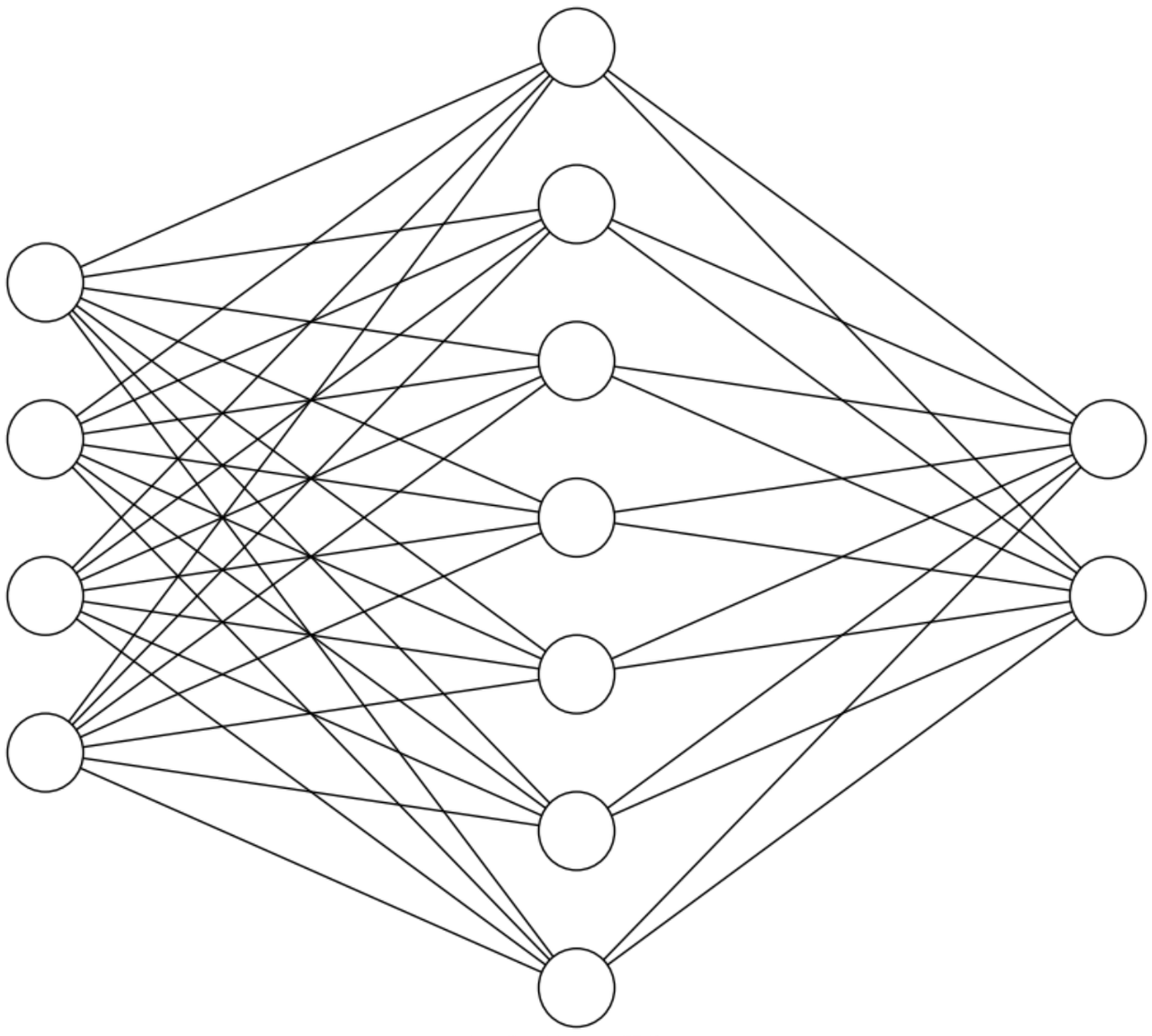
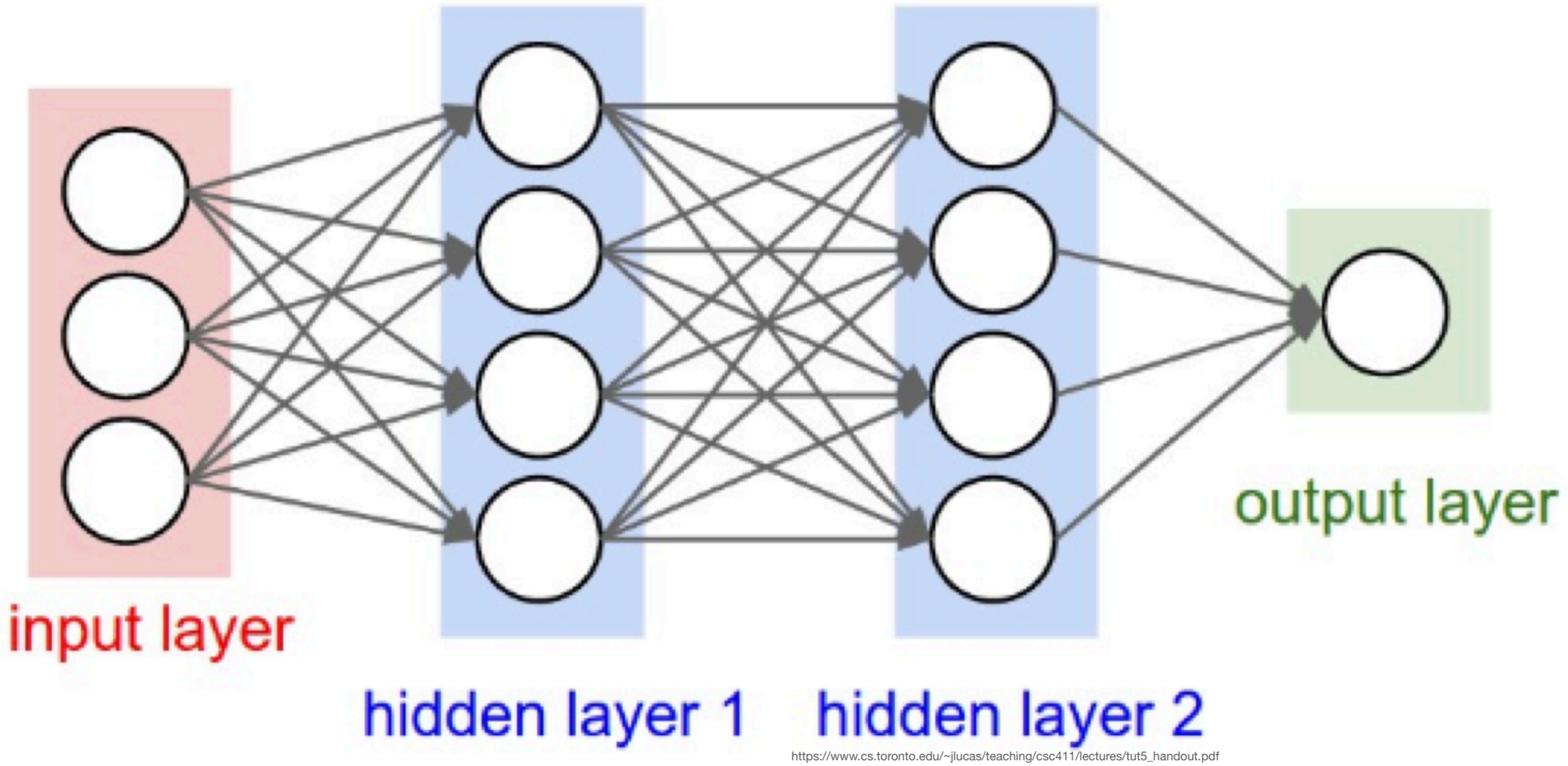
Neutral Networks



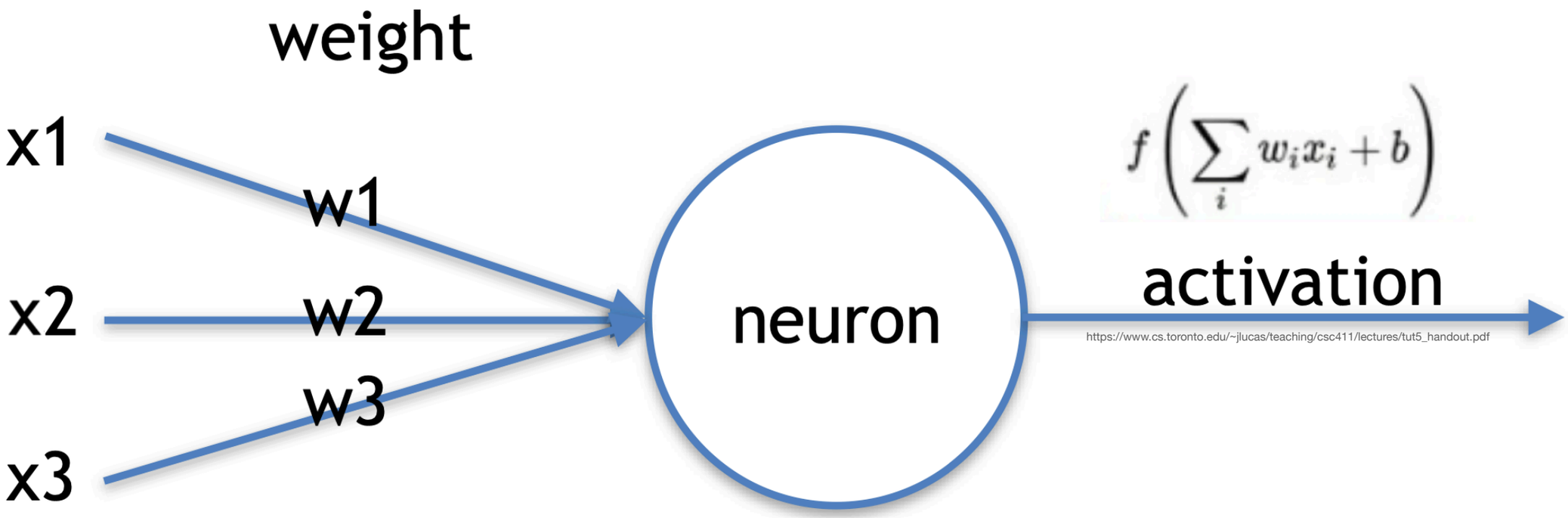
https://i.stack.imgur.com/PTwjO.png

Overview

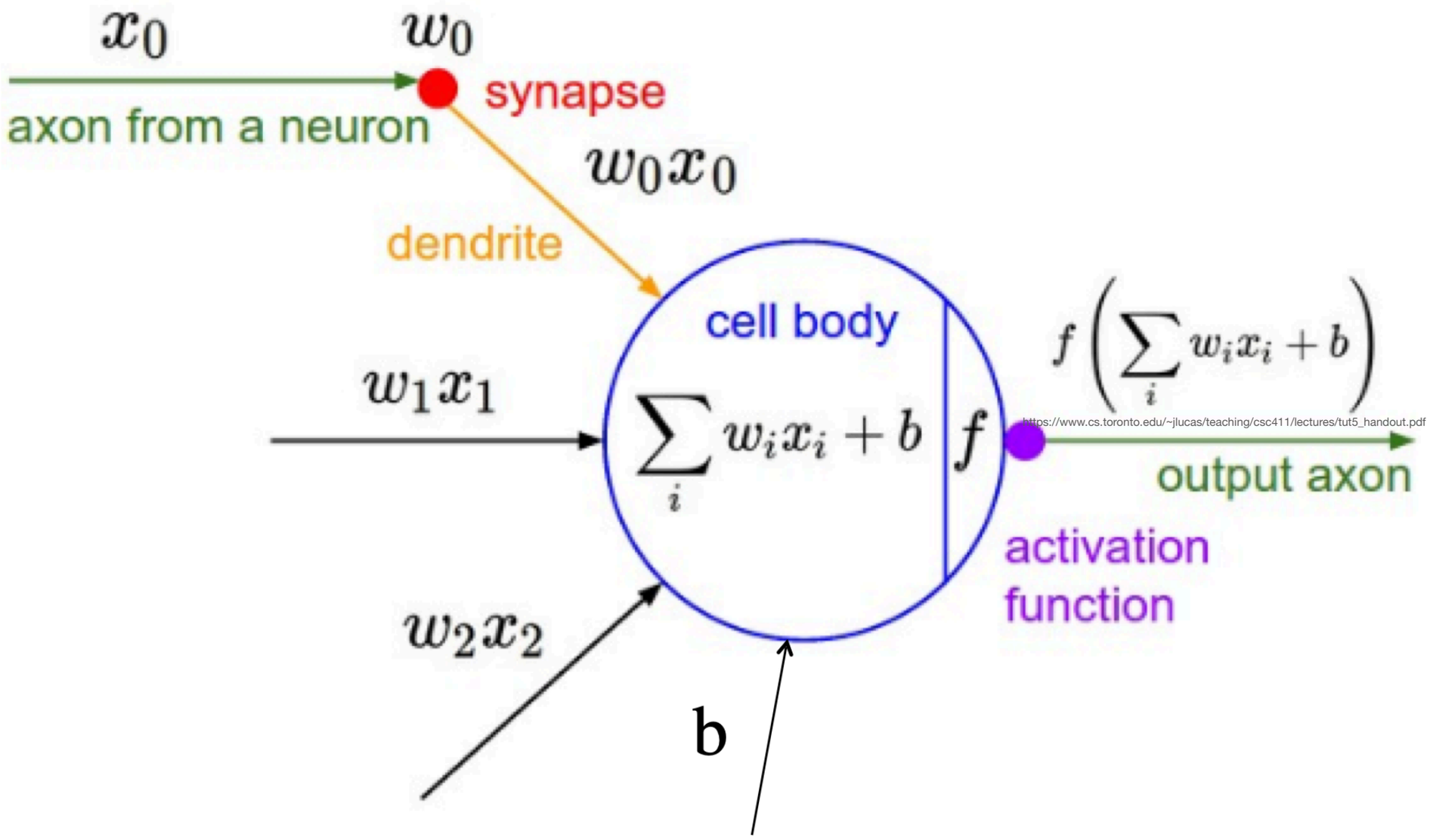


https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf

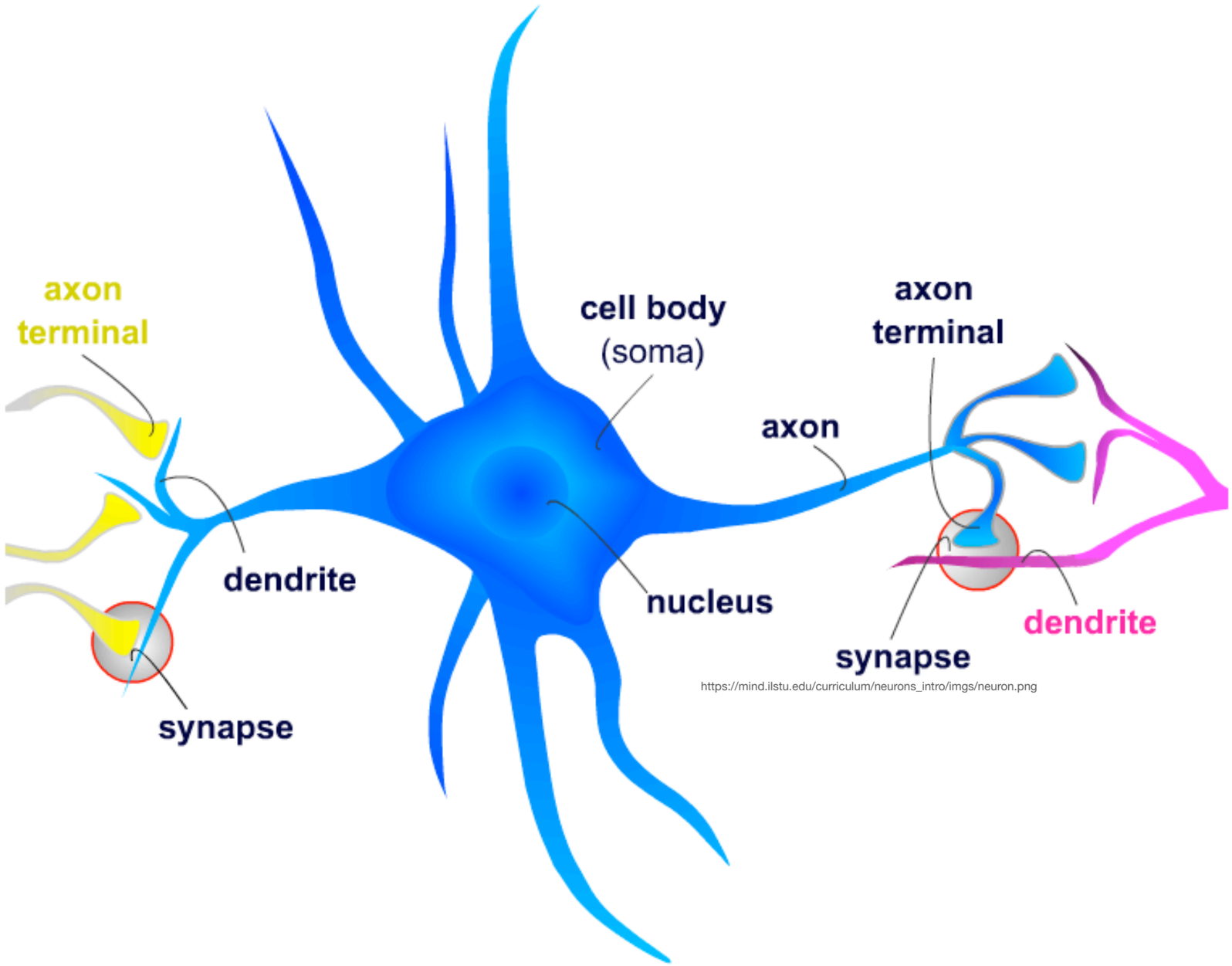
Neuron


weight
x1
w1
x2
w2
x3
w3
neuron
$f\left(\sum_i w_i x_i + b\right)$
activation
https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf

Biological Analogy



Biological Neuron



https://mind.ilstu.edu/curriculum/neurons_intro/imgs/neuron.png

A Neutral Network Playground

A Neutral Network Playground

Epoch
000,000

Learning rate: 0.03

Activation: Tanh

Regularization: None

Regularization rate: 0

Problem type: Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

X_1

X_2

X_1^2

X_2^2

X_1X_2

$\sin(X_1)$

$\sin(X_2)$

2 HIDDEN LAYERS

4 neurons

2 neurons

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

Test loss 0.522

Training loss 0.484

Colors shows data, neuron and weight values.

-1 0 1

Show test data

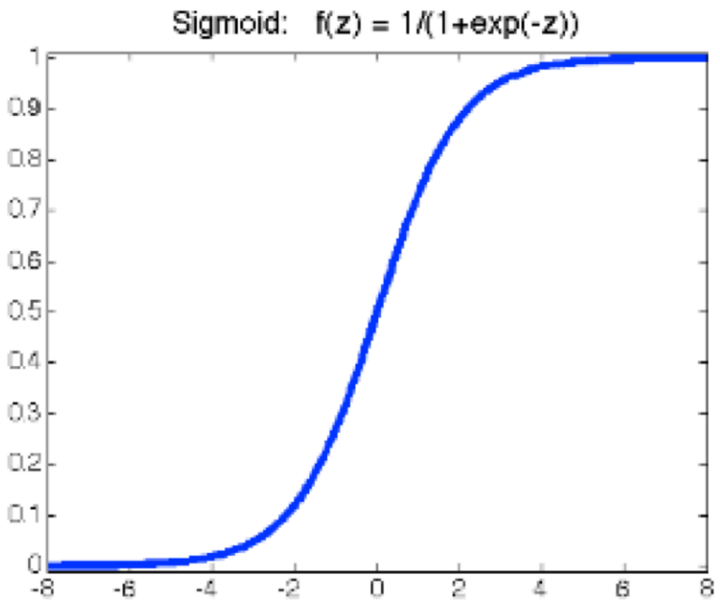
Discretize output

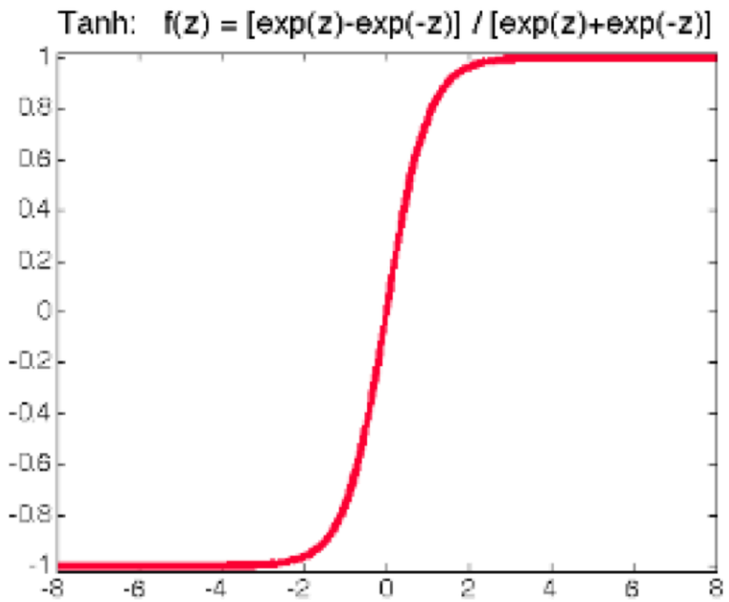
Link

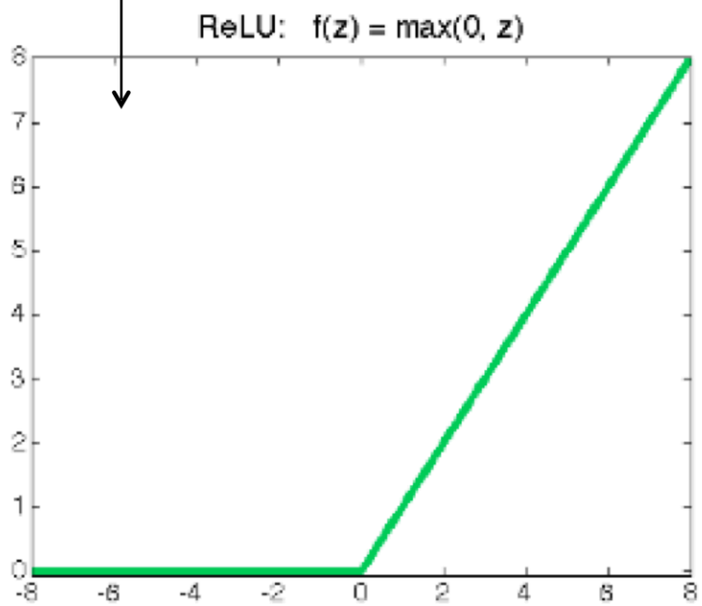
Activation Functions

- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$

Most popular recently for deep learning

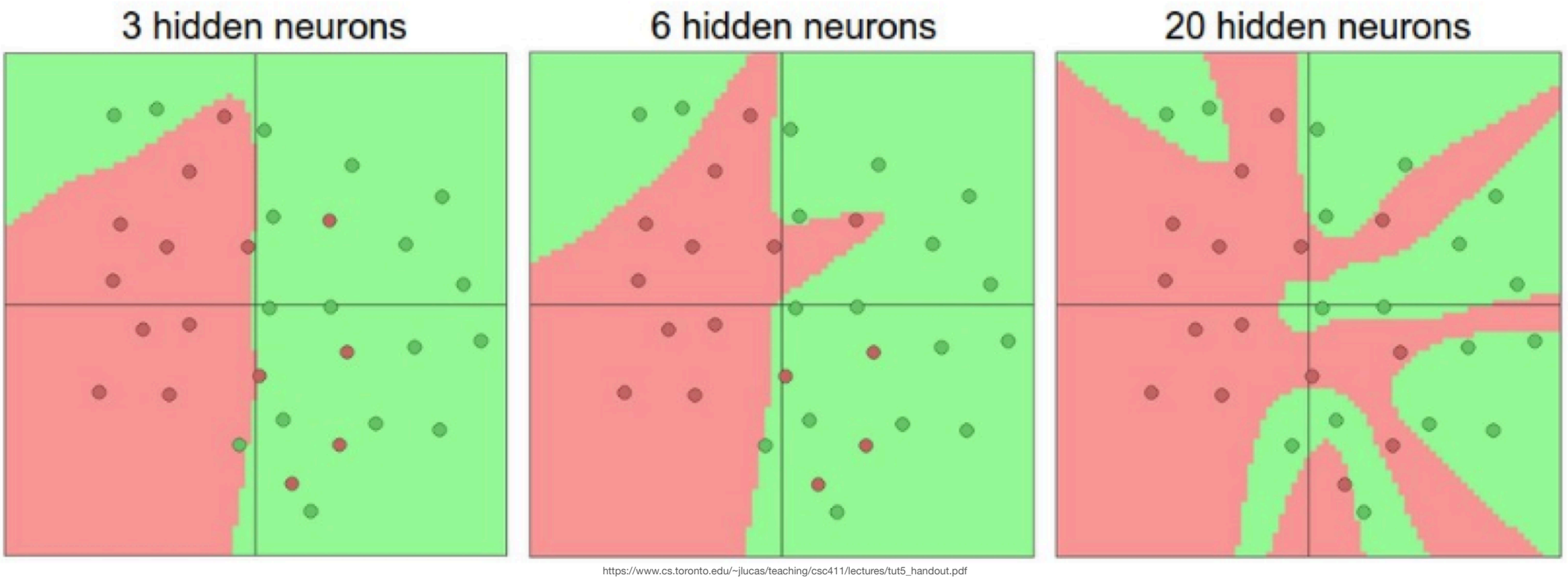






https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf

Representation Power



https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf

Training Neutral Networks

- Find weights:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} \operatorname{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network

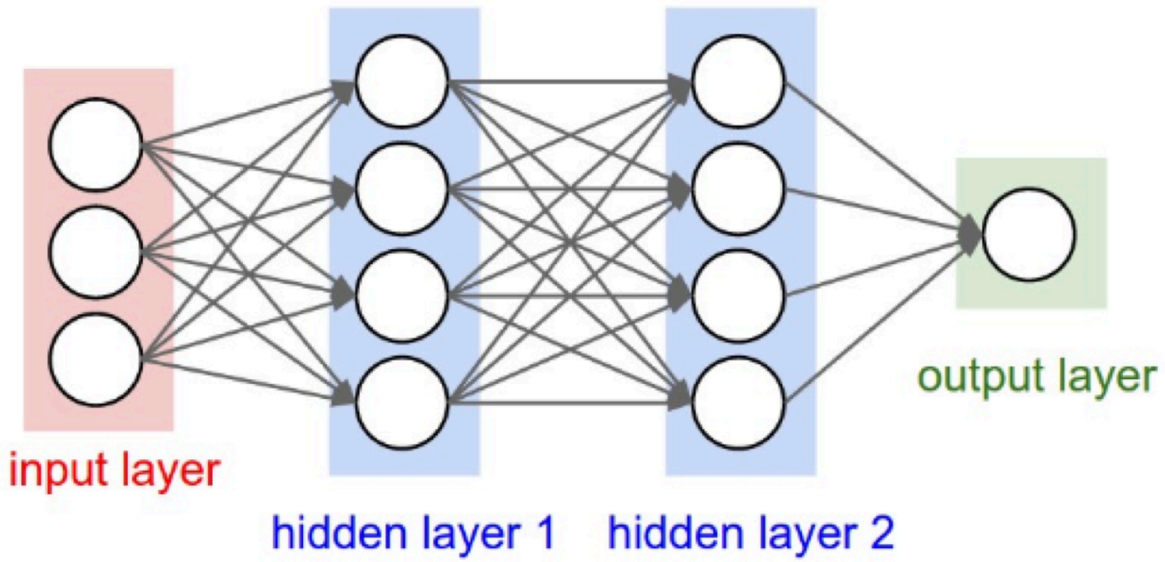
- Define a loss function, eg:
  - Squared loss: $\sum_k \frac{1}{2}(o_k^{(n)} - t_k^{(n)})^2$ **(Regression)**
  - Cross-entropy loss: $-\sum_k t_k^{(n)} \log o_k^{(n)}$ **(Classification)**
- Gradient descent:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\partial E}{\partial \mathbf{w}^t}$$

where η is the learning rate (and E is error/loss)

https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf

Forward Pass



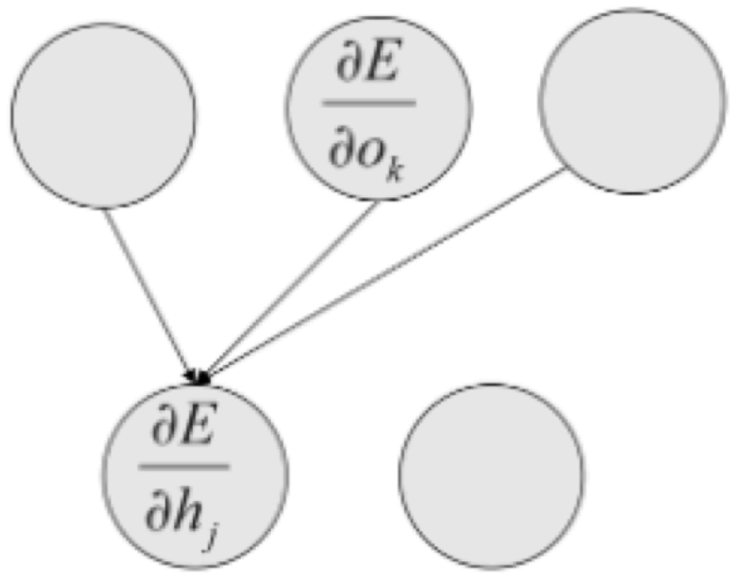
https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf

$$h_j(\mathbf{x}) = f(v_{j0} + \sum_{i=1}^{D} x_i v_{ji})$$

$$o_k(\mathbf{x}) = g(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj})$$

$$\sum_{n=1}^{N} \text{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

Backward Pass



https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf

Chain Rule extended

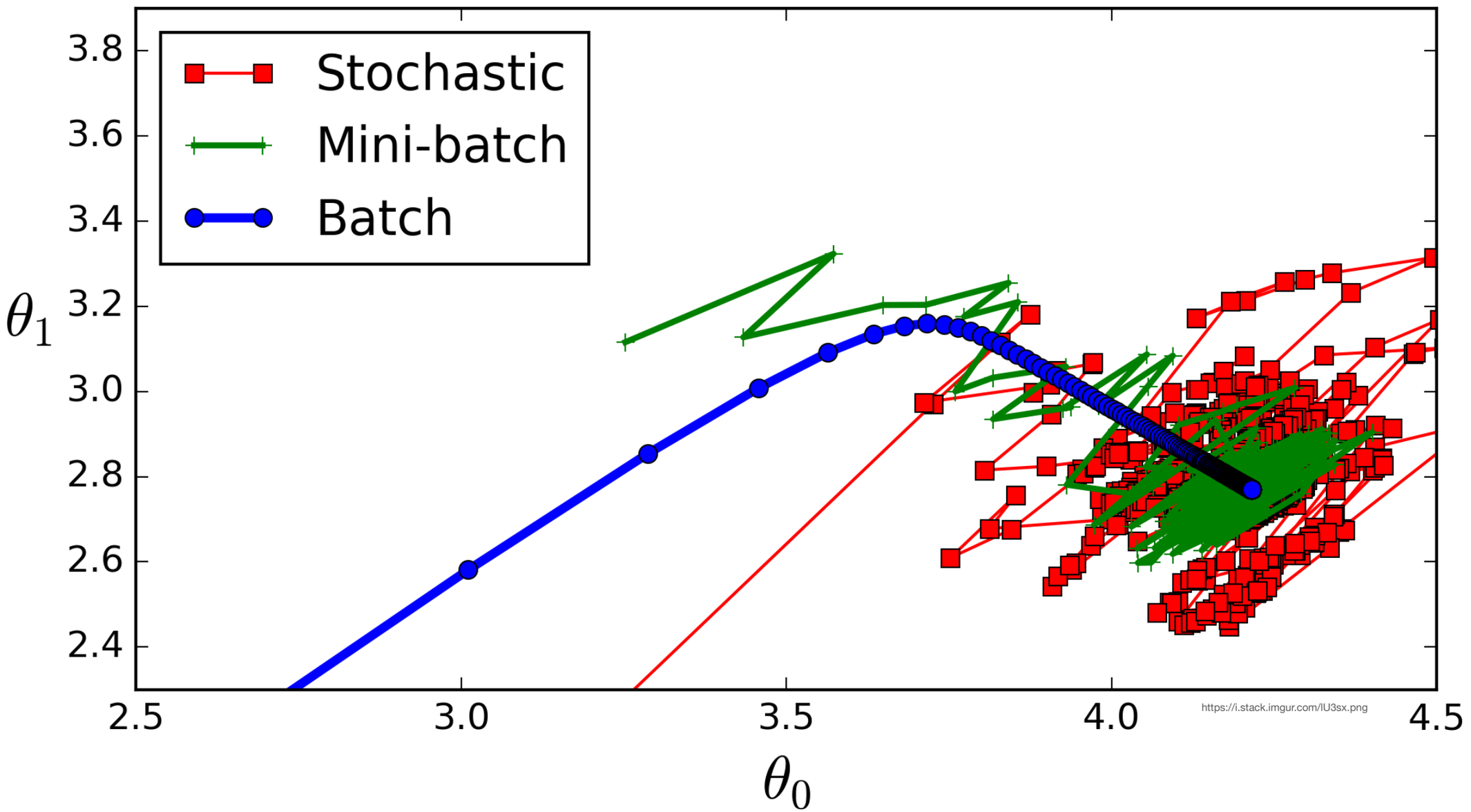
$$\frac{d}{dx}\left[f\left(g(h(x))\right)\right] = f'\left(g(h(x))\right)g'(h(x))h'(x)$$

https://andymath.com/wp-content/uploads/2019/08/chain-rule-extended.jpg

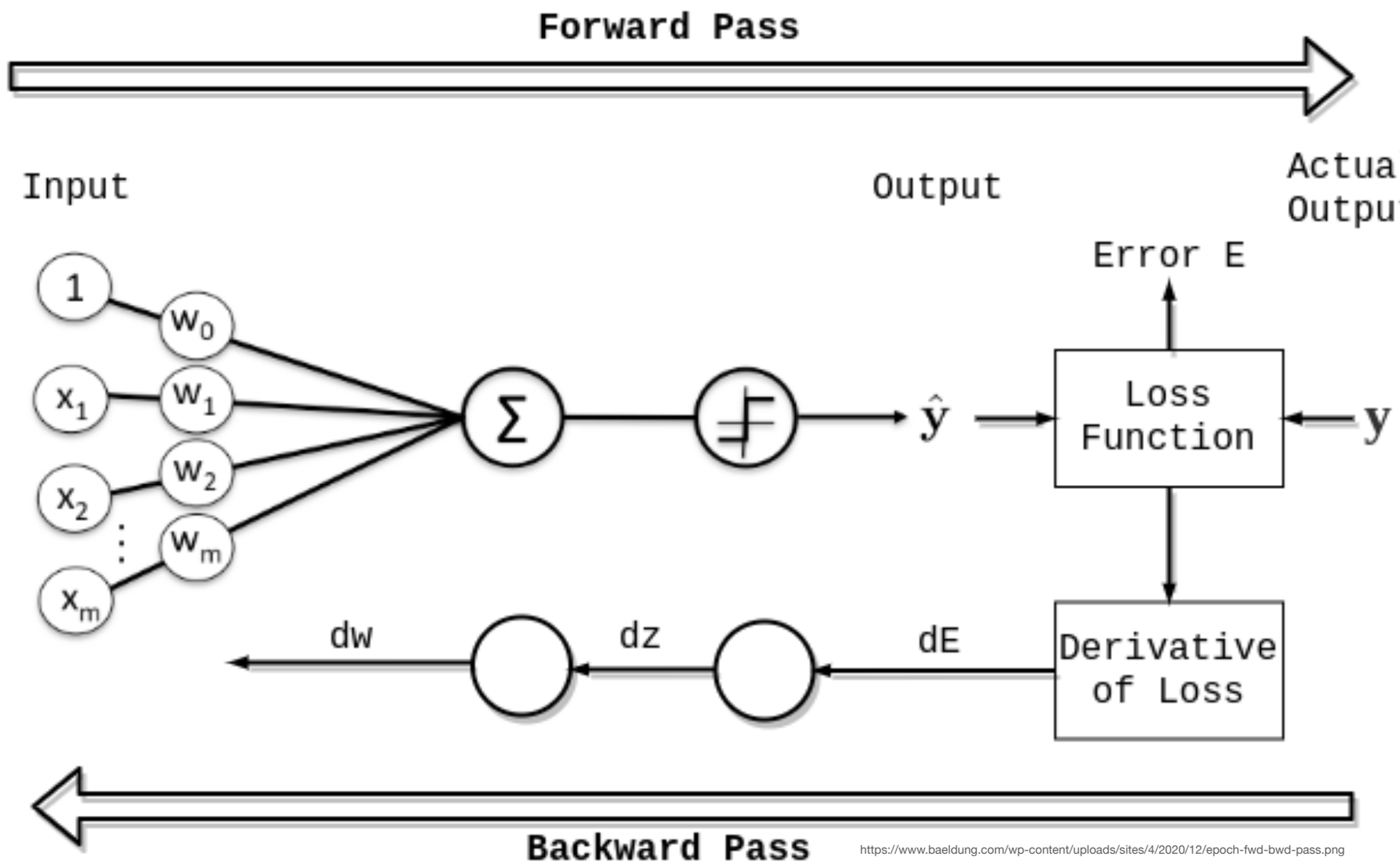
$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

https://www.gstatic.com/education/formulas2/553212783/en/chain_rule.svg

Batch Gradient Descent

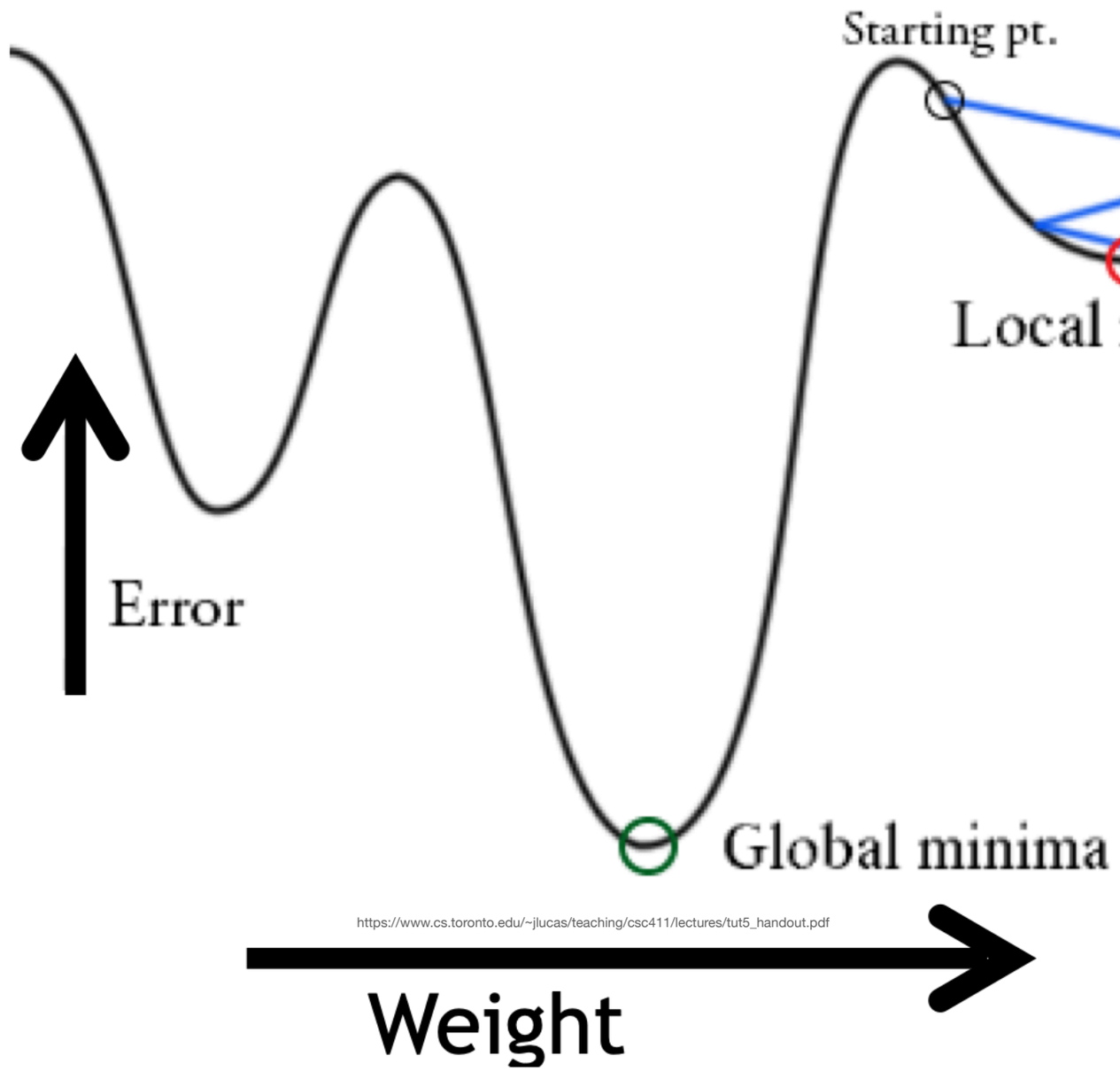


Epochs



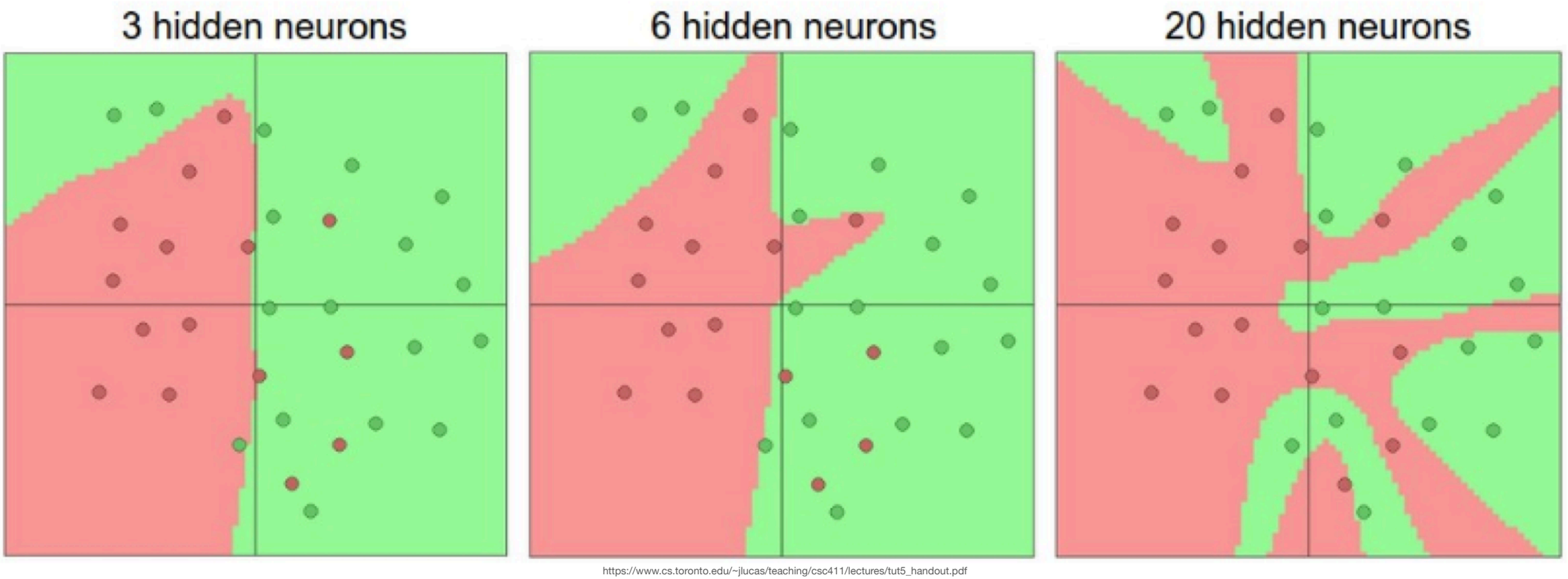
https://www.baeldung.com/wp-content/uploads/sites/4/2020/12/epoch-fwd-bwd-pass.png

Local Optimization



https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf

Preventing Overfitting



https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/tut5_handout.pdf