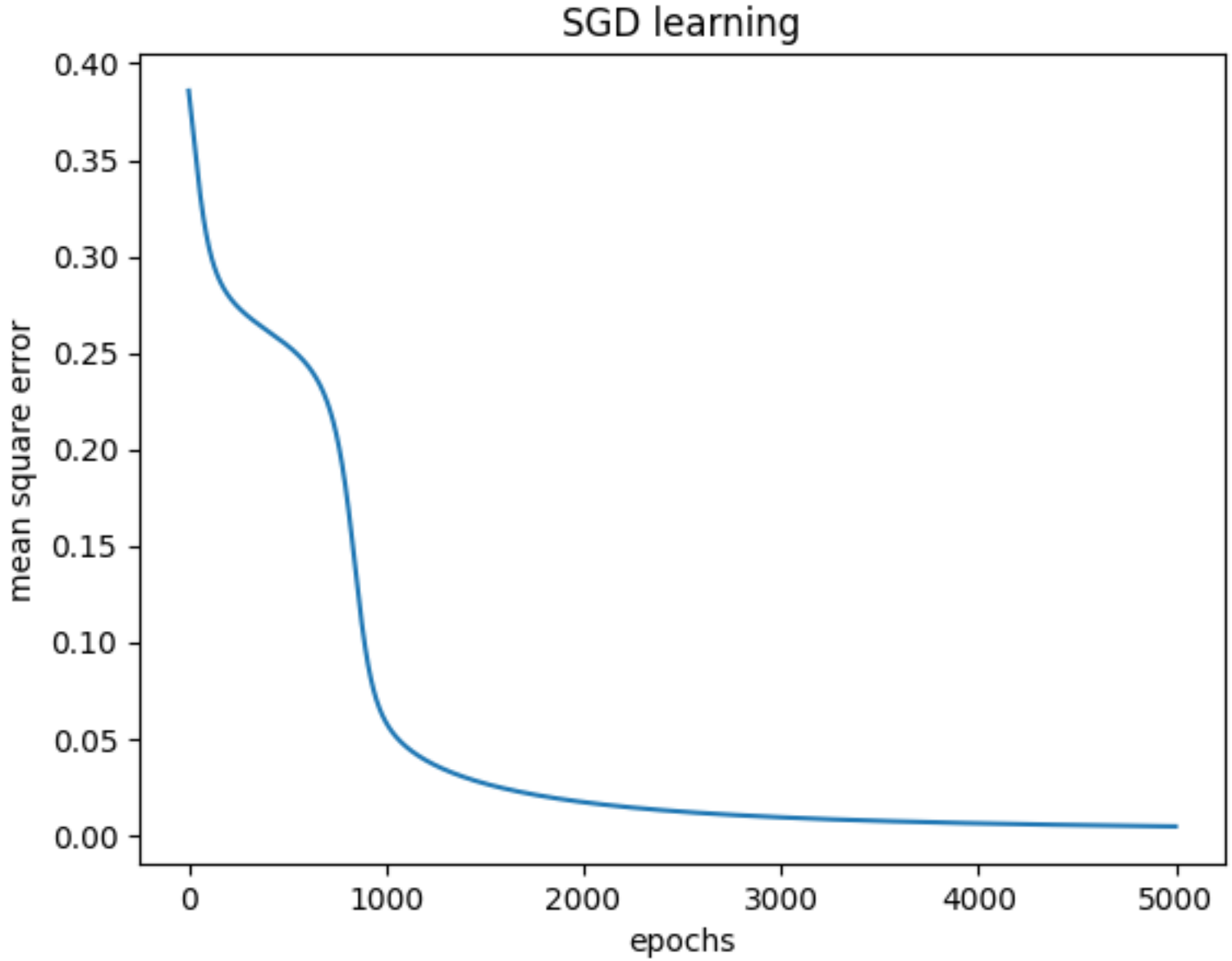
$$f'(\boldsymbol{u}) = \boldsymbol{y} \cdot (1 - \boldsymbol{y}) = \begin{pmatrix} 0.195 \\ 0.018 \end{pmatrix}$$
$$\nabla_{\boldsymbol{u}} J = -(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u}) = \begin{pmatrix} -0.14 \\ 0.018 \end{pmatrix}$$

$$g'(\boldsymbol{z}) = \boldsymbol{h} \cdot (1 - \boldsymbol{h}) = \begin{pmatrix} 0.007 \\ 0.007 \end{pmatrix}$$
$$\nabla_{\boldsymbol{z}} J = \boldsymbol{V} \nabla_{\boldsymbol{u}} J \cdot g'(\boldsymbol{z}) = \begin{pmatrix} -0.0008 \\ -0.0002 \end{pmatrix}$$

Updating weights:

$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \alpha \boldsymbol{h} (\nabla_{\boldsymbol{u}} J)^T = \begin{pmatrix} 1.007 & 0.99 \\ 0.0 & -2.0 \end{pmatrix}$$
$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha \nabla_{\boldsymbol{u}} J = \begin{pmatrix} -1.99 \\ 3.0 \end{pmatrix}$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{x} (\nabla_{\boldsymbol{z}} J)^T = \begin{pmatrix} 0.999 & 1.99 \\ -1.99 & 0.00 \end{pmatrix}$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \nabla_{\boldsymbol{z}} J = \begin{pmatrix} 3.00 \\ -1.00 \end{pmatrix}$$


SGD learning
mean square error
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1000
2000
3000
4000
5000
epochs

At convergence:

$$\boldsymbol{W} = \begin{pmatrix} 0.63 & 0.60 \\ -3.0 & -2.0 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 2.72 \\ -0.74 \end{pmatrix}$$

$$\boldsymbol{V} = \begin{pmatrix} 4.97 & -3.46 \\ 0.25 & -2.37 \end{pmatrix}, \mathbf{c} = \begin{pmatrix} -2.42 \\ 2.56 \end{pmatrix}$$

Predicted values:

$$\boldsymbol{y}_1 = \begin{pmatrix} 0.08 \\ 0.93 \end{pmatrix} \text{ and } \boldsymbol{y}_2 = \begin{pmatrix} 0.94 \\ 0.05 \end{pmatrix}$$

m.s.e. = 0.004

2. A feedforward neural network with one hidden layer to perform the following classification:

class	inputs
A	$(1.0, 1.0), (0.0, 1.0)$
B	$(3.0, 4.0), (2.0, 2.0)$
C	$(2.0, -2.0), (-2.0, -3.0)$

The network has a hidden layer consisting of three perceptrons and a softmax output layer.

Show one iteration of gradient descent learning and plot learning curves until convergence at a learning rate $\alpha = 0.1$.

Initialize the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ to the hidden layer, and the weights $\boldsymbol{V}$ and biases $\boldsymbol{c}$ to the output layer as follows:

$$\boldsymbol{W} = \begin{pmatrix} -0.10 & 0.97 & 0.18 \\ -0.70 & 0.38 & 0.93 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

$$\boldsymbol{V} = \begin{pmatrix} 1.01 & 0.09 & -0.39 \\ 0.79 & -0.45 & -0.22 \\ 0.28 & 0.96 & -0.07 \end{pmatrix}, \boldsymbol{c} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

Determine the weights and biases at convergence.

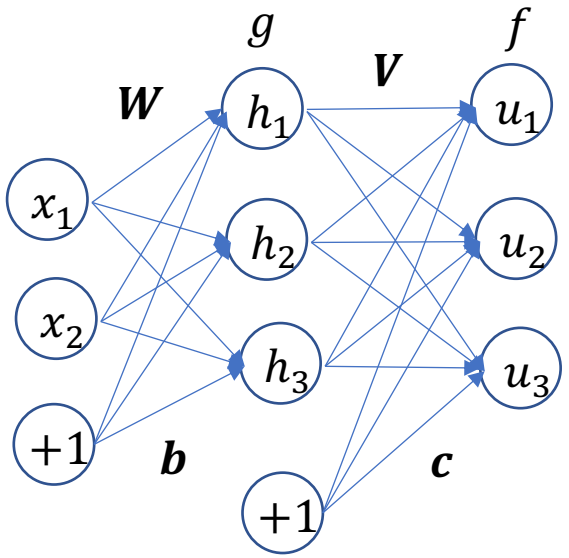
Find the class labels predicted by the trained network for patterns:

$\boldsymbol{x}_1 = \begin{pmatrix} 2.5 \\ 1.5 \end{pmatrix}$ and $\boldsymbol{x}_2 = \begin{pmatrix} -1.5 \\ 0.5 \end{pmatrix}$

Training examples (patterns):

class	*inputs*	*Target Label*
A	$(1.0, 1.0), (0.0, 1.0)$	1
B	$(3.0, 4.0), (2.0, 2.0)$	2
C	$(2.0, -2.0), (-2.0, -3.0)$	3

Feedforward network :
Perceptron hidden layer with 3 neurons
Softmax output layer with 3 neurons



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

$$f(\boldsymbol{U}) = \frac{e^{\boldsymbol{U}}}{\sum_{k=1}^{K} e^{\boldsymbol{U_k}}}$$

Training inputs and targets

$$\boldsymbol{X} = \begin{pmatrix} 1.0 & 1.0 \\ 0.0 & 1.0 \\ 3.0 & 4.0 \\ 2.0 & 2.0 \\ 2.0 & -2.0 \\ -2.0 & -3.0 \end{pmatrix} \text{ and } \boldsymbol{D} = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ 3 \end{pmatrix}$$

Targets as a one hot matrix:

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

Initial weights and biases:

To the hidden layer,

$$\boldsymbol{W} = \begin{pmatrix} -0.10 & 0.97 & 0.18 \\ -0.70 & 0.38 & 0.93 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

To the output-layer

$$\boldsymbol{V} = \begin{pmatrix} 1.01 & 0.09 & -0.39 \\ 0.79 & -0.45 & -0.22 \\ 0.28 & 0.96 & -0.07 \end{pmatrix}, \boldsymbol{c} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

Learning factor $\boldsymbol{\alpha = 0.1}$

Activation functions:

Hidden layer is a continuous perceptron layer: $g(\boldsymbol{Z}) = \frac{1}{1+e^{-\boldsymbol{Z}}}$

Output layer is a softmax layer: $f(\boldsymbol{U}) = \frac{e^{\boldsymbol{U}}}{\sum_{k=1}^{K} e^{\boldsymbol{U_k}}}$

GD for the feedforward network

Given a training dataset $(\boldsymbol{X}, \boldsymbol{D})$
Set learning parameter α
Initialize $\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{V}, \boldsymbol{c}$
Repeat until convergence:

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{B}$$
$$\boldsymbol{H} = g(\boldsymbol{Z})$$
$$\boldsymbol{U} = \boldsymbol{H}\boldsymbol{V} + \boldsymbol{C}$$
$$\boldsymbol{Y} = \arg\max_k f(\boldsymbol{U})$$

Forward propagation of activation

$$\nabla_{\boldsymbol{U}} J = -\big(\boldsymbol{K} - f(\boldsymbol{U})\big)$$
$$\nabla_{\boldsymbol{Z}} J = (\nabla_{\boldsymbol{U}} J)\boldsymbol{V}^T \cdot g'(\boldsymbol{Z})$$

Backward propagation of gradients

$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \alpha \boldsymbol{H}^T \nabla_{\boldsymbol{U}} J$$
$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P$$
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{Z}} J$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{Z}} J)^T \mathbf{1}_P$$

Weight updating

Synaptic input to hidden-layer,

$$\boldsymbol{Z} = \boldsymbol{XW} + \boldsymbol{B} = \begin{pmatrix} 1.0 & 1.0 \\ 0.0 & 1.0 \\ 3.0 & 4.0 \\ 2.0 & 2.0 \\ 2.0 & -2.0 \\ -2.0 & -3.0 \end{pmatrix} \begin{pmatrix} -0.10 & 0.97 & 0.18 \\ -0.70 & 0.38 & 0.93 \end{pmatrix} + \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix} = \begin{pmatrix} -0.80 & 1.35 & 1.10 \\ -0.70 & 0.38 & 0.93 \\ -3.08 & 4.44 & 4.23 \\ -1.59 & 2.70 & 2.21 \\ 1.20 & 1.18 & -1.50 \\ 2.29 & -3.08 & -3.13 \end{pmatrix}$$

Output of the hidden layer, $\boldsymbol{H} = g(\boldsymbol{Z}) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{Z}}} = \begin{pmatrix} 0.31 & 0.79 & 0.75 \\ 0.33 & 0.59 & 0.72 \\ 0.04 & 0.99 & 0.99 \\ 0.17 & 0.94 & 0.90 \\ 0.77 & 0.77 & 0.18 \\ 0.91 & 0.04 & 0.04 \end{pmatrix}$

Synaptic input to output-layer,

$$\boldsymbol{U} = \boldsymbol{HV} + \boldsymbol{C} = \begin{pmatrix} 0.31 & 0.79 & 0.75 \\ 0.33 & 0.59 & 0.72 \\ 0.04 & 0.99 & 0.99 \\ 0.17 & 0.94 & 0.90 \\ 0.77 & 0.77 & 0.18 \\ 0.91 & 0.04 & 0.04 \end{pmatrix} \begin{pmatrix} 1.01 & 0.09 & -0.39 \\ 0.79 & -0.45 & -0.22 \\ 0.28 & 0.96 & -0.07 \end{pmatrix} + \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix} = \begin{pmatrix} 1.15 & 0.40 & -0.34 \\ 1.01 & 0.46 & -0.31 \\ 1.10 & 0.51 & -0.30 \\ 1.16 & 0.47 & -0.33 \\ 1.43 & -0.09 & -0.48 \\ 0.96 & 0.11 & -0.36 \end{pmatrix}$$

Output layer activation $f(\boldsymbol{U}) = \frac{e^{\boldsymbol{U}}}{\sum_{k=1}^{K} e^{\boldsymbol{U_k}}} = \begin{pmatrix} 0.59 & 0.28 & 0.13 \\ 0.54 & 0.31 & 0.15 \\ 0.56 & 0.31 & 0.14 \\ 0.58 & 0.29 & 0.13 \\ 0.73 & 0.16 & 0.11 \\ 0.59 & 0.25 & 0.16 \end{pmatrix}$

Output $\boldsymbol{Y} = \arg\max_k f(\boldsymbol{U}) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

$$\boldsymbol{D} = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ 3 \end{pmatrix}, \qquad \boldsymbol{Y} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \qquad f(\boldsymbol{U}) = \begin{pmatrix} 0.59 & 0.28 & 0.13 \\ 0.54 & 0.31 & 0.15 \\ 0.56 & 0.31 & 0.14 \\ 0.58 & 0.29 & 0.13 \\ 0.73 & 0.16 & 0.11 \\ 0.59 & 0.25 & 0.16 \end{pmatrix}$$

Classification error = $\sum 1(\boldsymbol{D} \neq \boldsymbol{Y}) = 4$

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

$= -\big(log(0.59) + log(0.54) + log(0.31) + log(0.29) + log(0.11) + log(0.16)\big)$

$= 7.63$

$$\nabla_{\boldsymbol{U}} J = -\big(\boldsymbol{K} - f(\boldsymbol{U})\big) = -\left(\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0.59 & 0.28 & 0.13 \\ 0.54 & 0.31 & 0.15 \\ 0.56 & 0.31 & 0.14 \\ 0.58 & 0.29 & 0.13 \\ 0.73 & 0.16 & 0.11 \\ 0.59 & 0.25 & 0.16 \end{pmatrix}\right) = \begin{pmatrix} -0.41 & 0.28 & 0.13 \\ -0.46 & 0.31 & 0.15 \\ 0.56 & -0.69 & 0.14 \\ 0.58 & -0.71 & 0.13 \\ 0.73 & 0.16 & -0.89 \\ 0.59 & 0.25 & -0.84 \end{pmatrix}$$

$$g'(\boldsymbol{Z}) = \boldsymbol{H} \cdot (\boldsymbol{1} - \boldsymbol{H}) = \begin{pmatrix} 0.21 & 0.16 & 0.19 \\ 0.22 & 0.24 & 0.20 \\ 0.04 & 0.01 & 0.01 \\ 0.14 & 0.06 & 0.09 \\ 0.18 & 0.18 & 0.15 \\ 0.08 & 0.04 & 0.04 \end{pmatrix}$$

$$\nabla_{\boldsymbol{Z}} J = (\nabla_{\boldsymbol{U}} J)\boldsymbol{V}^T \cdot g'(\boldsymbol{Z}) = \begin{pmatrix} -0.41 & 0.28 & 0.13 \\ -0.46 & 0.31 & 0.15 \\ 0.56 & -0.69 & 0.14 \\ 0.58 & -0.71 & 0.13 \\ 0.73 & 0.16 & -0.89 \\ 0.59 & 0.25 & -0.84 \end{pmatrix} \begin{pmatrix} 1.01 & 0.09 & -0.39 \\ 0.79 & -0.45 & -0.22 \\ 0.28 & 0.96 & -0.07 \end{pmatrix}^T \cdot \begin{pmatrix} 0.21 & 0.16 & 0.19 \\ 0.22 & 0.24 & 0.20 \\ 0.04 & 0.01 & 0.01 \\ 0.14 & 0.06 & 0.09 \\ 0.18 & 0.18 & 0.15 \\ 0.08 & 0.04 & 0.04 \end{pmatrix} = \begin{pmatrix} -0.09 & -0.08 & 0.03 \\ -0.11 & -0.13 & 0.03 \\ 0.02 & 0.01 & -0.01 \\ 0.07 & 0.04 & -0.05 \\ 0.20 & 0.13 & 0.06 \\ 0.08 & 0.02 & 0.02 \end{pmatrix}$$

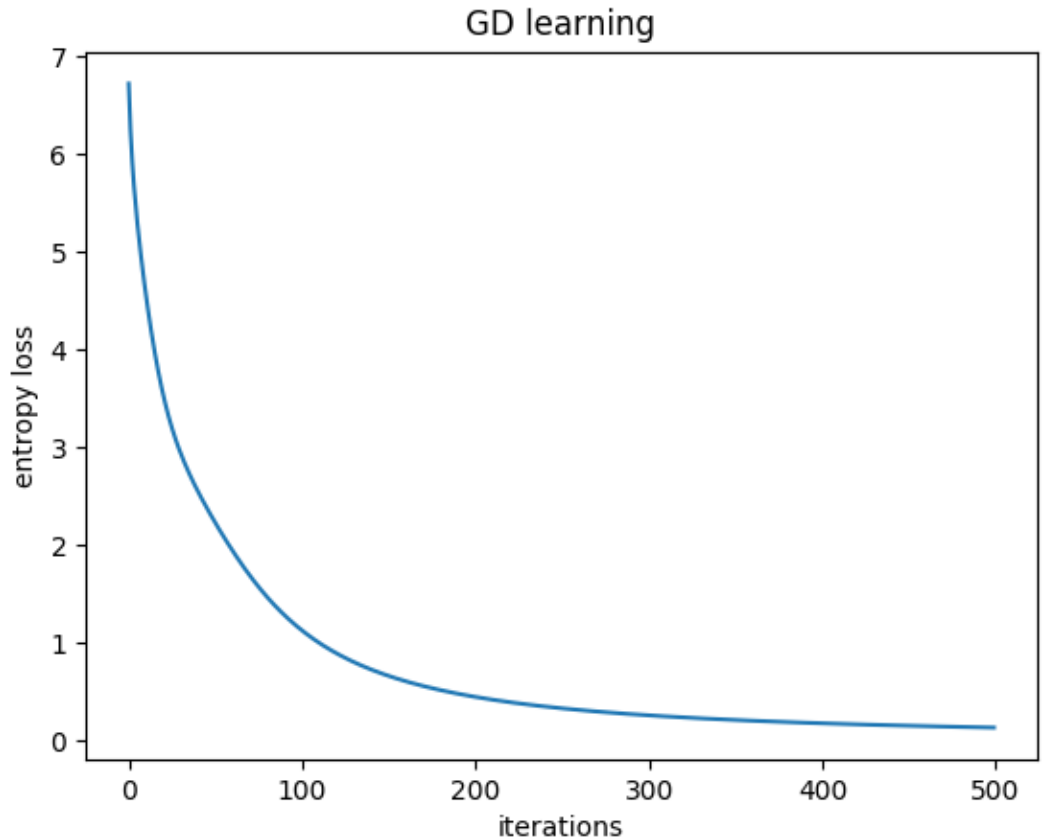
Learning rate $\alpha = 0.1$

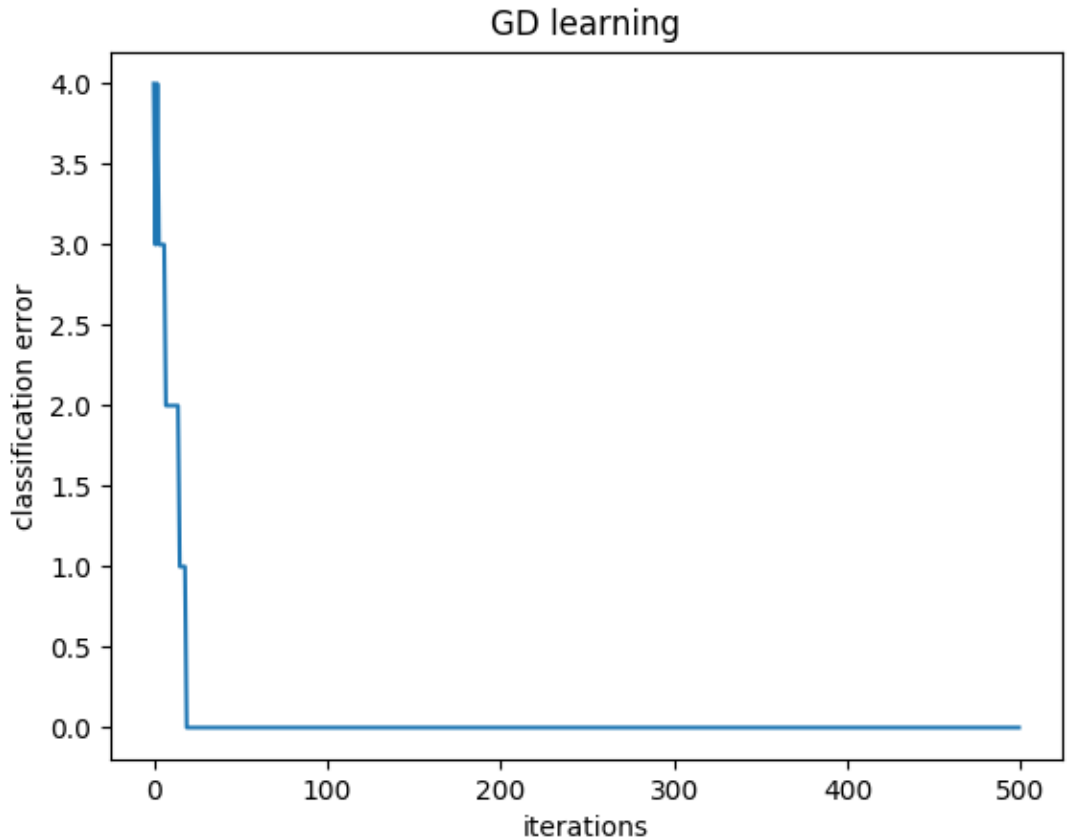
$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \alpha \boldsymbol{H}^T \nabla_{\boldsymbol{U}} J = \begin{pmatrix} 0.92 & 0.05 & -0.26 \\ 0.68 & -0.36 & -0.19 \\ 0.22 & 1.05 & -0.10 \end{pmatrix}$$

$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha (\nabla_{\boldsymbol{U}} J)^T \mathbf{1}_P = \begin{pmatrix} -0.16 \\ 0.04 \\ 0.12 \end{pmatrix}$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{Z}} J = \begin{pmatrix} -0.13 & 0.95 & 0.18 \\ -0.63 & 0.42 & 0.95 \end{pmatrix}$$

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha (\nabla_{\boldsymbol{Z}} J)^T \mathbf{1}_P = \begin{pmatrix} -0.02 \\ 0.00 \\ -0.01 \end{pmatrix}$$


GD learning
entropy loss
iterations
0
1
2
3
4
5
6
7
0
100
200
300
400
500


GD learning
classification error
iterations
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500

At convergence:

$$\boldsymbol{V} = \begin{pmatrix} 2.93 & -5.33 & 3.12 \\ 2.80 & 1.20 & -3.87 \\ 0.09 & 4.55 & -3.47 \end{pmatrix}, \qquad \boldsymbol{c} = \begin{pmatrix} -1.94 \\ -0.06 \\ 2.01 \end{pmatrix}$$

$$\boldsymbol{W} = \begin{pmatrix} -1.81 & 0.32 & 0.08 \\ -1.40 & 2.92 & 1.91 \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} 4.36 \\ 0.73 \\ -1.71 \end{pmatrix}$$

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

Entropy = 0.138

Error = 0

Testing patterns:

$$\boldsymbol{x}_1 = \begin{pmatrix} 2.5 \\ 1.5 \end{pmatrix} \text{ and } \boldsymbol{x}_2 = \begin{pmatrix} -1.5 \\ 0.5 \end{pmatrix}$$

In batch mode

$$\boldsymbol{X} = \begin{pmatrix} 2.5 & 1.5 \\ -1.5 & 0.5 \end{pmatrix}$$

Forward propagation of activations

$$\boldsymbol{Z} = \boldsymbol{XW} + \boldsymbol{B} = \begin{pmatrix} -2.26 & 5.91 & 1.36 \\ 6.35 & 1.72 & -0.91 \end{pmatrix}$$

$$\boldsymbol{H} = f(\boldsymbol{Z}) = \begin{pmatrix} 0.09 & 1.0 & 0.8 \\ 1.0 & 0.85 & 0.29 \end{pmatrix}$$

$$\boldsymbol{U} = \boldsymbol{HV} + \boldsymbol{C} = \begin{pmatrix} 1.2 & 4.25 & -4.33 \\ 3.38 & -3.006 & 0.82 \end{pmatrix}$$

$$g(\boldsymbol{U}) = \frac{e^{\boldsymbol{U}}}{\sum_{k=1}^{K} e^{\boldsymbol{U_k}}} = \begin{pmatrix} 0.05 & 0.95 & 0.0 \\ 0.93 & 0.0 & 0.07 \end{pmatrix}$$

$$\boldsymbol{Y} = \arg\max_k g(\boldsymbol{U}) = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

The class labels: $\boldsymbol{x}_1 \rightarrow class\ B, \boldsymbol{x}_2 \rightarrow class\ A$

3. Design a feedforward neural network consisting of two-hidden layers to approximate the following function:

$$\phi(x, y) = 0.8x^2 - y^3 + 2.5xy$$

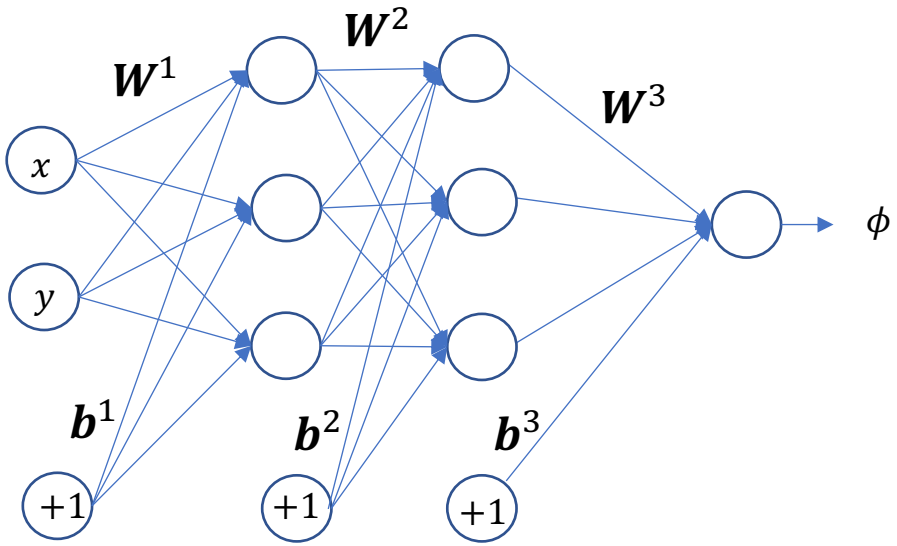
for $-1.0 \leq x, y \leq 1.0$.

Use three ReLU neurons at each hidden layer and a linear neuron at the output layer.

(a) Divide the input space equally into square regions of size 0.25×0.25 and use grid points as data to learn the function ϕ.
(b) Train the network using gradient decent learning at learning rate $\alpha = 0.01$ and plot the learning curve (mean square error vs. iterations) and the predicted data points.
(c) Compare the learning curves when learning the function at learning rates $\alpha = 0.005, 0.01, 0.05, \text{and } 0.1$.

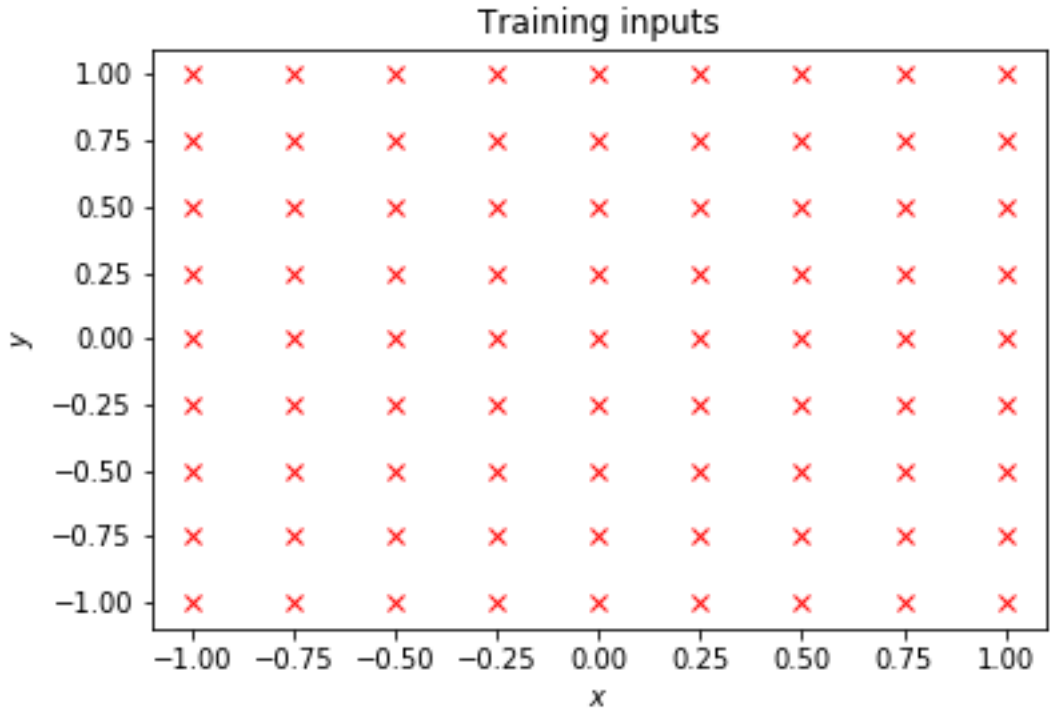
$$\phi(x, y) = 0.8x^2 - x^3 + 2.5xy \quad \text{for } -1.0 \leq x, y \leq 1.0$$

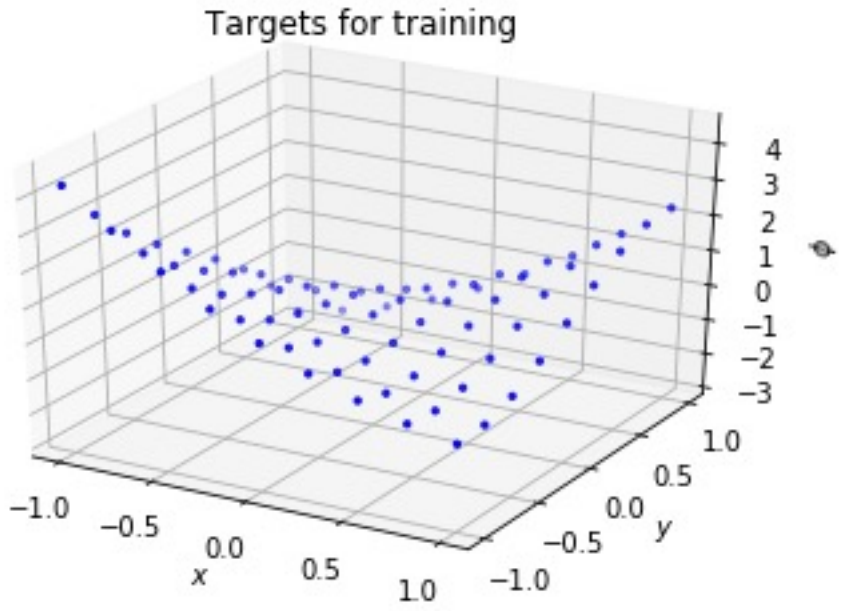
Feedforward neural network with two hidden layers:



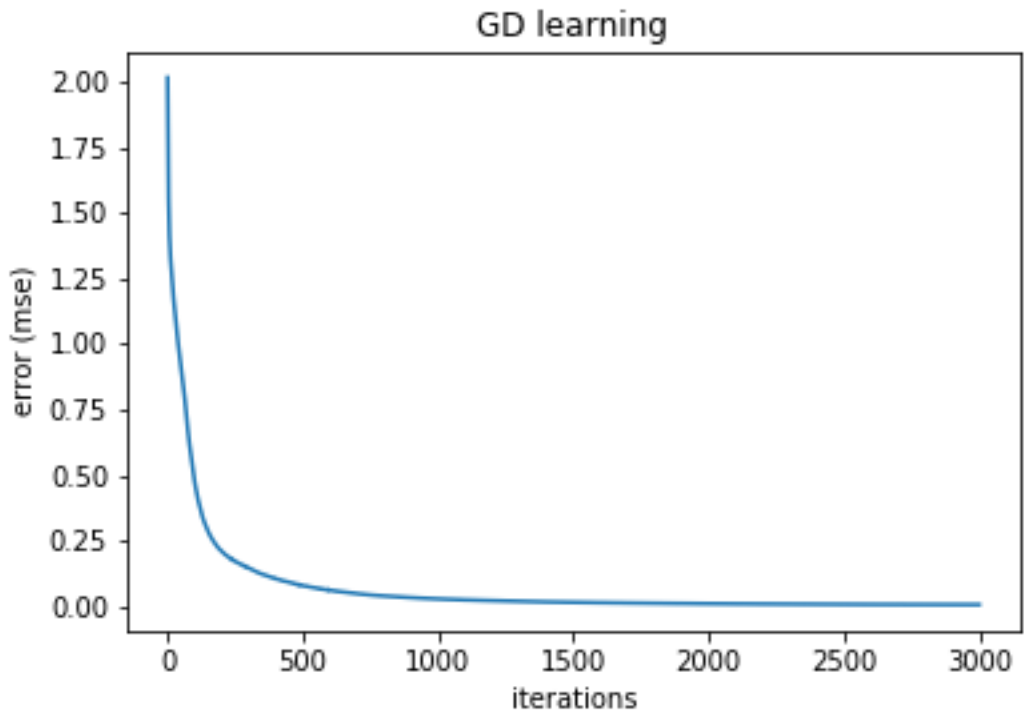
$$\phi(x, y) = 0.8{x_1}^2 - y^3 + 2.5xy \quad \text{for } -1.0 \leq x, y \leq 1.0$$

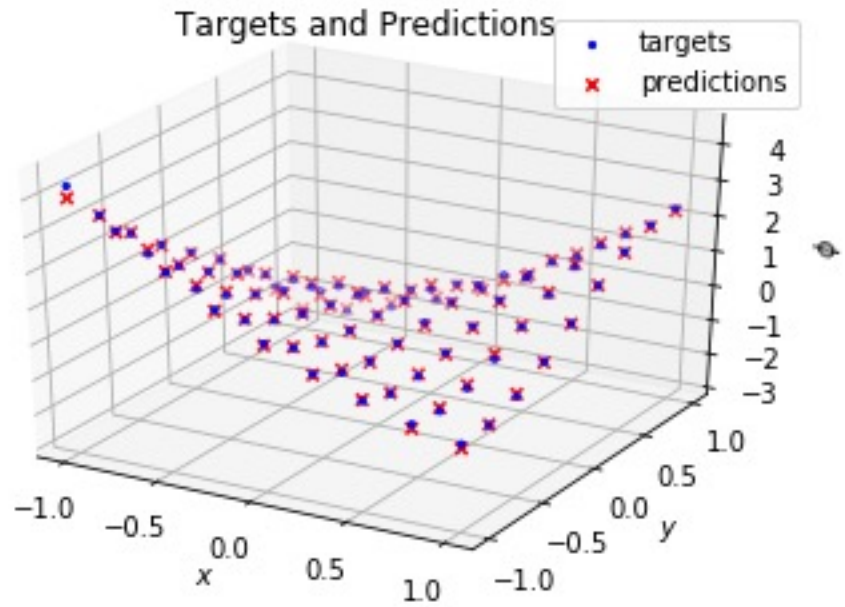
Data points in a grid of size 0.25x0.25:

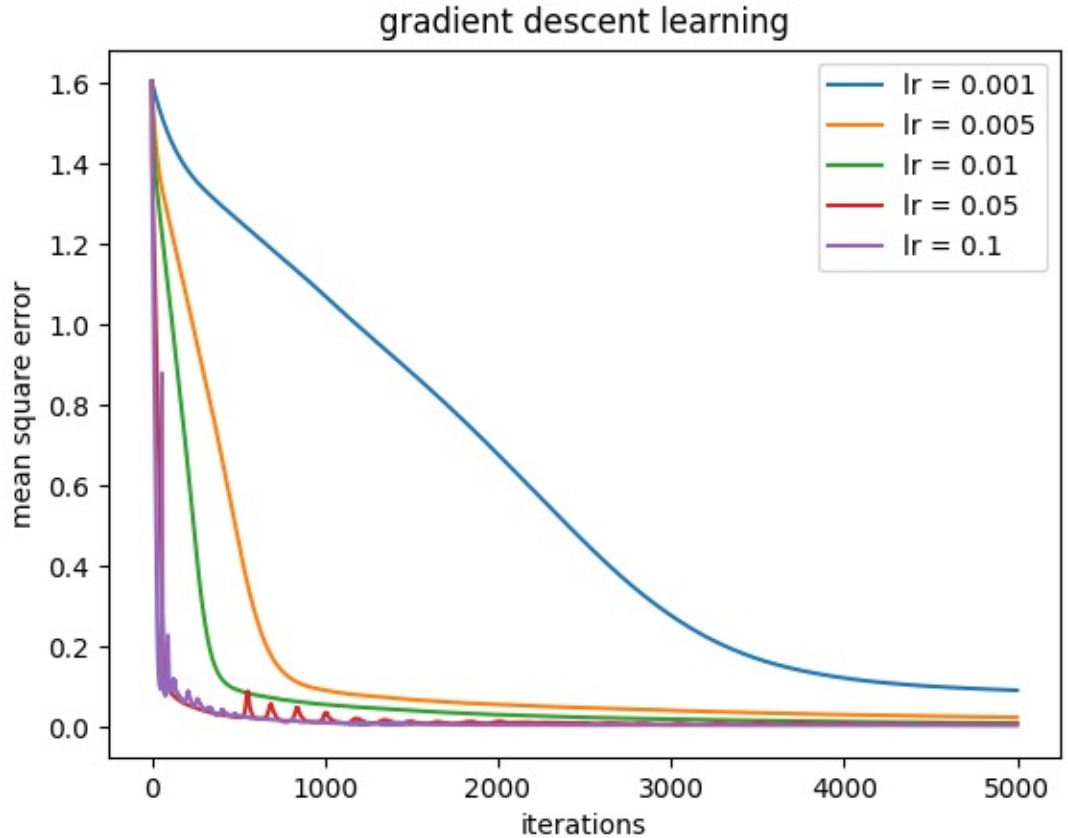




```
class FFN(nn.Module):
    def __init__(self):
        super().__init__()
        self.relu_stack = nn.Sequential(
            nn.Linear(2, 10),
            nn.ReLU(),
            nn.Linear(10, 5),
            nn.ReLU(),
            nn.Linear(5, 1),
        )

    def forward(self, x):
        logits = self.relu_stack(x)
        return logits
```


GD learning
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
error (mse)
0
500
1000
1500
2000
2500
3000
iterations


Targets and Predictions
targets
predictions
4
3
2
1
0
−1
−2
−3
z
1.0
0.5
0.0
−0.5
−1.0
y
−1.0
−0.5
0.0
0.5
1.0
x


gradient descent learning
lr = 0.001
lr = 0.005
lr = 0.01
lr = 0.05
lr = 0.1
mean square error
iterations
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
1000
2000
3000
4000
5000

SC 4001 CE/CZ 4042 Neural Network and Deep Learning

Last update: 24 Feb 2024

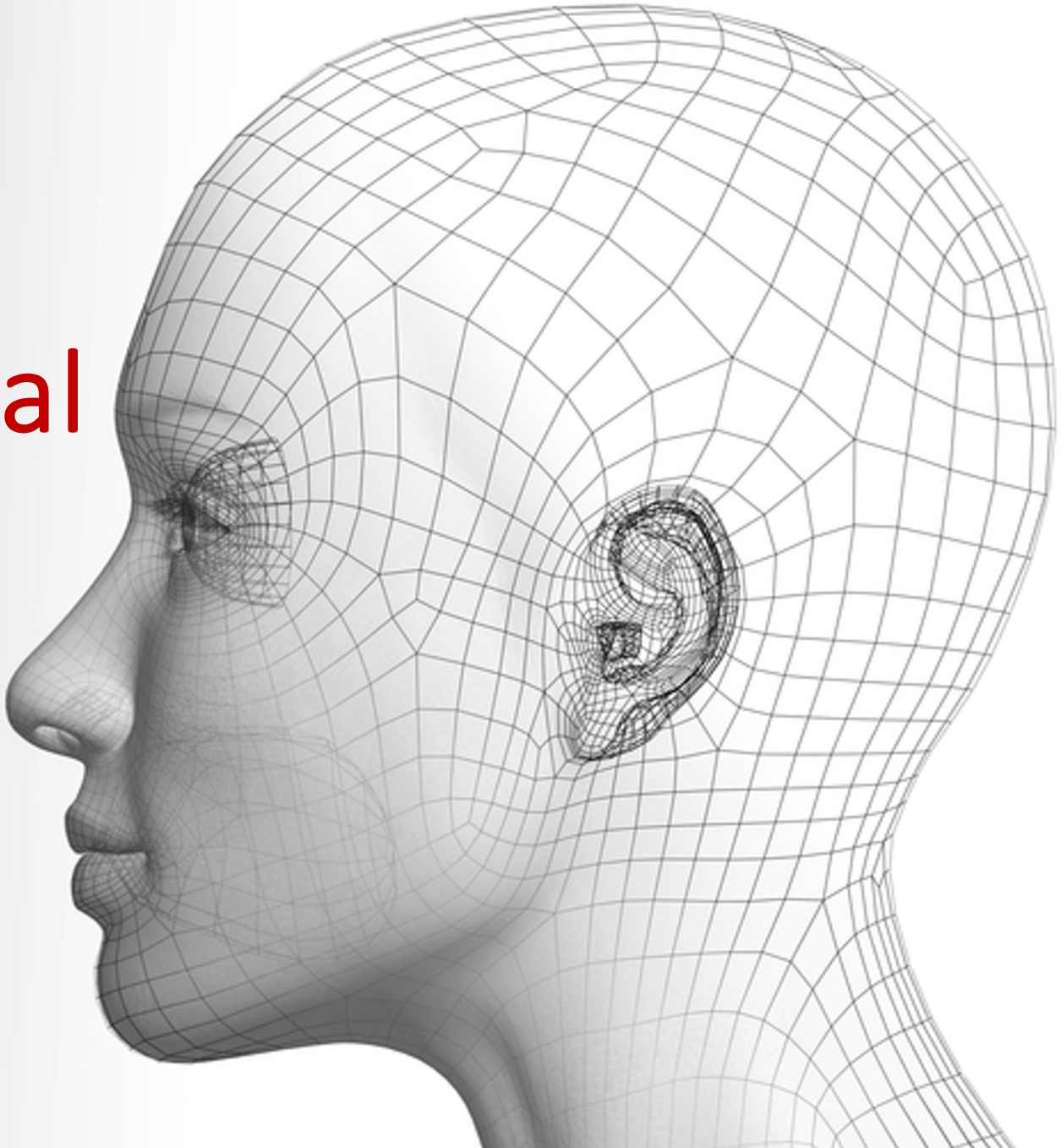
Tutorial 6 Convolutional Neural Networks I

Xingang Pan

潘新钢

https://xingangpan.github.io/

https://twitter.com/XingangP



Question 1

1. The first hidden layer of a convolution neural network (CNN) has a convolution layer consisting of two feature maps with filters $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$ and biases = 0.1, and neurons having sigmoid activation functions, and a pooling layer with a pooling windows of size 2x2:

$$\boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \text{ and } \boldsymbol{w}_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

The input layer is of 6x6 size and receives an input image $\boldsymbol{I}$:

Filters=weights

$$\boldsymbol{I} = \begin{pmatrix} 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

   a. Find the outputs at the first convolution layer if
      i. padding = 0 and strides = (1,1)
      ii. padding = 1 and strides = (2,2)
   b. Find the outputs at the first pooling layer for Part (a), assuming strides of (2, 2) and pooling is
      i. max pooling
      ii. mean pooling

t6q1.ipynb

Question 1

$$\boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \text{ and } b = 0.1.$$

$$\boldsymbol{I} = \begin{pmatrix} 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

Synaptic inputs to the feature map with filter $\boldsymbol{w}_1$:

$\boldsymbol{u}_1 = Conv(\boldsymbol{I}, \boldsymbol{w}_1) + b_1$

Padding = 0 and strides = (1,1)

$\boldsymbol{u}_1(1,1) = 0.7\times0 + 0.1\times1 + 0.2\times1 + 0.8\times1 + 0.1\times0 + 0.3\times1 + 1.0\times1 + 0.2\times1 + 0.0\times0 + 0.1 = 2.7$

$\boldsymbol{u}_1(1,2) = 0.1\times0 + 0.2\times1 + 0.3\times1 + 0.1\times1 + 0.3\times0 + 0.5\times1 + 0.2\times1 + 0.0\times1 + 0.3\times0 + 0.1 = 1.4$

$\boldsymbol{u}_1(2,1) = 0.8\times0 + 0.1\times1 + 0.3\times1 + 1.0\times1 + 0.2\times0 + 0.0\times1 + 0.8\times1 + 0.1\times1 + 0.5\times0 + 0.1 = 2.4$

Question 1

$$\boldsymbol{u}_1 = \begin{pmatrix} 2.7 & 1.4 & 1.4 & 1.9 \\ 2.4 & 2.0 & 2.0 & 2.1 \\ 1.7 & 2.0 & 2.6 & 2.6 \\ 2.8 & 2.0 & 3.1 & 2.0 \end{pmatrix}$$

Similarly, $\boldsymbol{u}_2 = Conv(\boldsymbol{I}, \boldsymbol{w}_2) + b_2 = \begin{pmatrix} 0.3 & 0.0 & -0.2 & 2.0 \\ 0.3 & 0.4 & 0.6 & 0.8 \\ -0.1 & 0.8 & 1.1 & -0.3 \\ 0.2 & -0.1 & -0.2 & 0.3 \end{pmatrix}$

Feature maps at the convolutional layer

$$\boldsymbol{y}_1 = f(\boldsymbol{u}_1) = \frac{1}{1 + e^{-\boldsymbol{u}_1}} = \begin{pmatrix} 0.94 & 0.8 & 0.8 & 0.87 \\ 0.92 & 0.88 & 0.88 & 0.89 \\ 0.85 & 0.88 & 0.93 & 0.93 \\ 0.94 & 0.88 & 0.96 & 0.88 \end{pmatrix}$$

$$\boldsymbol{y}_2 = f(\boldsymbol{u}_2) = \frac{1}{1 + e^{-\boldsymbol{u}_2}} = \begin{pmatrix} 0.57 & 0.5 & 0.45 & 0.55 \\ 0.57 & 0.60 & 0.65 & 0.69 \\ 0.48 & 0.69 & 0.75 & 0.43 \\ 0.55 & 0.48 & 0.45 & 0.57 \end{pmatrix}$$

Output spatial size
= (Input size – Filter size +2(Padding))/Stride + 1

= (6 – 3)/1 + 1 = 4

Question 1

Feature maps at the convolutional layer:

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 0.94 & 0.8 & 0.8 & 0.87 \\ 0.92 & 0.88 & 0.88 & 0.89 \\ 0.85 & 0.88 & 0.93 & 0.93 \\ 0.94 & 0.88 & 0.96 & 0.88 \end{pmatrix} \\ \begin{pmatrix} 0.57 & 0.5 & 0.45 & 0.55 \\ 0.57 & 0.60 & 0.65 & 0.69 \\ 0.48 & 0.69 & 0.75 & 0.43 \\ 0.55 & 0.48 & 0.45 & 0.57 \end{pmatrix} \end{pmatrix}$$

Pooling 2×2 and strides = 2

Max-pooling:

$$o = \begin{pmatrix} \begin{pmatrix} 0.94 & 0.89 \\ 0.94 & 0.96 \end{pmatrix} \\ \begin{pmatrix} 0.60 & 0.69 \\ 0.69 & 0.75 \end{pmatrix} \end{pmatrix}$$

Mean-pooling:

$$p_{ave} = \begin{pmatrix} \begin{pmatrix} 0.88 & 0.86 \\ 0.89 & 0.92 \end{pmatrix} \\ \begin{pmatrix} 0.56 & 0.58 \\ 0.55 & 0.55 \end{pmatrix} \end{pmatrix}$$

Question 1

$$\boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \text{ and } b = 0.1.$$

$$\boldsymbol{I} = \begin{pmatrix} 0 & 0 & 0 & & & \\ 0 & 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0 & 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ & 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ & 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ & 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ & 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

Synaptic inputs to the feature map with filter $\boldsymbol{w}_1$:

$\boldsymbol{u}_1 = Conv(\boldsymbol{I}, \boldsymbol{w}_1) + b_1$

Padding = 1 and strides = (2,2)

$\boldsymbol{u}_1(1,1) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.0\times1 + 0.7\times0 + 0.1\times1 + 0.0\times1 + 0.8\times1 + 0.1\times0 + 0.1 = 1.0$

Question 1

$$\boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \text{ and } b = 0.1.$$

$$\boldsymbol{I} = \begin{pmatrix} & 0 & 0 & 0 & & \\ 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

Synaptic inputs to the feature map with filter $\boldsymbol{w}_1$:

$\boldsymbol{u}_1 = Conv(\boldsymbol{I}, \boldsymbol{w}_1) + b_1$

Padding = 1 and strides = (2,2)

$\boldsymbol{u}_1(1,1) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.0\times1 + 0.7\times0 + 0.1\times1 + 0.0\times1 + 0.8\times1 + 0.1\times0 + 0.1 = 1.0$

$\boldsymbol{u}_1(1,2) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.1\times1 + 0.2\times0 + 0.3\times1 + 0.1\times1 + 0.3\times1 + 0.5\times0 + 0.1 = 0.9$

Question 1

$$\boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \text{ and } b = 0.1.$$

$$\boldsymbol{I} = \begin{pmatrix} & & & 0 & 0 & 0 \\ 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

Synaptic inputs to the feature map with filter $\boldsymbol{w}_1$:

$\boldsymbol{u}_1 = Conv(\boldsymbol{I}, \boldsymbol{w}_1) + b_1$

Padding = 1 and strides = (2,2)

$\boldsymbol{u}_1(1,1) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.0\times1 + 0.7\times0 + 0.1\times1 + 0.0\times1 + 0.8\times1 + 0.1\times0 + 0.1 = 1.0$

$\boldsymbol{u}_1(1,2) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.1\times1 + 0.2\times0 + 0.3\times1 + 0.1\times1 + 0.3\times1 + 0.5\times0 + 0.1 = 0.9$

$\boldsymbol{u}_1(1,3) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.3\times1 + 0.3\times0 + 0.5\times1 + 0.5\times1 + 0.1\times1 + 0.0\times0 + 0.1 = 1.5$

Question 1

$$\boldsymbol{w}_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \text{ and } b = 0.1.$$

$$\boldsymbol{I} = \begin{pmatrix} 0.7 & 0.1 & 0.2 & 0.3 & 0.3 & 05 \\ 0 & 0.8 & 0.1 & 0.3 & 0.5 & 0.1 & 0.0 \\ 0 & 1.0 & 0.2 & 0.0 & 0.3 & 0.2 & 0.7 \\ 0 & 0.8 & 0.1 & 0.5 & 0.6 & 0.3 & 0.4 \\ & 0.1 & 0.0 & 0.9 & 0.3 & 0.3 & 0.2 \\ & 1.0 & 0.1 & 0.4 & 0.5 & 0.2 & 0.8 \end{pmatrix}$$

Synaptic inputs to the feature map with filter $\boldsymbol{w}_1$:
$\boldsymbol{u}_1 = Conv(\boldsymbol{I}, \boldsymbol{w}_1) + b_1$

Padding = 1 and strides = (2,2)

$$\boldsymbol{u}_1(1,1) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.0\times1 + 0.7\times0 + 0.1\times1 + 0.0\times1 + 0.8\times1 + 0.1\times0 + 0.1 = 1.0$$

$$\boldsymbol{u}_1(1,2) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.1\times1 + 0.2\times0 + 0.3\times1 + 0.1\times1 + 0.3\times1 + 0.5\times0 + 0.1 = 0.9$$

$$\boldsymbol{u}_1(1,3) = 0.0\times0 + 0.0\times1 + 0.0\times1 + 0.3\times1 + 0.3\times0 + 0.5\times1 + 0.5\times1 + 0.1\times1 + 0.0\times0 + 0.1 = 1.5$$

$$\boldsymbol{u}_1(2,1) = 0.0\times0 + 0.8\times1 + 0.1\times1 + 0.0\times1 + 1.0\times0 + 0.2\times1 + 0.0\times1 + 0.8\times1 + 0.1\times0 + 0.1 = 2.0$$

Question 1

$$\boldsymbol{u}_1 = \begin{pmatrix} 1.0 & 0.9 & 1.5 \\ 2.0 & 2.0 & 2.1 \\ 2.0 & 2.0 & 2.0 \end{pmatrix}$$

Similarly, $\boldsymbol{u}_2 = Conv(\boldsymbol{I}, \boldsymbol{w}_2) + b_2 = \begin{pmatrix} 1.0 & 1.0 & 0.7 \\ 0.1 & 0.4 & 0.8 \\ 0.3 & -0.1 & 0.3 \end{pmatrix}$

Feature maps at the convolutional layer

$$\boldsymbol{y}_1 = f(\boldsymbol{u}_1) = \frac{1}{1 + e^{-\boldsymbol{u}_1}} = \begin{pmatrix} 0.73 & 0.71 & 0.82 \\ 0.88 & 0.88 & 0.89 \\ 0.88 & 0.88 & 0.88 \end{pmatrix}$$

$$\boldsymbol{y}_2 = f(\boldsymbol{u}_2) = \frac{1}{1 + e^{-\boldsymbol{u}_2}} = \begin{pmatrix} 0.73 & 0.73 & 0.67 \\ 0.52 & 0.60 & 0.69 \\ 0.57 & 0.48 & 0.57 \end{pmatrix}$$

Question 1

Feature maps at the convolutional layer:

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{pmatrix} = \left(\begin{pmatrix} 0.73 & 0.71 & 0.82 \\ 0.88 & 0.88 & 0.89 \\ 0.88 & 0.88 & 0.88 \end{pmatrix} \begin{pmatrix} 0.73 & 0.73 & 0.67 \\ 0.52 & 0.60 & 0.69 \\ 0.57 & 0.48 & 0.57 \end{pmatrix} \right)$$

Pooling VALID 2x2 strides = 2:

Max-pooling:

$$o = \begin{pmatrix} (0.88) \\ (0.73) \end{pmatrix}$$

Mean-pooling:

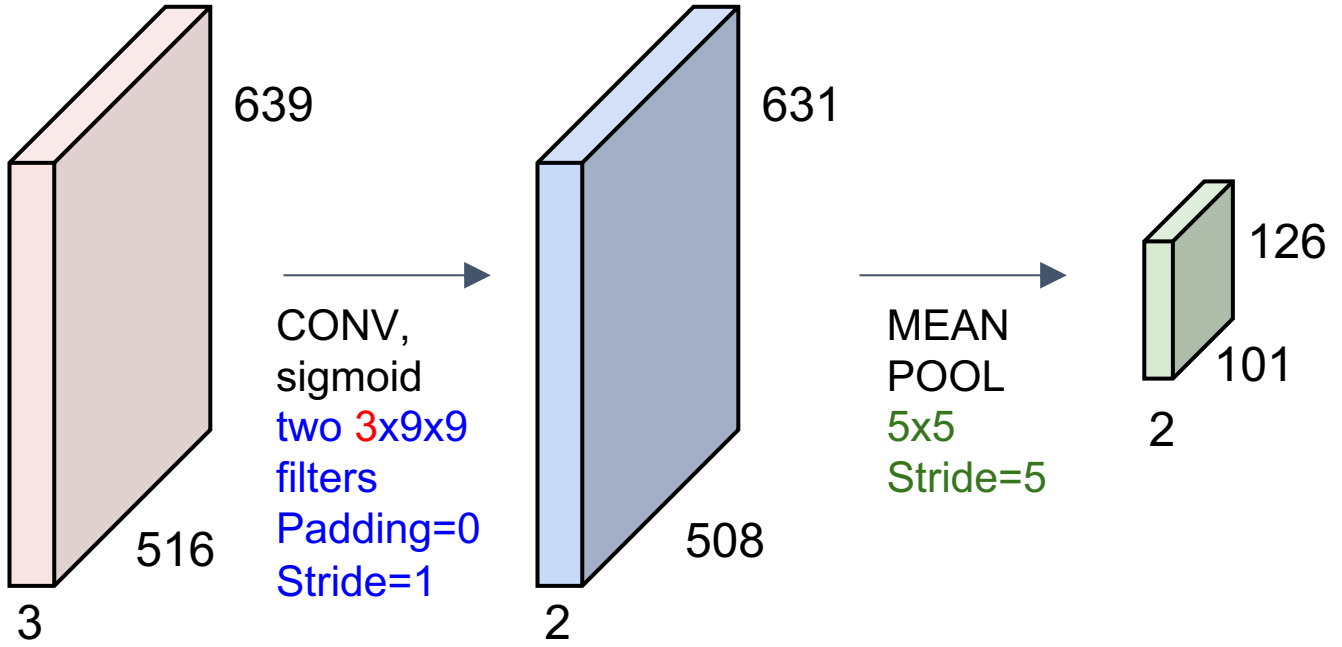
$$p_{ave} = \begin{pmatrix} (0.80) \\ (0.65) \end{pmatrix}$$

Question 2

2. Given '3wolfmoon.jpg' color image of size 639×516.
   a. Initialize weights and biases of a convolutional layer with two kernels of size 9×9. Note that the input image is in color and has three channels.
   b. Display the feature maps at the convolution layer, assuming sigmoid activation functions. Use VALID padding and strides = 1.
   c. Display the outputs of a mean pooling layer with a pooling window size 5×5 and strides = 5.

t6q2.ipynb

Question 2


639
516
3
CONV,
sigmoid
two 3x9x9
filters
Padding=0
Stride=1
631
508
2
MEAN
POOL
5x5
Stride=5
126
101
2

Question 2

```
# open image and normalize
img = Image.open('3wolfmoon.jpg')
img = np.asarray(img, dtype='float32') / 256.
print(img.shape)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-becdc6409bf5> in <module>()
      1 # open image and normalize
----> 2 img = Image.open('3wolfmoon.jpg')
      3 img = np.asarray(img, dtype='float32') / 256.
      4 print(img.shape)

/usr/local/lib/python3.7/dist-packages/PIL/Image.py in open(fp, mode)
   2841
   2842     if filename:
-> 2843         fp = builtins.open(filename, "rb")
   2844         exclusive_fp = True
   2845

FileNotFoundError: [Errno 2] No such file or directory: '3wolfmoon.jpg'
```

You need to mount Google Drive locally and use the correct path

https://colab.research.google.com/notebooks/io.ipynb#scrollTo=u22w3BFiOveA

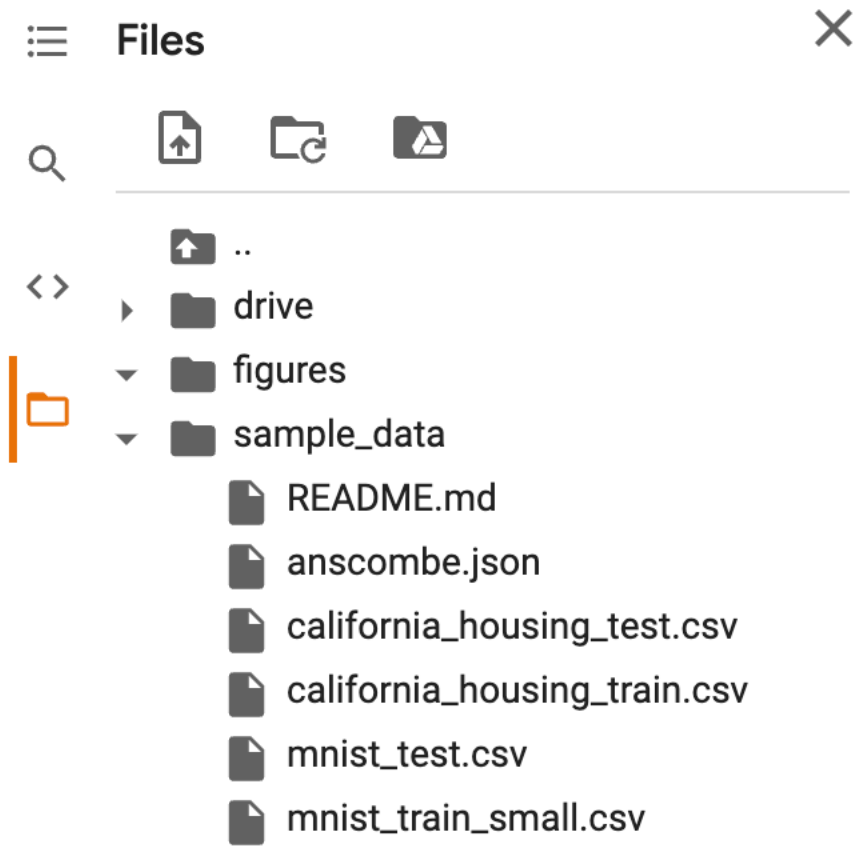
Question 2

```
# mount google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# open image and normalize
img = Image.open('/content/drive/MyDrive/3wolfmoon.jpg')
img = np.asarray(img, dtype='float32') / 256.
print(img.shape)
```

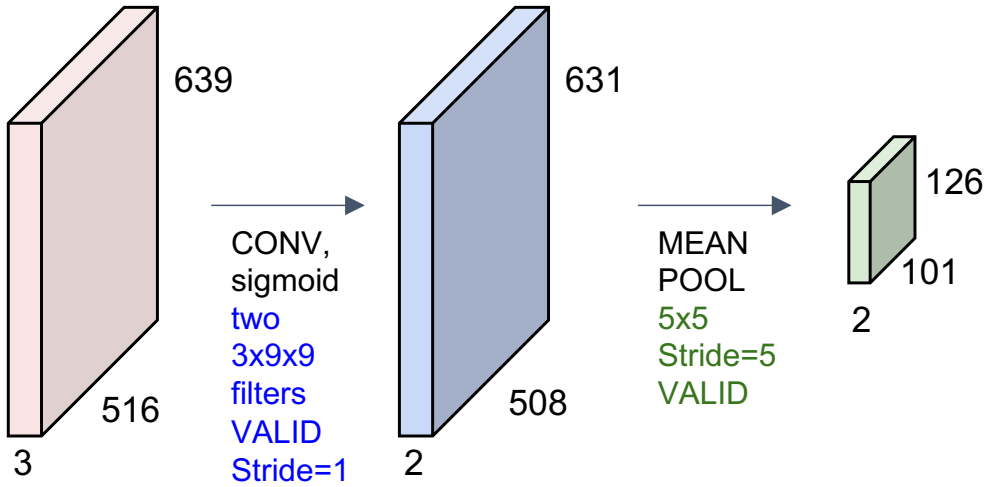
```
(639, 516, 3)
```

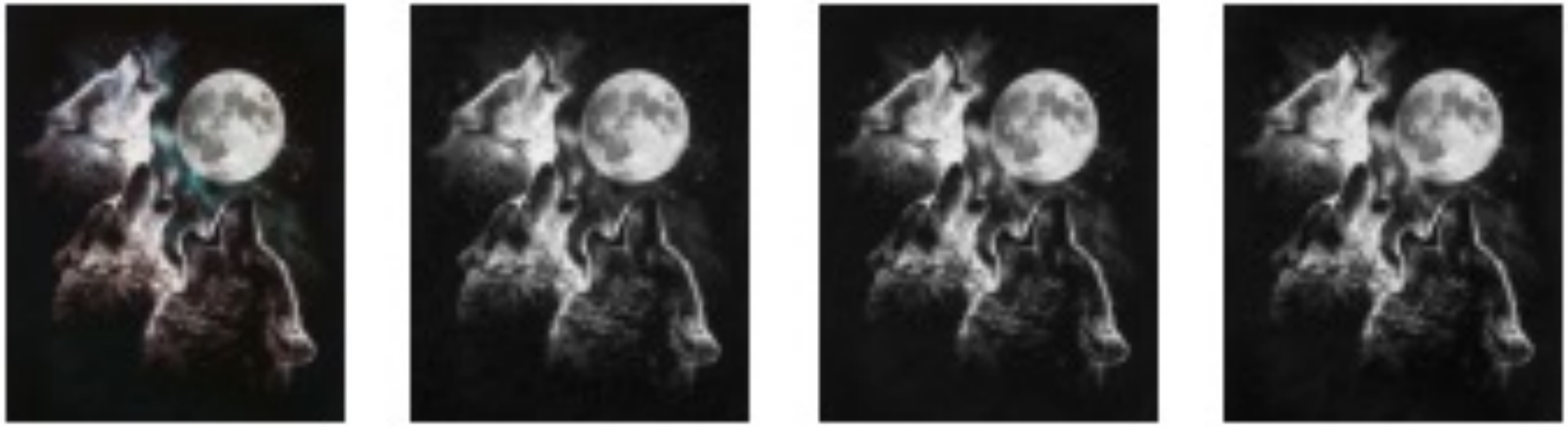


You can copy the path from the browser

Question 2

Plot the three channels of the input image

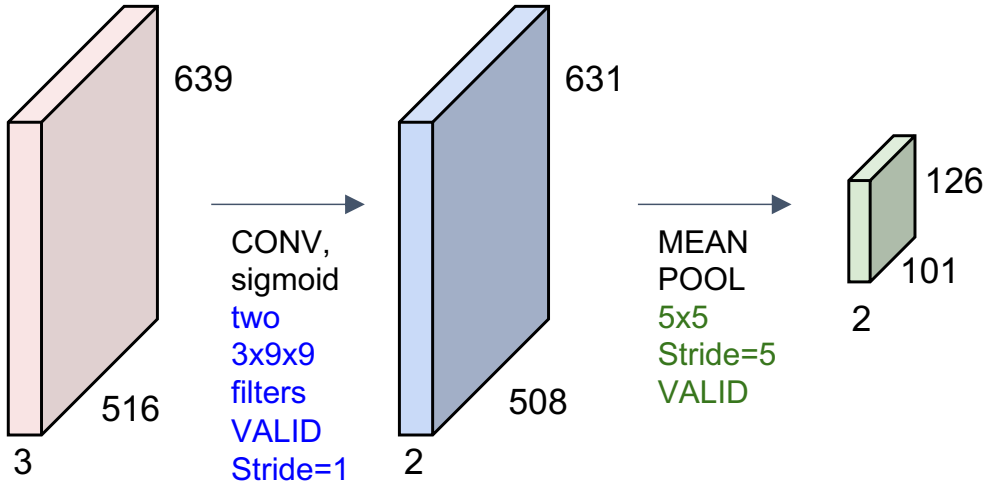




Original image | Red | Green | Blue

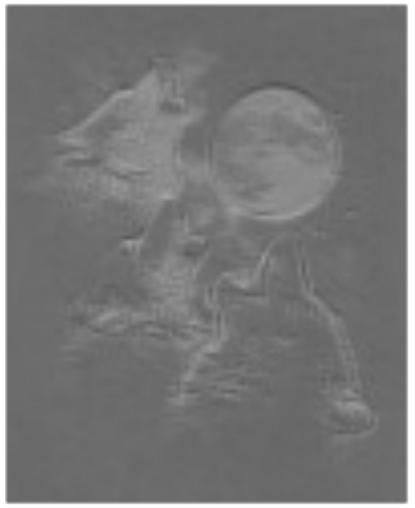
Question 2

Plot the original image and the first and second components of conv output



Original image

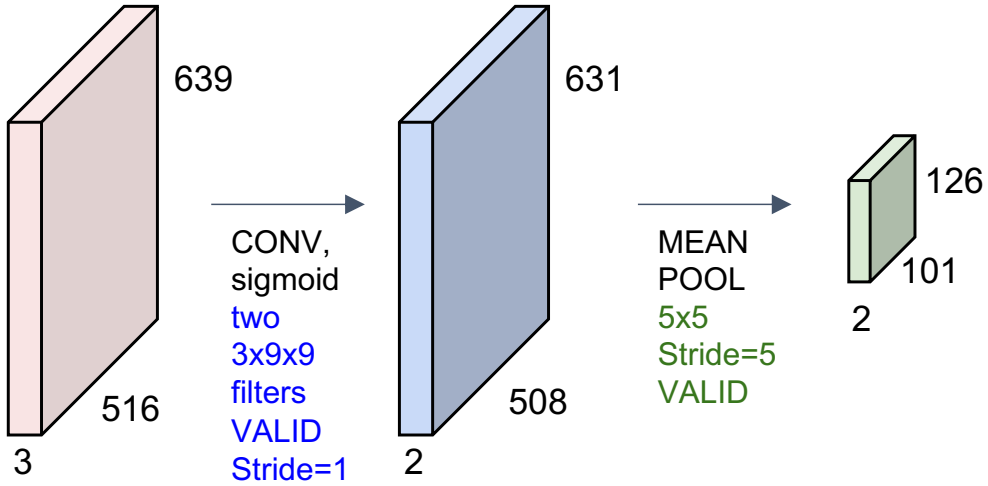
Feature map 1



Feature map 2

Question 2

Plot the original image and the first and second components of pooling output



Original image

Pooled feature map 1

Pooled feature map 2

Question 3

3. Design a CNN with one hidden layer to recognize digit images in MNIST database:

   http://yann.lecun.com/exdb/mnist/

   The convolution layer consists of 25 filters of dimensions 9x9 and the pooling layer has a pooling window size 4x4. Assume VALID padding and default strides for both convolution and pooling layer. Train the network with mini batch gradient decent learning with learning factor $\alpha = 10^{-3}$ and batch size = 128.

   Plot

   a. The training and test errors against learning epochs.
   b. Final filter weights
   c. Feature maps at the convolution and pooling layers for a representative test pattern
   d. Repeat training by introducing decay parameter $\beta = 10^{-6}$ and momentum term with $\gamma = 0.5$, and compare the learning curves

t6q3a.ipynb

t6q3b.ipynb

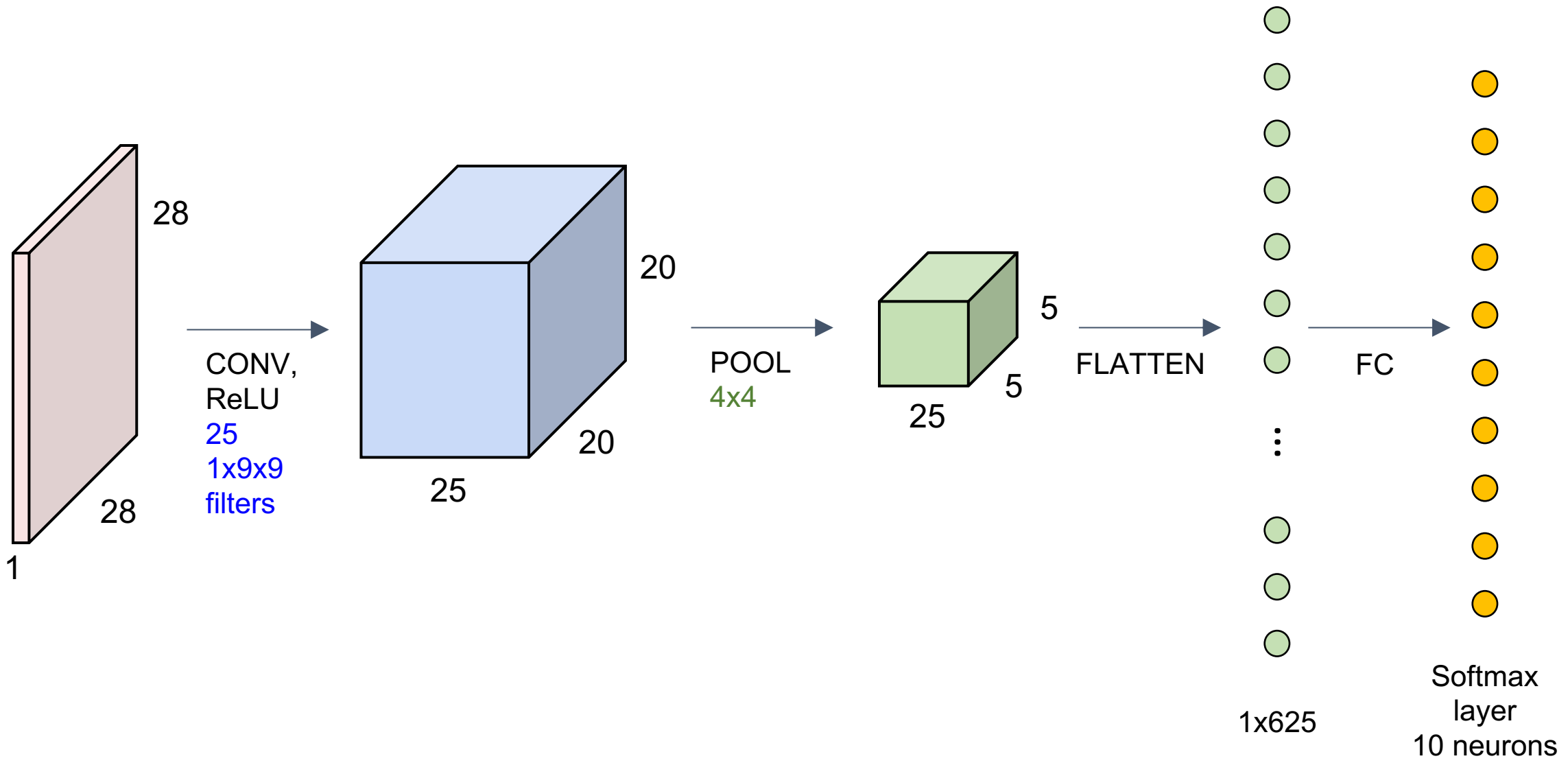
Question 3

MNIST

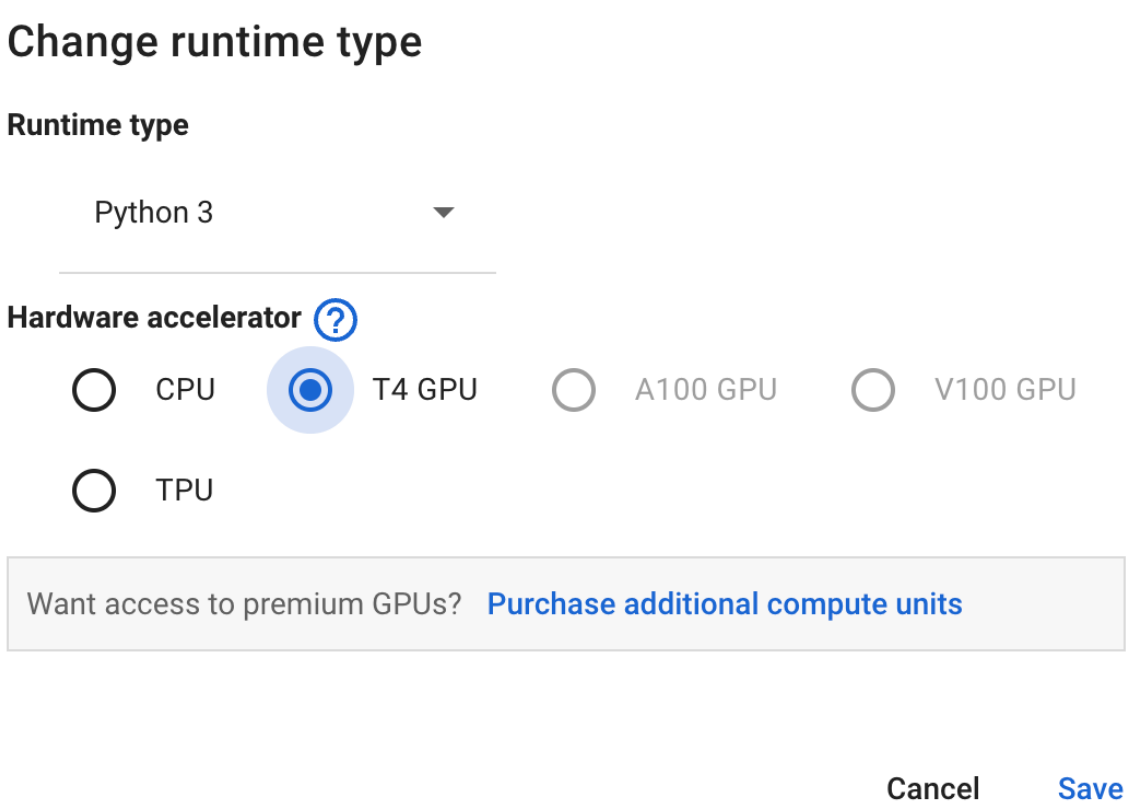
- Size-normalized and centred 1x28x28 =784 inputs
- Training set = 60,000 images
- Testing set = 10,000 images

http://yann.lecun.com/exdb/mnist/

Question 3

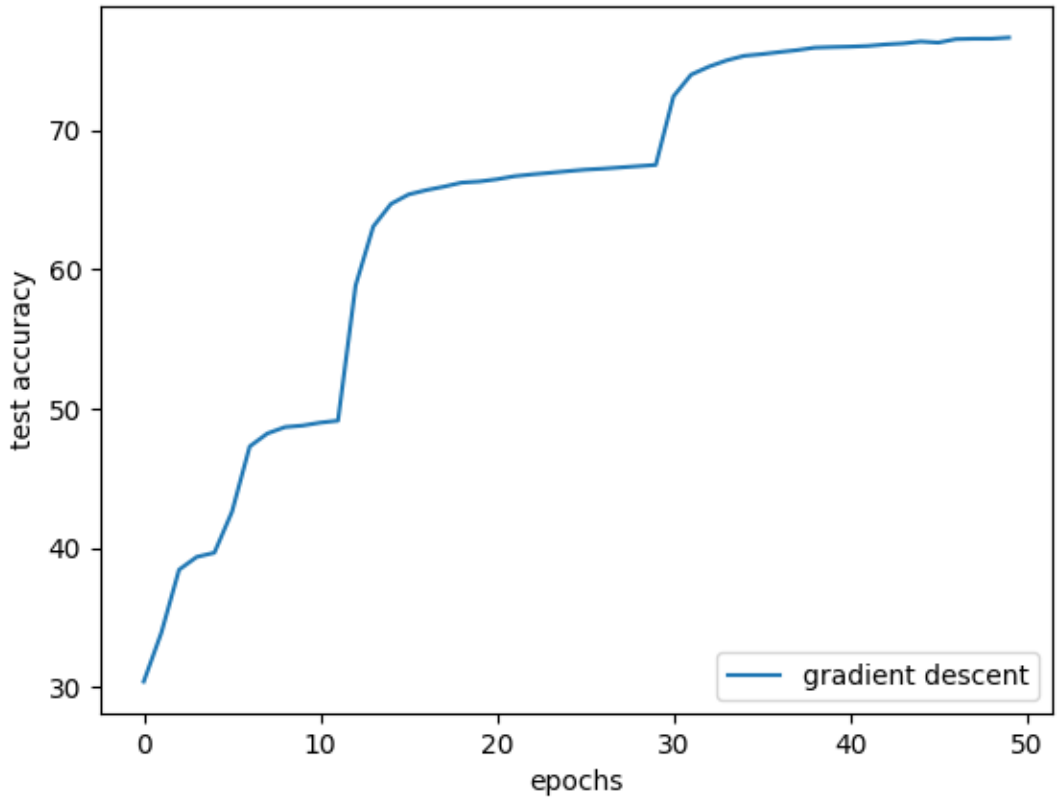

28
28
1
CONV,
ReLU
25
1x9x9
filters
20
20
25
POOL
4x4
5
5
25
FLATTEN
FC
1x625
Softmax
layer
10 neurons

Question 3


Change runtime type
Runtime type
Python 3
Hardware accelerator
CPU
T4 GPU
A100 GPU
V100 GPU
TPU
Want access to premium GPUs?
Purchase additional compute units
Cancel
Save

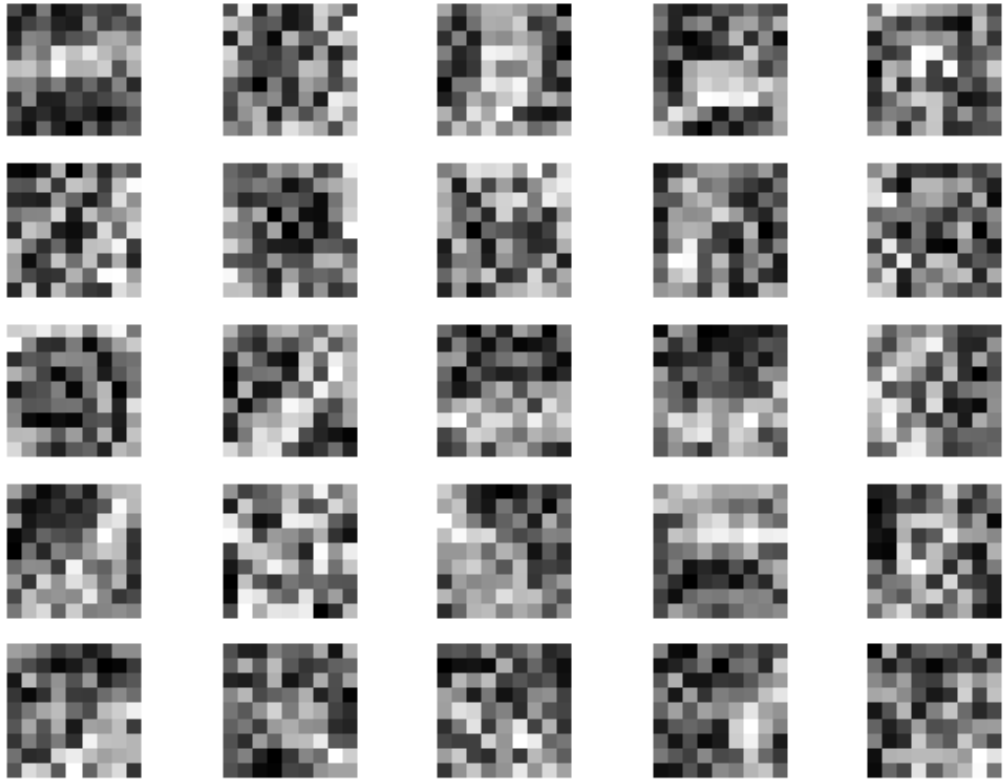
Question 3a

Plot the learning curve (I only show the test curve) - you should be able get better learning curve if you run for more epochs

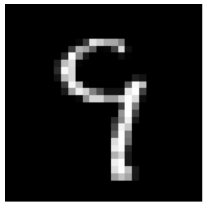


Question 3b

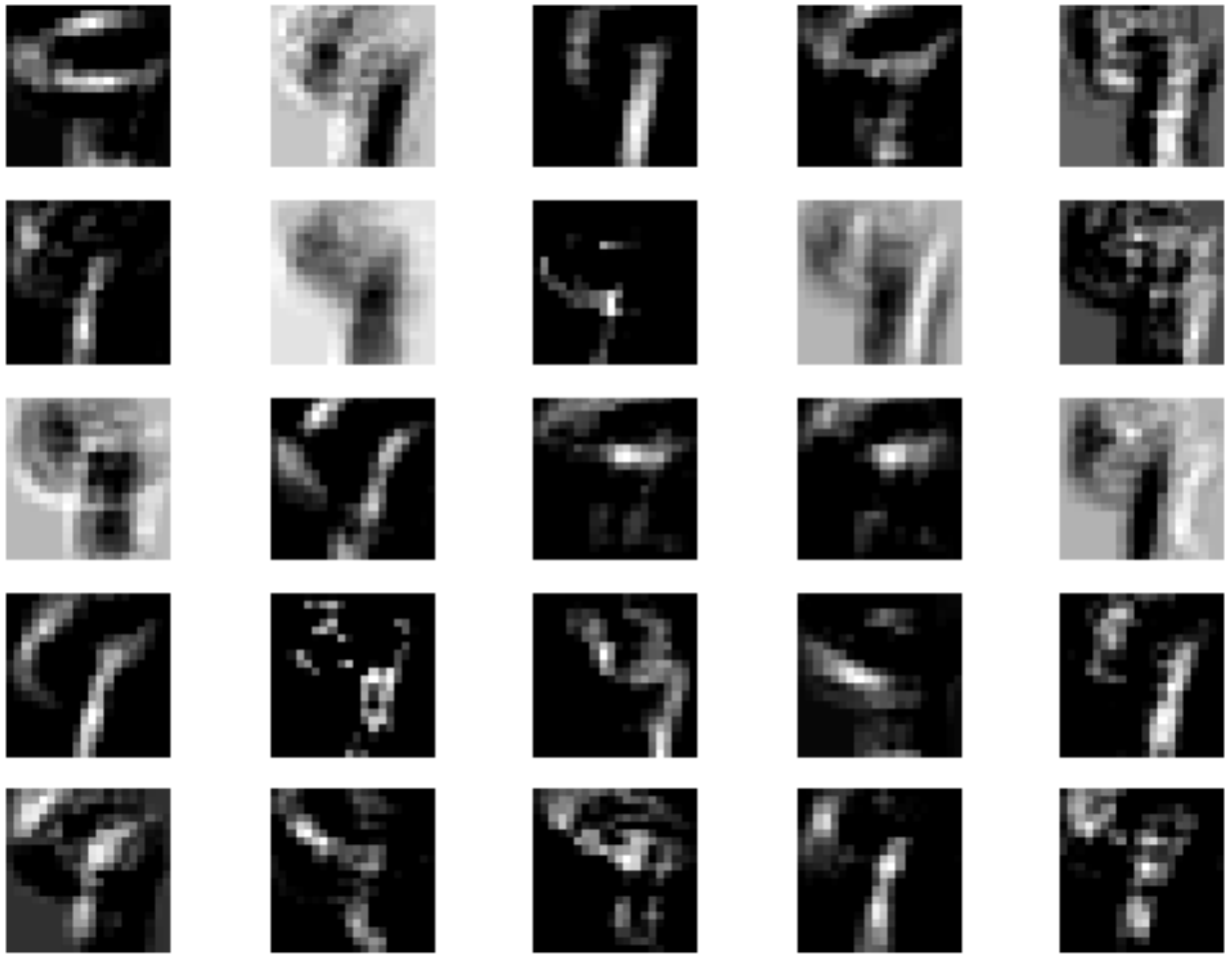
Plot filters learned in the conv layer



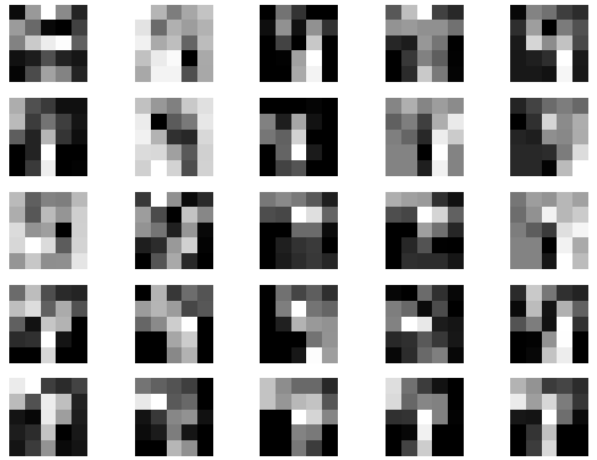
Question 3c



Input 28x28

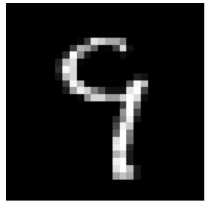


Feature maps of the
convolution layer 25x20x20



Feature maps of
pooling layer 25x5x5

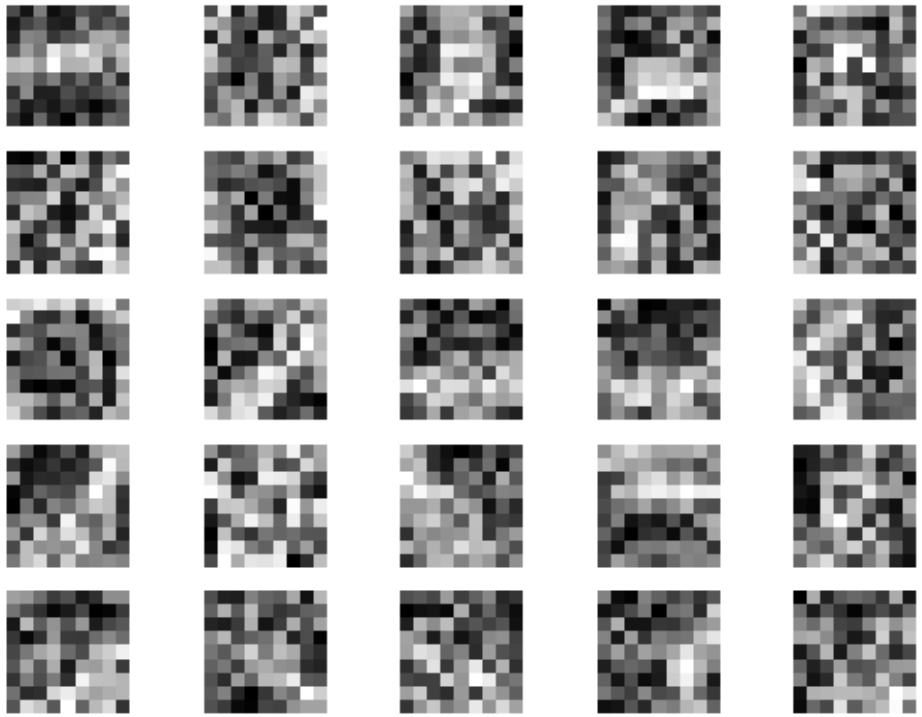
Question 3c



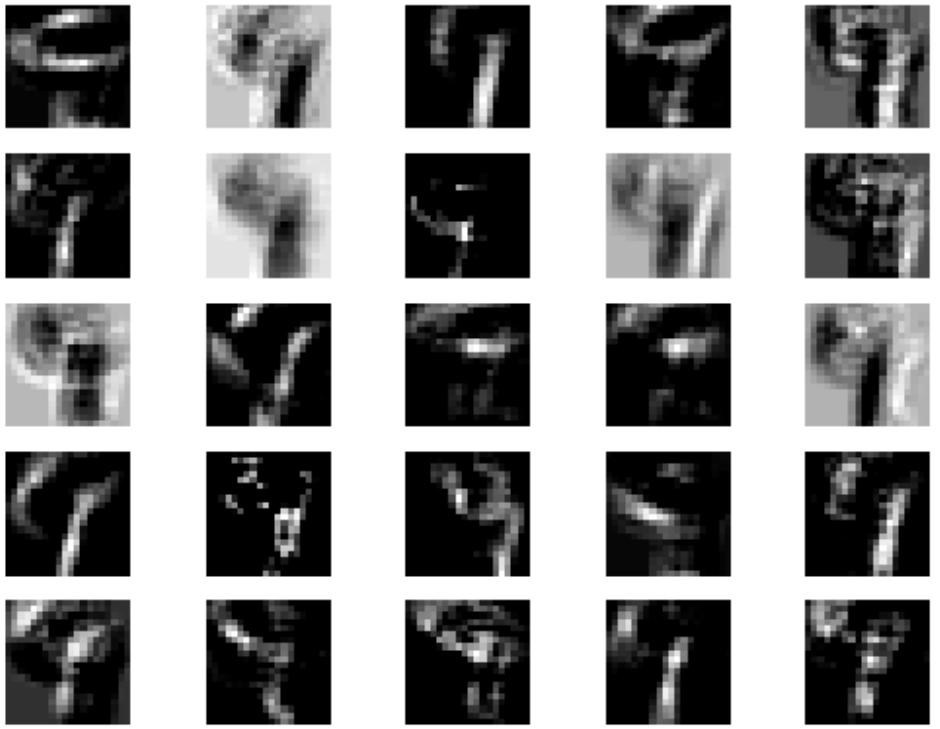
Input 28x28

Compare the filters learned and the corresponding feature maps.

What do you see?



Filters learned in the conv layer



Feature maps of the convolution layer 25x20x20

Question 3d

Gradient descent:

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \nabla_{\boldsymbol{W}} J$$

Weight decay (regularization):

$$J_1(\boldsymbol{W}, \boldsymbol{b}) = J(\boldsymbol{W}, \boldsymbol{b}) + \beta_2 \sum_{ij} (w_{ij})^2$$

Weight decay:

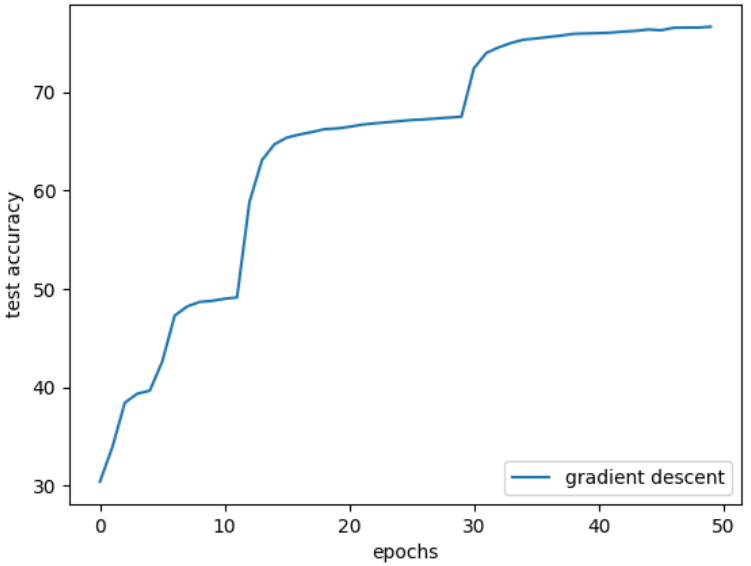
$$\boldsymbol{W} = \boldsymbol{W} - \alpha(\nabla_{\boldsymbol{W}} J + \beta \boldsymbol{W})$$

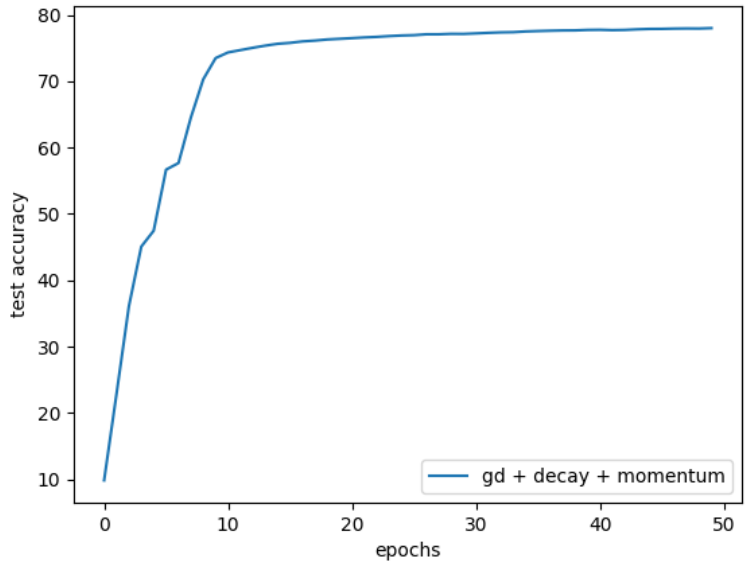
Sgd with momentum:

$$\boldsymbol{V} = \gamma \boldsymbol{V} - \alpha \nabla_{\boldsymbol{W}} J$$
$$\boldsymbol{W} = \boldsymbol{W} + \boldsymbol{V}$$

Sgd with decay and momentum:

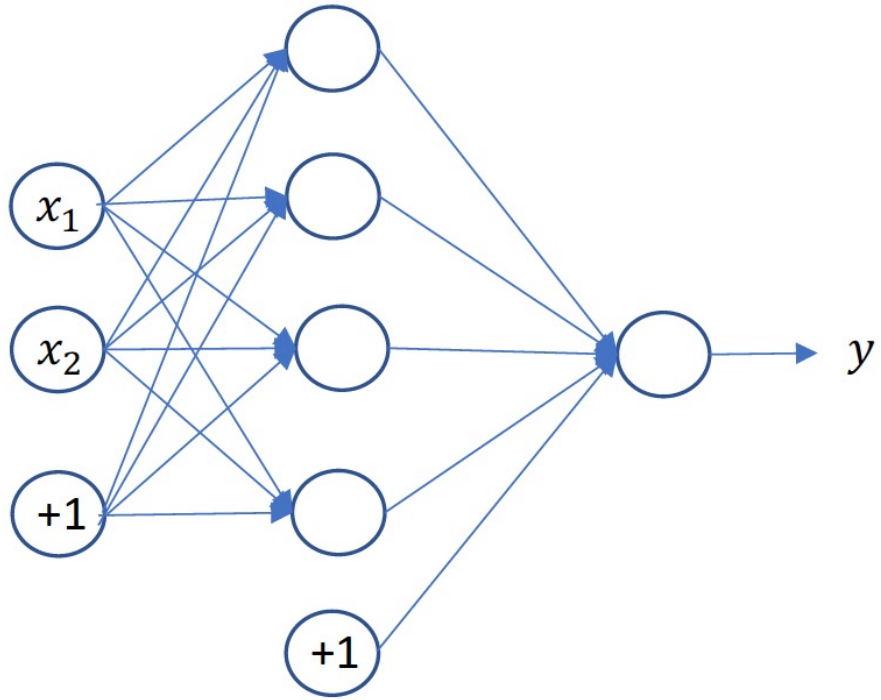
$$\boldsymbol{V} = \gamma \boldsymbol{V} - \alpha(\nabla_{\boldsymbol{W}} J + \beta \boldsymbol{W})$$
$$\boldsymbol{W} = \boldsymbol{W} + \boldsymbol{V}$$





Model selection and overfitting

CS4001 – Tutorial 5



1. A feedforward network consisting of one hidden layer of perceptrons and a linear output neuron receives inputs two-dimensional inputs (x_1, x_2). Train the network to predict the following function:

$$y = sin(\pi x_1)cos(2\pi x_2)$$

where $-1.0 \leq x_1, x_2 \leq +1.0$.

Use data points distributed in an equally spaced 10x10 grid spanning the input space and the following procedures for training and testing:

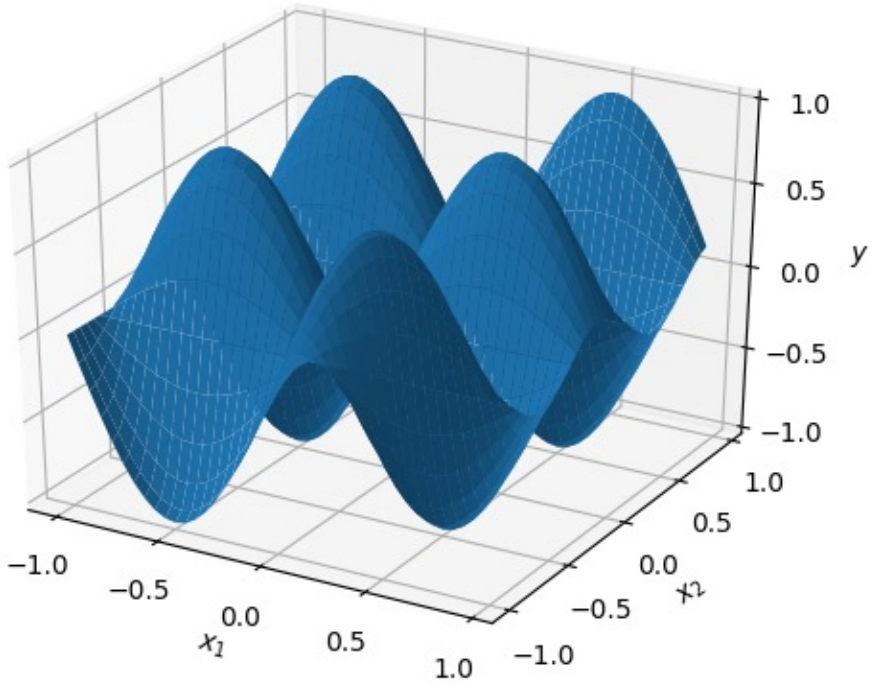
a) Random subsampling with 70:30 data-split for training and testing.
b) Five-fold cross validation
c) Three-way data split

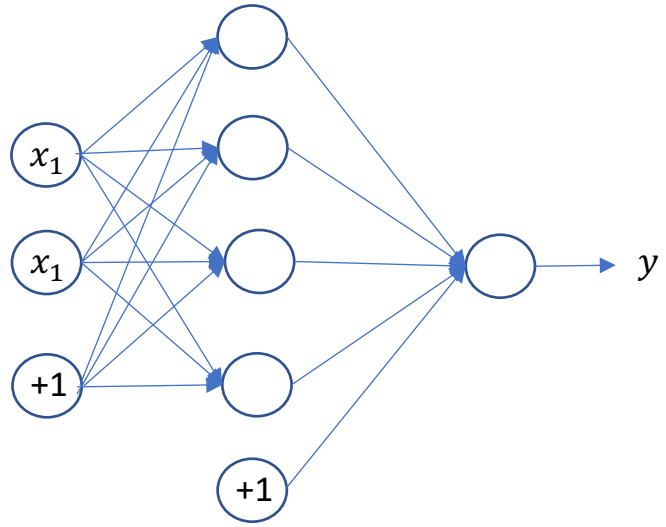
Repeat each procedure in 10 different experiments and determine the optimal number of hidden neurons in the space of $\{2, 4, 6, 8, 10\}$ and the error of the model.

Use a learning factor $\alpha = 0.05$ and early stopping to cease the learning epochs.

$y = sin(\pi x_1)\cos(2\pi x_2)$ where $-1.0 \leq x_1, x_2 \leq +1.0$.

Note that $-1.0 \leq y \leq +1$.

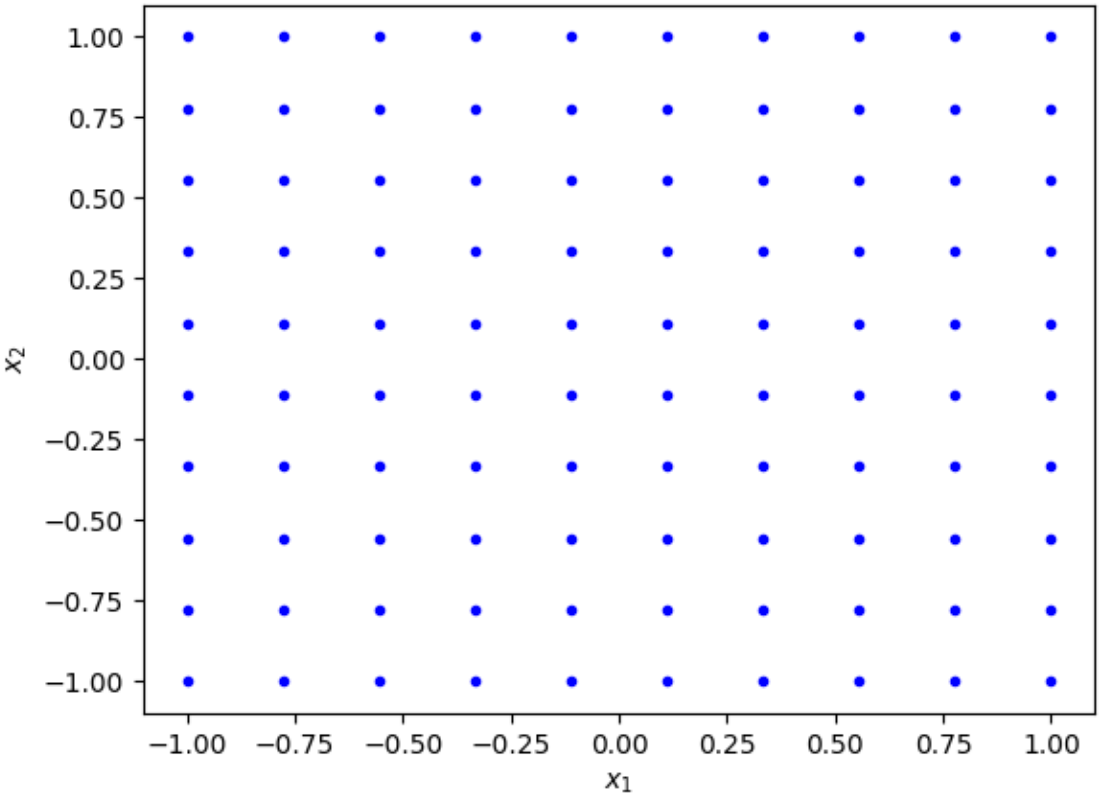


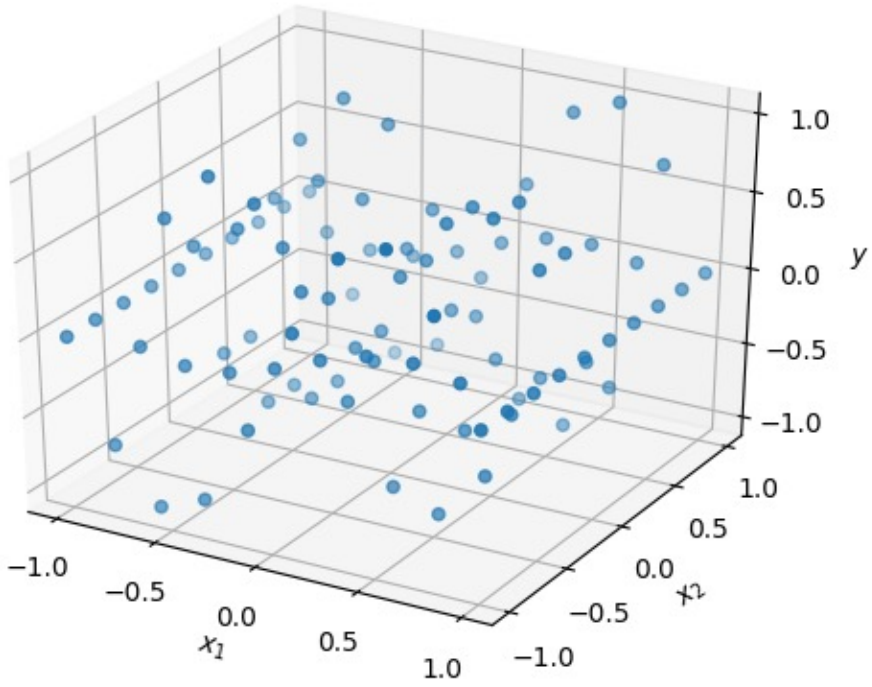


Hidden neurons perceptrons
the output neurons is a linear neuron.

$$y = sin(\pi x_1)\cos(2\pi x_2) \quad \text{where } -1.0 \leq x_1, x_2 \leq +1.0.$$

Training data is in an input grid of 10x10

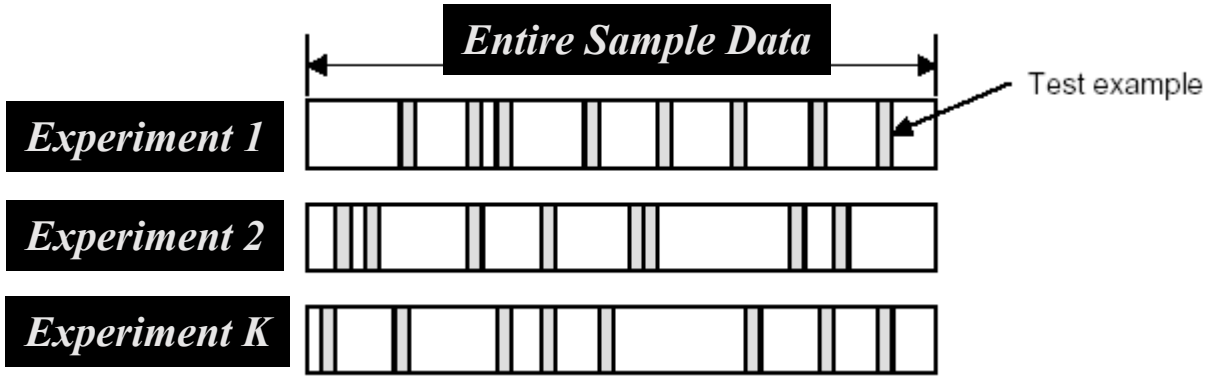




```
no_labels = 1
no_features = 2
no_exps = 10


class MLP(nn.Module):
    def __init__(self, no_features, no_hidden, no_labels):
        super().__init__()
        self.mlp_stack = nn.Sequential(
            nn.Linear(no_features, no_hidden),
            nn.Sigmoid(),
            nn.Linear(no_hidden, no_labels),
        )

    def forward(self, x):
        logits = self.mlp_stack(x)
        return logits
```

```
class EarlyStopper:
    def __init__(self, patience=10, min_delta=0):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.min_validation_loss = np.inf

    def early_stop(self, validation_loss):
        if validation_loss < self.min_validation_loss:
            self.min_validation_loss = validation_loss
            self.counter = 0
        elif validation_loss > (self.min_validation_loss + self.min_delta):
            self.counter += 1
            if self.counter >= self.patience:
                return True
        return False
```

K Data Splits: Random Subsampling



For each experiment k:

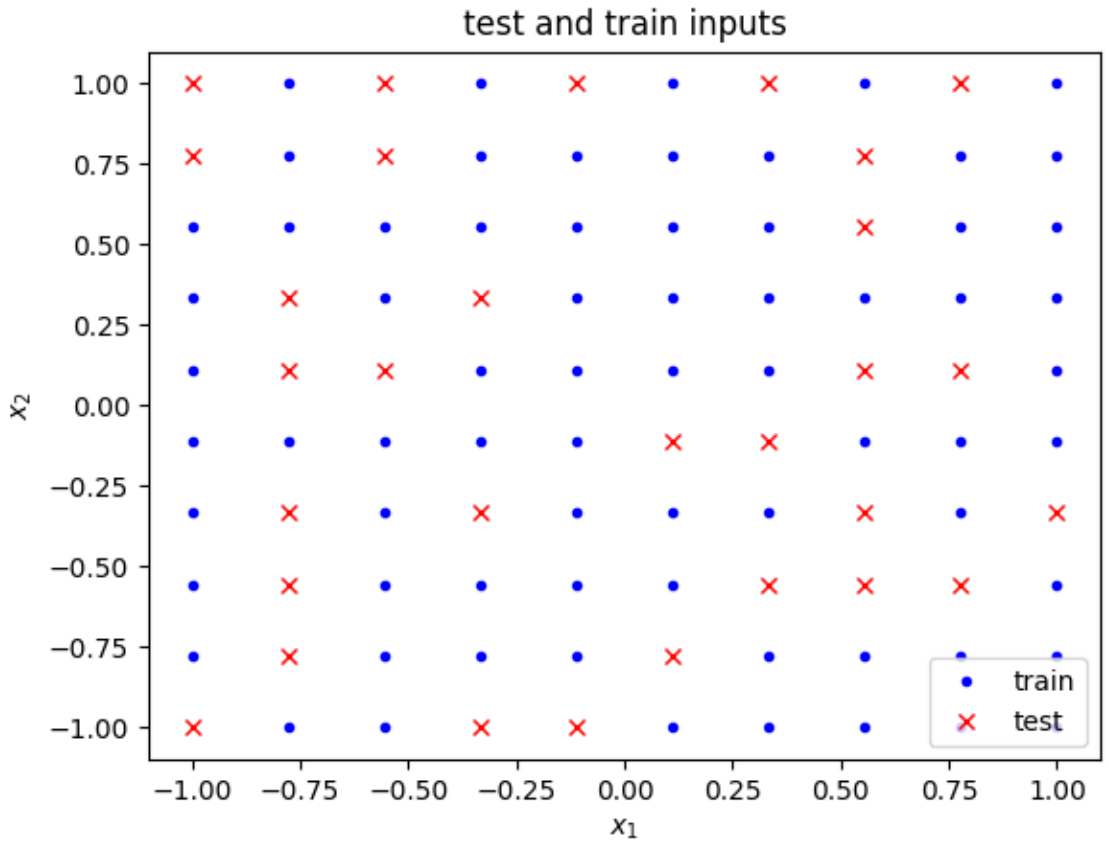
For each model m:

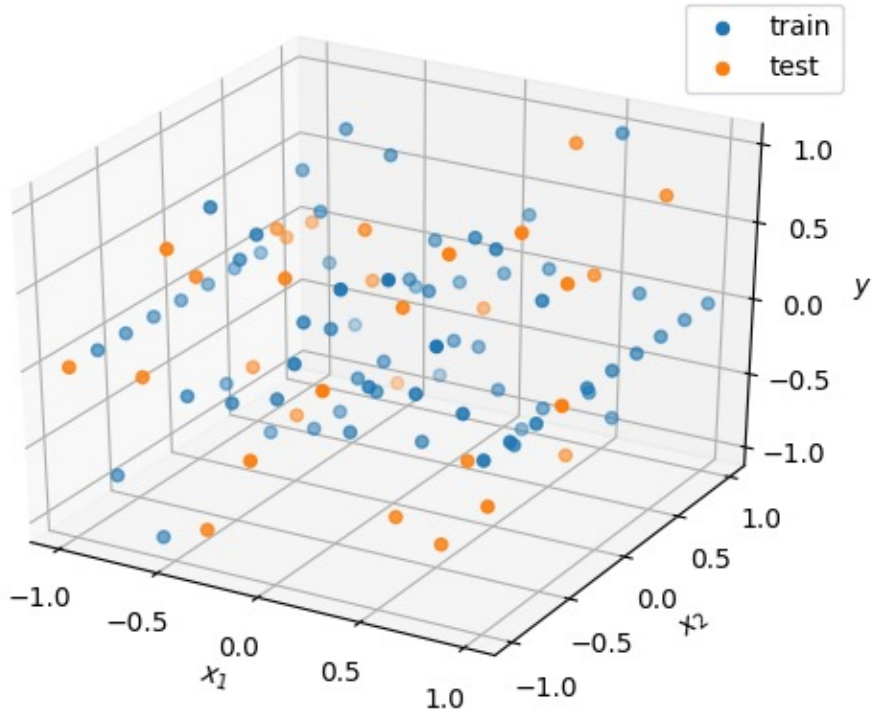
Compute error $e_{k,m}$

Compute mean error $e_m = \frac{1}{K}\sum_{k=1}^{K} e_{k,m}$

Optimal model $m^* = argmin\ e_m$

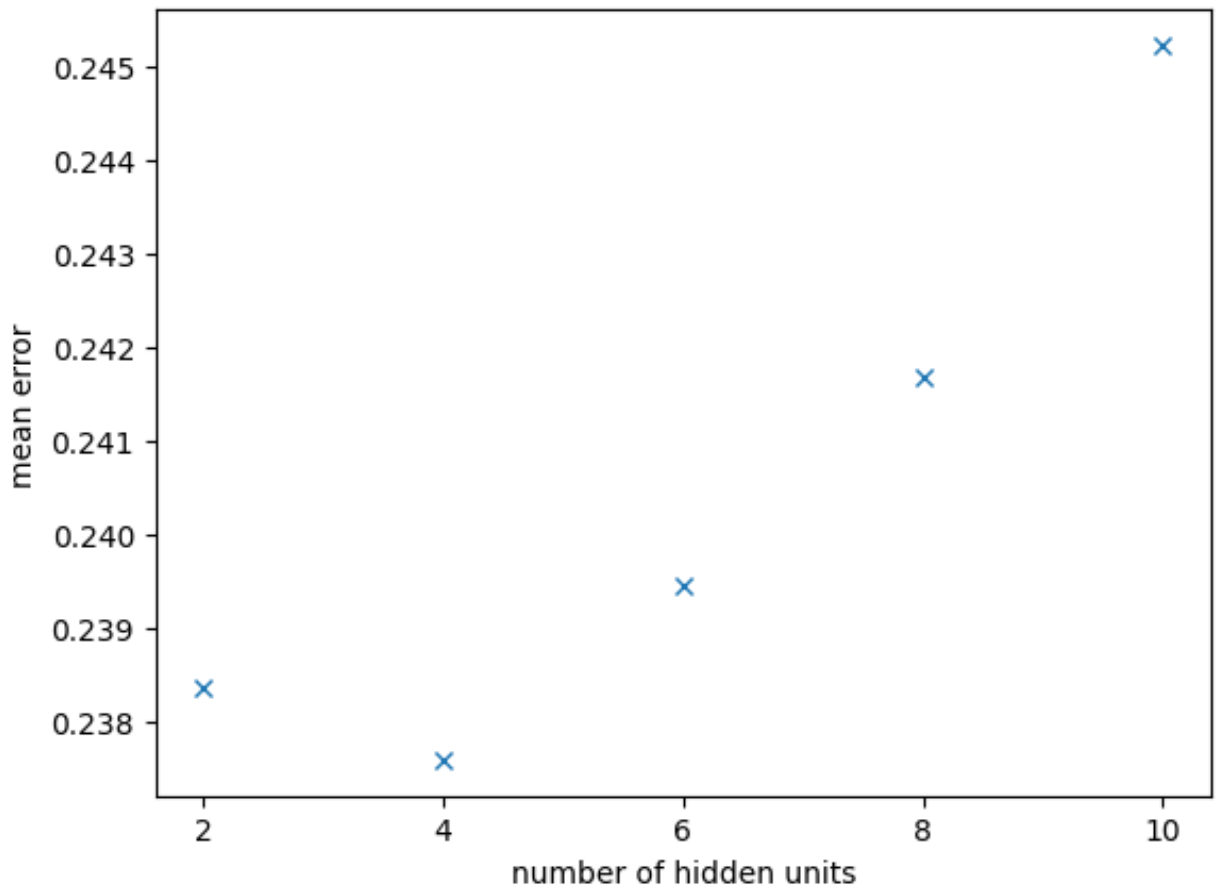
For each experiment, data is split into train and test data at [30: 70]





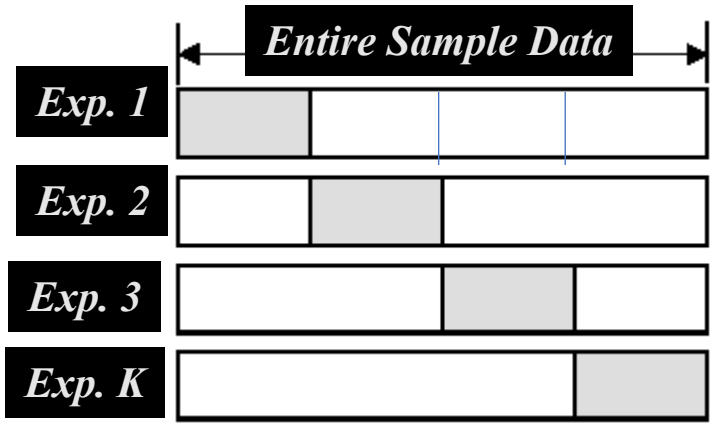
For each experiment, we build models with hidden neurons $\{2, 4, 6, 8, 10\}$

Mean error of 10 experiments



Optimum number of hidden neurons = **4**

K-fold Cross Validation



For every fold f:

For every model m

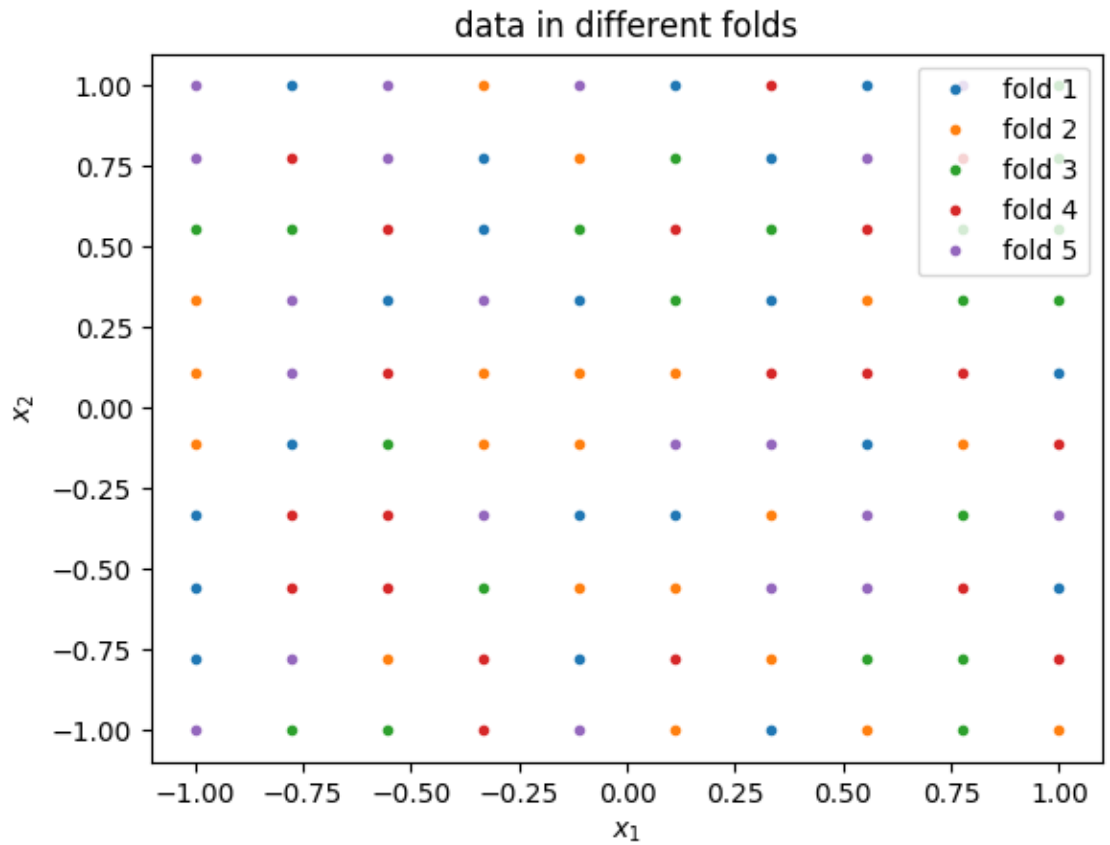
Train the model, using data not in fold f

Compute error $e_{m,f}$ on data in fold f

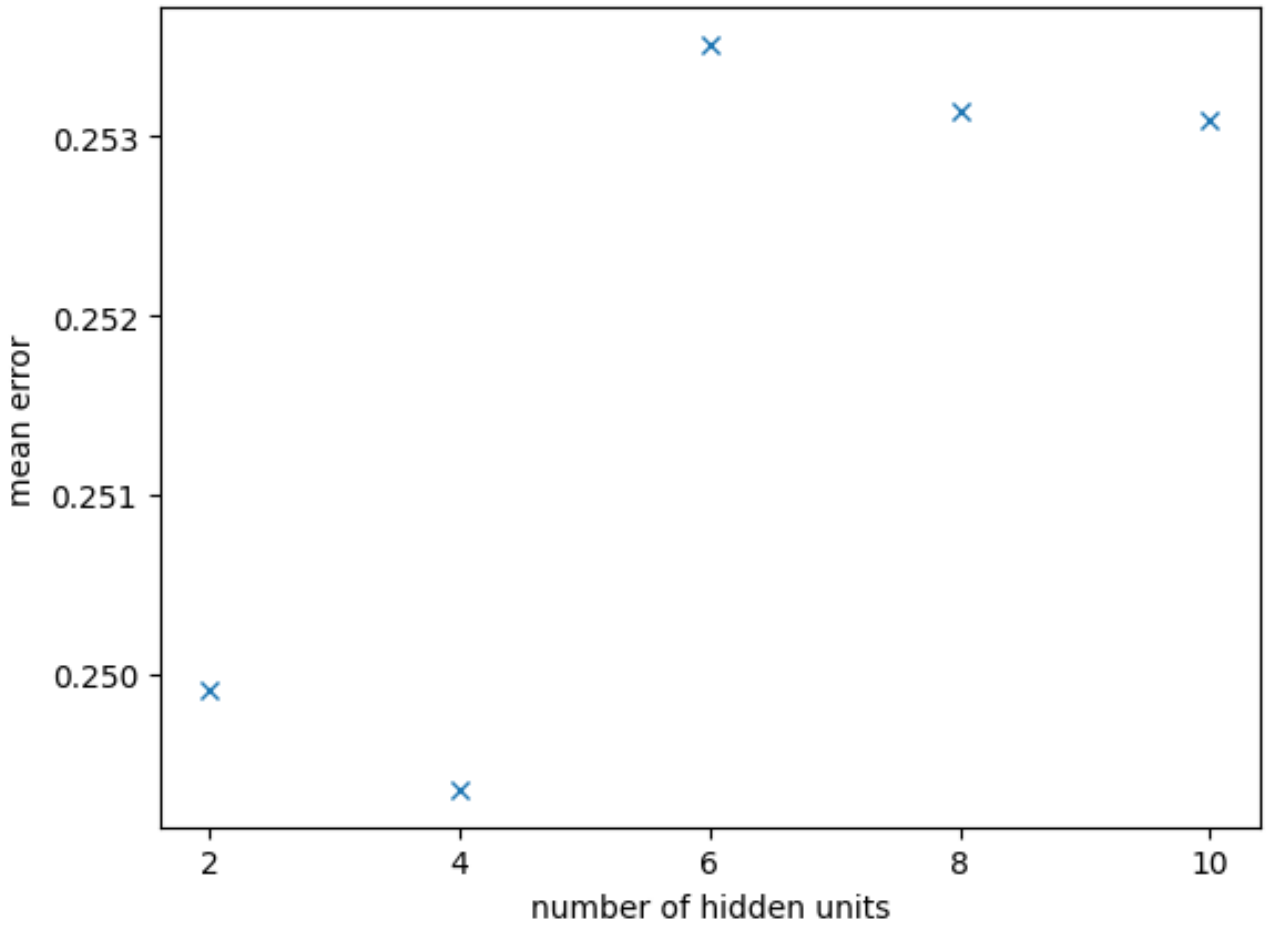
For every model m

CV error $e_m = \frac{1}{F}\sum_f e_{m,f}$

Select the model with minimum CV error, $m^* = argmin\ e_m$

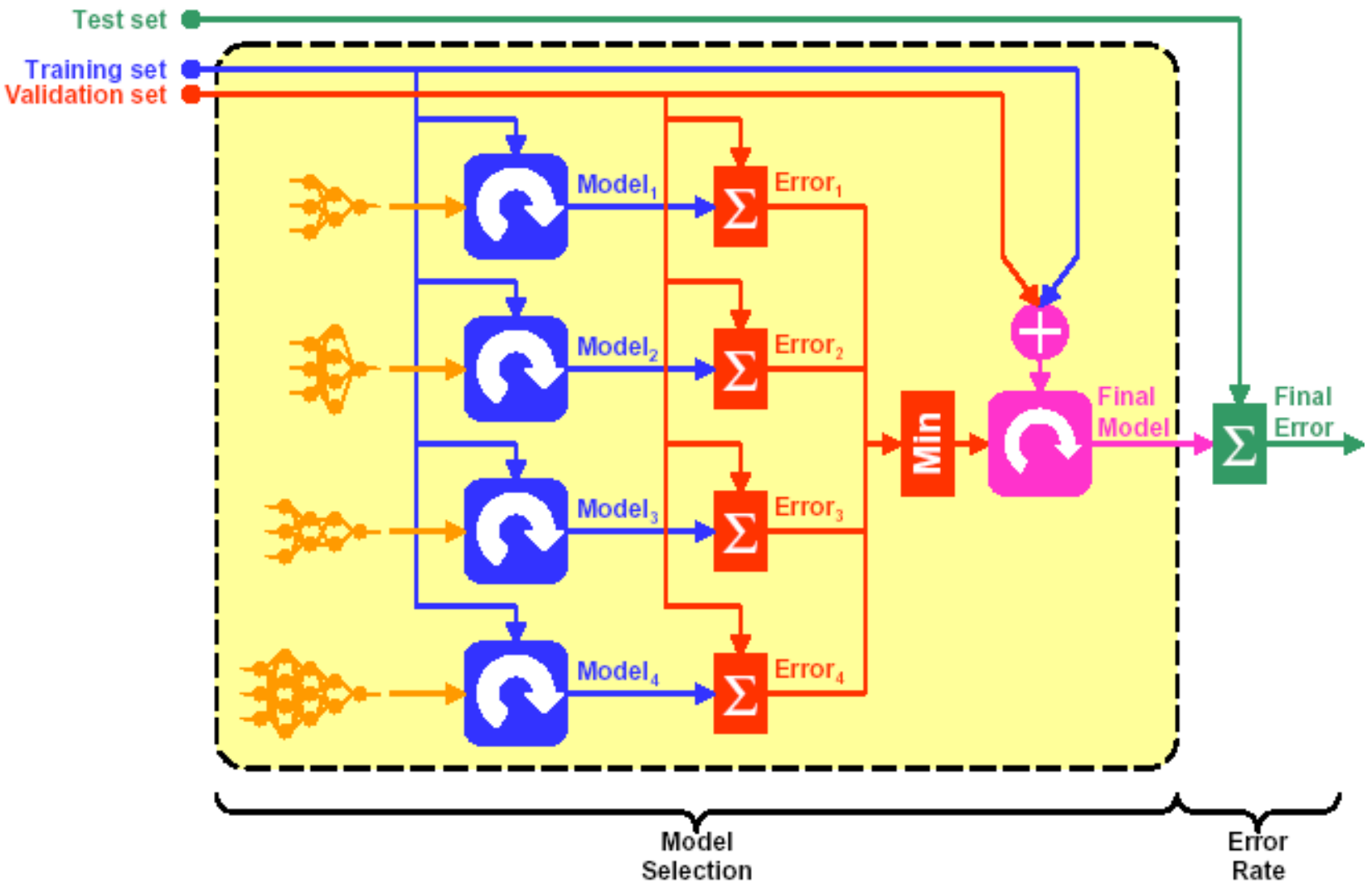

data in different folds
fold 1
fold 2
fold 3
fold 4
fold 5
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x_2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x_1

Mean cross-validation error of 10 experiments



Optimum number of hidden neurons = **4**

Three-Way Data Splits Method



Three-Way Data Splits Method

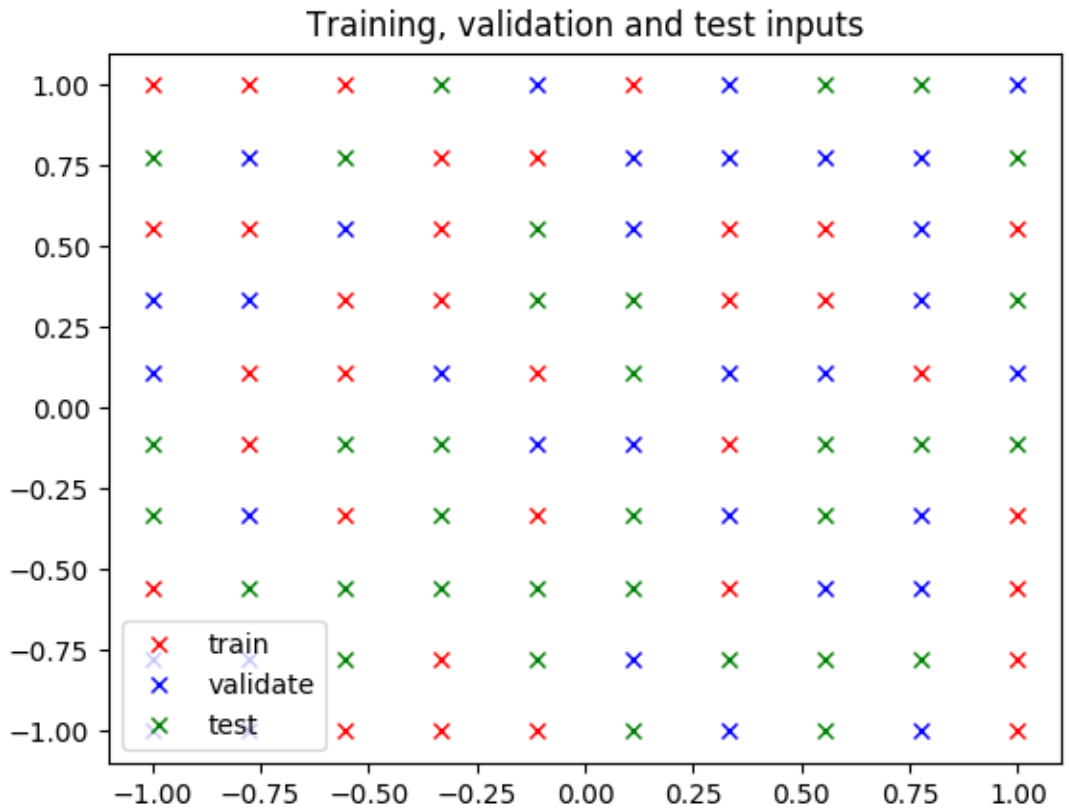
- **Training set**: Each model m, train the network and find the minimum error:

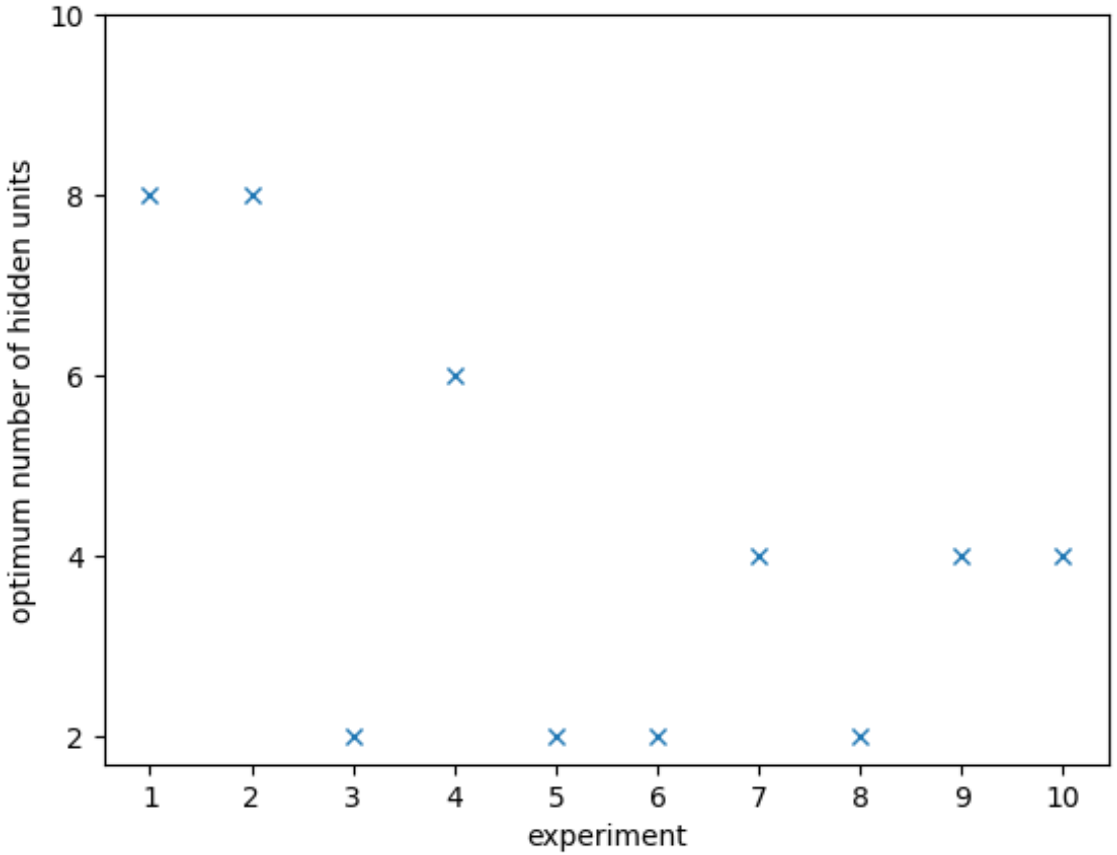
$$\boldsymbol{W}_m, \boldsymbol{b}_m = \underset{\boldsymbol{W},\boldsymbol{b}}{\operatorname{argmin}} J$$

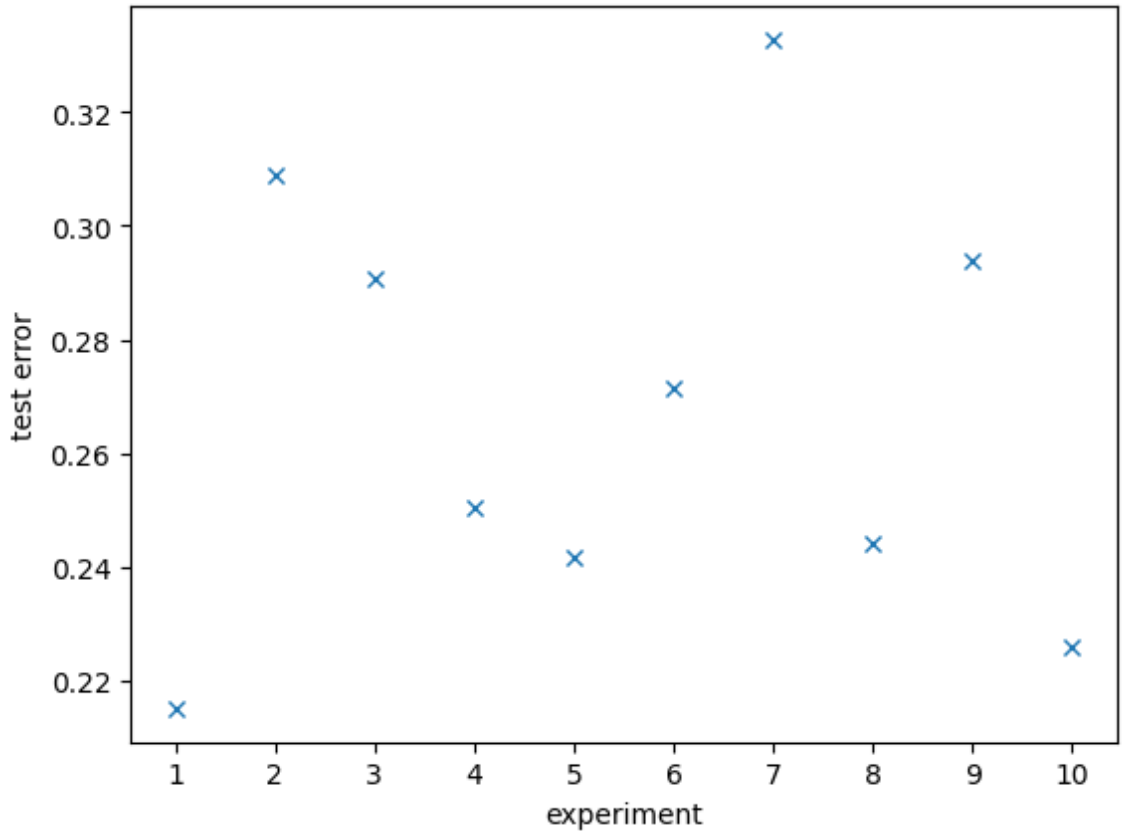
- **Validation set**: The optimal model m^* is given by

$$m^* = \underset{m}{\operatorname{arg}min} J_m$$

- **Training + Validation set:** Train the optimal model m^*
- **Test set**: determine test error on trained m^*


Training, validation and test inputs
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
train
validate
test





Optimal number of hidden neurons is 2 (Max number of times)
Average m.s.e.: 0.215 (by taking average of test error when hidden neurons = 2)

2. The CIFAR-10 dataset consists of 60,000 32x32 colour images from 10 classes, with 6,000 images per class:
https://www.cs.toronto.edu/~kriz/cifar.html

There are 50,000 training images and 10,000 test images. Read CIFAR-10 datasets from torchvision.datasets:

```
from torchvision.datasets import CIFAR10
```

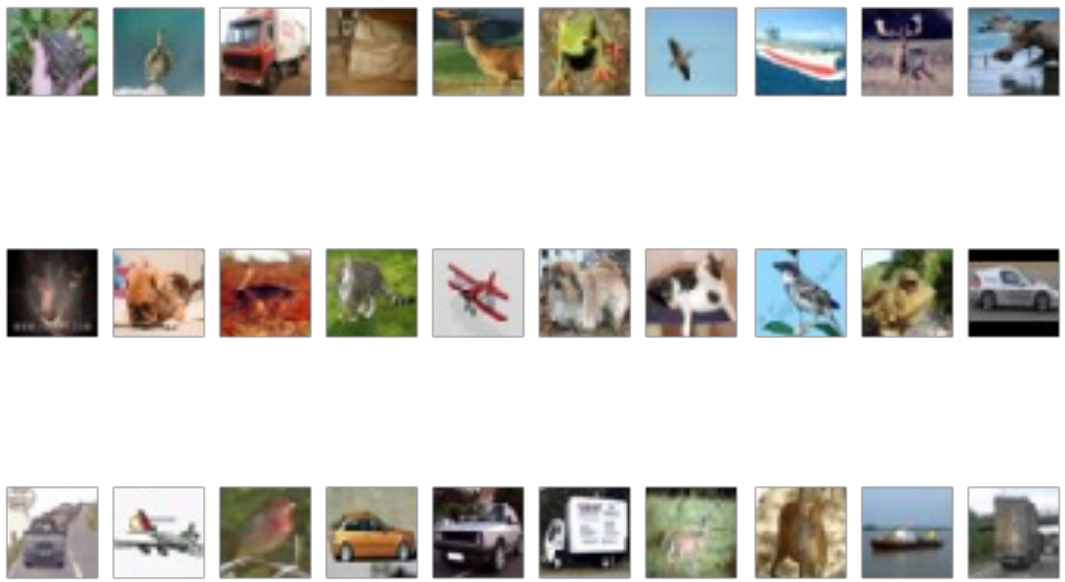
a) Build three DNN of different complexity with three hidden layers to classify the images:
   i. A **small network** with 50 neurons in every hidden layer
   ii. A **medium network** with 100 neurons in every hidden layer
   iii. A **large network** with 500 neurons in every layer

Plot cross-entropies against epochs for train and test datasets. Use the Adams optimizer with default parameter for training and early stopping criterion to terminate. Comment on the overfitting and underfitting of the networks if any.

The CIFAR-10 dataset: https://www.cs.toronto.edu/~kriz/cifar.html

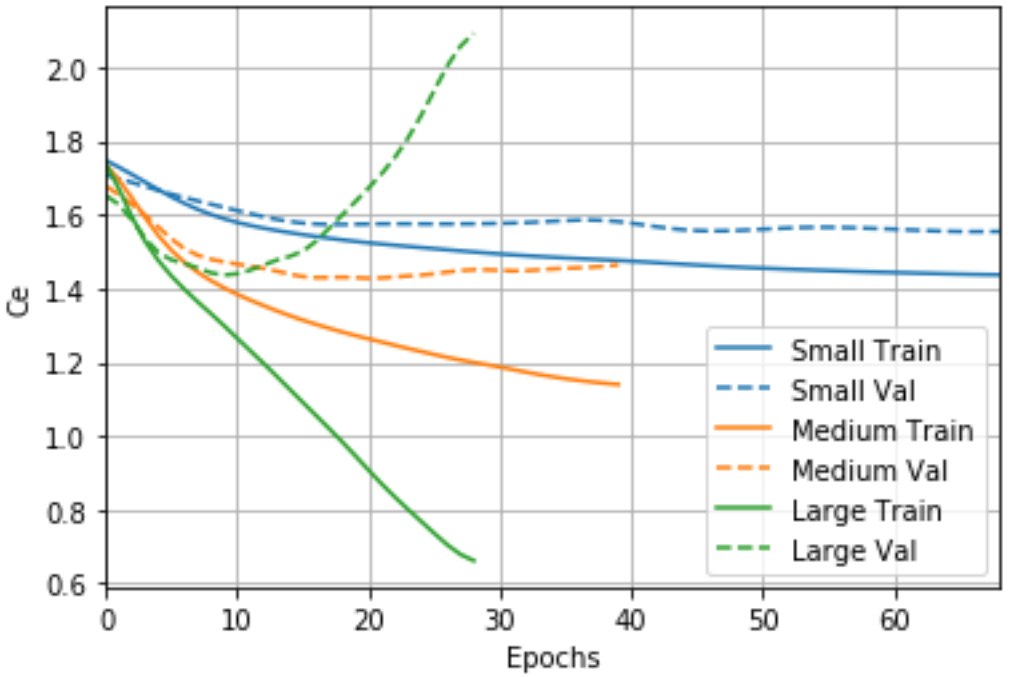
50000 training and 10000 test images; Images are 32x32 colour images

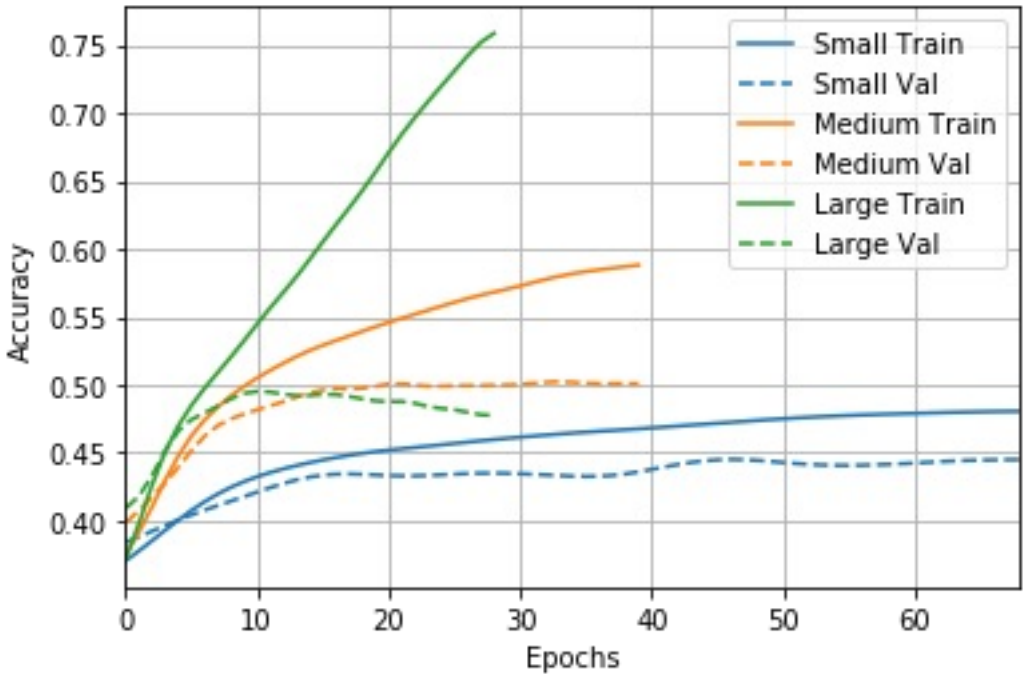
from torchvision.datasets **import** CIFAR10



```
class NeuralNetwork(nn.Module):
    def __init__(self, hidden_size=100):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(32*32*3, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, 10),
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

```
model = NeuralNetwork(hidden_size=50)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())
early_stopper = EarlyStopper(patience=10, min_delta=0)



for t in range(max_epochs):
    train_loss, train_correct = train_loop(train_dataloader, model, loss_fn, optimizer)
    test_loss, test_correct = test_loop(test_dataloader, model, loss_fn)

    if early_stopper.early_stop(test_loss):
        print("Done!")
        break
```





Small model is **underfitting**
Medium model seems just right
Large model is **overfitting**

b) Using the **large network**, demonstrate how the following methods applied to all the layers could overcome overfitting of the network:

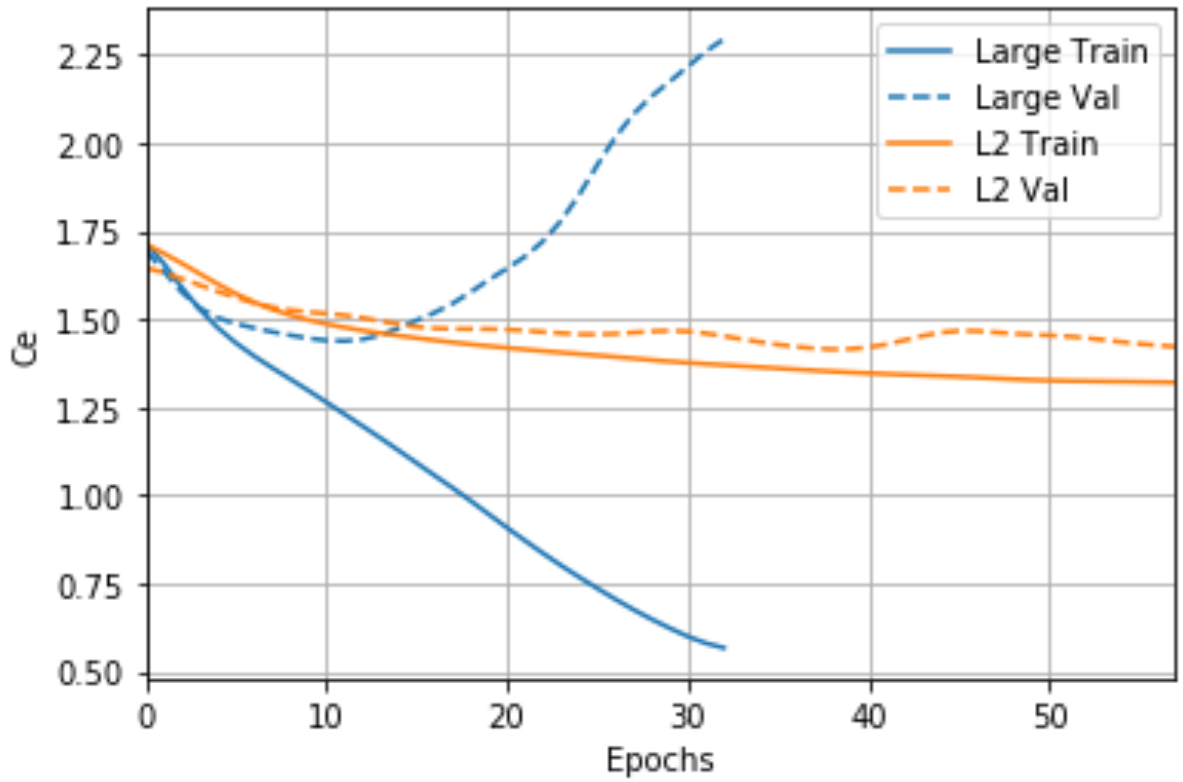
i. Weight regularization with L2 norm. Use weight decay parameter $\beta = 0.001$.

ii. Dropouts at a probability $p = 0.5$.

iii. Combining both weight regularization and dropouts from (i) and (ii), respectively.

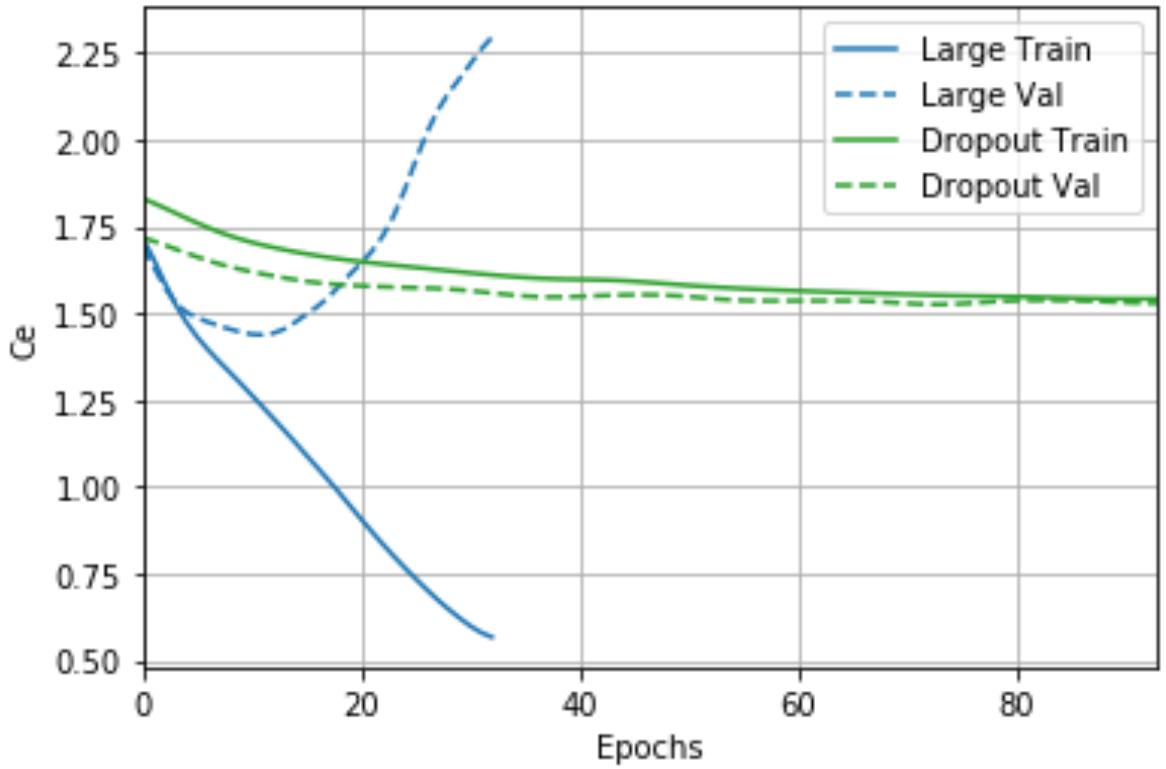
L2 weight regularization

optimizer = torch.optim.Adam(model.parameters(), weight_decay=0.001)

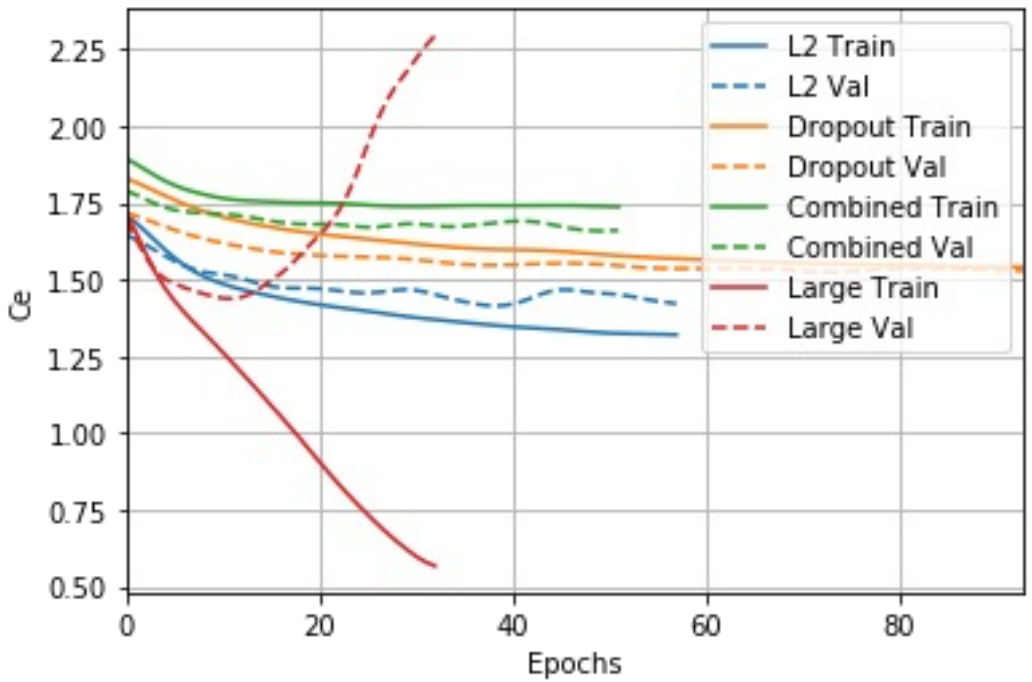


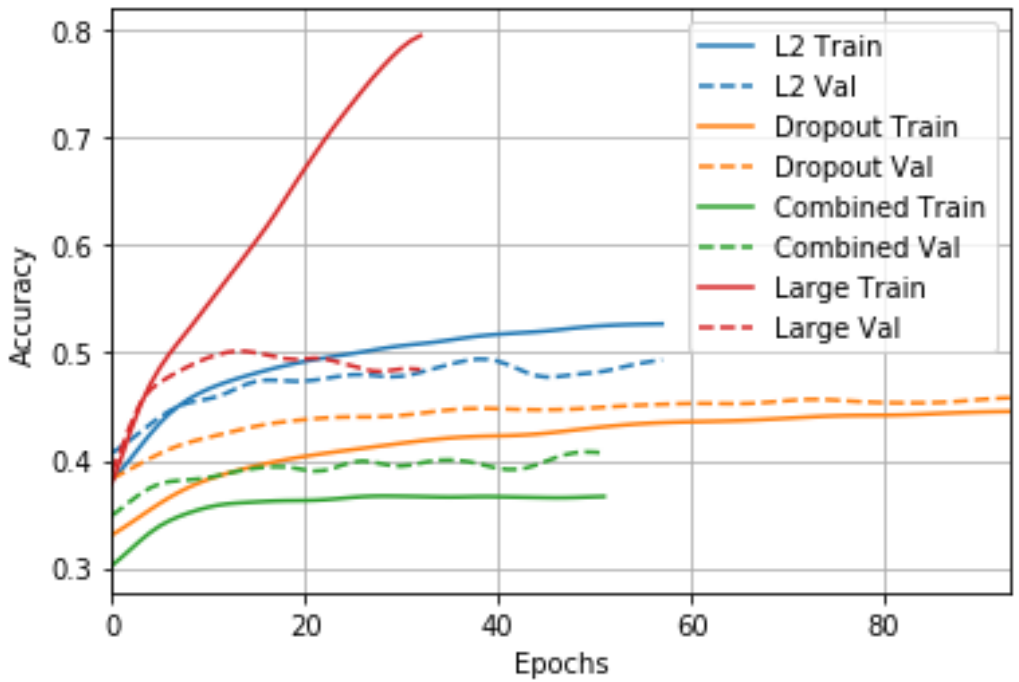
Dropouts

```
class NeuralNetwork_dropout(nn.Module):
    def __init__(self, hidden_size = 500, drop_out=0.5):
        super(NeuralNetwork_dropout, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(32*32*3, hidden_size),
            nn.ReLU(),
            nn.Dropout(p=drop_out),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Dropout(p=drop_out),
            nn.Linear(hidden_size, 10),
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```



Combined: L2 regularization + dropouts

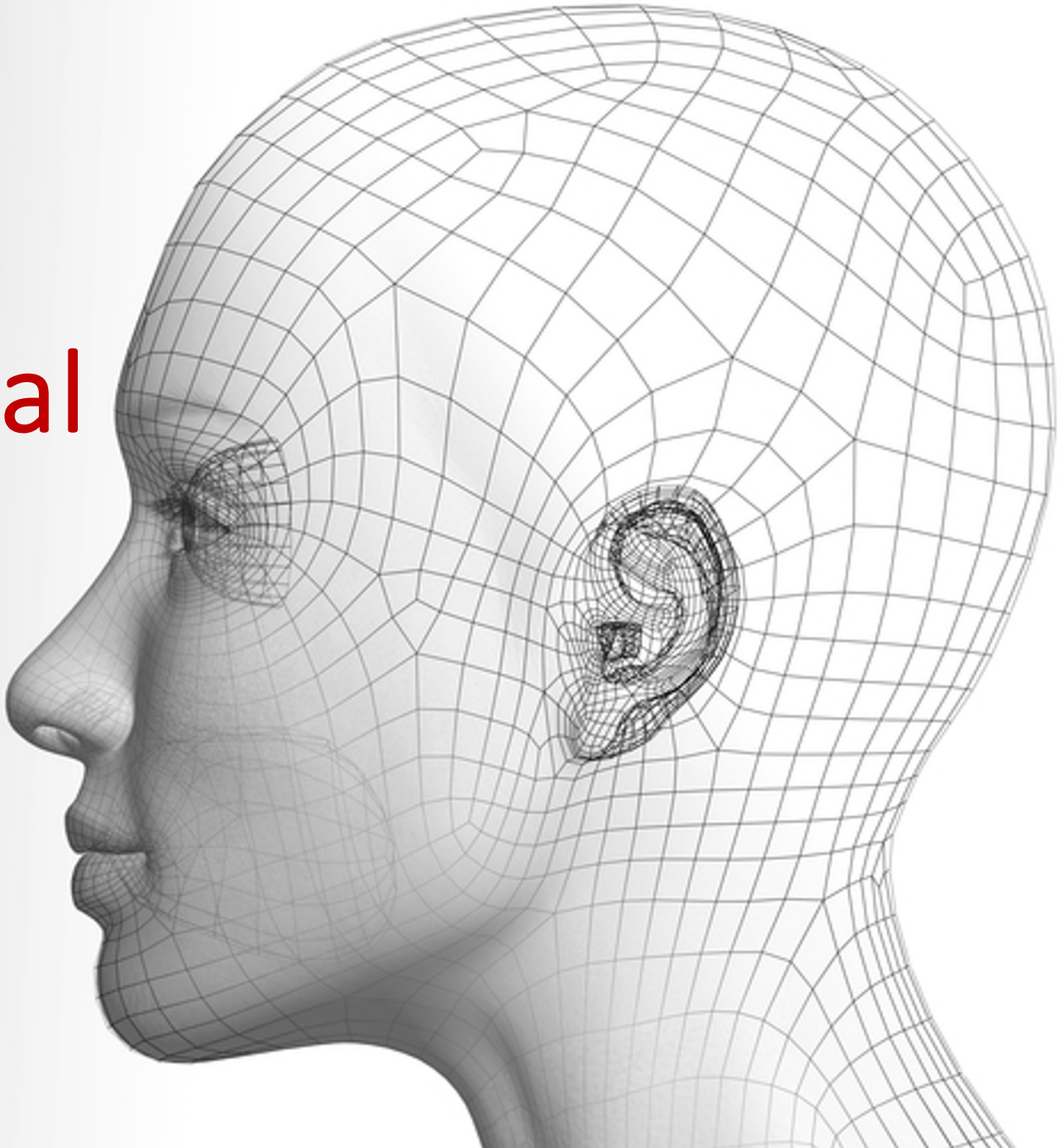




Tutorial 7 Convolutional Neural Networks II

Xingang Pan

潘新钢



https://xingangpan.github.io/

https://twitter.com/XingangP

Reminder: Feedback on Teaching

Dear Student,

The Online Faculty Teaching Evaluation for Semester 2 AY23-24 is now open.

Please click here to provide the feedback. The link is valid from 11-MAR-2024 to 24-MAR-2024.

We would like to strongly encourage you to participate in this exercise. Giving teaching feedback to the School will benefit the student population as we work towards refining and improving our teaching pedagogy and course materials. Together we can develop a more nurturing and effective learning environment for every student.

Thank you and with Best Regards,

SFT Administrator

Question 1

Figure Q1 depicts a block that consists of three convolutional layers. The input volume has a size of $256\times32\times32$ and the second layer has 32 convolution filters each with a size of $64\times3\times3$, stride = 1 and padding = 1.

Provide the values of n_1, d_1, F_1, n_2, d_2, and F_2 to form a valid block.
Explain your design

Answer is hidden so that you can try the question by yourself first

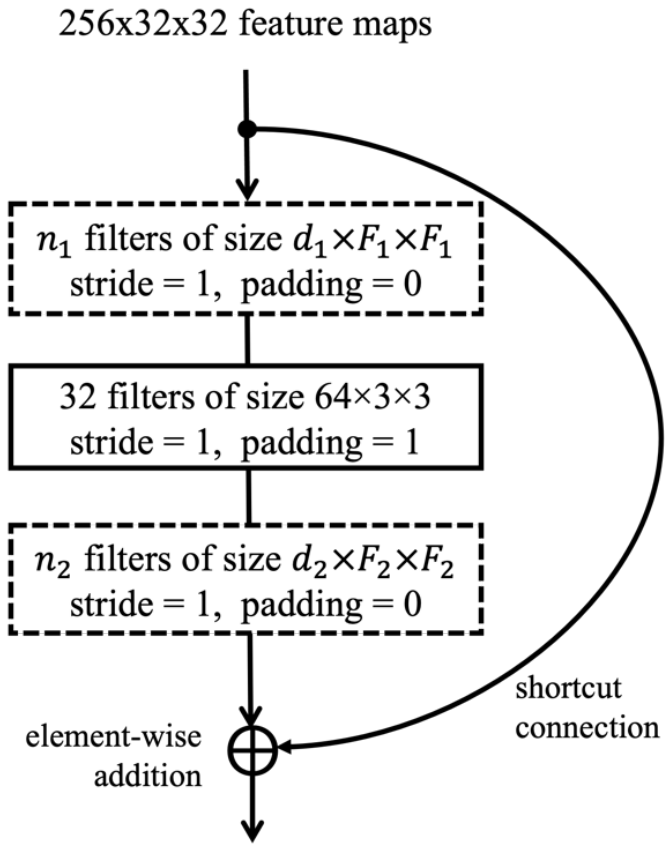


Figure Q1

Question 1

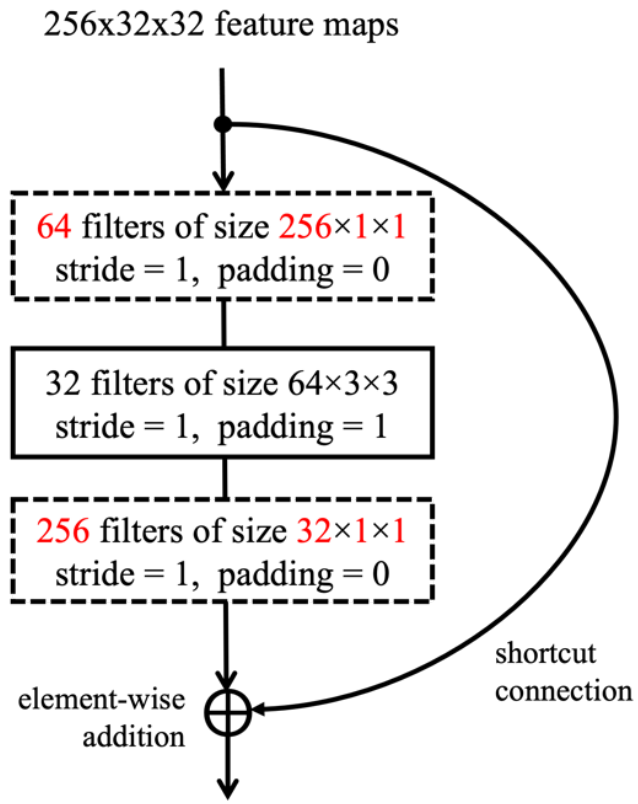
Answer:

$n_1 = 64, d_1 = 256, F_1 = 1$

$n_2 = 256, d_2 = 32, F_2 = 1$

To form a valid block, the size the of output volume at the residual branch has to be the same size as the input volume, which is $256 \times 32 \times 32$, such that element-wise addition can be performed, thus $n_2 = 256$ and 1×1 is chosen to keep the spatial resolution.

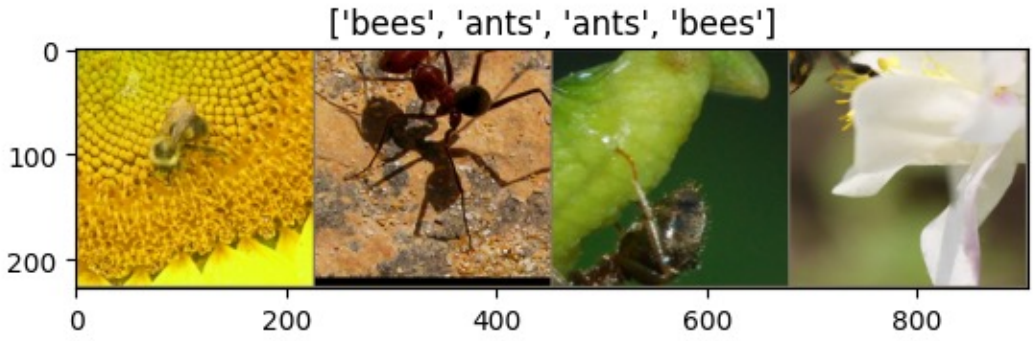
The values of d_1 and d_2 are chosen to match the depth of their corresponding input. n_1 is chosen to match the filter size of the second layer.



Question 2

Study and try the tutorial t7q2.ipynb on transfer learning. In particular,

- Understand how to data augmentation is performed
- Review the transfer learning steps
- Try the code to perform transfer learning on the classification of bees vs. ants

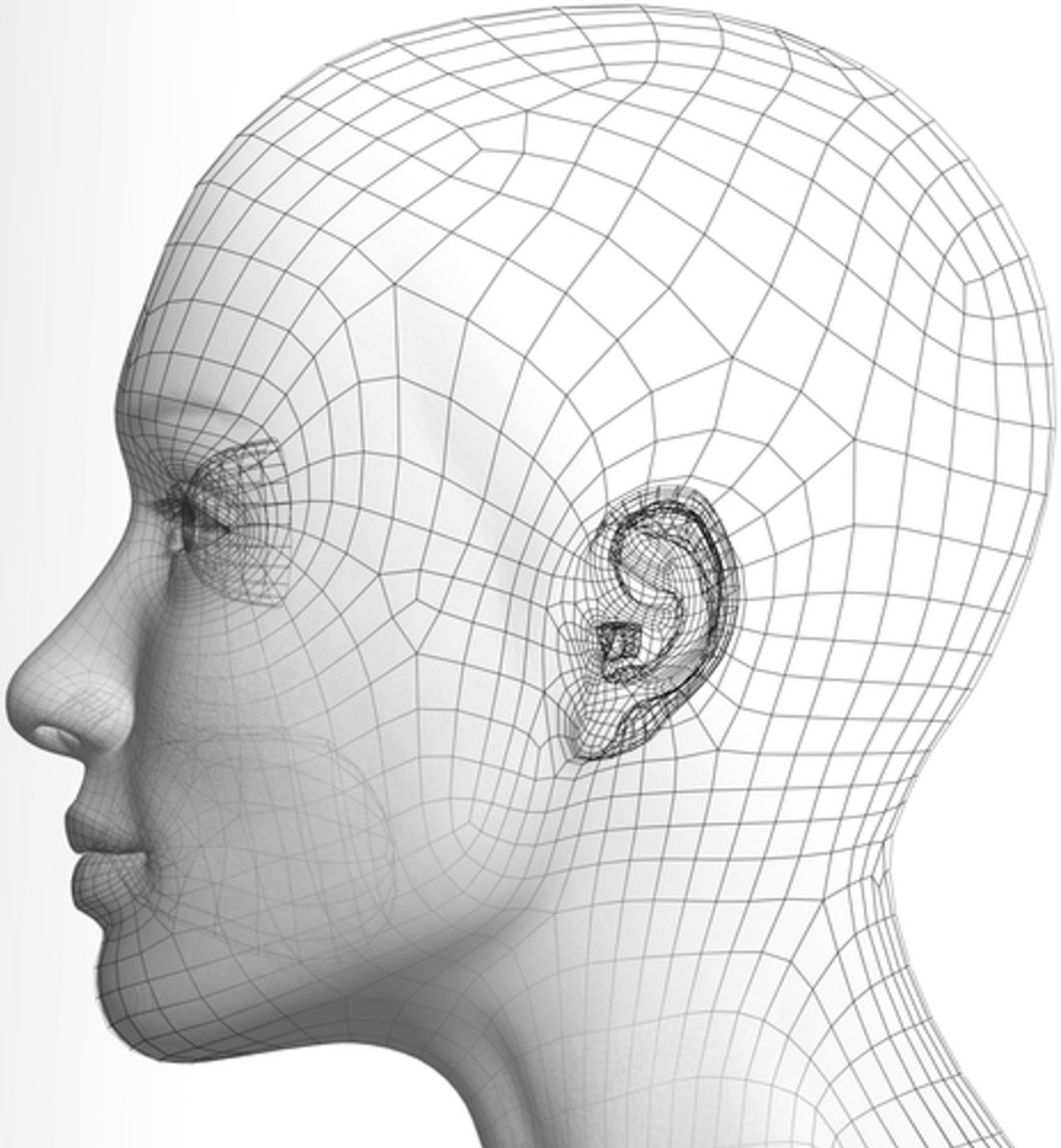


Check the code for more details

t7q2.ipynb

Tutorial 8
Recurrent Neural Networks

Xingang Pan
潘新钢



Question 1

1. A recurrent neural network (RNN) receives sequences of 3-dimensional inputs and has two hidden neurons and one output neuron. The weight matrix $\boldsymbol{U}$ connecting the input to the hidden layer, the weight matrix $\boldsymbol{V}$ connecting the hidden output to the output layer, the hidden layer bias $\boldsymbol{b}$ and the output layer bias $\boldsymbol{c}$ are given by

$$\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 \\ 0.5 & 0.1 \\ 0.2 & -2.0 \end{pmatrix}, \boldsymbol{V} = \begin{pmatrix} 2.0 \\ -1.5 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}, \text{ and } \boldsymbol{c} = 0.5.$$

Find the output sequence for an input sequence of $(\boldsymbol{x}(t))_{t=1}^{4}$ where

$$\boldsymbol{x}(1) = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}, \boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix}, \boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix}, \text{ and } \boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} \text{ if}$$

(a) The RNN is of hidden-recurrence (Elman) type with the recurrence weight matrix $\boldsymbol{W}$ connecting the previous hidden output to the current hidden layer input is given by

$$\boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 \\ 1.5 & 0.0 \end{pmatrix}.$$

Assume that the hidden activations are initialized to zero.

(b) The RNN is of top-down recurrence (Jordan) type with the weight matrix $\boldsymbol{W}$ connecting the previous output to the current state of hidden layer is given by

$$\boldsymbol{W} = (2.0 \quad 1.3).$$

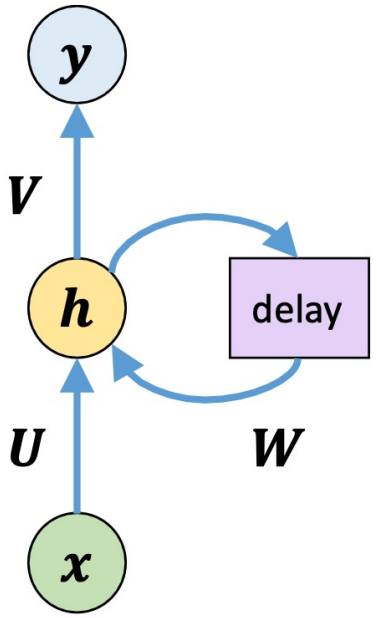
Assume that the output activations are initialized to zero.

t8q1.ipynb

Question 1

Hidden Recurrence:

$$\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 \\ 0.5 & 0.1 \\ 0.2 & -2.0 \end{pmatrix}, \boldsymbol{V} = \begin{pmatrix} 2.0 \\ -1.5 \end{pmatrix} \text{ and } \boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 \\ 1.5 & 0.0 \end{pmatrix}.$$



$$\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}, \text{ and } \boldsymbol{c} = 0.5$$

Three input neurons, two hidden neurons, and one output neuron.

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

$$\phi(u) = \tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$$
$$\sigma(u) = \text{sigmoid}(u) = \frac{1}{1 + e^{-u}}.$$

Assume $\boldsymbol{h}(0) = (0 \quad 0)^{T}$.

Question 1

At t=1, $\boldsymbol{x}(1) = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$:

$$\boldsymbol{h}(1) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(1) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(0) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} + \begin{pmatrix} 2.0 & 1.5 \\ 1.3 & 0.0 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}\right) = \begin{pmatrix} 0.0 \\ 0.99 \end{pmatrix}$$

$$\boldsymbol{y}(1) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(1) + \boldsymbol{c}\right) = \text{sigmoid}\left((2.0 \quad -1.5)\begin{pmatrix} 0.0 \\ 0.99 \end{pmatrix} + (0.5)\right) = (0.27)$$

Question 1

At t=2, $\boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix}$:

$$\boldsymbol{h}(2) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(2) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(1) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix} + \begin{pmatrix} 2.0 & 1.5 \\ 1.3 & 0.0 \end{pmatrix}\begin{pmatrix} 0.0 \\ 0.99 \end{pmatrix} + \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}\right) = \begin{pmatrix} 0.99 \\ 1.0 \end{pmatrix}$$

$$\boldsymbol{y}(2) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(2) + \boldsymbol{c}\right) = \text{sigmoid}\left((2.0 \quad -1.5)\begin{pmatrix} 0.99 \\ 1.0 \end{pmatrix} + (0.5)\right) = (0.73)$$

Similarly,

at t=3, $\boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix}$; $\boldsymbol{h}(3) = \begin{pmatrix} 1.0 \\ -0.21 \end{pmatrix}$ and $\boldsymbol{y}(3) = (0.94)$

at t=4, $\boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix}$; $\boldsymbol{h}(4) = \begin{pmatrix} -0.54 \\ 0.98 \end{pmatrix}$ and $\boldsymbol{y}(4) = (0.11)$

Question 1

The input sequence $\left(\boldsymbol{x}(1), \boldsymbol{x}(2), \boldsymbol{x}(3), \boldsymbol{x}(4)\right)$:

$$\left(\begin{pmatrix}1\\2\\-1\end{pmatrix} \quad \begin{pmatrix}-1\\1\\-2\end{pmatrix} \quad \begin{pmatrix}0\\3\\1\end{pmatrix} \quad \begin{pmatrix}2\\-1\\0\end{pmatrix}\right)$$

The output sequence $\left(\boldsymbol{y}(1), \boldsymbol{y}(2), \boldsymbol{y}(3), \boldsymbol{y}(4)\right)$:

$$((0.27) \quad (0.73) \quad (0.94) \quad (0.11))$$

Question 1

1. A recurrent neural network (RNN) receives sequences of 3-dimensional inputs and has two hidden neurons and one output neuron. The weight matrix $\boldsymbol{U}$ connecting the input to the hidden layer, the weight matrix $\boldsymbol{V}$ connecting the hidden output to the output layer, the hidden layer bias $\boldsymbol{b}$ and the output layer bias $\boldsymbol{c}$ are given by

$$\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 \\ 0.5 & 0.1 \\ 0.2 & -2.0 \end{pmatrix}, \boldsymbol{V} = \begin{pmatrix} 2.0 \\ -1.5 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}, \text{ and } \boldsymbol{c} = 0.5.$$

Find the output sequence for an input sequence of $(\boldsymbol{x}(t))_{t=1}^{4}$ where

$$\boldsymbol{x}(1) = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}, \boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix}, \boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix}, \text{ and } \boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} \text{ if}$$

(a) The RNN is of hidden-recurrence (Elman) type with the recurrence weight matrix $\boldsymbol{W}$ connecting the previous hidden output to the current hidden layer input is given by

$$\boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 \\ 1.5 & 0.0 \end{pmatrix}.$$

Assume that the hidden activations are initialized to zero.

(b) The RNN is of top-down recurrence (Jordan) type with the weight matrix $\boldsymbol{W}$ connecting the previous output to the current state of hidden layer is given by

$$\boldsymbol{W} = (2.0 \quad 1.3).$$

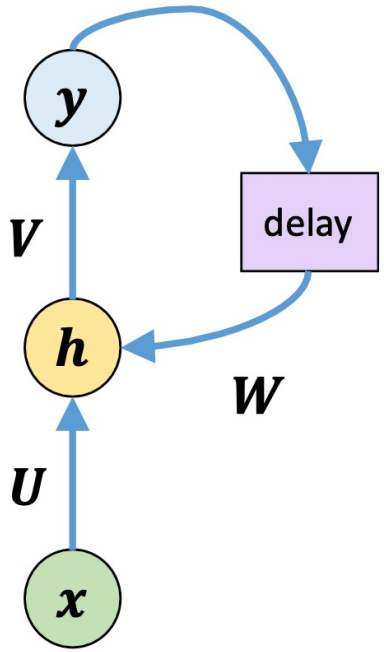
Assume that the output activations are initialized to zero.

t8q1.ipynb

Question 1

Top-down Recurrence:

$$\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 \\ 0.5 & 0.1 \\ 0.2 & -2.0 \end{pmatrix}, \boldsymbol{V} = \begin{pmatrix} 2.0 \\ -1.5 \end{pmatrix}, \boldsymbol{W} = (2.0 \quad 1.3)$$



$$\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}, \text{ and } \boldsymbol{c} = 0.5$$

Three input neurons, two hidden neurons, and one output neuron.

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^T\boldsymbol{x}(t) + \boldsymbol{W}^T\boldsymbol{y}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^T\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

Initially,

$$\boldsymbol{Y}(0) = (0.0)$$

Question 1

At t=1, $\boldsymbol{x}(1) = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$:

$$\boldsymbol{h}(1) = \phi\left(\boldsymbol{U^T x}(1) + \boldsymbol{W^T} y(0) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} + \begin{pmatrix} 2.0 \\ 1.3 \end{pmatrix}(0) + \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}\right) = \begin{pmatrix} 0.0 \\ 0.99 \end{pmatrix}$$

$$\boldsymbol{y}(1) = \sigma\left(\boldsymbol{V^T h}(1) + \boldsymbol{c}\right) = \text{sigmoid}\left((2.0 \quad -1.5 \quad 0.2)\begin{pmatrix} 0.0 \\ 0.99 \end{pmatrix} + (0.5)\right) = (0.27)$$

Question 1

At t=2, $\boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix}$:

$$\boldsymbol{h}(2) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(2) + \boldsymbol{W}^{\boldsymbol{T}}y(1) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix} + \begin{pmatrix} 2.0 \\ 1.3 \end{pmatrix}(0.27) + \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}\right) = \begin{pmatrix} 0.95 \\ 1.0 \end{pmatrix}$$

$$\boldsymbol{y}(2) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(2) + \boldsymbol{c}\right) = \text{sigmoid}\left((2.0 \quad -1.5 \quad 0.2)\begin{pmatrix} 0.95 \\ 1.0 \end{pmatrix} + (0.5)\right) = (0.71)$$

Similarly,

at t=3, $\boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix}$; $\boldsymbol{h}(3) = \begin{pmatrix} 1.0 \\ -0.52 \end{pmatrix}$ and $\boldsymbol{y}(3) = (0.96)$

at t=4, $\boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix}$; $\boldsymbol{h}(4) = \begin{pmatrix} -0.36 \\ 0.98 \end{pmatrix}$ and $\boldsymbol{y}(4) = (0.16)$

Question 1

The input sequence $\left(\boldsymbol{x}(1), \boldsymbol{x}(2), \boldsymbol{x}(3), \boldsymbol{x}(4)\right)$:

$$\left(\begin{pmatrix}1\\2\\-1\end{pmatrix} \quad \begin{pmatrix}-1\\1\\-2\end{pmatrix} \quad \begin{pmatrix}0\\3\\1\end{pmatrix} \quad \begin{pmatrix}2\\-1\\0\end{pmatrix}\right)$$

The output sequence $\left(\boldsymbol{y}(1), \boldsymbol{y}(2), \boldsymbol{y}(3), \boldsymbol{y}(4)\right)$:

$$((0.27) \quad (0.71) \quad (0.96) \quad (0.16))$$

Question 2

2. An RNN receives 3-dimensional input sequence and produces 2-dimensional output sequence. It has 5 hidden neurons and receives sequences of 100 time steps.

   - Generate 8 training input sequences by drawing values uniformly between 0 and 1.0.
   - If the sequence $\left(\boldsymbol{y}(t)\right)_{t=1}^{100}$ where $\boldsymbol{y}(t) = \left(y_k(t)\right)_{k=1}^{2} \in \boldsymbol{R}^2$ denotes the output sequence for an input sequence $\left(\boldsymbol{x}(t)\right)_{t=1}^{100}$ where $\boldsymbol{x}(t) = \left(x_i(t)\right)_{i=1}^{3} \in \boldsymbol{R}^3$, generate the corresponding output sequences for the input training sequences as follows:

$$y_1(t) = 5x_1(t) - 0.2x_3(t-1) + 0.1\varepsilon$$
$$y_2(t) = 25x_2(t-1)x_3(t-3) + 0.1\varepsilon$$

   where ε is standard normally distributed random variable.

Train an RNN to learn the above sequences by using gradient descent learning. Use a learning factor $\alpha = 0.01$ and Adam optimizer.

t8q2.ipynb

Question 2

Set random seed and initialize hyper-parameters

```
[2]  # Set random seed for reproducibility
     seed = 10
     np.random.seed(seed)
     torch.manual_seed(seed)
```

```
<torch._C.Generator at 0x7eeddd747dd0>
```

```
[3]  # Initialize hyper-parameters
     n_in = 3
     n_hidden = 5
     n_out = 2
     n_steps = 100
     n_seqs = 8
     n_iters = 10000
     lr = 0.01
```

Generate training data

```
[4]  x_train = np.random.rand(n_seqs, n_steps, n_in)
     y_train = np.zeros([n_seqs, n_steps, n_out])

     y_train[:, 1:, 0] = 5 * x_train[:, 1:, 0] - 0.2 * x_train[:, :-1, 2]
     y_train[:, 3:, 1] = 25 * x_train[:, 2:-1, 1] * x_train[:, :-3, 2]
     y_train += 0.1 * np.random.randn(n_seqs, n_steps, n_out)

     x_train = torch.tensor(x_train, dtype=torch.float32)
     y_train = torch.tensor(y_train, dtype=torch.float32)
```

$$\left.\begin{aligned} y_1(t) &= 5x_1(t) - 0.2x_3(t-1) + 0.1\varepsilon \\ y_2(t) &= 25x_2(t-1)x_3(t-3) + 0.1\varepsilon \end{aligned}\right\}$$

Question 2

The basic slice syntax is i:j:k where *i* is the starting index, *j* is the stopping index, and *k* is the step.

y_train[:,3:,1] = [$y_2(3), y_2(4), y_2(5) \ \ldots, y_2(99)$]

x_train[:,2:-1,1] = [$x_2(2), x_2(3), x_2(4) \ \ldots, x_2(98)$]

x_train[:, :-3,2] = [$x_3(0), x_3(1), x_3(2) \ \ldots, x_3(96)$]

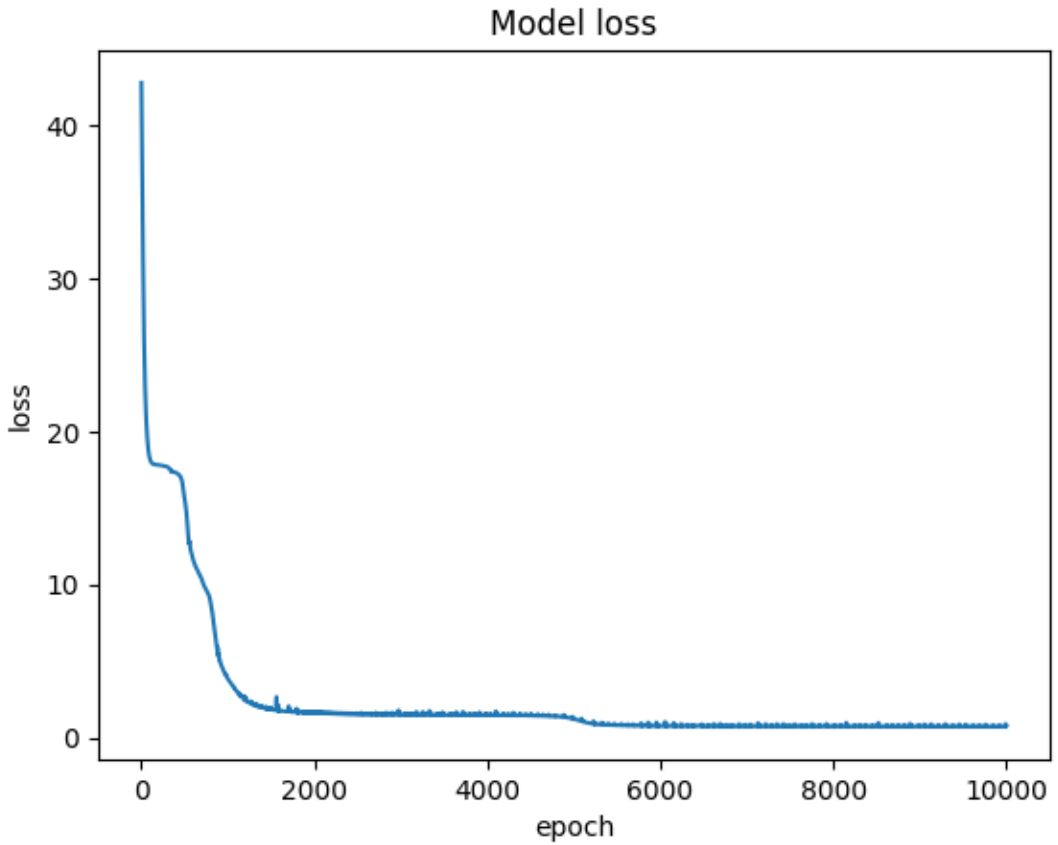
Assume *n* is the number of elements in the dimension being sliced

If *j* is not given it defaults to *n*

If *k* is not given it defaults to 1

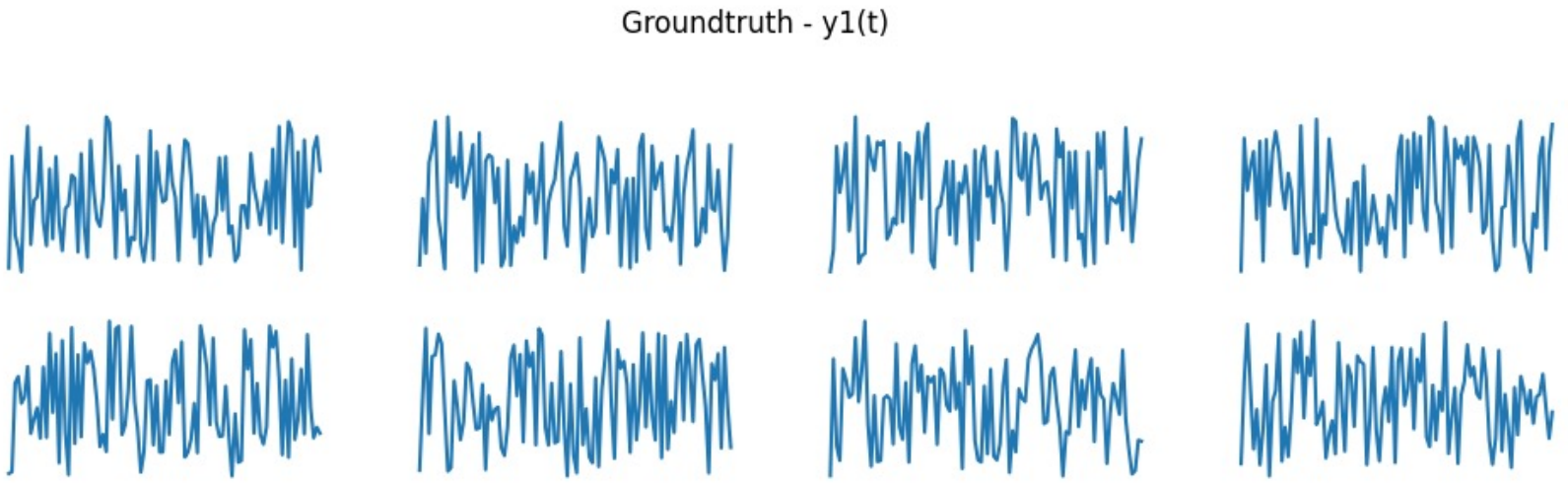
Negative *i* and *j* are interpreted as *n* + *i* and *n* + *j* where *n* is the number of elements in the corresponding dimension

Question 2



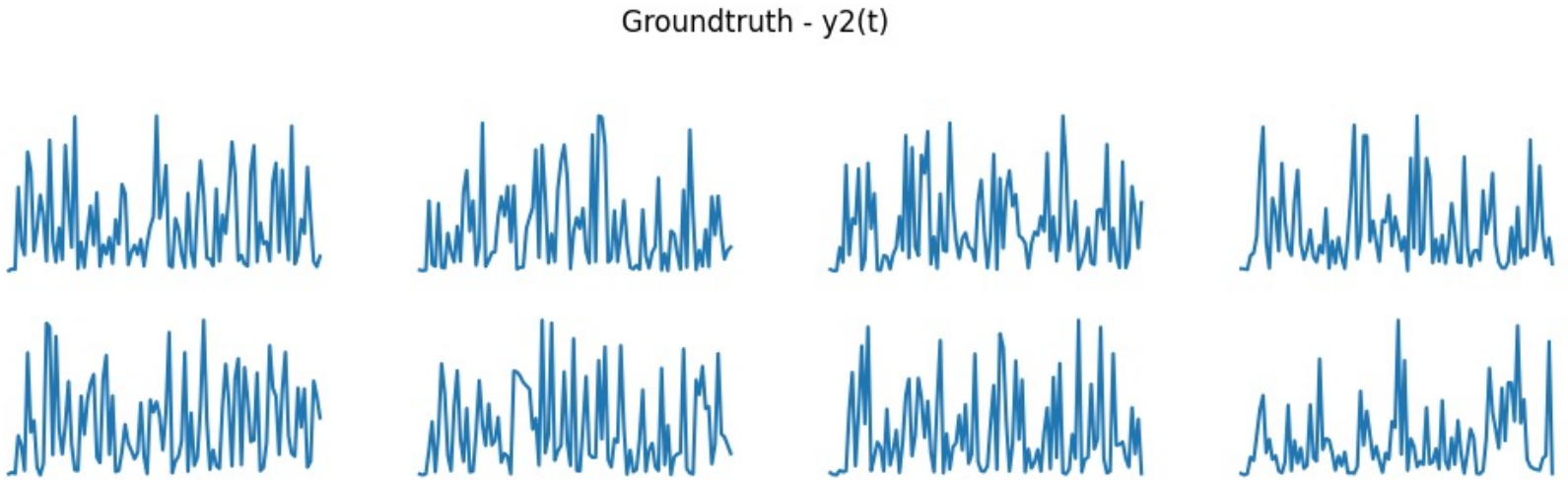
Question 2

$$y_1(t) = 5x_1(t) - 0.2x_3(t-1) + 0.1\varepsilon$$

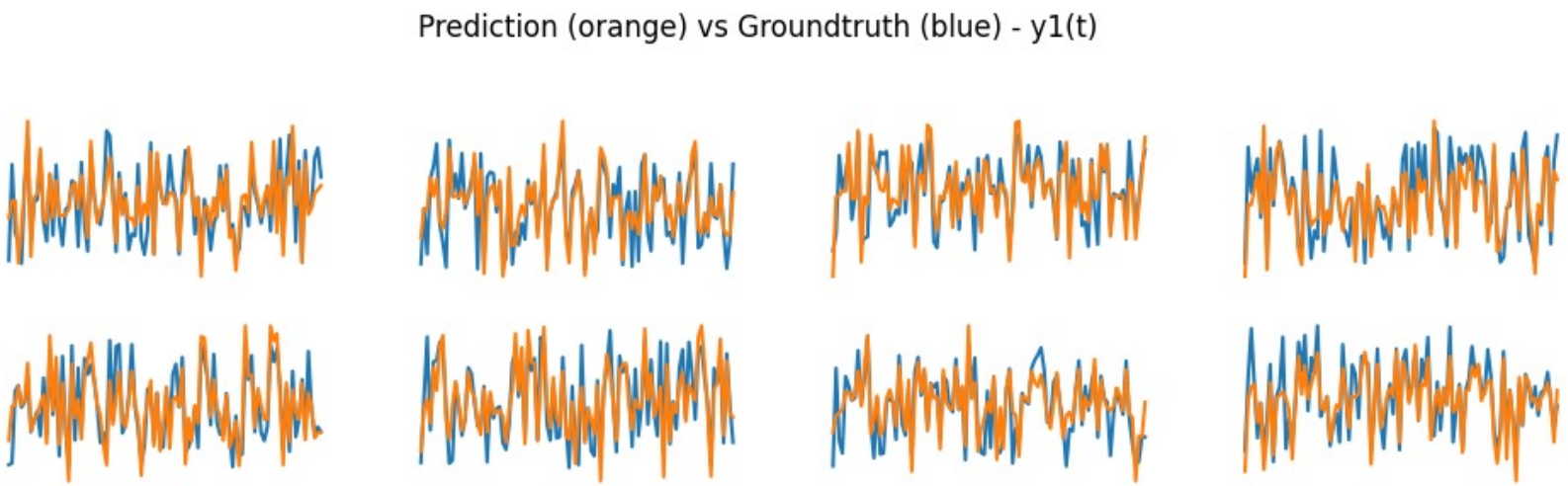


Question 2

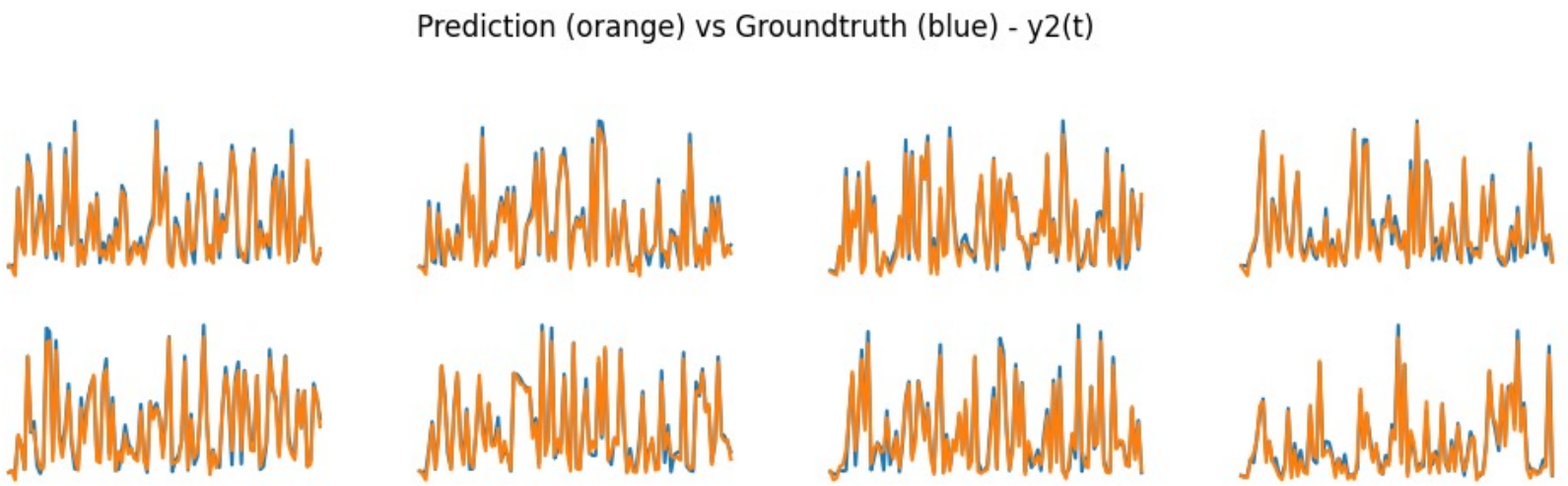
$$y_2(t) = 25x_2(t-1)x_3(t-3) + 0.1\varepsilon$$



Question 2



Question 2



Question 3

3. Design an LSTM layer with 10 units to map the following input and output sequences:

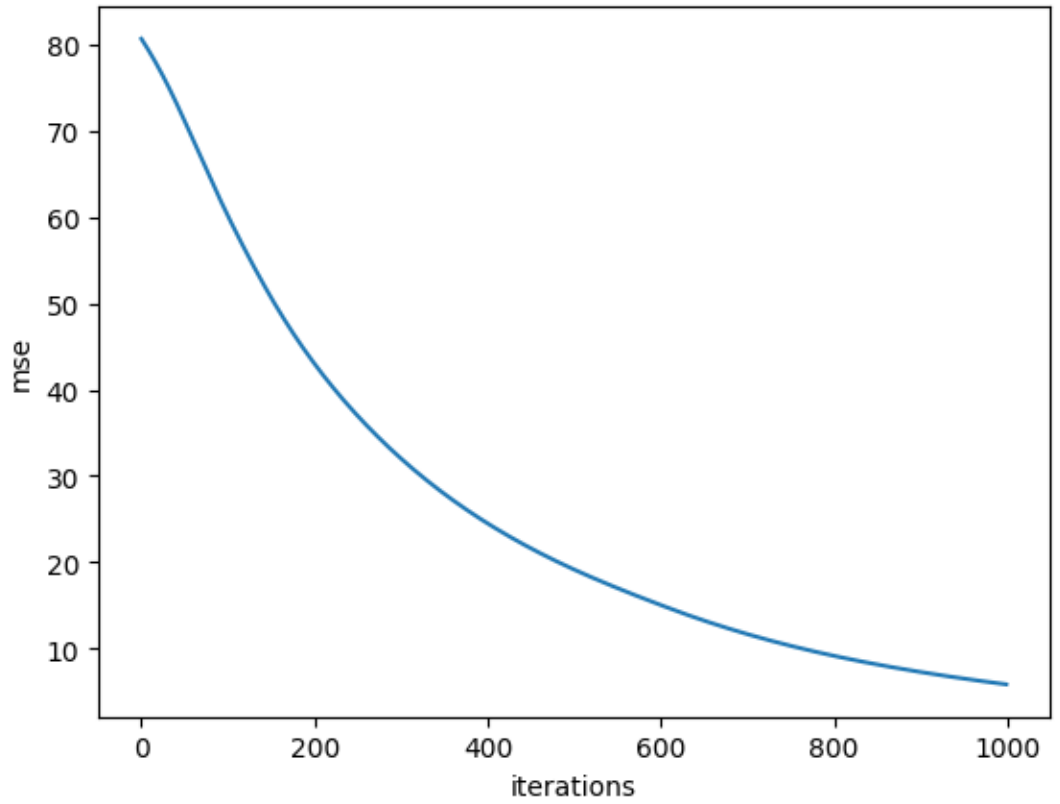
Input $\boldsymbol{x}$	Output $\boldsymbol{y}$
(1 2 5 6)	(1 3 7 11)
(5 7 7 8)	(5 12 14 15)
(3 4 5 7)	(3 7 9 12)

Plot the learning curves at a rate $\alpha = 0.001$.

Find the output sequences for the following input sequences:

(1 2 3 4)
(4 5 6 7)

Question 3



Question 3

Find the output sequences for the following input sequences:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \end{pmatrix}$$

```
test_x = torch.tensor(
    [[[1], [2], [3], [4]],  # 1, 3, 5, 7
    [[4], [5], [6], [7]]], dtype=torch.float32)  # 4, 9, 11, 13
out = predictor(test_x)
print(out)
```

```
tensor([[0.9988, 2.7404, 6.3064, 8.8883], [4.6289,
8.4632, 9.4166, 9.5583]], grad_fn=<SqueezeBackward1>)
```

Question 4

4. Design a character RNN layer with 10 GRU units to learn the sentiments about a movie, given in table 1.

Convert the input text into character ids and set the maximum text length to be 40. Train the network using gradient descent learning with the Adam optimizer and at a learning rate $\alpha = 0.001$. Use dropouts at the hidden layer of GRU at a probability $p = 0.7$.

Plot the cost and the accuracies against epochs during training.

Question 4

Table 1

Input	*Sentiment*
I did not like the movie	negative
The movie was not good	negative
I watched the movie with great interest	positive
I have seen better movies	negative
Good to see that movie	positive
I am not a fan of movies	negative
I liked the movie great	positive
The movie was of interest to me	positive
I thought they could show interesting scenes	negative
The movie did not have good scenes	negative
Family did not like the movie at all	negative

After training, find the likelihoods of the sentiments of the following statements:

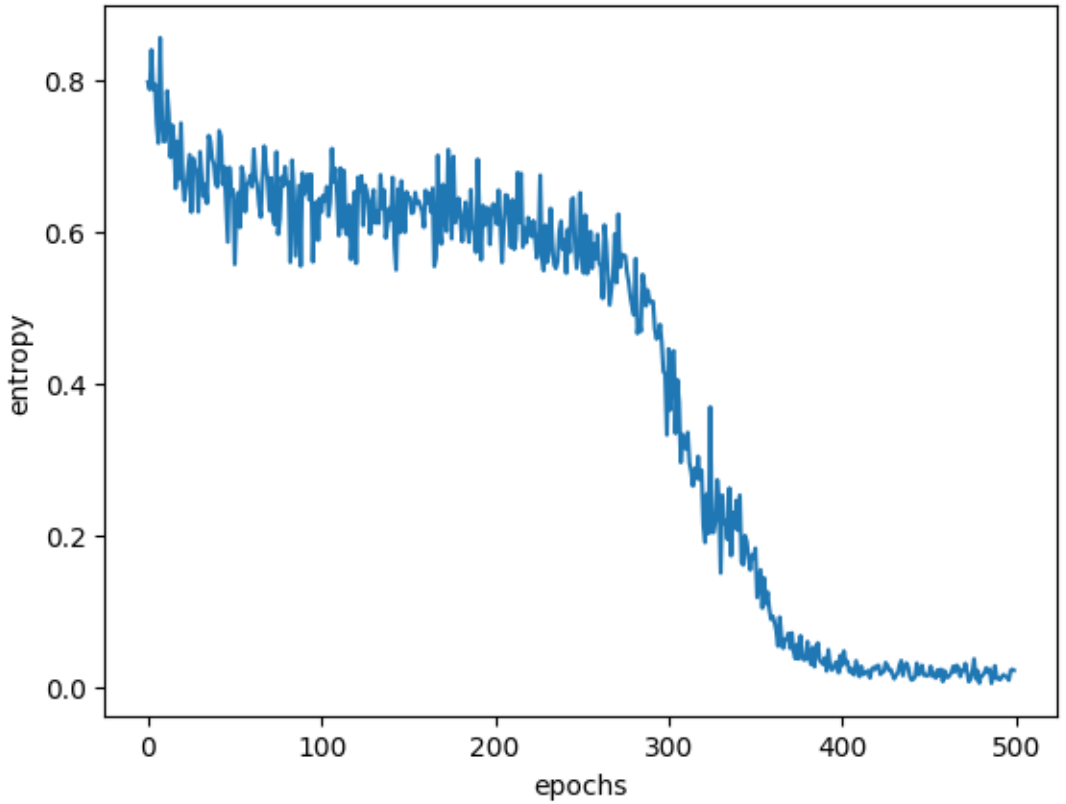
The movie was not interesting to me

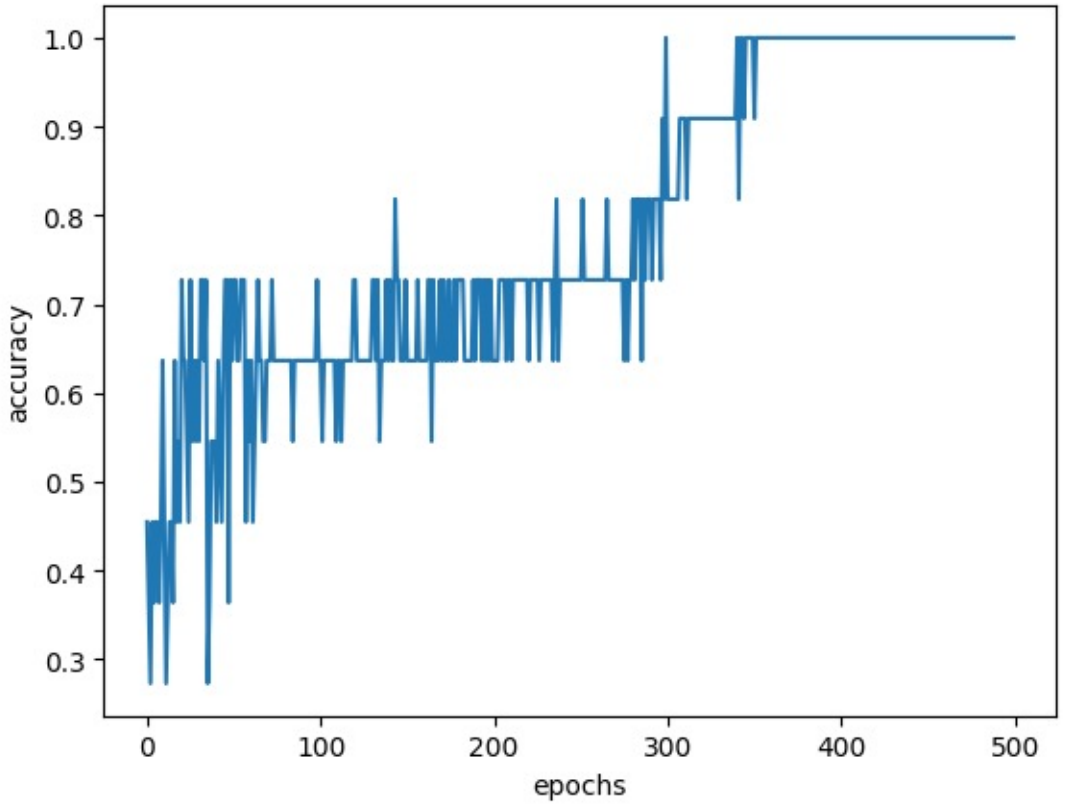
I liked the movie with great interest

Question 4

- Input: ‘Good to see that movie’
- Translated input as a list of ids:
  [7 14 14 4 0 17 14 0 16 5 5 0 17 8 1 17 0 12 14 19 9 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0]
- Targets:
  [0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0]
- Input text -> preprocess to max length -> convert to IDs

Question 4


entropy
0.0
0.2
0.4
0.6
0.8
0
100
200
300
400
500
epochs


accuracy
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
100
200
300
400
500
epochs

Question 4

```
test_data = ['The movie was not interesting to me',
'I liked the movie with great interest']
```

```
test_logits = model(x_test, drop_rate=0)
probs = nn.functional.softmax(test_logits, dim=1)
print(probs)
```

```
tensor(
[[0.0119, 0.9881],
[0.4359, 0.5641]])
```

‘The movie was not interesting to me’ is a negative sentiment with a probability of 0.9881

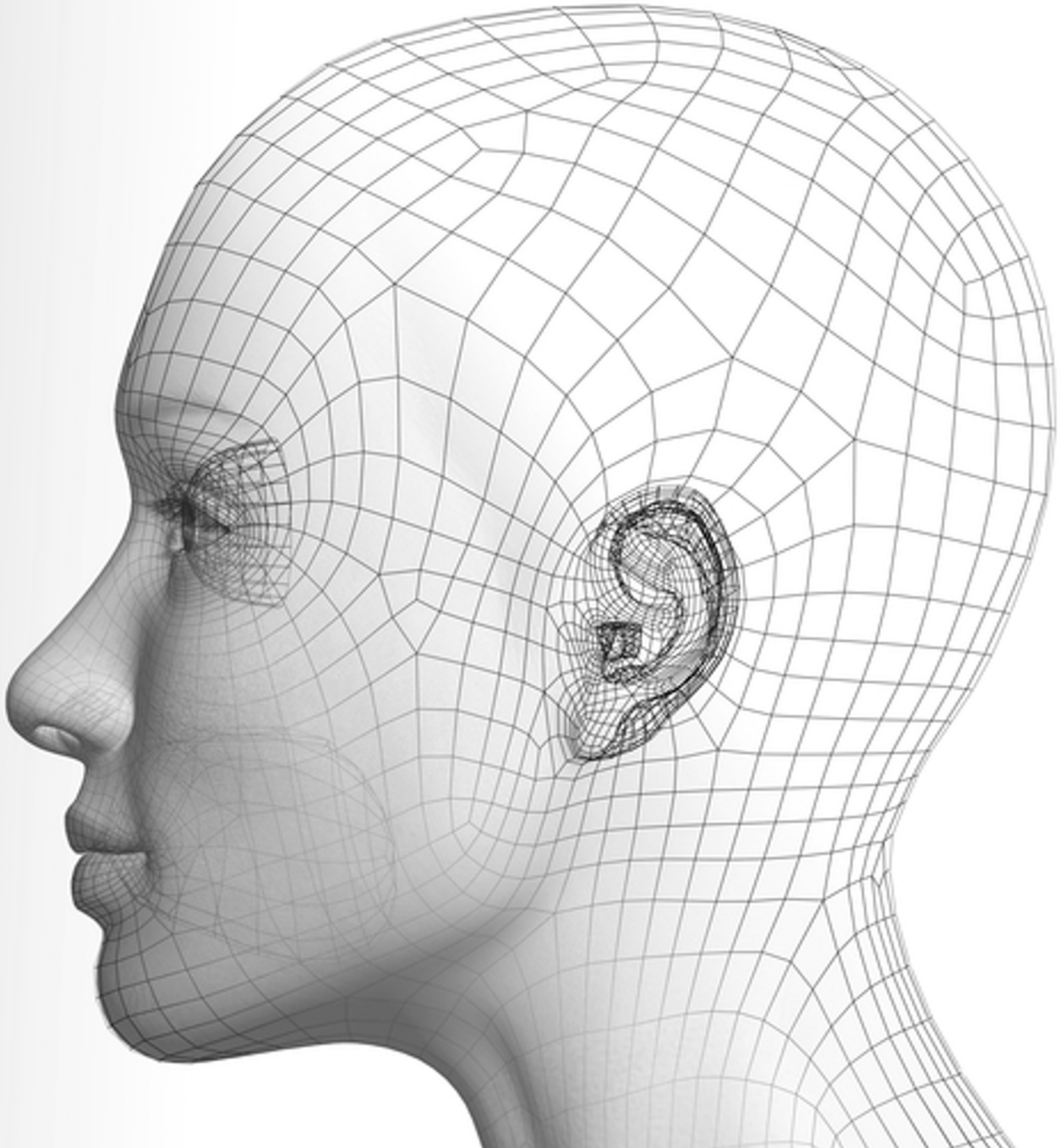
‘I liked the movie with great interest’ is a positive sentiment with a probability of 0.4359 (hasn’t trained well)

Tutorial 9
Attention

Xingang Pan
潘新钢

https://xingangpan.github.io/

https://twitter.com/XingangP



Question 1

The tutorial provides a simple walkthrough of the Vision Transformer. We hope you will be able to understand how it works by looking at the actual data flow during inference.

t9q1.ipynb

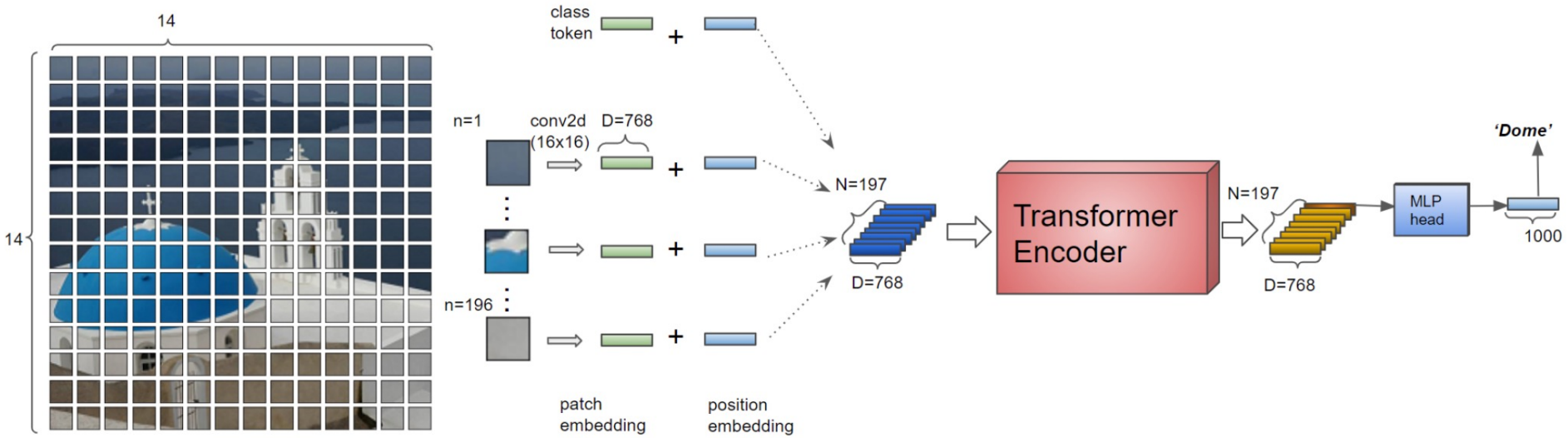


Figure 1. Vision Transformer inference pipeline.

1. Split Image into Patches
   The input image is split into 14 x 14 vectors with dimension of 768 by Conv2d (k=16x16) with stride=(16, 16).
2. Add Position Embeddings
   Learnable position embedding vectors are added to the patch embedding vectors and fed to the transformer encoder.
3. Transformer Encoder
   The embedding vectors are encoded by the transformer encoder. The dimension of input and output vectors are the same. Details of the encoder are depicted in Fig. 2.
4. MLP (Classification) Head
   The 0th output from the encoder is fed to the MLP head for classification to output the final classification results.

Question 1

t9q1.ipynb