Neural Networks II

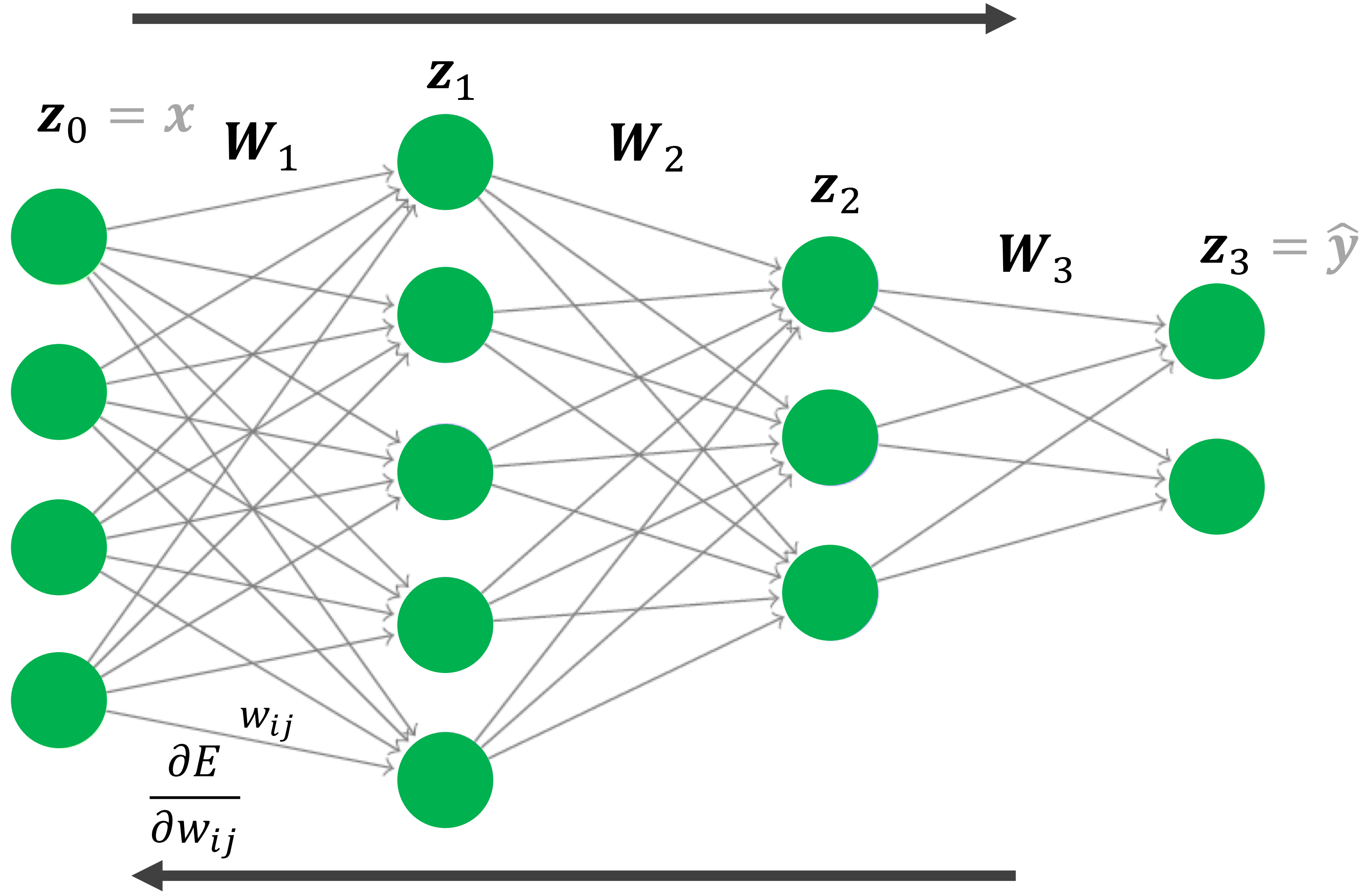
Lecture 19

What is a neural network and **how does it work**?

How do we **choose model weights**?
(i.e. how do we fit our model to data)

What are the challenges of using neural networks?

Forward propagation to create prediction and calculate training error



$$E = \frac{1}{2}\sum_k (\hat{y}_k - y_k)^2$$

Backpropagation lets us **assign the error** to each of the parameters so we can tune them

(gradient descent) $w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$

Backpropagation is simply the recursive application of the chain rule

Chain rule

Define a function:

$$f(x) = \sin[\ln(x)]$$

$$f(g(x)) = \sin[\ln(x)]$$

Component functions…

$$g(x) = \ln(x)$$

$$f(g) = \sin(g)$$

…with corresponding derivatives:

$$\frac{\partial g}{\partial x} = \frac{1}{x}, \qquad \frac{\partial f}{\partial g} = \cos(g) = \cos[\ln(x)]$$

Using the chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} = \cos[\ln(x)]\left(\frac{1}{x}\right) = \frac{\cos[\ln(x)]}{x}$$

Backpropagation intuitively

Consider a derivative of a complicated function that can be represented as a long chain rule application

$$\frac{\partial f}{\partial z} = \underbrace{\frac{\partial f}{\partial w}\frac{\partial w}{\partial x}}_{\frac{\partial f}{\partial x}}\frac{\partial x}{\partial y}\frac{\partial y}{\partial z}$$

Chain rule equality

This process of using the next step in the chain rule is backpropagation

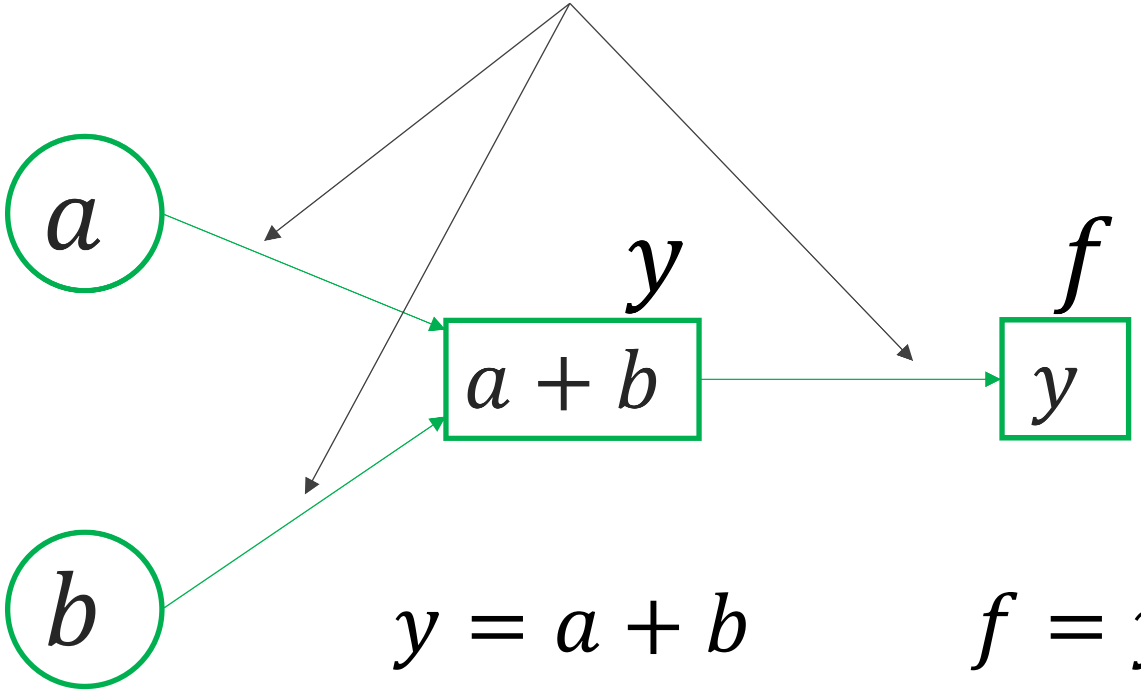
$$\frac{\partial f}{\partial z} = \underbrace{\frac{\partial f}{\partial w}\frac{\partial w}{\partial x}}_{\frac{\partial f}{\partial x}}\frac{\partial x}{\partial y}\frac{\partial y}{\partial z} = \underbrace{\frac{\partial f}{\partial x}\frac{\partial x}{\partial y}}_{\frac{\partial f}{\partial y}}\frac{\partial y}{\partial z} = \underbrace{\frac{\partial f}{\partial y}\frac{\partial y}{\partial z}}_{\frac{\partial f}{\partial z}} = \frac{\partial f}{\partial z}$$

Simple example

$$f(a, b) = a + b$$

This graph to the right represents this function

Edges are outputs from the last node and inputs to the next function.



Name of the variable at that node

$$y = a + b \qquad f = y$$

Operation that the node performs

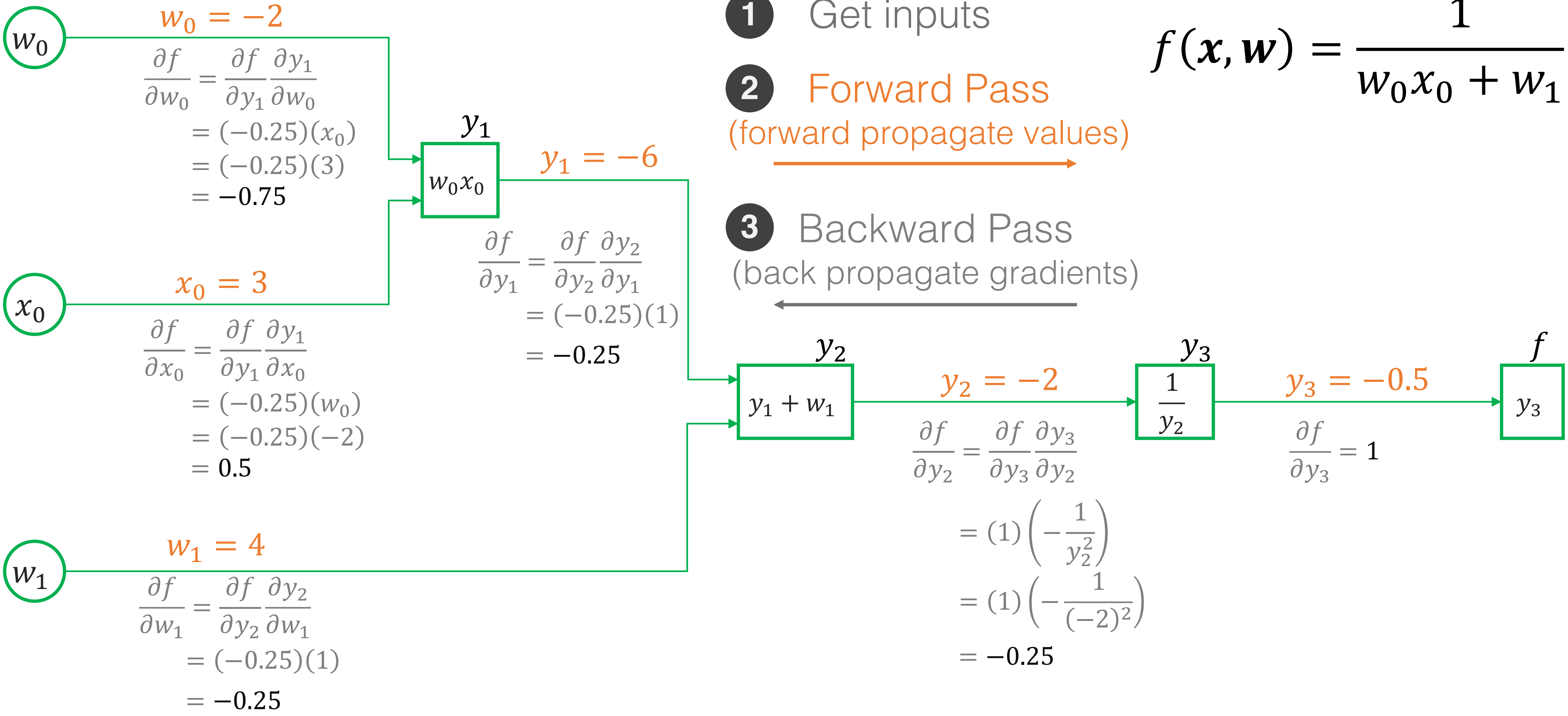
Local derivatives (one for each edge input into a node): $\dfrac{\partial y}{\partial a} = 1, \quad \dfrac{\partial y}{\partial b} = 1$

1 Get inputs

2 Forward Pass
(forward propagate values)

3 Backward Pass
(back propagate gradients)

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{w_0 x_0 + w_1}$$



Local derivatives:
(one per edge)

$$\frac{\partial y_1}{\partial w_0} = x_0, \quad \frac{\partial y_1}{\partial x_0} = w_0 \qquad \frac{\partial y_2}{\partial y_1} = \frac{\partial y_2}{\partial w_1} = 1 \qquad \frac{\partial y_3}{\partial y_2} = -\frac{1}{y_2^2} \qquad \frac{\partial f}{\partial y_3} = 1$$

Let’s try an example closer to a real neural network

input layer 0 | hidden layer 1 | hidden layer 2 | hidden layer 3

$z_i = \sigma(a_i)$

$$x = z_0 \xrightarrow{w_1} a_1\; z_1 \xrightarrow{w_2} a_2\; z_2 \xrightarrow{w_3} a_3\; z_3 = \hat{y}$$

$$E = \frac{1}{2}(\hat{y} - y)^2$$

$$z_3 = \hat{y} = \sigma(a_3)$$

$$a_3 = w_3 z_2$$

$$z_2 = \sigma(a_2)$$

$$a_2 = w_2 z_1$$

$$z_1 = \sigma(a_1)$$

$$a_1 = w_1 z_0 \to x$$

Forward propagation

In this particular case, this could be written as a single function of z_0

$$\hat{y} = z_3 = \sigma\left(w_3 \sigma\left(w_2 \sigma(w_1 z_0)\right)\right)$$

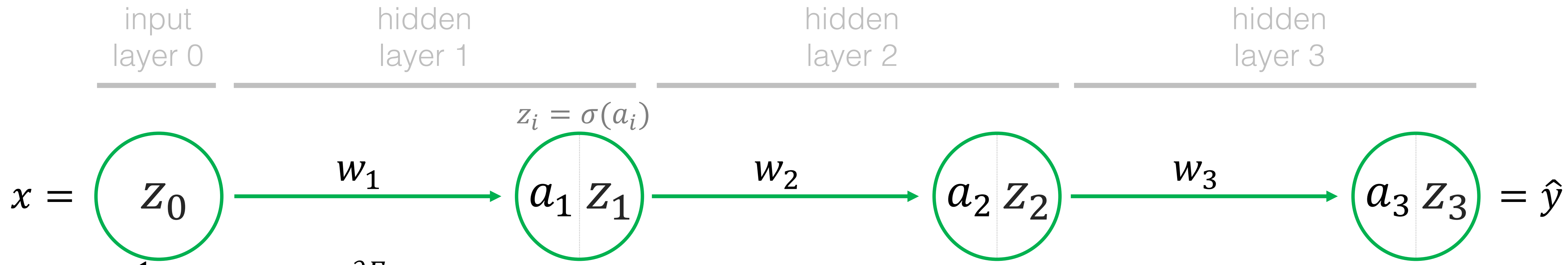
We can calculate the error:

$$E = \frac{1}{2}(\hat{y} - y)^2$$

We want to estimate the gradient with respect to each parameter

$$\frac{\partial E}{\partial w_i}$$

Backpropagation: an efficient way of calculating these values



Forward propagation

$$E = \frac{1}{2}(\hat{y} - y)^2$$

$$\hat{y} = \sigma(a_3)$$

$$a_3 = w_3 z_2$$

$$z_2 = \sigma(a_2)$$

$$a_2 = w_2 z_1$$

$$z_1 = \sigma(a_1)$$

$$a_1 = w_1 z_0$$

Backpropagation

$$\frac{\partial E}{\partial \hat{y}} = \hat{y} - \mathrm{y}$$

$$\frac{\partial \hat{y}}{\partial a_3} = \sigma'(a_3)$$

$$\frac{\partial a_3}{\partial z_2} = w_3$$

$$\frac{\partial z_2}{\partial a_2} = \sigma'(a_2)$$

$$\frac{\partial a_2}{\partial z_1} = w_2$$

$$\frac{\partial z_1}{\partial a_1} = \sigma'(a_1)$$

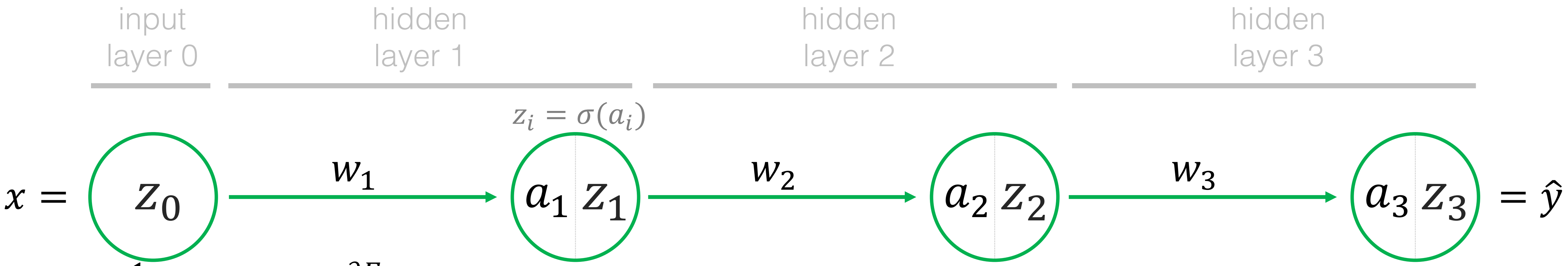
$$\frac{\partial a_1}{\partial w_1} = z_0 = x$$

Let's calculate $\frac{\partial E}{\partial w_1}$

$$\frac{\partial E}{\partial w_1} = \underline{\frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial z_2} \frac{\partial z_2}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial a_1}} \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial E}{\partial w_1} = \underline{(\hat{y} - \mathrm{y})\sigma'(a_3) w_3 \sigma'(a_2) w_2 \sigma'(a_1)} z_0$$

We know all these quantities from forward propagation



Forward propagation

$$E = \frac{1}{2}(\hat{y} - y)^2 \qquad \frac{\partial E}{\partial \hat{y}} = \hat{y} - y$$

$$\hat{y} = \sigma(a_3) \qquad \frac{\partial \hat{y}}{\partial a_3} = \sigma'(a_3)$$

$$a_3 = w_3 z_2 \qquad \frac{\partial a_3}{\partial z_2} = w_3$$

$$z_2 = \sigma(a_2) \qquad \frac{\partial z_2}{\partial a_2} = \sigma'(a_2)$$

$$a_2 = w_2 z_1 \qquad \frac{\partial a_2}{\partial z_1} = w_2$$

$$z_1 = \sigma(a_1) \qquad \frac{\partial z_1}{\partial a_1} = \sigma'(a_1)$$

$$a_1 = w_1 z_0 \qquad \frac{\partial a_1}{\partial w_1} = z_0 = x$$

Backpropagation

$$\frac{\partial E}{\partial w_1} = \underbrace{\frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial z_2} \frac{\partial z_2}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial a_1}}_{\frac{\partial E}{\partial a_1}} \frac{\partial a_1}{\partial w_1}$$

These derivatives with respect to the activations, a_i, allow us to quickly calculate each of our parameter derivatives:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial a_i} \frac{\partial a_i}{\partial w_{i-1}} = \underbrace{\frac{\partial E}{\partial a_i}}_{\delta_i} z_{i-1}$$

δ_i (common shorthand)

Backpropagation

1. Run forward propagation on an input and calculate all the activations, a_i

2. Evaluate $\delta_i = \frac{\partial E}{\partial a_i}$ for all nodes in the network

3. Compute the weight derivatives: $\frac{\partial E}{\partial w_{ij}} = \delta_i z_j$ for all nodes in the network

Now we have all the derivatives we need, so we can run gradient descent

Gradient Descent

Batch gradient descent

1. Calculate the average error across all the training observations $$E = \frac{1}{2N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$
2. Update all the parameters based on that error $$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$
3. Repeat 1 and 2 until convergence

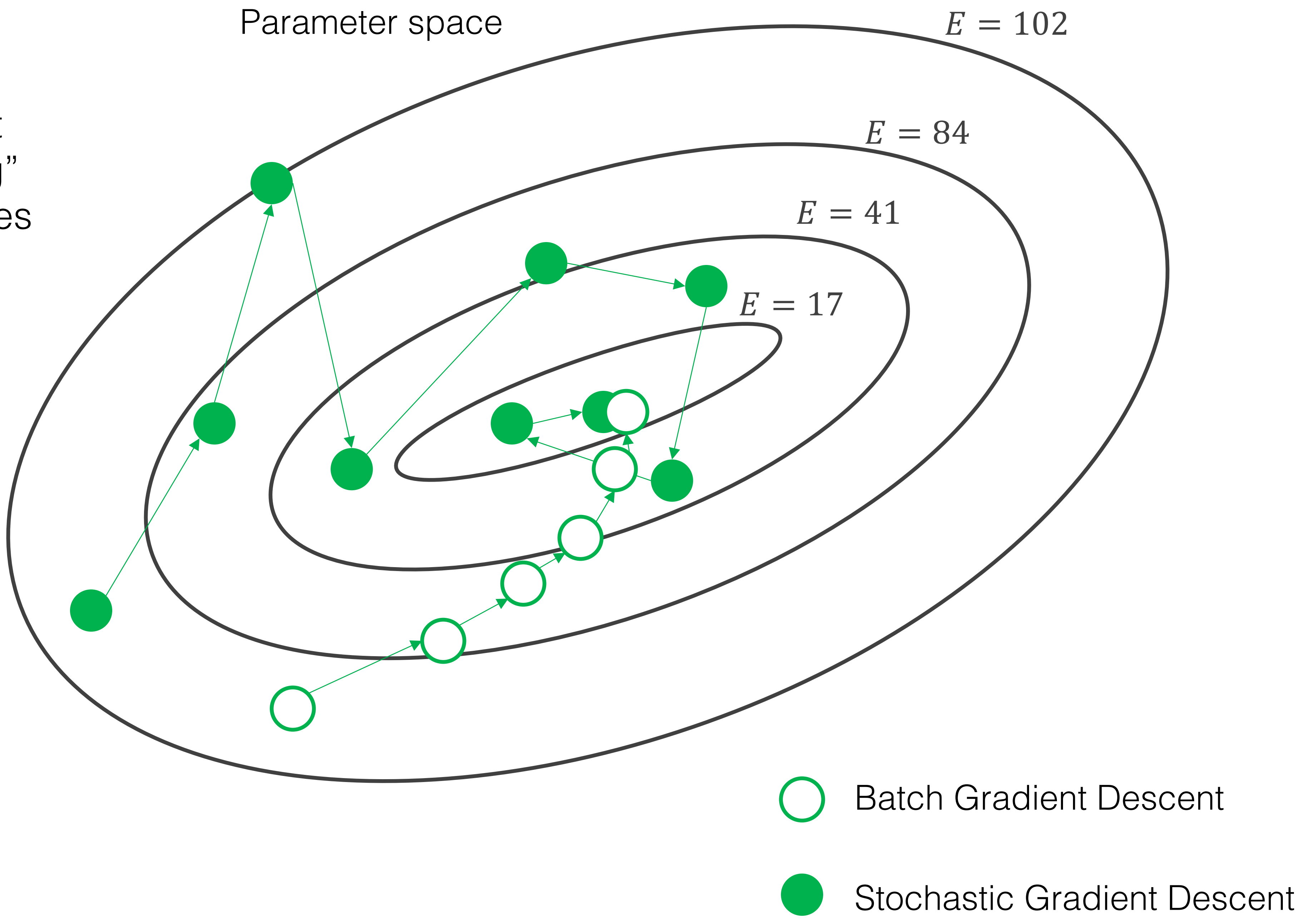
Stochastic gradient descent

1. Randomly sort the list of training observations
2. Calculate the error from one training sample $$E = \frac{1}{2}(\hat{y}_n - y_n)^2$$
3. Update all the parameters based on that error $$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$
4. Repeat 2 and 3 until all training samples have been used, then repeat 1-3 until convergence

Stochastic gradient descent (SGD) is better at "exploring" nonconvex parameter spaces

Batch gradient descent is rarely used in practiced because it's too computationally expensive

Often **minibatch** gradient descent is used (a small batch of data is used instead of a single sample in SGD)

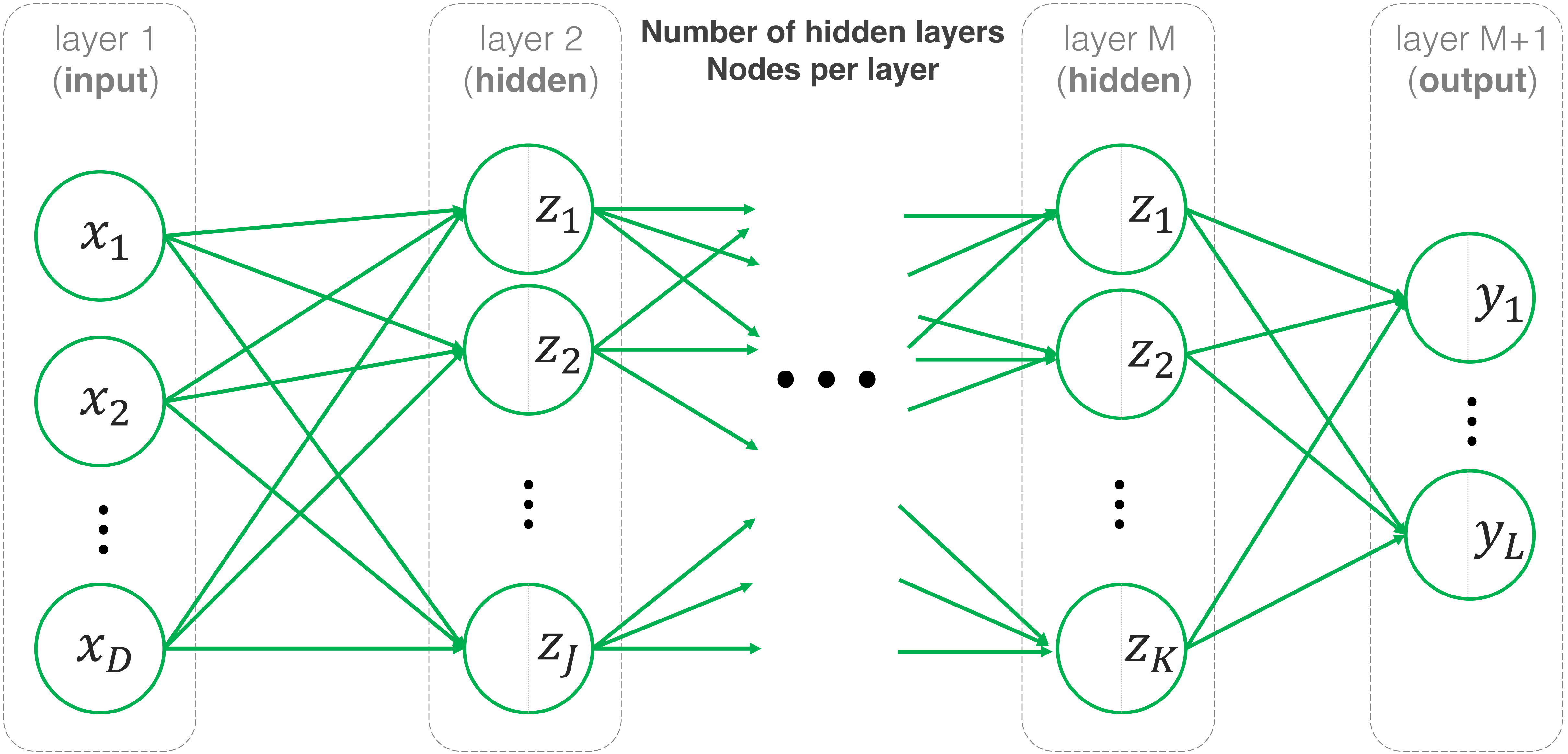


What is a neural network and **how does it work**?

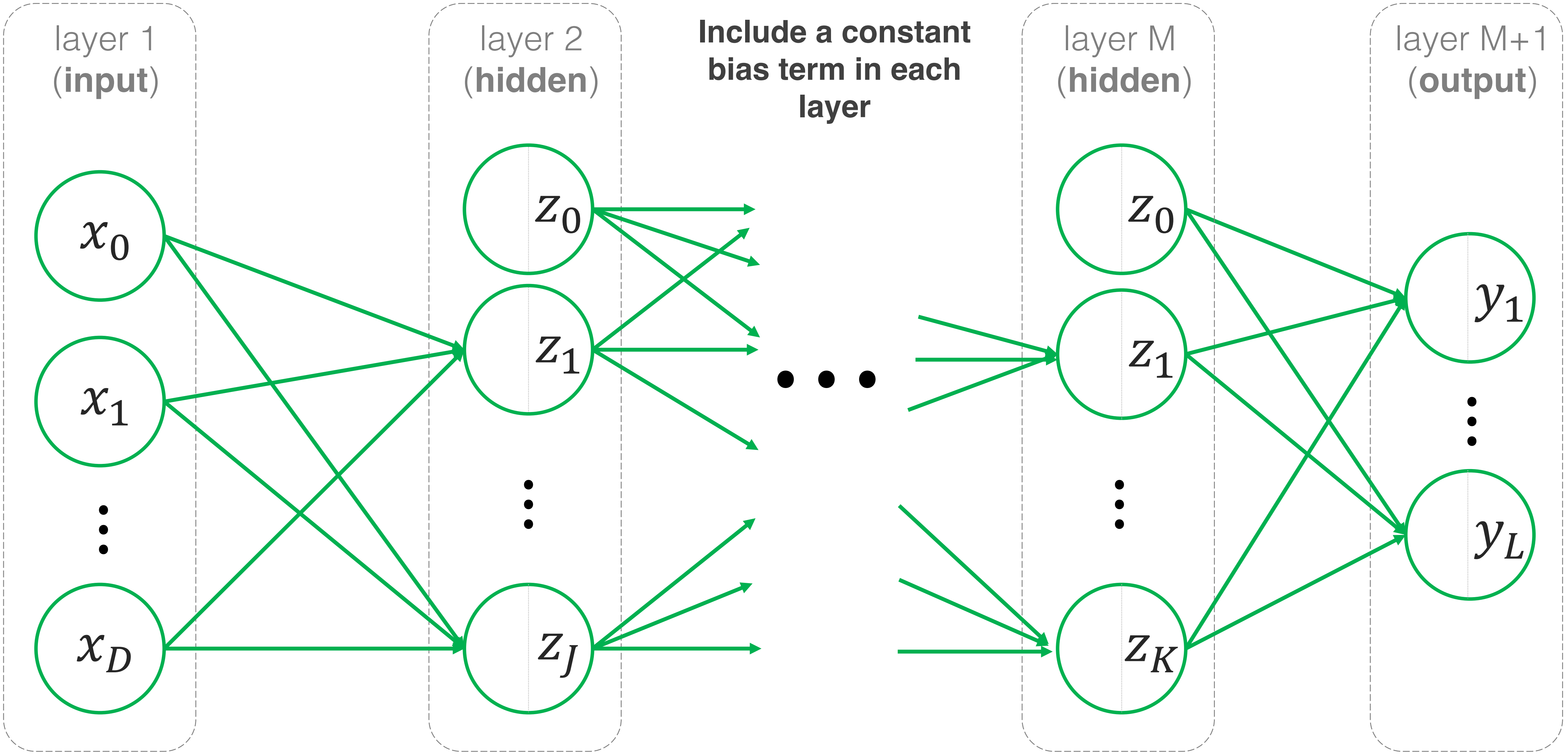
How do we **choose model weights**?
(i.e. how do we fit our model to data)

What are the challenges of using neural networks?

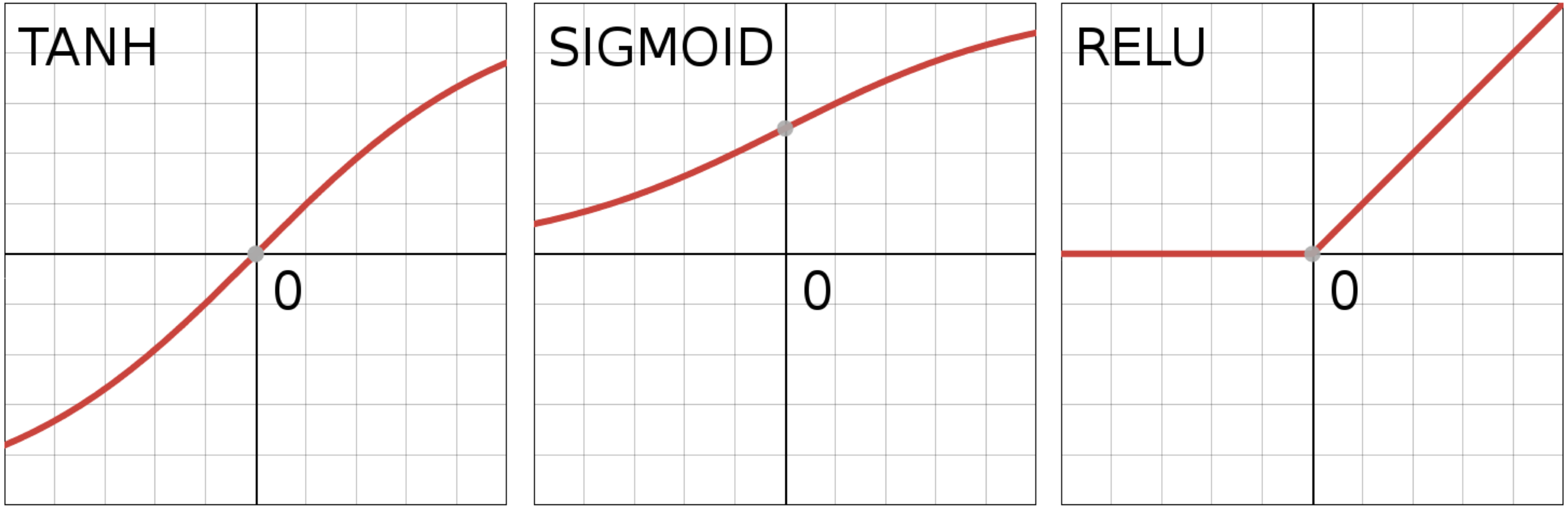
Architectural choices



Architectural choices



Activation Functions



Hyperbolic Tangent

Sigmoid Tangent

Rectified linear unit (ReLU)

Speeds up training and helps prevent vanishing gradients

Image from Danijar Hafner, Quora

Weight initialization

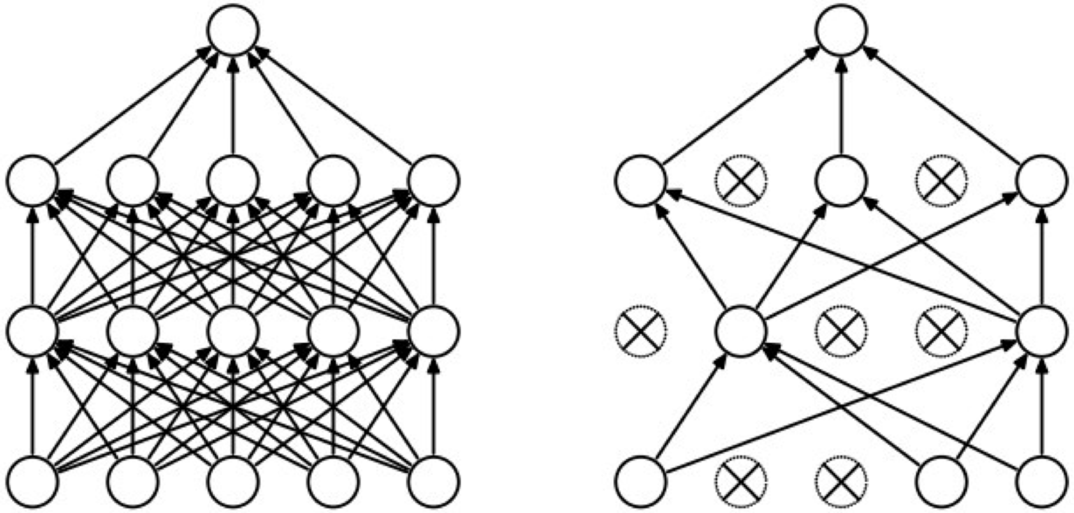
Set all parameters to zeros	Bad idea: leads to too much symmetry causing many gradients to be the same and the parameters will tend to all update the same way
Small random numbers	Better than all zeros, but may lead to the **vanishing gradient** problem during backpropagation
Batch normalization	Ensures activations are unit Gaussian at each layer by inserting a batch normalization layer

Regularization

L2 Regularization

L1 Regularization

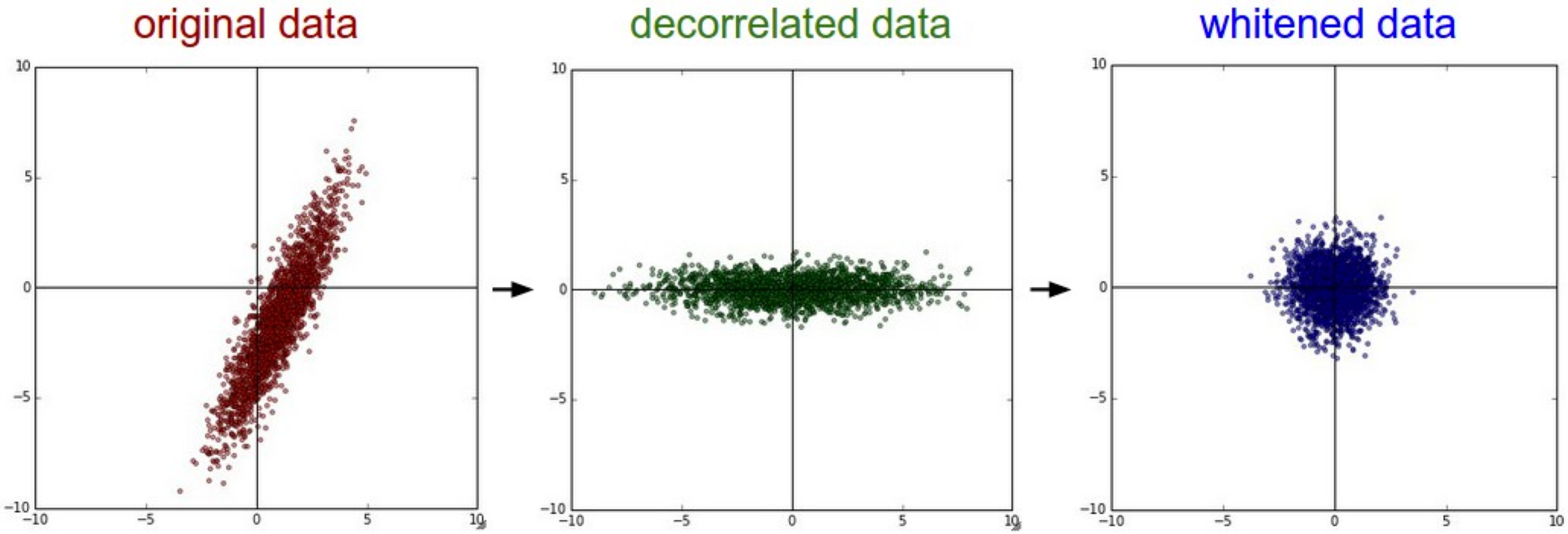
Dropout



(a) Standard Neural Net (b) After applying dropout.

While training, keep a neuron active with some probability p, or setting it to zero otherwise.

Data preprocessing



PCA and whitening (zero mean unit variance for all features)

Supervised Learning Techniques

Covered so far

- K-Nearest Neighbors
- Linear regression
- Perceptron
- Logistic Regression
- Fisher's Linear Discriminant / Linear Discriminant Analysis
- Quadratic Discriminant Analysis
- Naïve Bayes
- Decision Trees and Random Forests
- Ensemble methods (bagging, boosting, stacking)
- Neural Networks

Rely on a linear combination of weights and features: $\boldsymbol{w}^T\boldsymbol{x}$