

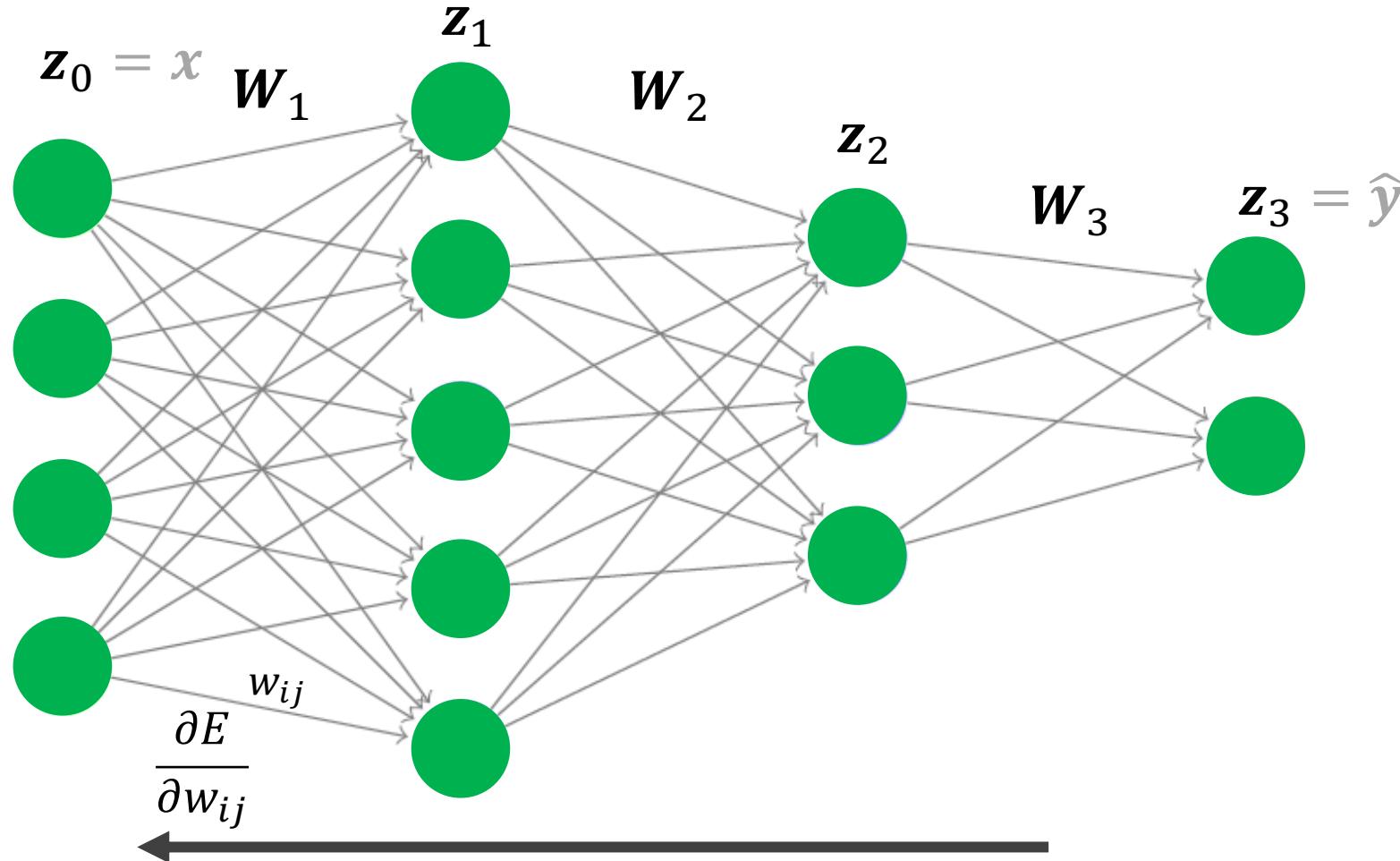
Neural Networks II

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



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

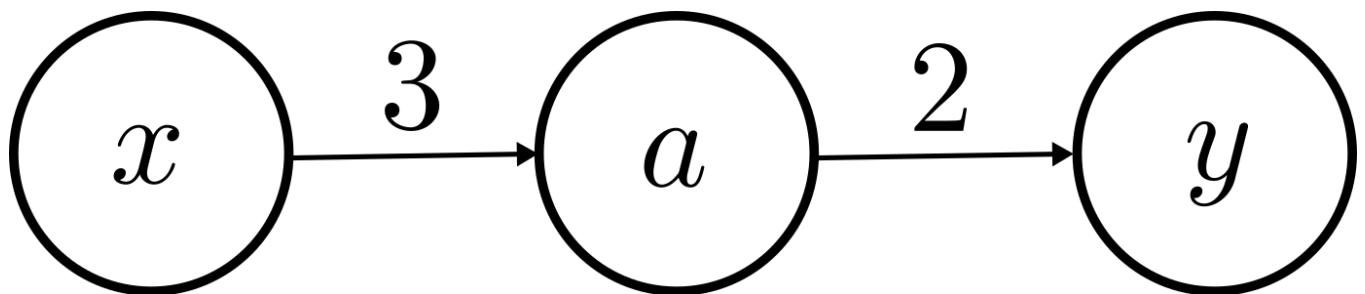
$$E = \frac{1}{2} \sum_k (\hat{y}_k - y_k)^2$$

(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



$$y = 2a$$

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

$$a = 3x$$

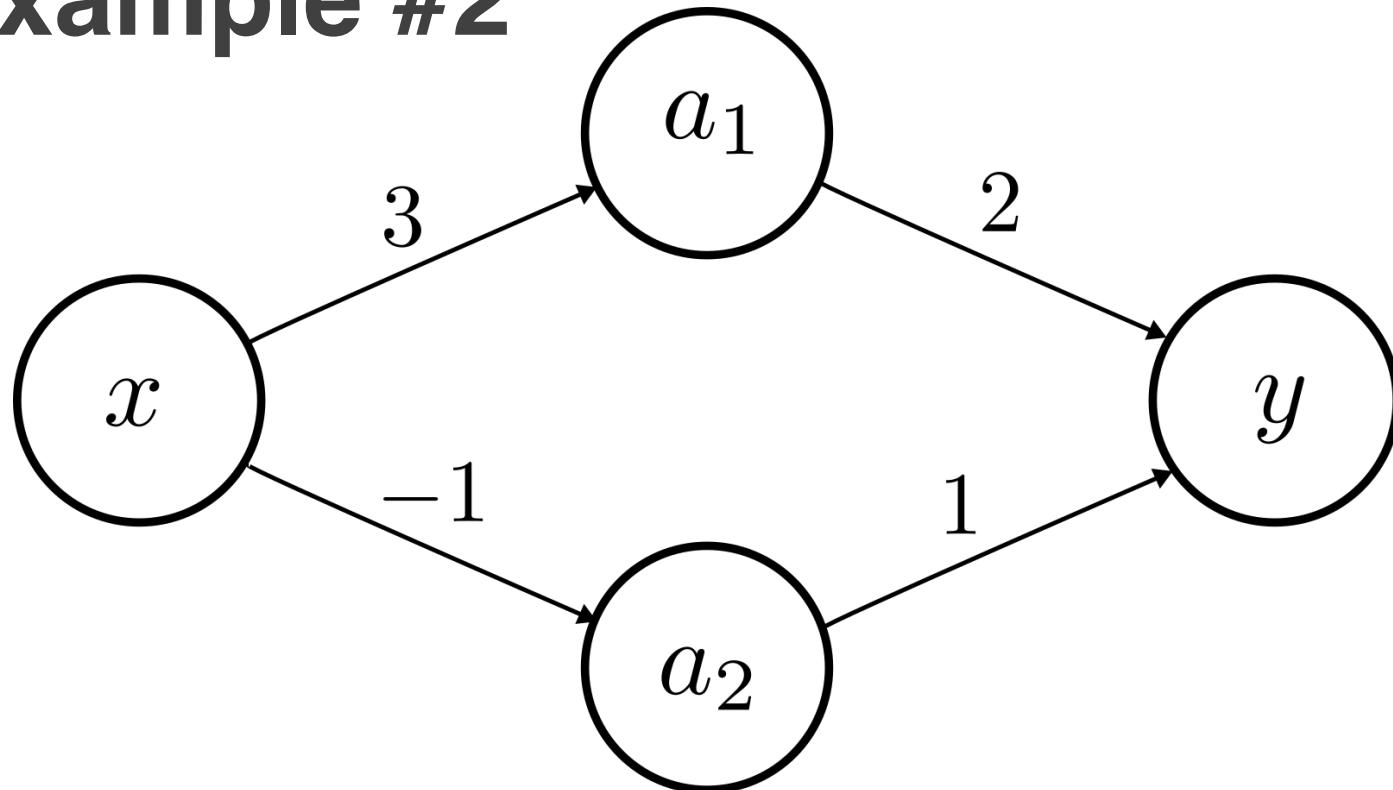
$$\frac{\partial a}{\partial x} = 3$$

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

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

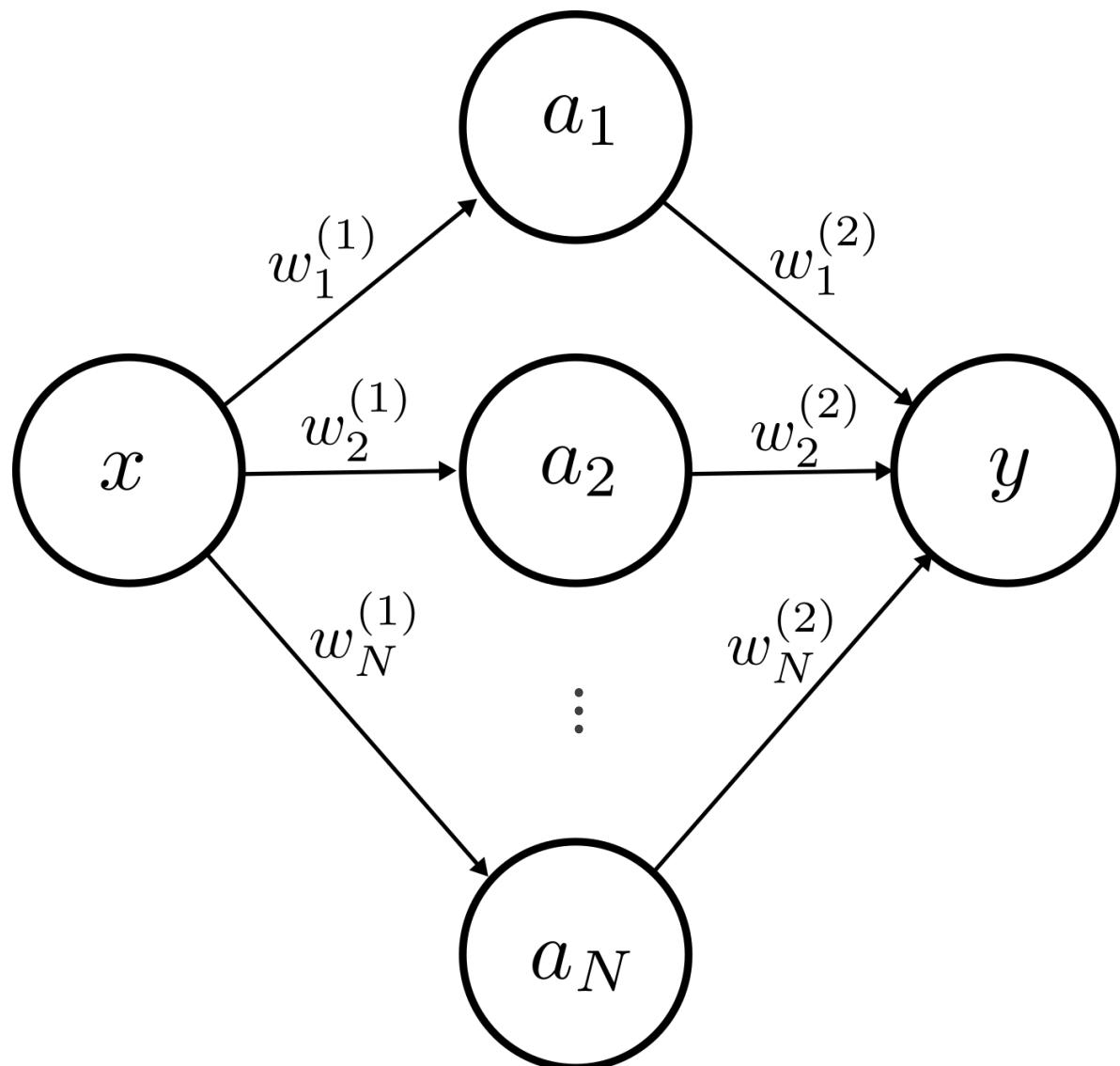
$$y = \sum_{j=1}^N w_j^{(2)} a_j$$

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

$$a_i = w_i^{(1)} x$$

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

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



Example #3

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

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

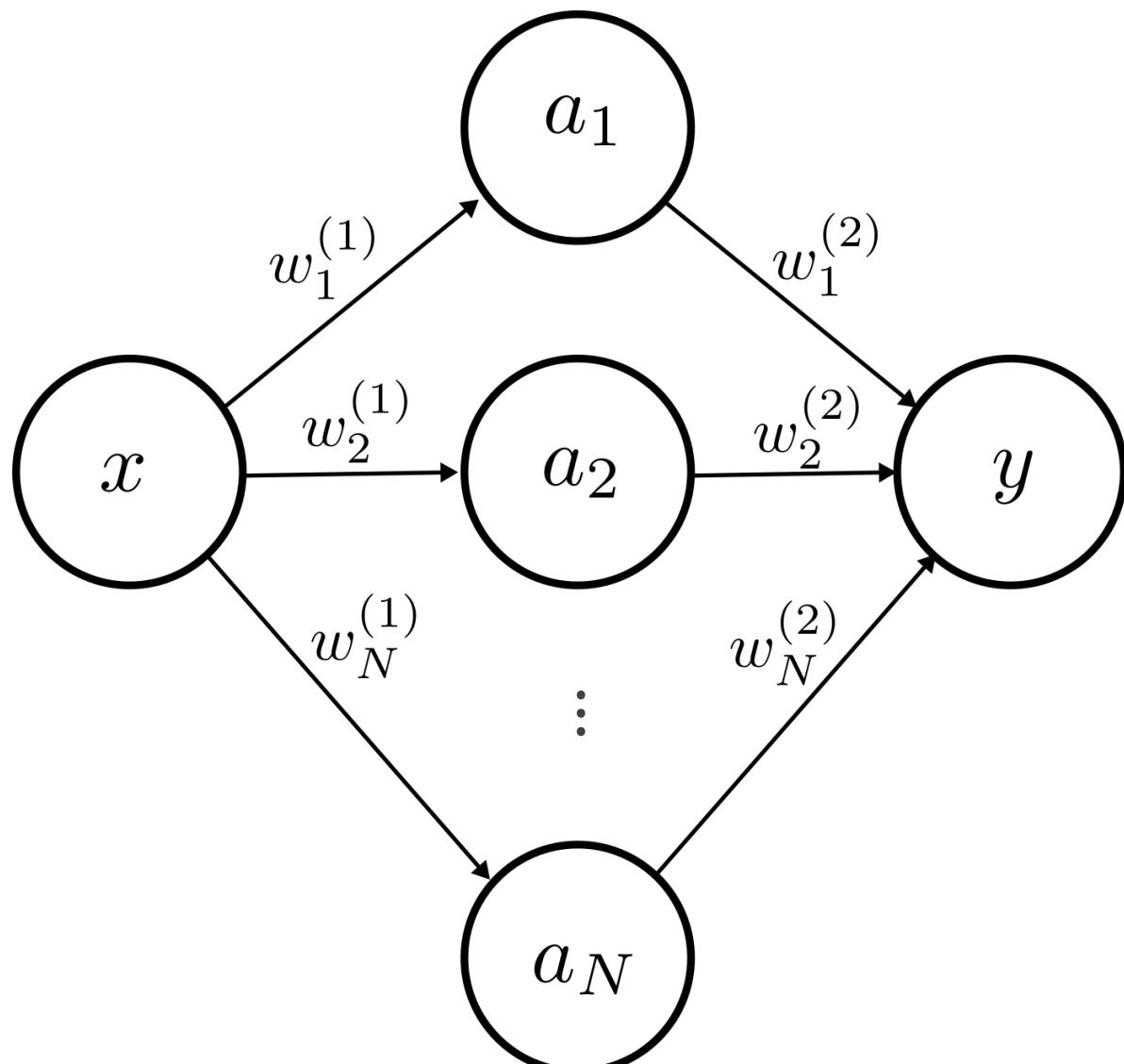
$$a_i = w_i^{(1)} x$$

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

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

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

$$\frac{\partial f}{\partial x} \quad \text{Chain rule equality}$$

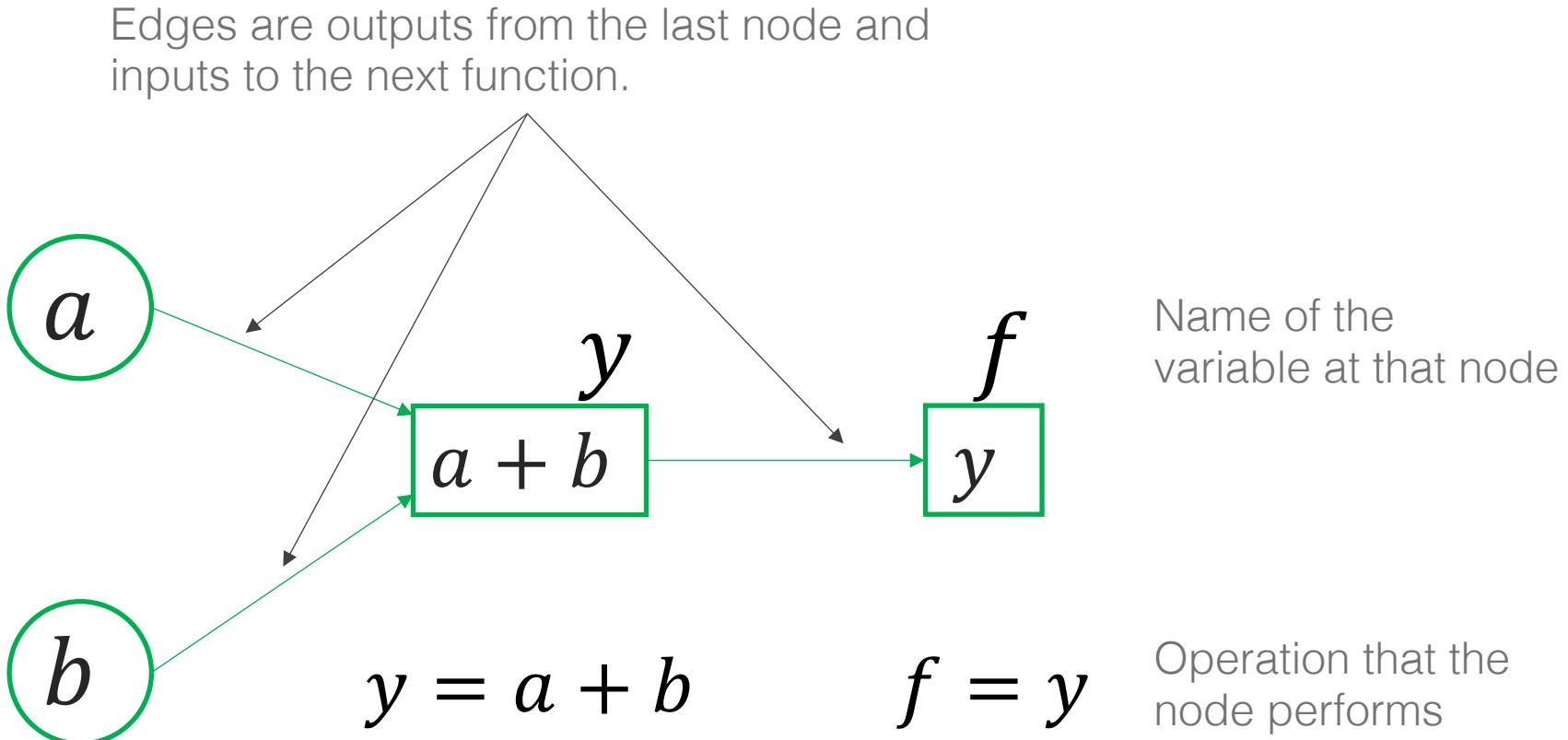
This process of using the next step in the chain rule is backpropagation

$$\begin{aligned} \frac{\partial f}{\partial z} &= \frac{\partial f}{\partial w} \underbrace{\frac{\partial w}{\partial x} \frac{\partial x}{\partial y} \frac{\partial y}{\partial z}} &= \frac{\partial f}{\partial x} \underbrace{\frac{\partial x}{\partial y} \frac{\partial y}{\partial z}} &= \frac{\partial f}{\partial y} \underbrace{\frac{\partial y}{\partial z}} \\ &\frac{\partial f}{\partial x} &\frac{\partial f}{\partial y} &= \frac{\partial f}{\partial z} \end{aligned}$$

Simple example

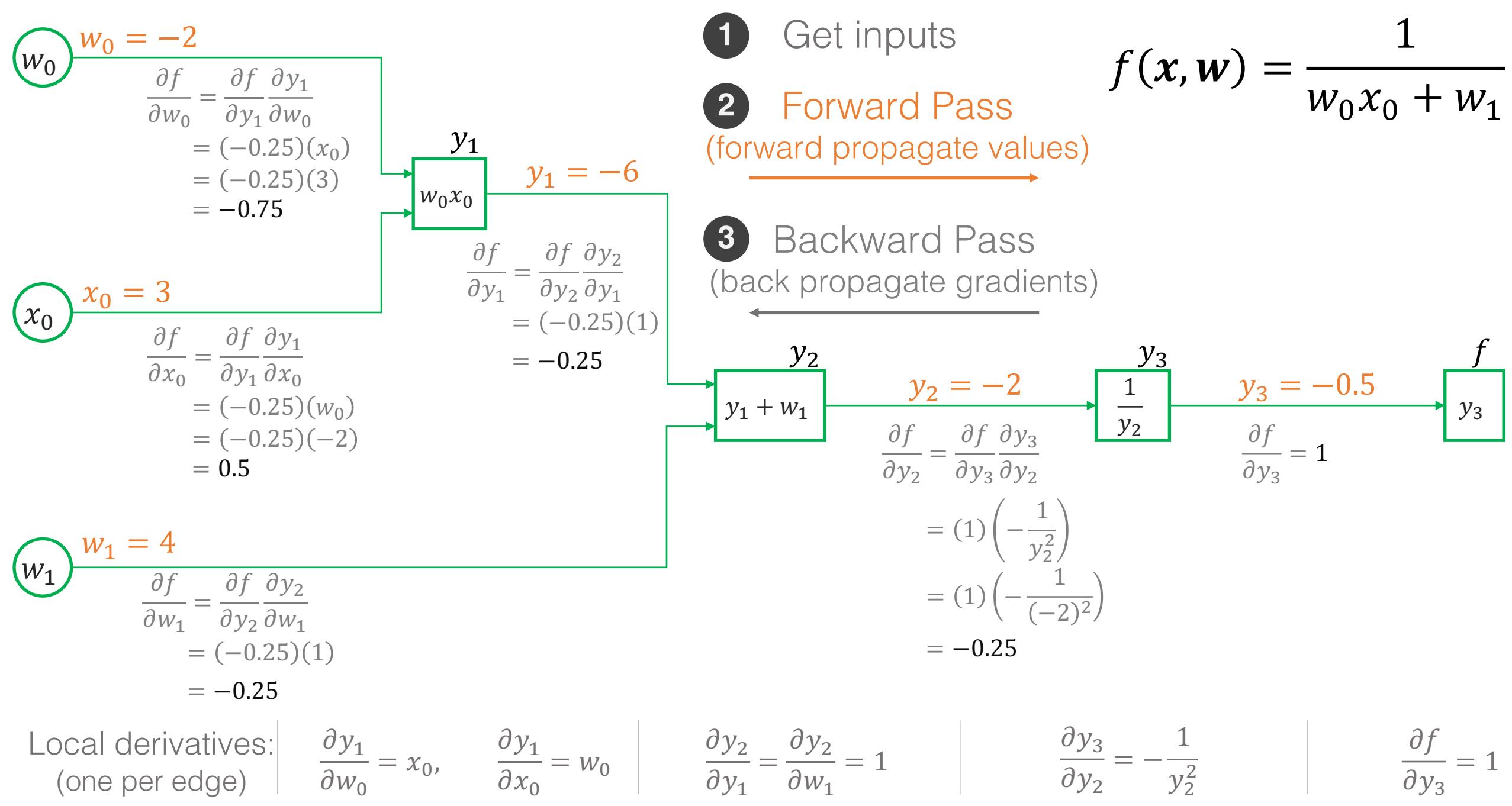
$$f(a, b) = a + b$$

This graph to the right represents this function

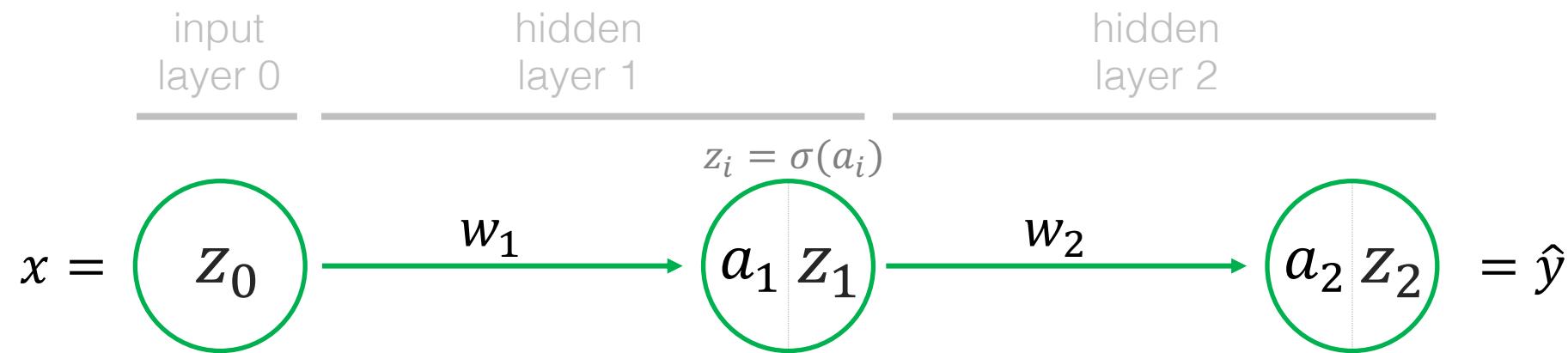


Local derivatives (one for each edge input into a node):

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



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



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

Forward
propagation

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

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

We can calculate the error:

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

We want to estimate the gradient with respect to each parameter

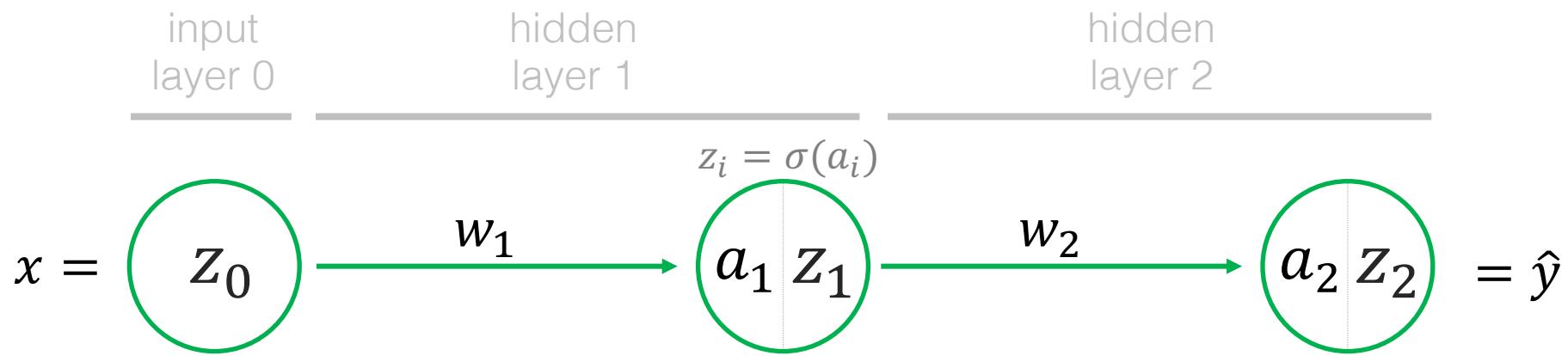
$\frac{\partial E}{\partial w_i}$ **Backpropagation**: an efficient way of calculating these values

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

$$a_2 = w_2 z_1$$

$$z_1 = \sigma(a_1)$$

$$a_1 = w_1 z_0 \rightarrow x$$



Forward propagation

$E = \frac{1}{2}(\hat{y} - y)^2$	$\frac{\partial E}{\partial \hat{y}} = \hat{y} - y$
$\hat{y} = \sigma(a_2)$	$\frac{\partial \hat{y}}{\partial a_2} = \sigma'(a_2)$
$a_2 = w_2 z_1$	$\frac{\partial a_2}{\partial z_1} = w_2$
$z_1 = \sigma(a_1)$	$\frac{\partial z_1}{\partial a_1} = \sigma'(a_1)$
$a_1 = w_1 z_0$	$\frac{\partial a_1}{\partial w_1} = z_0 = x$

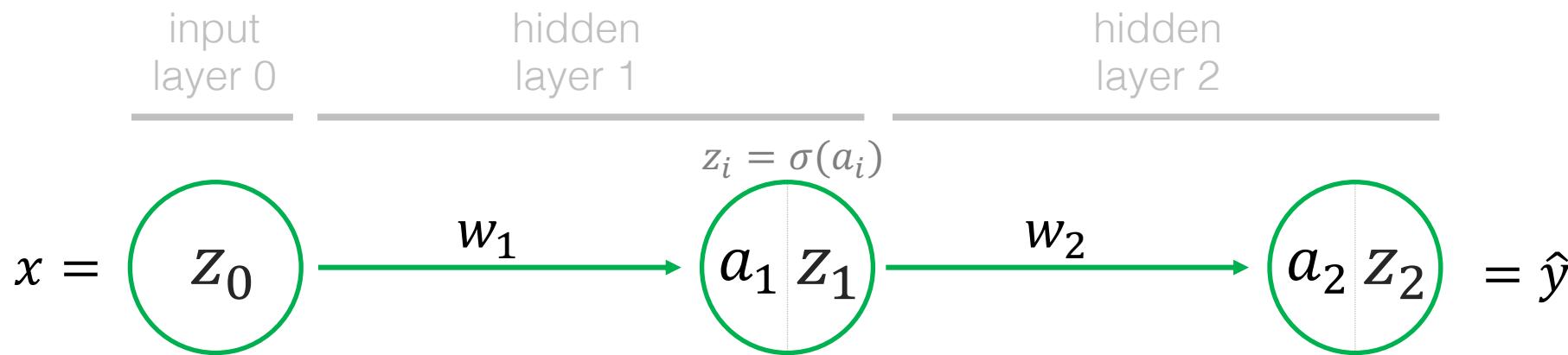
Backpropagation

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

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

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

We know all these quantities from forward propagation



Forward propagation

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

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

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

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

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

Backpropagation

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

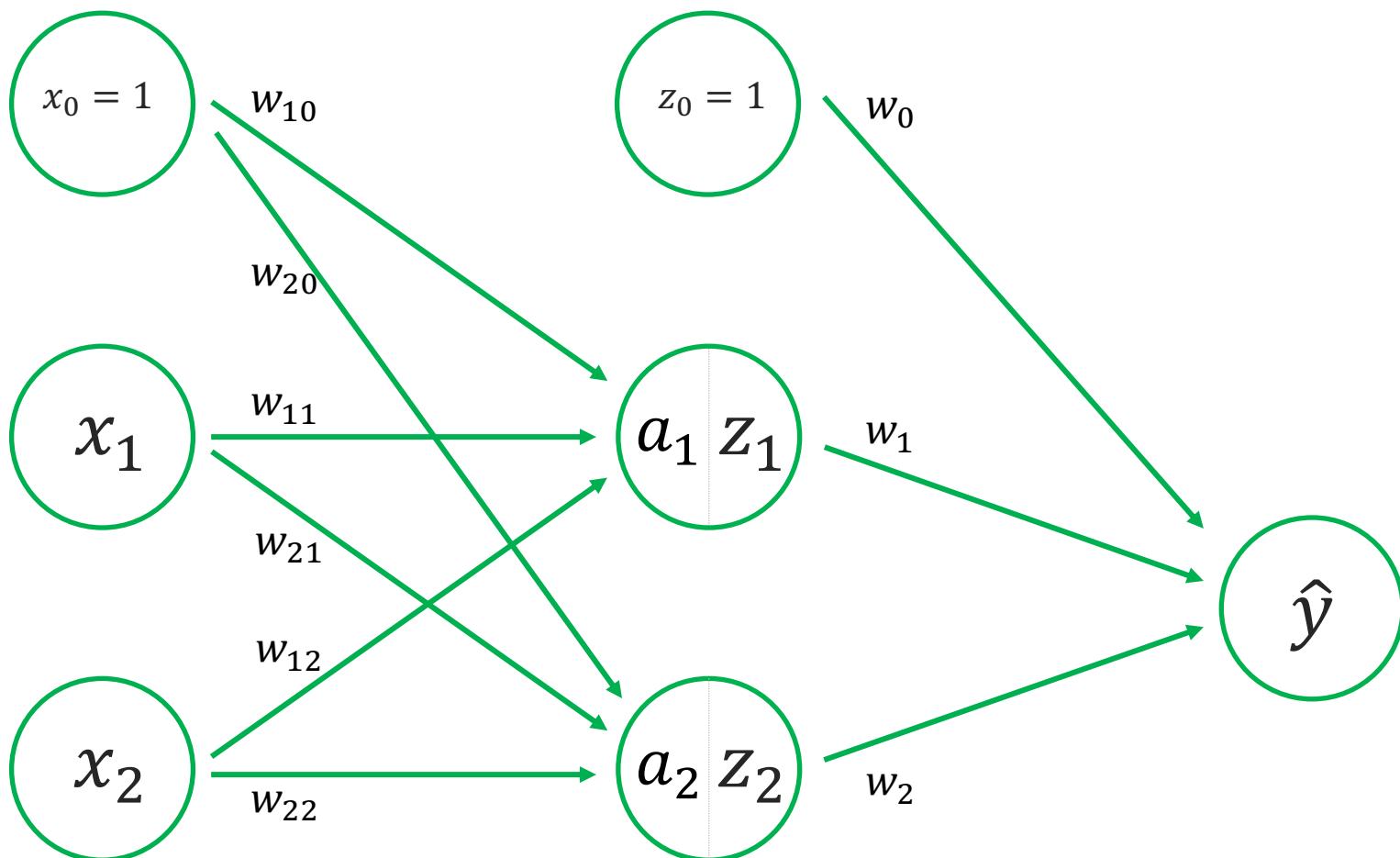
$$\frac{\partial E}{\partial a_1}$$

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

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

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)

$$a_i = w_{i0}x_0 + w_{i1}x_1 + w_{i2}x_2$$

$$z_i = g(a_i)$$

$$\hat{y} = w_0z_0 + w_1z_1 + w_2z_2$$

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

Local Derivative(s)

$$\frac{\partial a_i}{\partial w_{ij}} = x_j$$

$$\frac{\partial z_i}{\partial a_i} = g'(a_i)$$

$$\frac{\partial \hat{y}}{\partial z_i} = w_i$$

$$\frac{\partial \hat{y}}{\partial w_i} = z_i$$

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

Global Derivative(s)

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

$$\frac{\partial E}{\partial a_i} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i} \frac{\partial z_i}{\partial a_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i}$$

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

Quick reference for neural network math

5.1 Forward Propagation

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

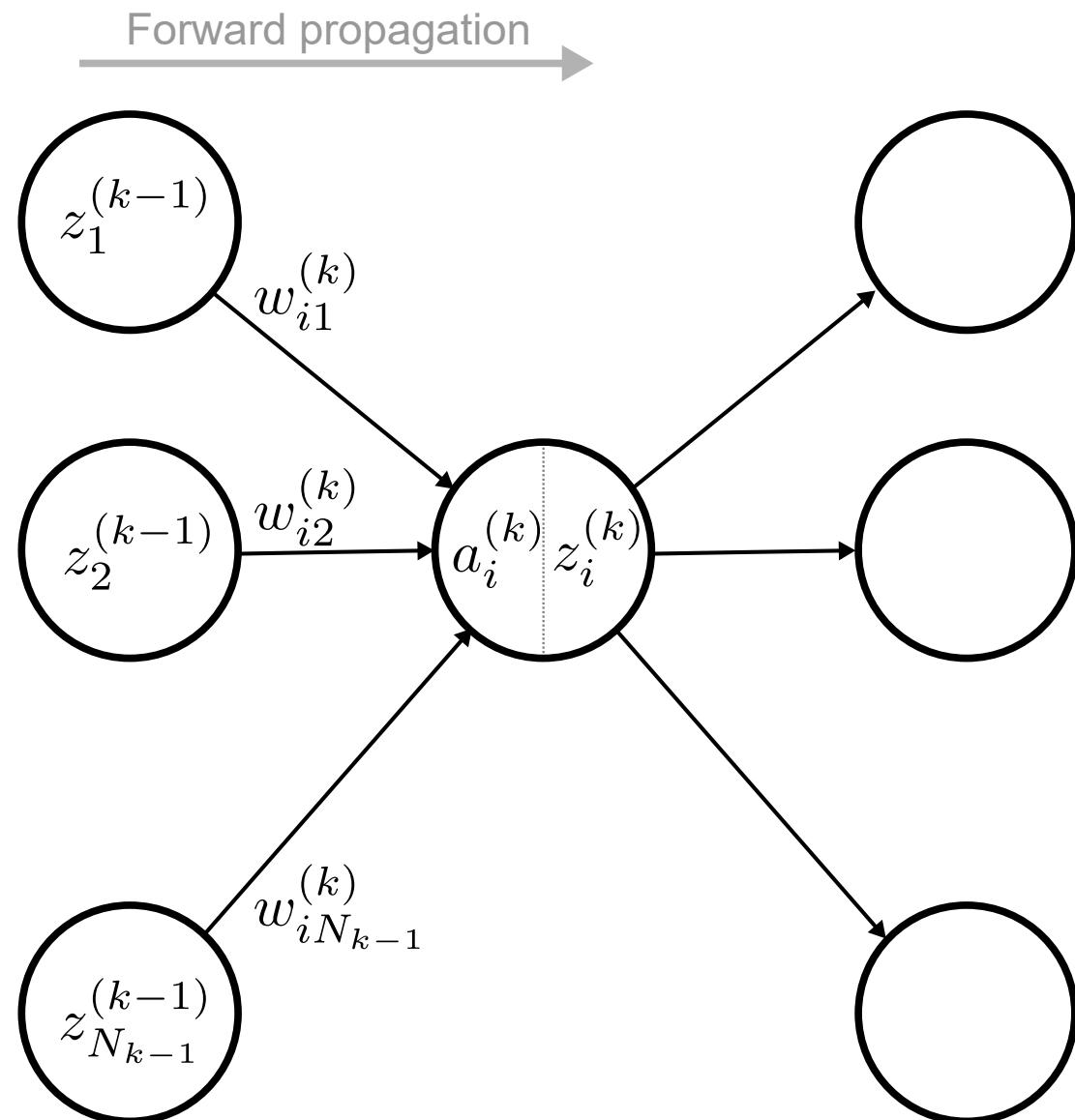
$$a_i^{(k)} = \sum_{j=1}^{N_{k-1}} w_{ij}^{(k)} z_j^{(k-1)}$$

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

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 $k = K - 1, \dots, 1$):

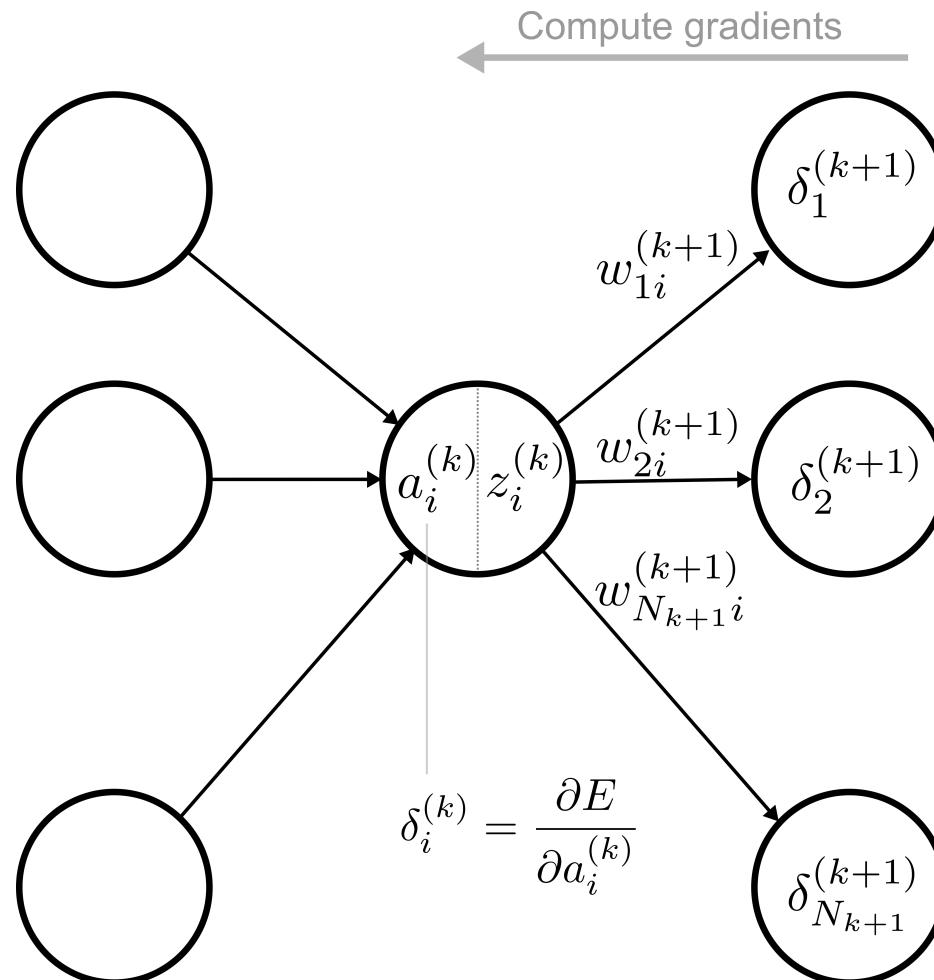
$$\delta_i^{(k)} \triangleq \frac{\partial E_n}{\partial a_i^{(k)}} = \sigma'(a_i^{(k)}) \sum_{j=1}^{N_{k+1}} \delta_j^{(k+1)} w_{ji}^{(k+1)}$$

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

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 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 $n = 1, 2, \dots, N$

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

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

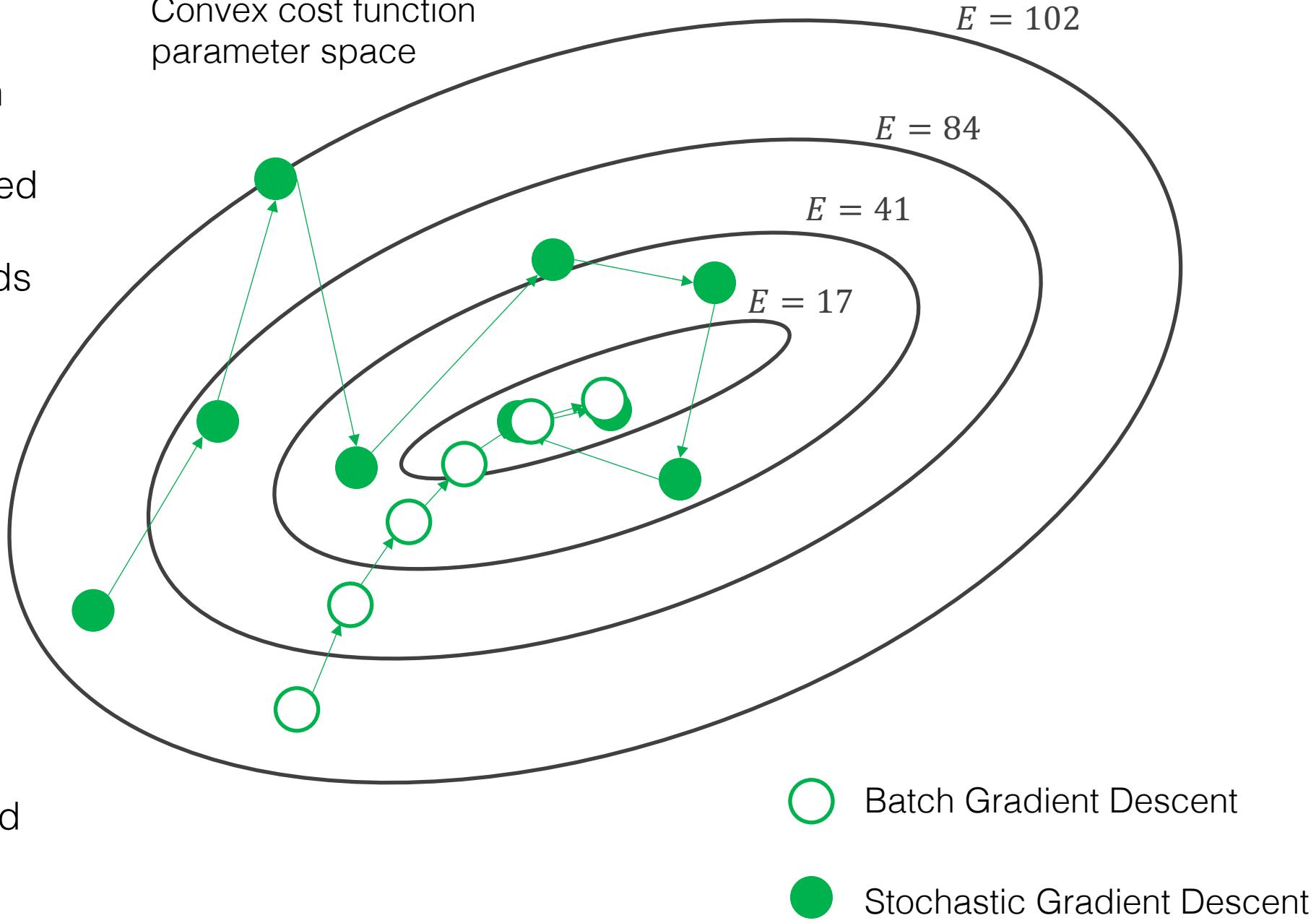
Batch gradient descent can work well if the cost function is convex (rare) and the learning rate is properly tuned

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



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

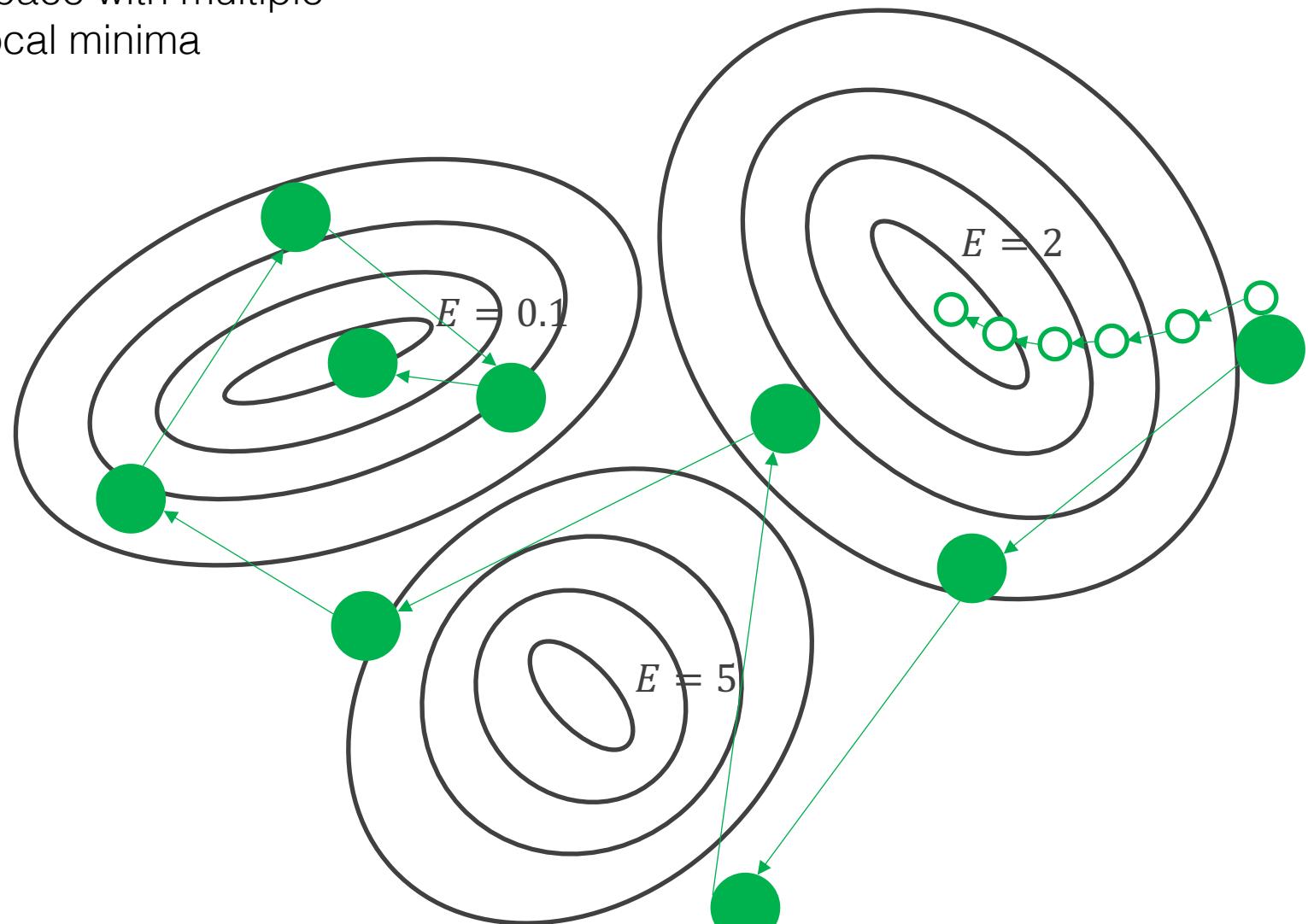
Stochastic gradient descent (SGD) is better at avoiding local minima, but injects noise into the training process

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



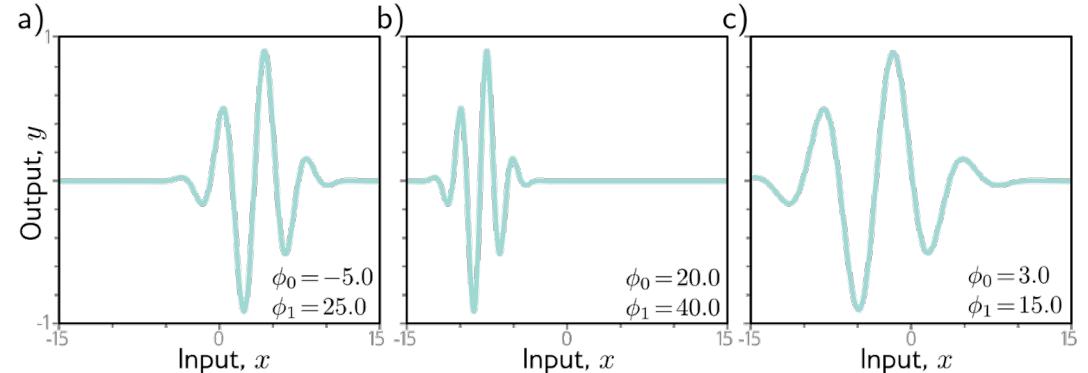
Batch Gradient Descent
Stochastic Gradient Descent



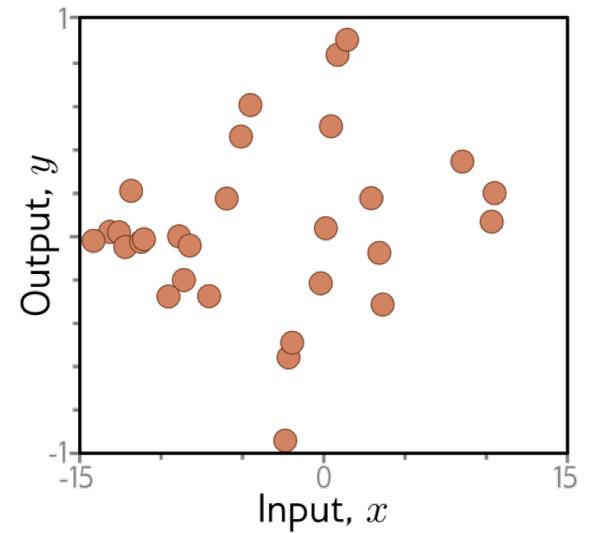
SGD Example

Gabor Model

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

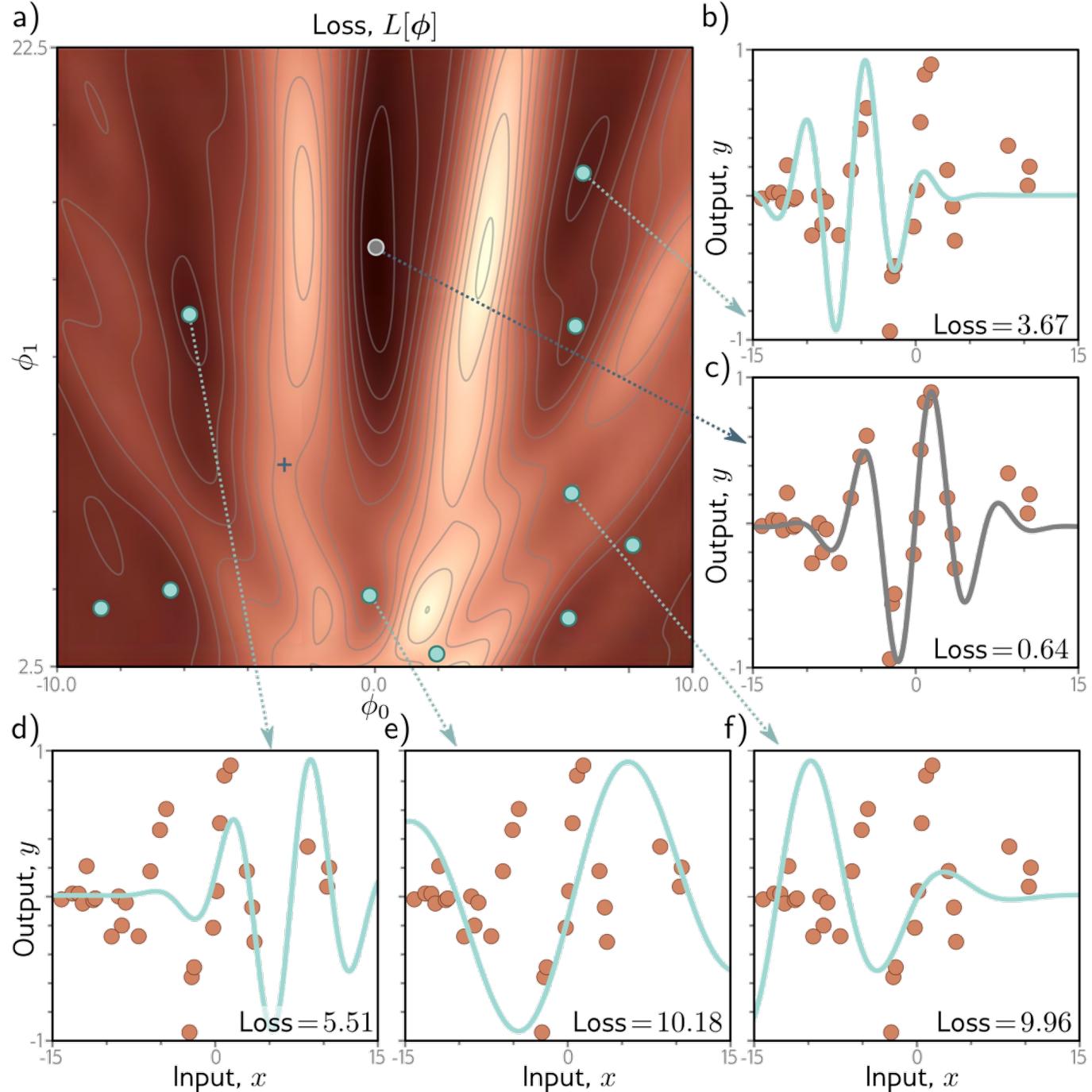


Training Data



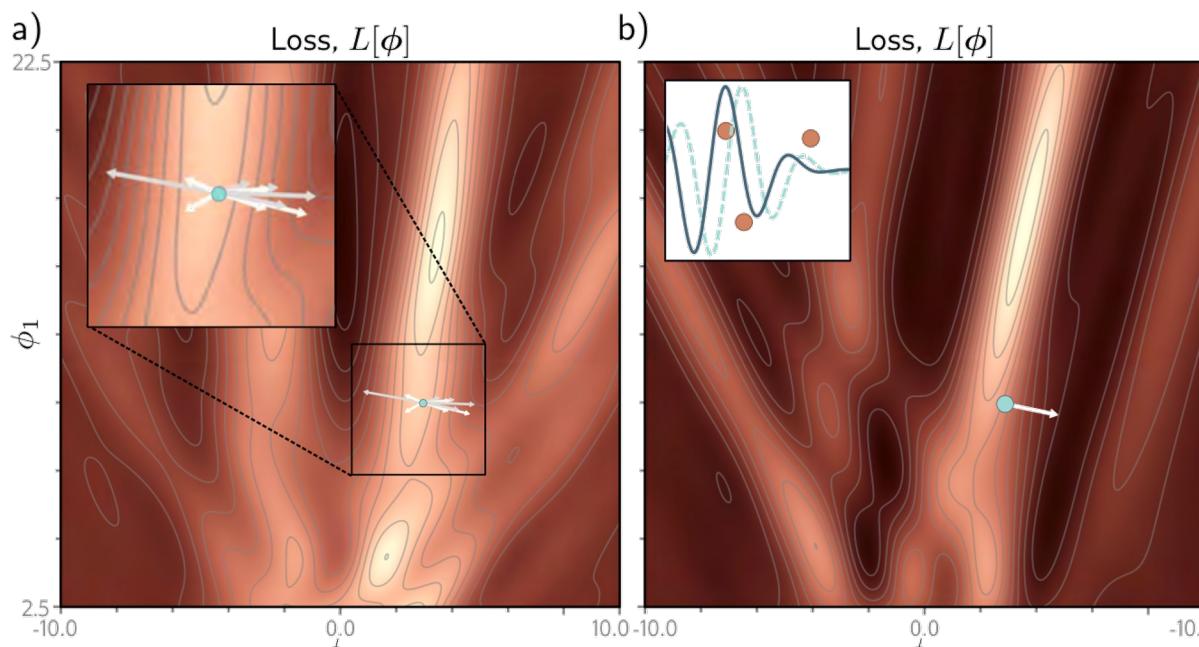
Global cost function for different parameter values

Images from Prince, Understanding Deep Learning, 2023



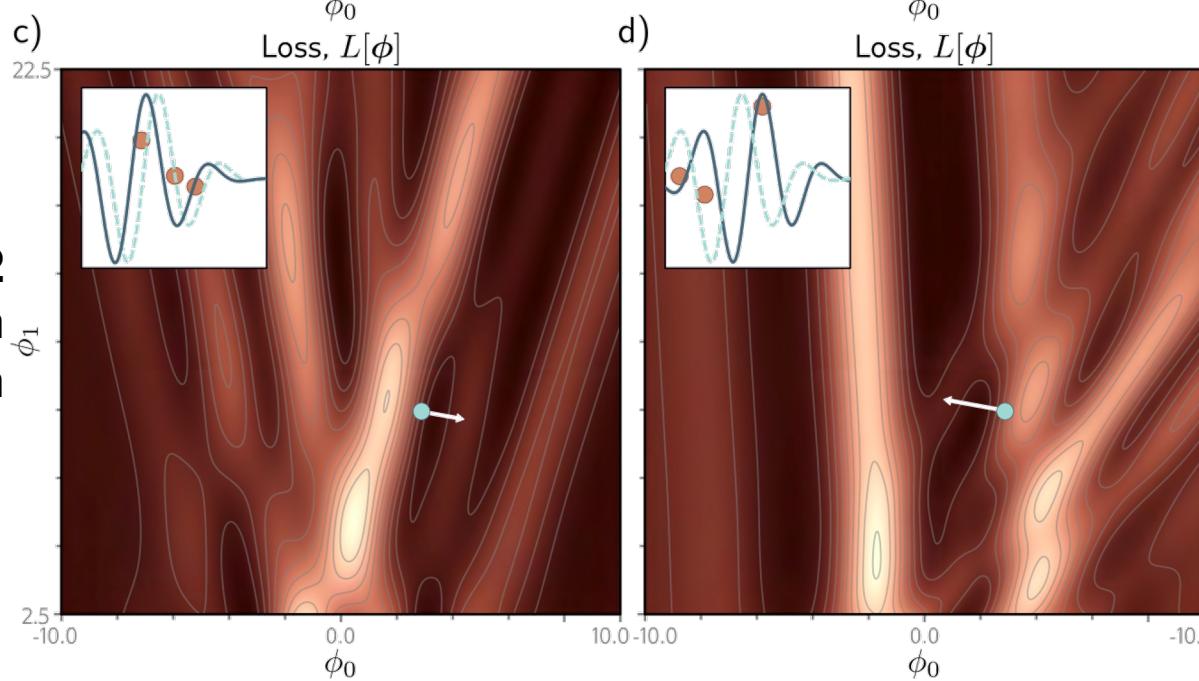
Global cost function

Cost function with the entire training dataset



Minibatch cost function 2

Cost function with minibatch



Minibatch cost function 1

Cost function with minibatch

Minibatch cost function 3

Cost function with minibatch

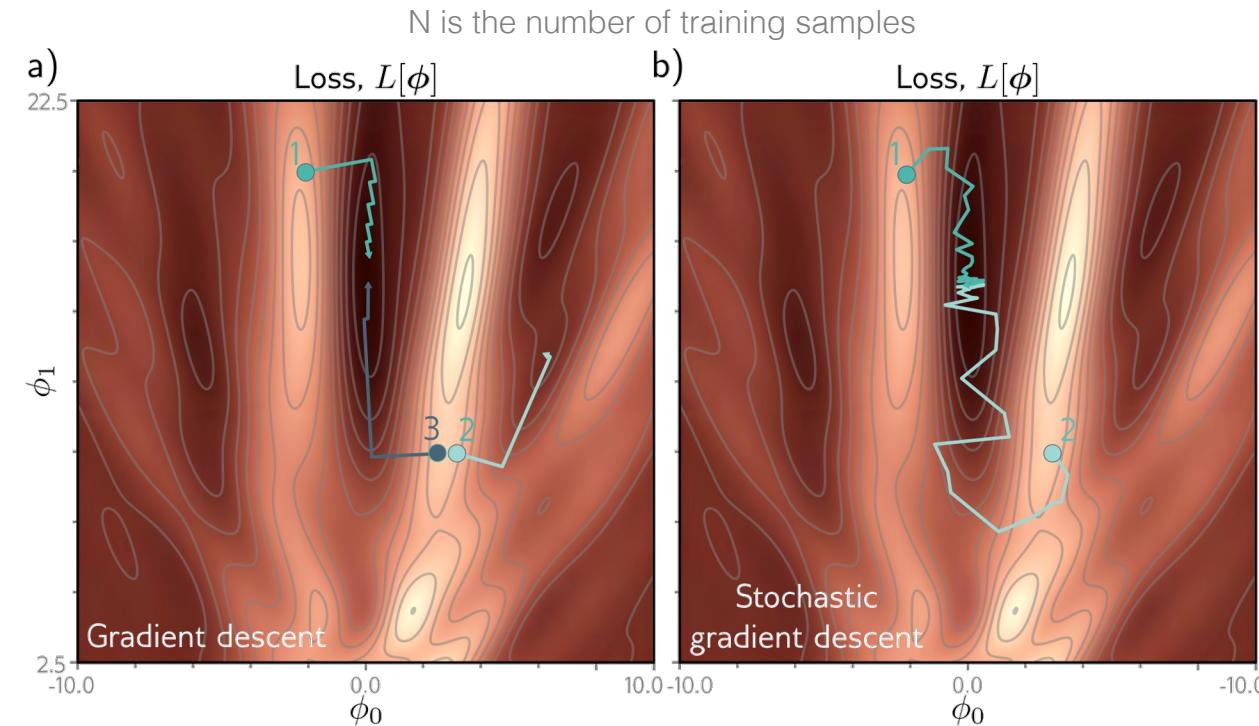
SGD Example

Images from Prince, Understanding Deep Learning, 2023

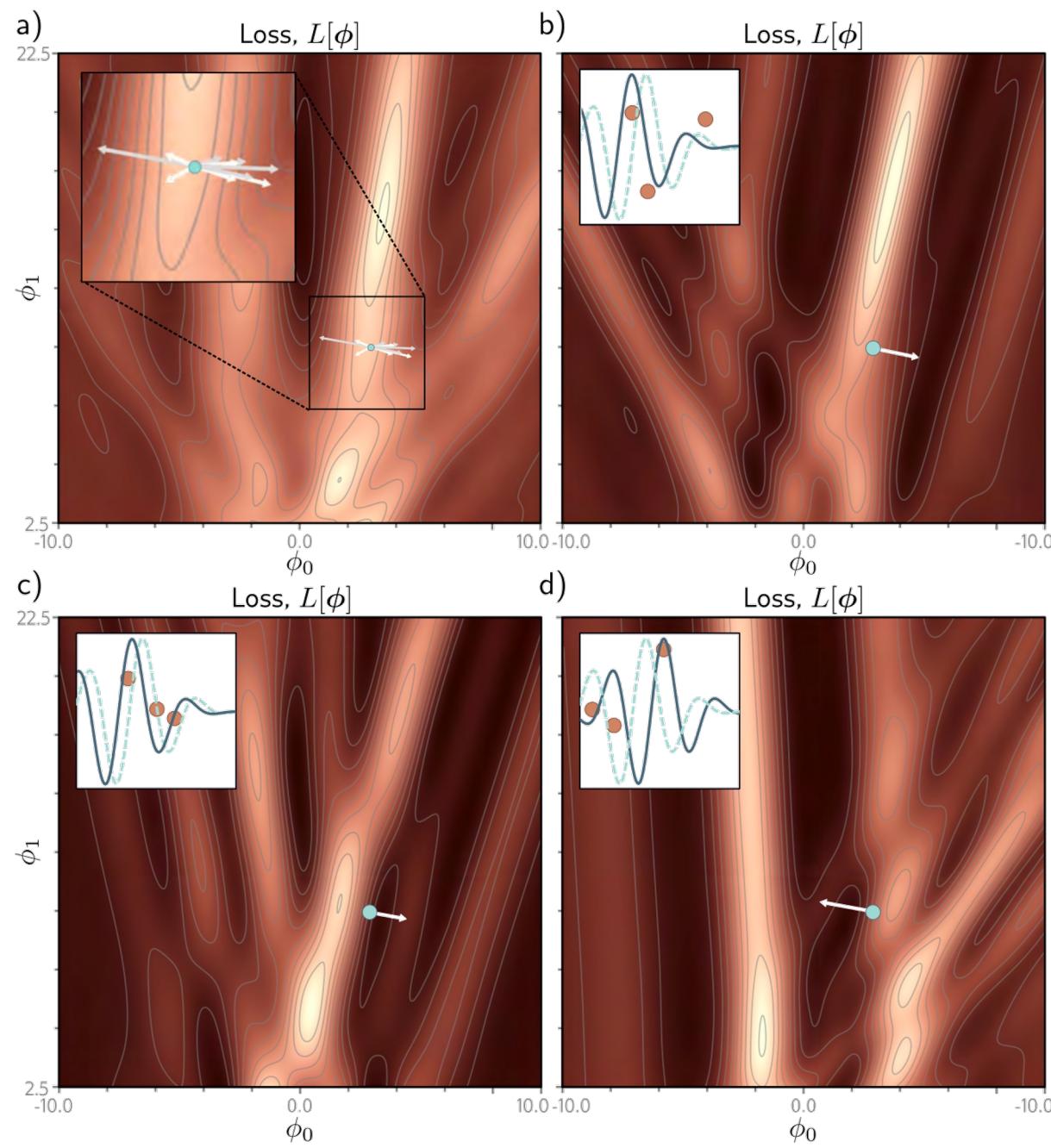
SGD Example

Images from Prince, Understanding Deep Learning, 2023

Gradient descent
Batch size = N



**Minibatch stochastic
gradient descent**
1 < batch size < N



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

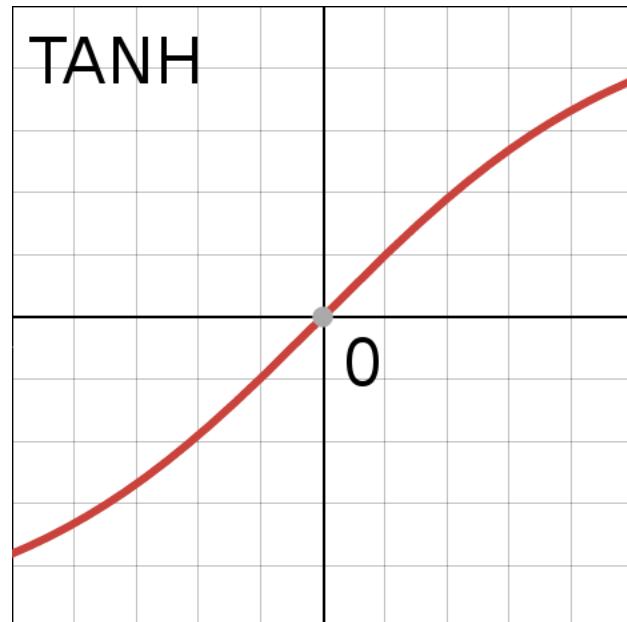
How do we **choose model weights?**
(i.e. how do we fit our model to data)

What are the challenges of using neural networks?

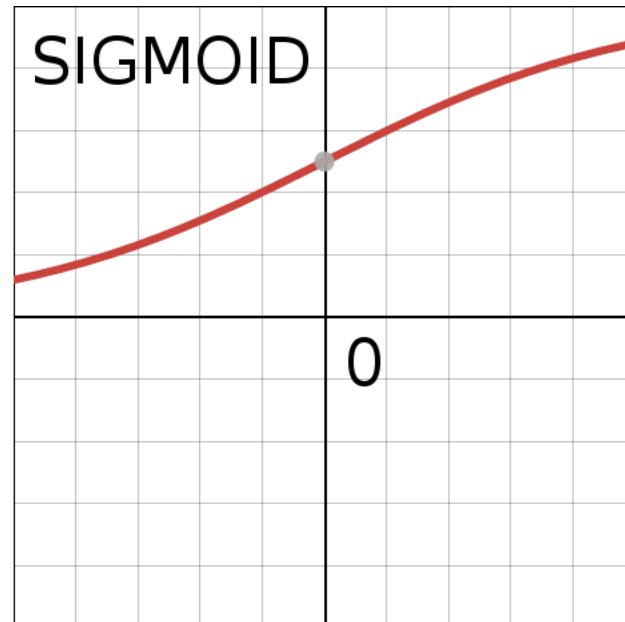
Hyperparameter / Architectural choices

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

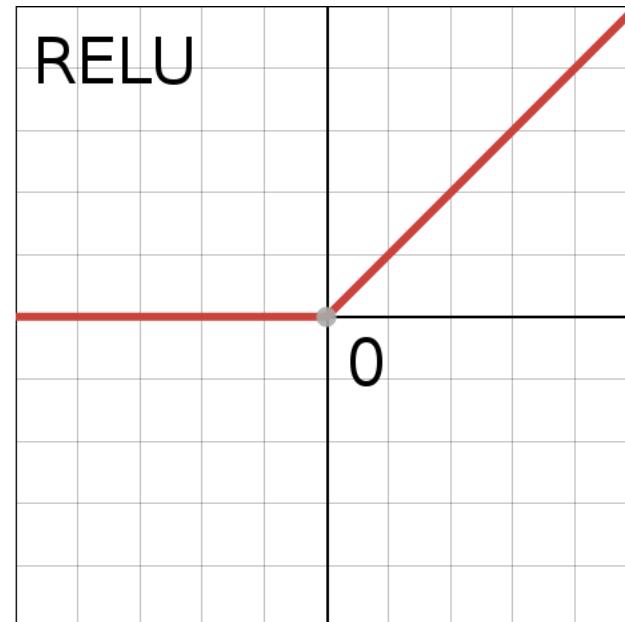
Activation Functions



Hyperbolic Tangent



Sigmoid Tangent



Rectified linear unit (ReLU)

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

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

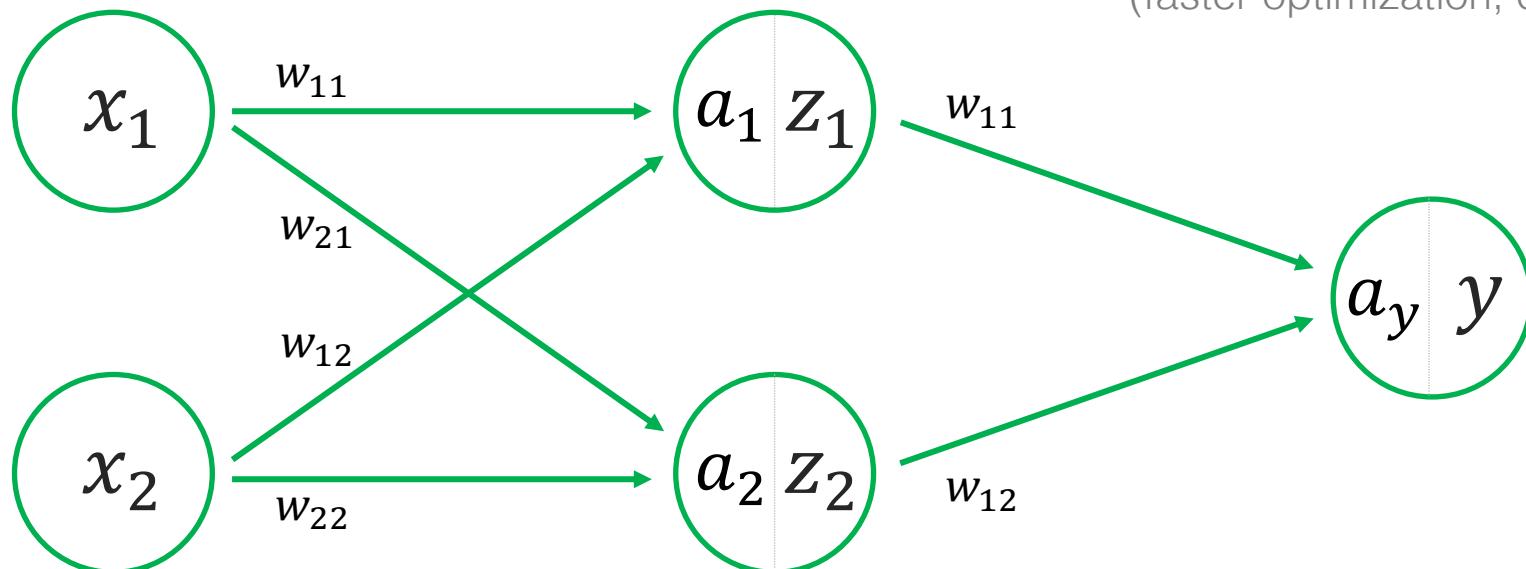
Random numbers

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

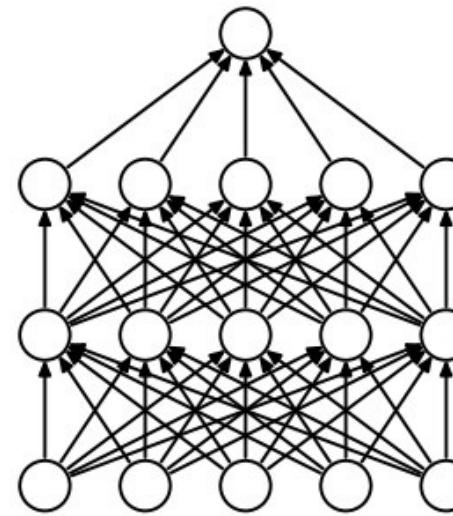
Regularization

L2 Regularization

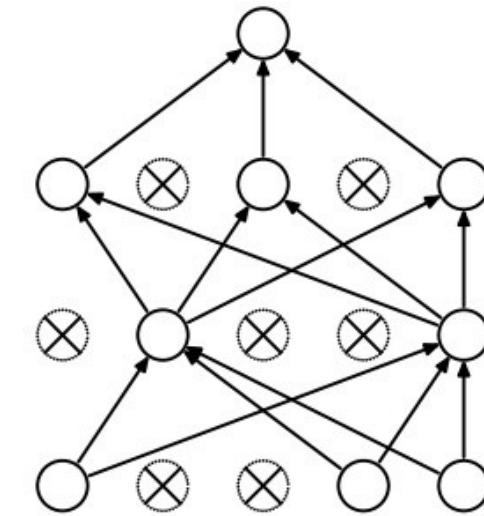
L1 Regularization

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.

Watch your learning curves

Questions to ask:

- Learning anything?
- Still learning?
- Overfitting?
- Learning rate too low/high?

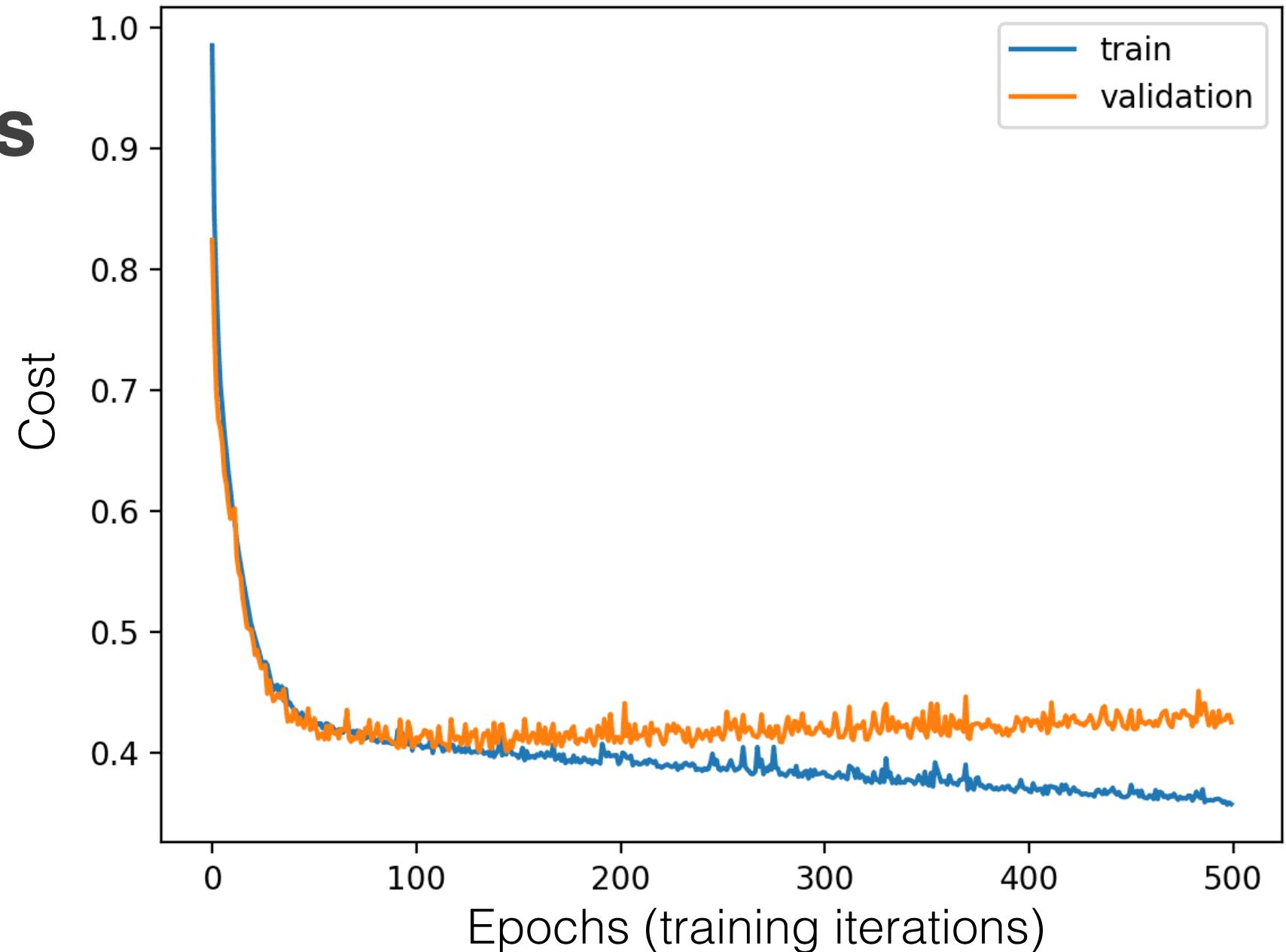


Image by Jason Brownlee

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

- Who is this document for?
- Why a tuning playbook?
- Guide for starting a new project
 - Choosing the model architecture
 - Choosing the optimizer
 - Choosing the batch size
 - Choosing the initial configuration
- A scientific approach to improving model performance
 - The incremental tuning strategy
 - Exploration vs exploitation
 - Choosing the goal for the next round of experiments
 - Designing the next round of experiments
 - Determining whether to adopt a training pipeline change or hyperparameter configuration
 - After exploration concludes
- Determining the number of steps for each training run
 - Deciding how long to train when training is not compute-bound

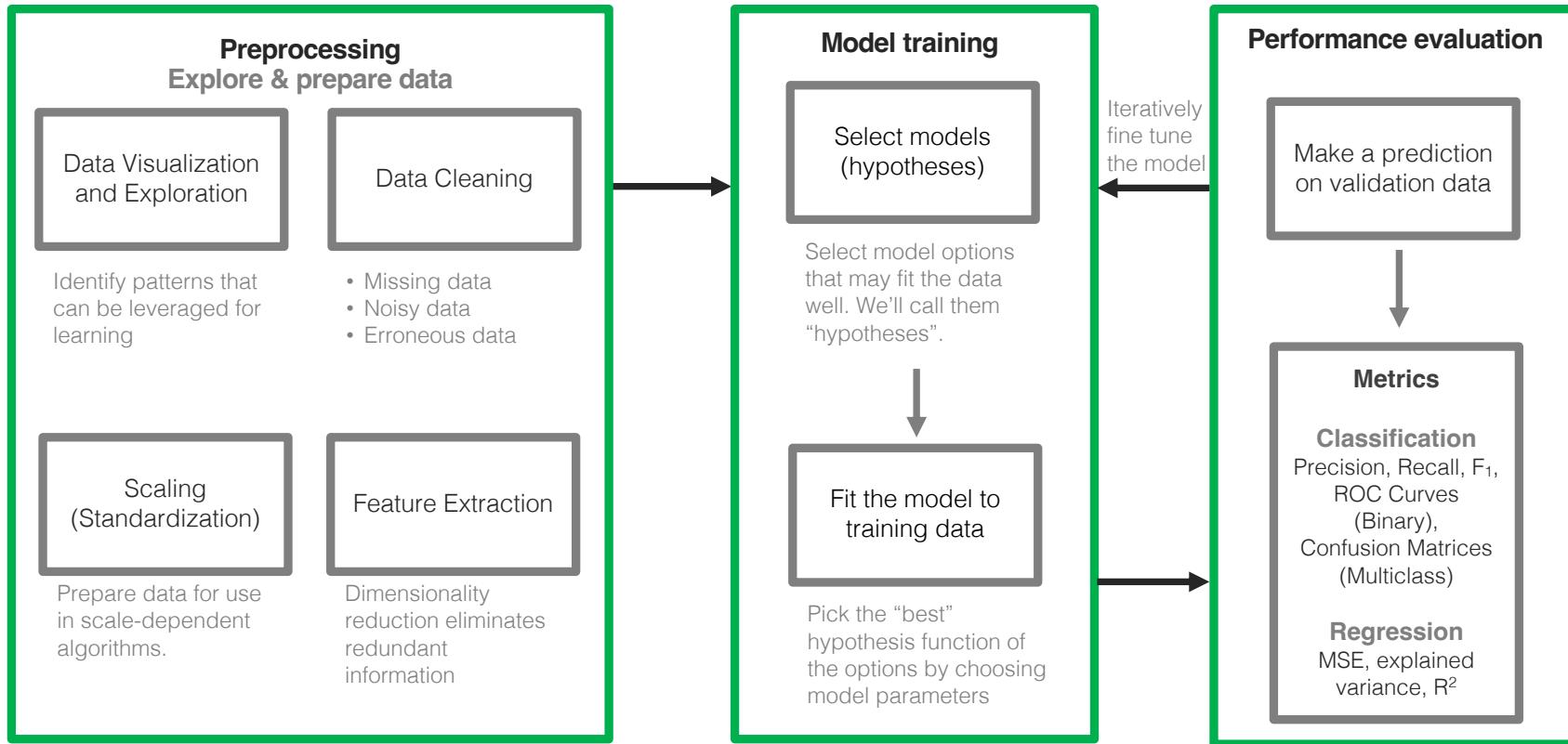
Experimental Design

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?



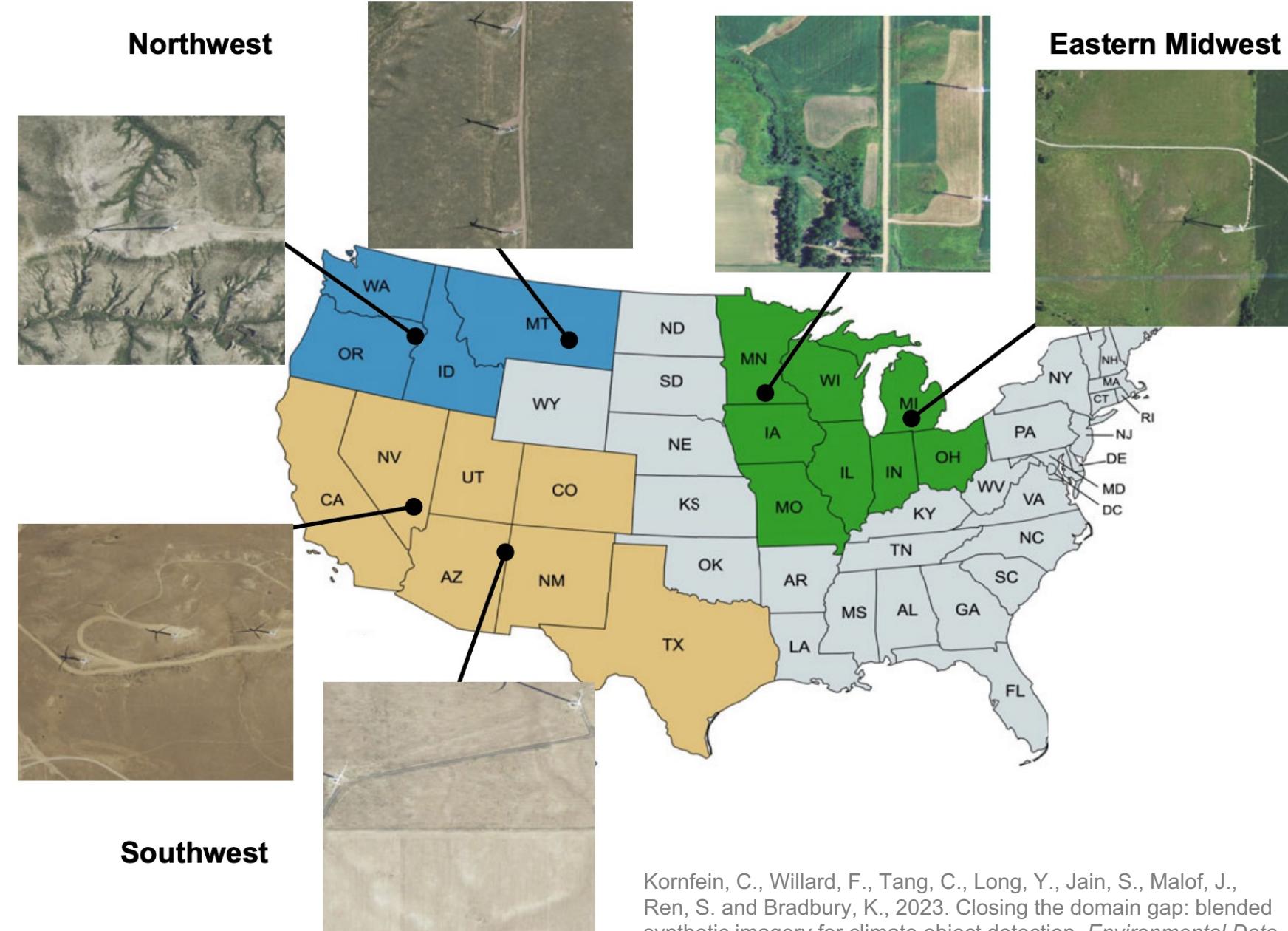
- Train/validation/test split
 - How it was split
 - The split itself
- Data imputation
- Feature selection
- Scaling
- Dimensionality reduction
- Feature transformations

- 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?

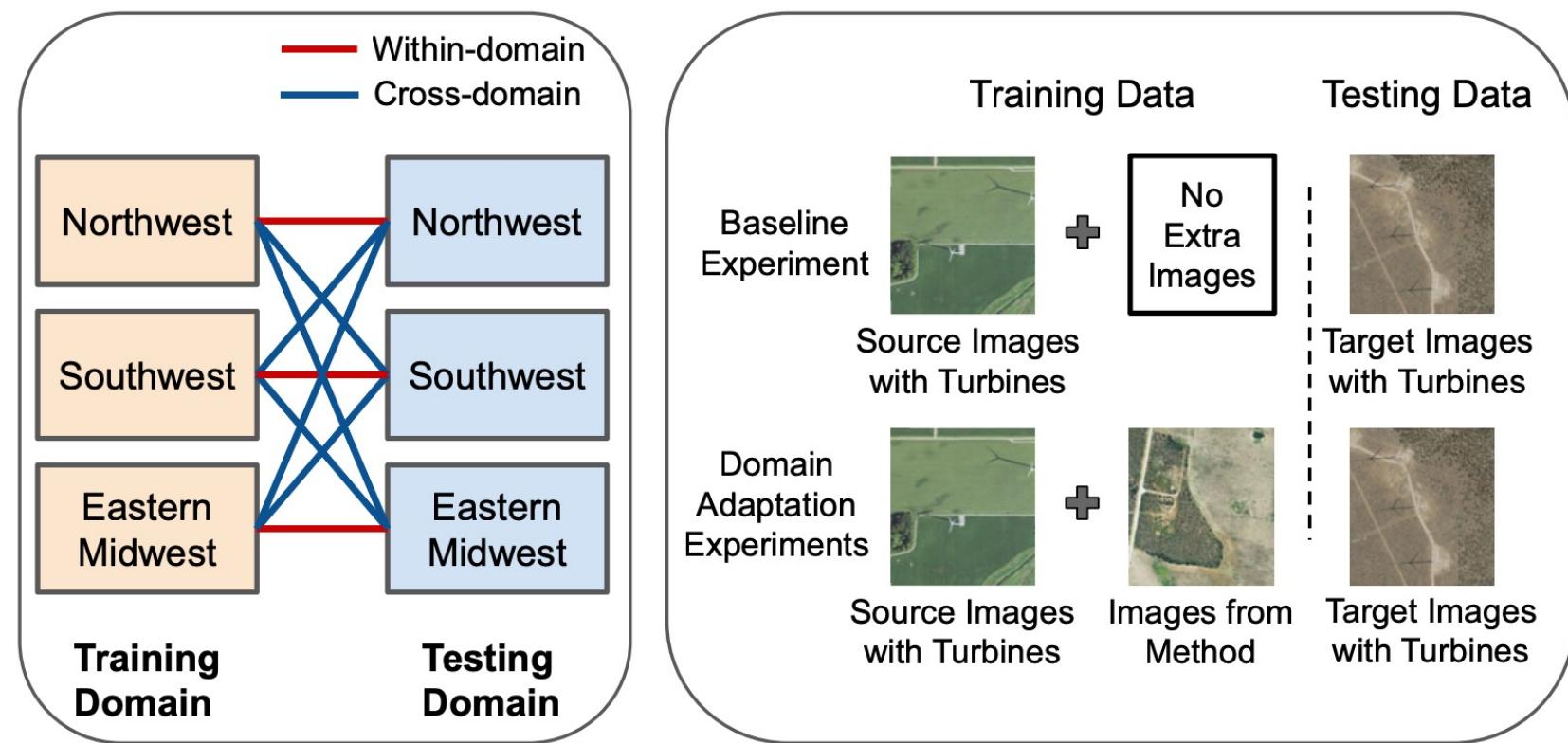


Kornfein, C., Willard, F., Tang, C., Long, Y., Jain, S., Malof, J., Ren, S. and Bradbury, K., 2023. Closing the domain gap: blended synthetic imagery for climate object detection. *Environmental Data Science*, 2, p.e39.

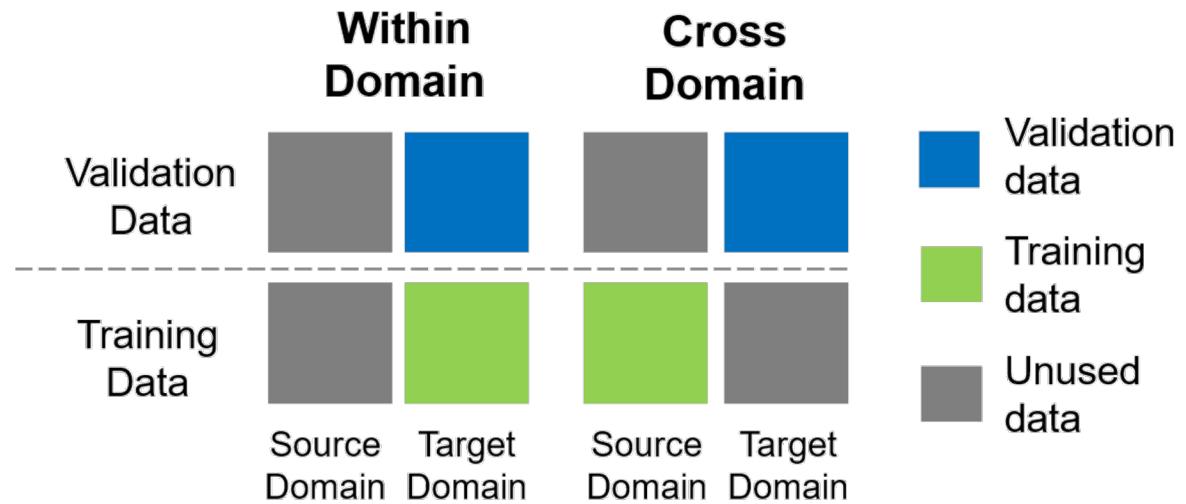
Experimental Design Example

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.

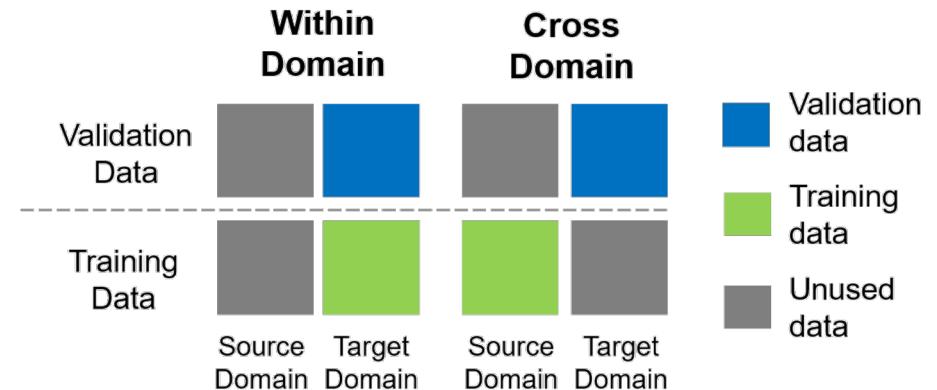
Kornfein, C., Willard, F., Tang, C., Long, Y., Jain, S., Malof, J., Ren, S. and Bradbury, K., 2022. "Closing the Domain Gap—Blended Synthetic Imagery for Climate Object Detection." In 2022 Neural Information Processing Systems (NeurIPS) Climate Change AI Workshop.

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