Registration Form

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

MSNM
PPM-015-08-16112017

AI and Deep Learning from First Principles

About the Talk

This talk presents Artificial Intelligence and Deep Learning from first principles, focusing on core mathematical foundations, intuitive learning mechanisms, and their relevance to numerical methods and computational modelling. The session aims to demystify AI beyond black-box usage and highlight its role in research, teaching, and engineering applications.

Target Audience

Academicians · Researchers · Postgraduate Students · Final-Year Undergraduate Students

Date : 6 February 2026
Time : 9:00 a.m. – 12:00 p.m.
Venue : Seminar Room M46
Faculty of Civil Engineering, UTM

Speaker :
Ir Dr Airil Yasreen Mohd Yassin
Associate Professor
School of Energy, Geoscience, Infrastructure and Society (EGIS)
Heriot-Watt University Malaysia

OWLEDGE SHARING

ganised by:

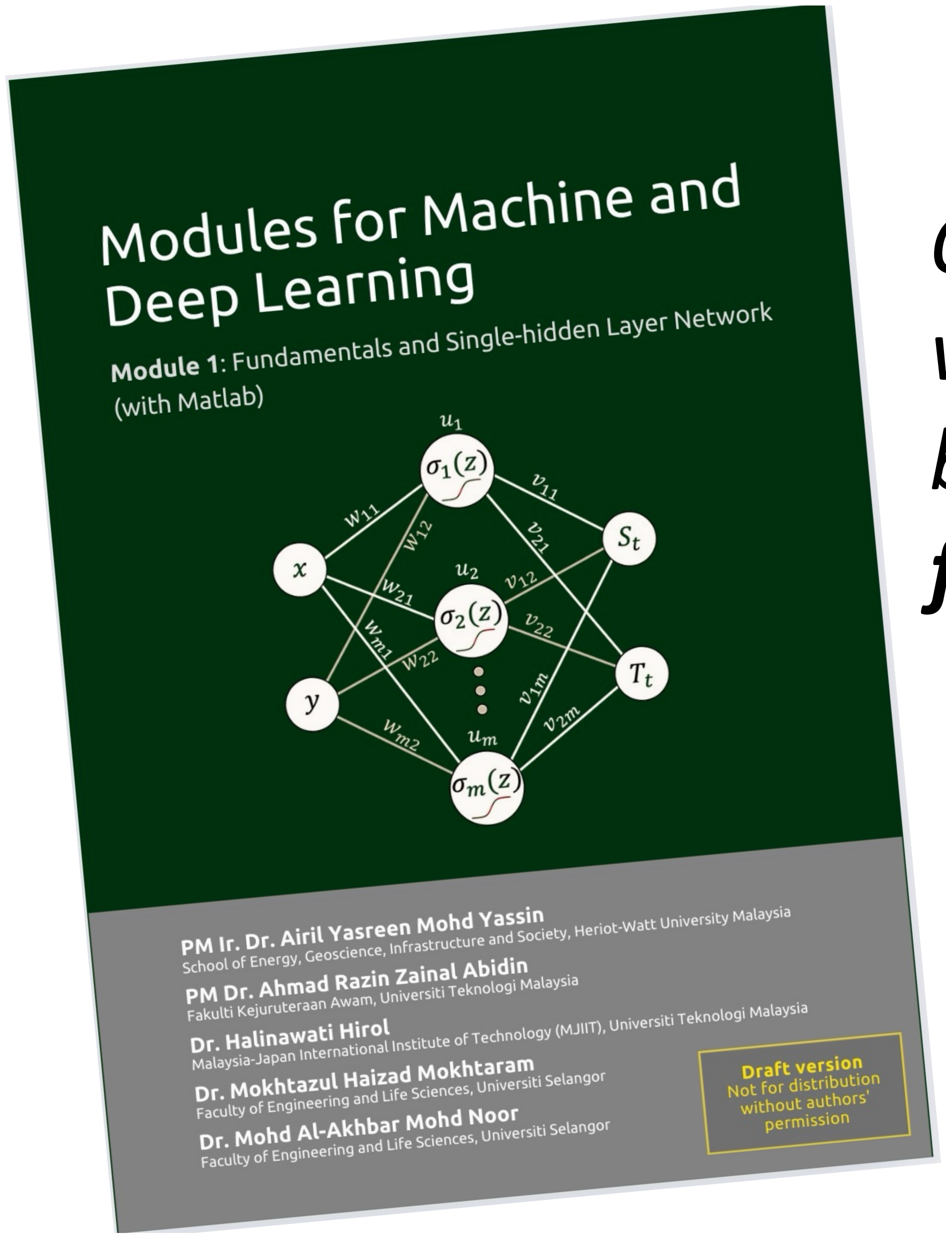
HERIOT WATT UNIVERSITY

UK | DUBAI | MALAYSIA

Objectives of the talk:

1. To introduce the first principles of AI Neural-Network/Deep-Learning (how and why it works) from engineers' point of view

2. To familiarize the audiences (e.g. engineering students and educators) with terms and ideas of deep learning so that they can do self-study on the topic/field (through readings and by watching videos available on YouTube)



Modules for Machine and Deep Learning

Module 1: Fundamentals and Single-hidden Layer Network (with Matlab)

PM Ir. Dr. Airil Yasreen Mohd Yassin
School of Energy, Geoscience, Infrastructure and Society, Heriot-Watt University Malaysia

PM Dr. Ahmad Razin Zainal Abidin
Fakulti Kejuruteraan Awam, Universiti Teknologi Malaysia

Dr. Halinawati Hirol
Malaysia-Japan International Institute of Technology (MJIIT), Universiti Teknologi Malaysia

Dr. Mokhtazul Haizad Mokhtaram
Faculty of Engineering and Life Sciences, Universiti Selangor

Dr. Mohd Al-Akhbar Mohd Noor
Faculty of Engineering and Life Sciences, Universiti Selangor

Draft version
Not for distribution without authors' permission

HERIOT WATT UNIVERSITY

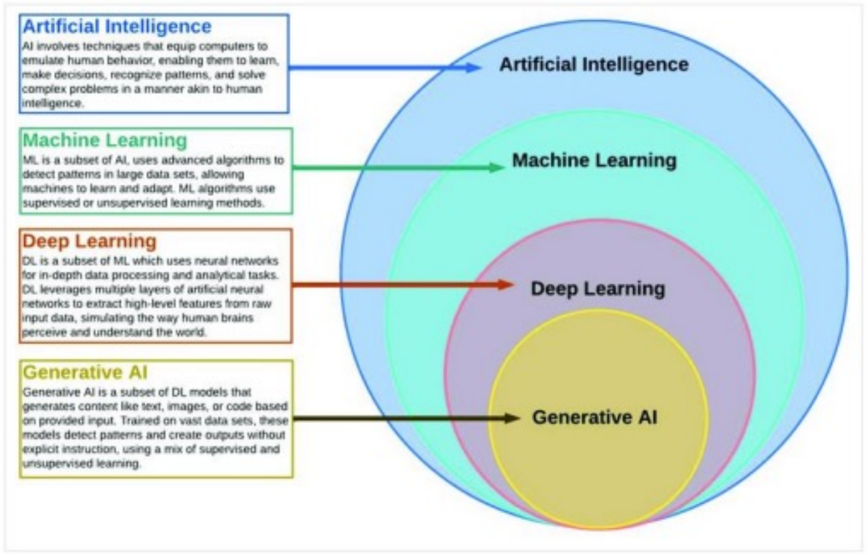
UK | DUBAI | MALAYSIA

Our module which will be shared ***for free!***

https://github.com/msnm-official/ai_modules

A very quick introduction

Machine Learning (ML) is a subset of Artificial Intelligence (AI). In turn, Deep Learning (DL) is a subset of ML. And today's very hot topics of Generative AI and Large Language Model (LLM) are subsets of DL. This is as shown in Figure 1 [1].



Artificial Intelligence
AI involves techniques that equip computers to emulate human behavior, enabling them to learn, make decisions, recognize patterns, and solve complex problems in a manner akin to human intelligence.

Machine Learning
ML is a subset of AI, uses advanced algorithms to detect patterns in large data sets, allowing machines to learn and adapt. ML algorithms use supervised or unsupervised learning methods.

Deep Learning
DL is a subset of ML which uses neural networks for in-depth data processing and analytical tasks. DL leverages multiple layers of artificial neural networks to extract high-level features from raw input data, simulating the way human brains perceive and understand the world.

Generative AI
Generative AI is a subset of DL models that generates content like text, images, or code based on provided input. Trained on vast data sets, these models detect patterns and create outputs without explicit instruction, using a mix of supervised and unsupervised learning.

Figure 1: Specialization of deep learning (DL) in the field of AI [1]

DL is a type of ML that employs Artificial Neural-Network (ANN, or just NN) as its architecture.

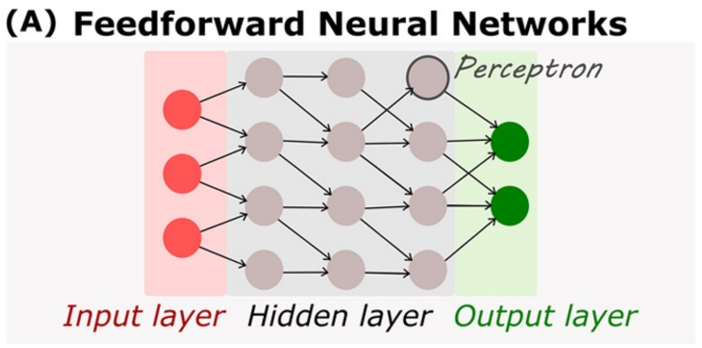
Current AI technologies that use NN and DL

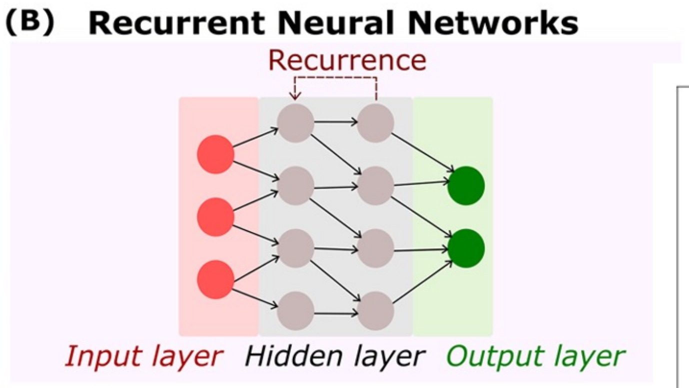
LLM
- ChatGPT
- Gemini
- Grok
- Claude
- DeepSeek

Non-LLM
- YOLO
- Google vision
- FSD (Full self driving)
- Alpha-GO
- Many more

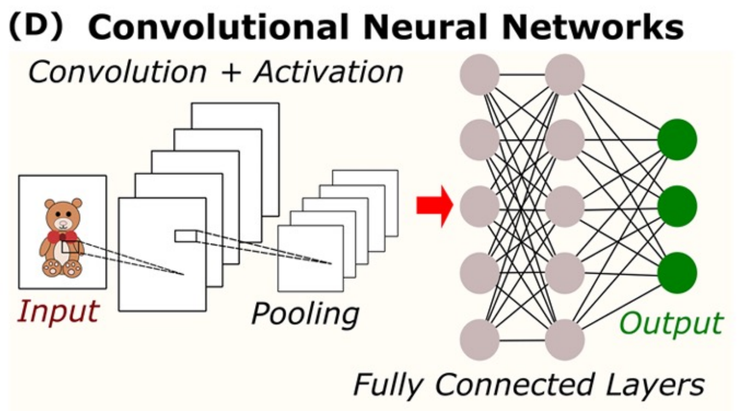
Variants of DL architecture:

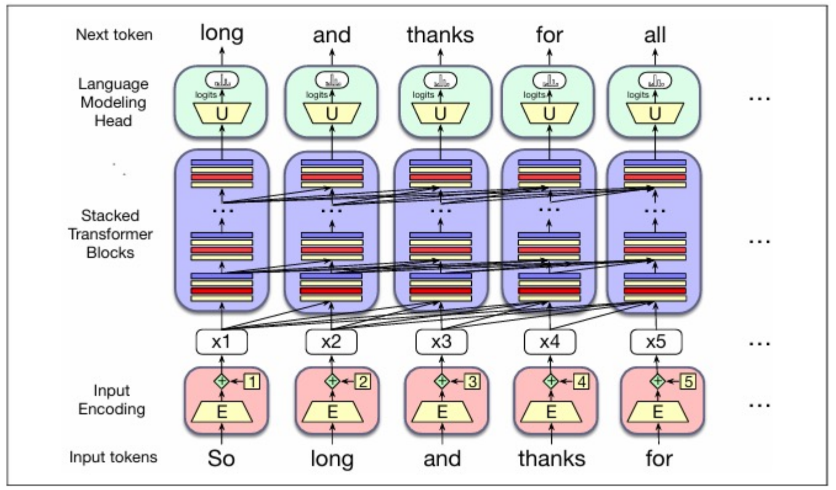
1. Feedforward Neural Networks (FFNN)
2. Recurrent Neural Networks (RNN)
3. Convolutional Neural Networks (CNN)
4. Transformer with Attention mechanism





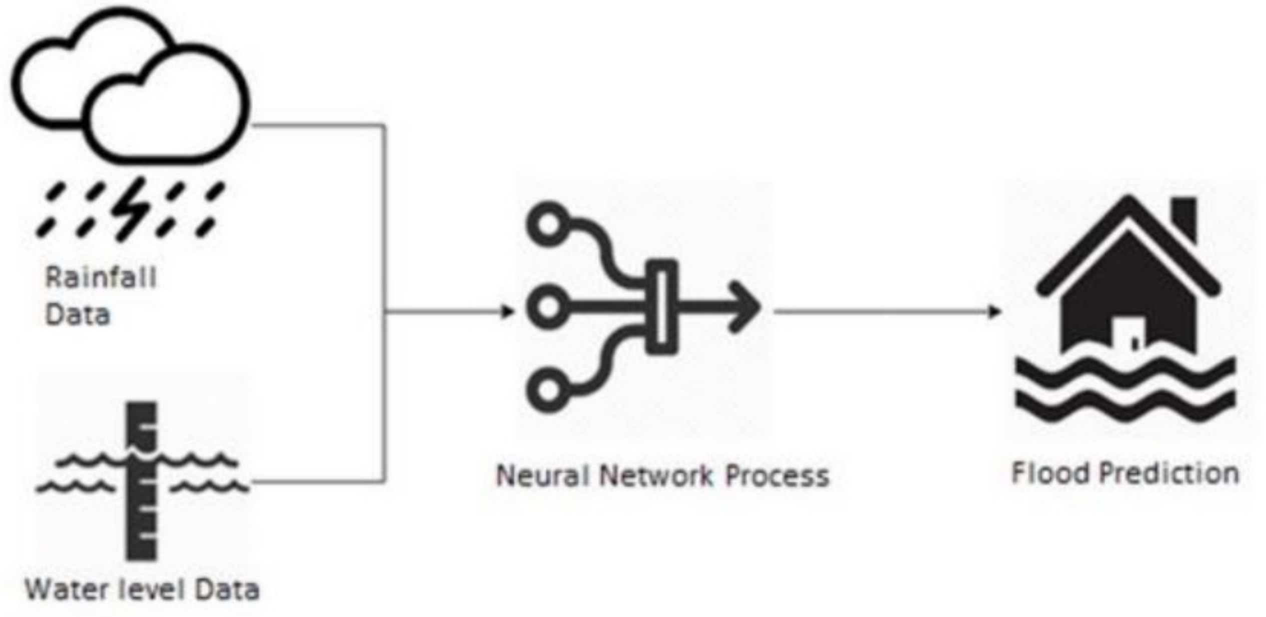
Ref: [4]





Transformer

Examples of application of deep learning in Civil Engineering



ANN flood prediction system [2]

How are we going to conduct the workshop?

Answer:

We are going to jump straight to the action and discuss the details along the way.

Therefore, in this workshop, a simultaneous equation is immediately given, and our task is to formulate a neural network (and deep neural network, if time permits) that able to produce “similar” results. As we progress with our derivation we will introduce and discuss specific features when it prompts explanation.

The “unknown” equations to be mimic

In this workshop, we will create a network that is “equivalent” to the following equations:

$$3x + 2y = s \qquad (1a)$$

$$2x + 6y = t \qquad (1b)$$

or in matrix forms,

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} s \\ t \end{Bmatrix} \qquad (1c)$$

But why we want to have these equations?

Answer:

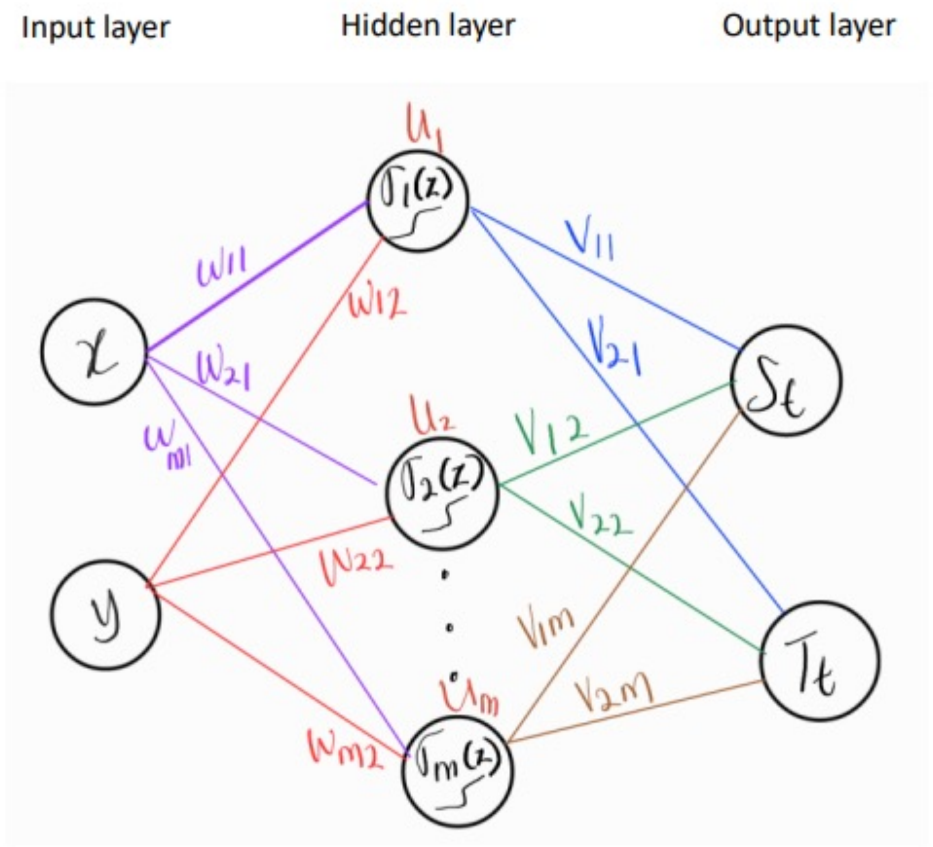
to create the sets of “labelled data”, that is, the set of paired input (e.g. *s*, *t*)

for the learning (e.g., training and testing) process of our network.

$$\begin{tikzcd} \underline{\text{Input}} & & \underline{\text{Actual Output}} \\ \begin{Bmatrix} x_1, y_1 \\ x_2, y_2 \\ \vdots \\ x_n, y_n \end{Bmatrix} \arrow[rr, "\text{Eq (1)}"] & & \begin{Bmatrix} s_1, t_1 \\ s_2, t_2 \\ \vdots \\ s_n, t_n \end{Bmatrix} \end{tikzcd}$$

Single hidden layer (shallow) neural network

To "mimic" Eqns. (1), we set a network below



A single hidden layer (to "mimic" Eqns. (1))

Neurons

Except for circle circling input x and y, the circles symbolize (or called) neurons.

Input

$$\{\bar{x}\} = \begin{Bmatrix} x_k \\ y_k \end{Bmatrix} = \begin{Bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{Bmatrix} \qquad (2^*)$$

Channel

The lines connecting the circles (neurons) are called the channel. They guide the flow or the travel of "information" from one neuron to another.

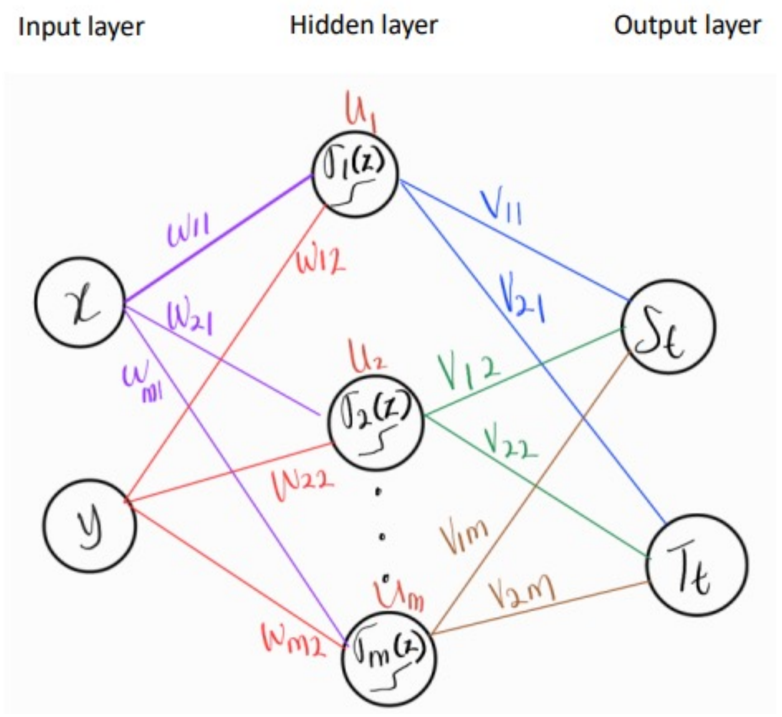
Weights, w_{ji} and v_{ji}

Coming from

$$[w] = \begin{bmatrix} w_{11} & w_{21} & \dots & w_{m1} \\ w_{12} & w_{22} & \dots & w_{m2} \end{bmatrix} \quad \begin{matrix} \leftarrow x \\ \leftarrow y \end{matrix} \qquad (2)$$

Going to

$$[v] = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1m} \\ v_{21} & v_{22} & \dots & v_{2m} \end{bmatrix} \quad \begin{matrix} \rightarrow \hat{s} \\ \rightarrow \hat{t} \end{matrix} \qquad (3)$$

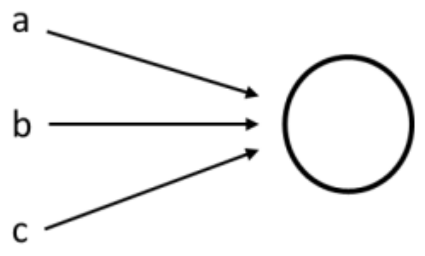


Biases, u_j

$$[u] = [u_1 \quad u_2 \quad \cdots \quad u_m] \tag{4}$$

Summing variable, z

$$z = \mathrm{a} + \mathrm{b} + \mathrm{c} \tag{5}$$



Input into neuron

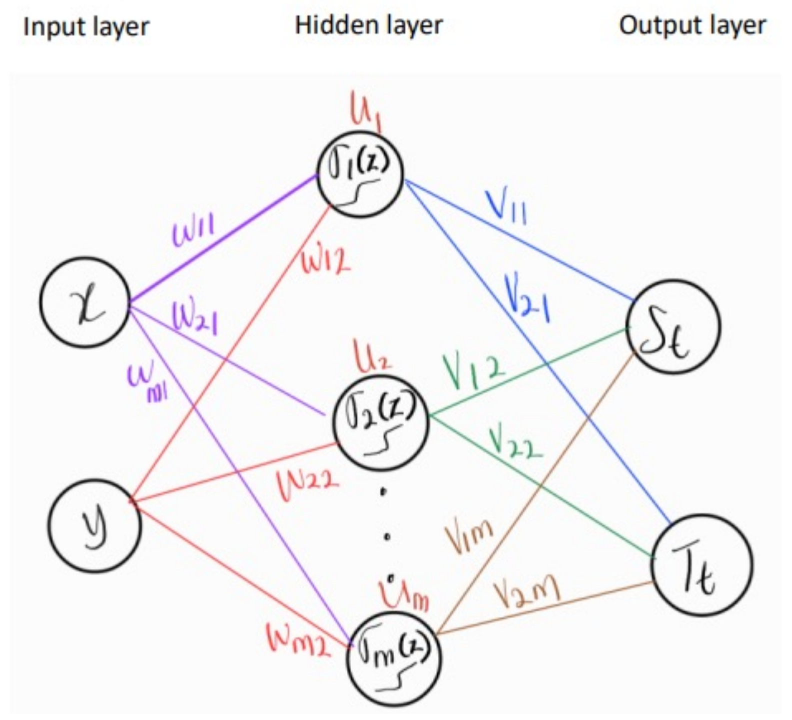
The activation functions, σ_j

$$\sigma_j = \frac{1}{1 + e^{-z_j}} \tag{6}$$

$$[\sigma] = [\sigma_1 \quad \sigma_2 \quad \cdots \quad \sigma_m] \tag{7a}$$

or

$$[\sigma] = \left[\frac{1}{1 + e^{-z_1}} \quad \frac{1}{1 + e^{-z_2}} \quad \cdots \quad \frac{1}{1 + e^{-z_m}}\right] \tag{7b}$$



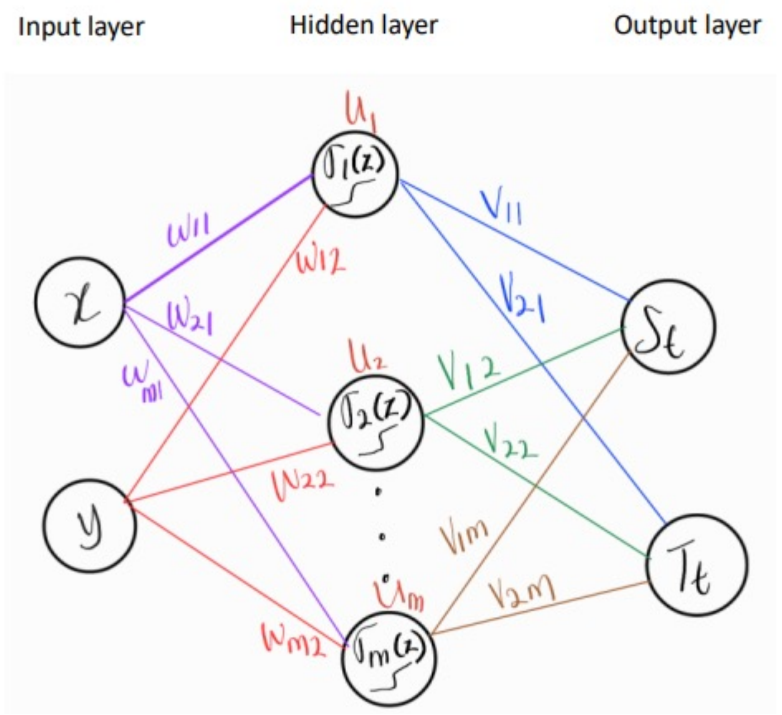
<u>The network output, S_t and T_t</u>

In Figure 4, S_t and T_t are the output and their neurons forms the output layer (the rightmost). What we want for these outputs to be as close as it can be to the correct ones. Specifically in our case to "mimic" Eq (1), we want S_t to be as close to s, and T_t to be as close to t. Once we able to do this, we can use the network to predict the output for the new or future input.

To get S_t and T_t to be as close as possible to s and t, respectively, we need to tune or train our weights and biases accordingly, a process we will detail later. Mathematically and by referring to Figure 4, we can state S_t and T_t as,

$$S_t = f(x, y, w, u, v) \quad (8a)$$

$$T_t = g(x, y, w, u, v) \quad (8b)$$



Training the network (allowing it to "learn")

Basically, there are four stages in building or completing an ML or a DL project:

i. Training
ii. Validation
iii. Testing
iv. Prediction/discovery (the ultimate use)

Training is the stage where we initialize the value of the parameters (e.g., weights and biases) then train or tune them iteratively (updating).

Validation would take place after the training. Validation involves the tuning of the hyperparameters such as learning rate, change of activation function, increase in the number of neurons and hidden layers etc. Validation might use the same sets of labelled data as in the training stage or different ones. The purpose of the validation is to check whether the performance of the network can be further optimized or be made more effective or/and economical.

Training the network (allowing it to "learn")

Basically, there are four stages in building or completing an ML or a DL project:

i. Training
ii. Validation
iii. Testing
iv. Prediction/discovery (the ultimate use)

Since stage ii, iii, and iv, are more a matter of implementation (rather than fundamentals or basic concepts), in this workshop, we focus on stage i, that is, on the training of the network.

In turn, there are two stages for the training of the network:

i. Forward-pass (calculating the network output)
ii. Backward-pass (backpropagation and updating)

Forward Pass (calculating the network output)

Step 1: Multiplication

Input x_k will be multiplied by weight w_{11} before goes into the first neuron through the channel. It will also be multiplied by weight w_{21} before goes into the second neuron, and so on. For input y_k, it will be multiplied by weight w_{12} before goes into the first neuron and so on.

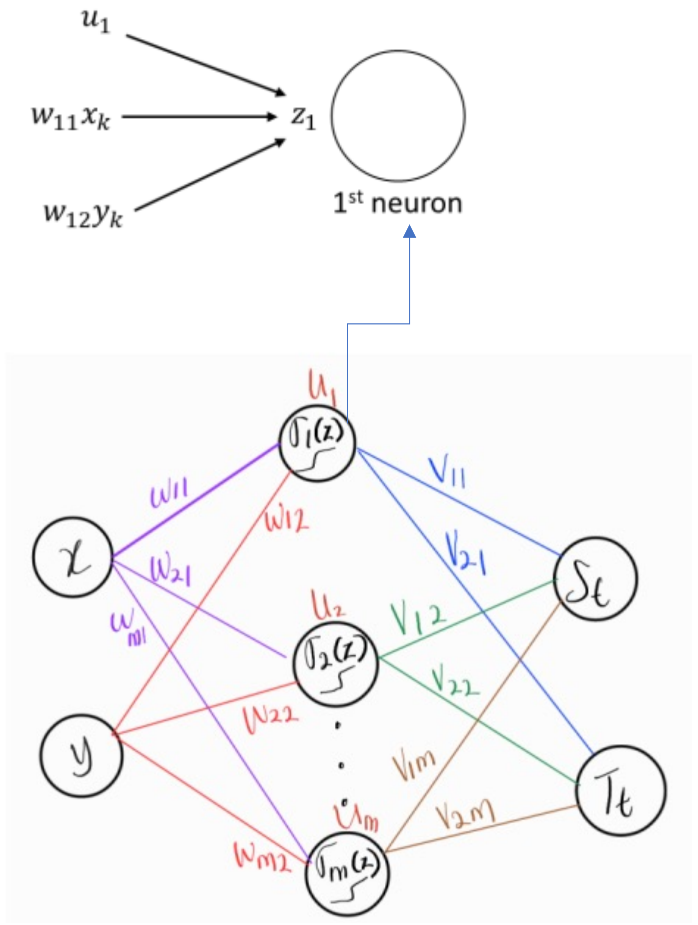
Step 2: Summation

But, before they enter the corresponding neuron, they will be summed first, then added by the corresponding bias. For example, for the 1st neuron, this will be

$$z_1 = w_{11}x_k + w_{12}y_k + u_1 \tag{9}$$

Whilst for the jth neuron,

$$z_j = w_{j1}x_k + w_{j2}y_k + u_j \tag{10}$$



Forward Pass (calculating the network output)

Step 3: Nonlinearization

In the neuron, the summed input (e.g. z) will be nonlinearized by the activation function. For the first neuron, this is given as (from Eqns. (7) and (9)),

$$\sigma_1 = \frac{1}{1+e^{-z_1}}$$

or

$$\sigma_1 = \frac{1}{1+e^{-(w_{11}x_k + w_{12}y_k + u_1)}} \tag{11}$$

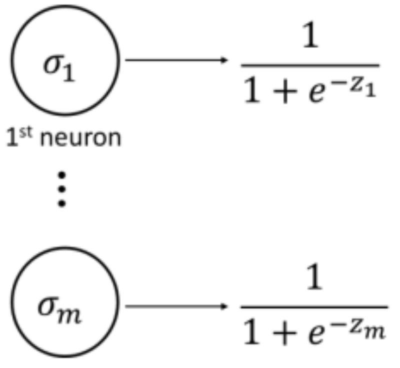
For the j-th neuron (from Eqns. (7) and (10)),

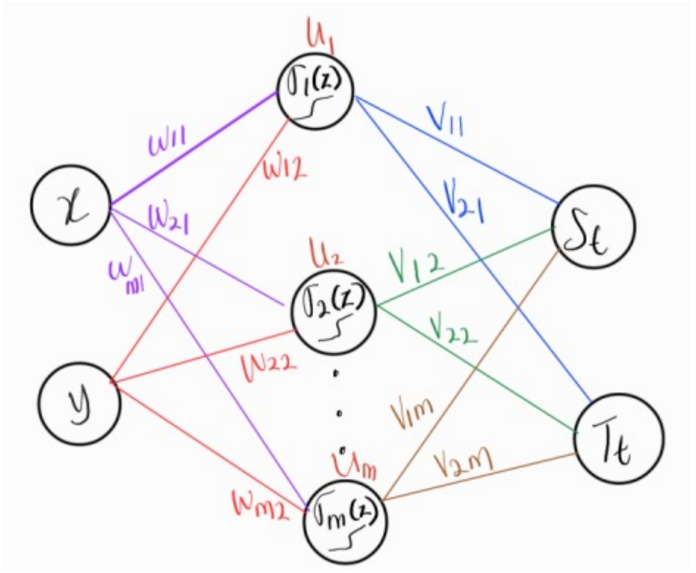
$$\sigma_j = \frac{1}{1+e^{-z_j}}$$

or

$$\sigma_j = \frac{1}{1+e^{-(w_{j1}x_k + w_{j2}y_k + u_j)}} \tag{12}$$

Note that, Eqns. (11) and (12) are also the output of the neurons in the hidden layer. This is graphically described by Figure 8.





Forward Pass (calculating the network output)

Step 4: Multiplication… again

For example, for the output of the first neuron, it will be multiplied by weight, v_{11} before it gets to output neuron, S_t. It will also be multiplied by v_{21} before it gets to output neuron, T_t. And so on.

Step 5: Summation and the calculation of network output, S_t and T_t

Finally, all the info (variables) will reach the output layer. They will reach S_t and T_t, accordingly. Usually, there will be no more nonlinearization in the output neuron, only summation, thus, (refer to Fig. 4);

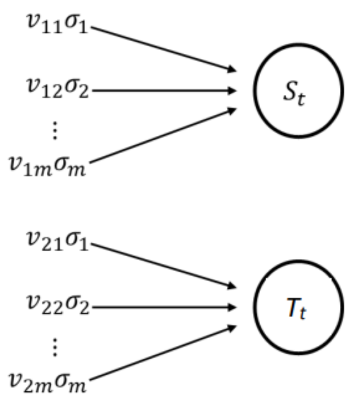
$$S_t = v_{11}\sigma_1 + v_{12}\sigma_2 \ \ldots + v_{1m}\sigma_m \tag{13}$$

$$T_t = v_{21}\sigma_1 + v_{22}\sigma_2 \ \ldots + v_{2m}\sigma_m \tag{14}$$

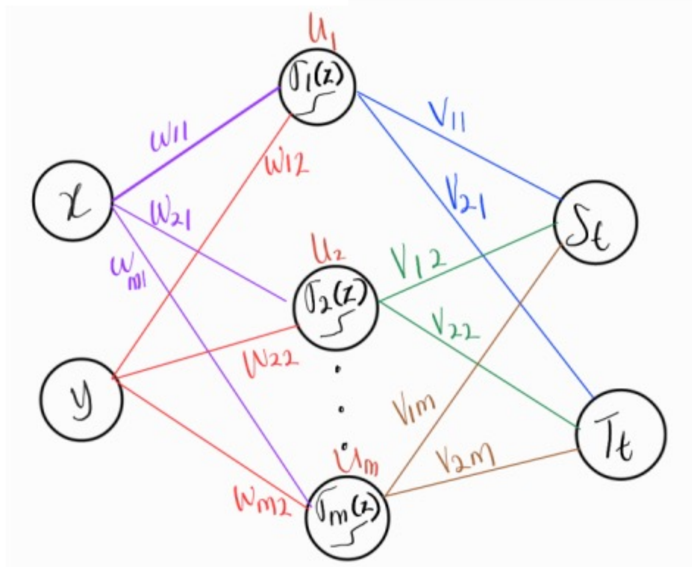
or (from Eqns. (11) and (12))

$$S_t = v_{11}\left(\frac{1}{1+\ e^{-(w_{11}x_k+w_{12}y_k+U_1)}}\right) + \ \ldots + v_{1m}\left(\frac{1}{1+\ e^{-(w_{m1}x_k+w_{m2}y_k+U_m)}}\right) \tag{15}$$

$$T_t = v_{21}\left(\frac{1}{1+\ e^{-(w_{11}x_k+w_{12}y_k+U_1)}}\right) + \ \ldots + v_{2m}\left(\frac{1}{1+\ e^{-(w_{m1}x_k+w_{m2}y_k+U_m)}}\right) \tag{16}$$



We have calculated the output!!!



Backward pass (backpropagation)

Errors

The errors specific for our network against Eqns. (1) can be given as

$$E_1 = (s - S_t) \quad (17a)$$
$$E_2 = (t - T_t) \quad (17b)$$

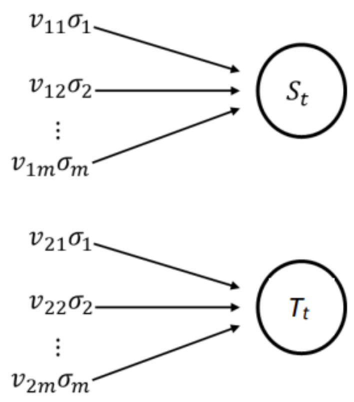
or from Eqns. (13) and (14) as,

$$E_1 = (s - (v_{11}\sigma_1 + v_{12}\sigma_2 + \cdots + v_{1m}\sigma_m)) \quad (18a)$$
$$E_2 = (t - (v_{21}\sigma_1 + v_{22}\sigma_2 + \cdots + v_{2m}\sigma_m)) \quad (18b)$$

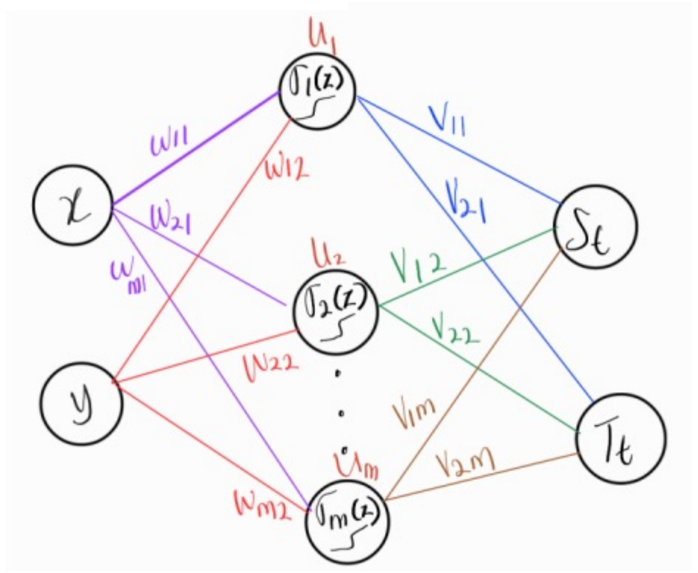
or from Eqns. (15) and (16) as,

$$E_1 = (s - (v_{11}(\frac{1}{1+e^{-(w_{11}x_k + w_{12}y_k + u_1)}}) + \cdots + v_{1m}(\frac{1}{1+e^{-(w_{m1}x_k + w_{m2}y_k + u_m)}}))) \quad (19a)$$

$$E_2 = (t - (v_{21}(\frac{1}{1+e^{-(w_{11}x_k + w_{12}y_k + u_1)}}) + \cdots + v_{2m}(\frac{1}{1+e^{-(w_{m1}x_k + w_{m2}y_k + u_m)}}))) \quad (19b)$$



We have calculated the output!!!



Backward pass (backpropagation)

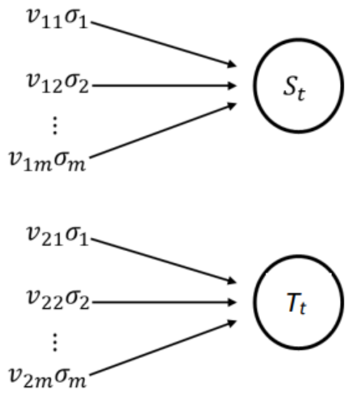
Loss function and its minimization by gradient descent

<u>Squared-Error Loss Function,</u>

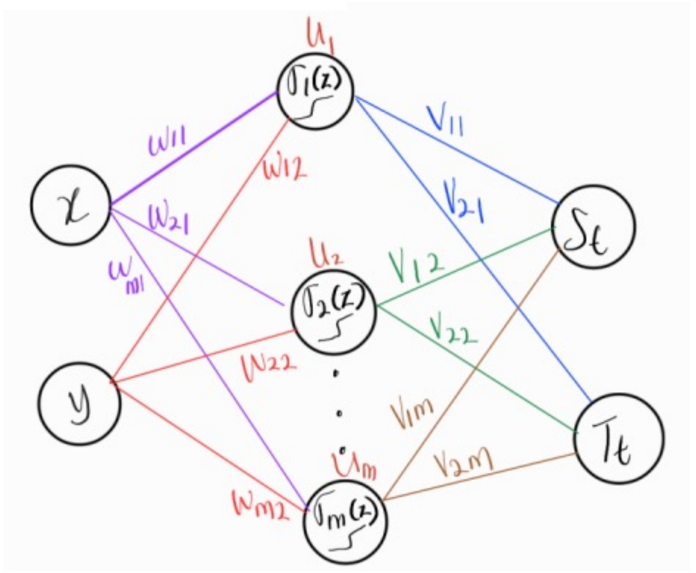
Loss function,
$$L = {E_1}^2 + {E_2}^2 \tag{20a}$$
$$L = (s - S_t)^2 + (t - T_t)^2 \tag{20b}$$

<u>Mean-Squared</u>

Loss function,
$$L = \frac{1}{n}\sum_{k=1}^{n}({E_{1,k}}^2 + {E_{2,k}}^2) \tag{21a}$$
$$L = \frac{1}{n}\sum_{k=1}^{n}((s_k - S_{t,k})^2 + (t_k - T_{t,k})^2) \tag{21b}$$

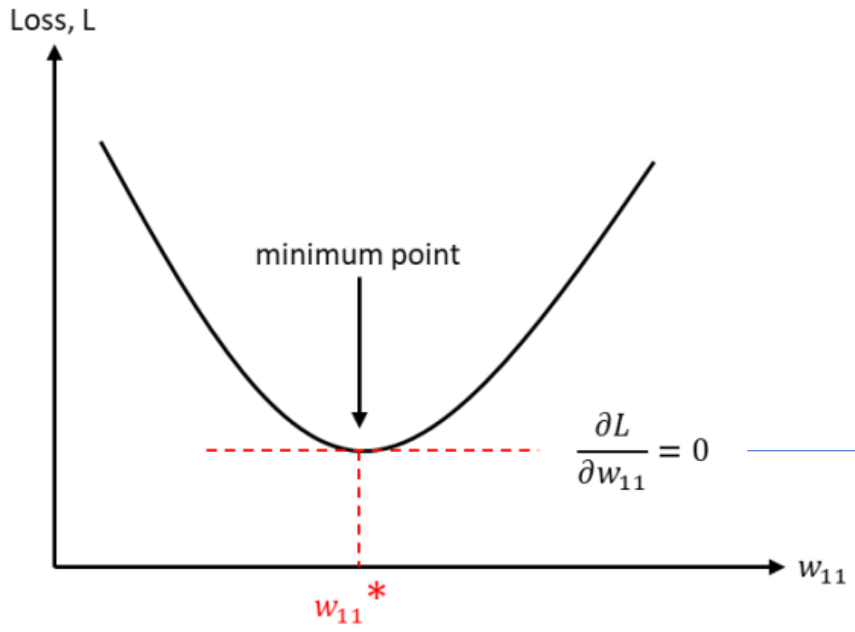


We have calculated the output!!!



Backward pass (backpropagation)

Loss function and its minimization by gradient descent

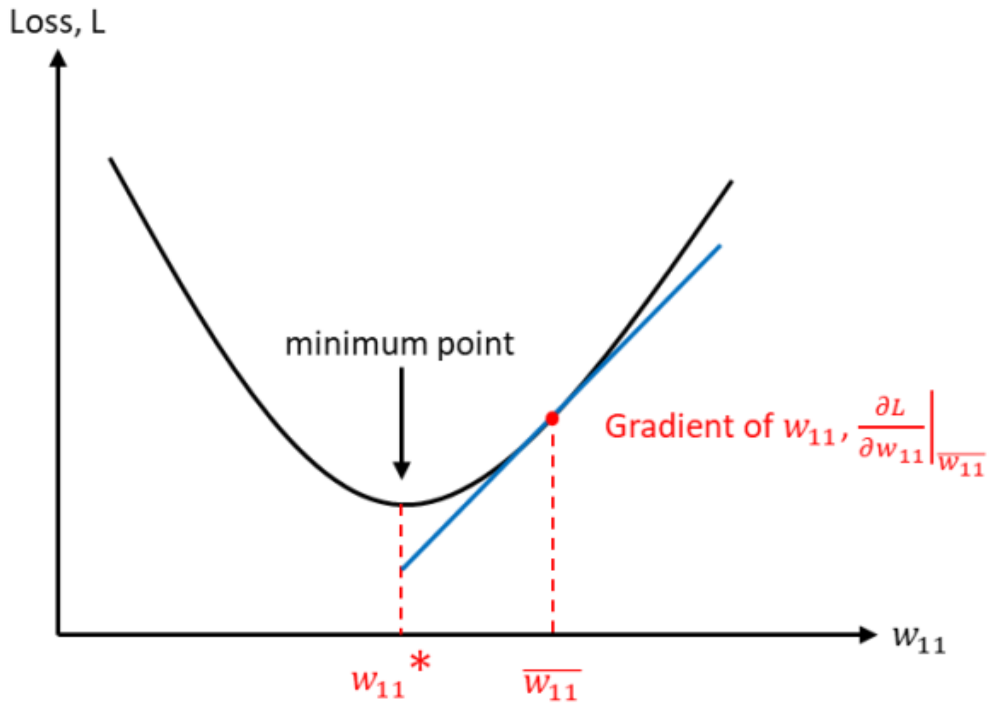


$$\frac{\partial L}{\partial w_{11}} = 0 \xrightarrow{} \text{Too "expensive"}$$

So, we resort to a different approach, that is the Gradient Descent (GD).

Backward pass (backpropagation)

Loss function and its minimization by gradient descent



Loss, L
minimum point
Gradient of w_{11}, $\left.\frac{\partial L}{\partial w_{11}}\right|_{\overline{w_{11}}}$
w_{11}
w_{11}^*
$\overline{w_{11}}$

Searching minimum point by gradient descent

$$w_{11}' = \overline{w_{11}} - \mu \left.\frac{\partial L}{\partial w_{11}}\right|_{\overline{w_{11}}} \tag{22}$$

where μ is a hyperparameter known as learning rate.

To express it in a familiar form you might read elsewhere, we rewrite Eqn. (22) below,

$$w_{11}' = w_{11} - \mu \frac{\partial L}{\partial w_{11}} \tag{23}$$

For the rest of weights and biases in our network, their update can be given as

$$w_{ji}' = w_{ji} - \mu \frac{\partial L}{\partial w_{ji}} \tag{24a}$$

$$u_j' = u_j - \mu \frac{\partial L}{\partial u_j} \tag{24b}$$

$$v_{ij}' = v_{ij} - \mu \frac{\partial L}{\partial v_{ij}} \tag{24c}$$

Gradients with respect to the parameters by chain-rule

For v_{ij}

$$v_{ij}' = v_{ij} \boxed{- \mu \frac{\partial L}{\partial v_{ij}}} \longrightarrow \frac{\partial L}{\partial v_{ij}} = 2\left(-E_1 \frac{\partial S_t}{\partial v_{ij}} - E_2 \frac{\partial T_t}{\partial v_{ij}}\right) \tag{25a}$$

For u_j

$$u_j' = u_j \boxed{- \mu \frac{\partial L}{\partial u_j}} \longrightarrow \frac{\partial L}{\partial u_j} = 2\left(-E_1 \frac{\partial S_t}{\partial u_j} - E_2 \frac{\partial T_t}{\partial u_j}\right) \tag{25b}$$

For w_{ji}

$$w_{ji}' = w_{ji} \boxed{- \mu \frac{\partial L}{\partial w_{ji}}} \longrightarrow \frac{\partial L}{\partial w_{ji}} = 2\left(-E_1 \frac{\partial S_t}{\partial w_{ji}} - E_2 \frac{\partial T_t}{\partial w_{ji}}\right) \tag{25c}$$

Chain-rule

Now, we will discuss how to get the six gradients in Eqns. (24) which are, $\frac{\partial S_t}{\partial v_{ij}}$, $\frac{\partial T_t}{\partial v_{ij}}$, $\frac{\partial S_t}{\partial u_j}$, $\frac{\partial T_t}{\partial u_j}$, $\frac{\partial S_t}{\partial w_{ji}}$, and $\frac{\partial T_t}{\partial w_{ji}}$, by chain-rule. But, why chain-rule? For the following reasons:

i. it is effective.

ii. to prepare you for future discussion on auto-differentiation or autograd (remember these terms) which chain-rule is key.

<u>i. Differentiation of *z*</u>

$$z_j = w_{j1}x_k + w_{j2}y_k + u_j \tag{26}$$

thus,

$$\frac{\partial z_j}{\partial w_{j1}} = x_k \tag{27a}$$

$$\frac{\partial z_j}{\partial w_{j2}} = y_k \tag{27b}$$

$$\frac{\partial z_j}{\partial u_j} = 1 \tag{27c}$$

ii. Differentiation of sigmoid,

$$\sigma_j = \frac{1}{1 + e^{-z_j}} \tag{28}$$

thus,

$$\frac{\partial \sigma_j}{\partial z_j} = \sigma_j(1 - \sigma_j) \tag{29}$$

iii. Differentiation of the output (S_t and T_t) (Eqns. (13) and (14))

$$S_t = v_{1j}\sigma_j \tag{30a}$$

$$T_t = v_{2j}\sigma_j \tag{30b}$$

thus,

$$\frac{\partial S_t}{\partial \sigma_j} = v_{1j} \tag{31a}$$

$$\frac{\partial T_t}{\partial \sigma_j} = v_{2j} \tag{31b}$$

Now, we are ready to employ the chain-rule to obtain our output gradients with respect to the parameters (weights and biases).

i. Output gradient with respect to w_{ij}

$$\frac{\partial S_t}{\partial w_{j1}} = \frac{\partial S_t}{\partial \sigma_j}\frac{\partial \sigma_j}{\partial z_j}\frac{\partial z_j}{\partial w_{j2}} \longrightarrow \frac{\partial S_t}{\partial w_{j1}} = v_{1j} x_k \sigma_j (1 - \sigma_j)$$

$$\frac{\partial S_t}{\partial w_{j2}} = \frac{\partial S_t}{\partial \sigma_j}\frac{\partial \sigma_j}{\partial z_j}\frac{\partial z_j}{\partial w_{j2}} \longrightarrow \frac{\partial S_t}{\partial w_{j2}} = v_{1j} y_k \sigma_j (1 - \sigma_j)$$

Repeating the same process, the gradients for T_t can be given as,

$$\frac{\partial T_t}{\partial w_{j1}} = v_{2j} x_k \sigma_j (1 - \sigma_j)$$

$$\frac{\partial T_t}{\partial w_{j2}} = v_{2j} y_k \sigma_j (1 - \sigma_j)$$

HERIOT WATT UNIVERSITY

UK | DUBAI | MALAYSIA

ii. Output gradient with respect to u_j

$$\frac{\partial S_t}{\partial u_j} = \frac{\partial S_t}{\partial \sigma_j}\frac{\partial \sigma_j}{\partial z_j}\frac{\partial z_j}{\partial u_j} \longrightarrow \frac{\partial S_t}{\partial u_j} = v_{1j}(1)\sigma_j(1-\sigma_j)$$

$$\frac{\partial T_t}{\partial u_j} = \frac{\partial T_t}{\partial \sigma_j}\frac{\partial \sigma_j}{\partial z_j}\frac{\partial z_j}{\partial u_j} \longrightarrow \frac{\partial T_t}{\partial u_j} = v_{2j}(1)\sigma_j(1-\sigma_j)$$

iii. Output gradient with respect to v_{ij}

$$\frac{\partial S_t}{\partial v_{1j}} = \sigma_j$$

$$\frac{\partial T_t}{\partial v_{2j}} = \sigma_j$$

Updating the network parameters (w_{ji}, u_j, v_{ij})

By inserting Eqns. (25) into (24) accordingly,

$$w'_{j1} = w_{j1} - 2\mu\,(-E_1 \frac{\partial S_t}{\partial w_{j1}} - E_2 \frac{\partial T_t}{\partial w_{j1}}) \tag{39a}$$

$$w'_{j2} = w_{j2} - 2\mu\,(-E_1 \frac{\partial S_t}{\partial w_{j2}} - E_2 \frac{\partial T_t}{\partial w_{j2}}) \tag{39b}$$

$$u'_j = u_j - 2\mu\,(-E_1 \frac{\partial S_t}{\partial u_j} - E_2 \frac{\partial T_t}{\partial u_j}) \tag{39c}$$

$$v'_{1j} = v_{1j} - 2\mu\,(-E_1 \frac{\partial S_t}{\partial v_{1j}}) \tag{39d}$$

$$v'_{2j} = v_{2j} - 2\mu\,(-E_2 \frac{\partial T_t}{\partial v_{2j}}) \tag{39e}$$

where,

E_1 and E_2 are given by Eqns. (17)

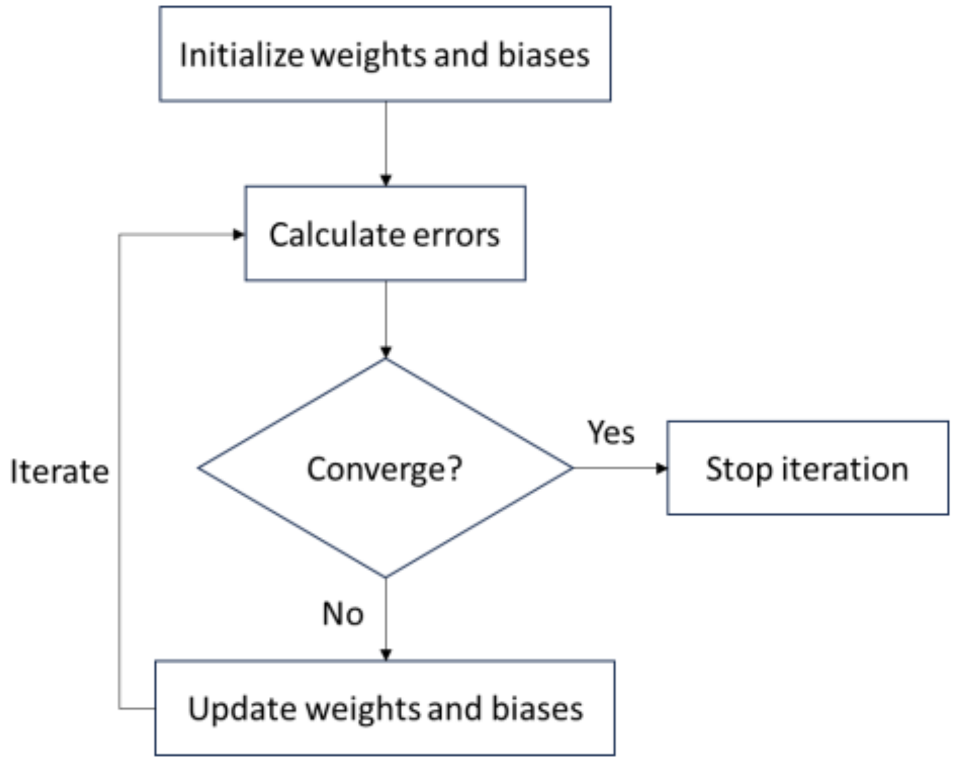
$\frac{\partial S_t}{\partial w_{j1}}$ and $\frac{\partial S_t}{\partial w_{j2}}$ are given by Eqns. (34)

$\frac{\partial T_t}{\partial w_{j1}}$ and $\frac{\partial T_t}{\partial w_{j2}}$ are given by Eqns. (35)

$\frac{\partial S_t}{\partial u_j}$ and $\frac{\partial T_t}{\partial u_j}$ are given by Eqns. (37)

$\frac{\partial S_t}{\partial v_{1j}}$ and $\frac{\partial T_t}{\partial v_{2j}}$ are given by Eqns. (38)

The network training flowchart



Flowchart of the training of network

Matlab code

The script

```
% STEP 1 : Generate inital Value for weight and biases
M = 2;
v = rand(2,M)-1/2;
u = rand(1,M)-1/2;
w = rand(2,M)-1/2;
```

The equations coded

$$[w] = \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{m1} \\ w_{12} & w_{22} & \cdots & w_{m2} \end{bmatrix} \tag{2}$$

$$[v] = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2m} \end{bmatrix} \tag{3}$$

$$[u] = \begin{bmatrix} u_1 & u_2 & \cdots & u_m \end{bmatrix} \tag{4}$$

Box 1

The script

```
xk = (round(rand(1,N)*L,3));
yk = (round(rand(1,N)*L,3));
```

The equations coded

$$\{\bar{x}\} = \begin{Bmatrix} x_k \\ y_k \end{Bmatrix} = \begin{Bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \end{Bmatrix} \qquad (2^*)$$

Box 2

The script

```
z = [];
for j = 1:M %loop over neuron
    zj = w(1,j)*xk + w(2,j)*yk + u(j);
    z = cat(1,z,zj);
```

The equations coded

$$z_m = w_{m1}x_k + w_{m2}y_k + u_m \qquad (10)$$

Box 3

The script

```
sig = 1 ./ (1 + exp(-z));
dsigdz = sig .* (1-sig);
```

The equations coded

$$\sigma_m = \frac{1}{1 + e^{-z_m}} \qquad \text{(12a) or (28)}$$

$$\frac{\partial \sigma_j}{\partial z_j} = \sigma_j(1 - \sigma_j) \qquad (29)$$

Box 4

The script

```
St = 0;
Tt = 0;
for j = 1:M %loop over neuron
    St = St + v(1,j) *sig(j,:);
    Tt = Tt + v(2,j) *sig(j,:);
end
```

The equations coded

$$S_t = v_{1j}\sigma_j \qquad (30a)$$

$$T_t = v_{2j}\sigma_j \qquad (30b)$$

Box 5

The script

```
% STEP 5 : Calculate the actual equations
s = c1*xk + c2*yk;
t = d1*xk + d2*yk;
```

The equations coded

$$3x + 2y = s \qquad (1a)$$

$$2x + 6y = t \qquad (1b)$$

Box 6

The script

```
% STEP 6 : Calculate the residual errors
E1 = (s - St);
E2 = (t - Tt);
```

The equations coded

$$E_1 = (s - S_t) \tag{17a}$$

$$E_2 = (t - T_t) \tag{17b}$$

Box 7

The script

```
% diff over v1
dStdv1 = -sig(j,:);

% diff over v2
dTtdv2 = -sig(j,:);
```

The equations coded

$$\frac{\partial S_t}{\partial v_{1j}} = \sigma_j \tag{38a}$$

$$\frac{\partial T_t}{\partial v_{2j}} = \sigma_j \tag{38b}$$

Box 8

The script

```
% diff over u
dStdu = -v(1,j) *dsigdz(j,:);
dTtdu = -v(2,j) *dsigdz(j,:);
```

The equations coded

$$\frac{\partial S_t}{\partial u_j} = v_{1j}(1)\sigma_j(1-\sigma_j) \qquad (37a)$$

$$\frac{\partial T_t}{\partial u_j} = v_{2j}(1)\sigma_j(1-\sigma_j) \qquad (37b)$$

Box 9

The script

```
% diff over w1
dStdw1 = -v(1,j) *xk .*dsigdz(j,:);
dTtdw1 = -v(2,j) *xk .*dsigdz(j,:);

% diff over w2
dStdw2 = -v(1,j) *yk .*dsigdz(j,:);
dTtdw2 = -v(2,j) *yk .*dsigdz(j,:);
```

The equations coded

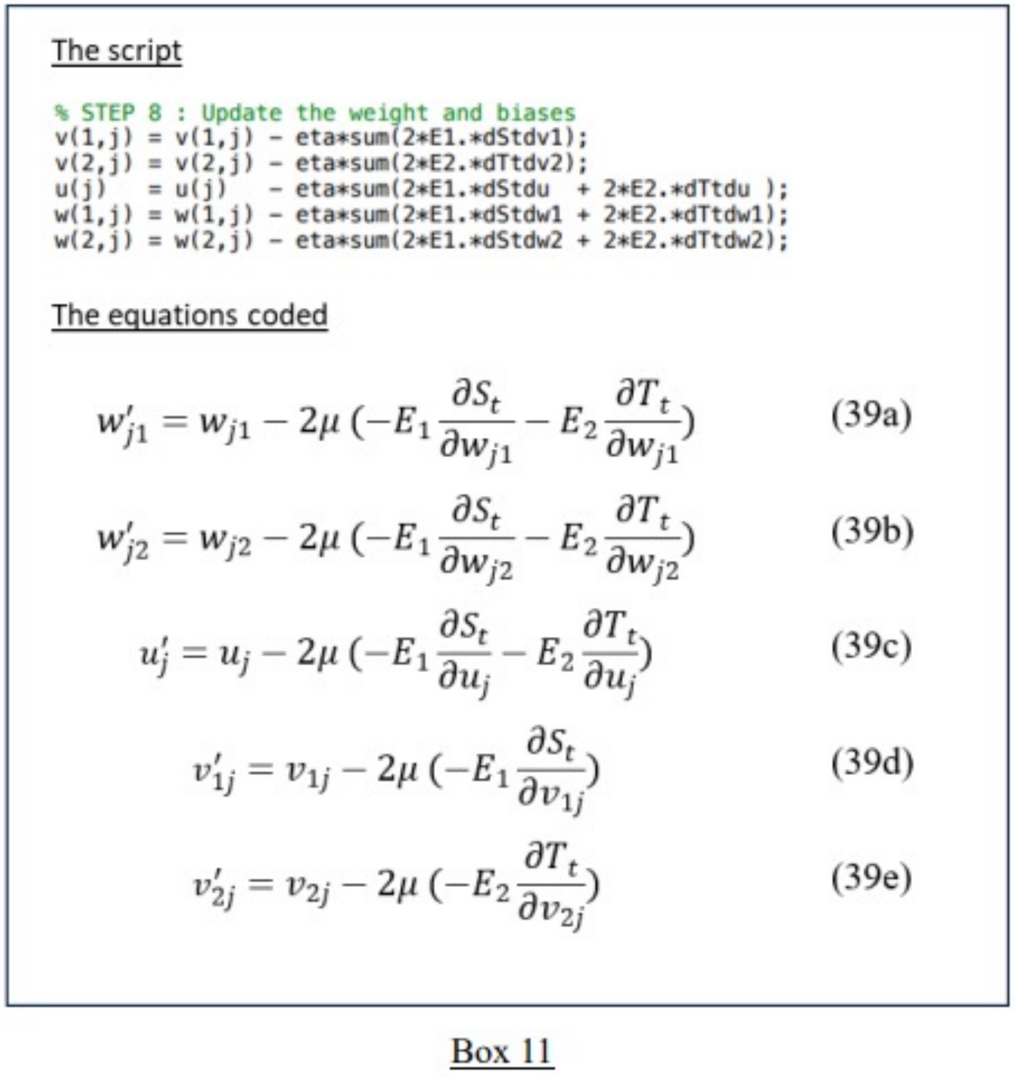
$$\frac{\partial S_t}{\partial w_{j1}} = v_{1j}x_k\sigma_j(1 - \sigma_j) \qquad (34a)$$

$$\frac{\partial S_t}{\partial w_{j2}} = v_{1j}y_k\sigma_j(1 - \sigma_j) \qquad (34b)$$

$$\frac{\partial T_t}{\partial w_{j1}} = v_{2j}x_k\sigma_j(1 - \sigma_j) \qquad (35a)$$

$$\frac{\partial T_t}{\partial w_{j2}} = v_{2j}y_k\sigma_j(1 - \sigma_j) \qquad (35b)$$

Box 10



The script

```
% STEP 8 : Update the weight and biases
v(1,j) = v(1,j) - eta*sum(2*E1.*dStdv1);
v(2,j) = v(2,j) - eta*sum(2*E2.*dTtdv2);
u(j)   = u(j)   - eta*sum(2*E1.*dStdu  + 2*E2.*dTtdu );
w(1,j) = w(1,j) - eta*sum(2*E1.*dStdw1 + 2*E2.*dTtdw1);
w(2,j) = w(2,j) - eta*sum(2*E1.*dStdw2 + 2*E2.*dTtdw2);
```

The equations coded

$$w'_{j1} = w_{j1} - 2\mu\,(-E_1 \frac{\partial S_t}{\partial w_{j1}} - E_2 \frac{\partial T_t}{\partial w_{j1}}) \tag{39a}$$

$$w'_{j2} = w_{j2} - 2\mu\,(-E_1 \frac{\partial S_t}{\partial w_{j2}} - E_2 \frac{\partial T_t}{\partial w_{j2}}) \tag{39b}$$

$$u'_j = u_j - 2\mu\,(-E_1 \frac{\partial S_t}{\partial u_j} - E_2 \frac{\partial T_t}{\partial u_j}) \tag{39c}$$

$$v'_{1j} = v_{1j} - 2\mu\,(-E_1 \frac{\partial S_t}{\partial v_{1j}}) \tag{39d}$$

$$v'_{2j} = v_{2j} - 2\mu\,(-E_2 \frac{\partial T_t}{\partial v_{2j}}) \tag{39e}$$

Box 11

HERIOT WATT UNIVERSITY

UK | DUBAI | MALAYSIA

That’s it!

Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs)

Consider the following wave PDE:

$$\frac{\partial^2 u}{\partial t^2} + a\frac{\partial^2 u}{\partial x^2} = 0 \qquad (1)$$

where u is displacement and a is a constant coefficient.

The boundary conditions are:

$$u|_{x=0} = u|_{x=1} = 0 \qquad (2)$$

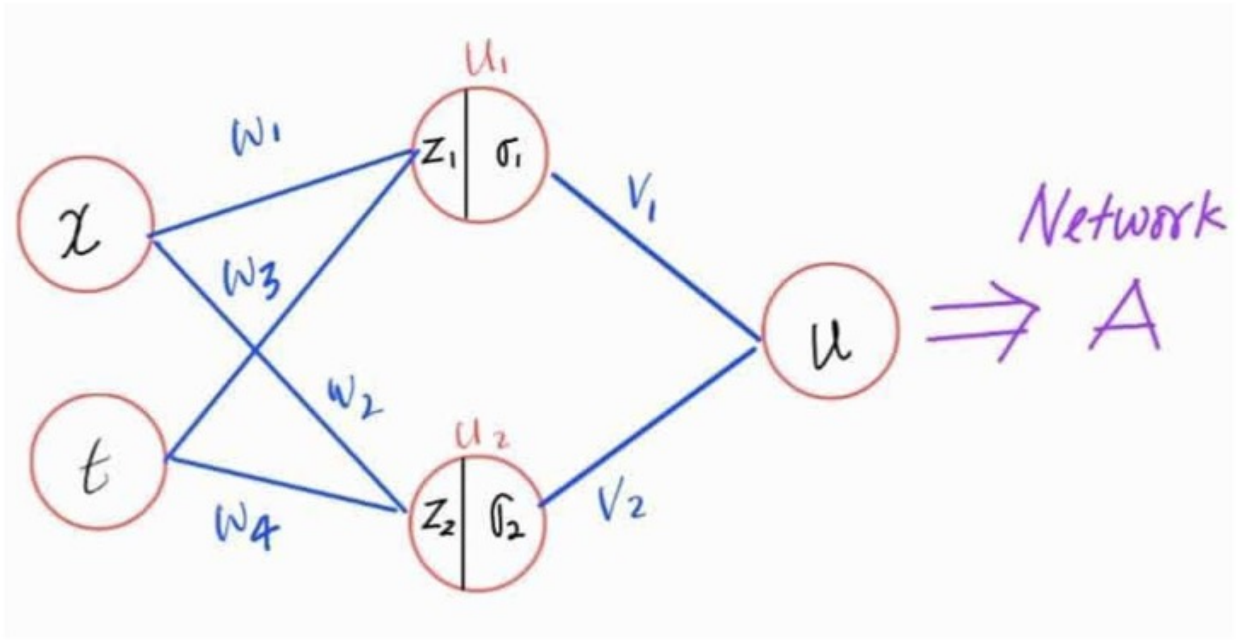
and the initial conditions are:

$$u|_{t=0} = \sin(\pi x) \qquad (3)$$

$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = 0 \qquad (4)$$

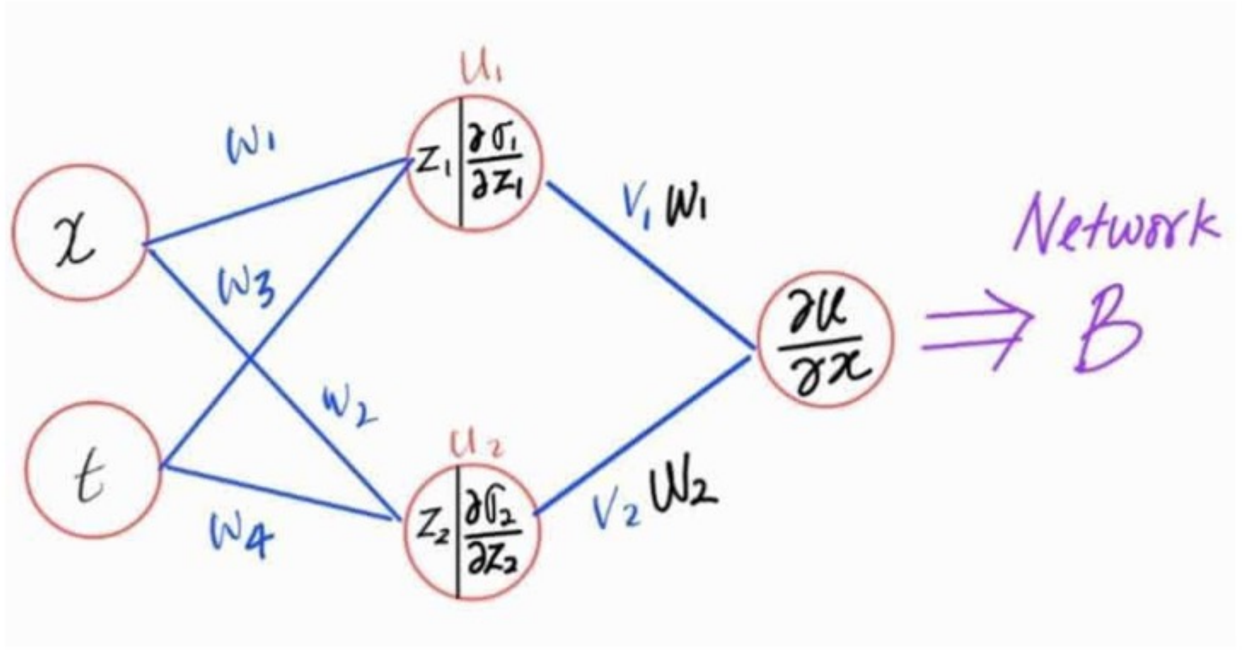
Physics-Informed Neural Networks (PINNs)

We can represent the "derivatives" of the network as follows:



The zeroth derivative, the network itself

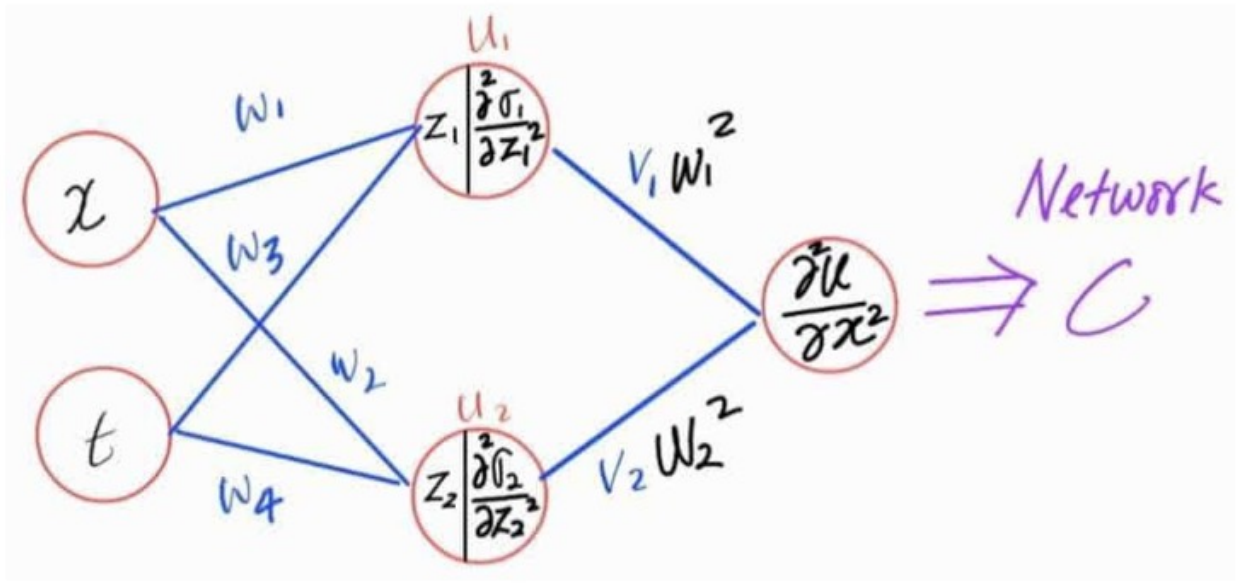
Physics-Informed Neural Networks (PINNs)



u_1
w_1
z_1
$\frac{\partial \sigma_1}{\partial z_1}$
x
w_3
$v_1 w_1$
Network
$\frac{\partial u}{\partial x}$
$\Rightarrow B$
w_2
t
u_2
z_2
$\frac{\partial \sigma_2}{\partial z_2}$
$v_2 w_2$
w_4

The first derivative

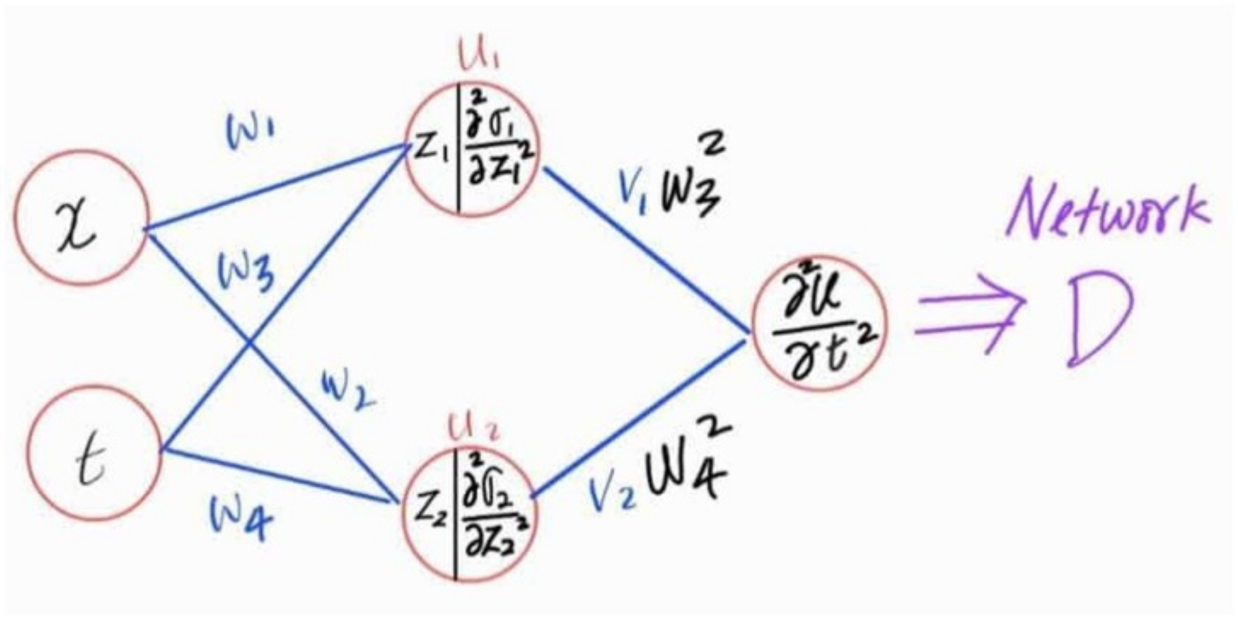
Physics-Informed Neural Networks (PINNs)



u_1
u_2
x
t
w_1
w_2
w_3
w_4
z_1
$\frac{\partial^2 \sigma_1}{\partial z_1^2}$
z_2
$\frac{\partial^2 \sigma_2}{\partial z_2^2}$
$v_1 w_1^2$
$v_2 w_2^2$
$\frac{\partial^2 u}{\partial x^2}$
Network
$\Rightarrow C$

The 2nd derivative with respect to x

Physics-Informed Neural Networks (PINNs)



x
t
w_1
w_3
w_2
w_4
u_1
u_2
z_1
$\frac{\partial^2 \sigma_1}{\partial z_1^2}$
z_2
$\frac{\partial^2 \sigma_2}{\partial z_2^2}$
$v_1 w_3^2$
$v_2 w_4^2$
$\frac{\partial^2 u}{\partial t^2}$
Network
$\Rightarrow D$

The 2nd derivative with respect to *t*

Physics-Informed Neural Networks (PINNs)

With these "networks" we can represent Eq. (1) as (with the error denoted by E1):

$$D + a*C = E1 \qquad (5)$$

Eq. (3) and (4) as (with the error denoted by E2):

$$\mathrm{A} - \sin(\pi x) + \mathrm{B} = \mathrm{E2} \qquad (6)$$

and Eq. (2) as (with the error denoted by E3):

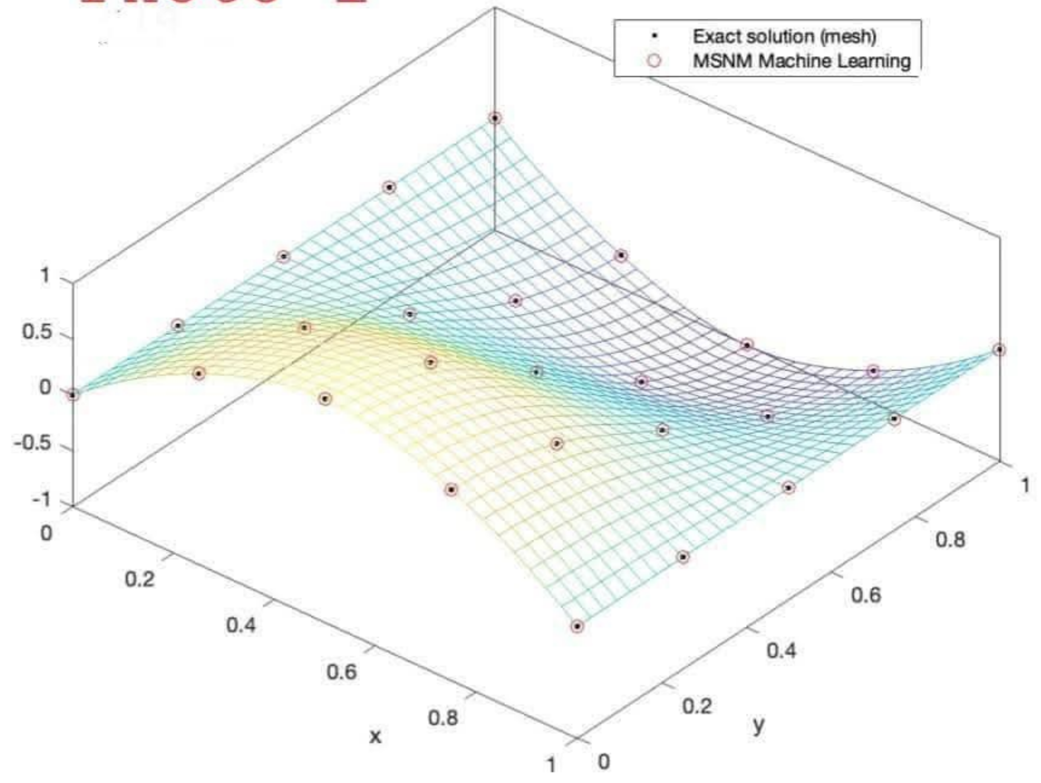
$$\mathrm{A} = \mathrm{E3} \qquad (7)$$

Now, the Loss, L is calculated as

$$\text{Loss, L} = \mathrm{E1}^2 + \mathrm{E2}^2 + \mathrm{E3}^2 \qquad (8)$$

Physics-Informed Neural Networks (PINNs)

Photo 1



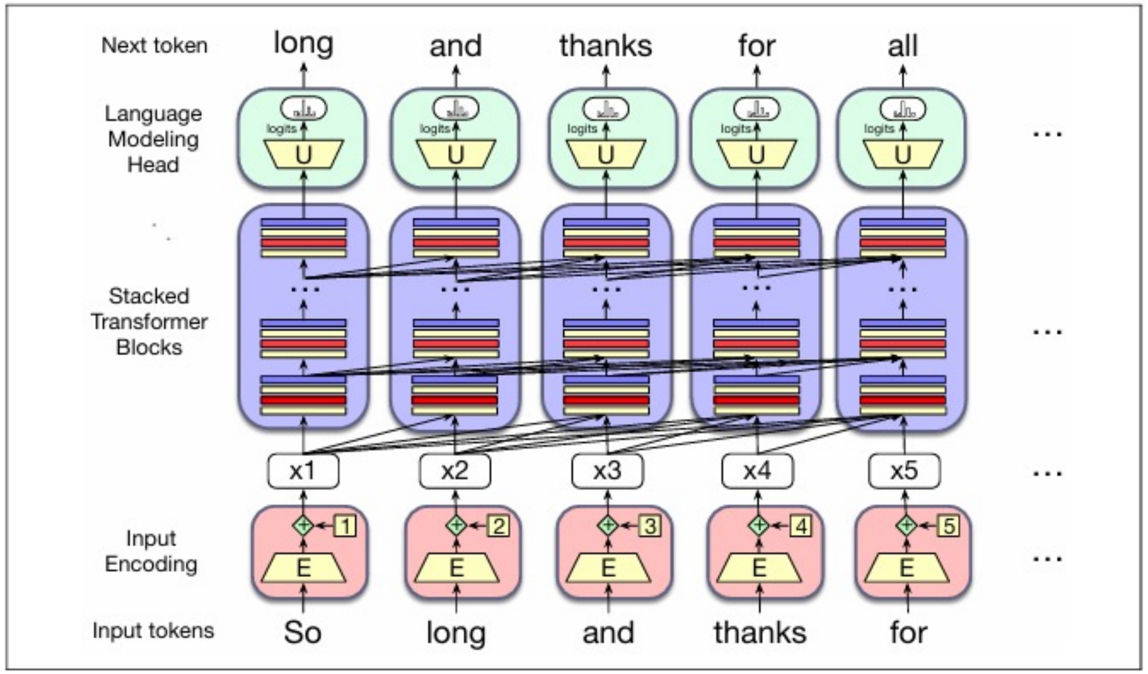
The result and its verification of the wave equation code I wrote with Dr Razin in Matlab.

HERIOT WATT UNIVERSITY

UK | DUBAI | MALAYSIA

Large Language Model (LLM) and Transformers

The basics of Transformer thus LLM



The main questions:

1. How the words become numbers then input?
2. How the model identify the target of the output?
3. How the model "inject" sentiments to the words?

The basics of Transformer thus LLM

Transformer Forward + Training Pipeline (Encoder with **2 layers**)

1. Tokenize input → get token IDs

Text → tokenizer → integer IDs.

2. Convert IDs into embeddings (semantic + positional) → X_0

X0 = Etoken(ID)+Epos(position)

This becomes the input to the first Transformer layer.

Transformer Layer 1

3. Apply self-attention (Q, K, V computed from X_0)

$Q = X_0W_Q$,

$K = X_0W_K$,

$V = X_0W_V$,

The basics of Transformer thus LLM

4. Compute attention scores, softmax and weighted sum of V

$$\boxed{\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V}$$

5. Concatenate heads + linear projection

6. Residual + LayerNorm (Post-Attention Norm)
X_1=LayerNorm(X_0+Attention(X_0))
X_1 is the input to FFN

7. Feedforward Network (FFN) on X_1

8. Residual + LayerNorm (Post-FFN Norm)
X_2=LayerNorm(X_1+FFN(X_1))

This completes Layer 1 output where X_2 will become input to the next stack

The basics of Transformer thus LLM

Transformer Layer 2 (Second Stack)

9. Second Self-Attention using X_2

Repeat the same steps:

- Q, K, V from X_2

- Attention, Residual + LayerNorm

Produces X_3.

10. Second Feedforward → residual + normalization → X_4.

This is the final encoder output, X_4

The basics of Transformer thus LLM

Training Phase

11. Compute loss based on model output (X_4)

Compare model predictions to target labels using cross-entropy or another loss.

12. Backpropagation (gradients flow through ALL layers including embeddings)

13. Update weights using optimizer (Adam, etc.)

14. Repeat until convergence

The basics of Transformer thus LLM

Simple hand-calculation example

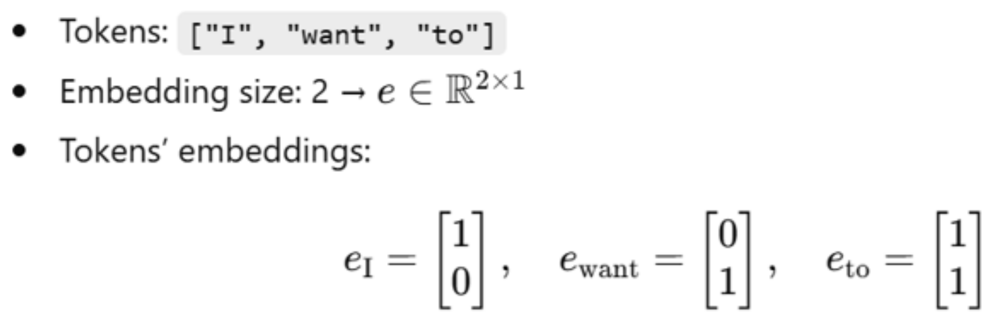
1. Tokenize input → get token IDs

Text → tokenizer → integer IDs.

2. Convert IDs into embeddings (semantic + positional) → X_0

X0 = Etoken(ID)+Epos(position)

This becomes the input to the first Transformer layer.



- Tokens: `["I", "want", "to"]`
- Embedding size: 2 → $e \in \mathbb{R}^{2\times 1}$
- Tokens' embeddings:

$$e_{\mathrm{I}} = \begin{bmatrix}1\\0\end{bmatrix}, \quad e_{\mathrm{want}} = \begin{bmatrix}0\\1\end{bmatrix}, \quad e_{\mathrm{to}} = \begin{bmatrix}1\\1\end{bmatrix}$$

The basics of Transformer thus LLM

Transformer Layer 1

3. Apply self-attention (Q, K, V computed from X_0)

- Weight matrices to generate **Q, K, V** (all 2×2 for simplicity):

$$W_Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad W_K = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad W_V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$Q = X_0 W_Q$,

$$Q_{\mathrm{I}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$Q_{\text{want}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad Q_{\text{to}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The basics of Transformer thus LLM

$K = X_0 W_K$,

$$K_{\mathrm{I}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

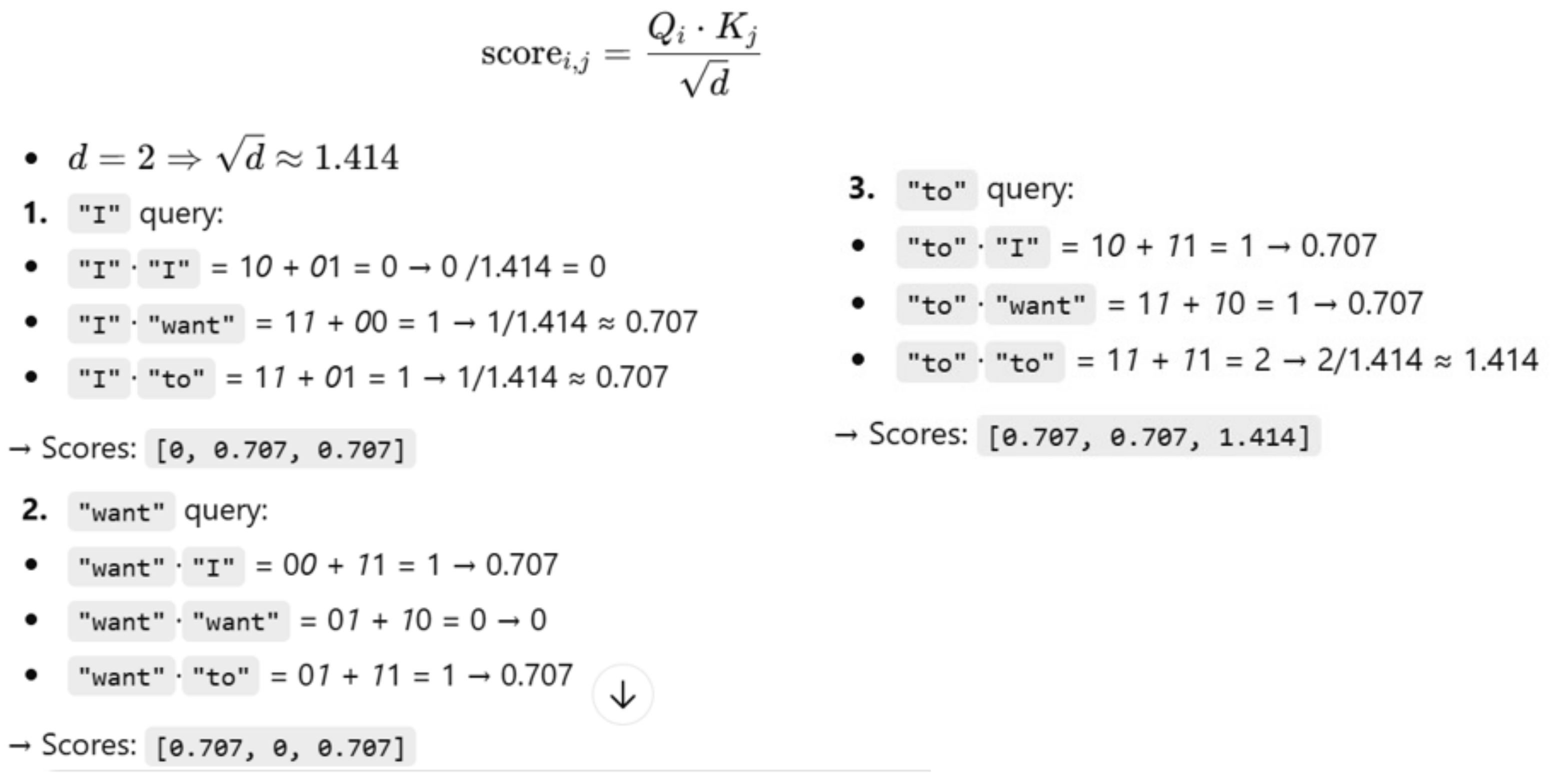
$$K_{\text{want}} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad K_{\text{to}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$V = X_0 W_V$,

$$V_{\mathrm{I}} = [1, 0]^T, \quad V_{\text{want}} = [0, 1]^T, \quad V_{\text{to}} = [1, 1]^T$$

The basics of Transformer thus LLM

4. Compute attention scores, softmax and weighted sum of V



$$\text{score}_{i,j} = \frac{Q_i \cdot K_j}{\sqrt{d}}$$

- $d = 2 \Rightarrow \sqrt{d} \approx 1.414$

1. `"I"` query:

- `"I"` · `"I"` = 10 + 01 = 0 → 0 /1.414 = 0
- `"I"` · `"want"` = 11 + 00 = 1 → 1/1.414 ≈ 0.707
- `"I"` · `"to"` = 11 + 01 = 1 → 1/1.414 ≈ 0.707

→ Scores: `[0, 0.707, 0.707]`

2. `"want"` query:

- `"want"` · `"I"` = 00 + 11 = 1 → 0.707
- `"want"` · `"want"` = 01 + 10 = 0 → 0
- `"want"` · `"to"` = 01 + 11 = 1 → 0.707

→ Scores: `[0.707, 0, 0.707]`

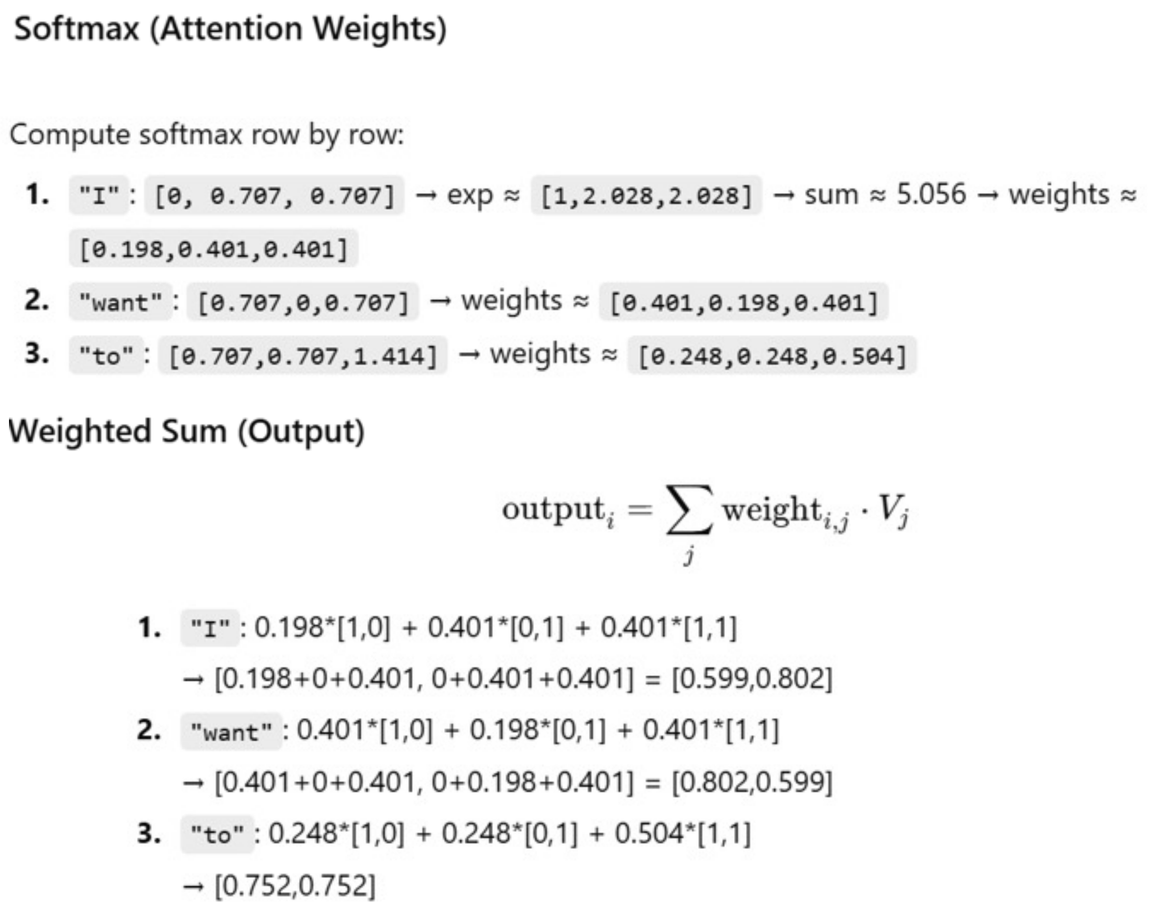
3. `"to"` query:

- `"to"` · `"I"` = 10 + 11 = 1 → 0.707
- `"to"` · `"want"` = 11 + 10 = 1 → 0.707
- `"to"` · `"to"` = 11 + 11 = 2 → 2/1.414 ≈ 1.414

→ Scores: `[0.707, 0.707, 1.414]`

The basics of Transformer thus LLM

4. Compute attention scores, softmax and weighted sum of V



Softmax (Attention Weights)

Compute softmax row by row:

1. `"I"`: `[0, 0.707, 0.707]` → exp ≈ `[1,2.028,2.028]` → sum ≈ 5.056 → weights ≈ `[0.198,0.401,0.401]`
2. `"want"`: `[0.707,0,0.707]` → weights ≈ `[0.401,0.198,0.401]`
3. `"to"`: `[0.707,0.707,1.414]` → weights ≈ `[0.248,0.248,0.504]`

Weighted Sum (Output)

$$\text{output}_i = \sum_j \text{weight}_{i,j} \cdot V_j$$

1. `"I"`: 0.198*[1,0] + 0.401*[0,1] + 0.401*[1,1]
   → [0.198+0+0.401, 0+0.401+0.401] = [0.599,0.802]
2. `"want"`: 0.401*[1,0] + 0.198*[0,1] + 0.401*[1,1]
   → [0.401+0+0.401, 0+0.198+0.401] = [0.802,0.599]
3. `"to"`: 0.248*[1,0] + 0.248*[0,1] + 0.504*[1,1]
   → [0.752,0.752]

HERIOT WATT UNIVERSITY

UK | DUBAI | MALAYSIA

References:

[1] Zhuhadar, Lily Popova & Lytras, Miltiadis. (2023). The Application of AutoML Techniques in Diabetes Diagnosis: Current Approaches, Performance, and Future Directions. Sustainability. 15. 13484. 10.3390/su151813484

[2] Sanubari, A. R., Kusuma, P. D., Setianingsih, C., Flood Modelling and Prediction Using Artificial Neural Network, The 2018 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)

[3] Michael Nielsen, Neural Networks and Deep Learning, http://neuralnetworksanddeeplearning.com

[4] Eva Prašnikar, Martin Ljubič. Andrej Perdih, Jure Borišek, Machine learning heralding a new development phase in molecular dynamics simulations, Artifcial Intelligence Review (2024) 57:102

[5] Kumar Puran Tripathy, Ashok K Mishra, Deep Learning in Hydrology and Water Resources: concepts, methods, applications and research directions, Journal of Hydrology, 628 (2024)

Feedback Form

a.mohd_yassin@hw.ac.uk

Airil Yasreen Mohd Yassin