

Neural Networks II

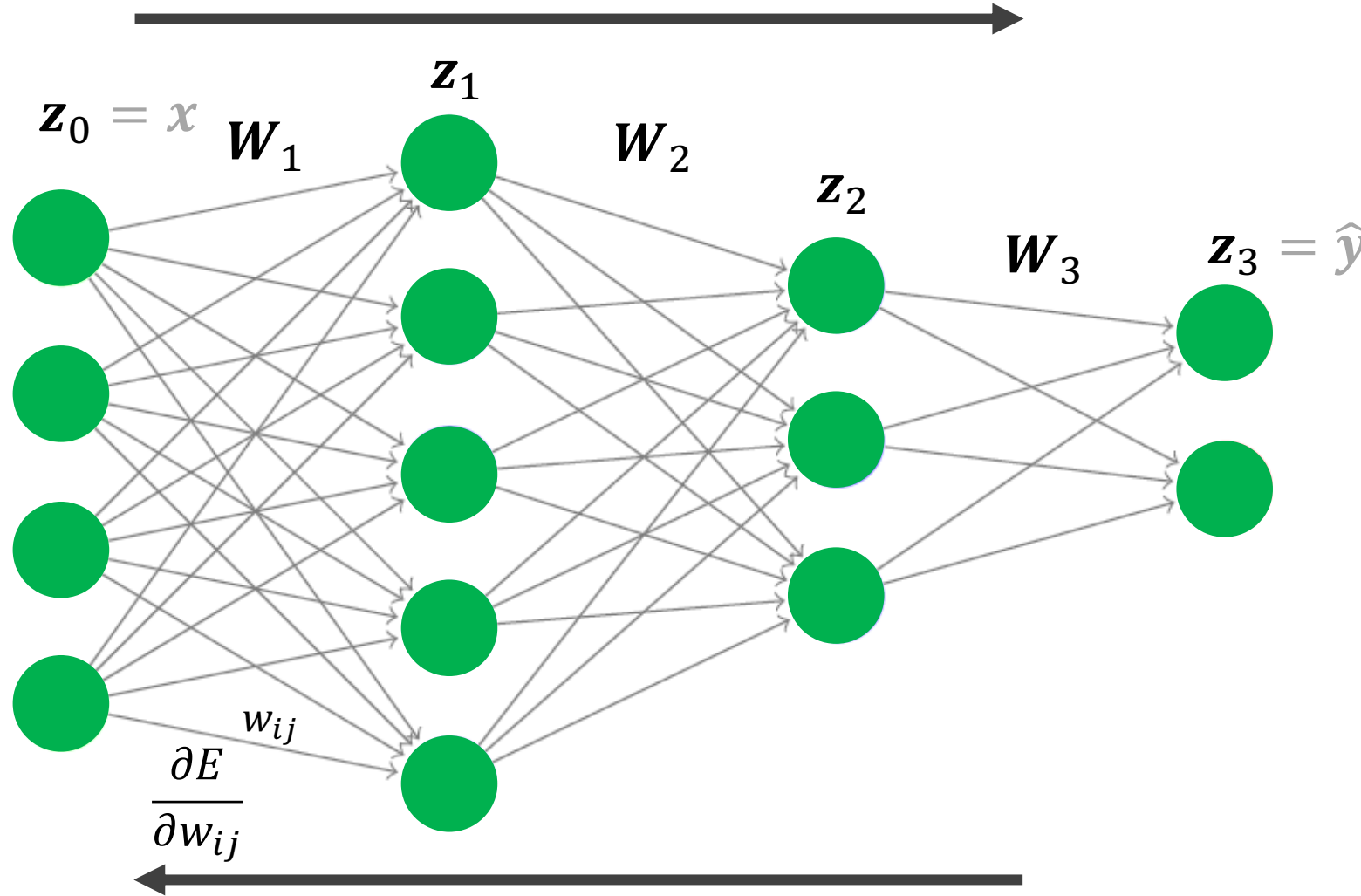
Lecture 17

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



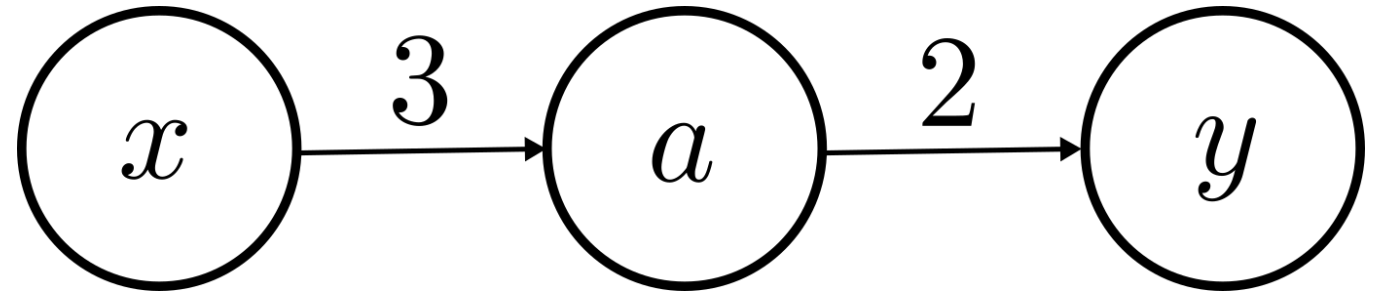
$$E = \frac{1}{2} \sum_k (\hat{y}_k - y_k)^2$$

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

Example #1



$$y = 2a \quad \frac{\partial y}{\partial a} = 2$$

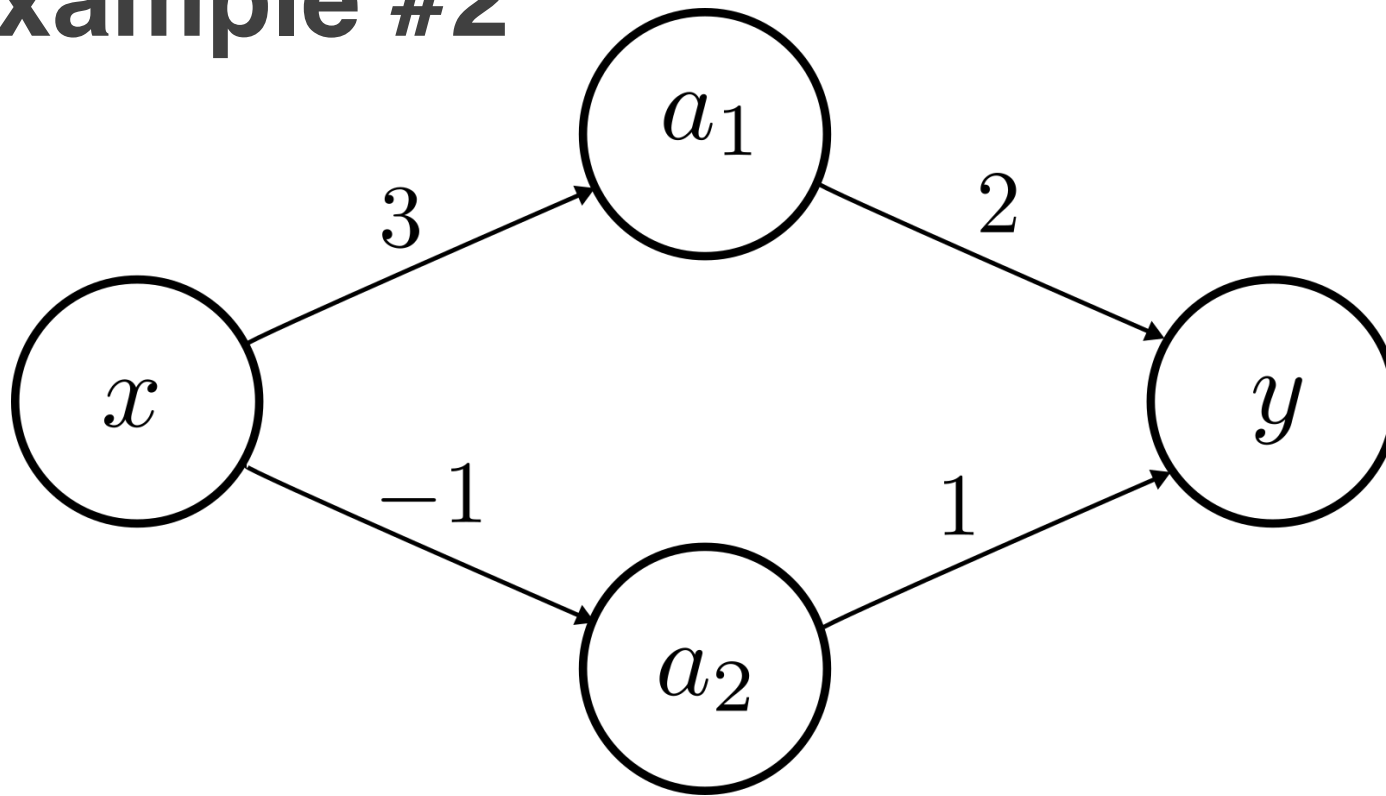
$$a = 3x \quad \frac{\partial a}{\partial x} = 3$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial a} \frac{\partial a}{\partial x} = (2)(3) = 6$$

Chain Rule

Along a path we apply the chain rule

Example #2



$$y = 2a_1 + a_2$$

$$a_1 = 3x$$

$$a_2 = -x$$

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x} (2a_1 + 1a_2)$$

Sum Rule

Across paths we apply the sum rule

$$= (2) \frac{\partial a_1}{\partial x} + (1) \frac{\partial a_2}{\partial x}$$

Chain Rule

$$= \frac{\partial y}{\partial a_1} \frac{\partial a_1}{\partial x} + \frac{\partial y}{\partial a_2} \frac{\partial a_2}{\partial x}$$

$$= (2)(3) + (1)(-1)$$

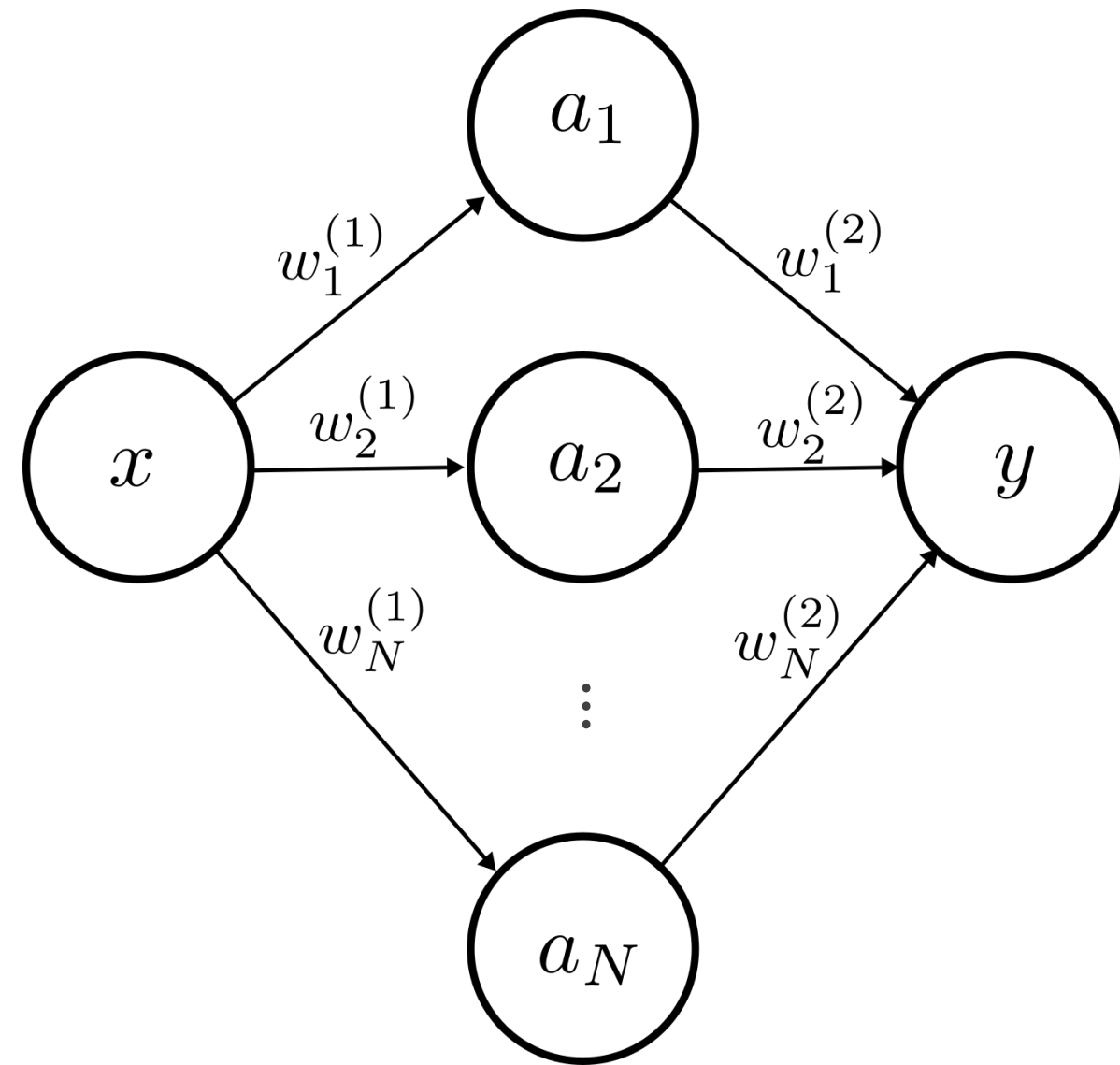
$$= 5$$

Example #3

$$y = \sum_{j=1}^N w_j^{(2)} a_j \quad \frac{\partial y}{\partial a_i} = w_i^{(2)}$$

$$a_i = w_i^{(1)} x \quad \frac{\partial a_i}{\partial x} = w_i^{(1)}$$

$$\frac{\partial y}{\partial x} = \sum_{j=1}^N \frac{\partial y}{\partial a_j} \frac{\partial a_j}{\partial x} = \sum_{j=1}^N w_j^{(2)} w_j^{(1)}$$



Example #3

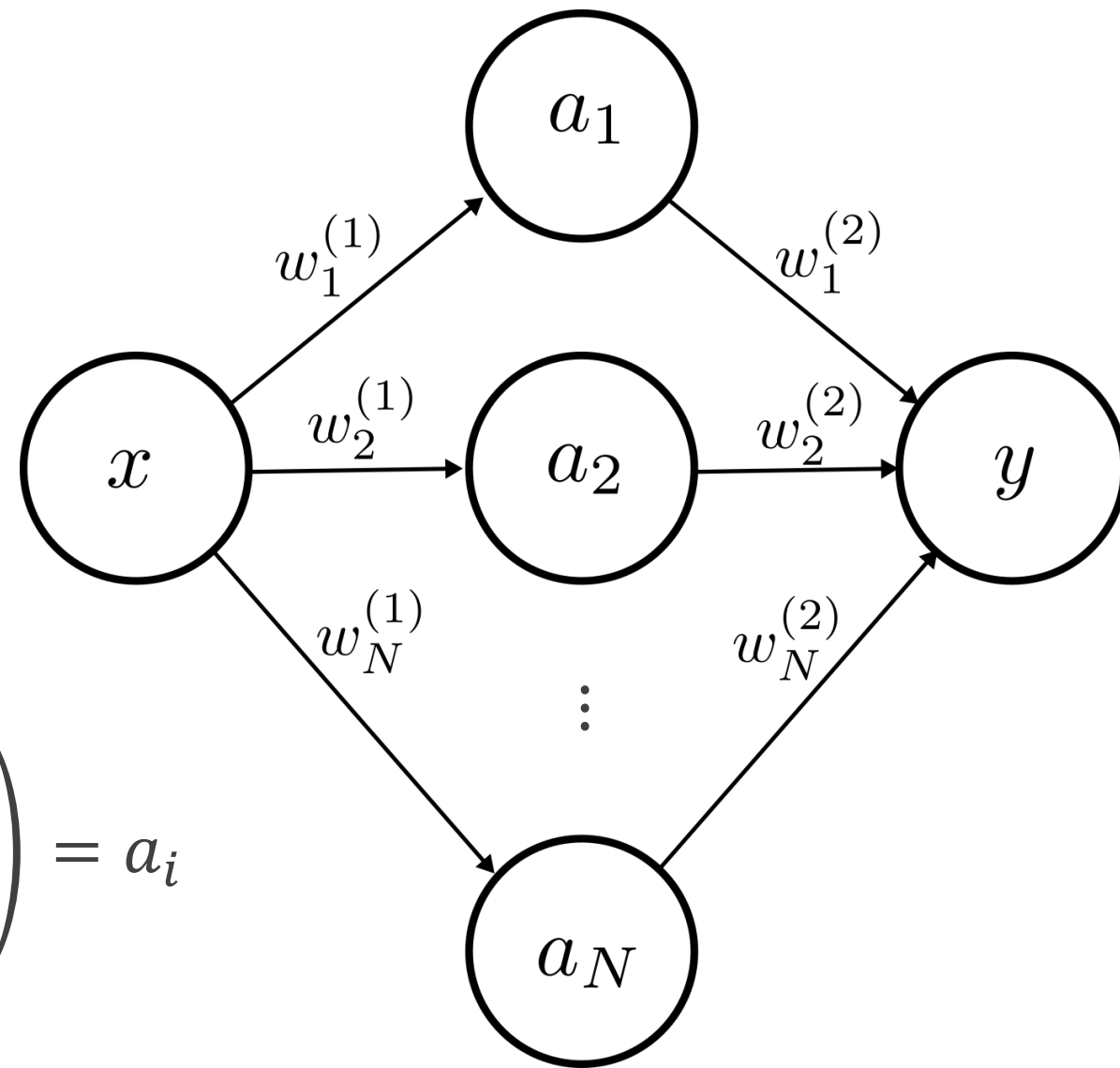
$$y = \sum_{j=1}^N w_j^{(2)} a_j \quad \frac{\partial y}{\partial a_i} = w_i^{(2)}$$

$$a_i = w_i^{(1)} x \quad \frac{\partial a_i}{\partial x} = w_i^{(1)}$$

Derivatives with respect to the weights:

$$\frac{\partial y}{\partial w_i^{(2)}} = \frac{\partial y}{\partial a_i} \frac{\partial a_i}{\partial w_i^{(2)}} = \frac{\partial}{\partial w_i^{(2)}} \left(\sum_{j=1}^N w_j^{(2)} a_j \right) = a_i$$

$$\frac{\partial y}{\partial w_i^{(1)}} = \frac{\partial y}{\partial a_i} \frac{\partial a_i}{\partial w_i^{(1)}} = w_i^{(2)} x$$



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

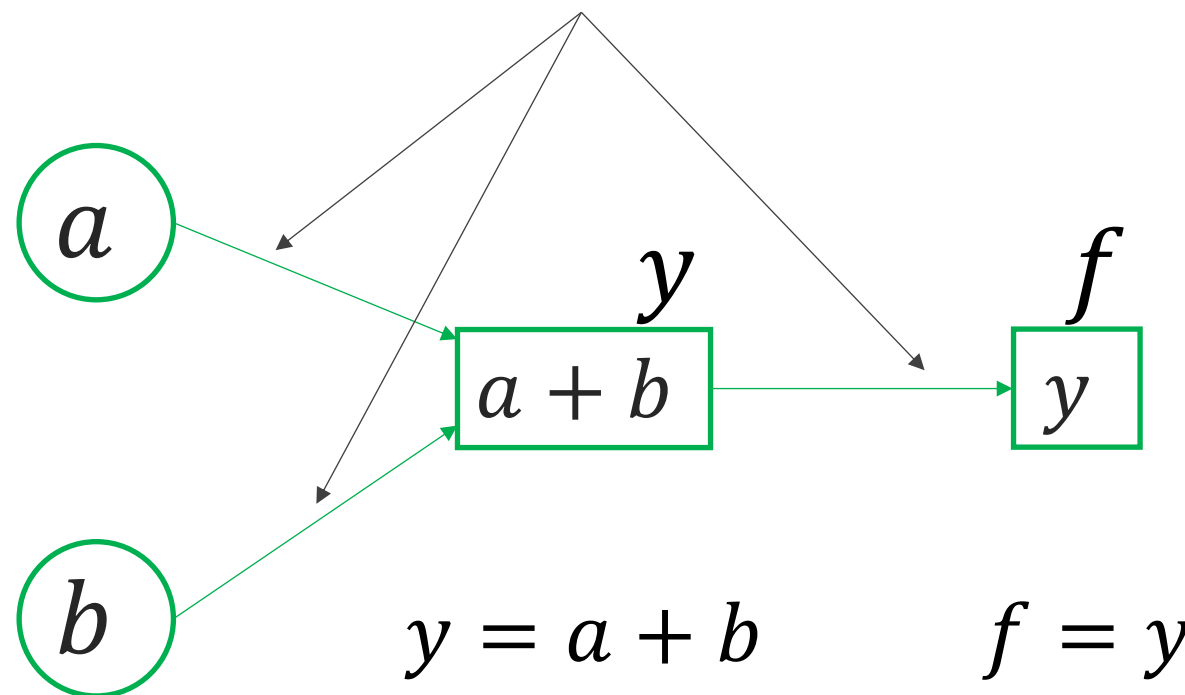
$$\frac{\partial f}{\partial z} = \underbrace{\frac{\partial f}{\partial w} \frac{\partial w}{\partial x}}_{\frac{\partial f}{\partial x}} \frac{\partial x}{\partial y} \frac{\partial y}{\partial z} = \underbrace{\frac{\partial f}{\partial x} \frac{\partial x}{\partial y}}_{\frac{\partial f}{\partial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

Simple example

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

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

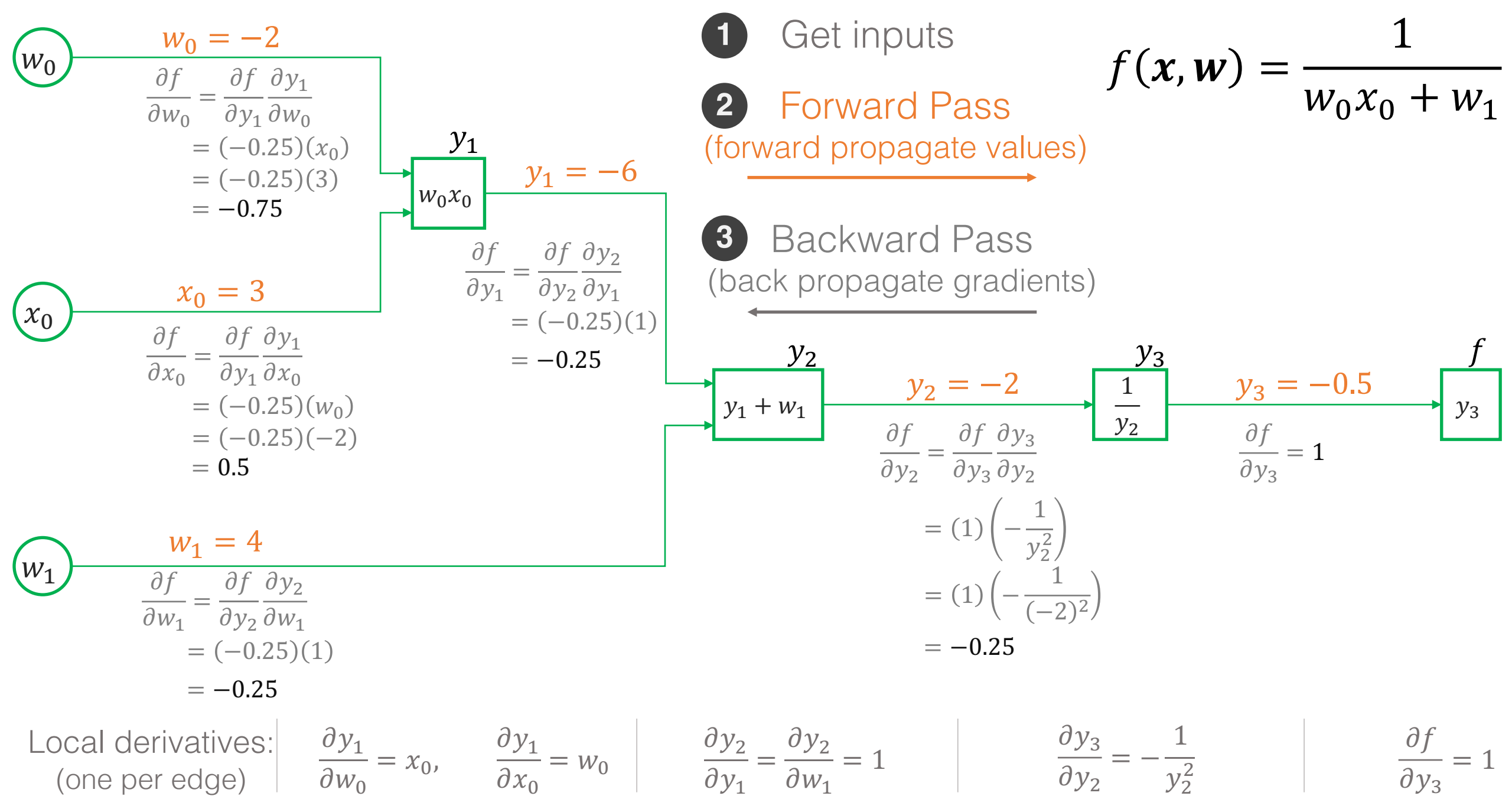
This graph to the right represents this function



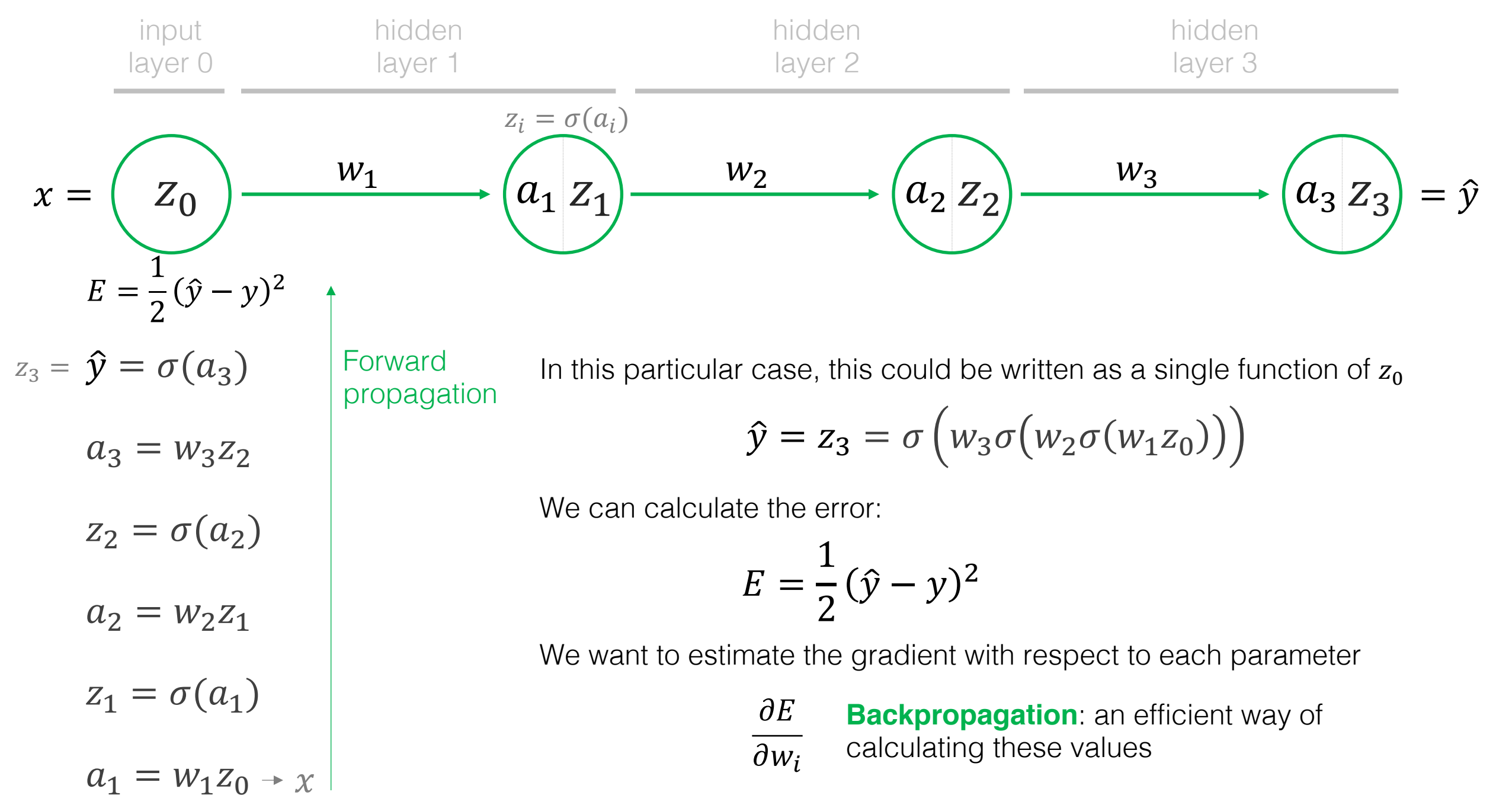
Name of the variable at that node

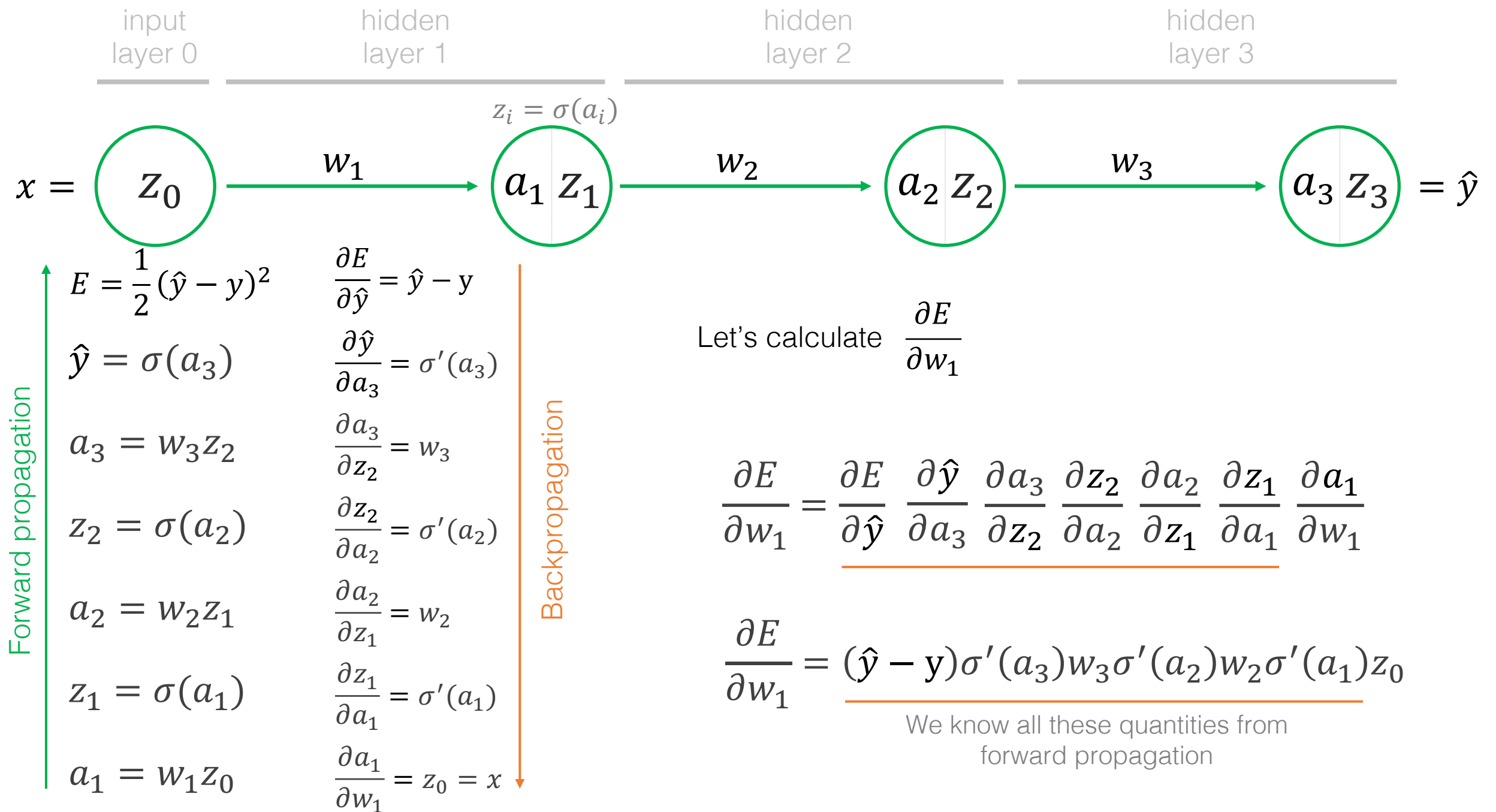
Operation that the node performs

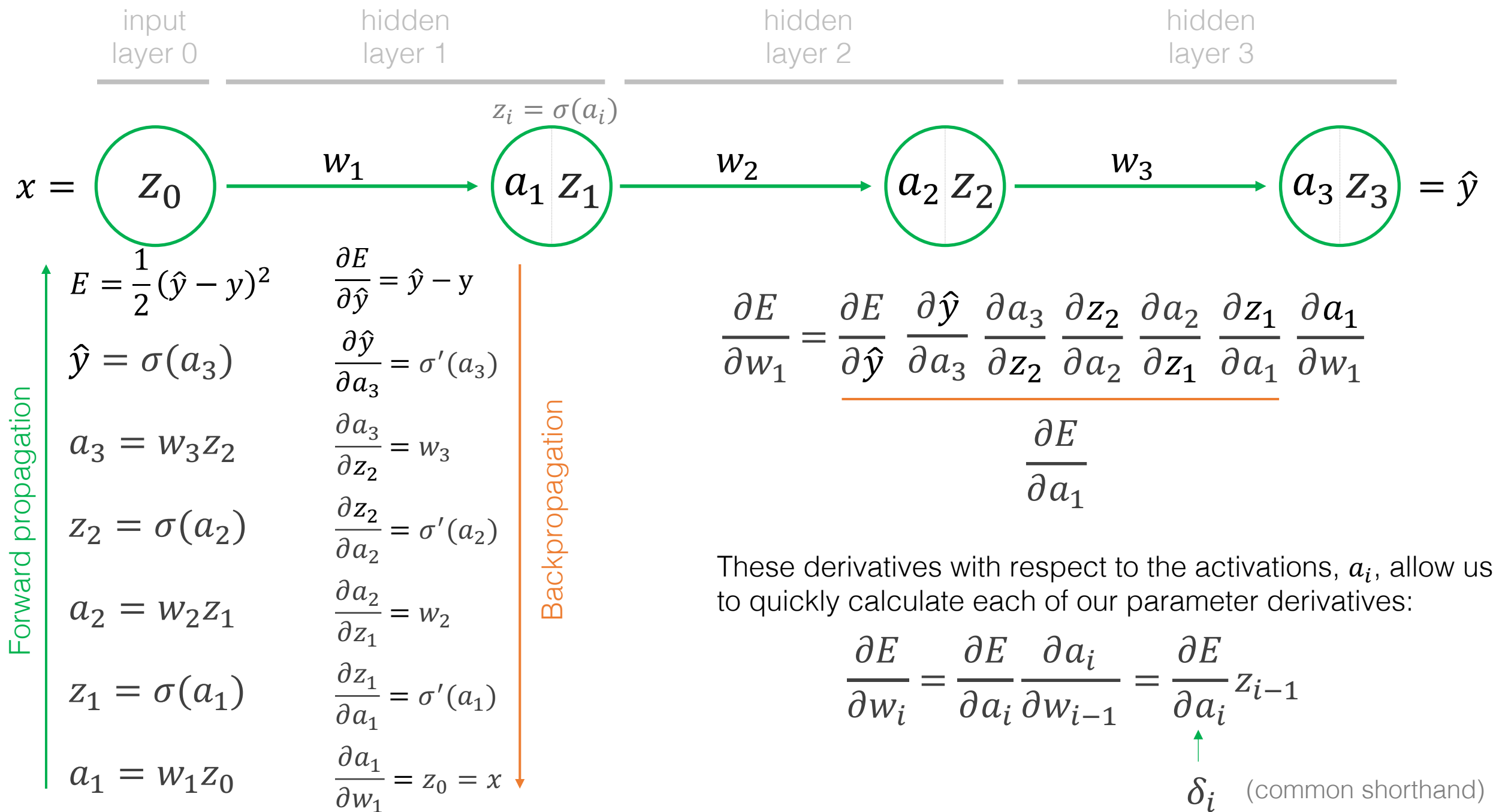
Local derivatives (one for each edge input into a node): $\frac{\partial y}{\partial a} = 1, \quad \frac{\partial y}{\partial b} = 1$



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







Dive deeper into neural network math

5.1 Forward Propagation

Forward propagation is the iterative application of the following two equations:

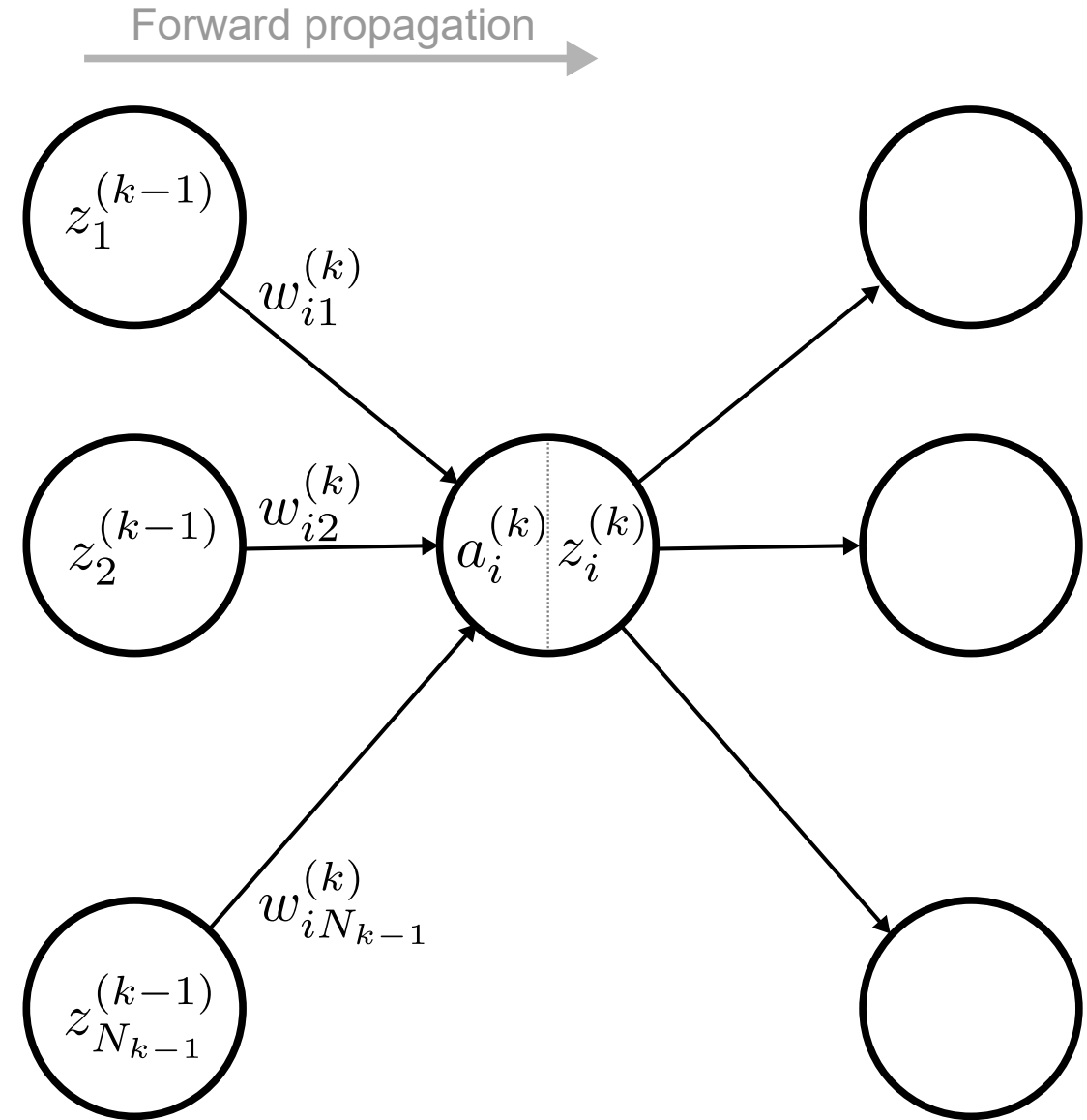
$$a_i^{(k)} = \sum_{j=1}^{N_{k-1}} w_{ij}^{(k)} z_j^{(k-1)}$$

$$z_i^{(k)} = \sigma(a_i^{(k)})$$

In matrix form those equations are:

$$\mathbf{a}^{(k)} = \mathbf{W}^{(k)} \mathbf{z}^{(k-1)}$$

$$\mathbf{z}^{(k)} = \sigma(\mathbf{a}^{(k)})$$



https://github.com/kylebradbury/neural-network-math/raw/master/neural_network_math.pdf

5.2 Backpropagation

Backpropagation begins with the calculation of the gradient of the error with respect to the final set of activations, which for mean square error with sigmoidal activation and K -layer neural network is:

$$\delta_i^{(K)} \triangleq \frac{\partial E_n}{\partial a_i^{(K)}} = (z_i^{(3)} - y_i) \sigma'(a_i^{(3)})$$

We then propagate that back through the neural network and calculate the gradients with respect to each weight along the way (for $k = K - 1, \dots, 1$):

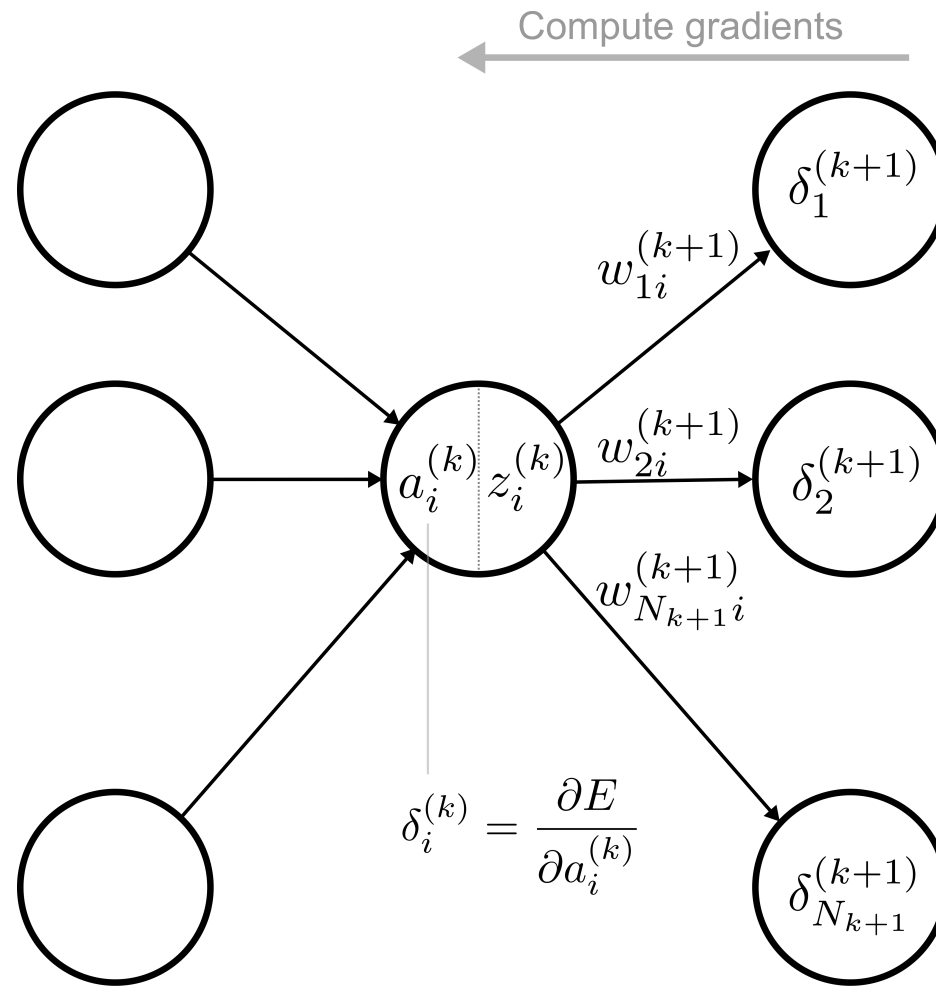
$$\delta_i^{(k)} \triangleq \frac{\partial E_n}{\partial a_i^{(k)}} = \sigma'(a_i^{(k)}) \sum_{j=1}^{N_{k+1}} \delta_j^{(k+1)} w_{ji}^{(k+1)}$$

$$\frac{\partial E_n}{\partial w_{ij}^{(k)}} = \frac{\partial E_n}{\partial a_i^{(k)}} \frac{\partial a_i^{(k)}}{\partial w_{ij}^{(k)}} = \delta_i^{(k)} z_j^{(k-1)}$$

Or in matrix form:

$$\boldsymbol{\delta}^{(k)} \triangleq \frac{\partial E_n}{\partial \mathbf{a}^{(k)}} = \mathbf{W}^{(k+1)\top} \boldsymbol{\delta}^{(k+1)} \circ \sigma'(\mathbf{a}^{(k)})$$

$$\frac{\partial E_n}{\partial \mathbf{w}^{(k)}} = \boldsymbol{\delta}^{(k)} \mathbf{z}^{(k-1)\top}$$



https://github.com/kylebradbury/neural-network-math/raw/master/neural_network_math.pdf

Backpropagation

- 1 Run forward propagation on an input and calculate all the activations, a_i
- 2 Evaluate $\delta_i^{(k)} = \frac{\partial E}{\partial a_i^{(k)}}$ for all nodes in the network
- 3 Compute the weight derivatives: $\frac{\partial E}{\partial w_{ij}^{(k)}} = \delta_i^{(k)} z_j^{(k-1)}$ 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 gradient for each sample and average them
- 2 Update all the parameters based on that average gradient
- 3 Repeat 1 and 2 until convergence

$$\overline{\frac{\partial E}{\partial w_{ij}}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial E}{\partial w_{ij}}$$
$$w_{ij} \leftarrow w_{ij} - \eta \overline{\frac{\partial E}{\partial w_{ij}}}$$

Stochastic gradient descent

- 1 Randomly sort the list of training observations
- 2 Calculate the gradient from one training sample
- 3 Update all the parameters based on that error
- 4 Repeat 2 and 3 until all training samples have been used, then repeat 1-3 until convergence

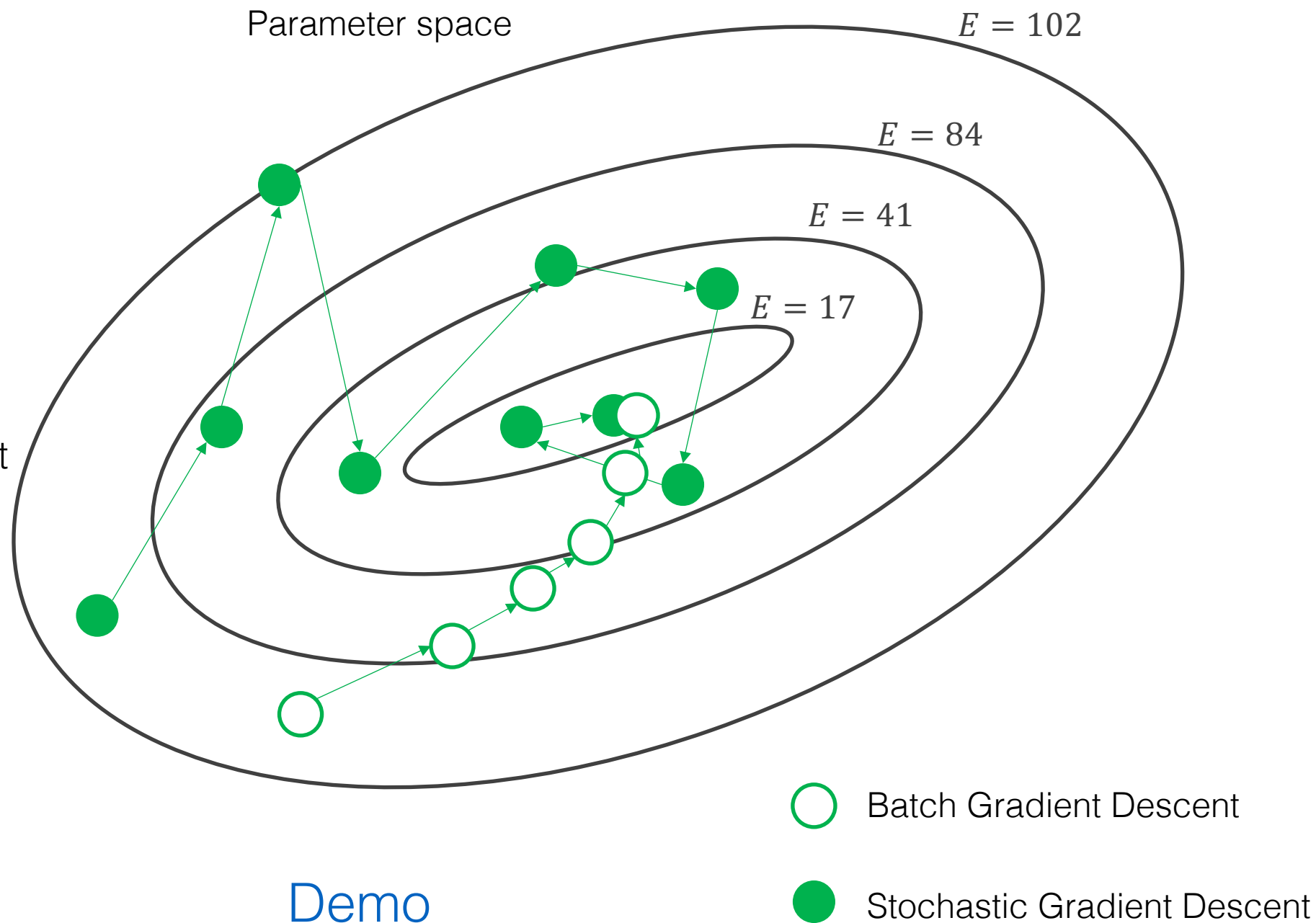
$$\frac{\partial E}{\partial w_{ij}}$$
$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

Minibatch gradient descent

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

Stochastic gradient descent (SGD) is better at avoiding local minima

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?

Successfully training neural networks

Advice from Andrej Karpathy: <http://karpathy.github.io/2019/04/25/recipe/>

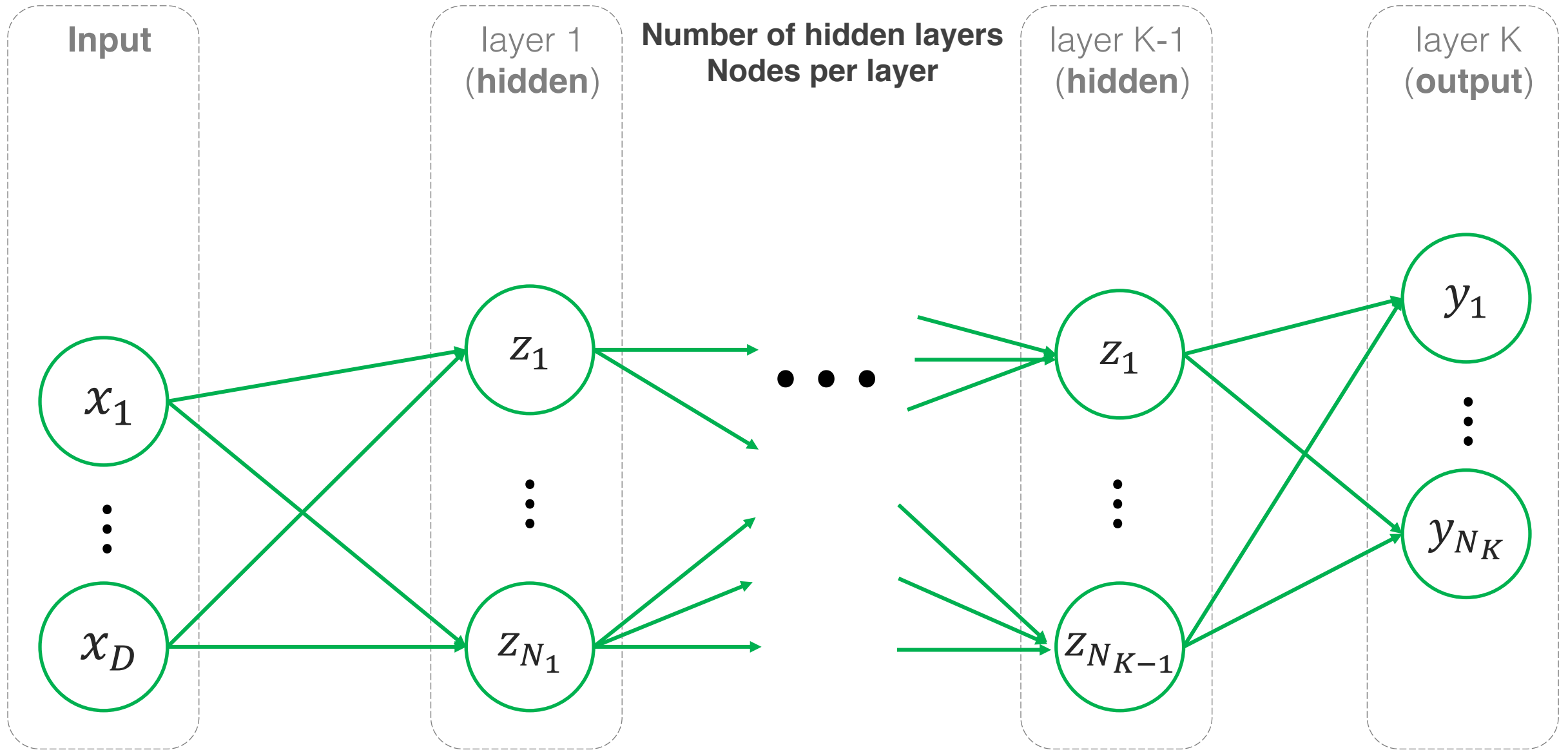
Challenges

1. Neural network training is not plug and play.
You need to understand the methods.
2. It is difficult to tell when there is a mistake

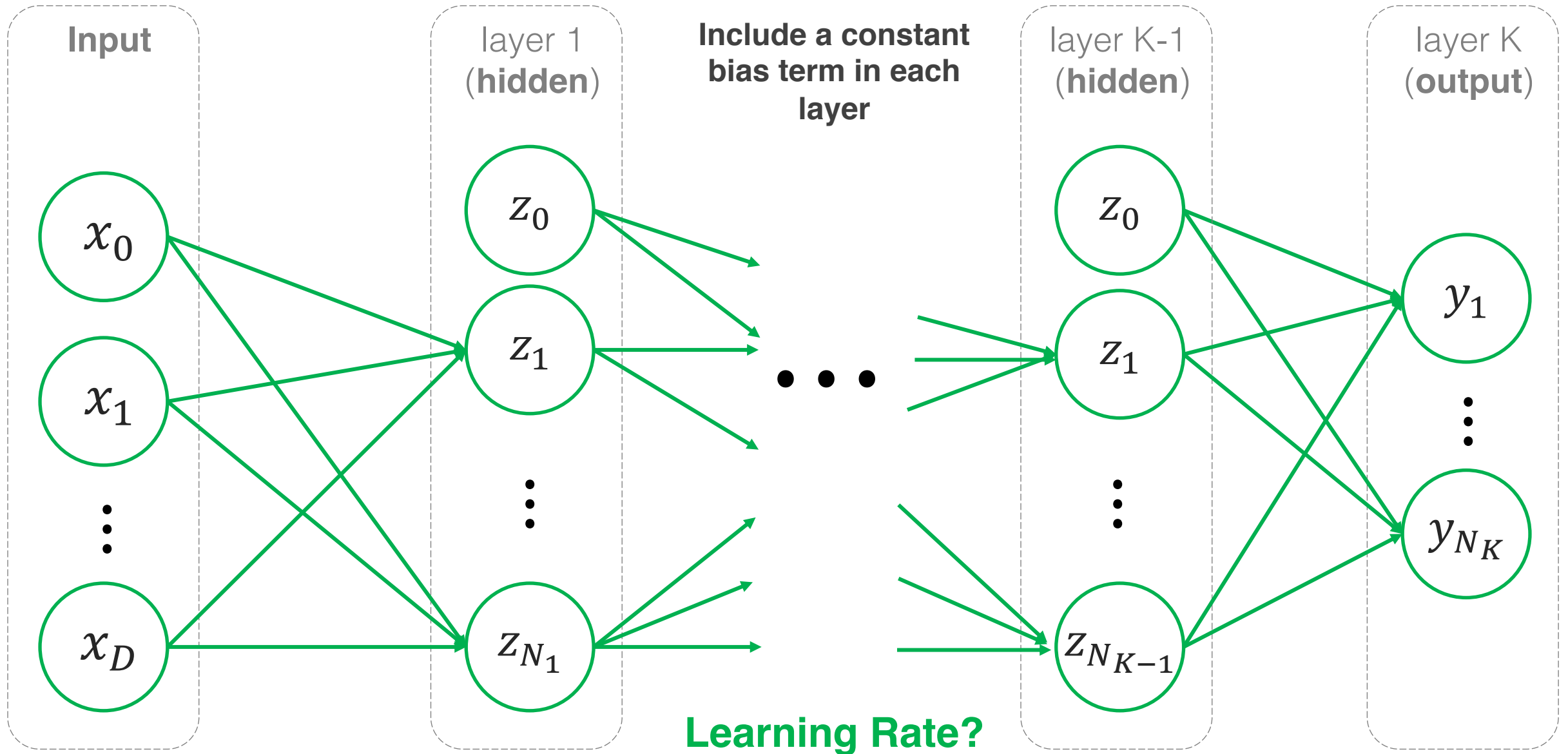
Recipe for training

1. Understand your data
2. Setup an end-to-end training/evaluation pipeline and test simple baselines
3. Overfit your model to the data to make sure you can do it
4. Regularize the model
 1. Add data
 2. Augmentation
 3. Use dropout
 4. Early stopping
5. Tune your model (identify hyperparameters)
6. "Squeeze out the juice"
 1. Model ensembles
 2. Let the model train longer

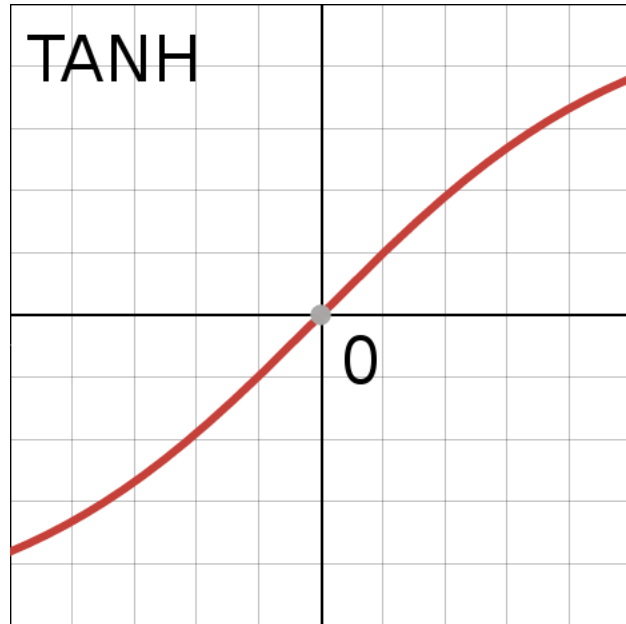
Hyperparameter / Architectural choices



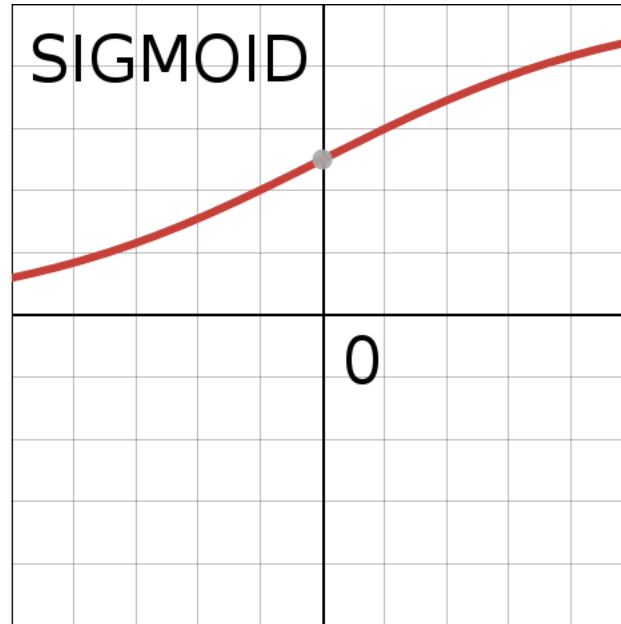
Hyperparameter / Architectural choices



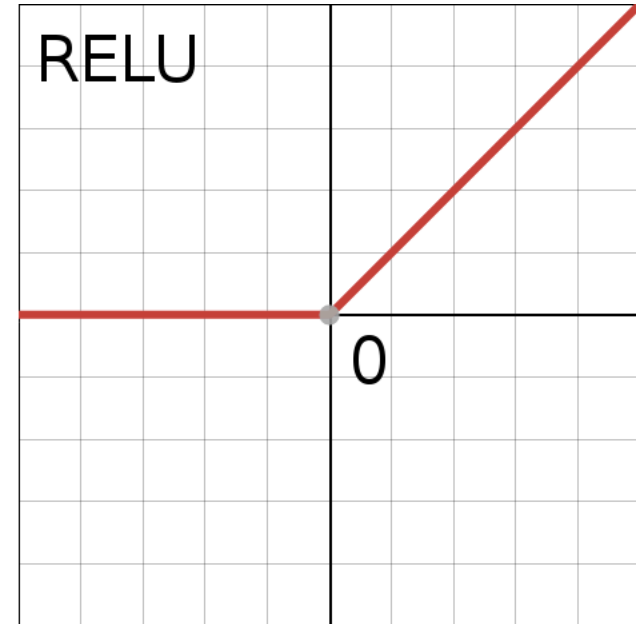
Activation Functions



Hyperbolic Tangent



Sigmoid Tangent



Rectified linear unit (ReLU)

Helps prevent vanishing
gradients; adds sparsity;
increases speed

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

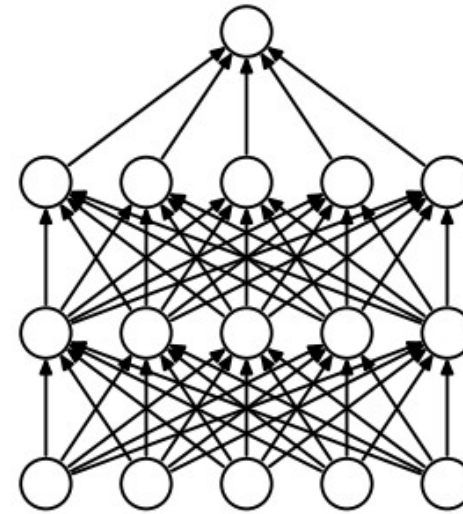
Regularization

L2 Regularization

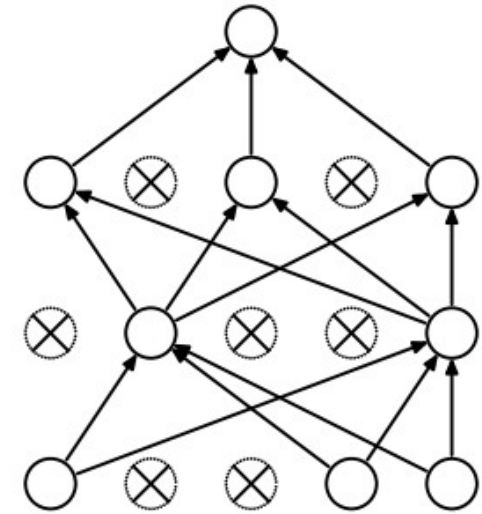
L1 Regularization

Dropout

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



(a) Standard Neural Net



(b) After applying dropout.

Supervised Learning Techniques

Covered so far

- Linear Regression
- K-Nearest Neighbors
- 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

Appropriate for:
● Classification
● Regression

Can be used with many machine learning techniques