Deep Learning Frameworks

Non-Linear Boundaries, Activation Functions, Backprop, Autograd, Dataset, DataLoader and Regularization

By Sakharam

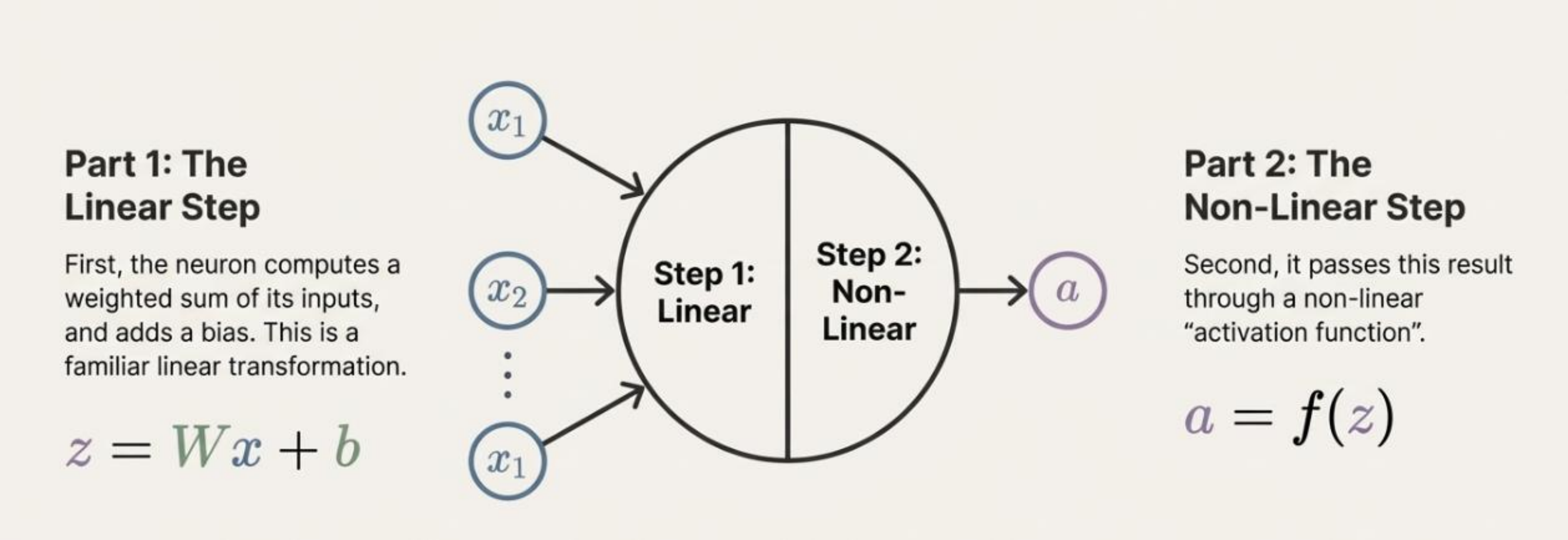
https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Recap

- Computation Graph
- Tensor and Operations
- Toy Dataset – Initializing Tensors (Random, Zeroes, Ones)
- Neural Network – nn.Sequential
- Training Loop - Skeleton

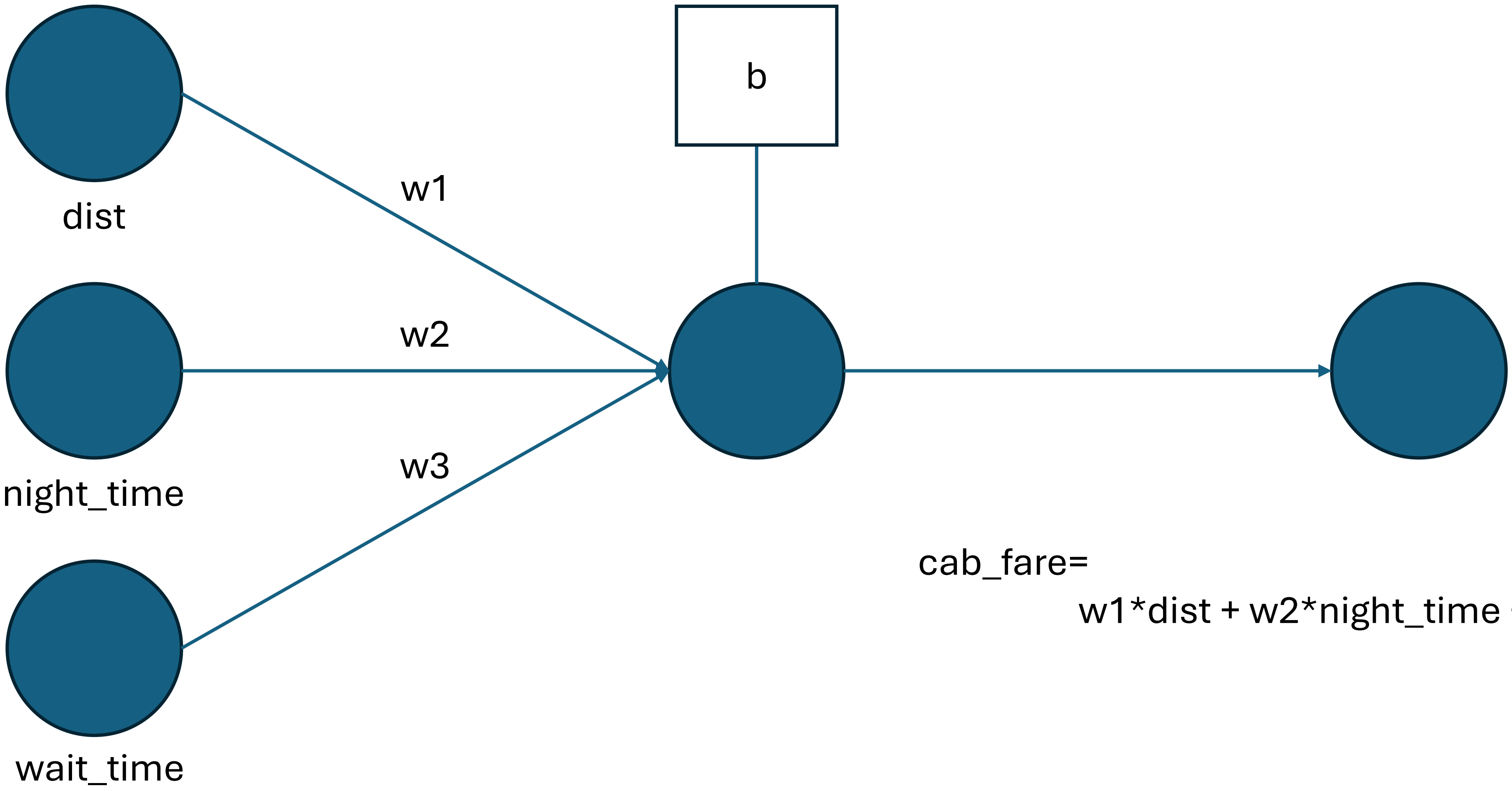
Neuron: Building Block of Neural Network



https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Neural Network with Linear Neurons

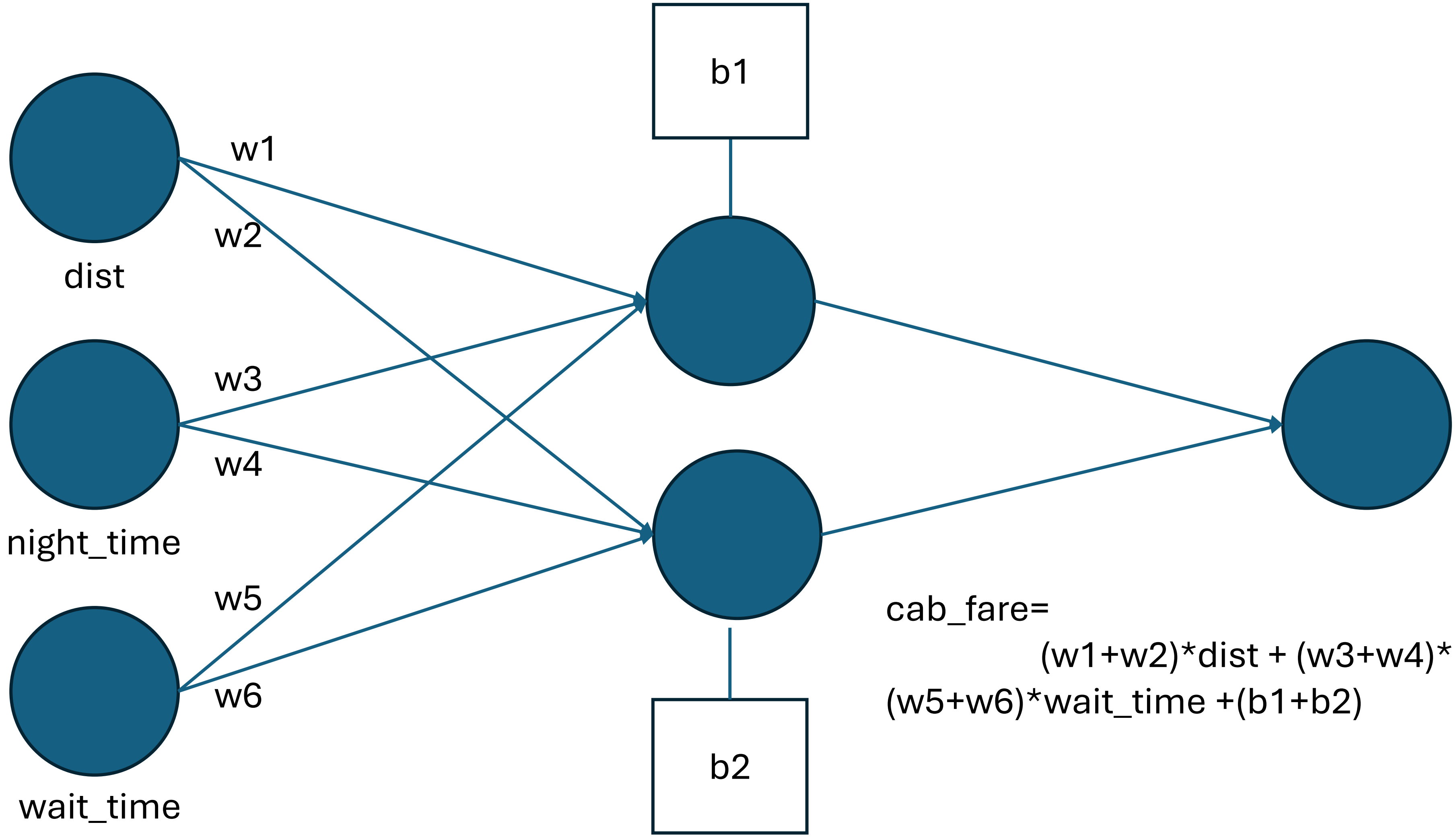


cab_fare=
w1*dist + w2*night_time + w3*wait_time +b

https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Neural Network with Linear Neurons

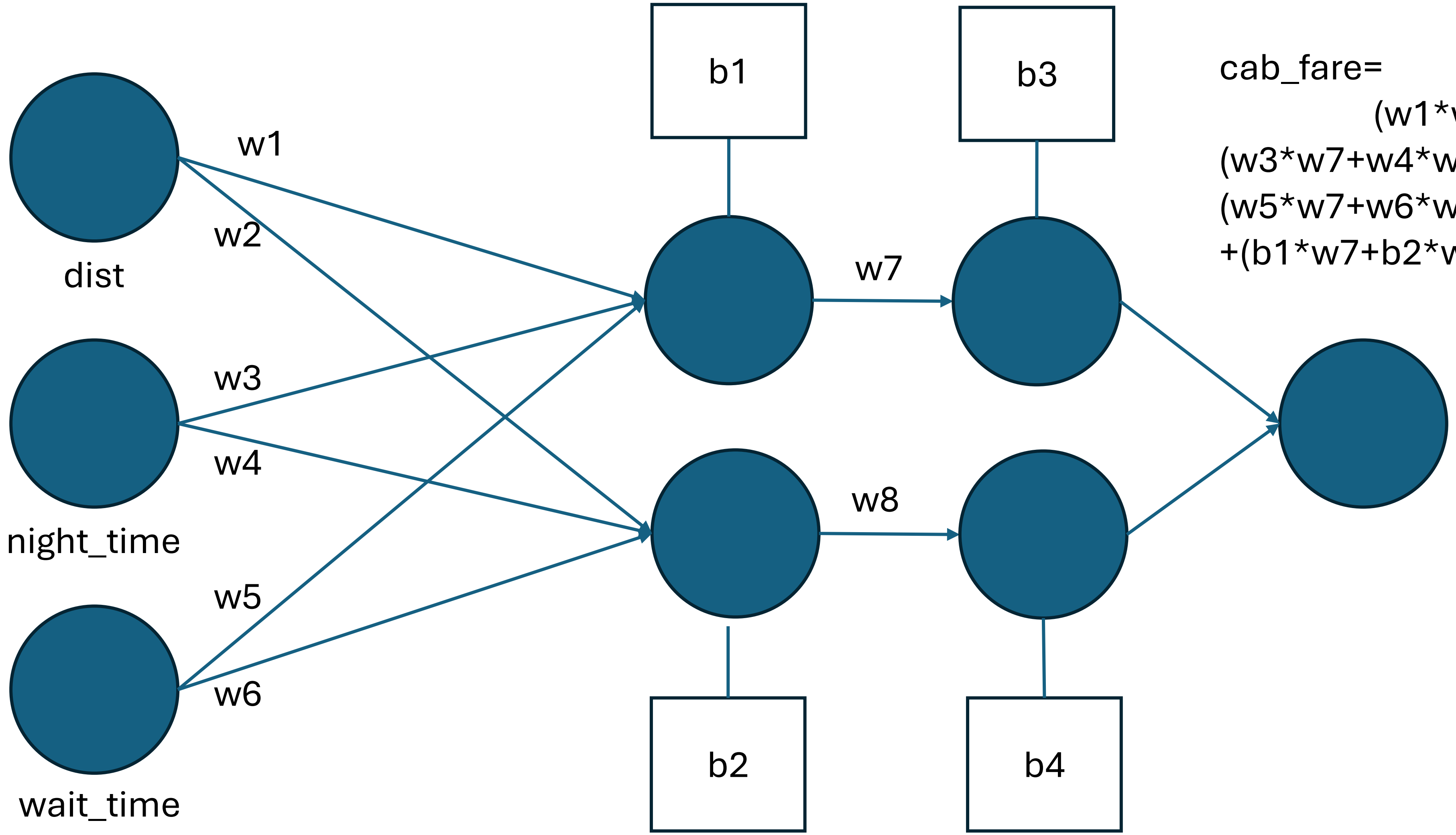


cab_fare=
(w1+w2)*dist + (w3+w4)*night_time +
(w5+w6)*wait_time +(b1+b2)

https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Neural Network with Linear Neurons

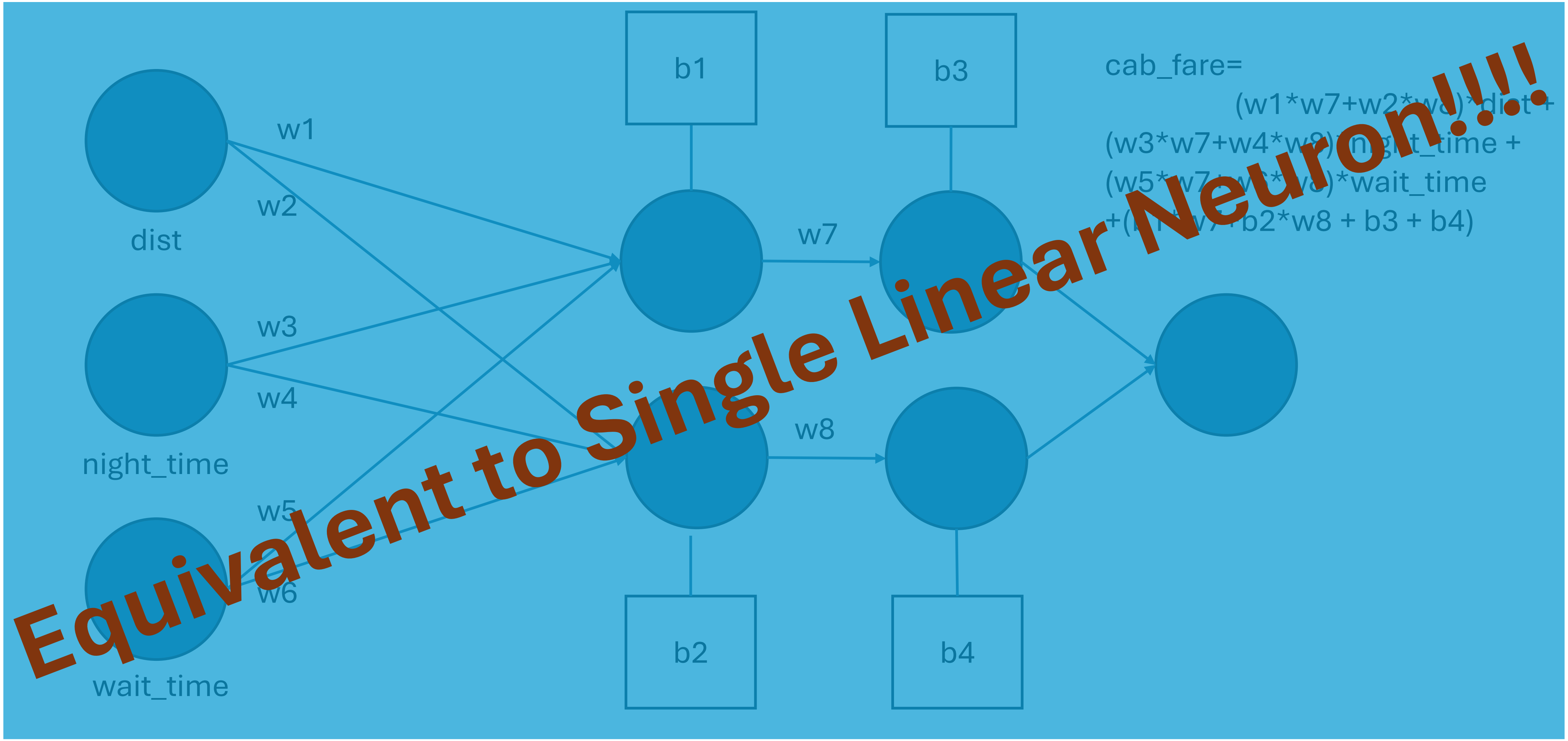


cab_fare=
(w1*w7+w2*w8)*dist +
(w3*w7+w4*w8)*night_time +
(w5*w7+w6*w8)*wait_time
+(b1*w7+b2*w8 + b3 + b4)

https://tinyurl.com/dlframeworks

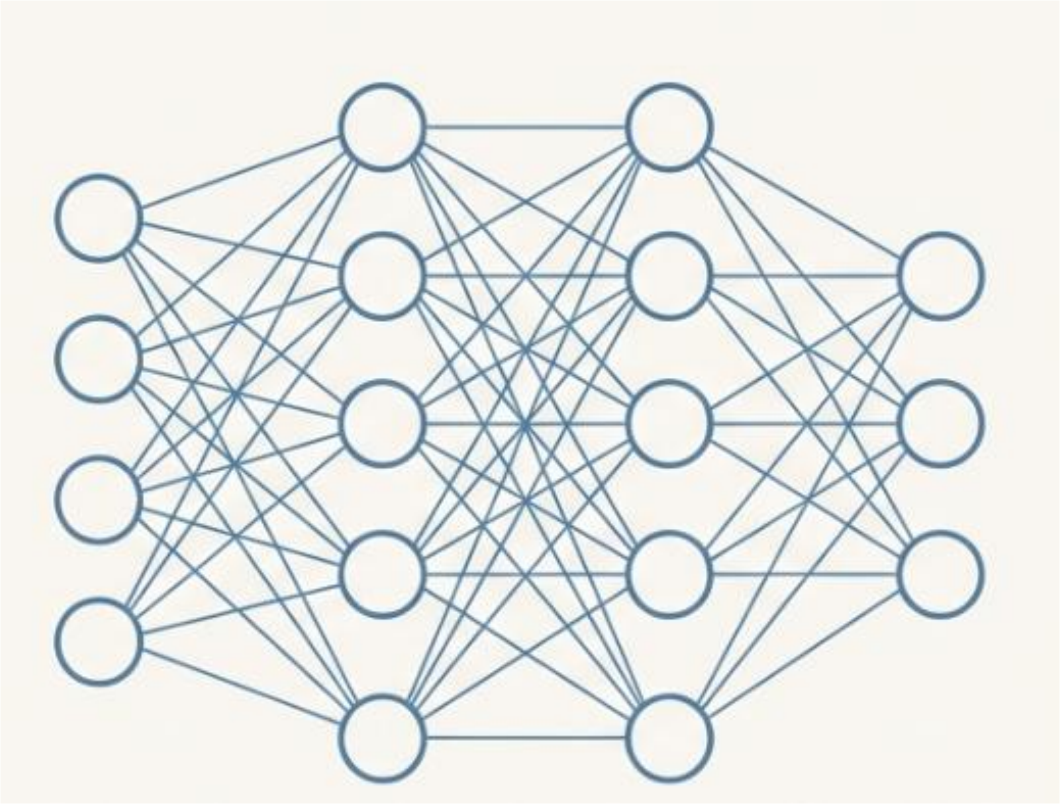
https://github.com/sakharamg/DeepLearningFrameworks

Neural Network with Linear Neurons



Neural Network with Linear Neurons

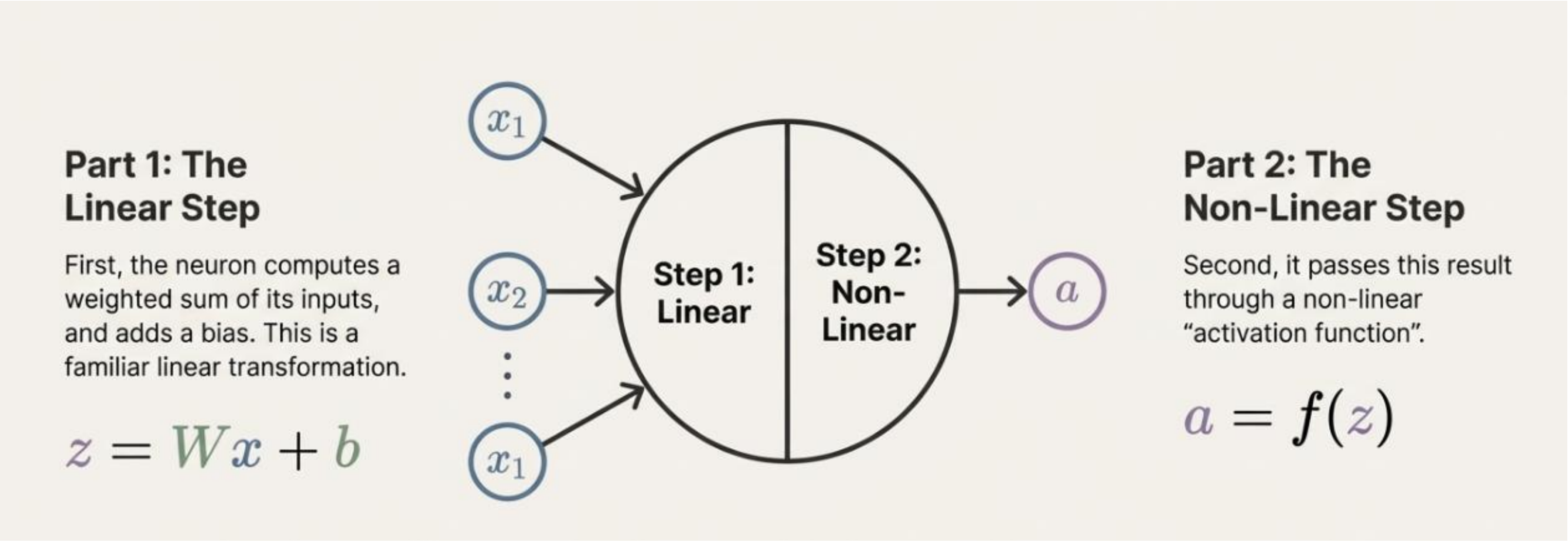
Still Linear



https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

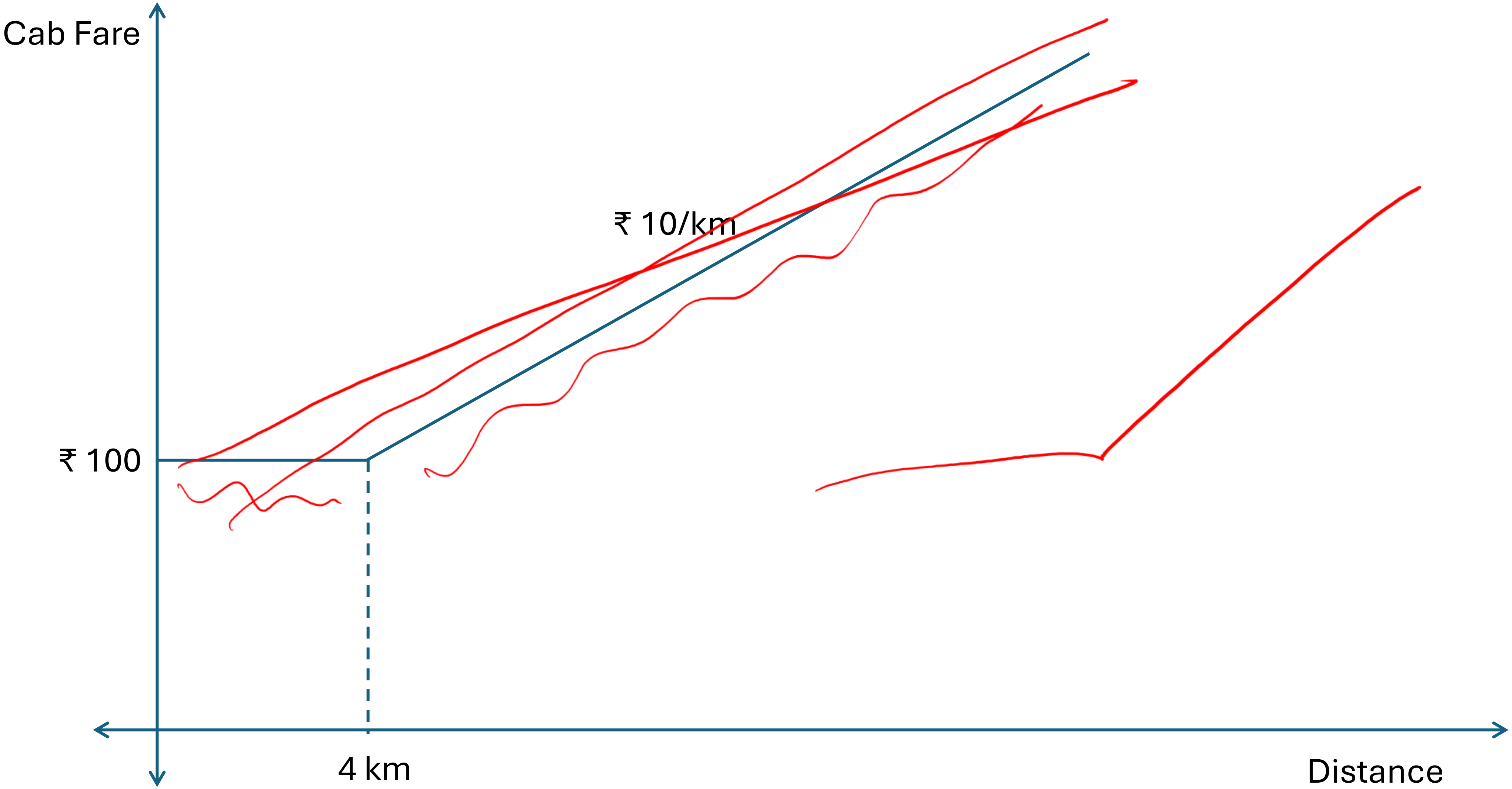
Activation Functions – Non Linearity



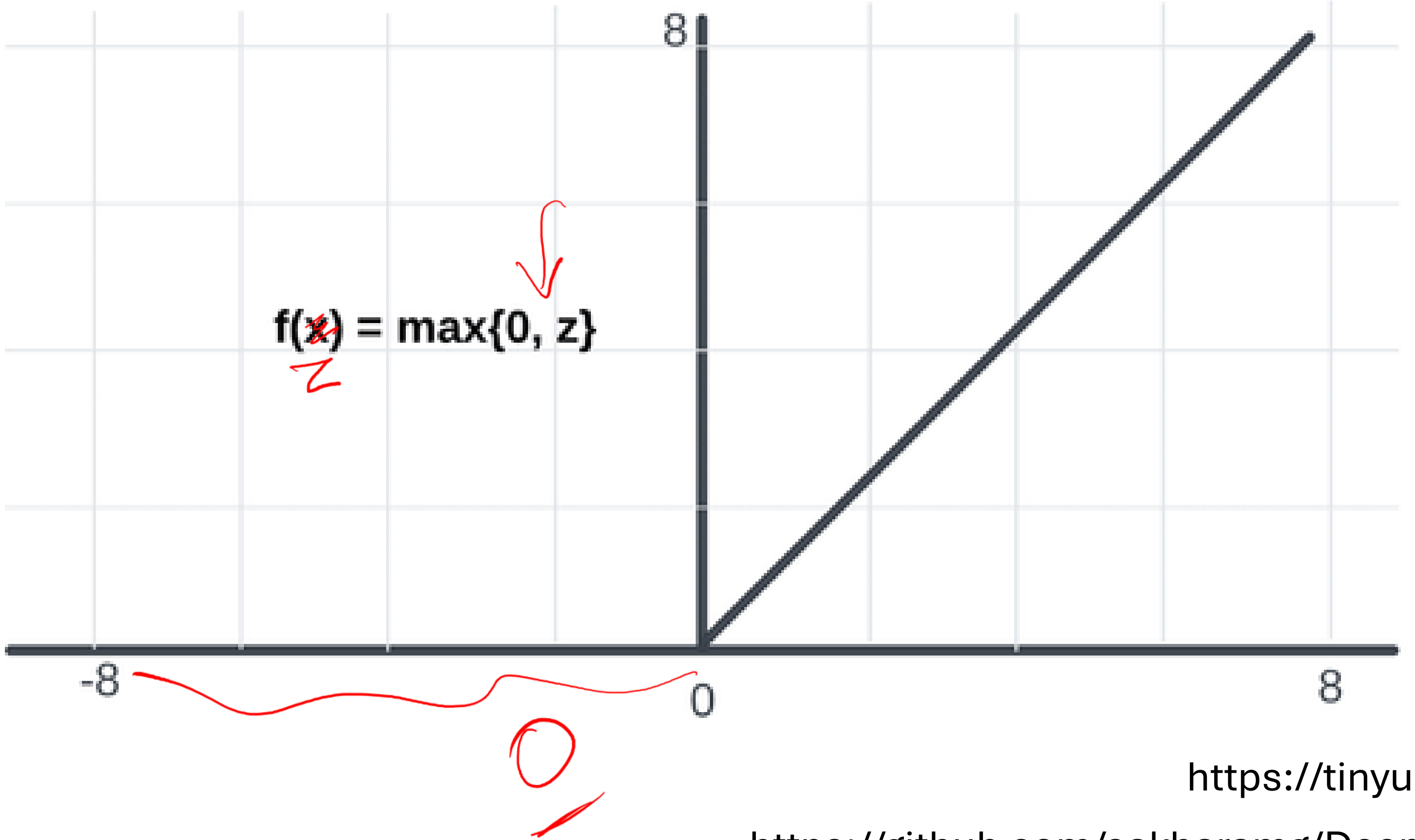
https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Towards Non Linearity – Handling Bends

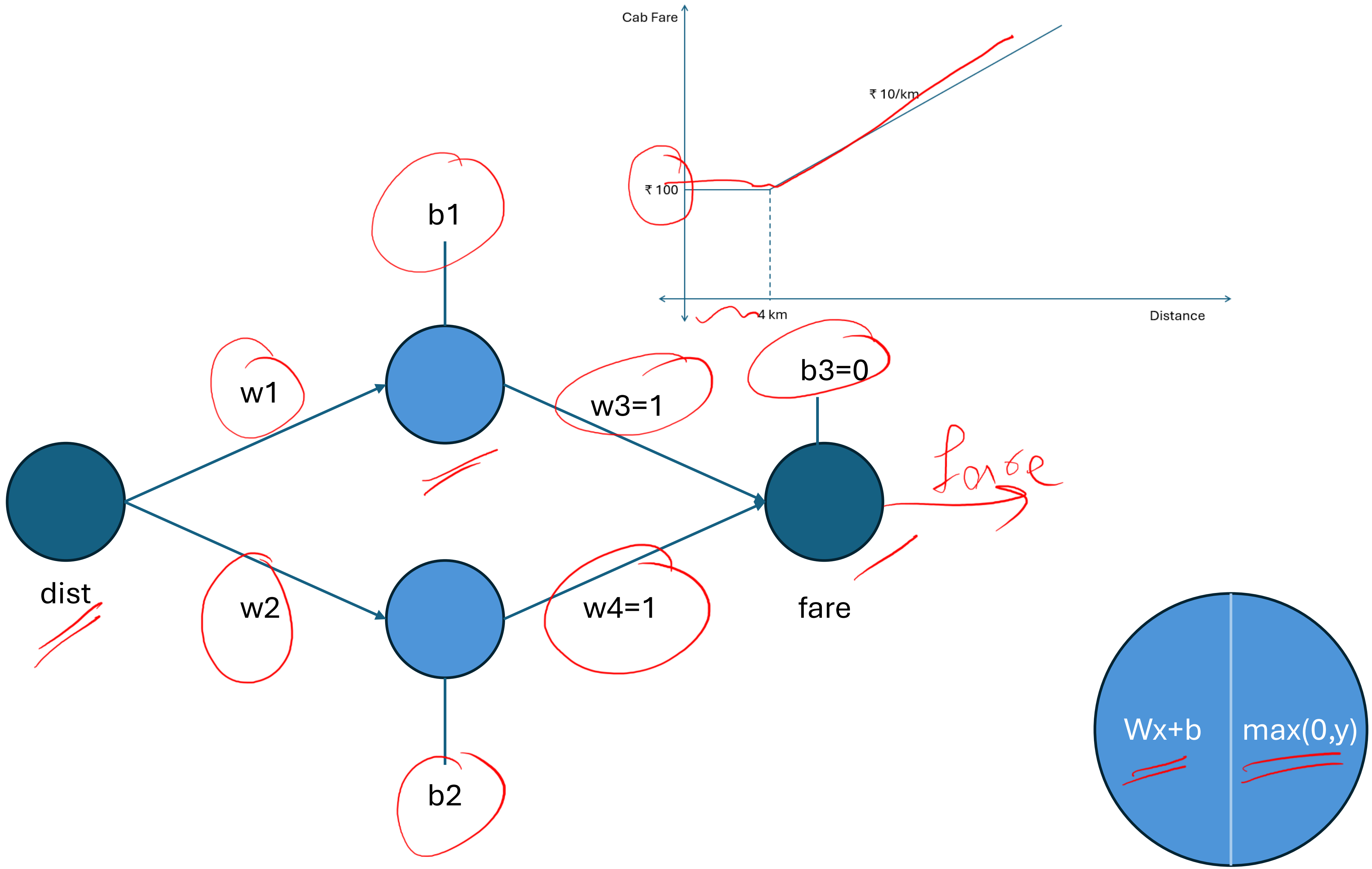


ReLU

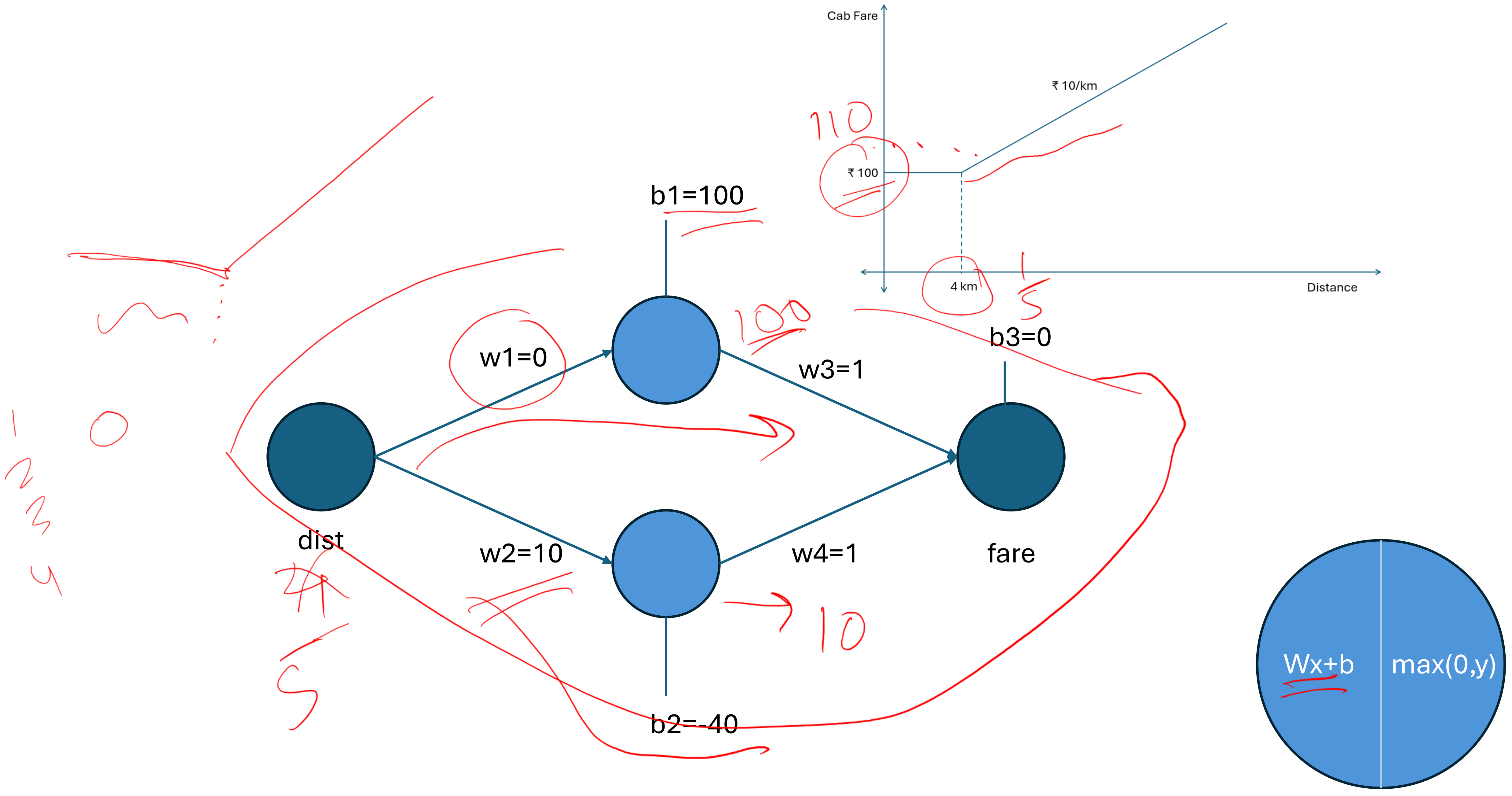


https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks



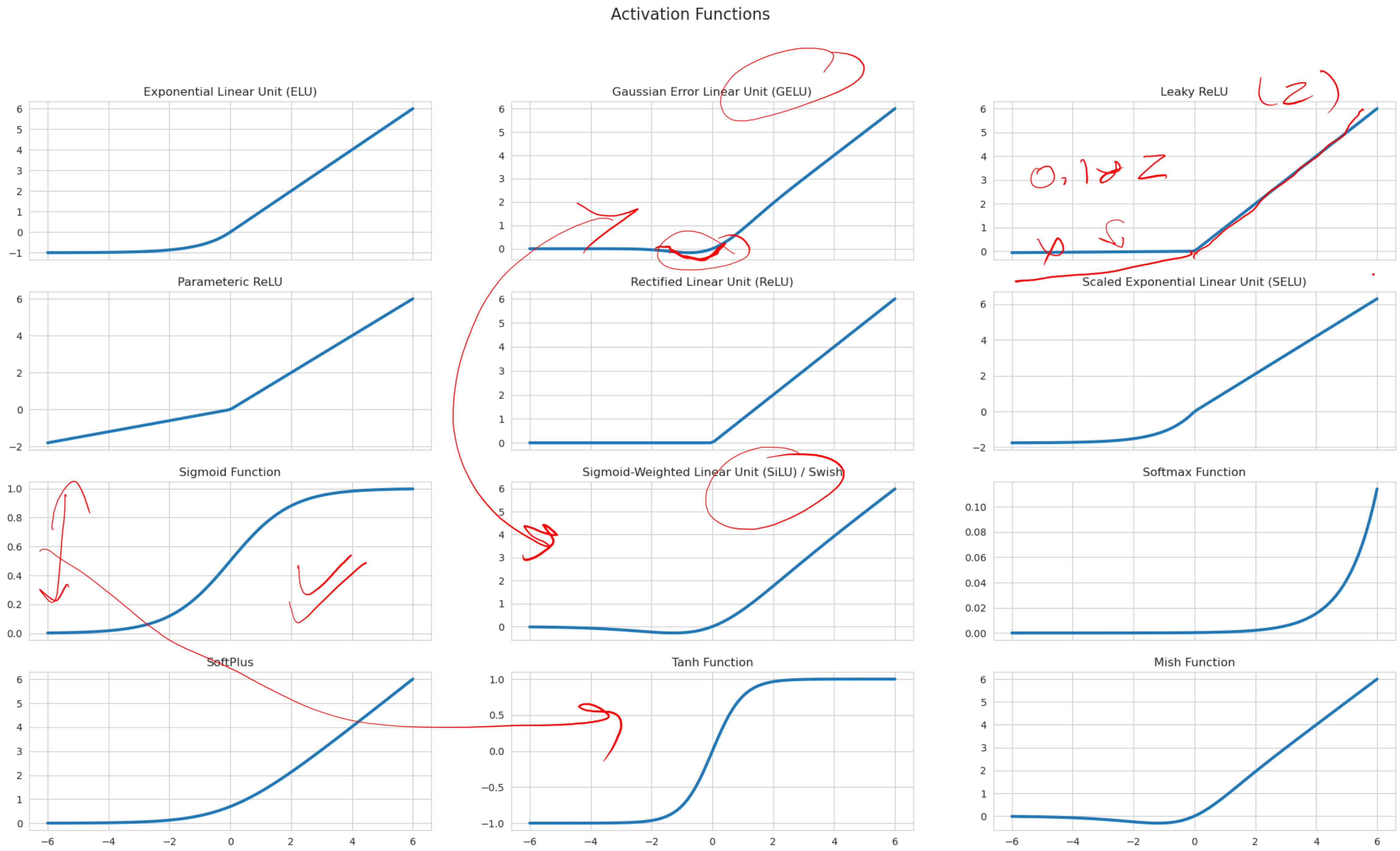
Cab Fare
₹ 100
₹ 10/km
4 km
Distance
b1
w1
w3=1
b3=0
dist
w2
w4=1
fare
fare
b2
Wx+b
max(0,y)



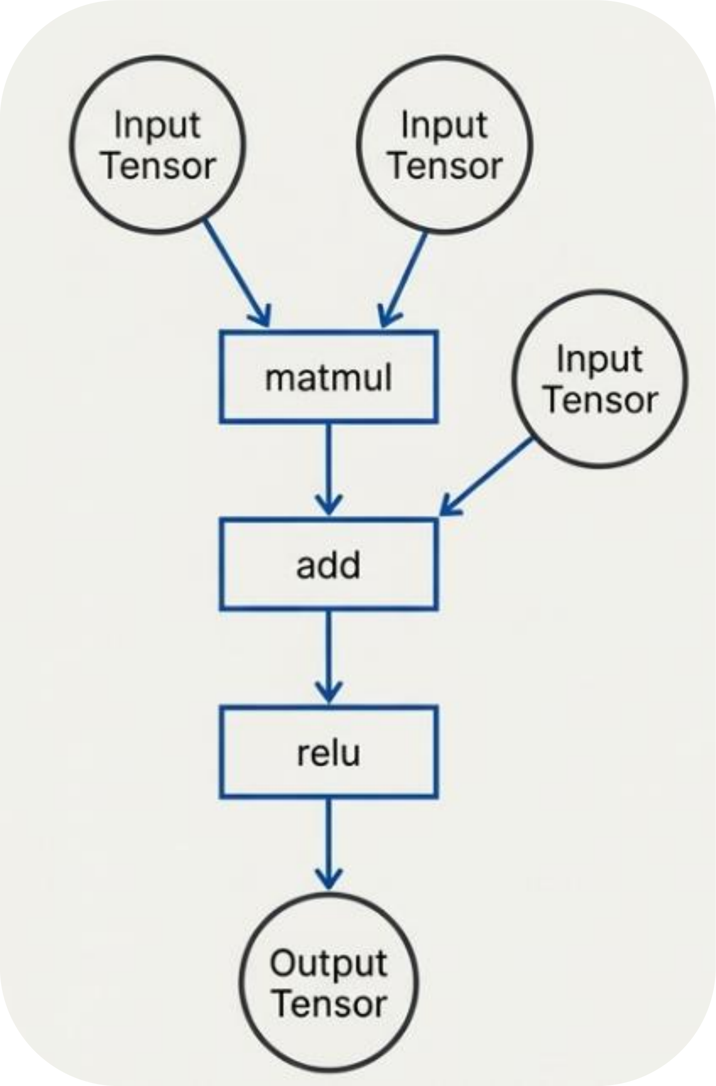
Cab Fare
₹ 10/km
₹ 100
4 km
Distance
b1=100
w1=0
w3=1
b3=0
dist
w2=10
w4=1
fare
b2=-40
Wx+b
max(0,y)
110
100
10
1
0
2
3
4
5

Classification

https://playground.tensorflow.org/

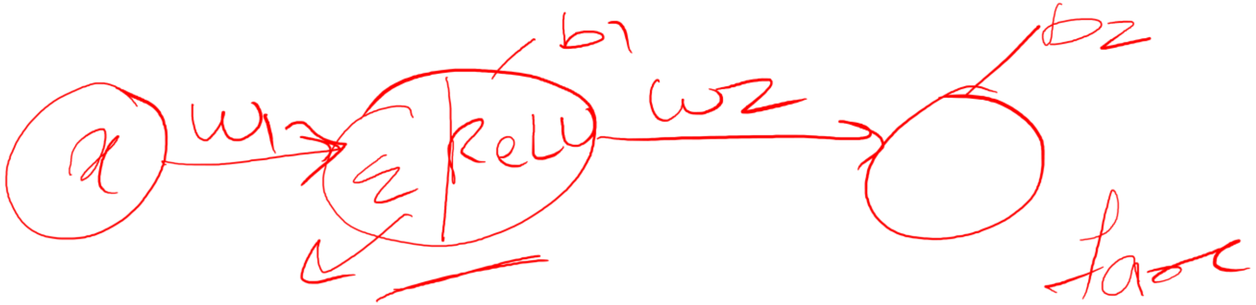

Activation Functions
Exponential Linear Unit (ELU)
Gaussian Error Linear Unit (GELU)
Leaky ReLU
Parameteric ReLU
Rectified Linear Unit (ReLU)
Scaled Exponential Linear Unit (SELU)
Sigmoid Function
Sigmoid-Weighted Linear Unit (SiLU) / Swish
Softmax Function
SoftPlus
Tanh Function
Mish Function

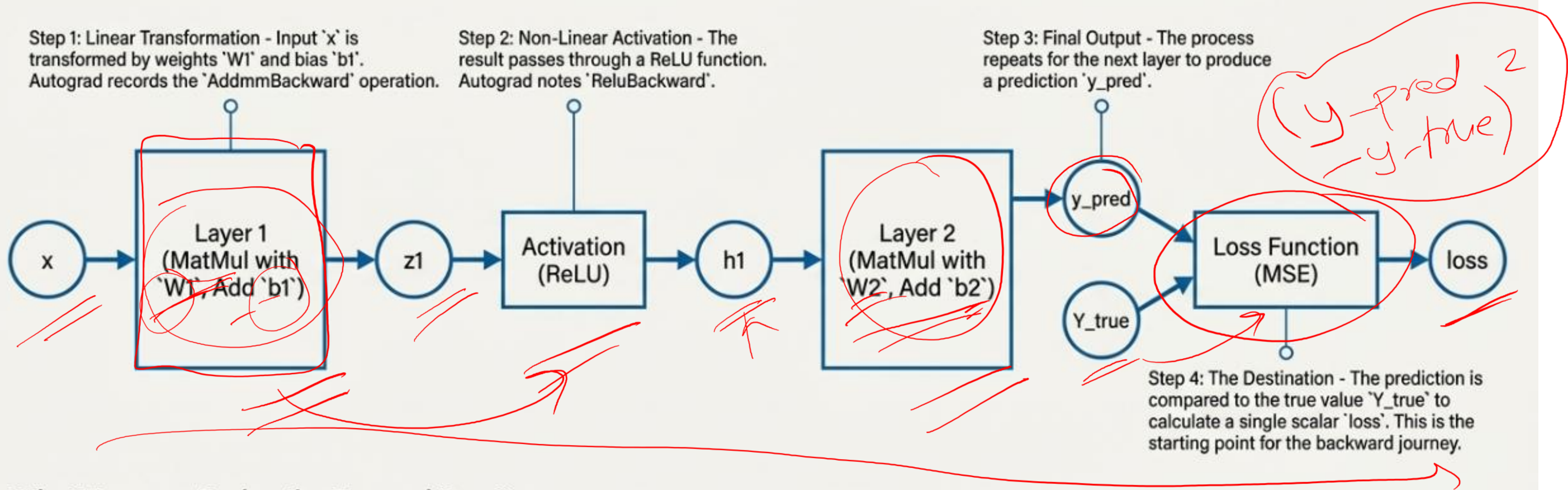
Recap: Computation Graph



- **Eager Execution**: ops run immediately (no "compile step")
- **Debugging = inspection**: print **shape / dtype / device** early
- **Core Primitives**: **Tensor + Autograd** → everything else builds on this

Forward Pass





What Happens During the Forward Pass?

The forward pass executes the model's operations sequentially to compute an output. Crucially, as this happens, PyTorch builds the dynamic computation graph in the background, linking each output tensor to its creating function via `grad_fn`. No gradients are calculated yet, but the 'map' is now complete.

loss.backward()

The Signal is Born

After the forward pass, we have a single scalar value: the loss. The entire goal of training is to minimize this value.

To do this, we need to know how each parameter (W, b) affects the loss. This is the gradient.

The chain rule allows us to compute this:

If $loss = f(y)$ and $y = g(W)$, then the gradient of the loss with respect to the weight W is:

$$\frac{\partial loss}{\partial W} = \frac{\partial loss}{\partial y} * \frac{\partial y}{\partial W}.$$

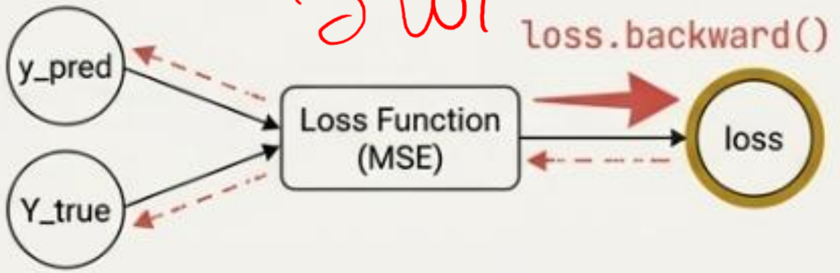
Think of it as assigning blame. The chain rule mathematically distributes the total error back through each operation that contributed to it.

The Trigger: `loss.backward()`

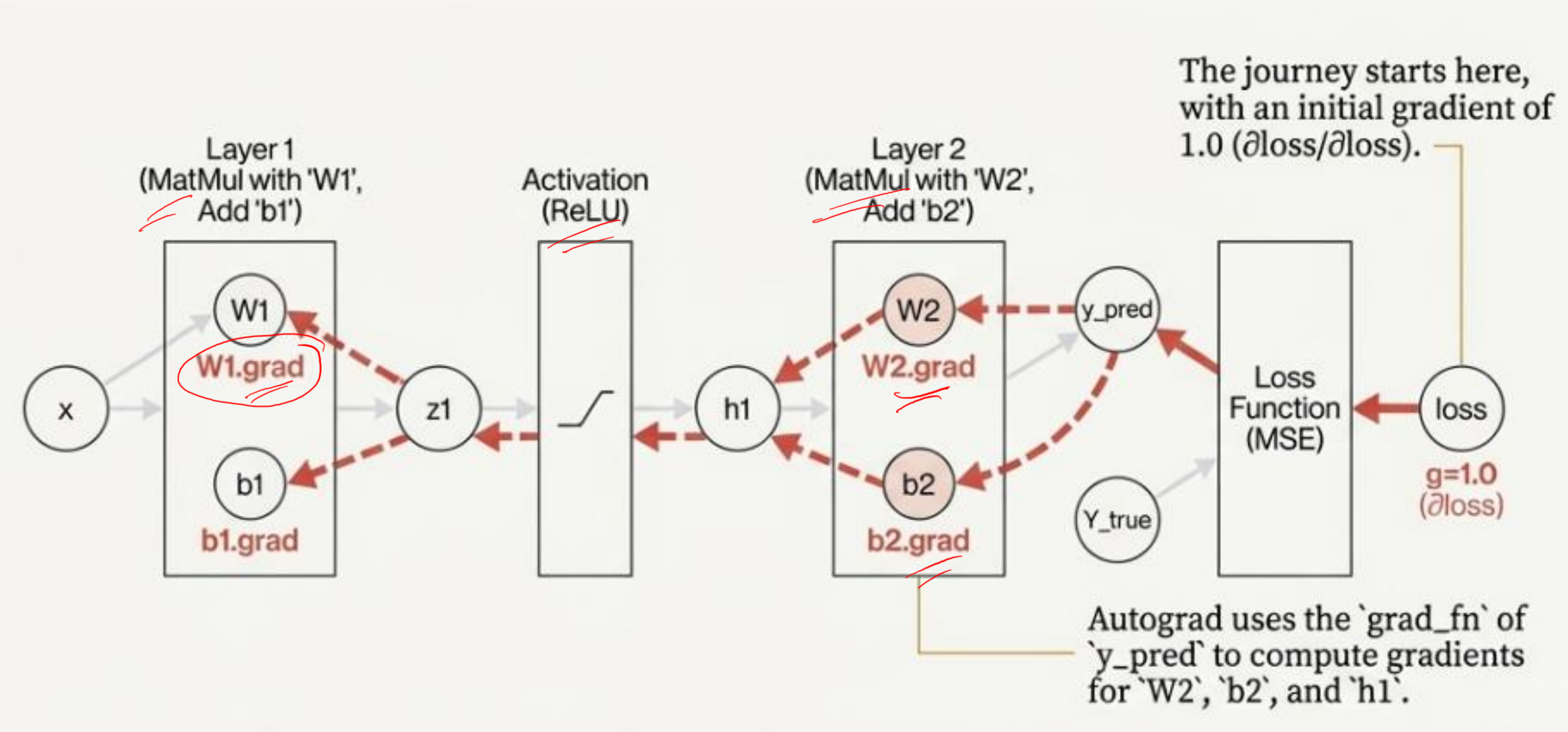
In PyTorch, the entire backpropagation process is triggered by a single command on the final loss tensor.

```
# After computing the final loss...
loss = F.mse_loss(y_pred, Y_true)

# This one line starts the journey home for the gradients.
loss.backward()

# The gradients are now stored in the .grad attribute
# of each parameter.
print(W1.grad)
```

$\frac{\partial L}{\partial W_1}$

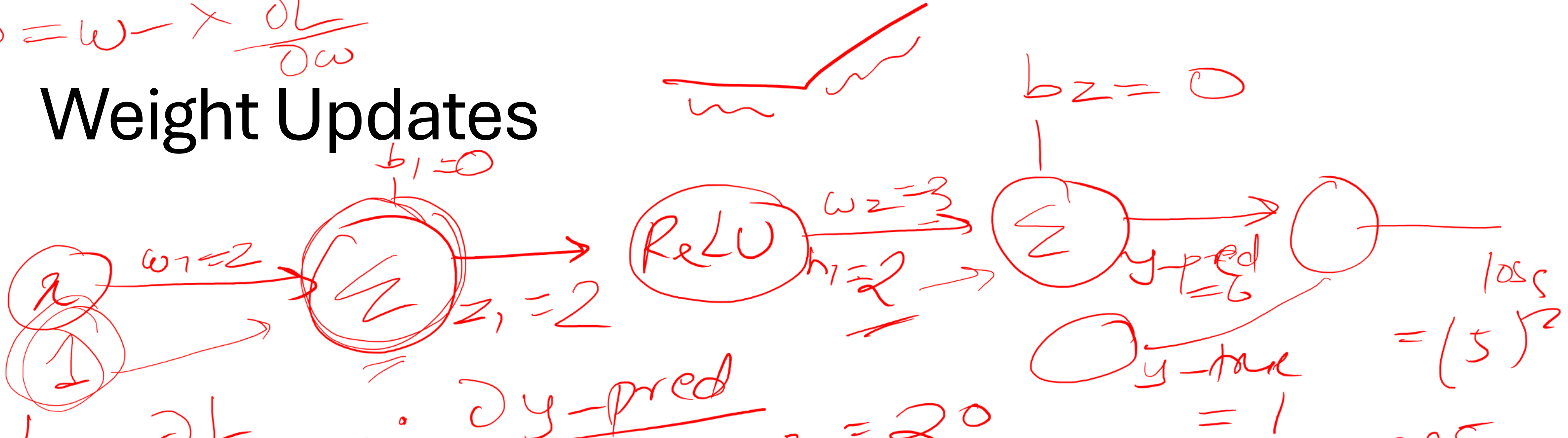


Back Propagation



$w = w - \alpha \frac{\partial L}{\partial w}$

Weight Updates



$$\frac{\partial L}{\partial w_2} = \underbrace{\frac{\partial L}{\partial y\text{-pred}}}_{=10} \cdot \underbrace{\frac{\partial y\text{-pred}}{\partial w_2}}_{=2} = 20$$

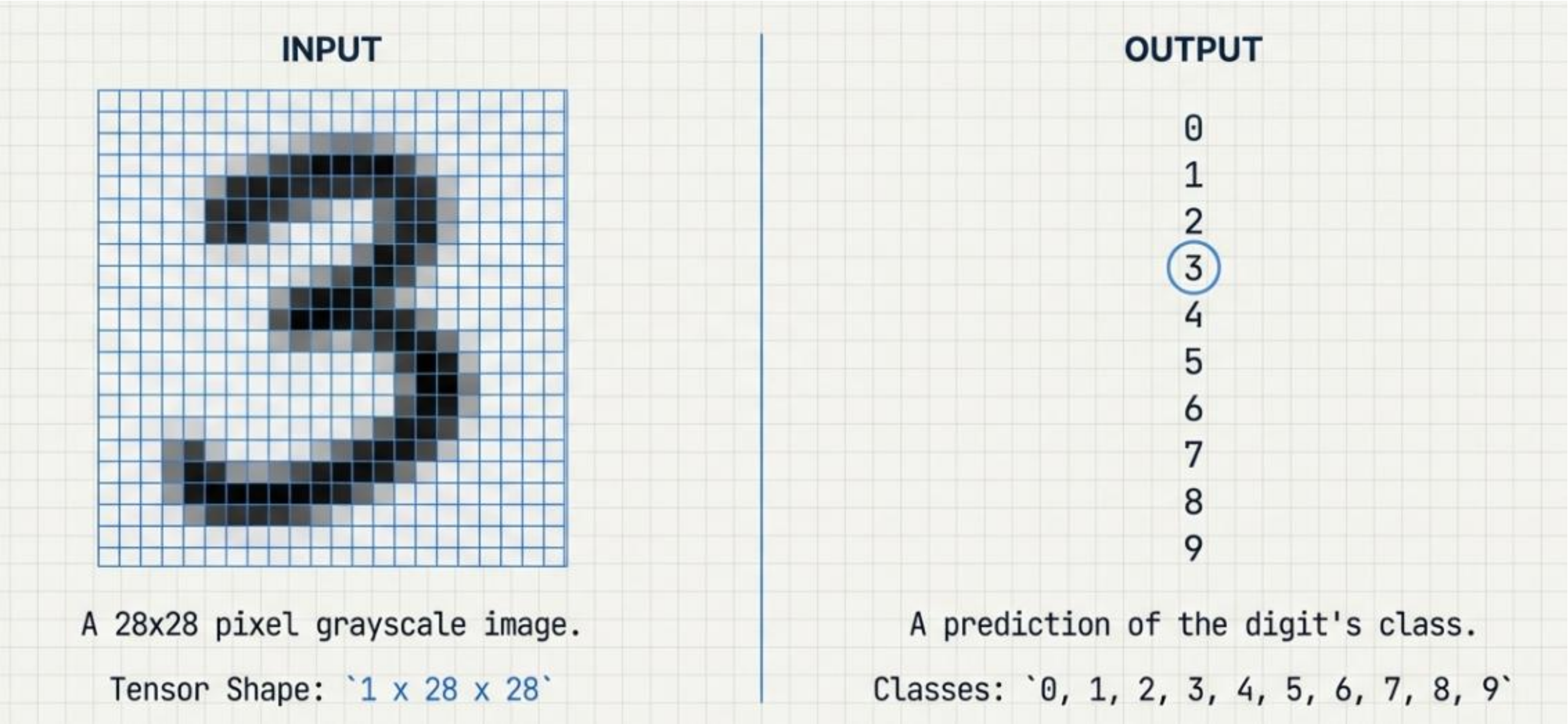
$$\frac{\partial L}{\partial b_2} = \underbrace{\frac{\partial L}{\partial y\text{-pred}}}_{10} \cdot \underbrace{\frac{\partial y\text{-pred}}{\partial b_2}}_{1} = 10$$

$$\frac{\partial L}{\partial w_1} = \underbrace{\frac{\partial L}{\partial y\text{-pred}}}_{=10} \cdot \underbrace{\frac{\partial y\text{-pred}}{\partial h_1}}_{=3} \cdot \underbrace{\frac{\partial h_1}{\partial z_1}}_{=1} \cdot \underbrace{\frac{\partial z_1}{\partial w_1}}_{2} = \cancel{60}$$

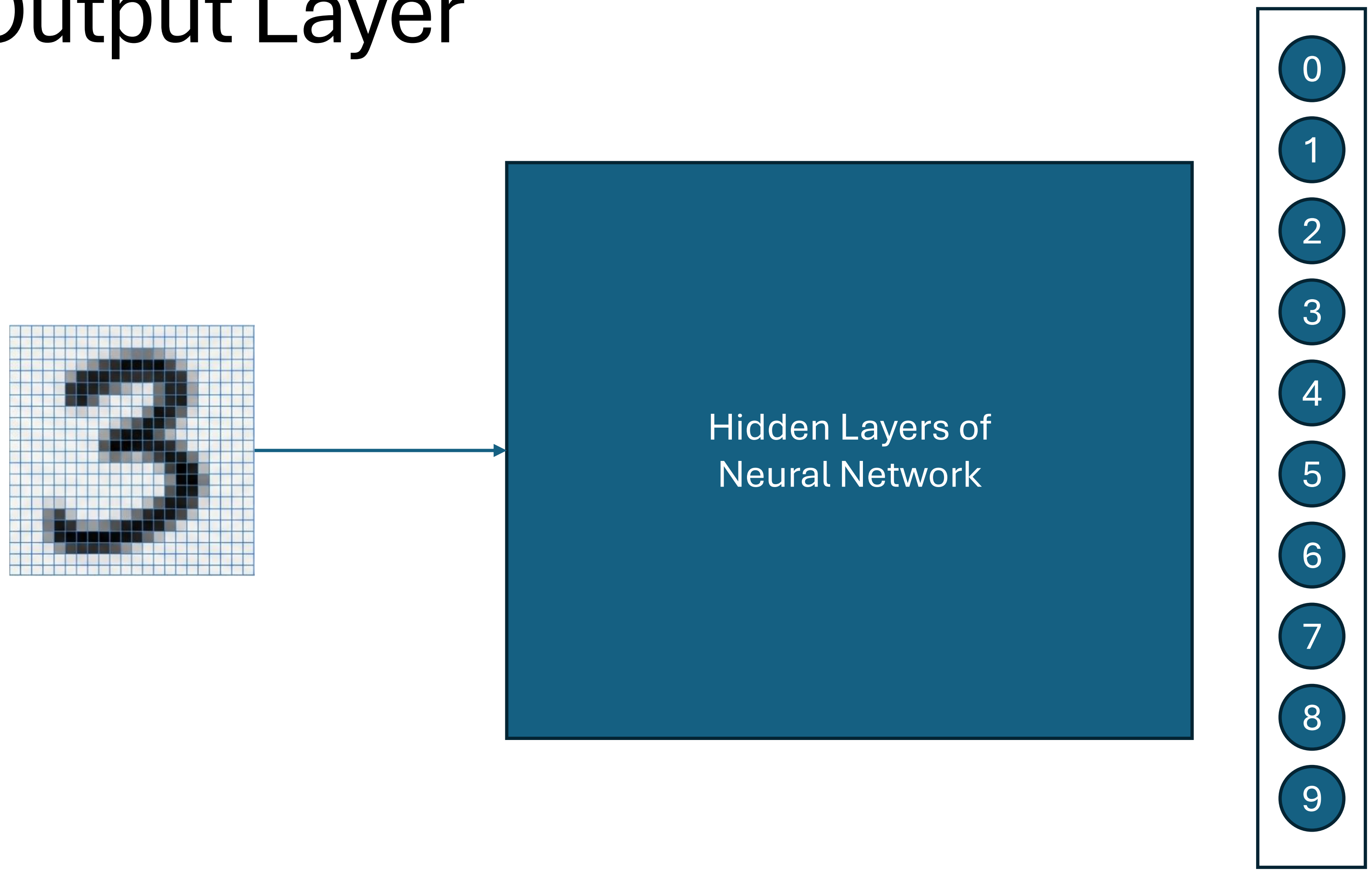
$$30$$

Lab

MNIST


INPUT
OUTPUT
0
1
2
3
4
5
6
7
8
9
A 28x28 pixel grayscale image.
Tensor Shape: `1 x 28 x 28`
A prediction of the digit's class.
Classes: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`

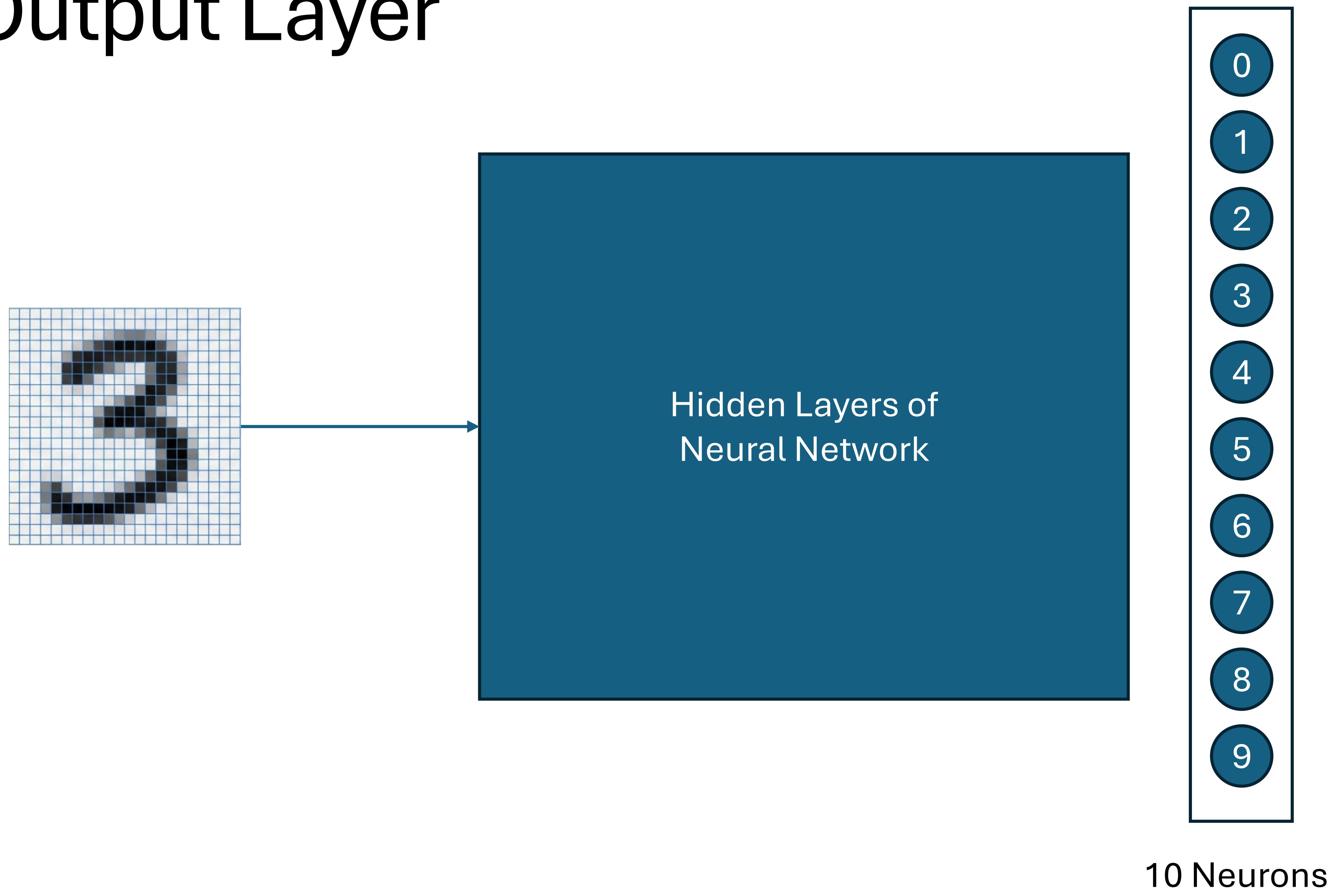
Output Layer



https://tinyurl.com/dlframeworks

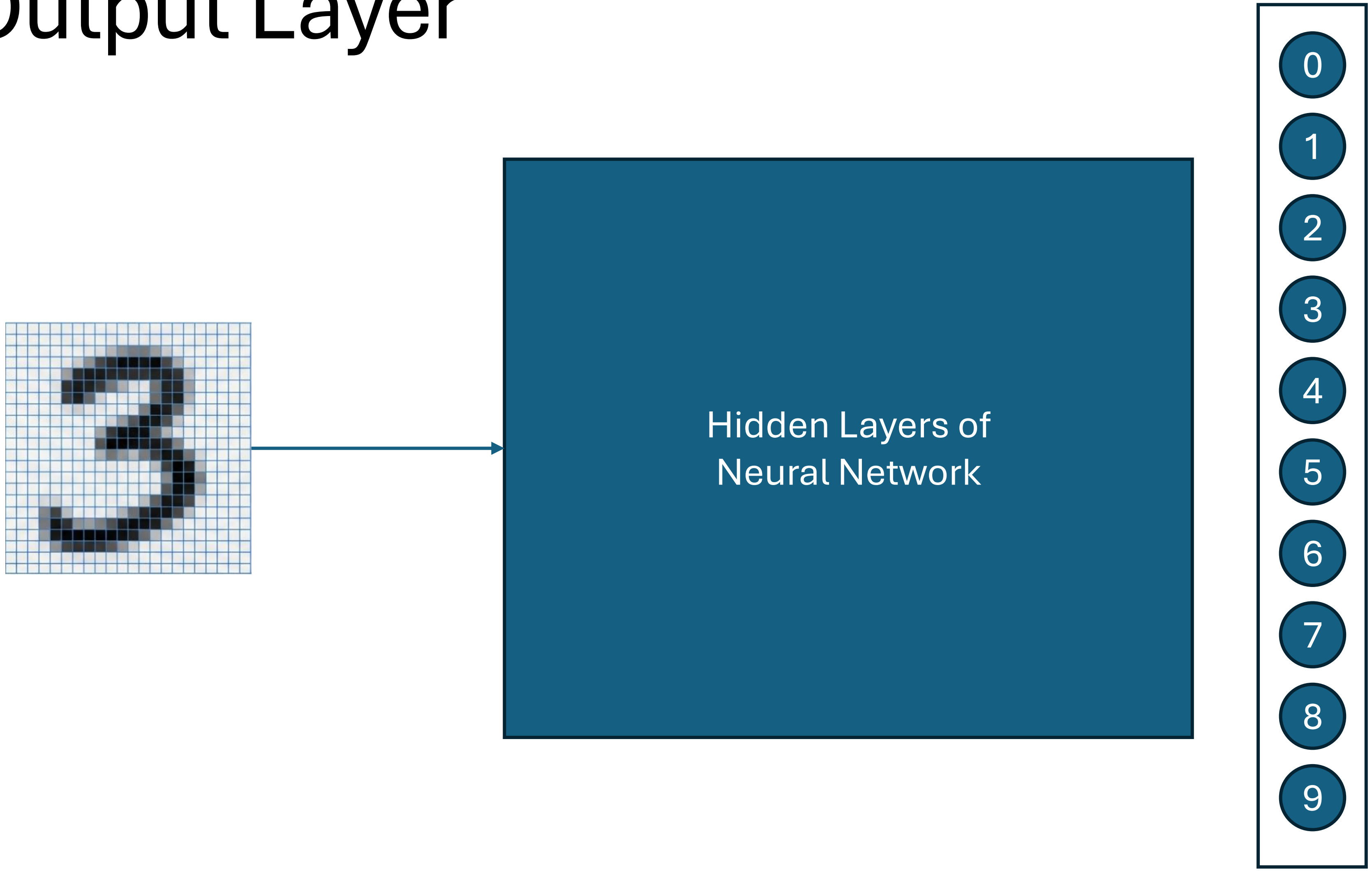
https://github.com/sakharamg/DeepLearningFrameworks

Output Layer



https://github.com/sakharamg/DeepLearningFrameworks

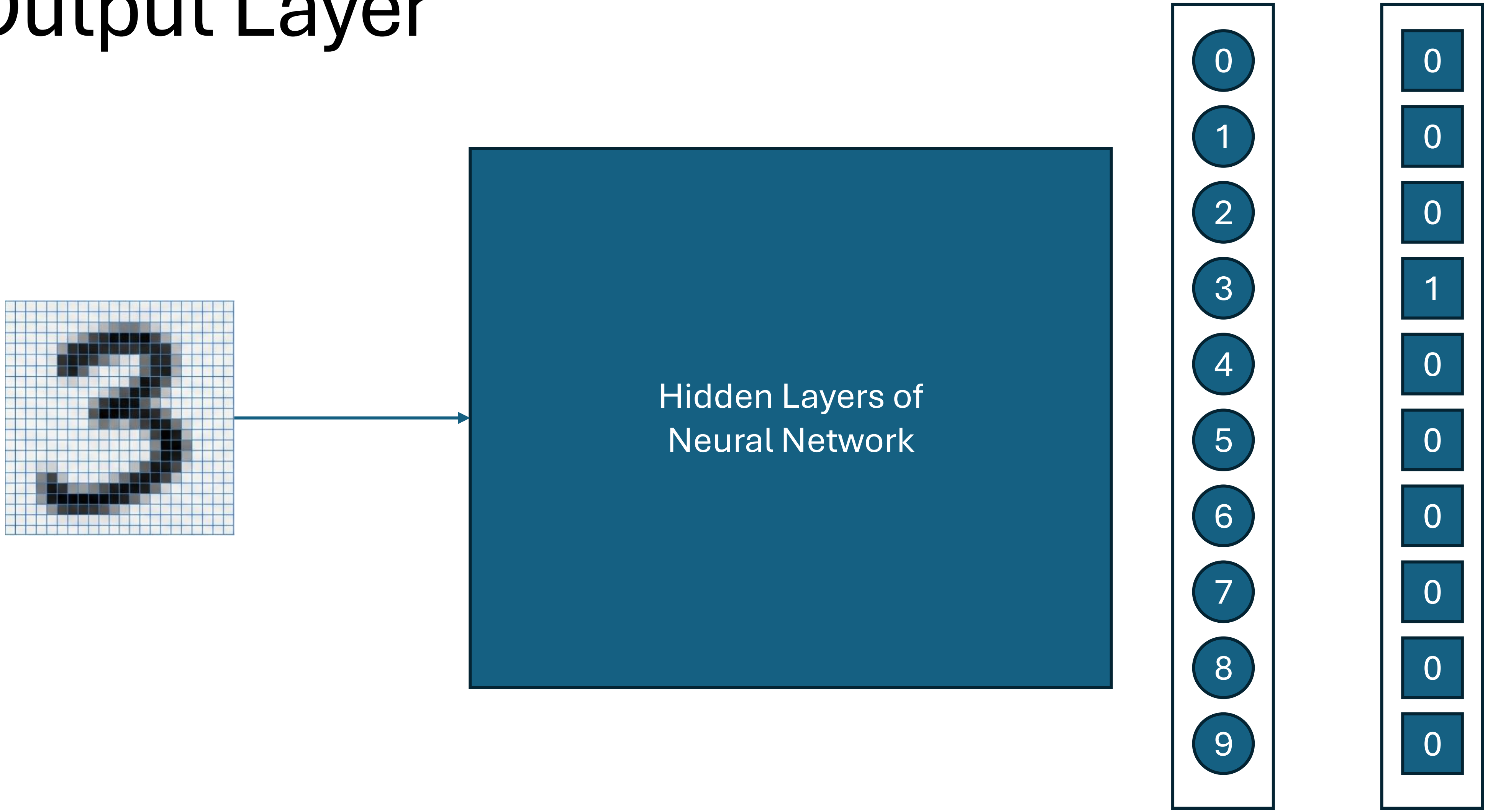
Output Layer



nn.Linear(__ , 10)

https://github.com/sakharamg/DeepLearningFrameworks

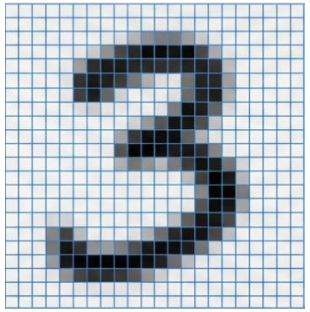
Output Layer



nn.Linear(__ , 10)

https://github.com/sakharamg/DeepLearningFrameworks

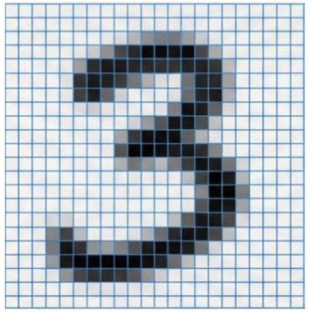
Softmax



https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Softmax



Output Layer

-0.7

1.3

-3.4

9.7

0.9

-2.3

0.1

1.8

4.1

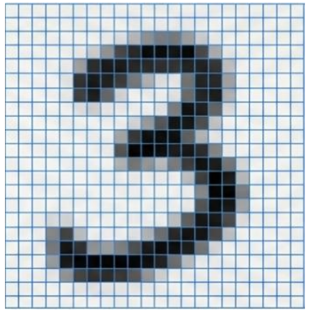
-1.2

Logits

https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Softmax



Output Layer

-0.7

1.3

-3.4

9.7

0.9

-2.3

0.1

1.8

4.1

-1.2

Logits

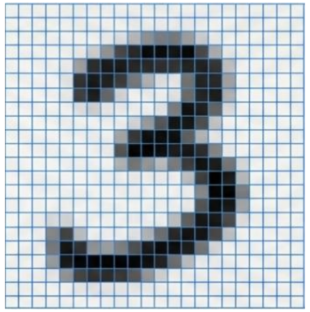
Softmax
activation function

$$\boxed{\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}}$$

https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Softmax



Output Layer

$$\begin{pmatrix} -0.7 \\ 1.3 \\ -3.4 \\ \mathbf{9.7} \\ 0.9 \\ -2.3 \\ 0.1 \\ 1.8 \\ 4.1 \\ -1.2 \end{pmatrix}$$

Logits

Softmax
activation function

$$\boxed{\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}}$$

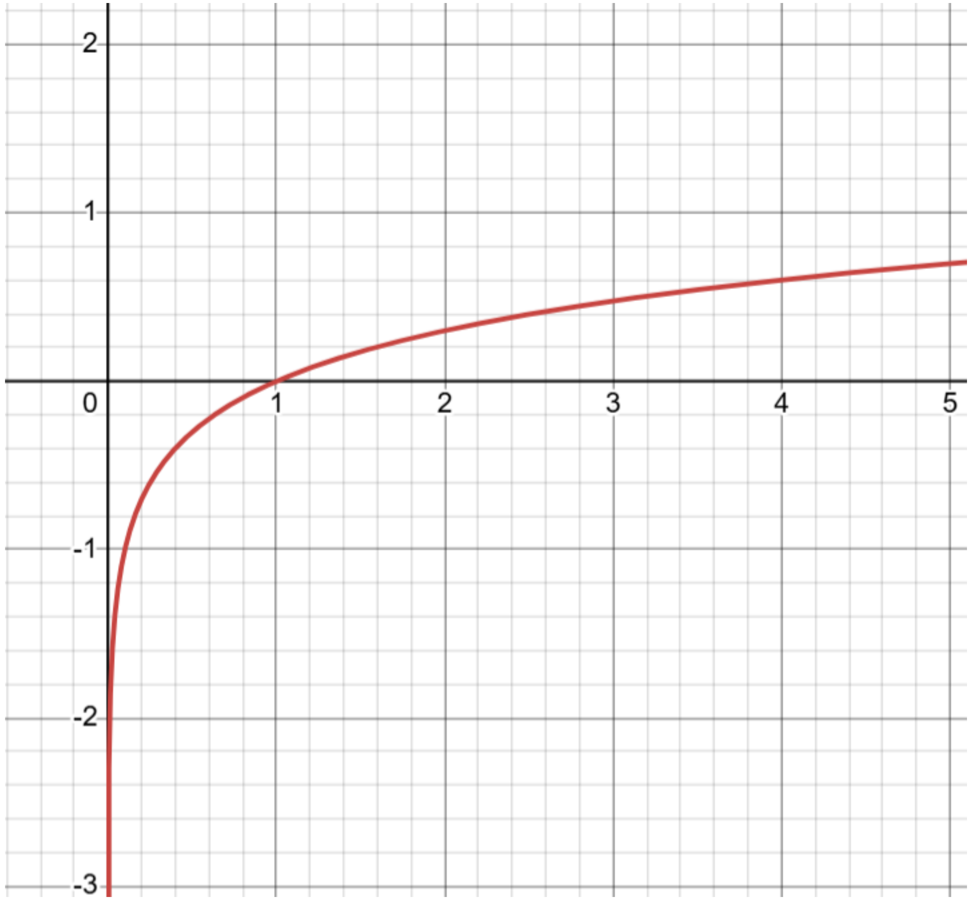
$$\begin{pmatrix} 0.0000302941 \\ 0.0002238446 \\ 0.0000020359 \\ 0.9954517644 \\ 0.0001500475 \\ 0.0000061163 \\ 0.0000674207 \\ 0.0003690573 \\ 0.0036810450 \\ 0.0000183743 \end{pmatrix}$$

Negative Log Likelihood

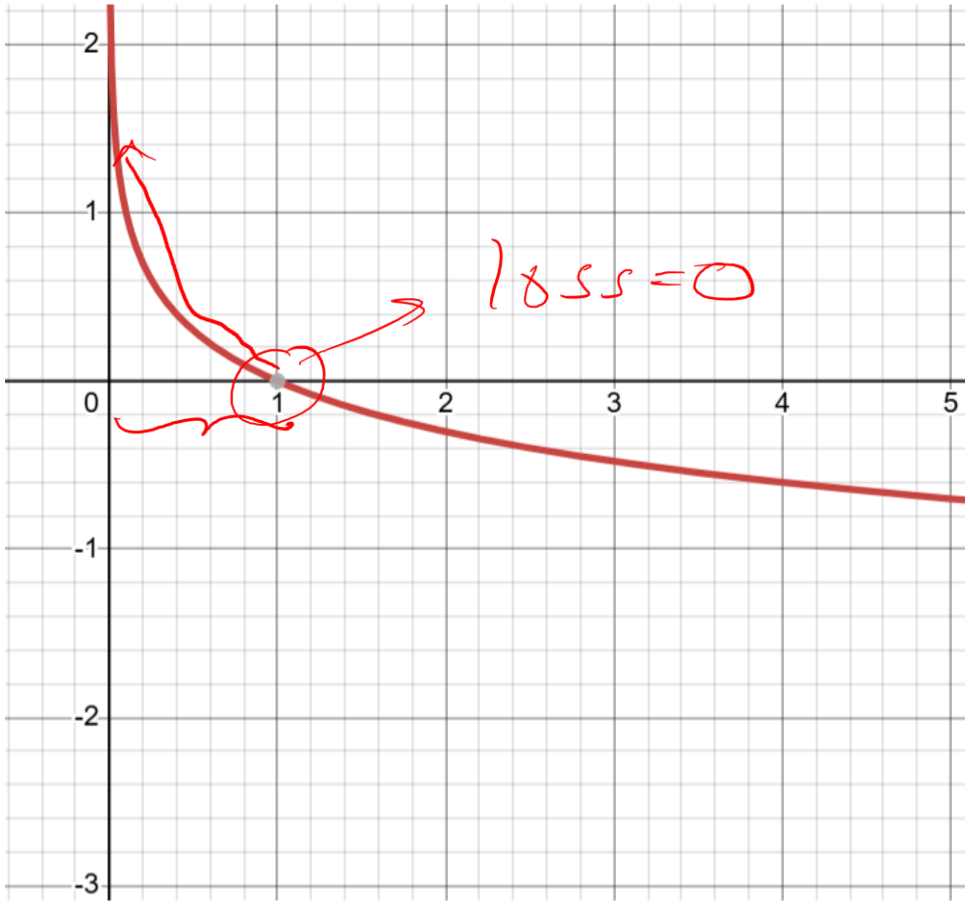
$$\begin{pmatrix} 0.0000302941 \\ 0.0002238446 \\ 0.0000020359 \\ 0.9954517644 \\ 0.0001500475 \\ 0.0000061163 \\ 0.0000674207 \\ 0.0003690573 \\ 0.0036810450 \\ 0.0000183743 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- log (0.99)

Negative Log Likelihood

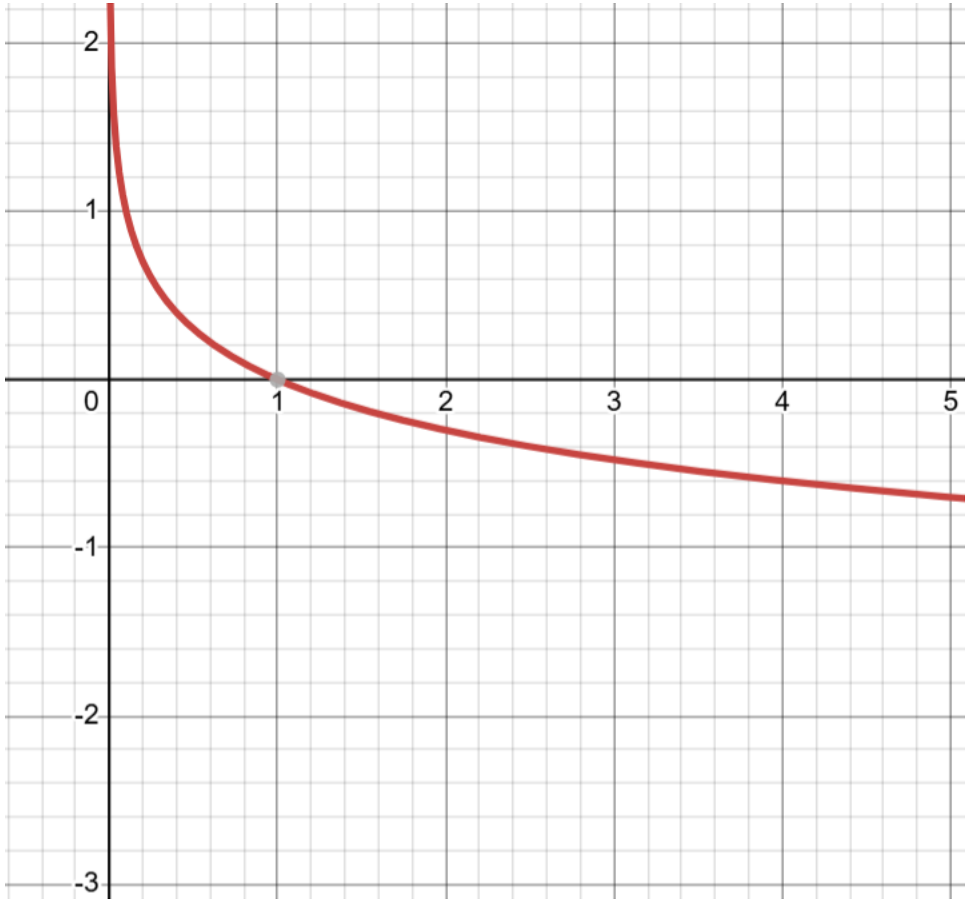


log(x)



-log(x)

Negative Log Likelihood



-log(x)

Lesser Confidence -> More Penality

Cross Entropy Loss

logsumexp

loop

e^{z_j}

1. P = log_softmax(logits) ... Supress Probablities
2. NLL(log_P) = -log(P_{true_class})

Softmax activation function

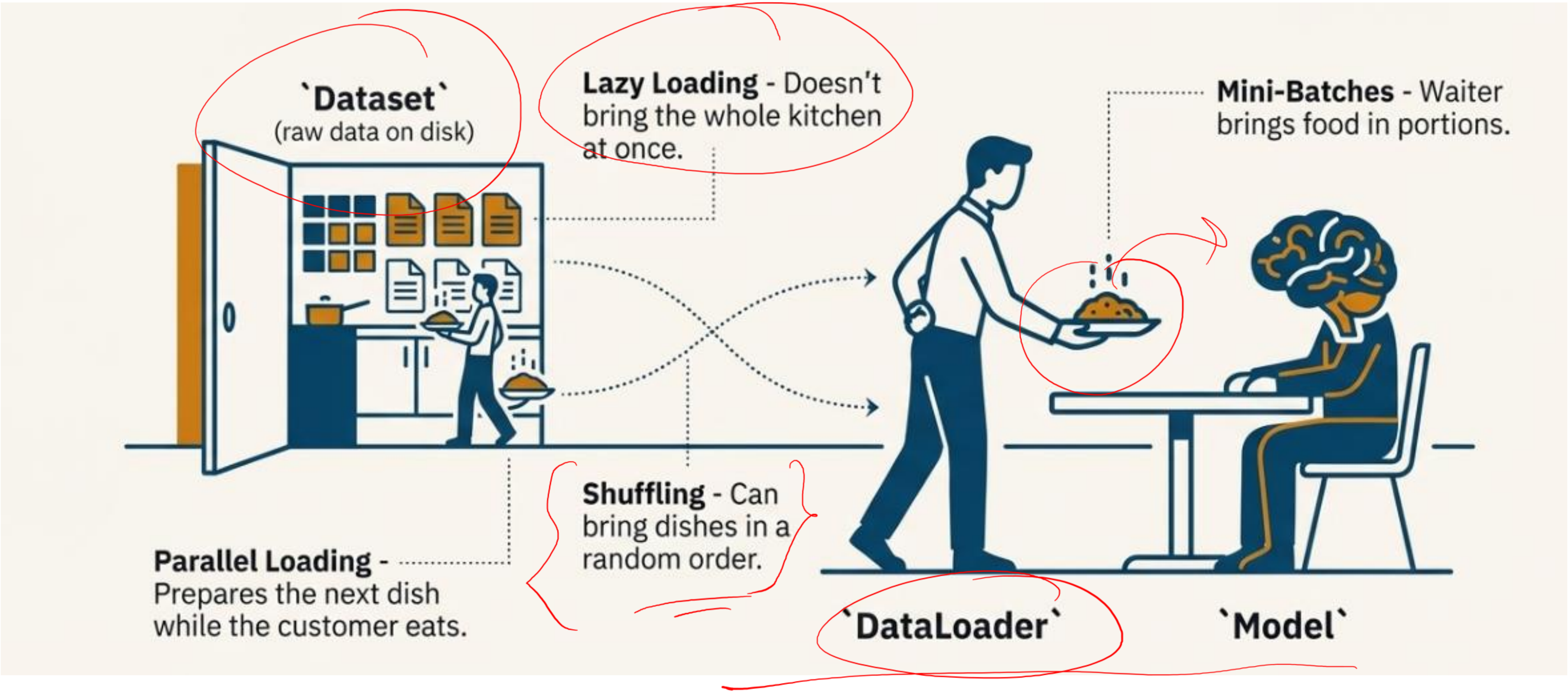
log $\dfrac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

$$\log \operatorname{softmax}(z)_k = z_k - \log \sum_j e^{z_j}$$

- if logits are large (e.g., 80, 100), e^z can overflow to inf
- if logits are very negative (e.g., -80), e^z underflows to 0
- then log(softmax) becomes log(0) or log(inf) → -inf/inf, NaNs, unstable gradients

https://gregorygundersen.com/blog/2020/02/09/log-sum-exp/

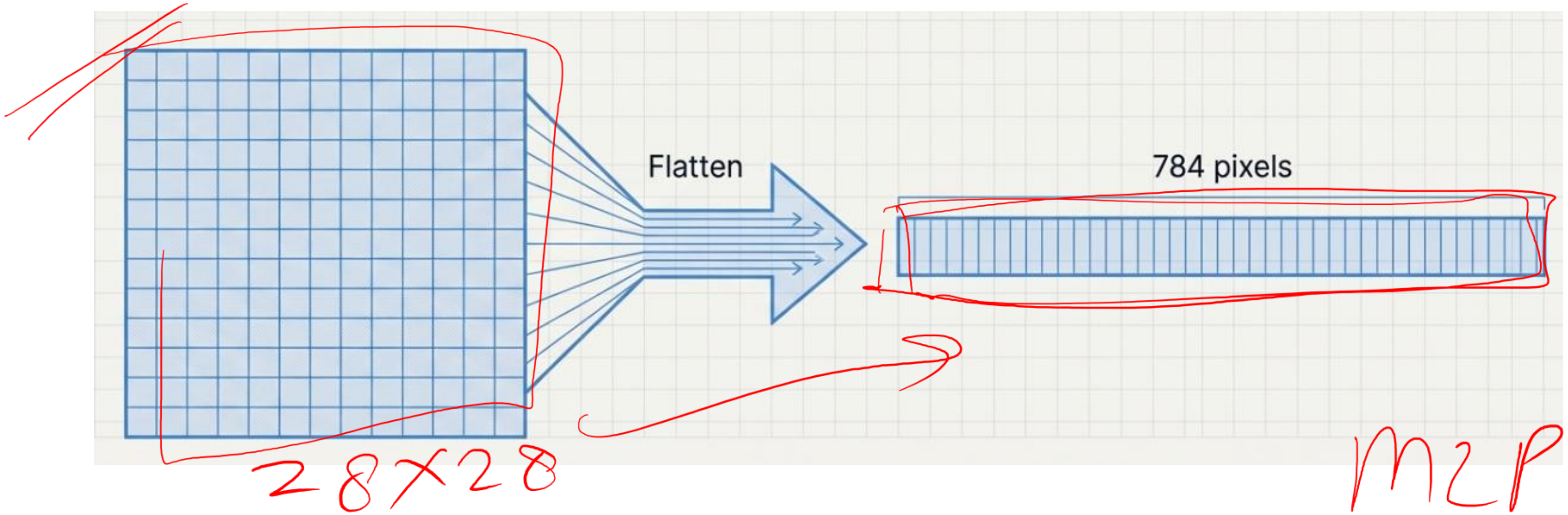
DataLoader



https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Flatten

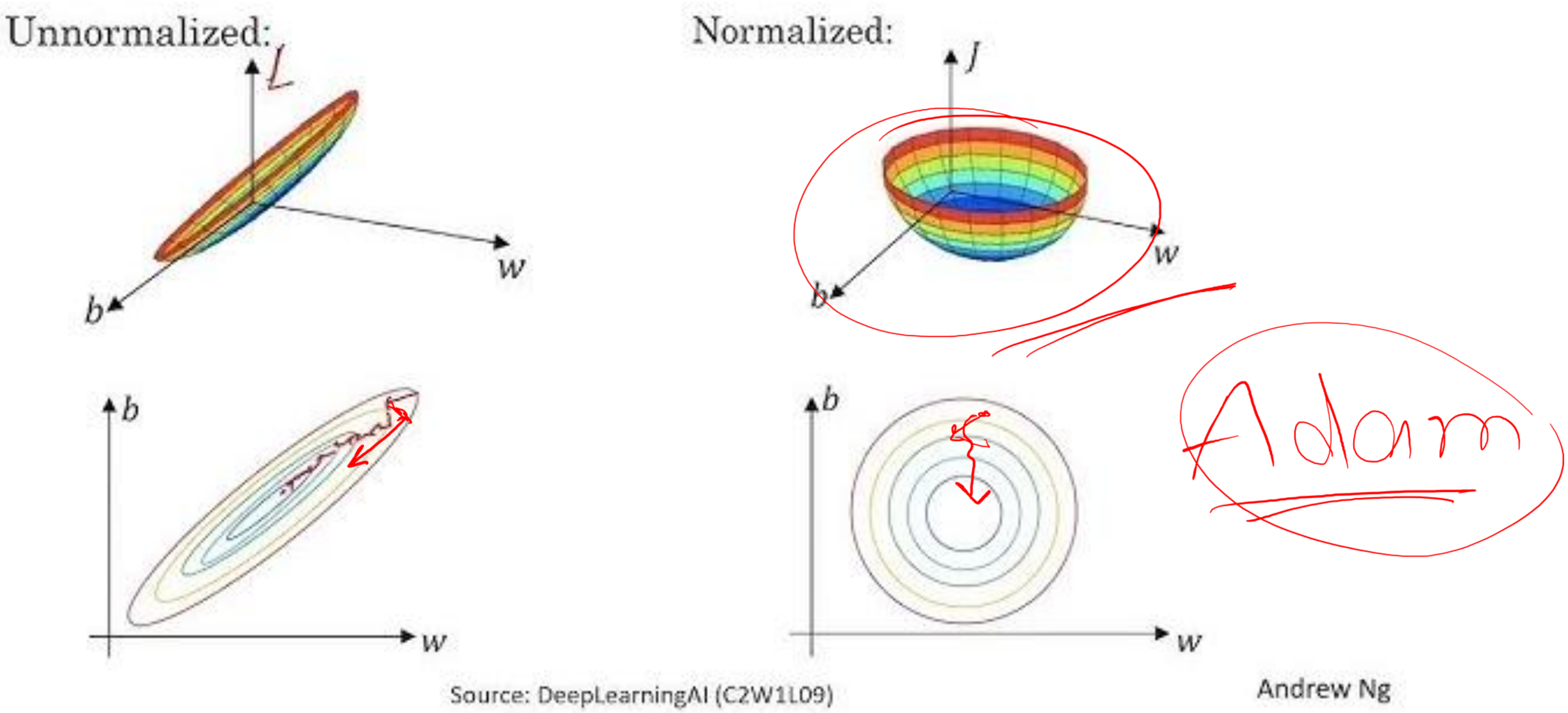


A 28x28 images contains 784 pixels.
MLP cannot process 2D spatial data directly. It requires a 1D vector as input.
The first step in our model's logic must be to **Flatten** the (N,1,28,28) image tensor into a (N,784) vector.

https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Normalize()

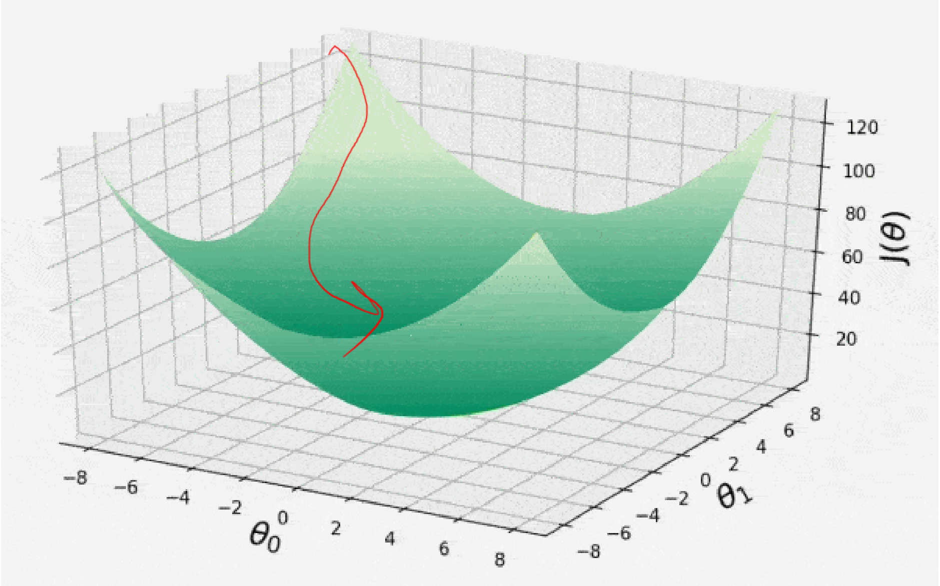


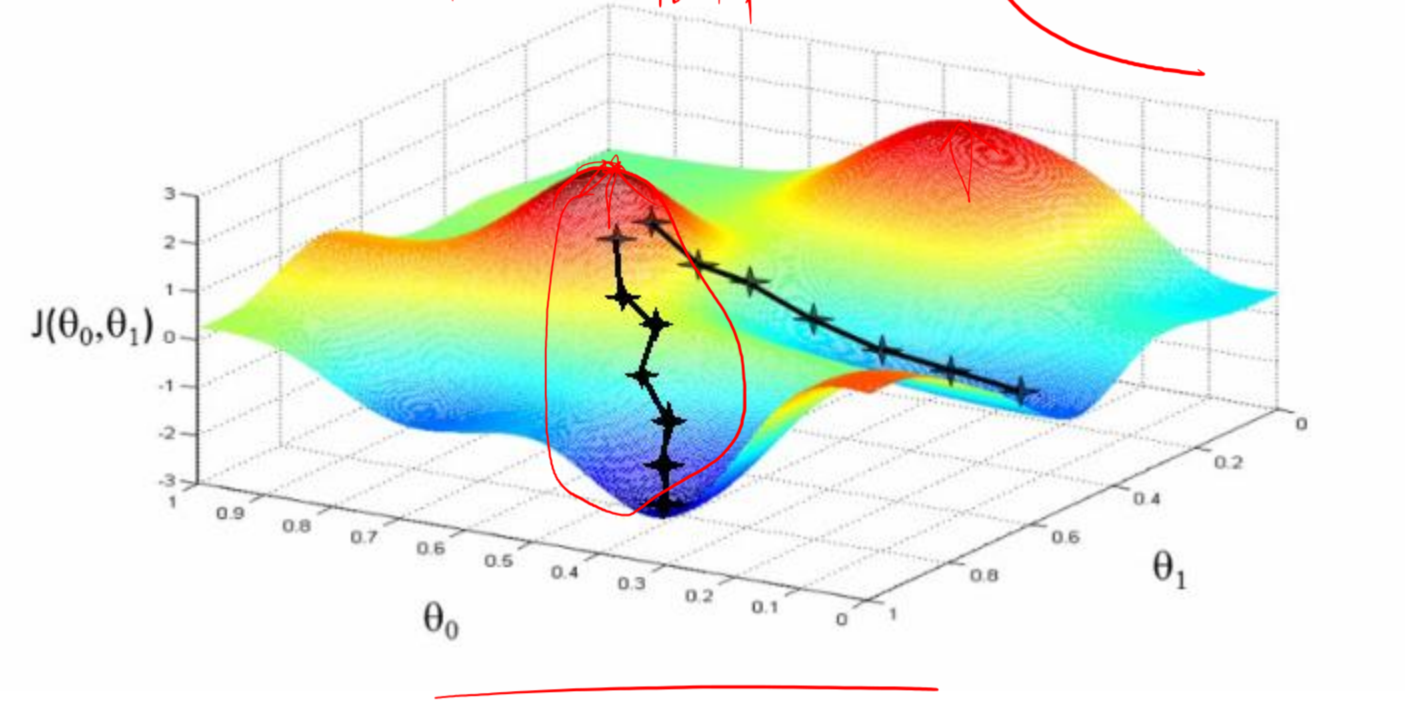
https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Gradient Descend - DL

local minima





https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Lab

https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Regularization: Weight Decay and Dropout

Weight Decay: L2 Regularization

Dropout: Drop fraction of neurons

Weight Decay

$$L_{total} = L_{data} + \lambda \sum w^2$$

Mechanism

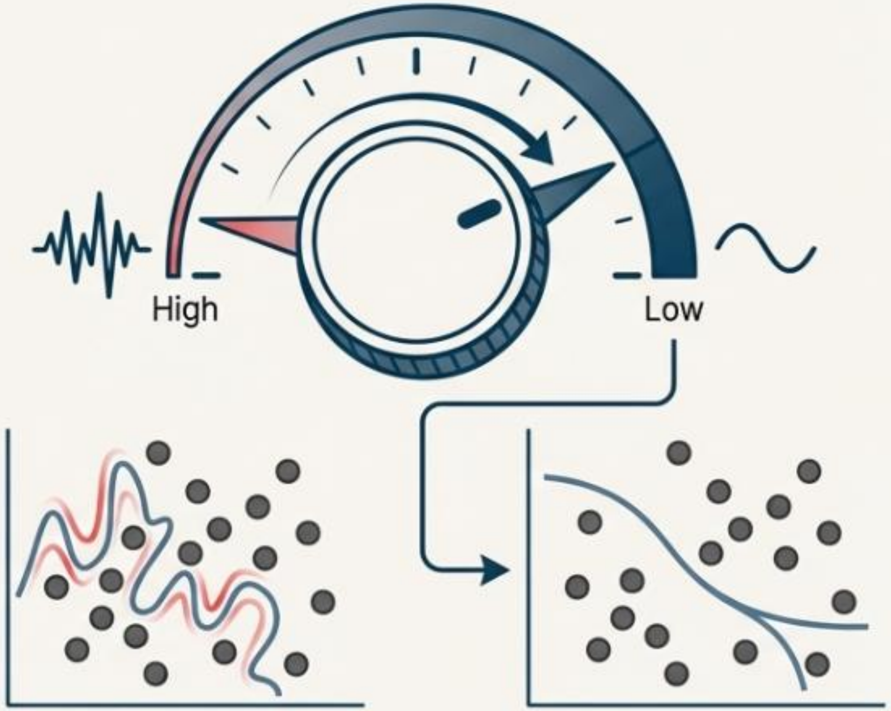
Adds a penalty term to the loss function proportional to the square of the weights ($\lambda \sum w^2$). During training, this continuously pulls weights towards zero, making large weights "expensive."

Problem Solved

Prevents overly sharp and sensitive models. Large weights mean small input changes can cause large output changes.

Intuition Analogy

Think of a microphone's gain. High gain is sensitive and amplifies noise. Lowering the gain produces a clearer, more stable signal.

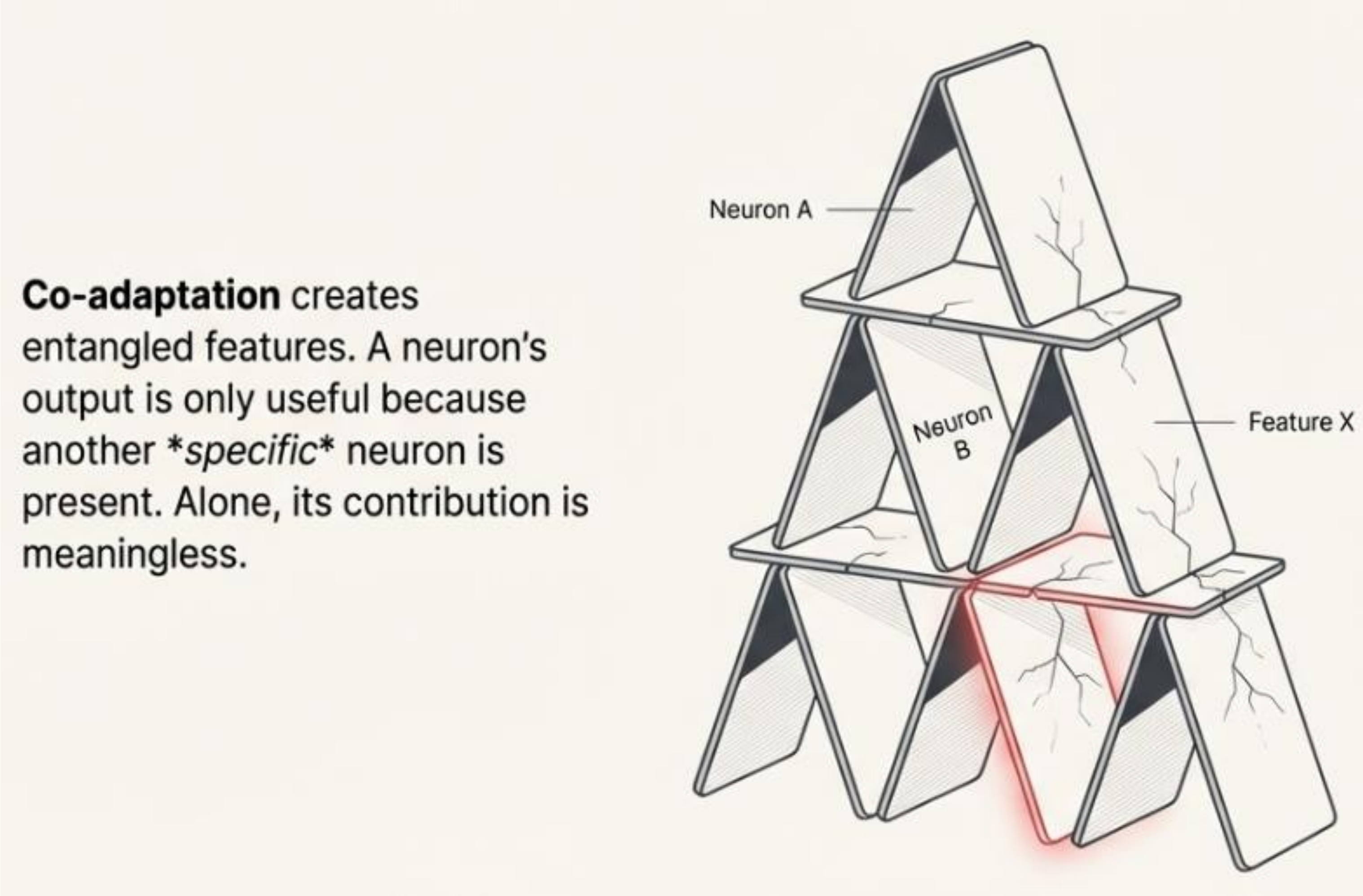


Before: High Sensitivity

After: Low Sensitivity

"Use many small contributions, not a few dominant ones."

Co-Adaptation



Am I a Siberian Husky?

https://tinyurl.com/dlframeworks

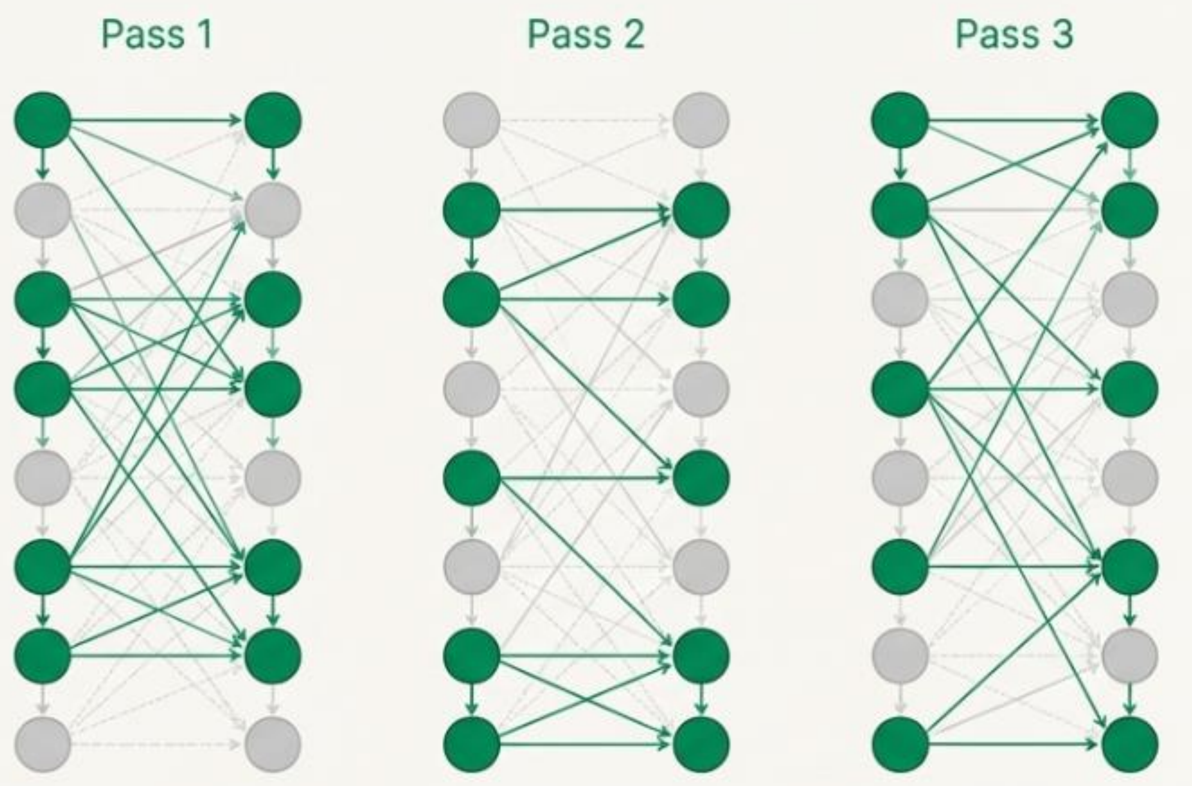
https://github.com/sakharamg/DeepLearningFrameworks

Dropout

Mechanism
During each training pass, randomly sets a fraction of neuron activations to zero. This forces the network to function even with missing components. The network changes with every batch.

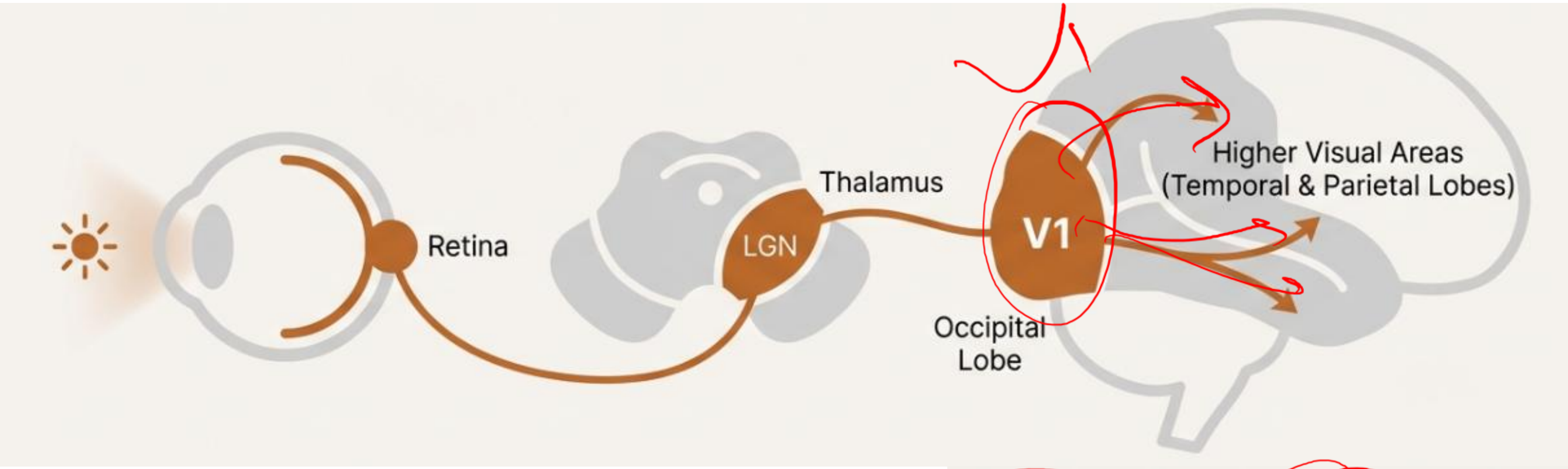
Problem Solved
Directly targets fragile feature dependencies (co-adaptation).

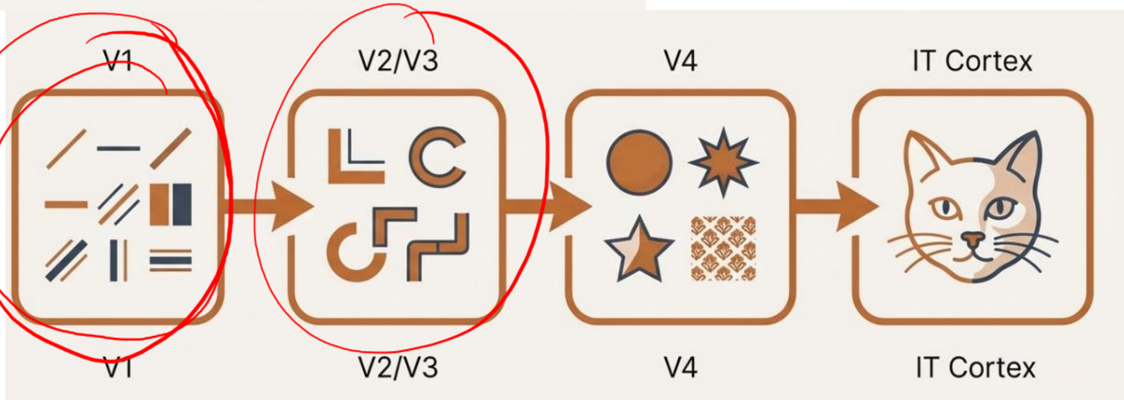
Intuition Analogy
A team project where any member might be absent on any given day. To succeed, everyone must learn the full context instead of relying on one person.



> *"No neuron is guaranteed to exist — learn features that stand alone."*

How Brain Processes Images -> CNN





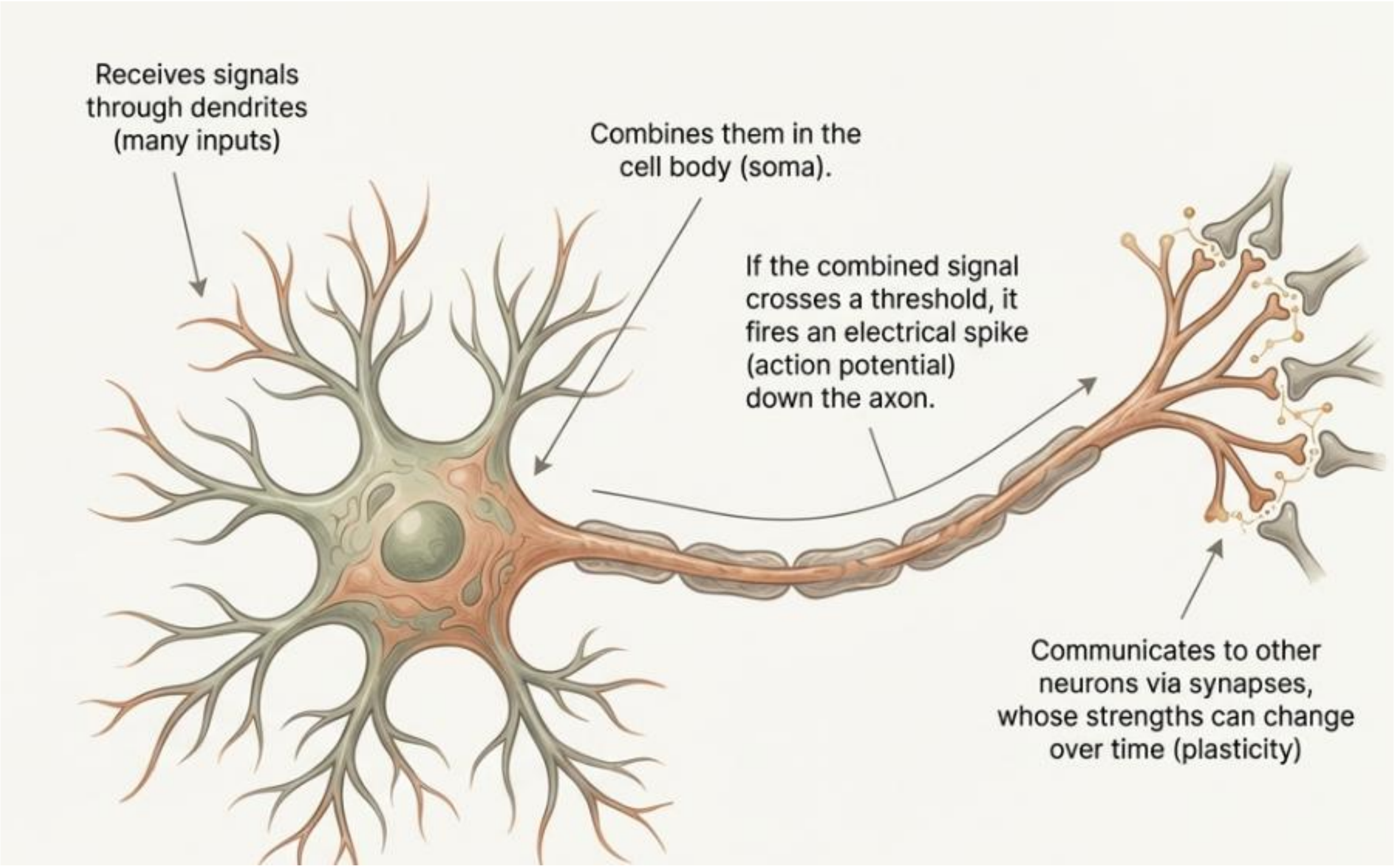
Thank You

https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Appendix

Neuron in Brain



Log Softmax

$$
\begin{pmatrix} 0.0000302941 \\ 0.0002238446 \\ 0.0000020359 \\ 0.9954517644 \\ 0.0001500475 \\ 0.0000061163 \\ 0.0000674207 \\ 0.0003690573 \\ 0.0036810450 \\ 0.0000183743 \end{pmatrix}
\qquad
\begin{pmatrix} -10.4045575839 \\ -8.4045584969 \\ -13.1045725764 \\ -0.0045586103 \\ -8.8045586473 \\ -12.0045532194 \\ -9.6045584655 \\ -7.9045586414 \\ -5.6045585997 \\ -10.9045576088 \end{pmatrix}
$$