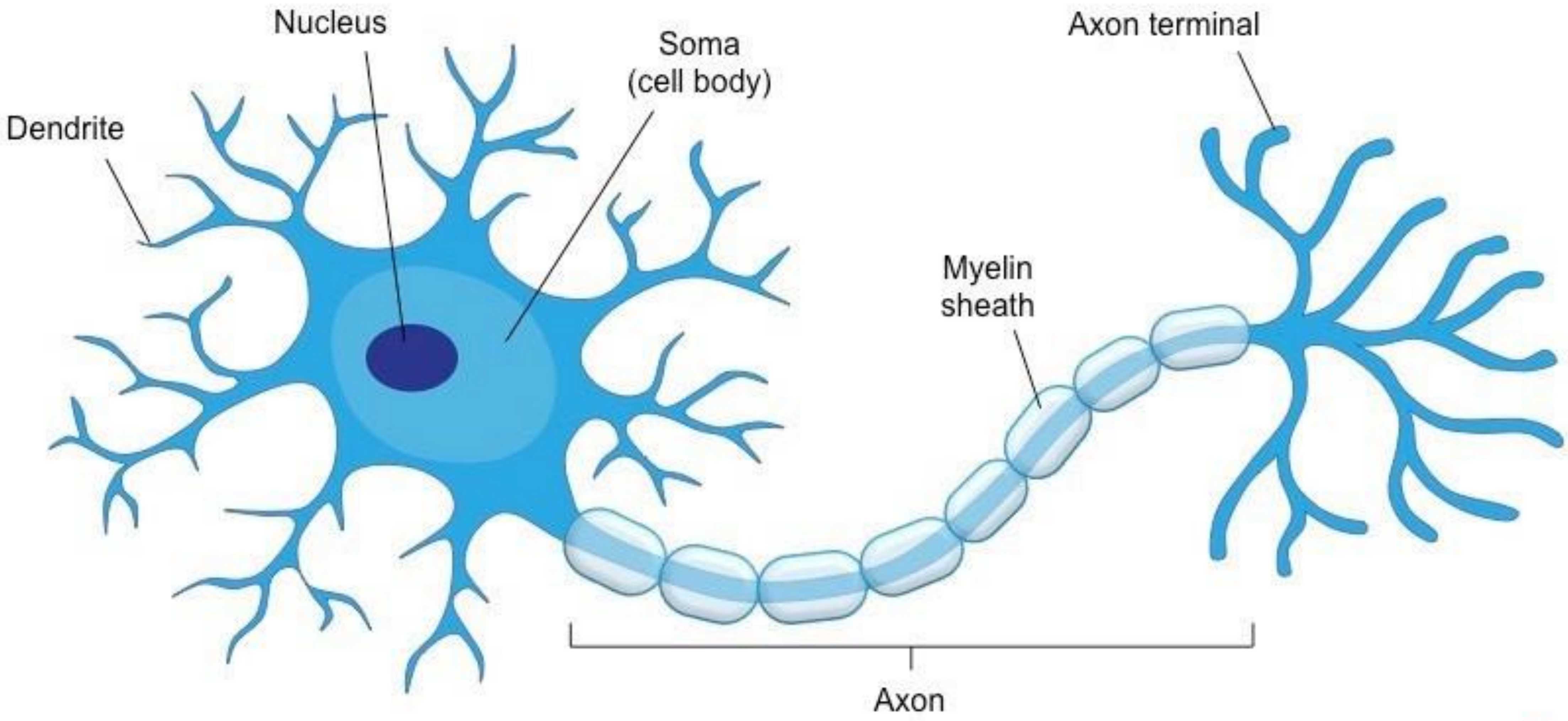
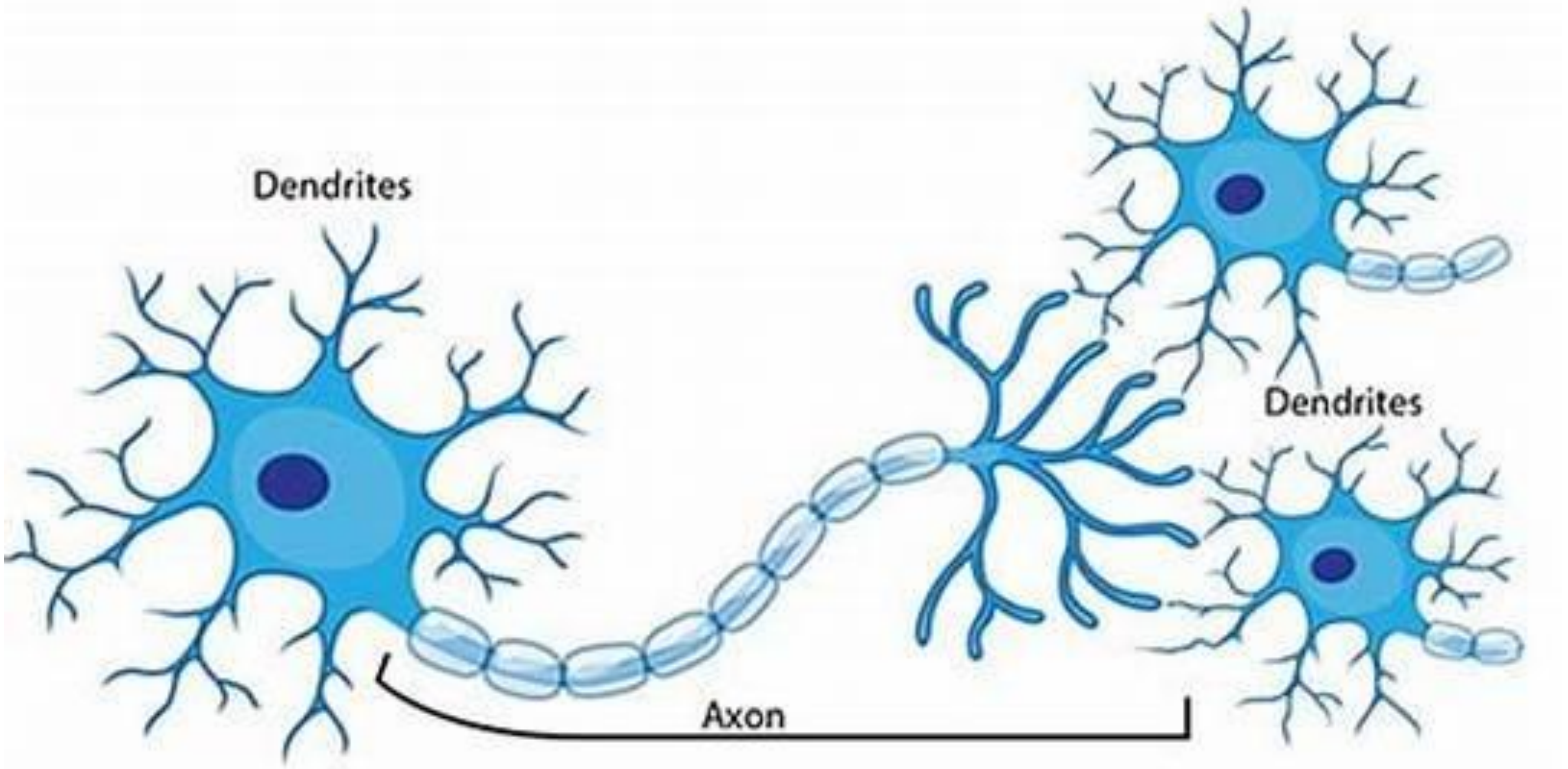
NEURAL NETWORKS

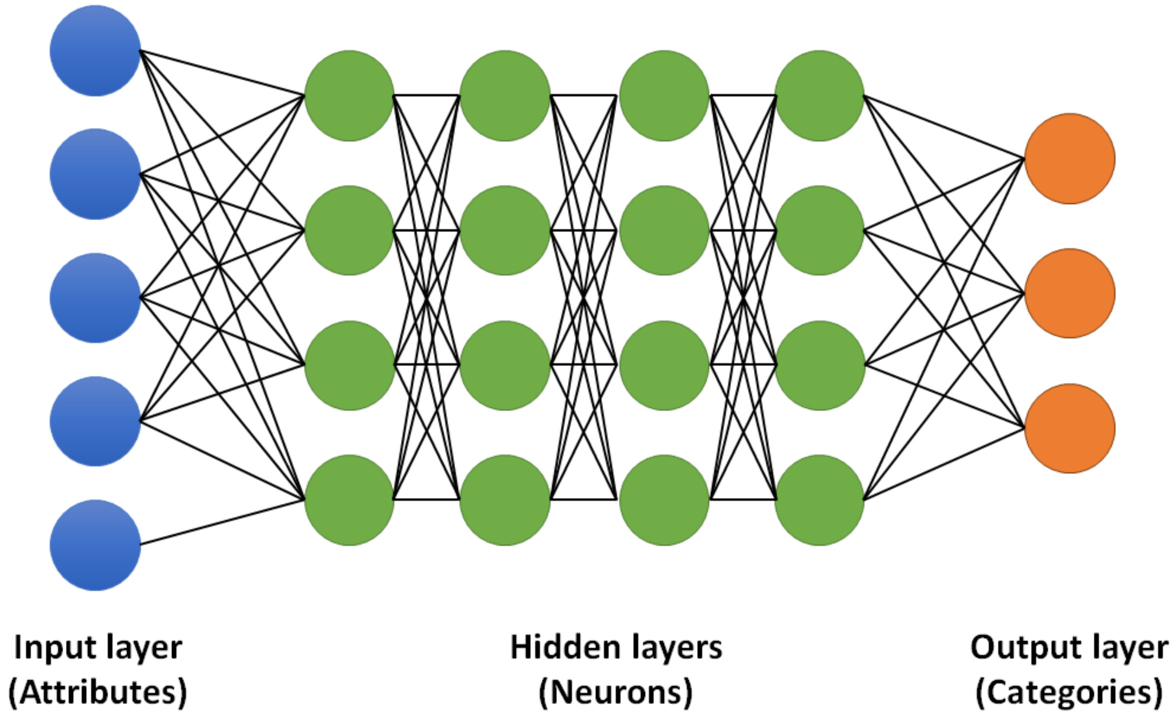
PERCEPTRON’S BIG BOY

FELIPE BUCHBINDER

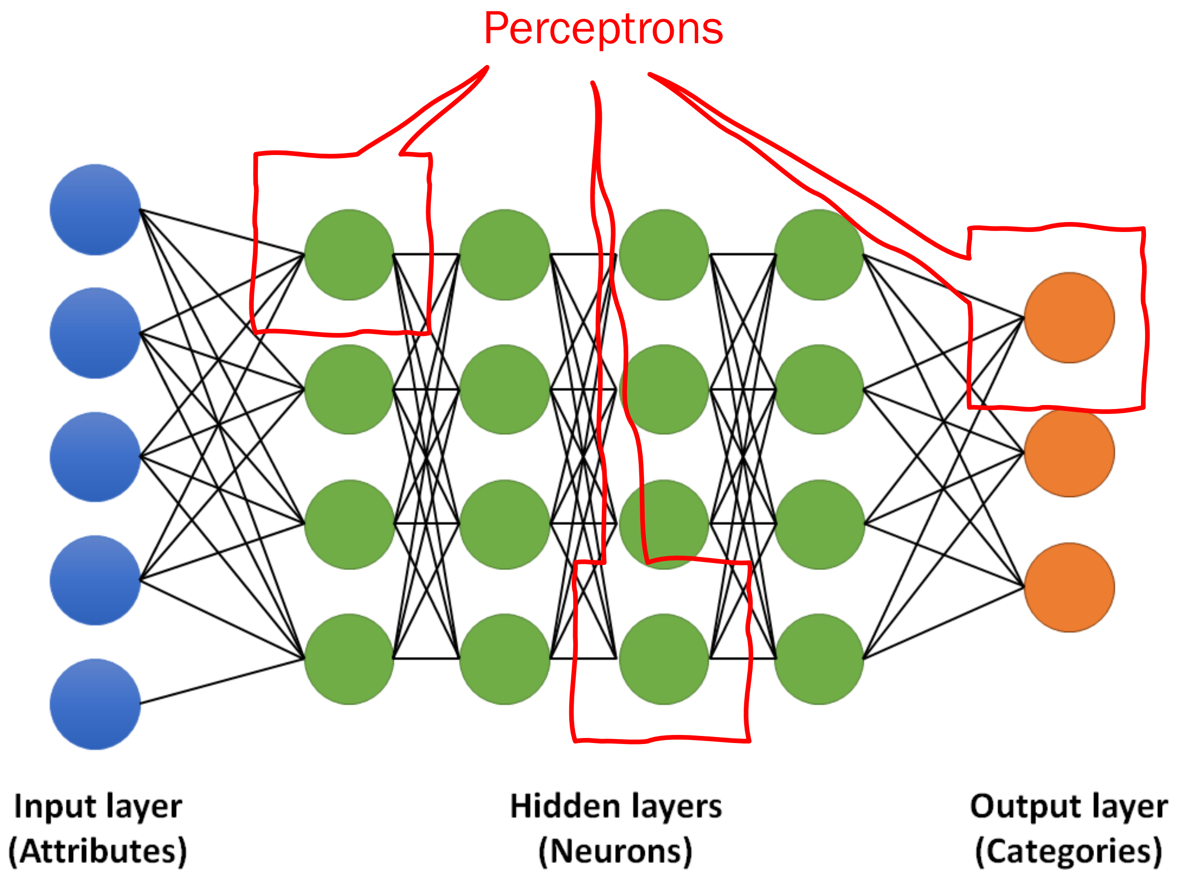

Nucleus
Soma
(cell body)
Axon terminal
Dendrite
Myelin
sheath
Axon


Dendrites
Dendrites
Axon

NEURAL NETWORK

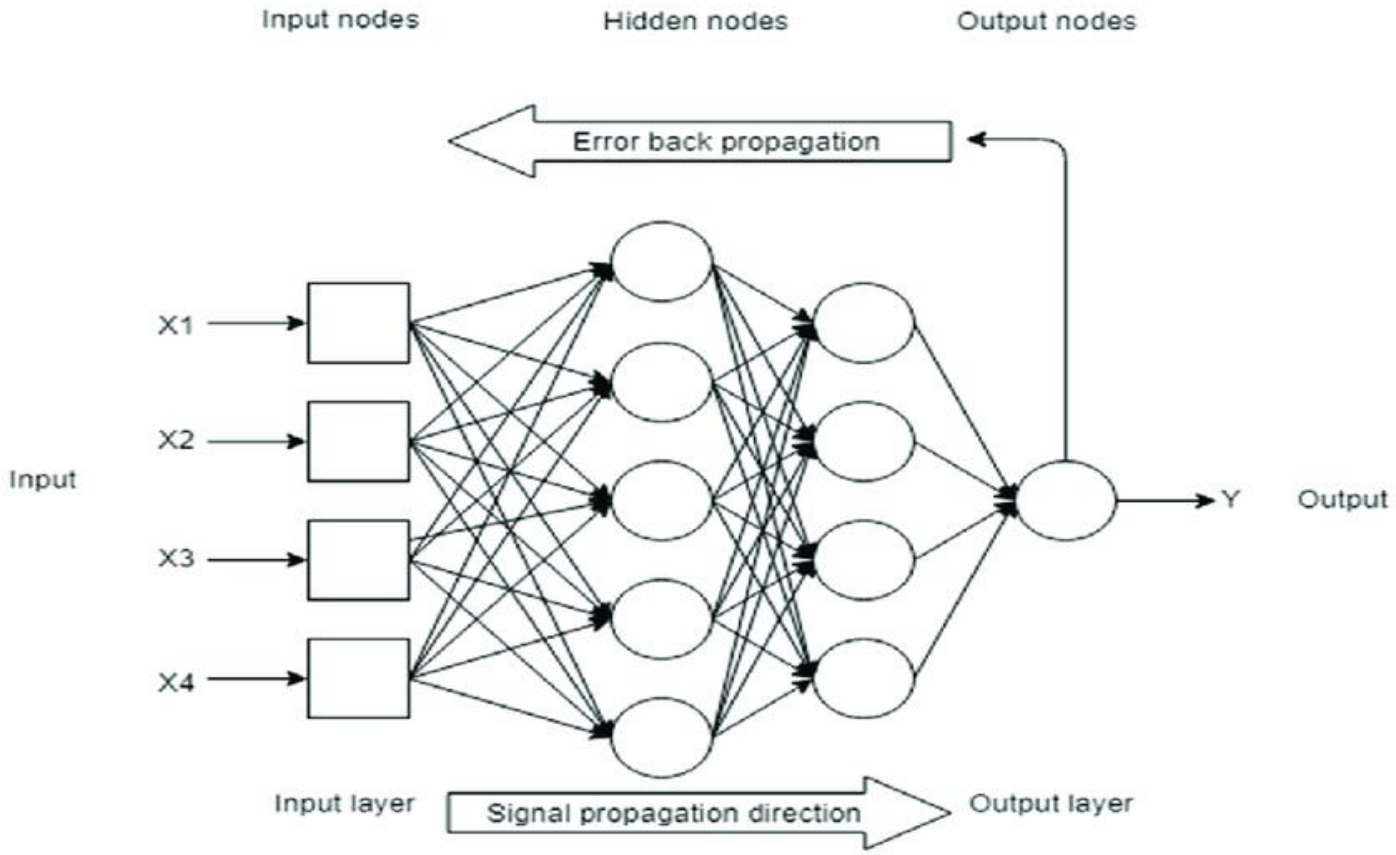


NEURAL NETWORK



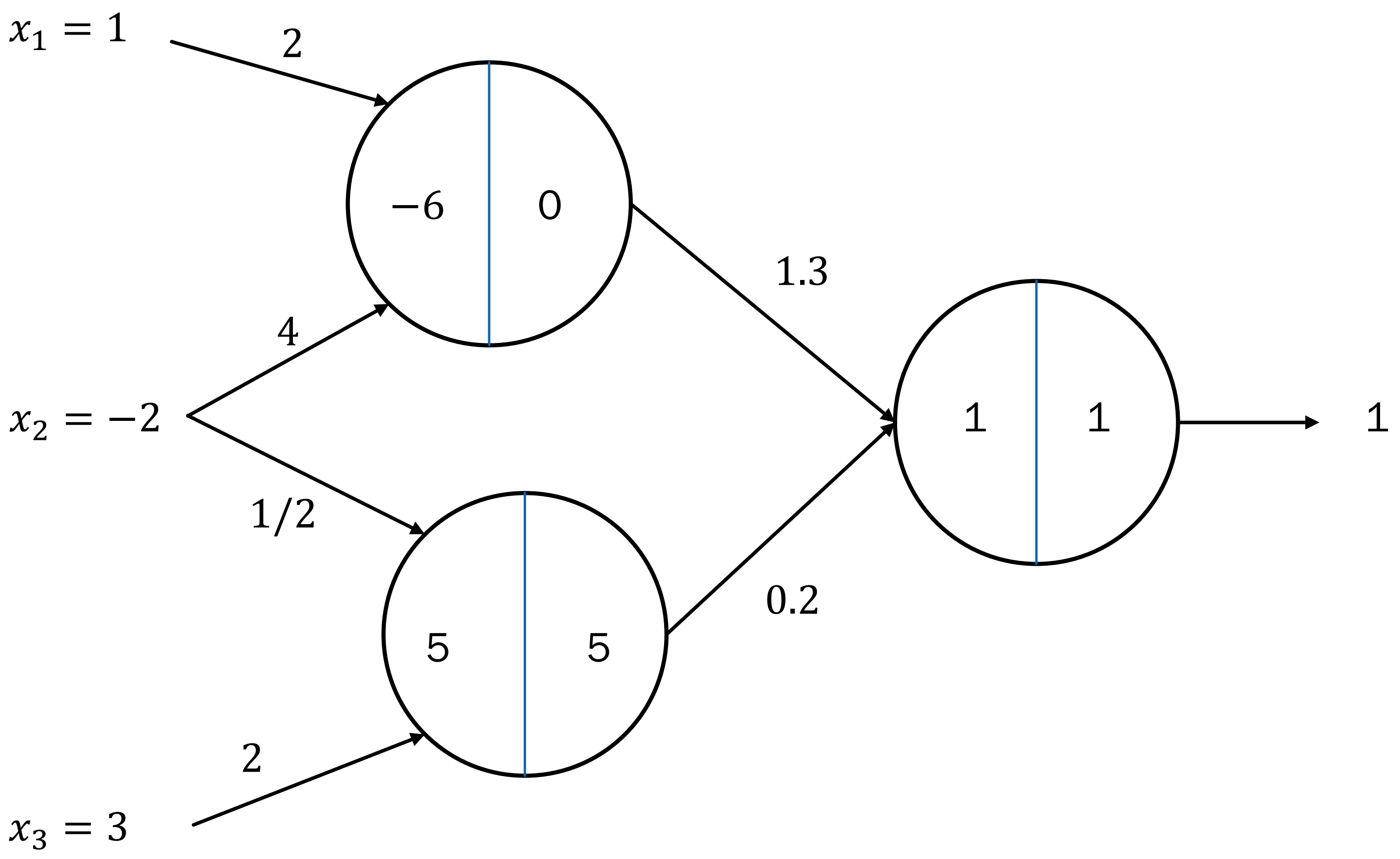
FORWARD PASS& BACKWARD PASS

- Forward propagation is used for making predictions.
- Backward propagation is used for adjusting weights (i.e. learning)



FORWARD PROPAGATION

MAKING PREDICTIONS



All activation functions are ReLU
$\mathrm{ReLU(x)} = \max(0, x)$

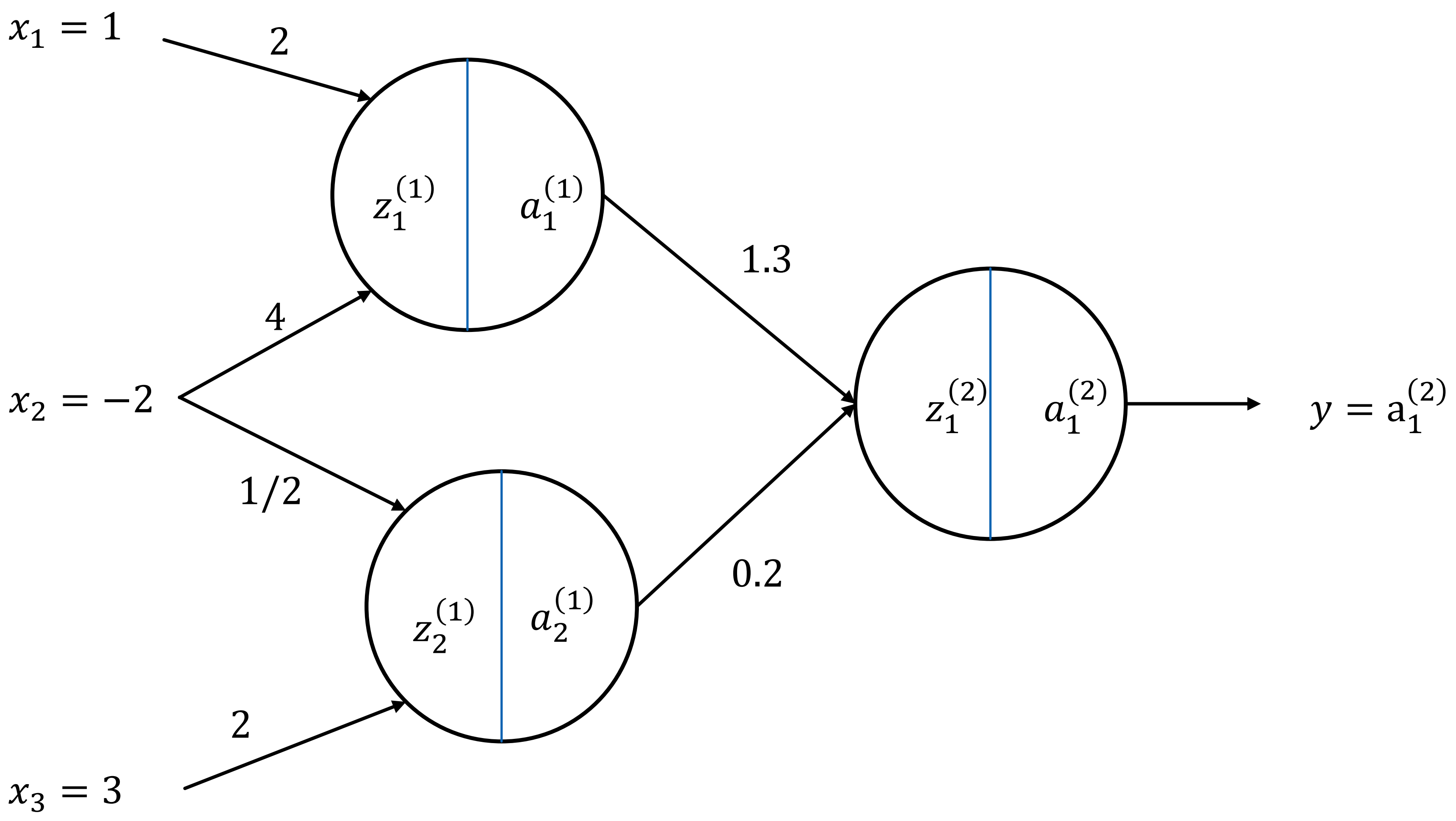
BACKPROPAGATION

LEARNING OPTIMUM WEIGHTS

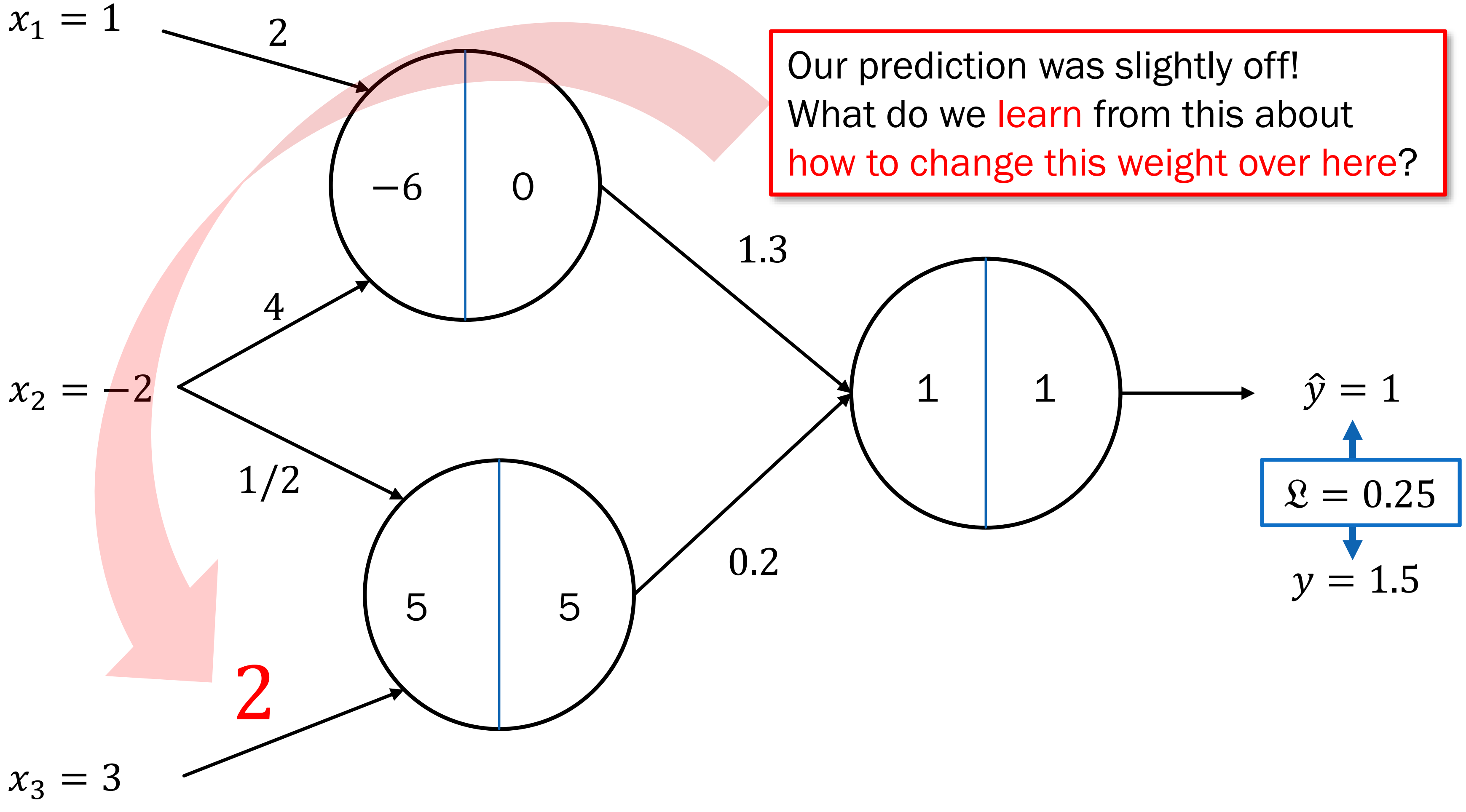
LEARNING IS A MINIMIZATION PROBLEM THAT'S SOLVED WITH BACKPROPAGATION

- For a Neural Network, learning means finding the weights that yield predictions with the minimum error, according to some error function.
- This error function depends, quite naturally, on the weights of the network. We therefore write it as $\mathfrak{L}(\mathbf{w})$.
- To solve this minimization problem, we use gradient descent. To calculate the gradient, we use backpropragation.

NOTATION



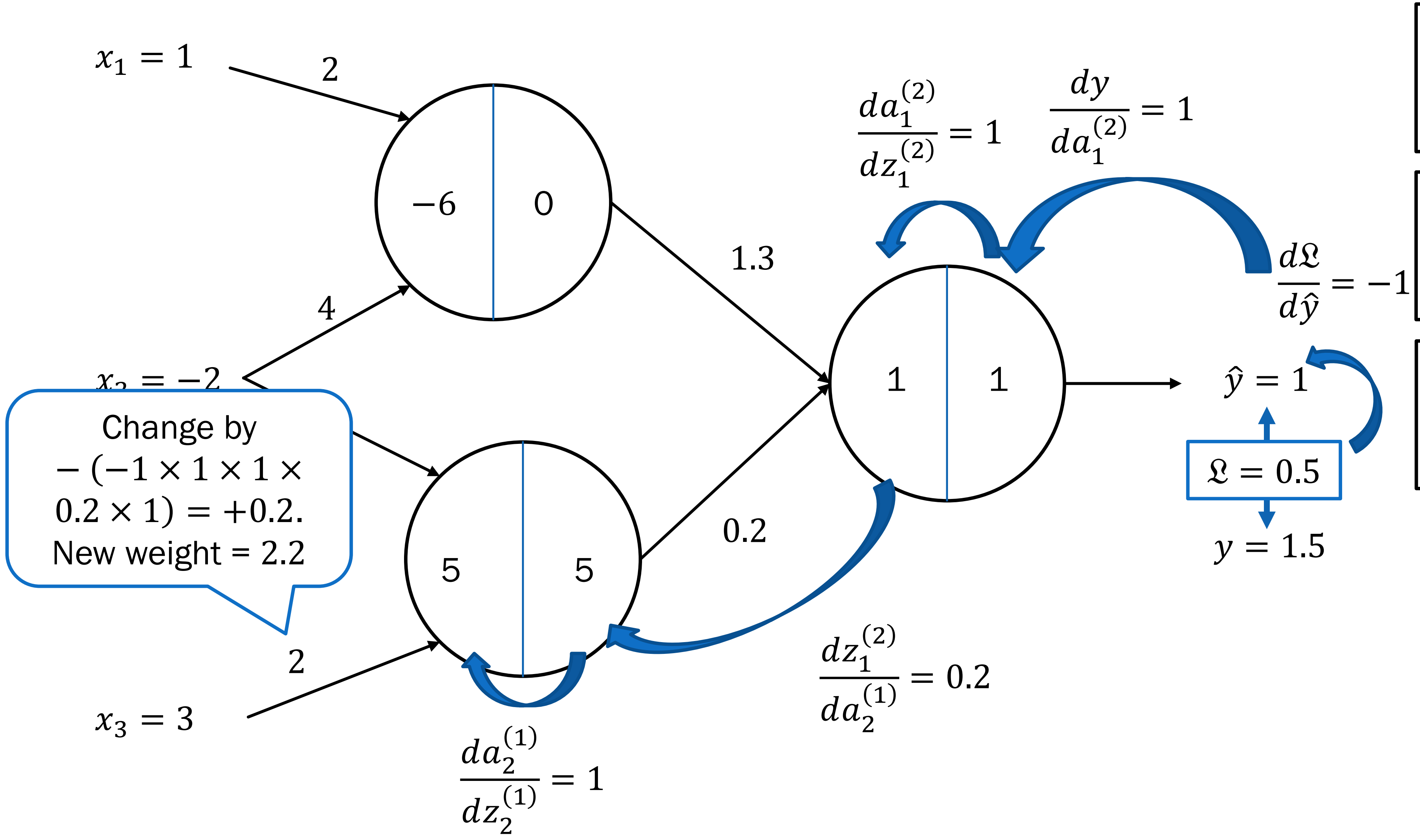
Our prediction was slightly off!
What do we learn from this about
how to change this weight over here?



All activation functions are ReLU
$\mathrm{ReLU}(\mathrm{x}) = \max(0, x)$

Loss function is MSE
$\mathfrak{L}(\mathbf{w}) = \frac{1}{2}(y - \hat{y})^2$

Learning rate
$\eta = 1$



All activation functions are ReLU
$\mathrm{ReLU(x)} = \max(0, x)$

Loss function is MAE
$\mathfrak{L}(\mathbf{w}) = |y - \hat{y}|$

Learning rate
$\eta = 1$


BACKPROPAGATION
IS ALL ABOUT THE
CHAIN RULE

BACKPROPAGATION (PSEUDO-)MATHS

$$\boxed{\frac{d\mathfrak{L}}{dw_i^{(l)}} = \frac{d\mathfrak{L}}{dz^{(n-1)}} \cdot \frac{dz^{(n-1)}}{da^{(n-2)}} \cdot \frac{da^{(n-2)}}{dz^{(n-3)}} \cdot \frac{dz^{(n-3)}}{da^{(n-4)}} \cdot \dots}$$

…along the path that connects y to $w_i^{(l)}$

This gives you the gradient to use to update the weights by gradient descent

$$w_{i\,[n+1]}^{(l)} \leftarrow w_{i\,[n]}^{(l)} - \eta \frac{d\mathfrak{L}}{dw_i^{(l)}}$$

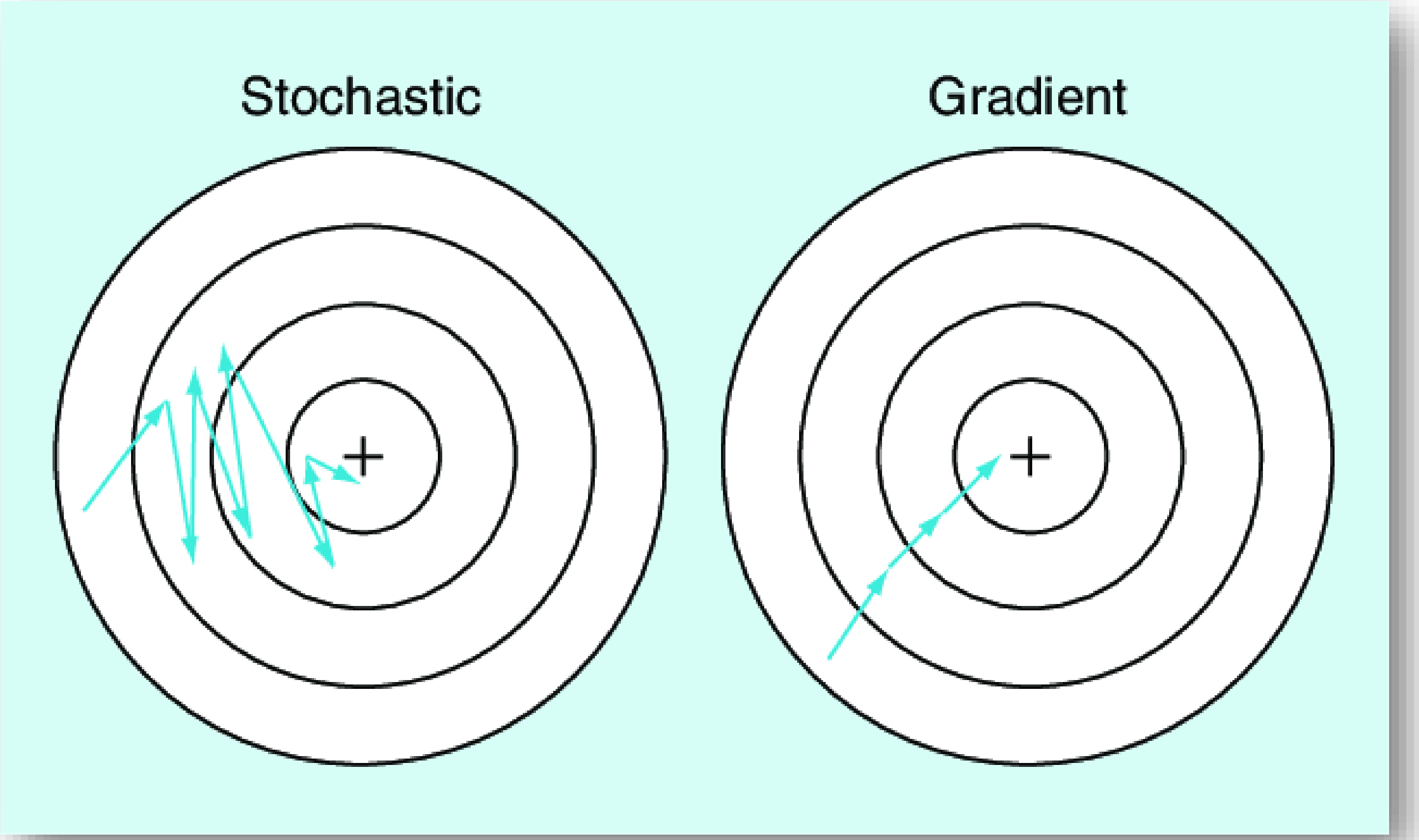
THERE ARE MANY IMPORTANT DECISIONS TO MAKE WHEN BUILDING & TRAINING A NEURAL NETWORK, PINKY

DECISIONS TO MAKE WHEN BUILDING A NEURAL NETWORK

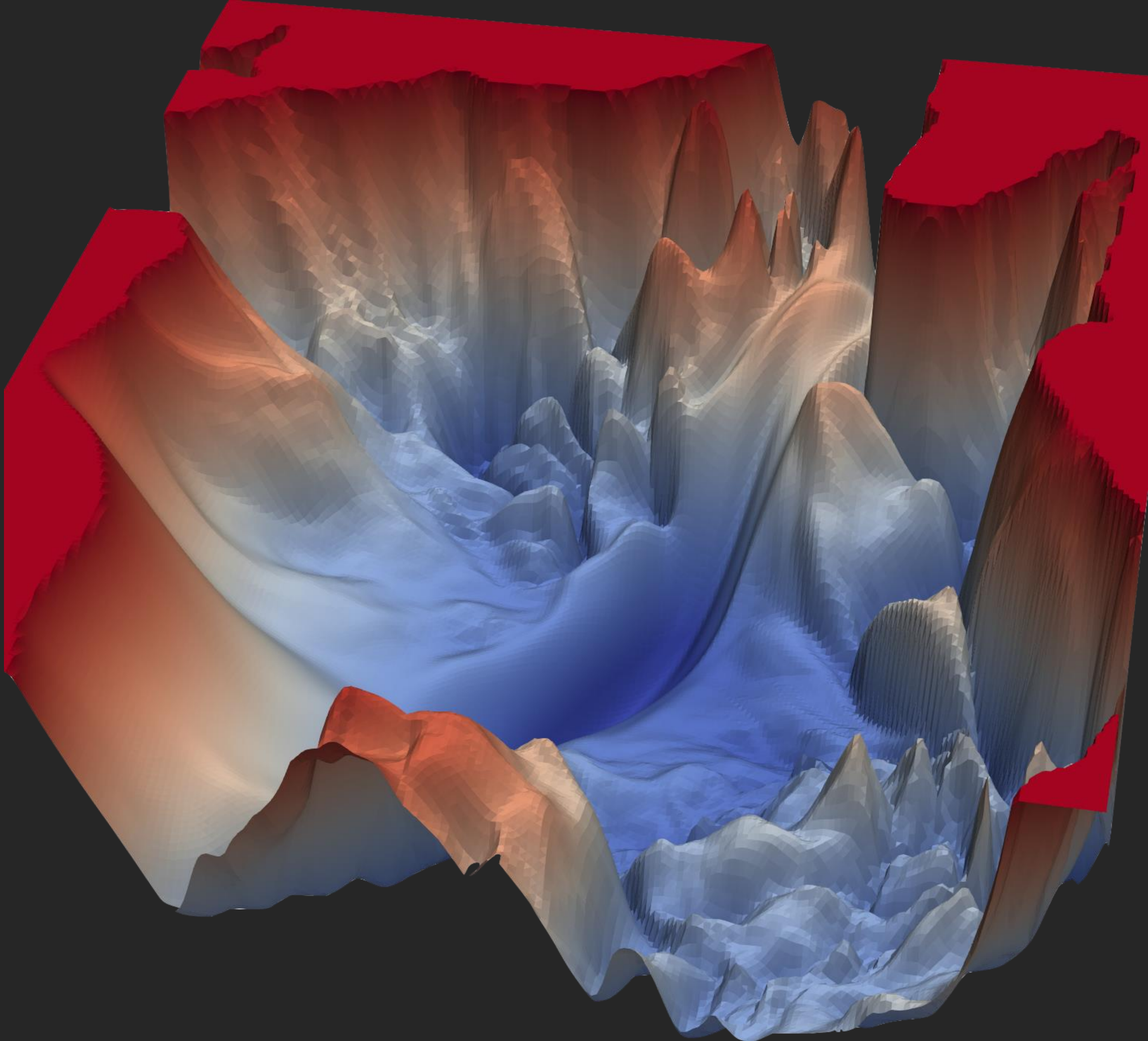
- Architecture
  - How many layers?
  - How many nodes in each layer?
- Activation functions
  - ReLU? Tanh? Sigmoid? Softmax? …
  - Can be different in each layer (and often is in the output layer)
  - Last layer is often sigmoid, tanh or softmax. Hidden layers are often ReLU or Leaker ReLU.
- Train from scratch or use transfer learning on a pretrained model? (more on this later)

DECISIONS TO MAKE WHEN TRAINING A NEURAL NETWORK

- Learning rate
- Number of epochs
- Batch size
  - Gradient Descent
  - Stochastic Gradient Descent
  - Batch Gradient Descent
  - Mini-Batch Gradient Descent
- Momentum (if any)
- Dropout rate (if any)



THE LOSS FUNCTION OF A NEURAL NETWORK IS TYPICALLY VERY COMPLICATED AND HAS MANY OF LOCAL MINIMA



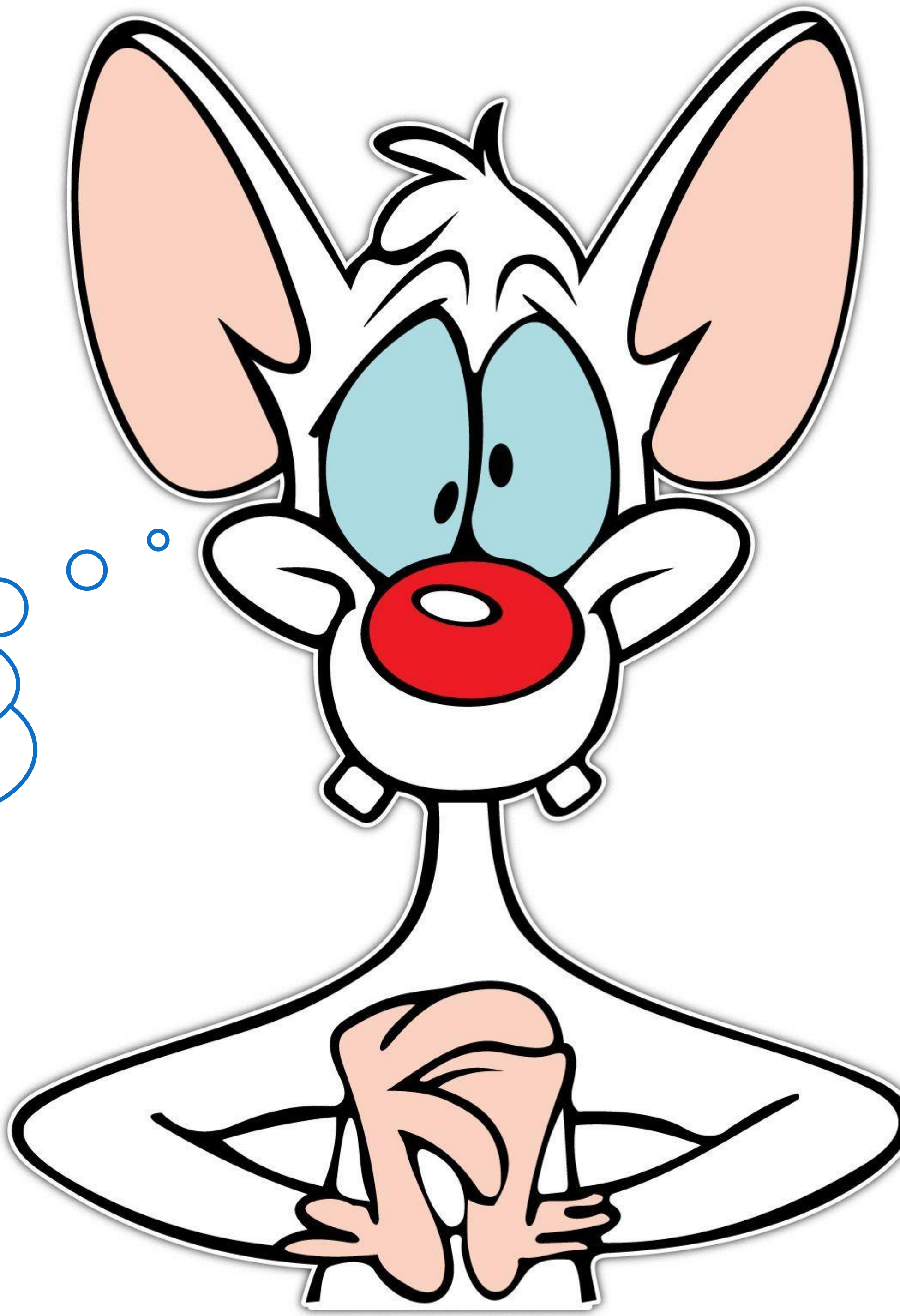
WAYS TO AVOID GETTING TRAPPED IN A LOCAL MINIMUM

- Increase learning rate
- Decrease batch size
- Increase momentum

WAYS TO AVOID GETTING TRAPPED IN A LOCAL MINIMUM

- Increase learning rate
- Decrease batch size
- Increase momentum

Wait!
Don’t I *want* to reach a minimum of the cost function?
Why is this a problem?



WAYS TO AVOID GETTING TRAPPED IN A LOCAL MINIMUM

- Increase learning rate
- Decrease batch size
- Increase momentum

How does each of these help avoid getting stuck in a local minimum?