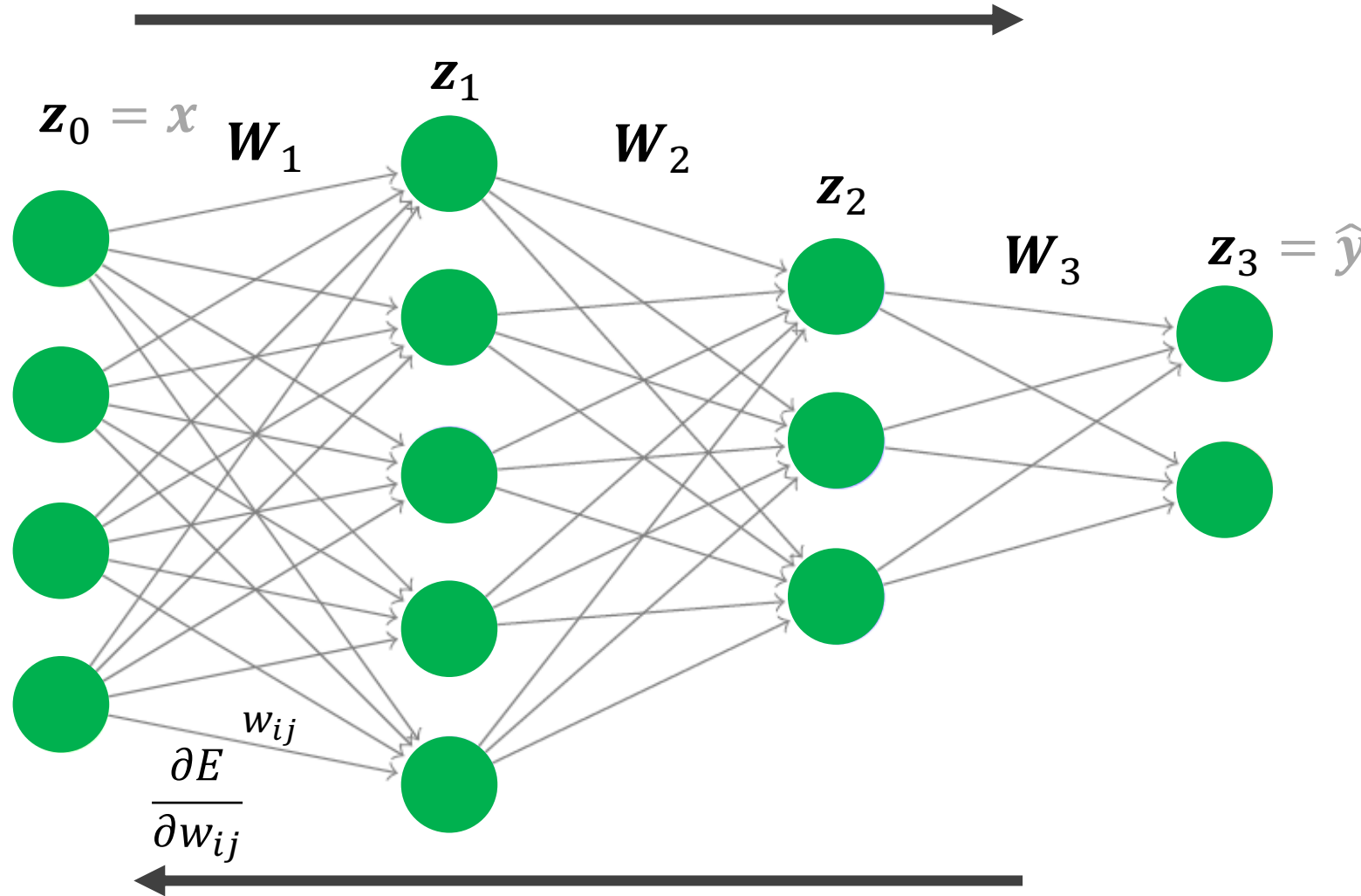# Neural Networks II

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 / cost

$$z_0 = x$$

$$W_1$$

$$z_1$$

$$W_2$$

$$z_2$$

$$W_3$$

$$z_3 = \hat{y}$$

$$w_{ij}$$

$$\frac{\partial E}{\partial w_{ij}}$$

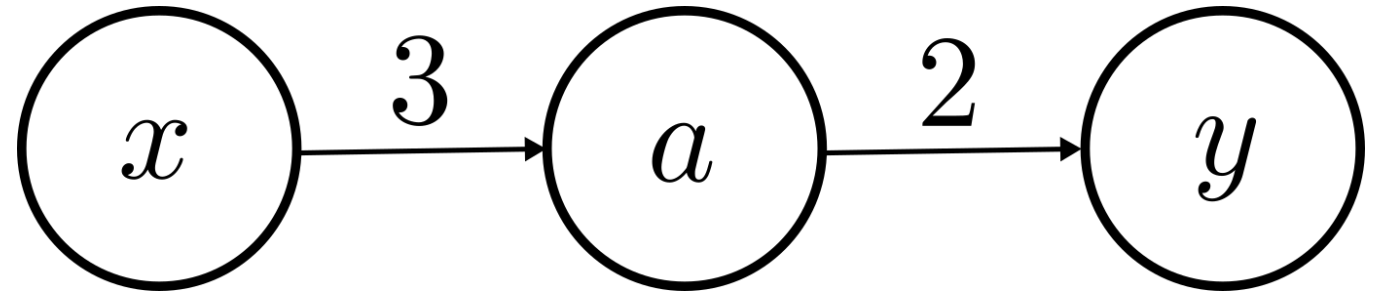$$E = \frac{1}{2} \sum_k (\hat{y}_k - y_k)^2$$

Backpropagation lets us **compute the gradient** with respect to each of the parameters so we can tune them with **gradient descent**

(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

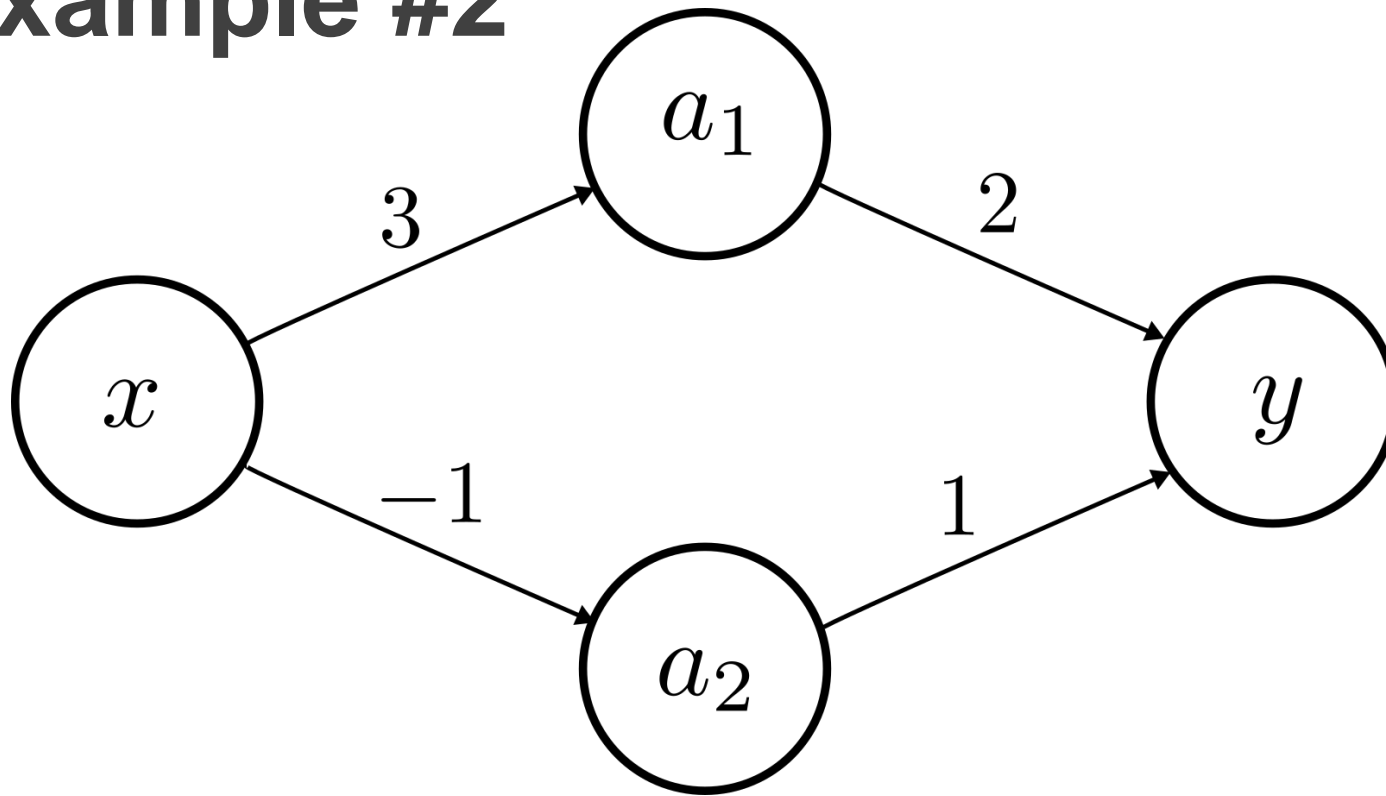$$y = 2a \qquad \frac{\partial y}{\partial a} = 2$$

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

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

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

$$= \frac{\partial y}{\partial a_1}\frac{\partial a_1}{\partial x} + \frac{\partial y}{\partial a_2}\frac{\partial a_2}{\partial x}$$
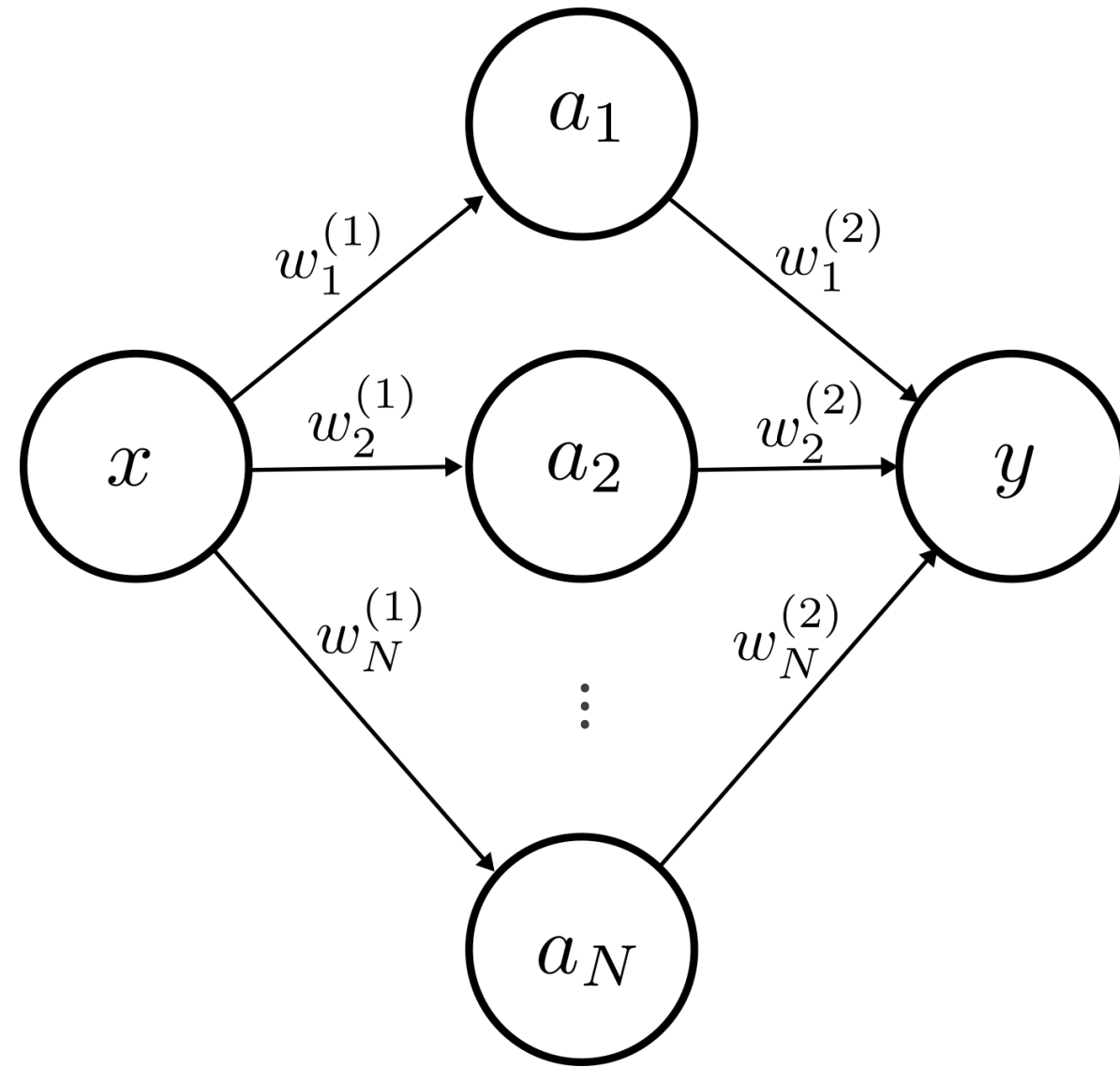
$$= (2)(3) + (1)(-1)$$

$$= 5$$

Example #3

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

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

$$\frac{\partial y}{\partial x} = \sum_{j=1}^{N} \frac{\partial y}{\partial a_j} \frac{\partial a_j}{\partial x} = \sum_{j=1}^{N} w_j^{(2)} w_j^{(1)}$$
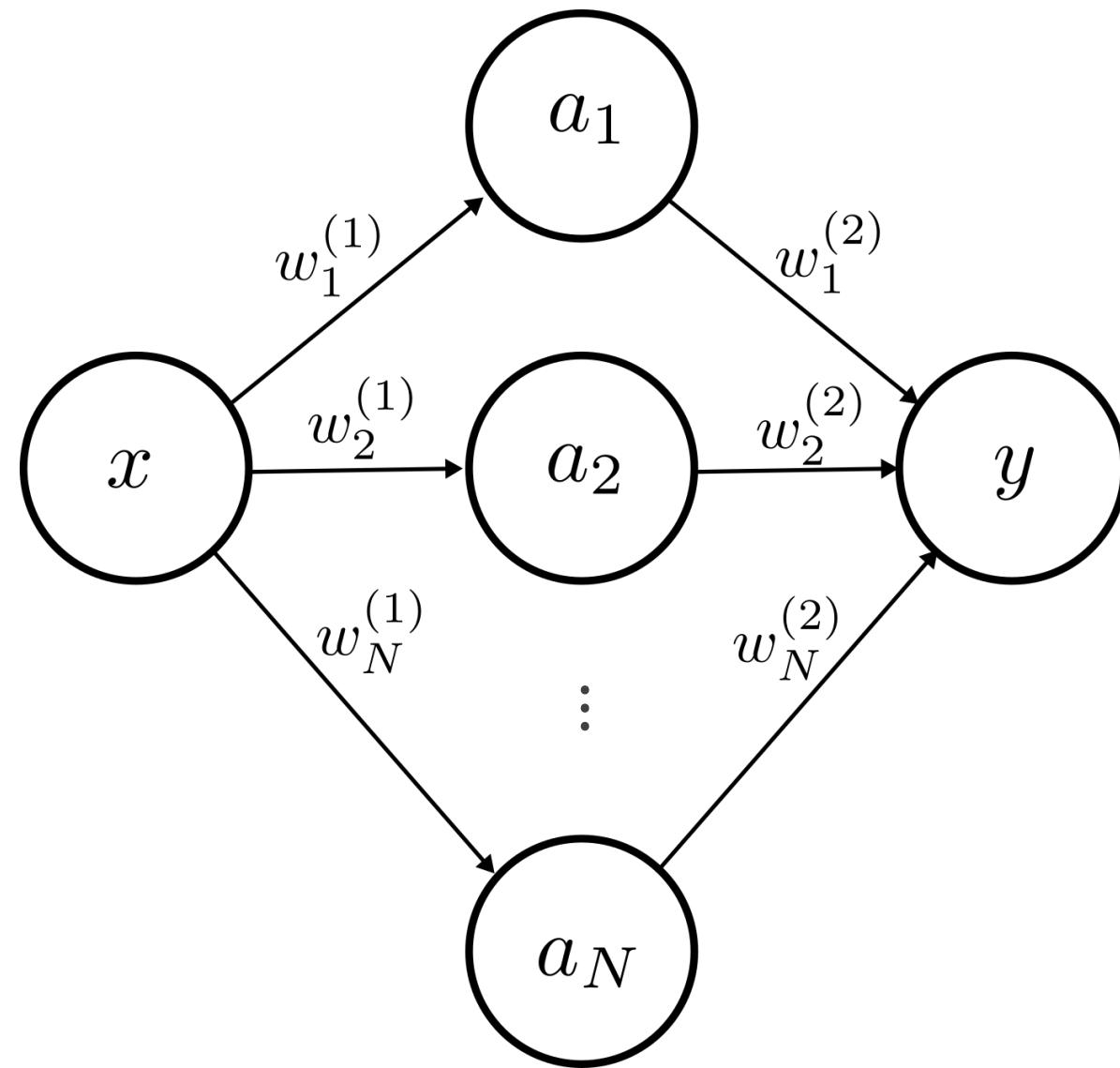
Example #3

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

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

Derivatives with respect to the weights:

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

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

Backpropagation intuitively

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

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

$$\underbrace{\frac{\partial f}{\partial x}}$$ Chain rule equality

This process of using the next step in the chain rule is backpropagation
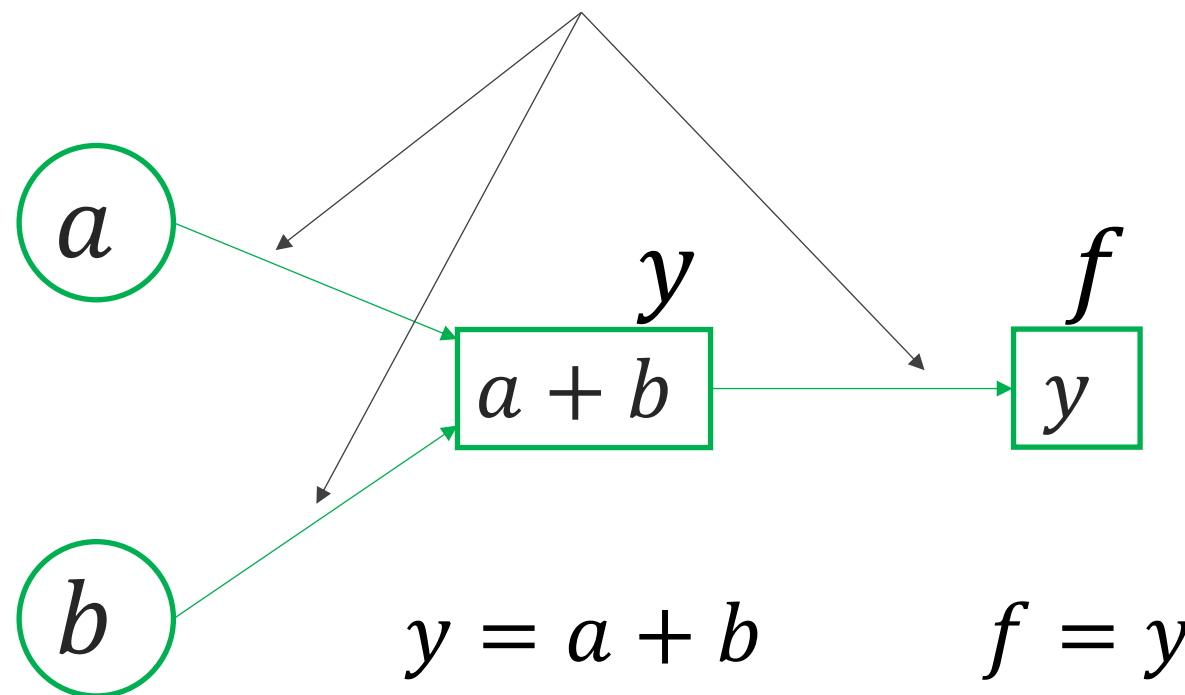
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial w}\frac{\partial w}{\partial x}\frac{\partial x}{\partial y}\frac{\partial y}{\partial z} \qquad = \frac{\partial f}{\partial x}\frac{\partial x}{\partial y}\frac{\partial y}{\partial z} \qquad = \frac{\partial f}{\partial y}\frac{\partial y}{\partial z} \qquad = \frac{\partial f}{\partial z}$$

$$\underbrace{\qquad} \qquad\qquad\qquad \underbrace{\qquad} \qquad\qquad\qquad \underbrace{\qquad}$$

$$\frac{\partial f}{\partial x} \qquad\qquad\qquad\qquad \frac{\partial f}{\partial y} \qquad\qquad\qquad\qquad \frac{\partial f}{\partial z}$$

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
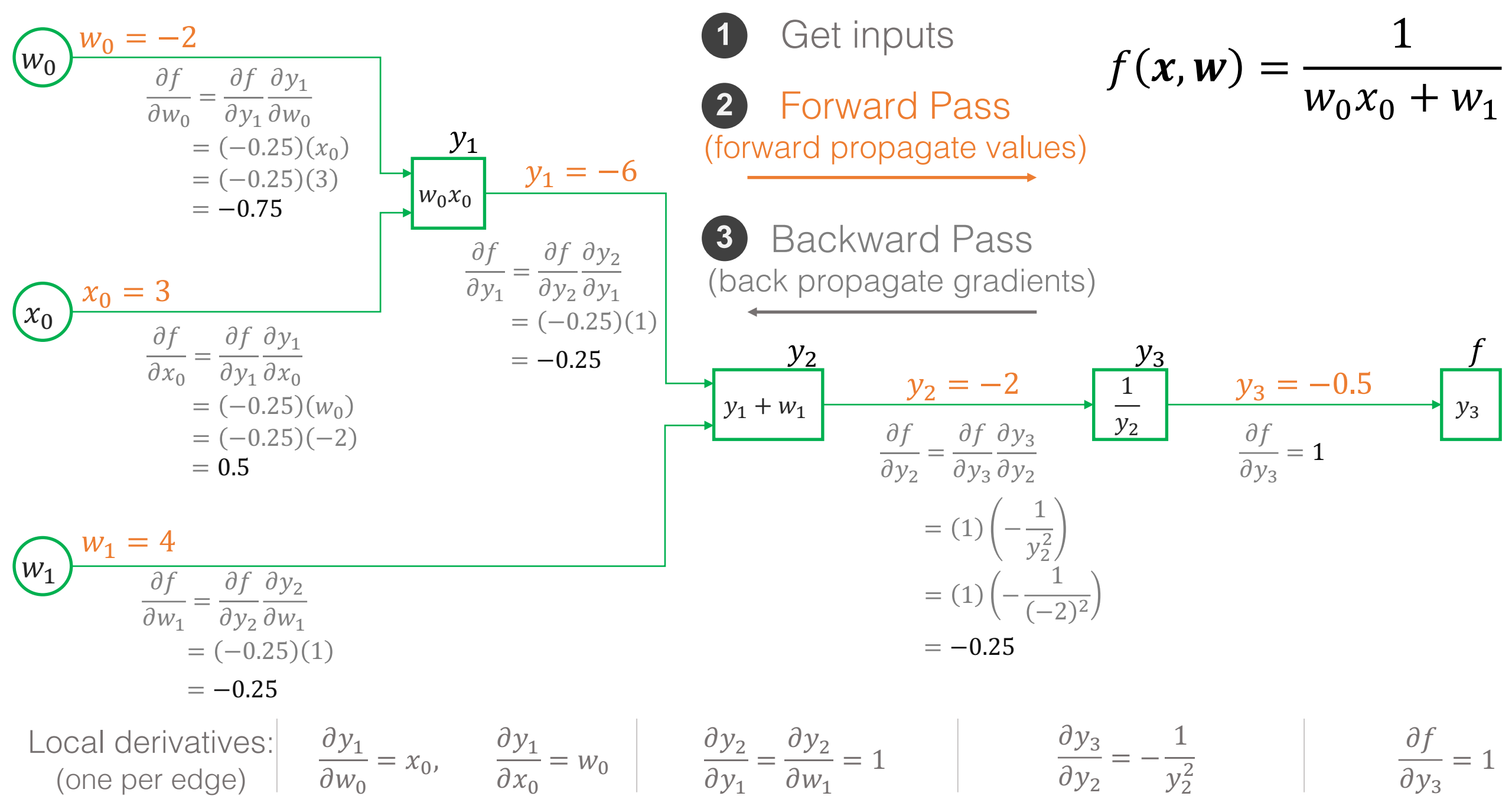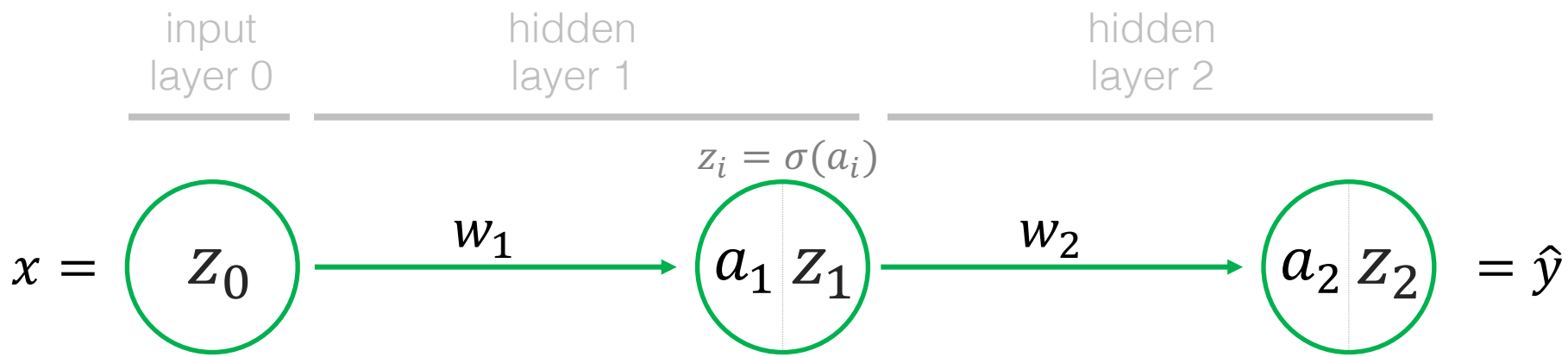

Name of the variable at that node

$y = a + b$

$f = y$

Operation that the node performs

Local derivatives (one for each edge input into a node):

$$\frac{\partial y}{\partial a} = 1, \qquad \frac{\partial y}{\partial b} = 1$$

$w_0 = -2$

$$\frac{\partial f}{\partial w_0} = \frac{\partial f}{\partial y_1}\frac{\partial y_1}{\partial w_0}$$
$$= (-0.25)(x_0)$$
$$= (-0.25)(3)$$
$$= -0.75$$

y_1

$w_0 x_0$

$y_1 = -6$

$x_0 = 3$

$$\frac{\partial f}{\partial x_0} = \frac{\partial f}{\partial y_1}\frac{\partial y_1}{\partial x_0}$$
$$= (-0.25)(w_0)$$
$$= (-0.25)(-2)$$
$$= 0.5$$

$$\frac{\partial f}{\partial y_1} = \frac{\partial f}{\partial y_2}\frac{\partial y_2}{\partial y_1}$$
$$= (-0.25)(1)$$
$$= -0.25$$

$w_1 = 4$

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial y_2}\frac{\partial y_2}{\partial w_1}$$
$$= (-0.25)(1)$$
$$= -0.25$$

1 Get inputs

2 Forward Pass
(forward propagate values)

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

3 Backward Pass
(back propagate gradients)

y_2

$y_1 + w_1$

$y_2 = -2$

$$\frac{\partial f}{\partial y_2} = \frac{\partial f}{\partial y_3}\frac{\partial y_3}{\partial y_2}$$

$$= (1)\left(-\frac{1}{y_2^2}\right)$$

$$= (1)\left(-\frac{1}{(-2)^2}\right)$$

$$= -0.25$$

y_3

$\frac{1}{y_2}$

$y_3 = -0.5$

$$\frac{\partial f}{\partial y_3} = 1$$

f

y_3

Local derivatives: | $\frac{\partial y_1}{\partial w_0} = x_0,$ $\frac{\partial y_1}{\partial x_0} = w_0$ | $\frac{\partial y_2}{\partial y_1} = \frac{\partial y_2}{\partial w_1} = 1$ | $\frac{\partial y_3}{\partial y_2} = -\frac{1}{y_2^2}$ | $\frac{\partial f}{\partial y_3} = 1$
(one per edge) |

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

input
layer 0

hidden
layer 1

hidden
layer 2

$z_i = \sigma(a_i)$

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

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

Forward propagation

$$z_2 = \hat{y} = \sigma(a_2)$$

$$a_2 = w_2 z_1$$

$$z_1 = \sigma(a_1)$$

$$a_1 = w_1 z_0 \rightarrow x$$

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

$$\hat{y} = z_2 = \sigma\big(w_2 \sigma(w_1 z_0)\big)$$

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
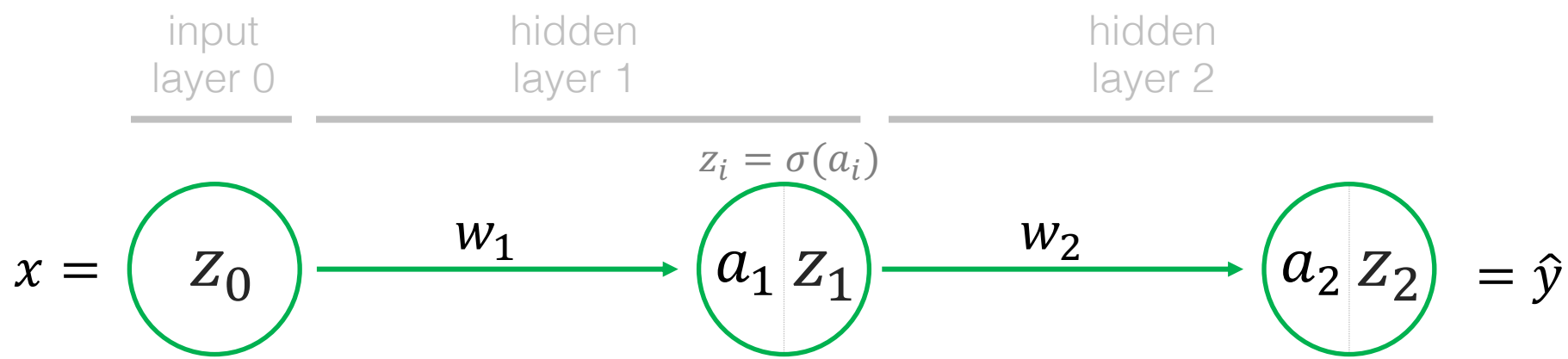
input
layer 0

hidden
layer 1

hidden
layer 2

$$z_i = \sigma(a_i)$$

$$x = \boxed{z_0} \xrightarrow{w_1} \boxed{a_1 \ z_1} \xrightarrow{w_2} \boxed{a_2 \ z_2} = \hat{y}$$

Forward propagation

Backpropagation

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

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

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

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

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

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

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

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

We know all these quantities from
forward propagation

input
layer 0

hidden
layer 1

hidden
layer 2

$z_i = \sigma(a_i)$

$x =$ z_0 $\xrightarrow{w_1}$ $a_1\ z_1$ $\xrightarrow{w_2}$ $a_2\ z_2$ $= \hat{y}$
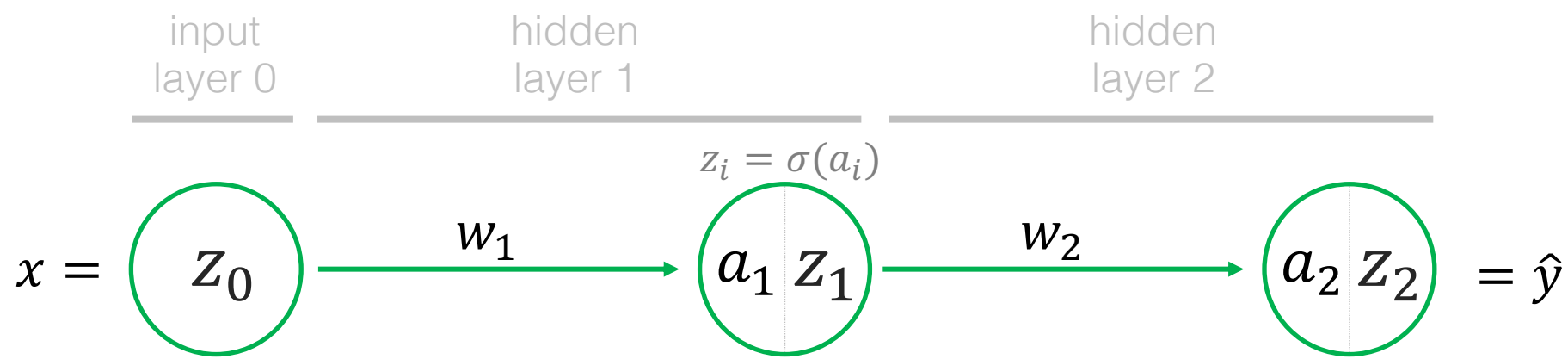
Forward propagation

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

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

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

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

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

Backpropagation

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

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

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial a_i}\frac{\partial a_i}{\partial w_i} \qquad = \frac{\partial E}{\partial a_i} z_{i-1}$$

δ_i (common shorthand)

And here's an actual mini neural network...
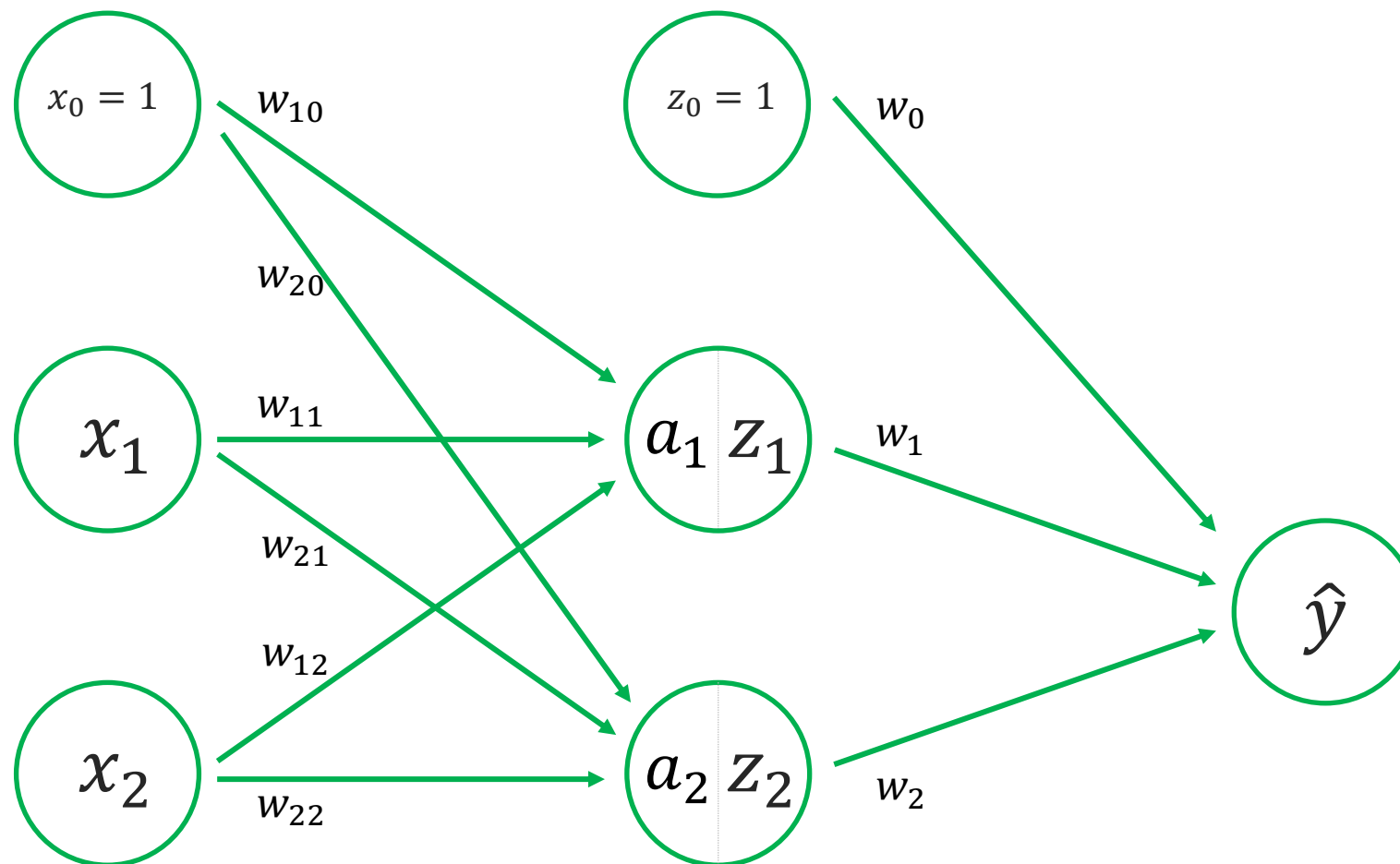
Activations in the other layer are the function $g(x)$, with derivative $g'(x)$

Linear activation for regression in the output layer

Local Relationship(s)

$$a_i = w_{i0}x_0 + w_{i1}x_1 + w_{i2}x_2 \qquad z_i = g(a_i) \qquad \hat{y} = w_0 z_0 + w_1 z_1 + w_2 z_2 \qquad C(y, \hat{y}) = E = \frac{1}{2}(\hat{y} - y)^2$$

Local Derivative(s)

$$\frac{\partial a_i}{\partial w_{ij}} = x_i \qquad \frac{\partial z_i}{\partial a_i} = g'(a_i) \qquad \frac{\partial \hat{y}}{\partial z_i} = w_i \qquad \frac{\partial \hat{y}}{\partial w_i} = z_i \qquad \frac{\partial E}{\partial \hat{y}} = \hat{y} - y$$

Global Derivative(s)

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i} \frac{\partial z_i}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}} \qquad \frac{\partial E}{\partial a_i} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i} \frac{\partial z_i}{\partial a_i} \qquad \frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i} \qquad \frac{\partial E}{\partial \hat{y}} = \hat{y} - y$$

Quick reference for 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:

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

$$\mathbf{z}^{(k)} = \sigma(\mathbf{a}^{(k)})$$
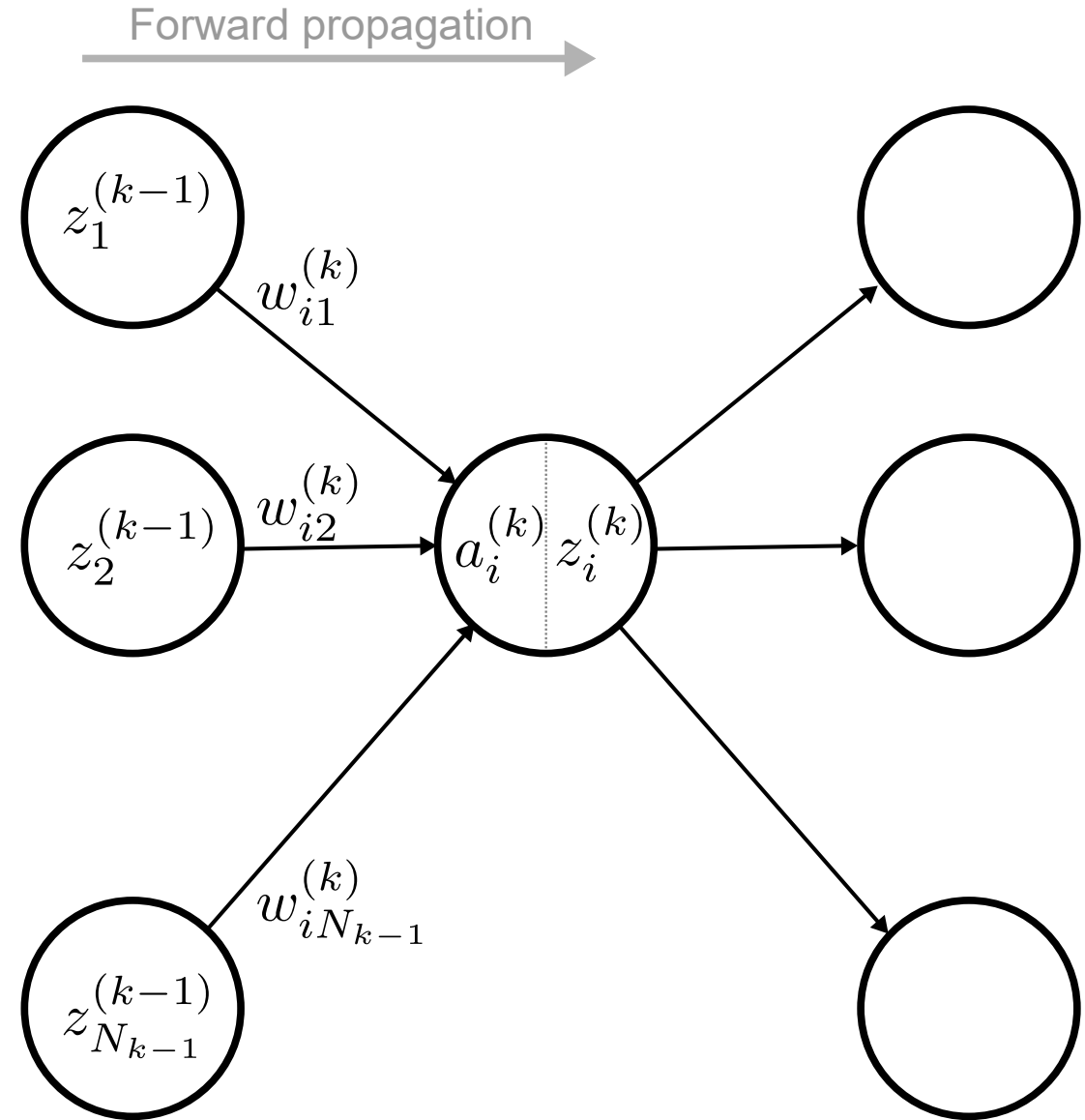
https://github.com/kylebradbury/neural-network-math/raw/master/neural_network_math.pdf

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 $k = K - 1, ...1$):

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

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

Or in matrix form:

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

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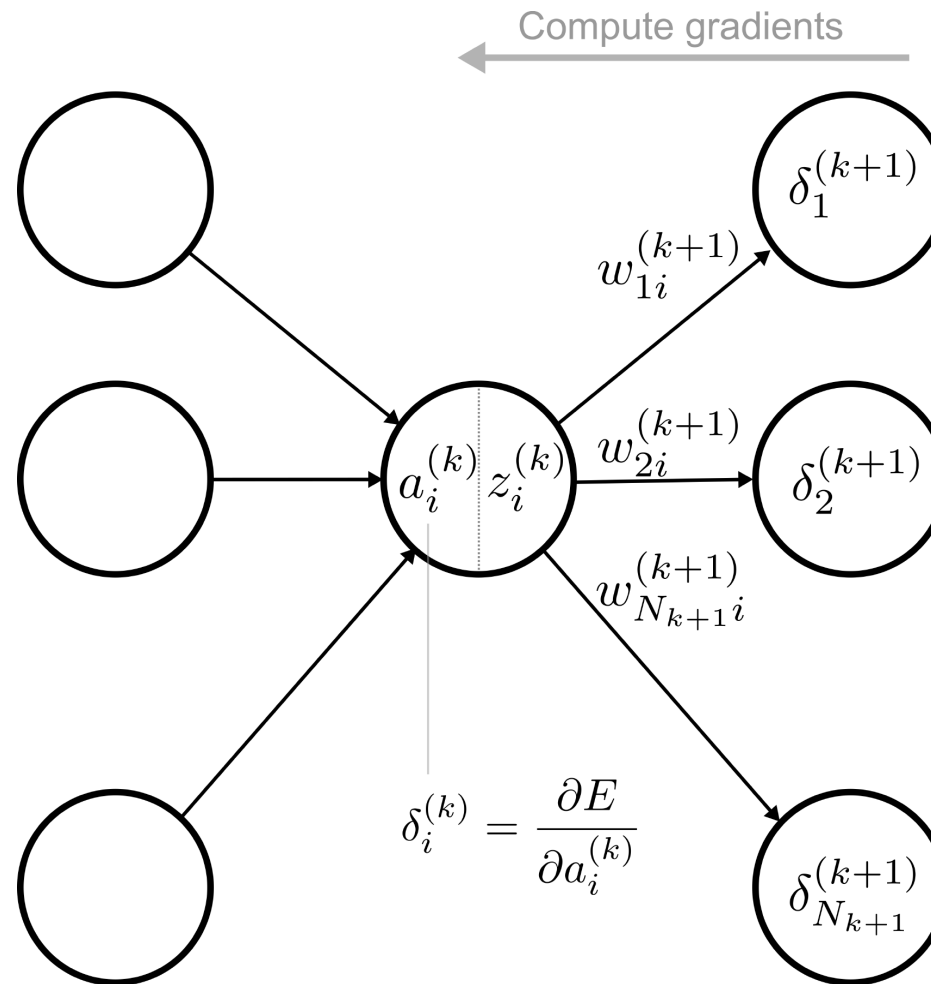
$$\delta_i^{(k)} = \frac{\partial E}{\partial a_i^{(k)}}$$

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 $\delta_i^{(k)} = \dfrac{\partial E}{\partial a_i^{(k)}}$ for all nodes in the network

3 Compute the weight derivatives: $\dfrac{\partial E}{\partial w_{ij}^{(k)}} = \delta_i^{(k)} z_j^{(k-1)}$ 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 training sample and average them

② Update all the parameters based on that average gradient

③ Repeat 1 and 2 until stopping criteria met

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

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

Our loss function (E_n) is calculated for EACH training sample $n = 1, 2, \ldots, N$

$$E_n = \frac{1}{2}(\hat{y}_n - y_n)^2$$

The gradient also needs to be calculated for each sample (i.e. backprop needs to be run for each sample)

Stochastic gradient descent (SGD)

① Randomly sort the list of training samples

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 stopping criteria met

$$\frac{\partial E_n}{\partial w_{ij}}$$

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

Minibatch gradient descent

A tweak to SGD where you use a small batch of training samples rather than the whole dataset.

The average gradient across this minibatch is used for taking a gradient descent step

**Other optimizers exist:**
e.g. momentum, RMSprop, adam

Batch gradient descent can work well if the cost function is convex (rare) and the learning rate is properly tuned

Batch gradient descent tends to converge more slowly

Stochastic gradient descent (SGD) is better at avoiding local minima, but injects noise into the training process

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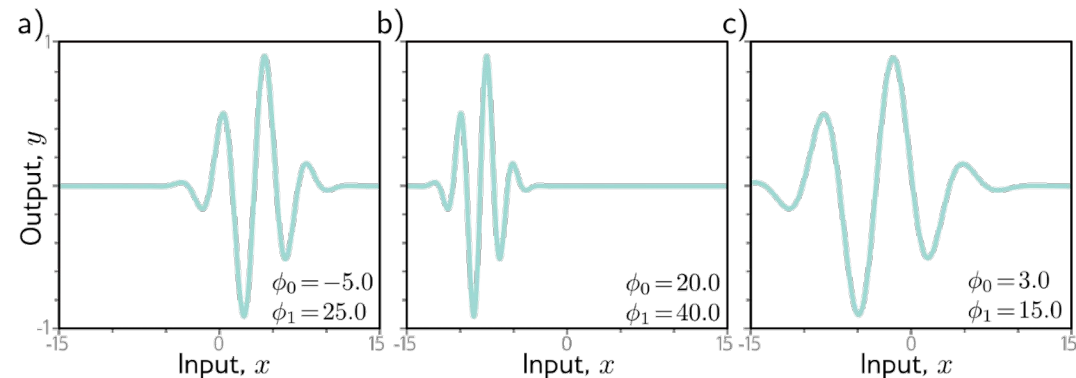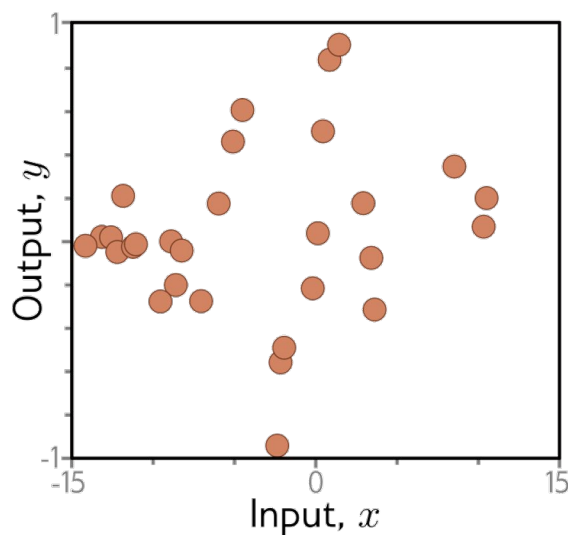

Convex cost function parameter space

$E = 102$

$E = 84$

$E = 41$

$E = 17$

○ Batch Gradient Descent

● Stochastic Gradient Descent

Batch gradient descent can work well if the cost function is convex (rare) and the learning rate is properly tuned

Batch gradient descent tends to converge more slowly

Stochastic gradient descent (SGD) is better at avoiding local minima, but injects noise into the training process

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

Cost function parameter space with multiple local minima

$E = 2$

$E = 0.1$

$E = 5$

○ Batch Gradient Descent
● Stochastic Gradient Descent
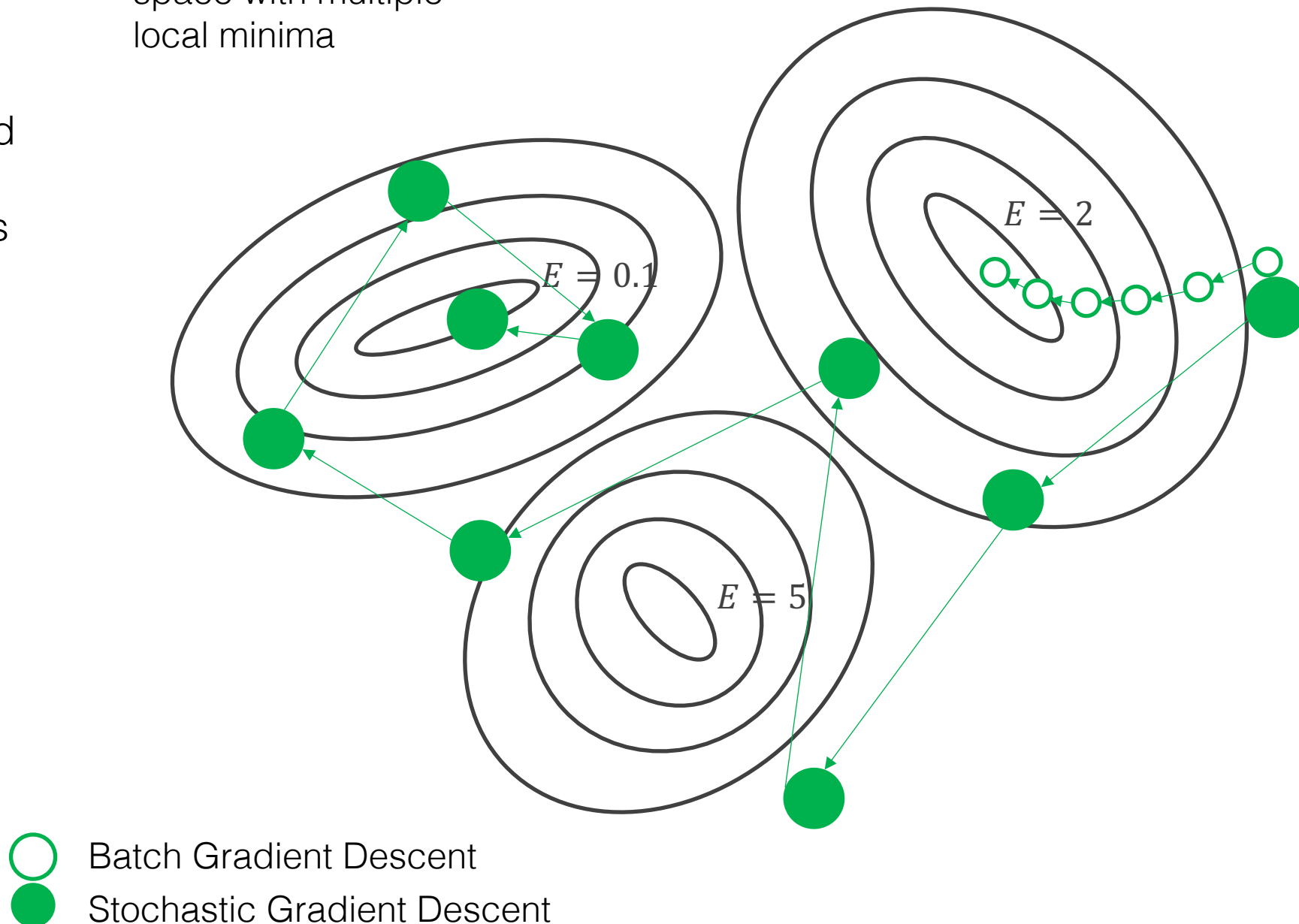
SGD Example

Gabor Model

$$\mathrm{f}[x, \boldsymbol{\phi}] = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{32.0}\right)$$

a) $\phi_0 = -5.0$ $\phi_1 = 25.0$
b) $\phi_0 = 20.0$ $\phi_1 = 40.0$
c) $\phi_0 = 3.0$ $\phi_1 = 15.0$

Training Data

Images from Prince, Understanding Deep Learning, 2023

Global cost function for different parameter values

a) Loss, $L[\boldsymbol{\phi}]$

b) Loss $= 3.67$

c) Loss $= 0.64$

d) Loss $= 5.51$

e) Loss $= 10.18$

f) Loss $= 9.96$
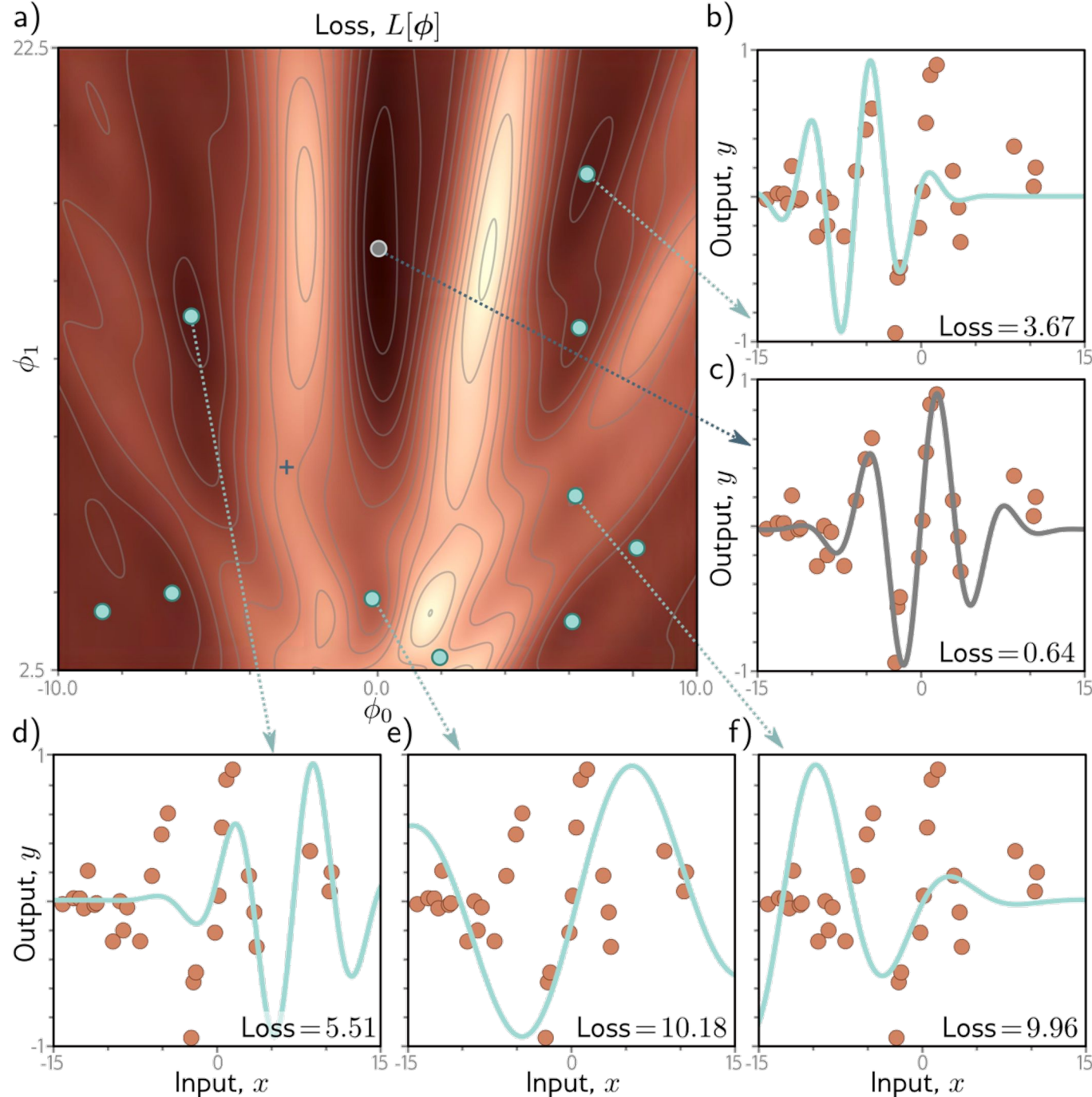
Global cost function
Cost function with the entire training dataset
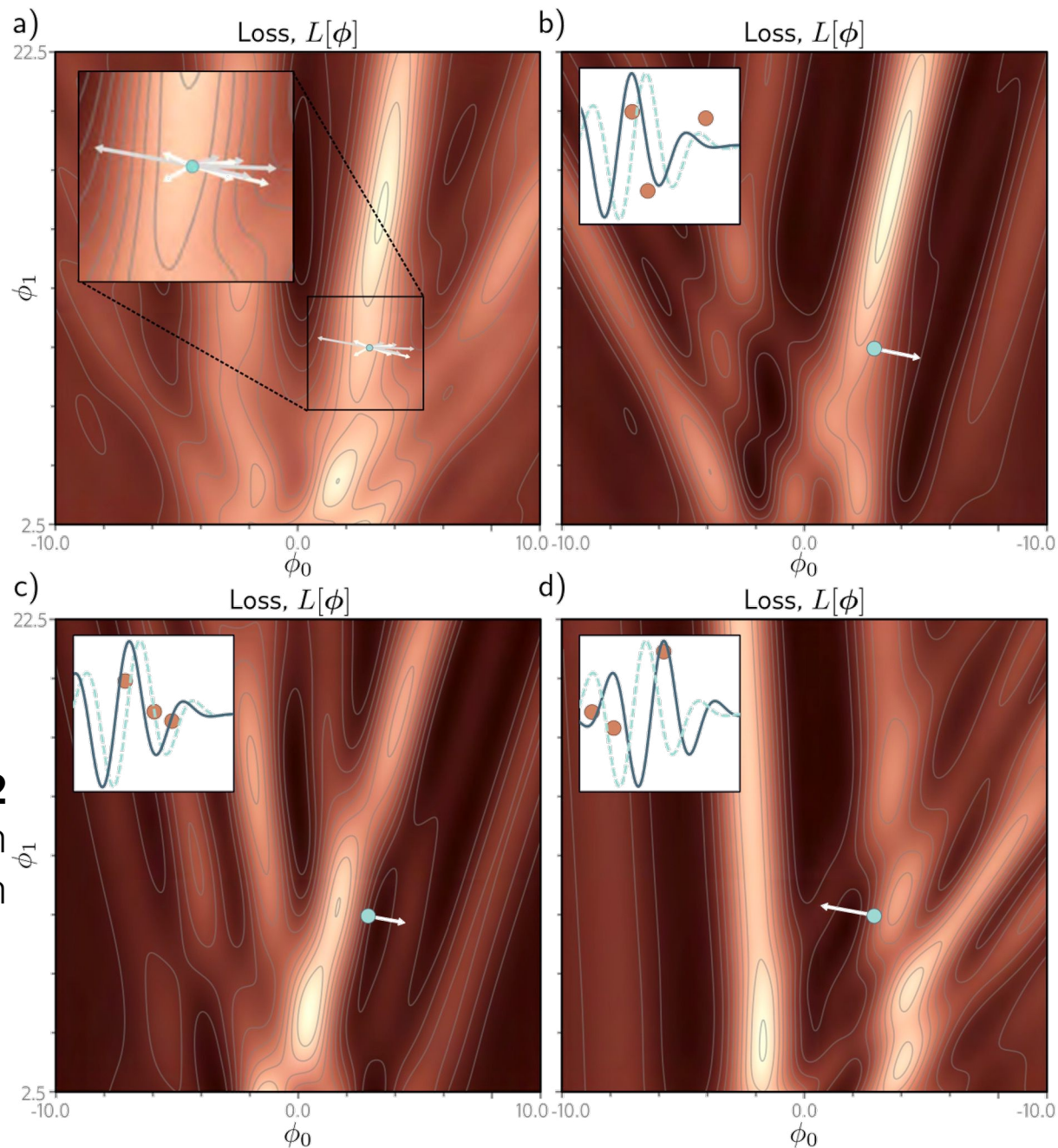
Minibatch cost function 1
Cost function with minibatch

Minibatch cost function 2
Cost function with minibatch

Minibatch cost function 3
Cost function with minibatch

SGD Example

Images from Prince, Understanding Deep Learning, 2023

SGD Example

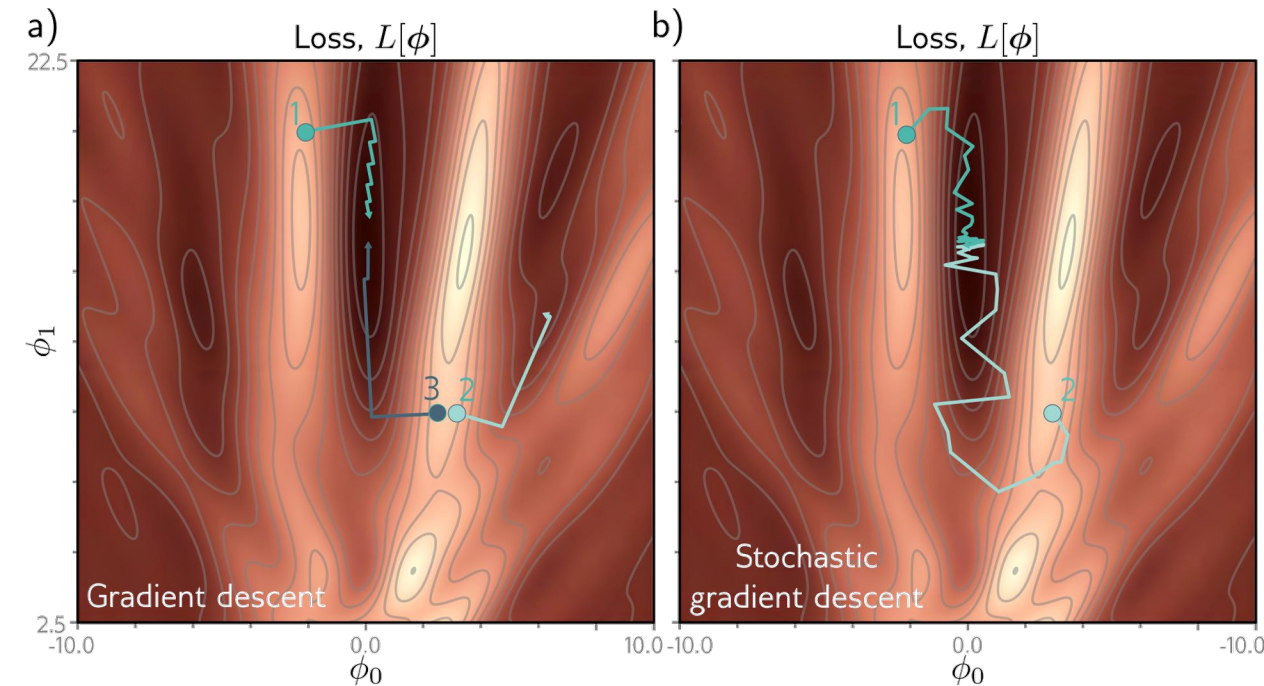Images from Prince, Understanding Deep Learning, 2023

Gradient descent
Batch size = N

Minibatch stochastic gradient descent
1 < batch size < N

N is the number of training samples
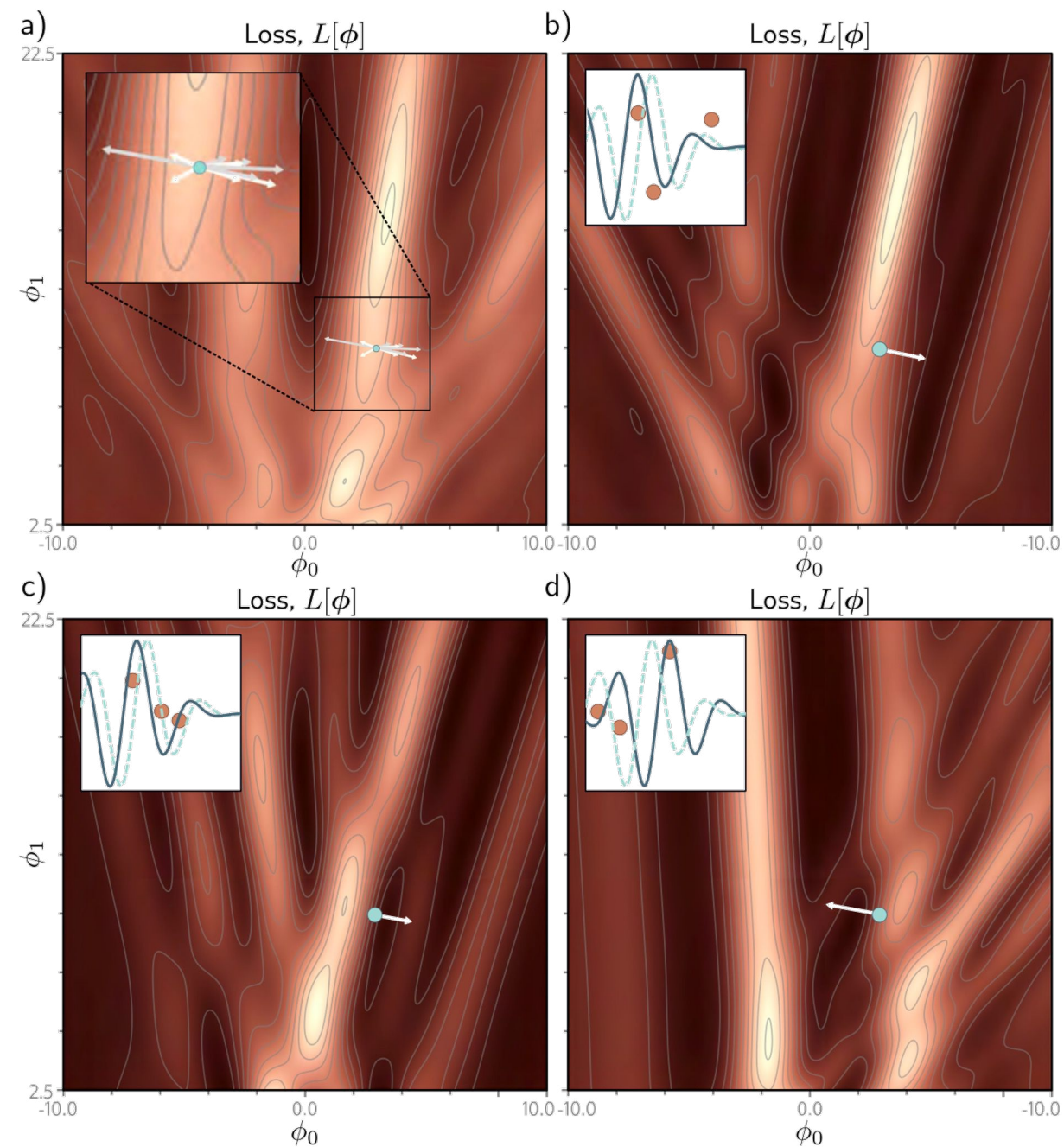
What is a neural network and **how does it work**?

How do we **choose model weights**?
(i.e. how do we fit our model to data)
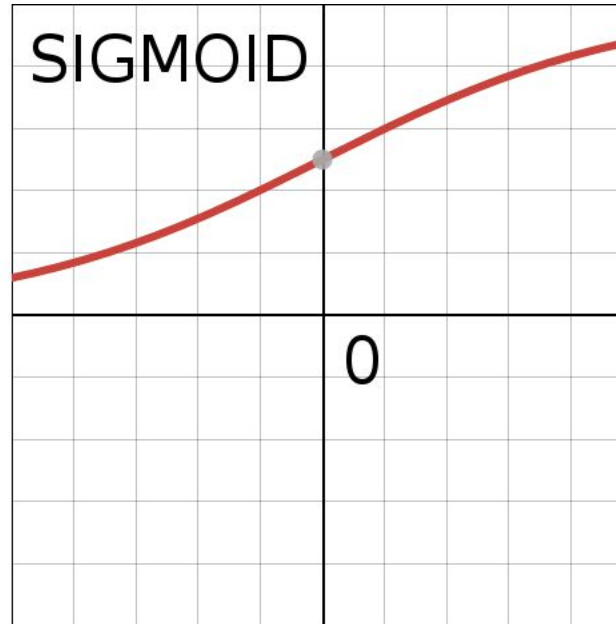
What are the challenges of using neural networks?

Hyperparameter / Architectural choices

- Learning Rate
- Minibatch size
- Architecture (number of nodes, number of layers, types of layers)
- Activation functions
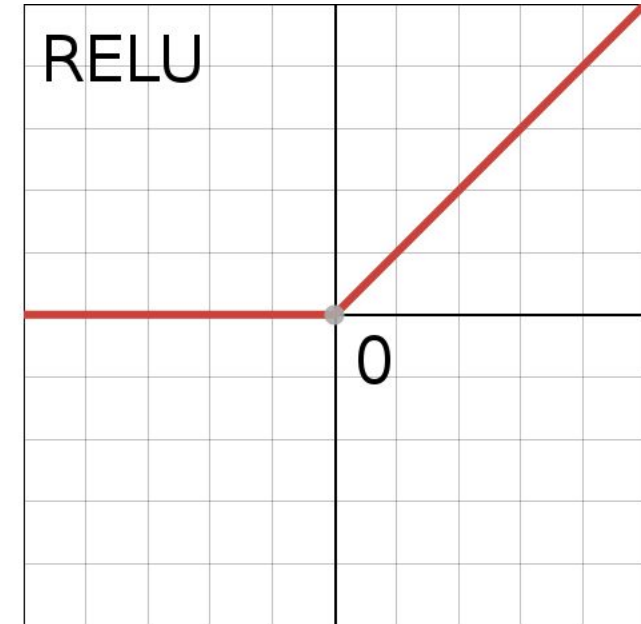- Weight initialization
- Regularization

Activation Functions
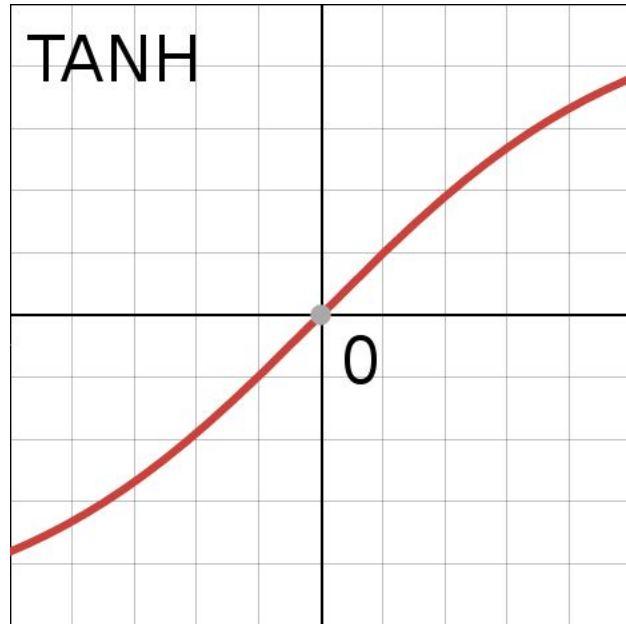
Hyperbolic Tangent

Sigmoid Tangent

Rectified linear unit (ReLU)

Increases training/prediction speed, sparse activation, reduces vanishing gradient

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

Random numbers   Need to be neither too small nor too big. A number of heuristics exist (Xavier, He, etc.)

Batch normalization

Ensures activations are unit Gaussian at each layer, improving optimization

(faster optimization, enables higher learning rates)
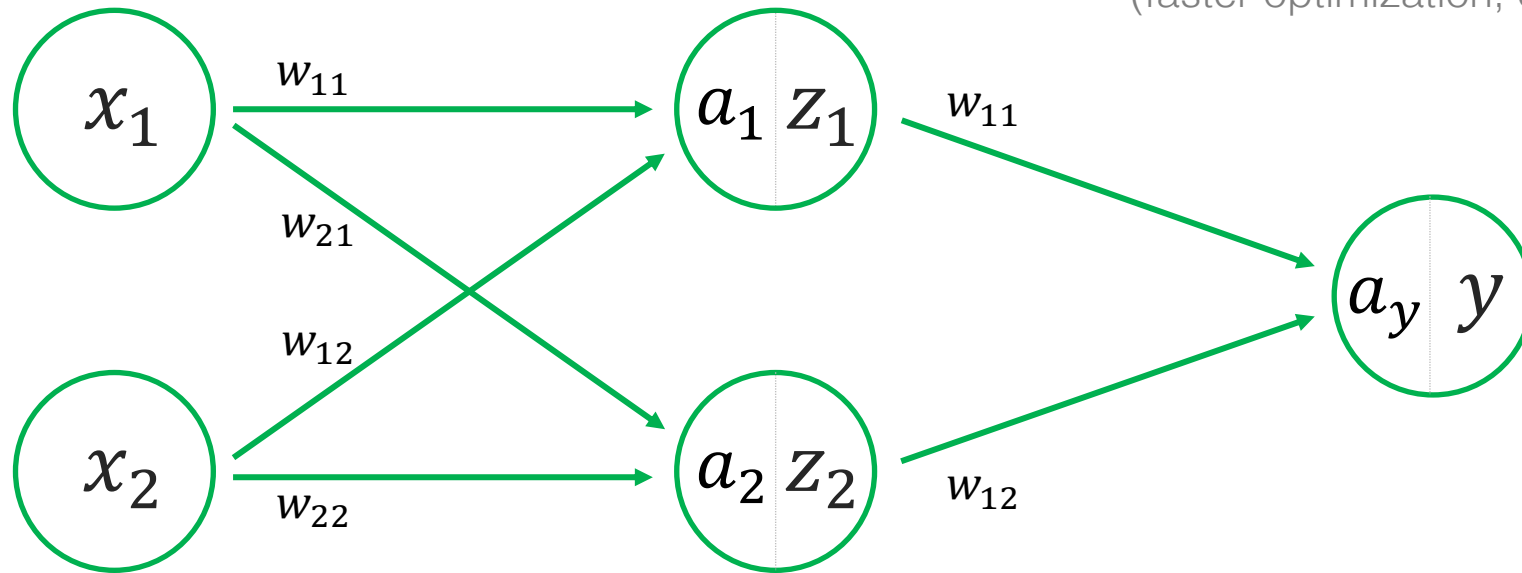
We calculate the activation at each layer for each of the training samples in each minibatch

1 Subtract the mean of that activation value averaged across the minibatch

2 Divide by the standard deviation of the activation value computed across the minibatch
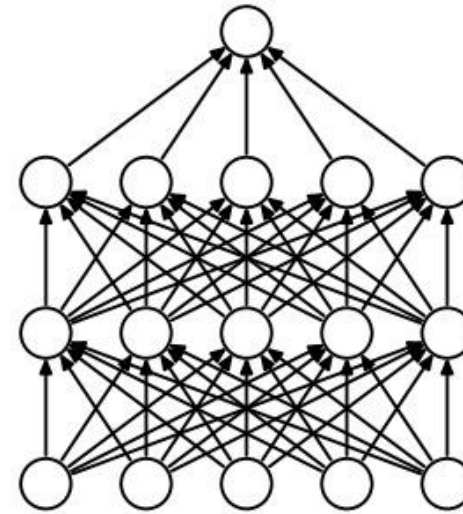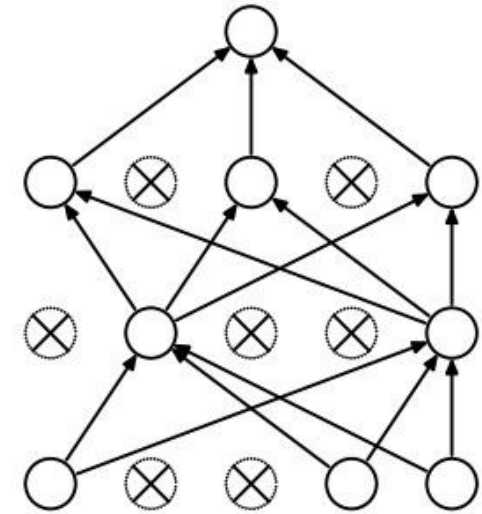
Regularization

L2 Regularization

L1 Regularization

(a) Standard Neural Net

(b) After applying dropout.

Dropout

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

Watch your learning curves

Questions to ask:
- Learning anything?
- Still learning?
- Overfitting?
- Learning rate too low/high?

Image by Jason Brownlee
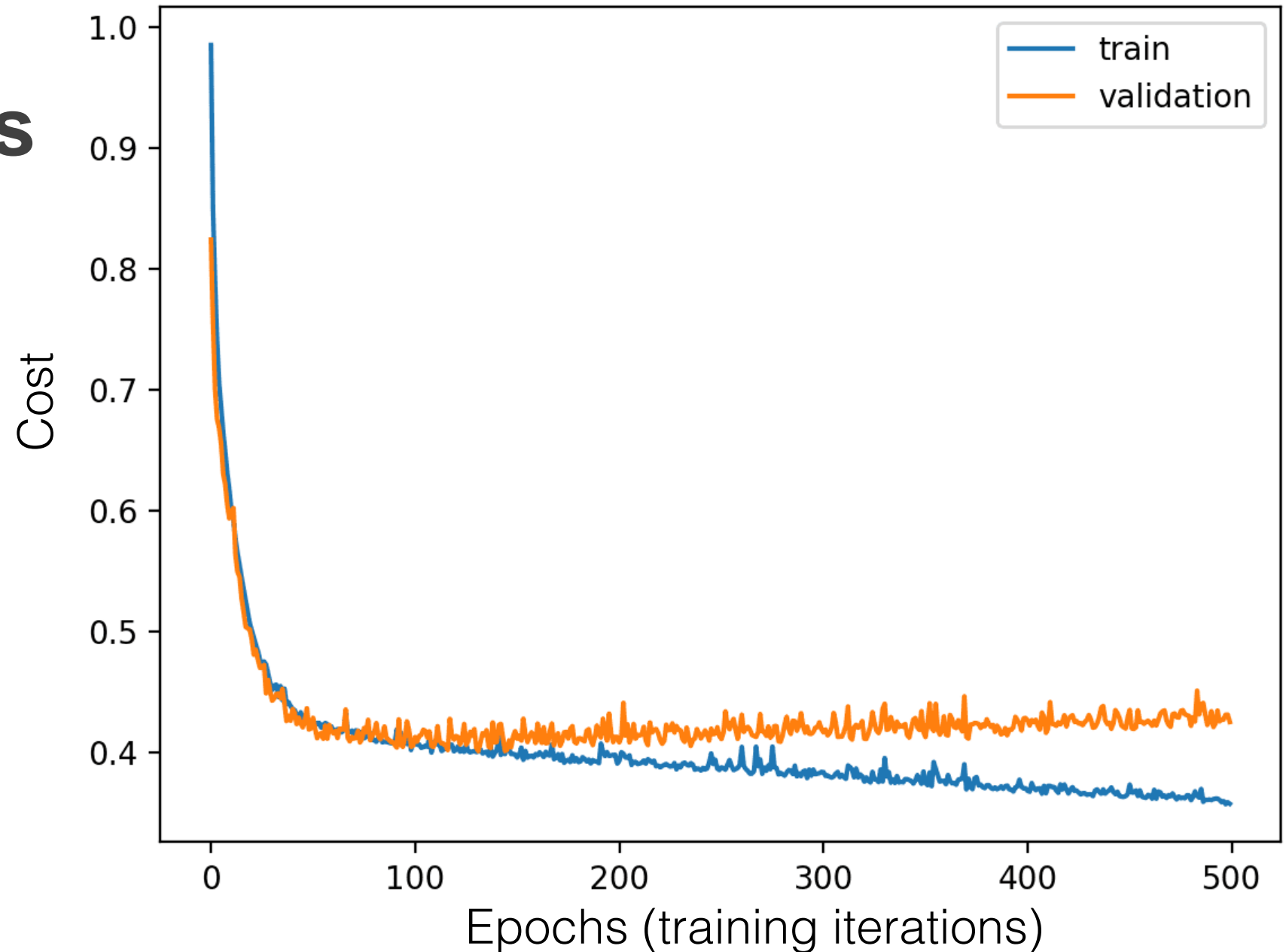
Successfully training neural networks

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

Advice on using practical neural networks

Deep Learning Tuning Playbook

Deep Learning Tuning Playbook

This is not an officially supported Google product.

Varun Godbole[+], George E. Dahl[+], Justin Gilmer[+], Christopher J. Shallue[‡], Zachary Nado[+]

[+] Google Research, Brain Team

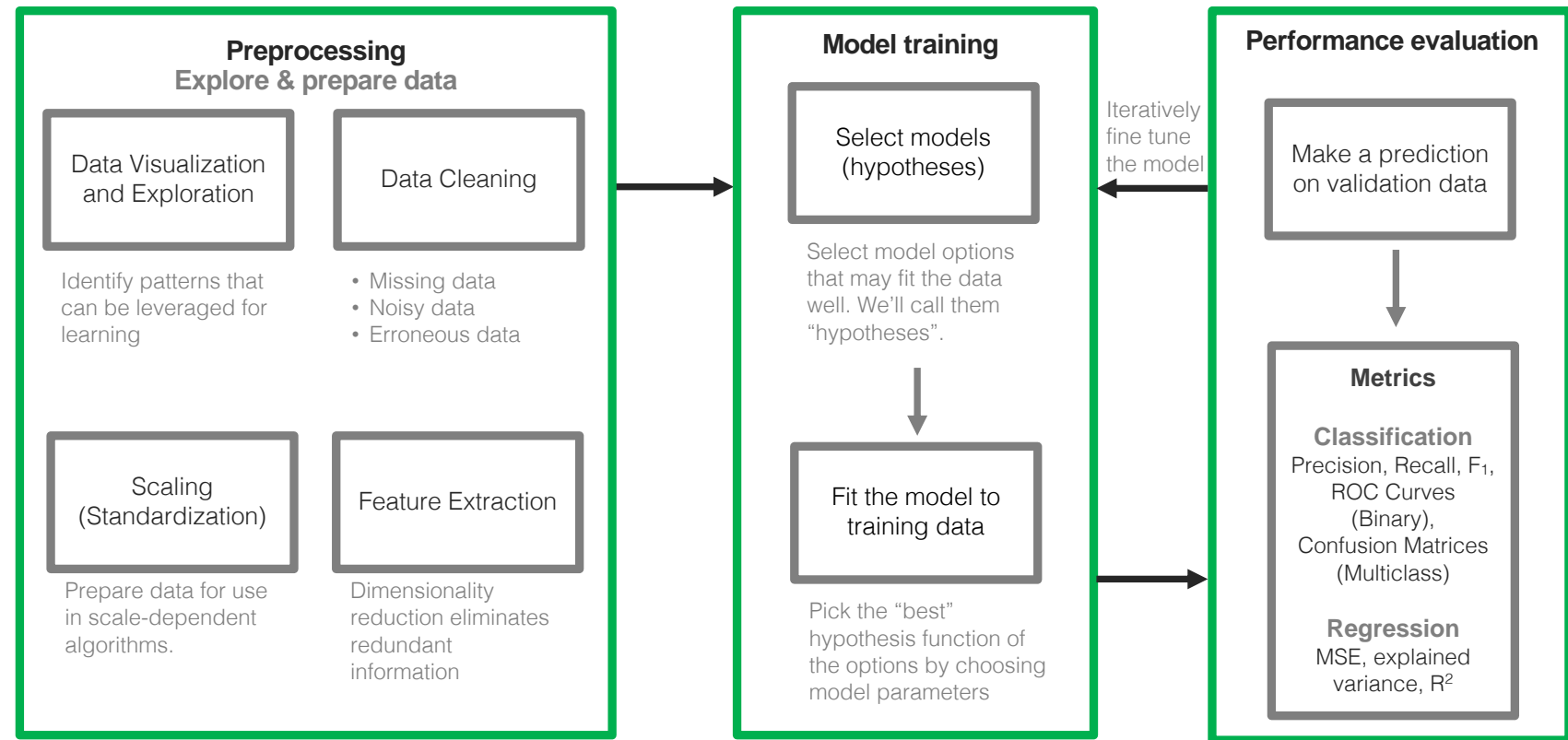[‡] Harvard University

Table of Contents

Experimental Design

Experimental Design

- Enables us to answer questions, typically related to comparing models (e.g. does Model A or Model B perform better)

- Requires that you ONLY vary one characteristic at a time (e.g. model architecture, hyperparameters, training data)

- Requires that you control for uncertainty (in the model AND in the data)

Experimental Design

What do we need to check to make sure that there is only one change between experimental conditions?

Preprocessing
Explore & prepare data

Data Visualization and Exploration	Data Cleaning

Identify patterns that can be leveraged for learning

- Missing data
- Noisy data
- Erroneous data

Scaling (Standardization)	Feature Extraction

Prepare data for use in scale-dependent algorithms.

Dimensionality reduction eliminates redundant information

Model training

Select models (hypotheses)

Select model options that may fit the data well. We'll call them "hypotheses".

Fit the model to training data

Pick the "best" hypothesis function of the options by choosing model parameters

Iteratively fine tune the model

Performance evaluation

Make a prediction on validation data

Metrics

Classification
Precision, Recall, F_1, ROC Curves (Binary), Confusion Matrices (Multiclass)

Regression
MSE, explained variance, R^2

- Train/validation/test split
  - How it was split
  - The split itself
- Data imputation
- Feature selection
- Scaling
- Dimensionality reduction
- Feature transformations

- Model architecture
- Hyperparameters (e.g. learning rate, batch size, stopping criterion, choice of regularization)

Validation / test data should remain unchanged

Experimental Design Example
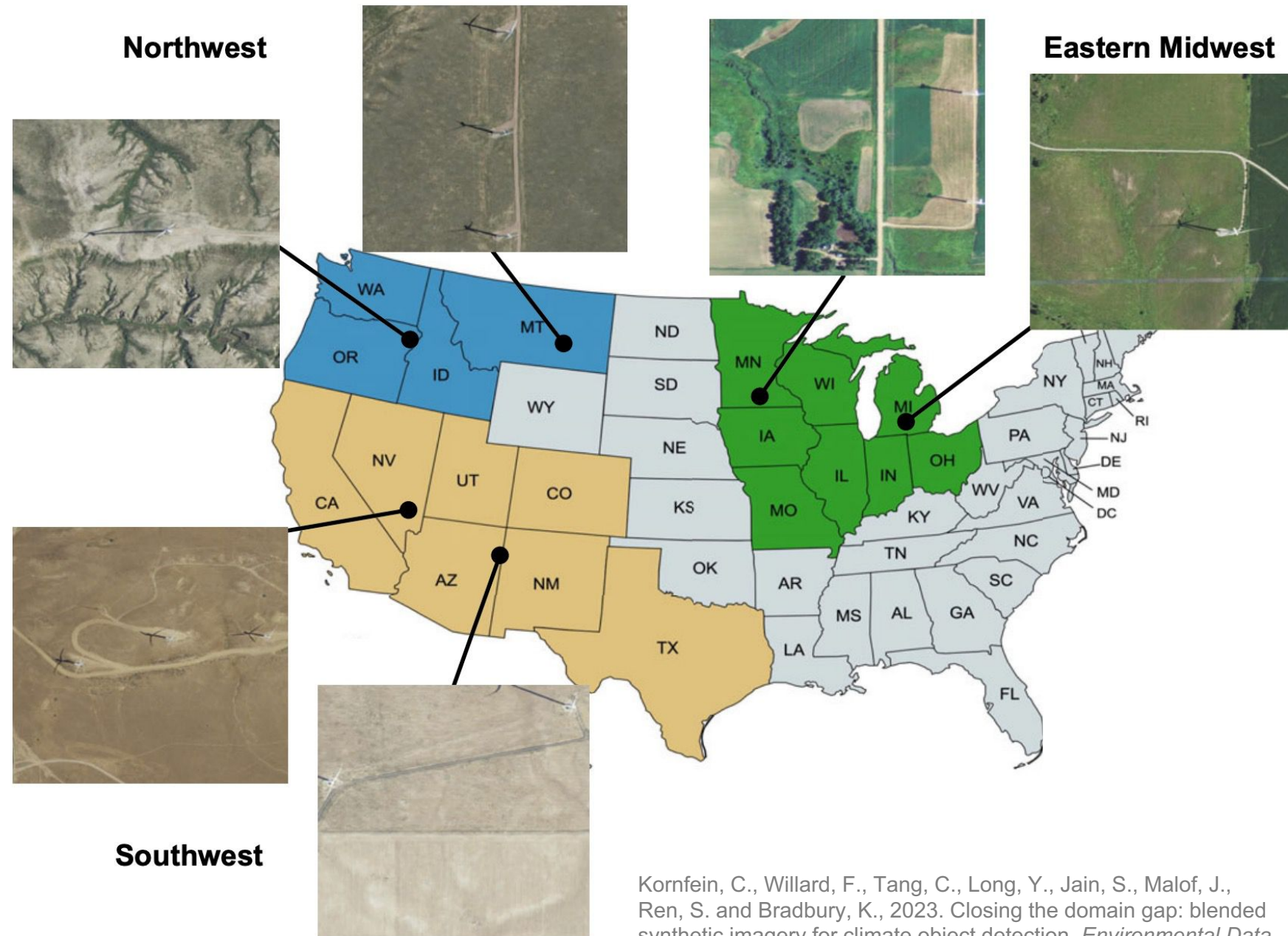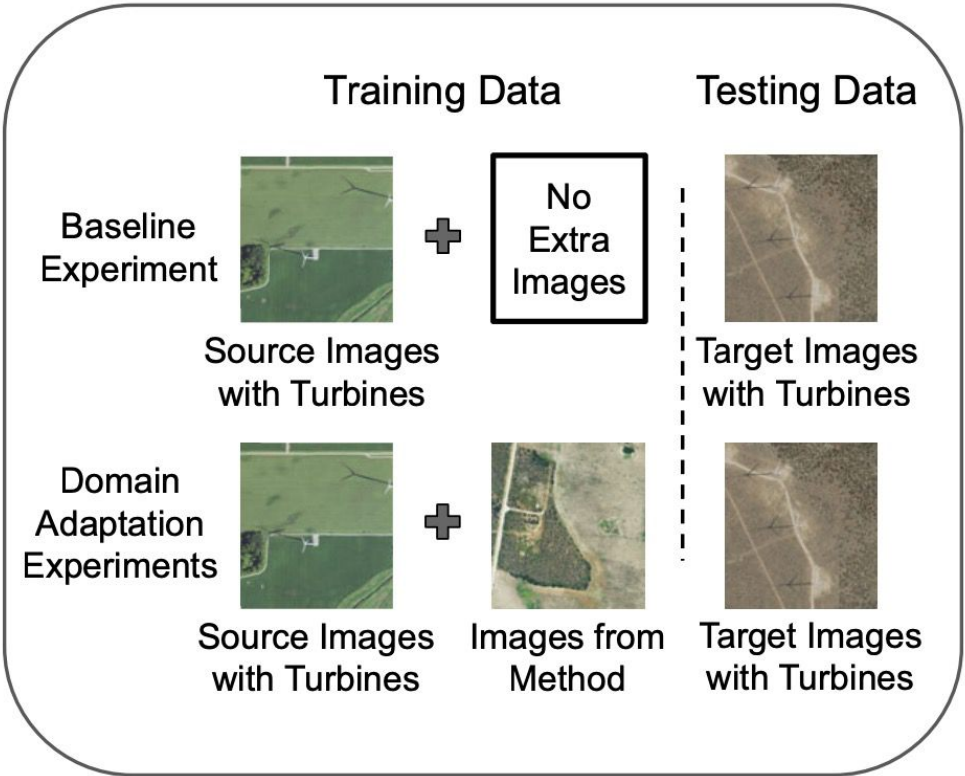
How well do models transfer to new domains?

Kornfein, C., Willard, F., Tang, C., Long, Y., Jain, S., Malof, J., Ren, S. and Bradbury, K., 2023. Closing the domain gap: blended synthetic imagery for climate object detection. *Environmental Data Science*, 2, p.e39.

Experimental Design Example

Experimental Design
The only change between experimental conditions was the content of the training data
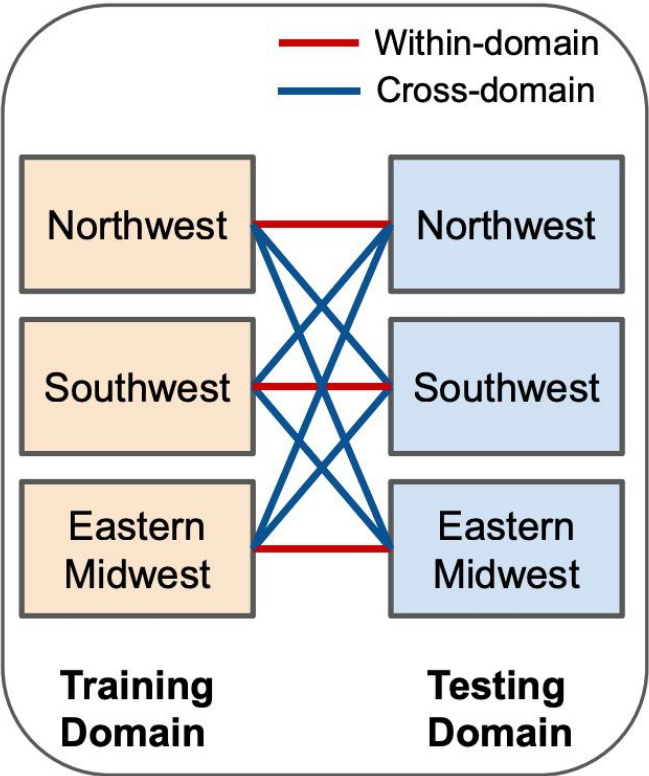

Training Procedure

Kornfein, C., Willard, F., Tang, C., Long, Y., Jain, S., Malof, J., Ren, S. and Bradbury, K., 2023. Closing the domain gap: blended synthetic imagery for climate object detection. *Environmental Data Science*, 2, p.e39.
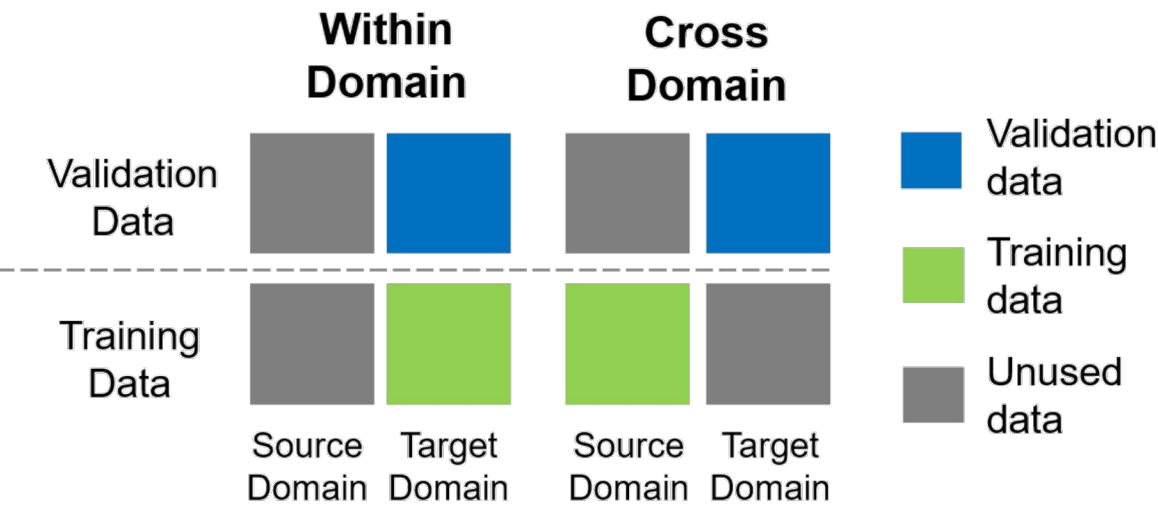
Kornfein, C., Willard, F., Tang, C., Long, Y., Jain, S., Malof, J., Ren, S. and Bradbury, K., 2022. "Closing the Domain Gap–Blended Synthetic Imagery for Climate Object Detection." In 2022 Neural Information Processing Systems (NeurIPS) Climate Change AI Workshop.

Experimental Design Example

How well do models transfer to new domains?

Results shown in units of mean average precision (mAP)

Source domain	Target domain	Baseline $\pm 2\sigma$	Adding synthetic $\pm 2\sigma$	Average improvement%
EM		0.822 ± 0.067	$\mathbf{0.919} \pm 0.016$	11.8%
NE	EM	0.567 ± 0.019	$\mathbf{0.698} \pm 0.038$	23.1%
NW		0.358 ± 0.061	$\mathbf{0.424} \pm 0.114$	18.4%
SW		0.449 ± 0.160	$\mathbf{0.626} \pm 0.180$	39.4%
EM		0.387 ± 0.031	$\mathbf{0.487} \pm 0.114$	25.8%
NE	NE	0.812 ± 0.028	$\mathbf{0.842} \pm 0.013$	3.7%
NW		0.666 ± 0.061	$\mathbf{0.709} \pm 0.049$	6.5%
SW		0.412 ± 0.045	$\mathbf{0.521} \pm 0.089$	26.5%
EM		0.485 ± 0.064	$\mathbf{0.521} \pm 0.054$	7.4%
NE	NW	0.746 ± 0.018	$\mathbf{0.770} \pm 0.032$	3.2%
NW		0.895 ± 0.071	$\mathbf{0.915} \pm 0.023$	2.2%
SW		0.659 ± 0.111	$\mathbf{0.693} \pm 0.066$	5.2%
EM		0.093 ± 0.016	$\mathbf{0.113} \pm 0.008$	20.9%
NE	SW	0.121 ± 0.029	$\mathbf{0.134} \pm 0.030$	10.7%
NW		0.149 ± 0.029	$\mathbf{0.197} \pm 0.024$	32.2%
SW		0.566 ± 0.035	$\mathbf{0.568} \pm 0.104$	0.4%
Within-domain average		0.774 ± 0.050	$\mathbf{0.811} \pm 0.039$	4.8%
Cross-domain average		0.425 ± 0.054	$\mathbf{0.491} \pm 0.067$	15.7%

Bolded items represent best model (comparing a baseline model and experimental condition model)

Standard deviations contextualize model uncertainty
- This represents retraining the model multiple times to measure performance variability
- For a result to be significant, the performance change needs to be large enough to be unlikely to be due to model variability

Kornfein, C., Willard, F., Tang, C., Long, Y., Jain, S., Malof, J., Ren, S. and Bradbury, K., 2023. Closing the domain gap: blended synthetic imagery for climate object detection. *Environmental Data Science*, 2, p.e39.
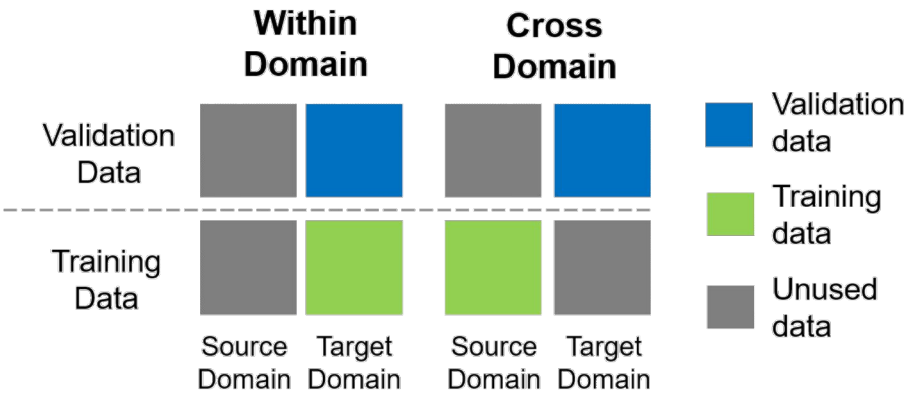
Kornfein, C., Willard, F., Tang, C., Long, Y., Jain, S., Malof, J., Ren, S. and Bradbury, K., 2022. "Closing the Domain Gap–Blended Synthetic Imagery for Climate Object Detection." In 2022 Neural Information Processing Systems (NeurIPS) Climate Change AI Workshop.