PAT 463/563 (Fall 2025)

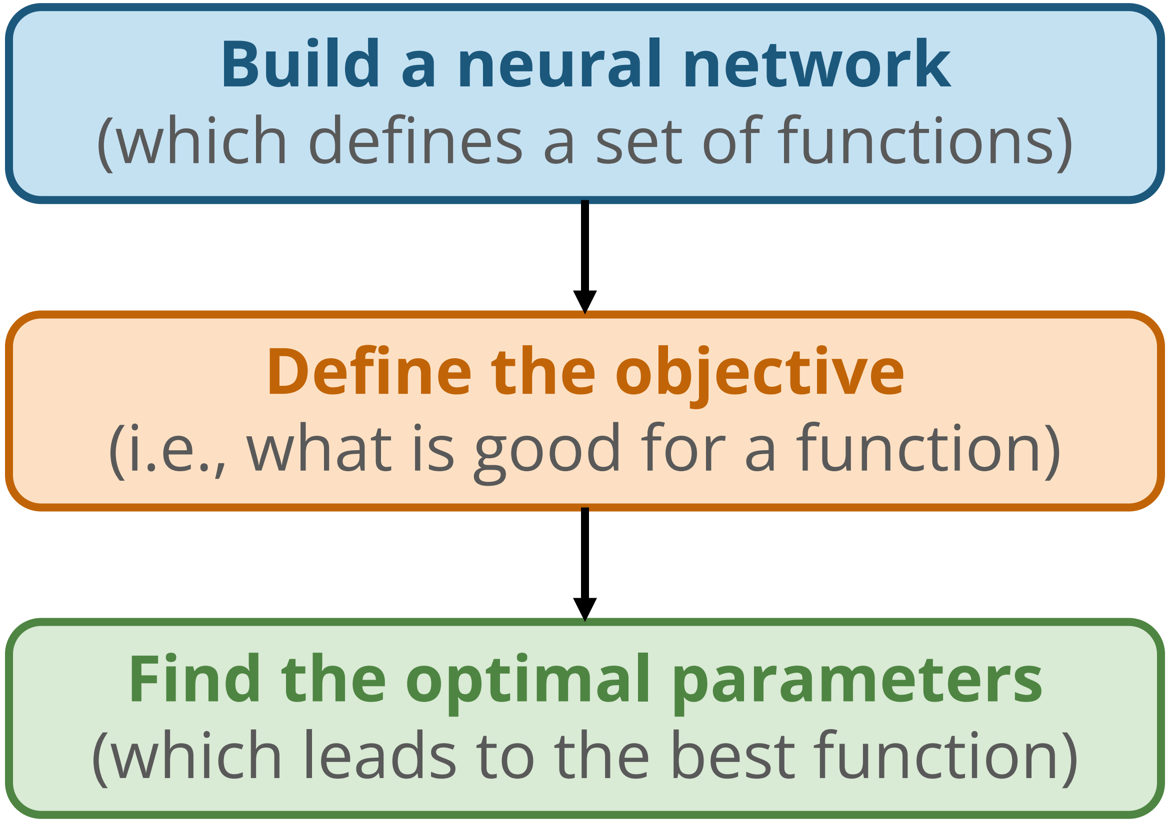
Music & AI

Lecture 7: Deep Learning Fundamentals II

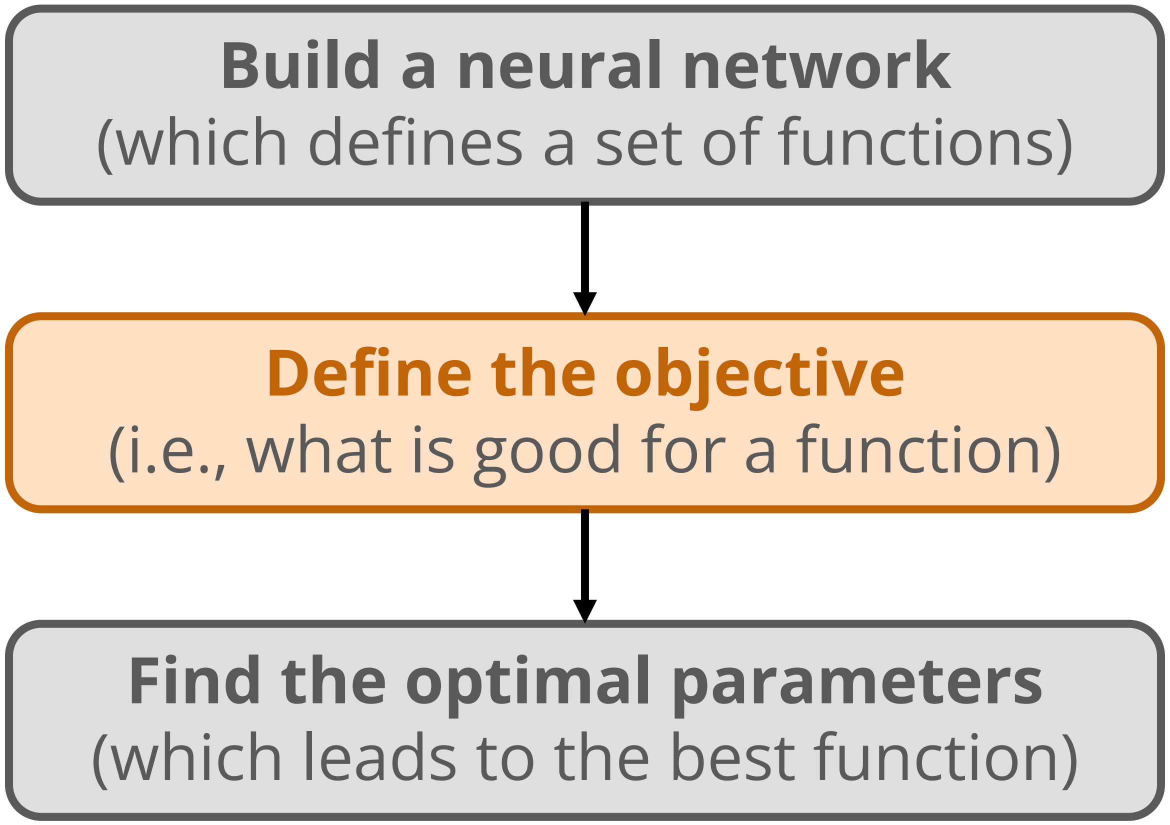
Instructor: Hao-Wen Dong

Training a Neural Network

Training a Neural Network

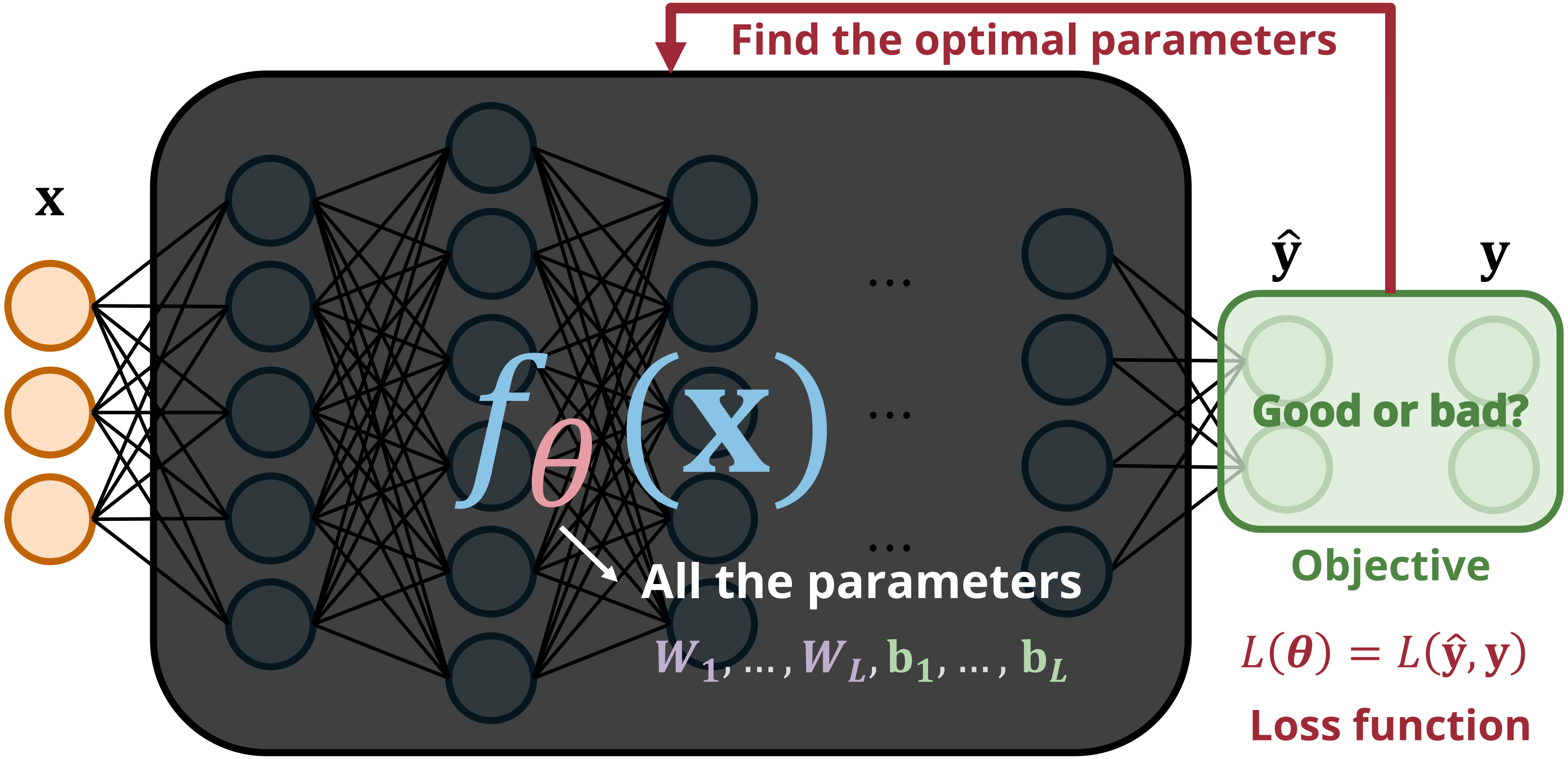


Training a Neural Network



Neural Networks are Parameterized Functions

- A neural network represents **a set of functions**



Loss Function

- Measure **how well the model perform** (in the opposite way)
- The choice of loss function depends on the task and the goals

$$L(\boldsymbol{\theta}) = L(\hat{\mathbf{y}}, \mathbf{y})$$

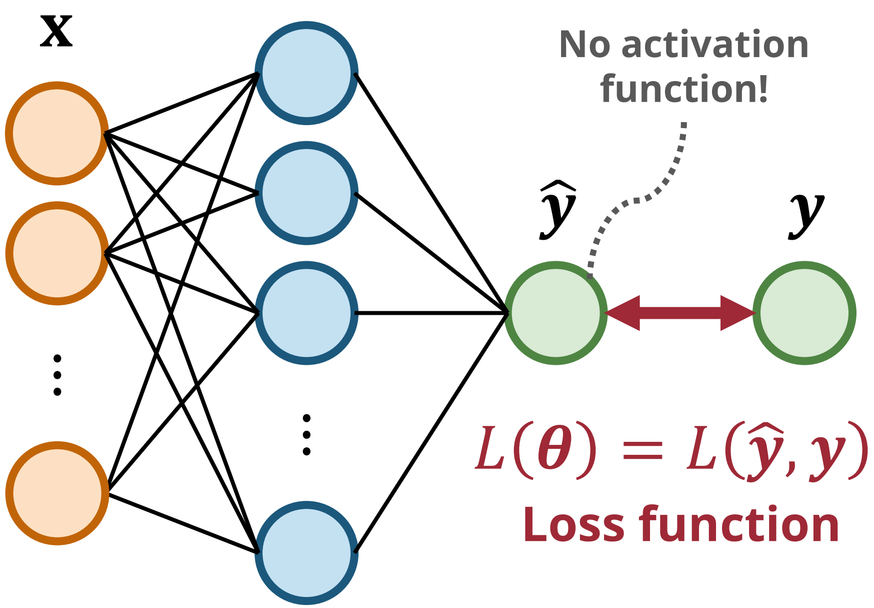
Loss Function: The Many Names

- Sometimes called
  - **Cost** function
  - **Error** function
- The opposite is known as
  - **Objective** function
  - **Reward** function (reinforcement learning)
  - **Fitness** function (evolutionary algorithms & genetic algorithms)
  - **Utility** function (economics)
  - **Profit** function (economics)

Example: Audio Codec

- What would be **a good objective to train a neural audio codec**?
- What do we **care about** for a codec?
  - Reconstruction quality **Trainable**
  - Bit rate (compression rate) **Likely not trainable but searchable**
  - Encoding/decoding speed **Likely not trainable but searchable**
- How do we measure **reconstruction quality**?
  - Difference in raw waveforms?
  - Difference in spectrograms?
  - Perceptual quality (psychoacoustics)?

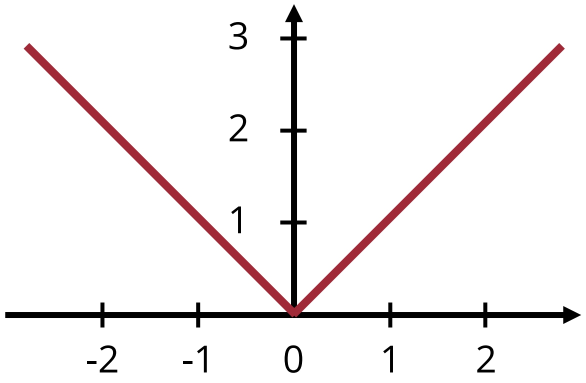
Common Loss Functions for Regression



Why not $L(\hat{y}, y) = \hat{y} - y$?

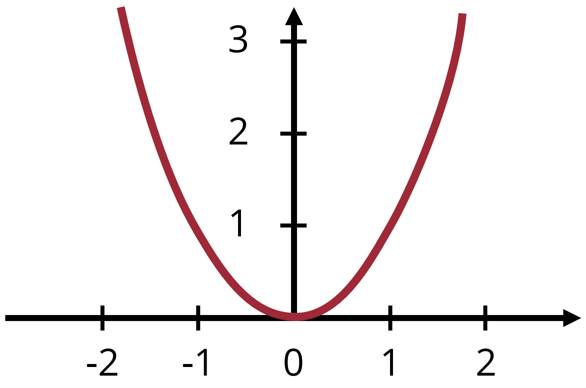
L1 loss

$$L(\hat{y}, y) = |\hat{y} - y|$$



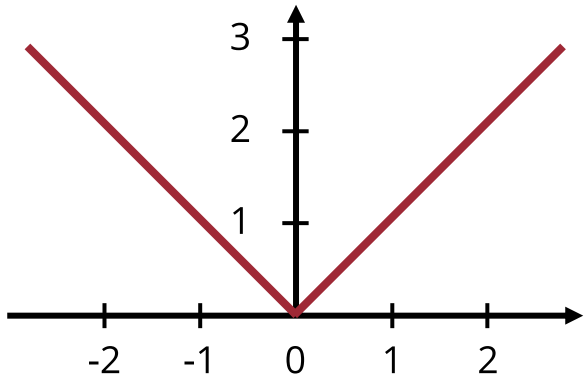
L2 loss

$$L(\hat{y}, y) = (\hat{y} - y)^2$$



L1 vs L2 Losses

L1 loss

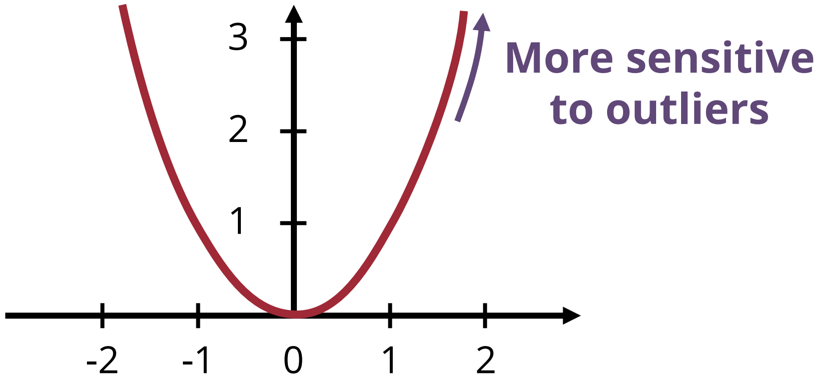


$$L(\hat{y}, y) = |\hat{y} - y|$$

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \mathbf{MAE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Mean Absolute Error (MAE)

L2 loss



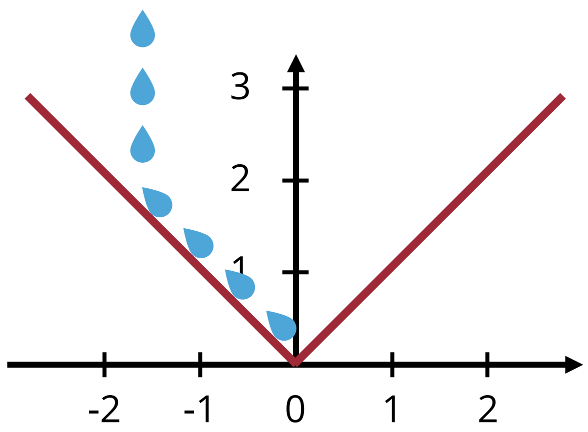
$$L(\hat{y}, y) = (\hat{y} - y)^2$$

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \mathbf{MSE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Mean Squared Error (MSE)

L1 vs L2 Losses

L1 loss

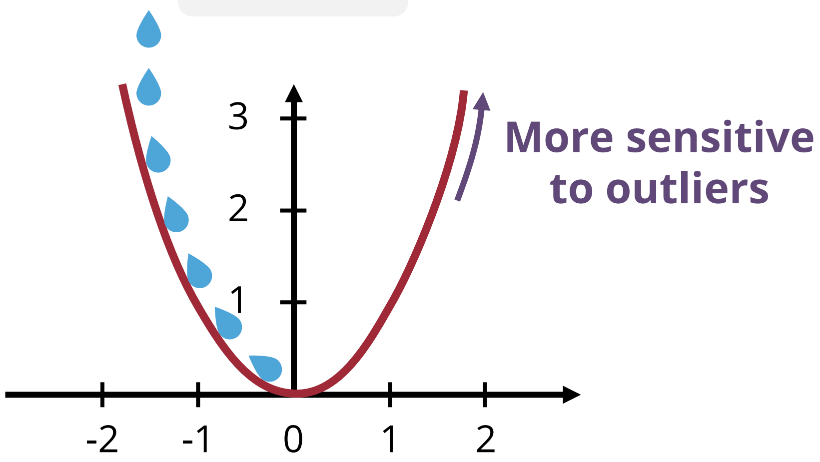


$$L(\hat{y}, y) = |\hat{y} - y|$$

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \mathbf{MAE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Mean Absolute Error (MAE)

L2 loss



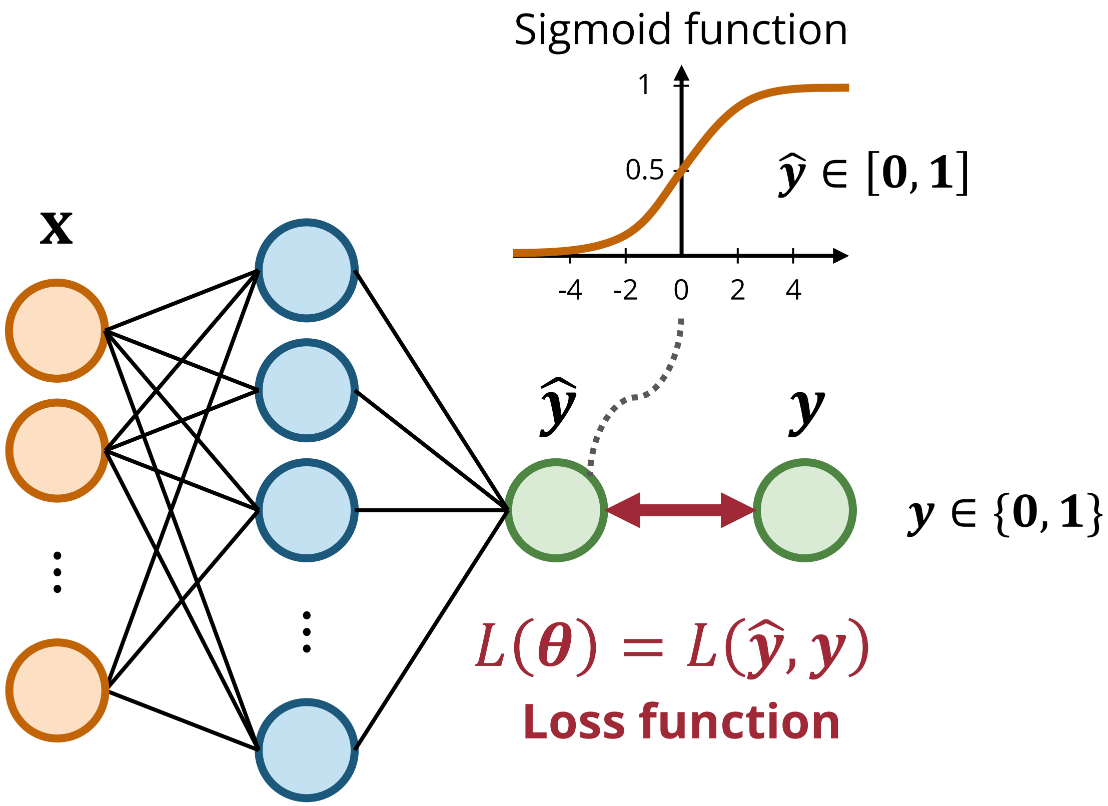
$$L(\hat{y}, y) = (\hat{y} - y)^2$$

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \mathbf{MSE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Mean Squared Error (MSE)

Binary Cross Entropy for Binary Classification

- **Logistic regression** approaches classification like regression

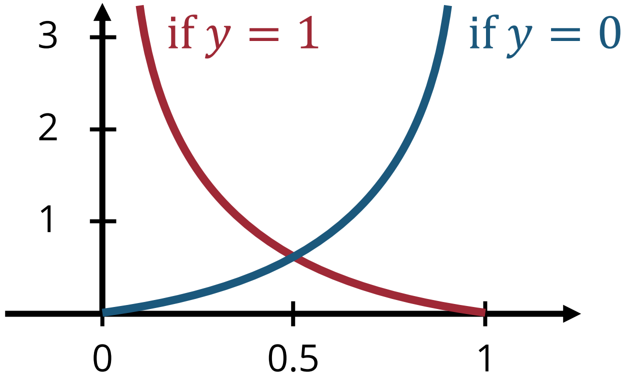


Binary cross entropy

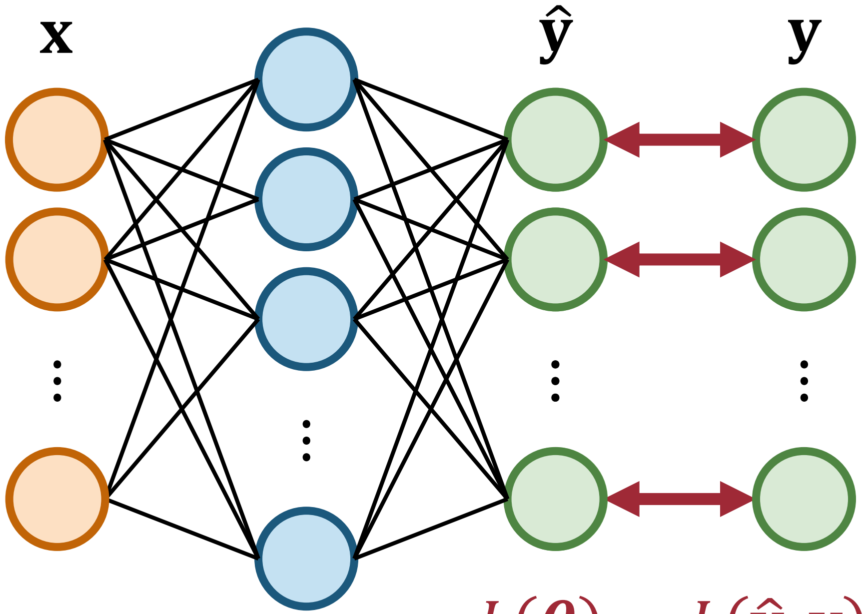
(Also called log loss)

$$L(\hat{y}, y) = \begin{cases} -\log \hat{y}\,, & \text{if } y = 1 \\ -\log(1 - \hat{y})\,, & \text{if } y = 0 \end{cases}$$

$$= -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$



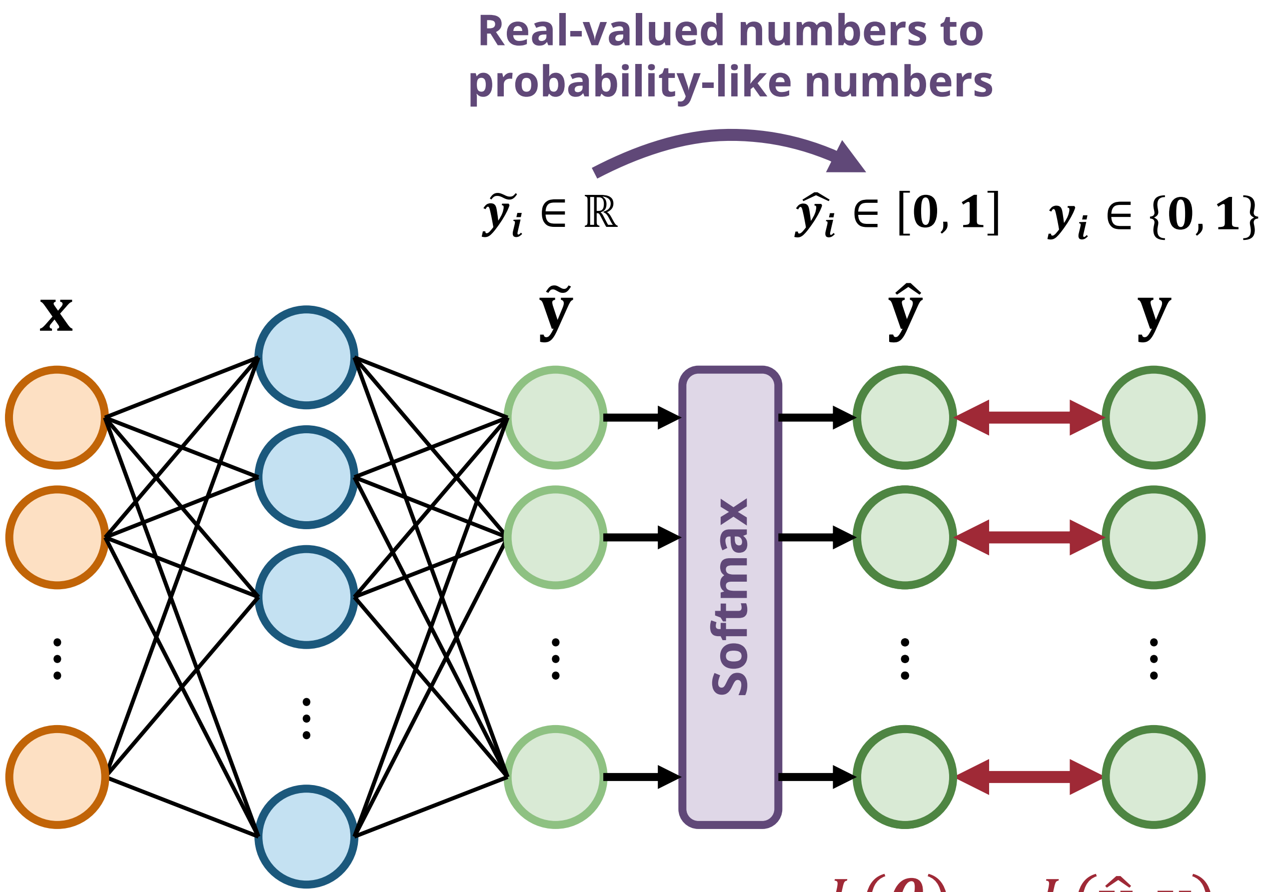
Cross Entropy for Multiclass Classification



$L(\boldsymbol{\theta}) = L(\hat{\mathbf{y}}, \mathbf{y})$

Loss function

Cross Entropy for Multiclass Classification



$$L(\boldsymbol{\theta}) = L(\hat{\mathbf{y}}, \mathbf{y})$$

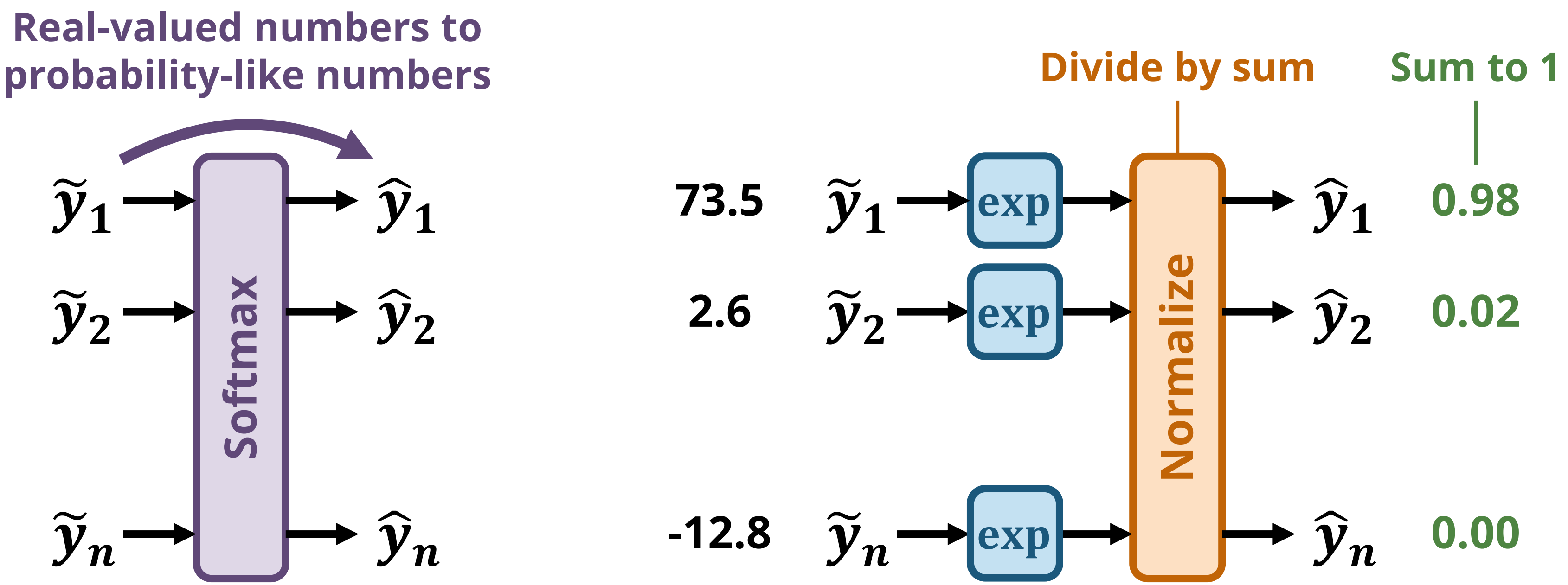
Loss function

Softmax

$$\hat{y}_i = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n} e^{\tilde{y}_j}}$$

Softmax

- **Intuition**: Map several numbers to $[0, 1]$ while **keeping their relative magnitude**
  - Softmax is like the **multivariate version of sigmoid**



Cross Entropy for Multiclass Classification

Binary Cross Entropy

Only one of them will be one!

$$L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Cross Entropy

Only one of them will be one!

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -y_1 \log \hat{y}_1 - y_2 \log \hat{y}_2 - \cdots - y_i \log \hat{y}_n$$

$$= -\sum_{i}^{n} y_i \log \hat{y}_i$$

Log likelihood

Optimization

Training a Neural Network

Build a neural network
(which defines a set of functions)

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x})$$

Define the objective
(i.e., what is good for a function)

$$L(\boldsymbol{\theta})$$

Find the optimal parameters
(which leads to the best function)

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

Training a Neural Network

Build a neural network
(which defines a set of functions)

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x})$$

↓

Define the objective
(i.e., what is good for a function)

$$L(\boldsymbol{\theta})$$

↓

Find the optimal parameters
(which leads to the best function)

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

Optimizing the Parameters of a Neural Network

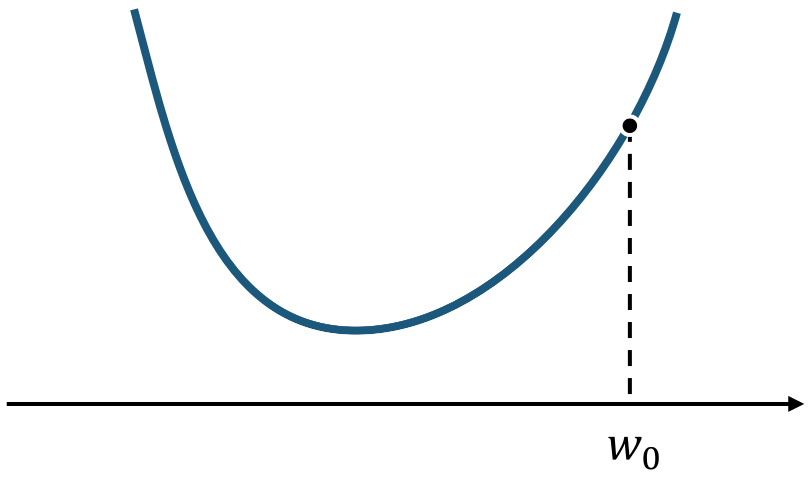
- Many, many ways…
- Most commonly through **gradient descent** in deep learning
- Alternatively, we can use search or genetic algorithm

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

Gradient Descent

- **Intuition**: Gradient can suggest a good direction to tune the parameters

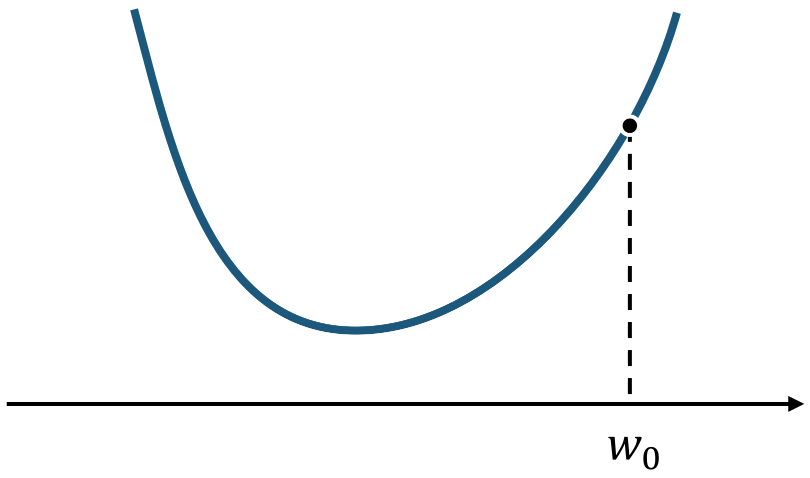
Derivative for a vector, matrix or tensor



Gradient Descent: Pseudocode

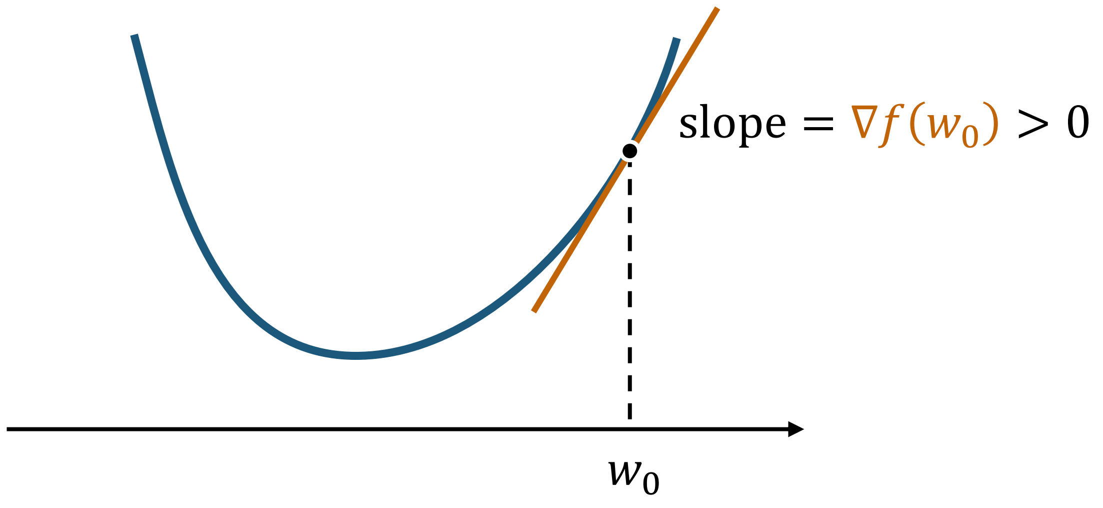
- Pick an **initial weight vector** w_0 and **learning rate** η
- Repeat until convergence: $w_{t+1} = w_t - \eta \nabla f(w_t)$

Gradient of function f with respect to weight w



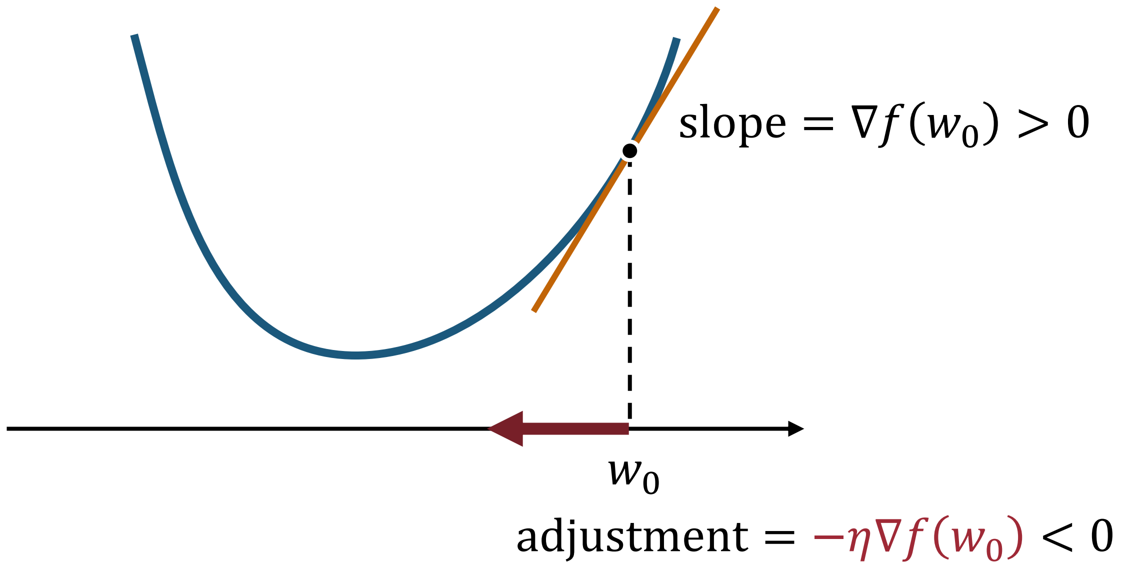
Gradient Descent: Pseudocode

- Pick an initial weight vector w_0 and learning rate η
- Repeat until convergence: $w_{t+1} = w_t - \eta \nabla f(w_t)$



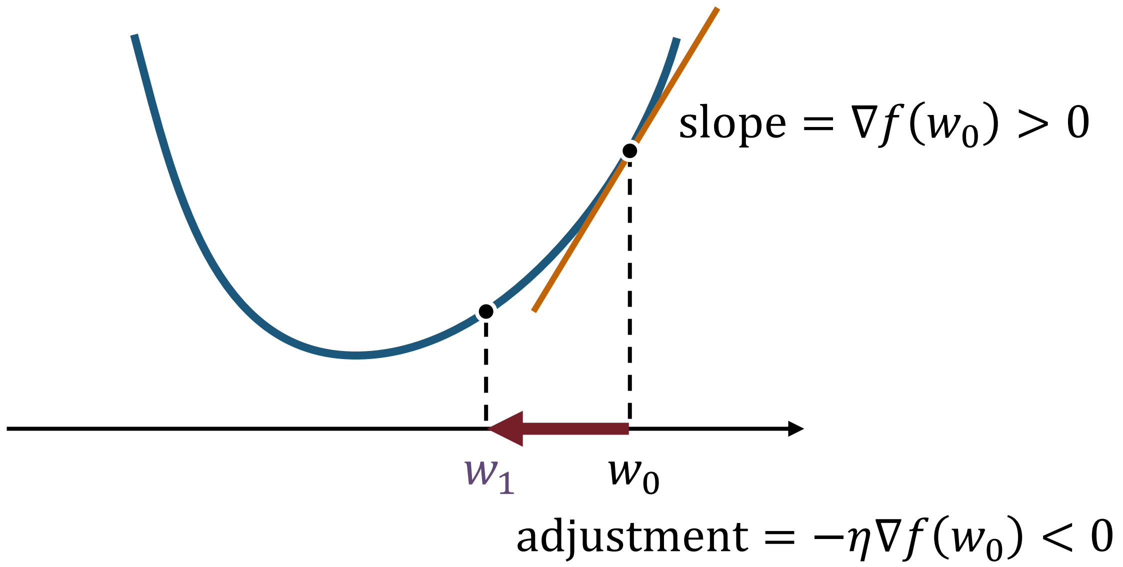
Gradient Descent: Pseudocode

- Pick an initial weight vector w_0 and learning rate η
- Repeat until convergence: $w_{t+1} = w_t - \eta \nabla f(w_t)$



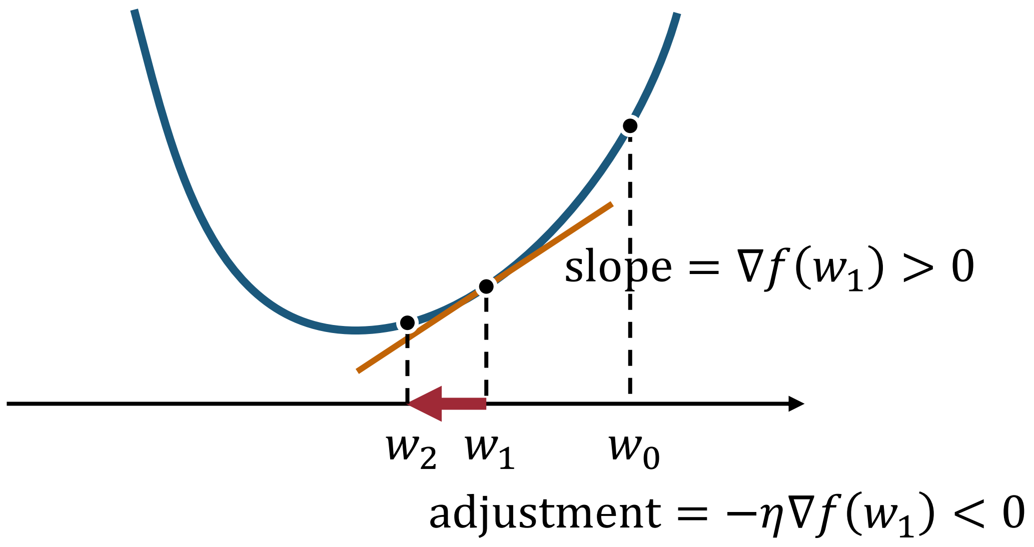
Gradient Descent: Pseudocode

- Pick an initial weight vector w_0 and learning rate η
- Repeat until convergence: $w_{t+1} = w_t - \eta \nabla f(w_t)$



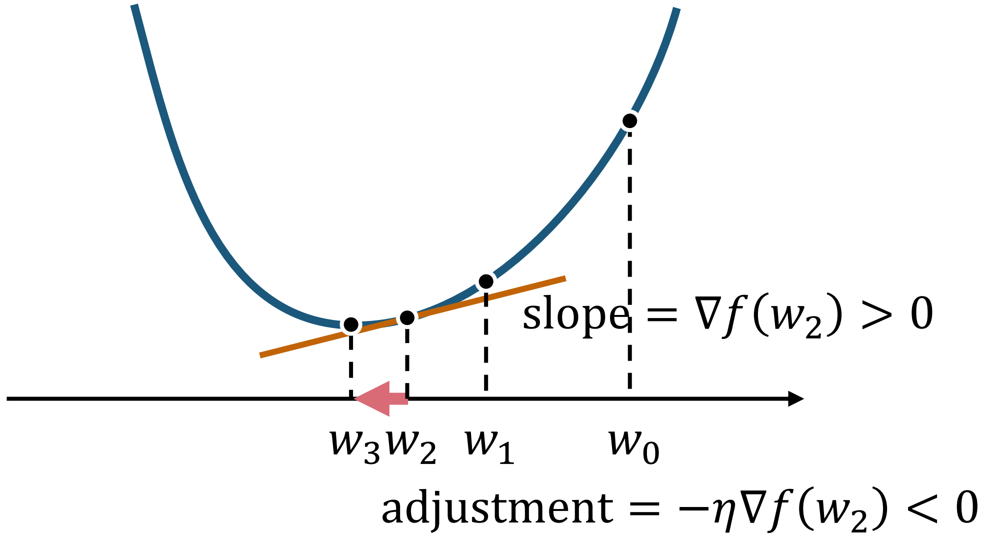
Gradient Descent: Pseudocode

- Pick an initial weight vector w_0 and learning rate η
- Repeat until convergence: $w_{t+1} = w_t - \eta \nabla f(w_t)$



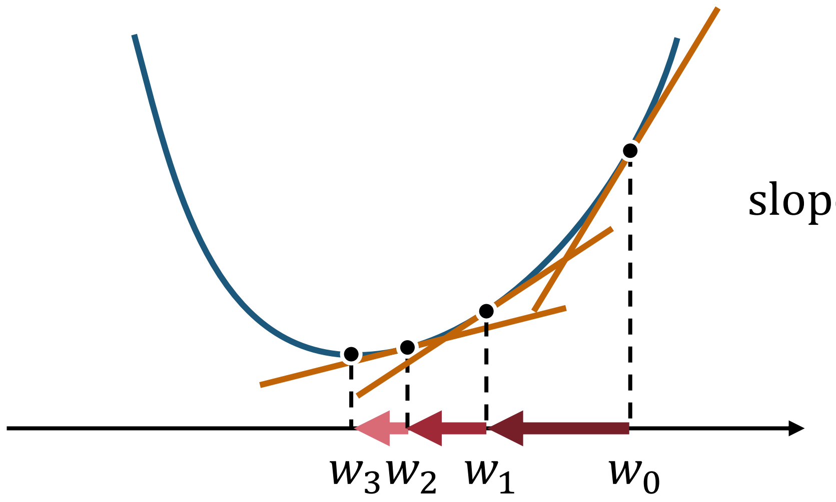
Gradient Descent: Pseudocode

- Pick an initial weight vector w_0 and learning rate η
- Repeat until convergence: $w_{t+1} = w_t - \eta \nabla f(w_t)$



Gradient Descent: Pseudocode

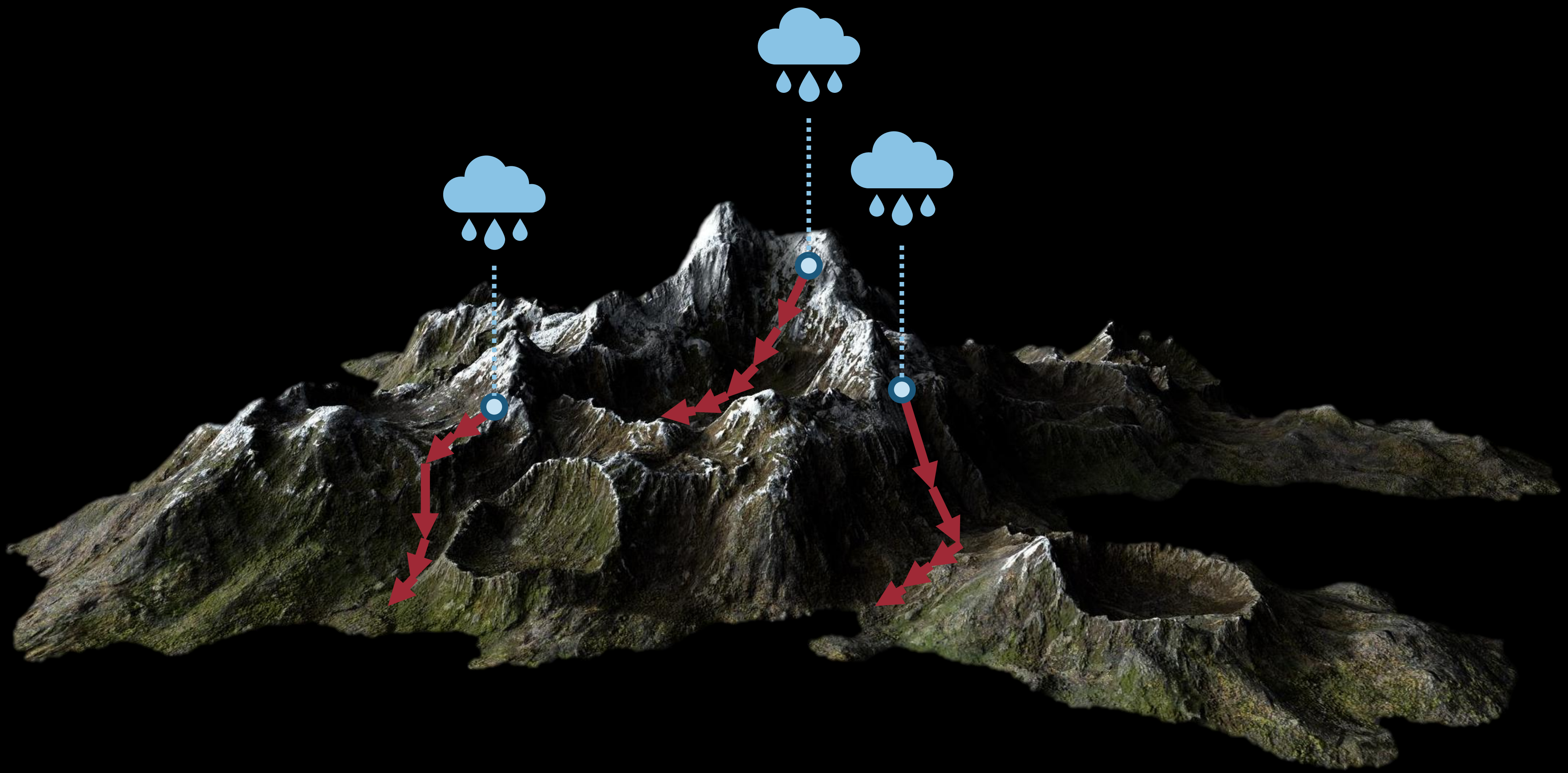
- Pick an initial weight vector w_0 and learning rate η
- Repeat until convergence: $w_{t+1} = w_t - \eta \nabla f(w_t)$



slope $= \nabla f(w_t) > 0$

adjustment $= -\eta \nabla f(w_t) < 0$

Gradient Descent: 3D Case



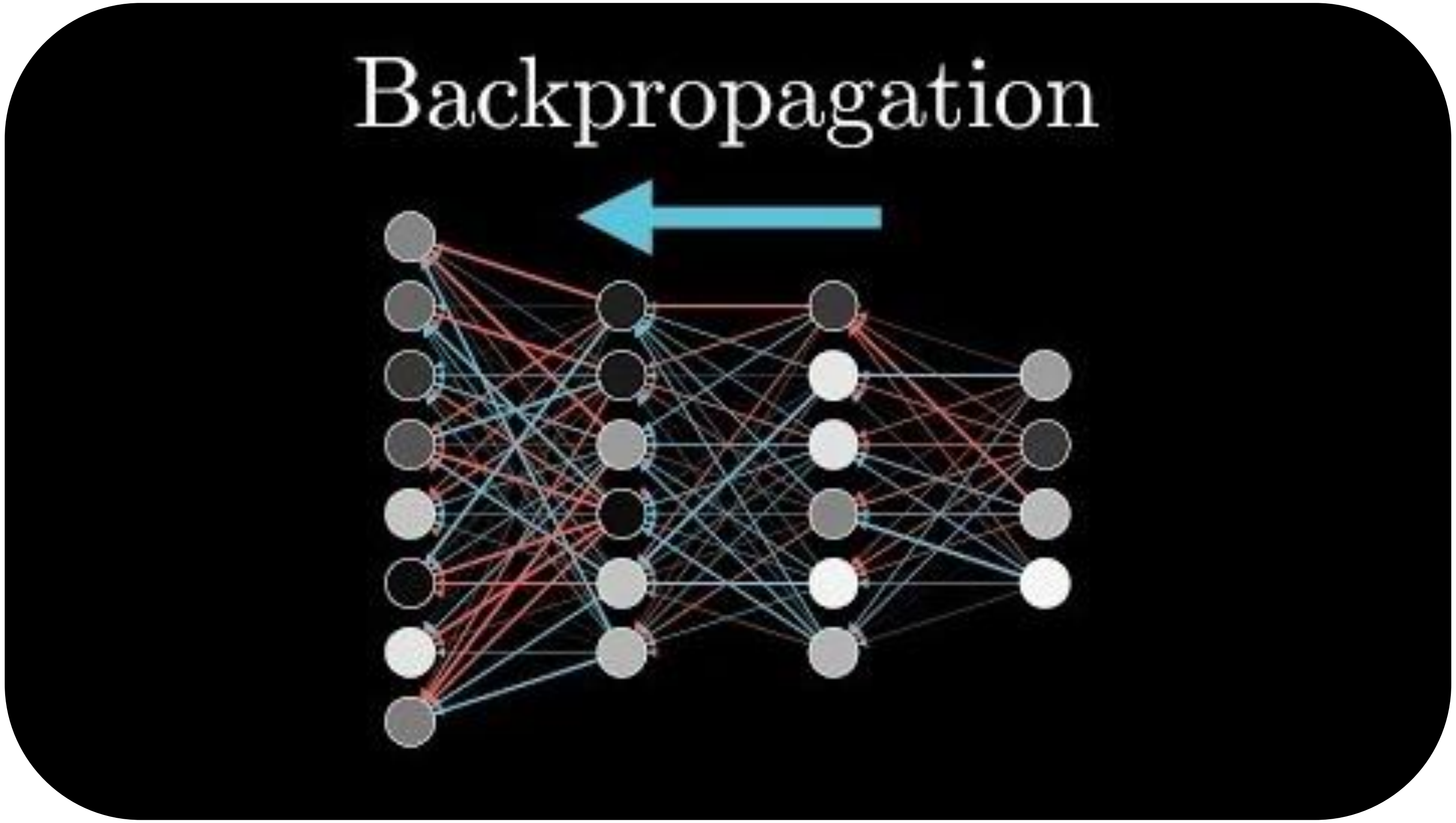
substance3d.adobe.com/community-assets/assets/9fe77d4279e12fe2fbd90b8f7ca18146a565f7ee

Backpropagation: Efficiently Computing the Gradients

- An efficient way of **computing gradients** using chain rule
- The reason why we want **everything to be differentiable** in deep learning

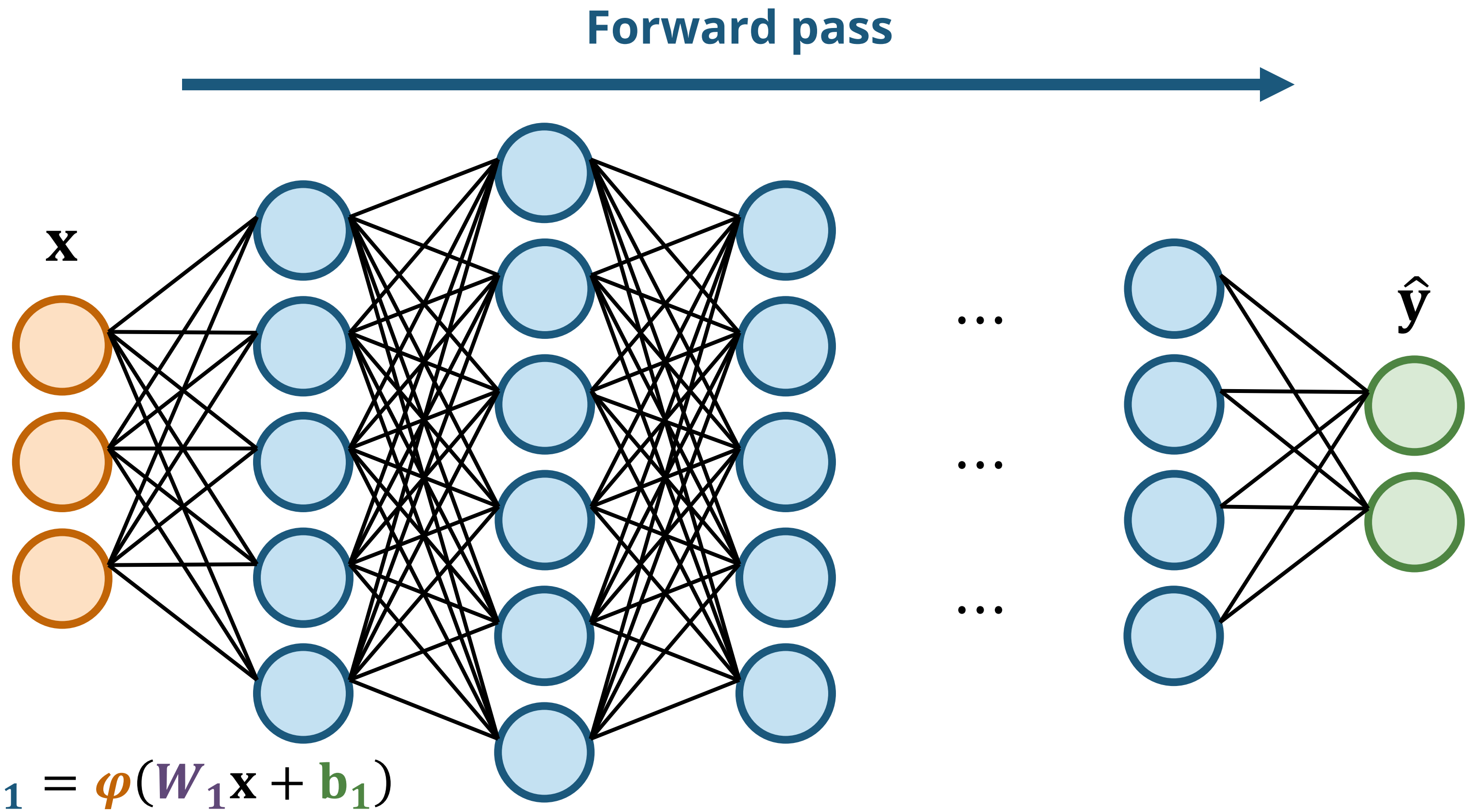
$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

Backpropagation: Efficiently Computing the Gradients



youtu.be/Ilg3gGewQ5U?t=196

Forward Pass & Backward Pass



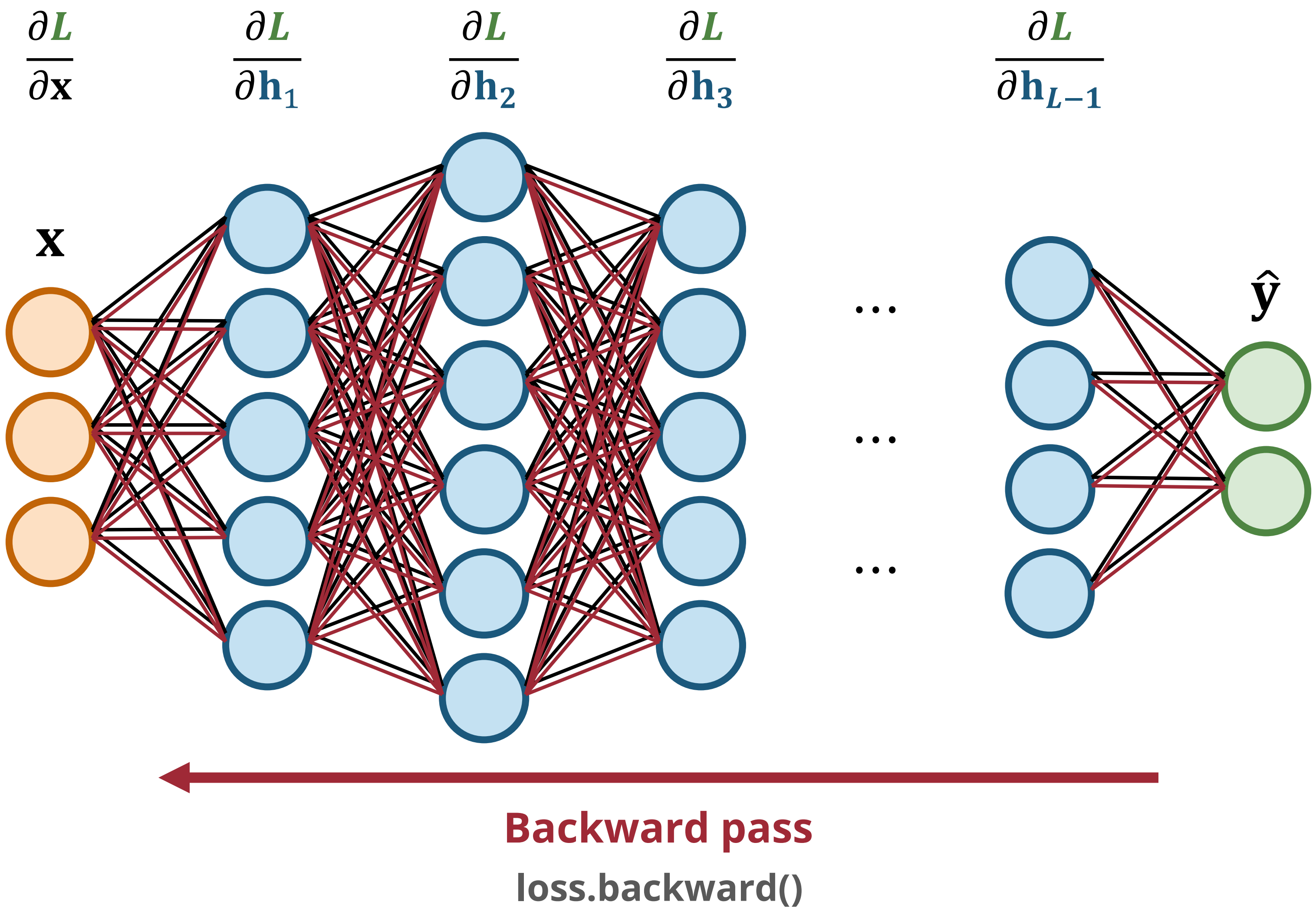
$$\mathbf{h}_1 = \varphi(W_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \varphi(W_2\mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \varphi(W_2\mathbf{h}_2 + \mathbf{b}_2)$$

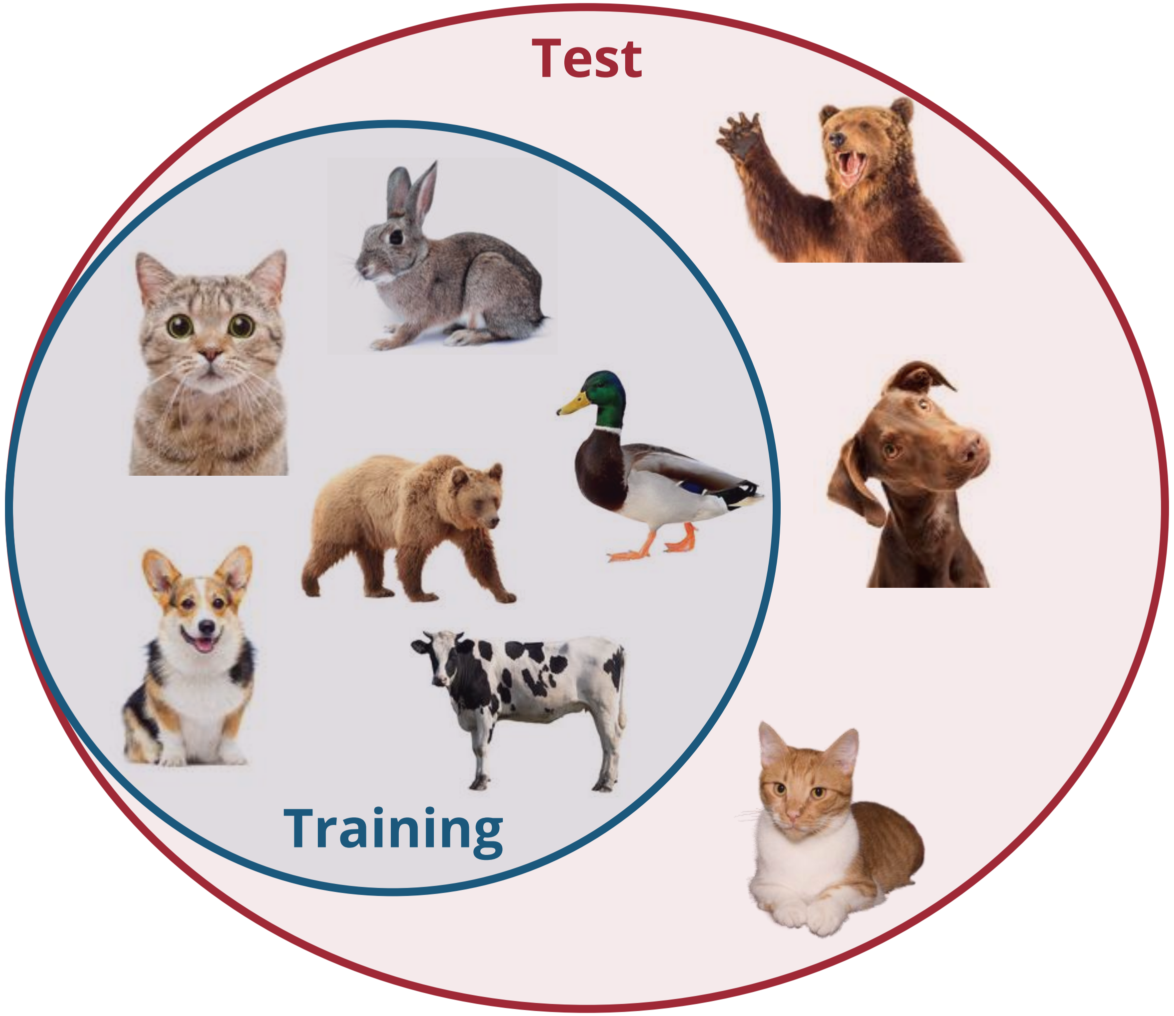
$$\hat{\mathbf{y}} = \varphi(W_L\mathbf{h}_{L-1} + \mathbf{b}_L)$$

Forward Pass & Backward Pass

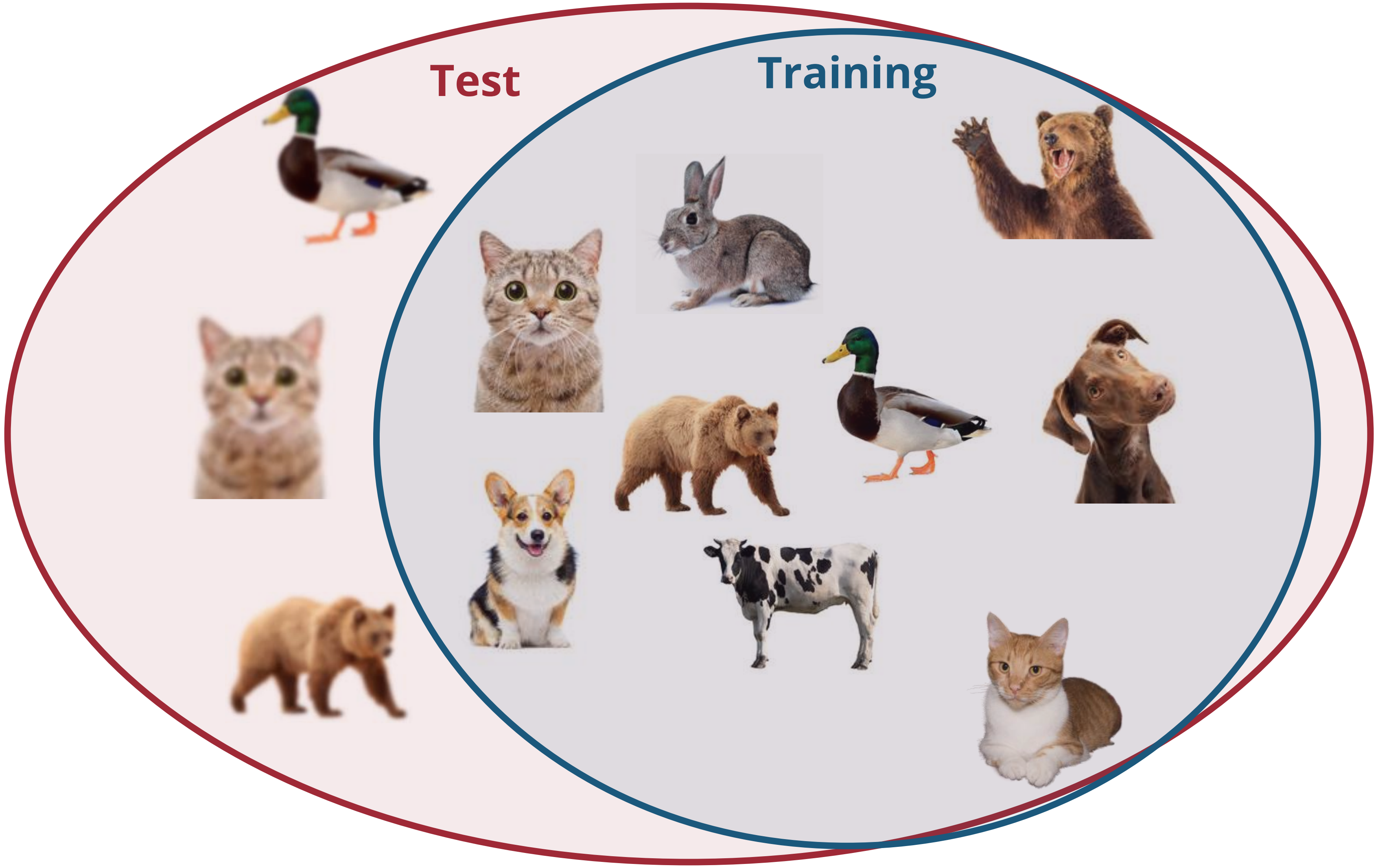


Training–Validation–Test

In-distribution vs Out-of-distribution



In-distribution vs Out-of-distribution



In-distribution vs Out-of-distribution

In-distribution vs Out-of-distribution

- **Key**: Make the training distribution closer to the target distribution
- First, we need to **define our target distribution**
- Then, we can try to
  - Collect a **diverse** dataset covering that covers different parts of the target distribution
  - Apply **data augmentation** to fill the gaps in the distribution

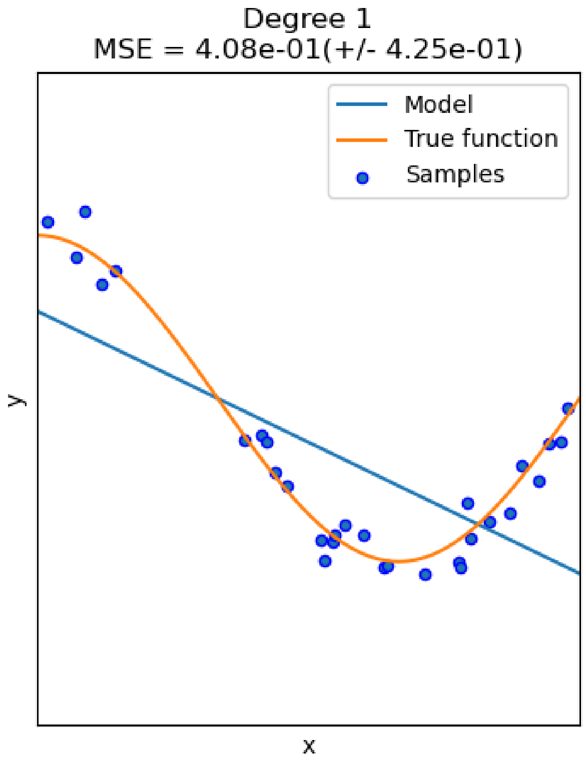
In-distribution vs Out-of-distribution

- What do we really want?
  - Good performance on the **training samples** **We already have their answers**
  - Good performance on **unseen samples in the target distribution** **Yep, we can do this!**
  - Good performance on **out-of-distribution samples** **Hopefully, but not guaranteed**

How to achieve good performance on
unseen samples in the target distribution

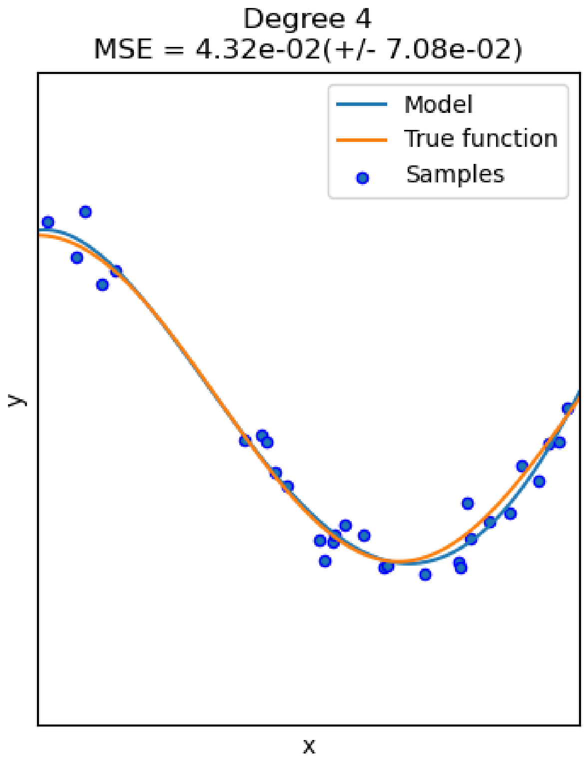
Overfitting & Underfitting

Underfitting

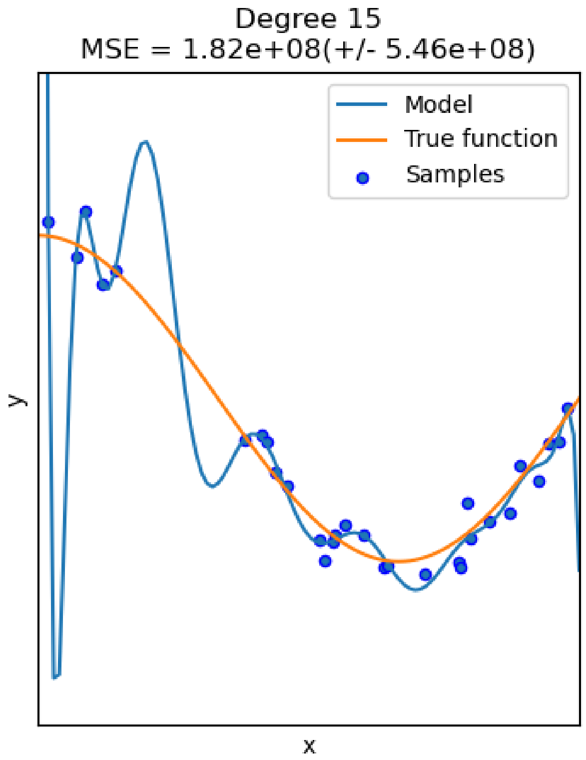


Model too inexpressive

Good fit!



Overfitting

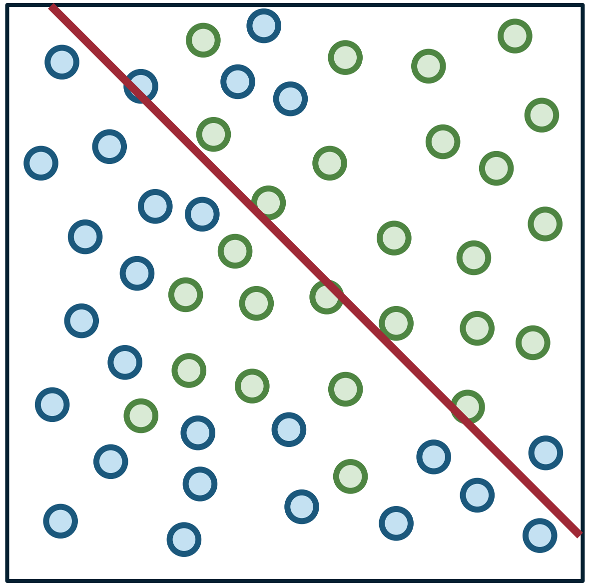


Model too expressive

scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

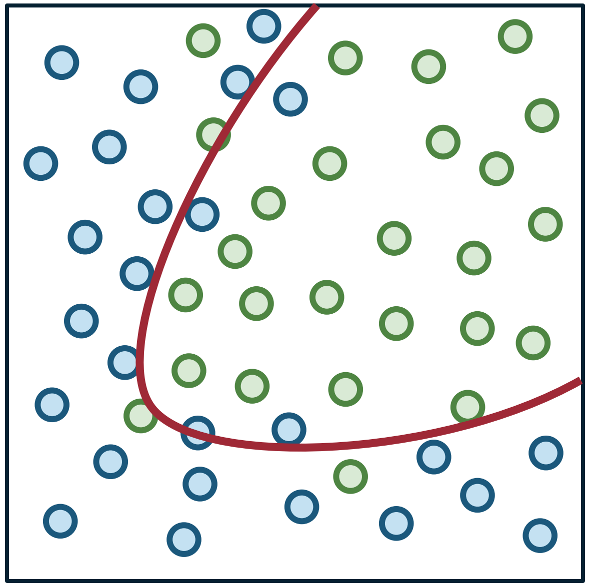
Overfitting & Underfitting

Underfitting

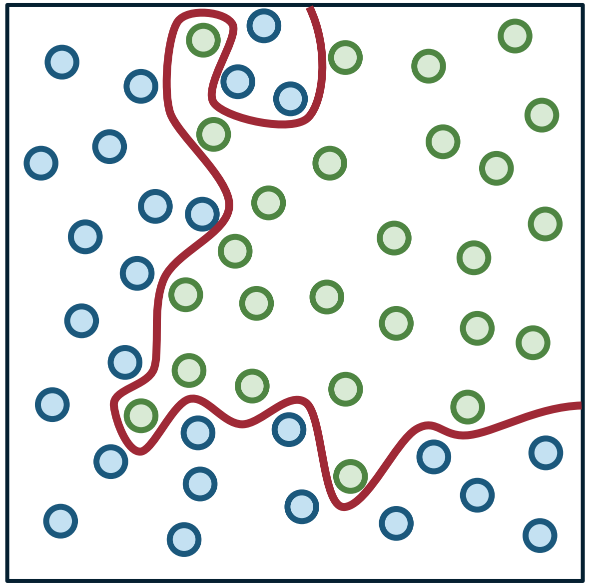


Model too inexpressive

Good fit!



Overfitting



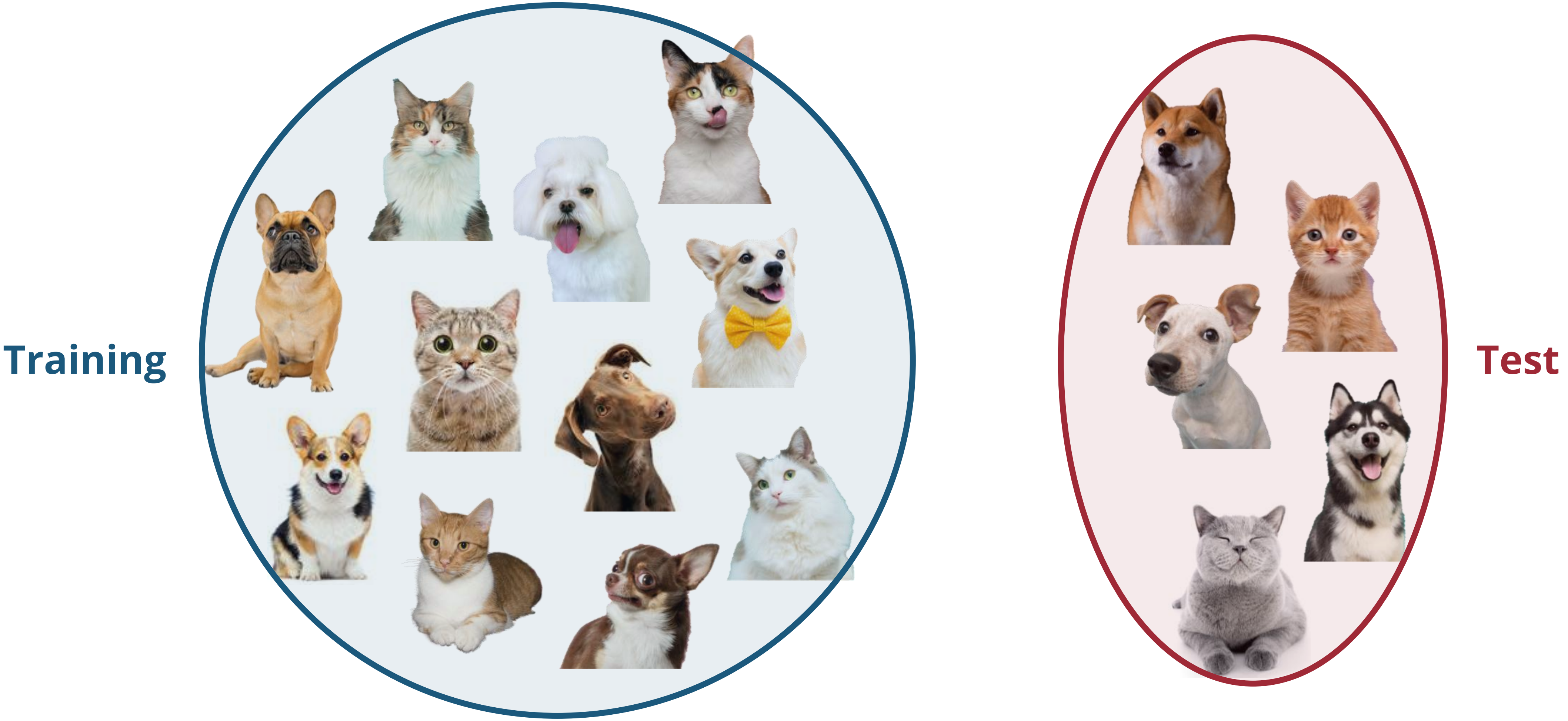
Model too expressive

Train–Test Split

- **Goal**: Good performance on **unseen samples in the target distribution**

Train–Test Split

- **Goal**: Good performance on **unseen samples in the target distribution**

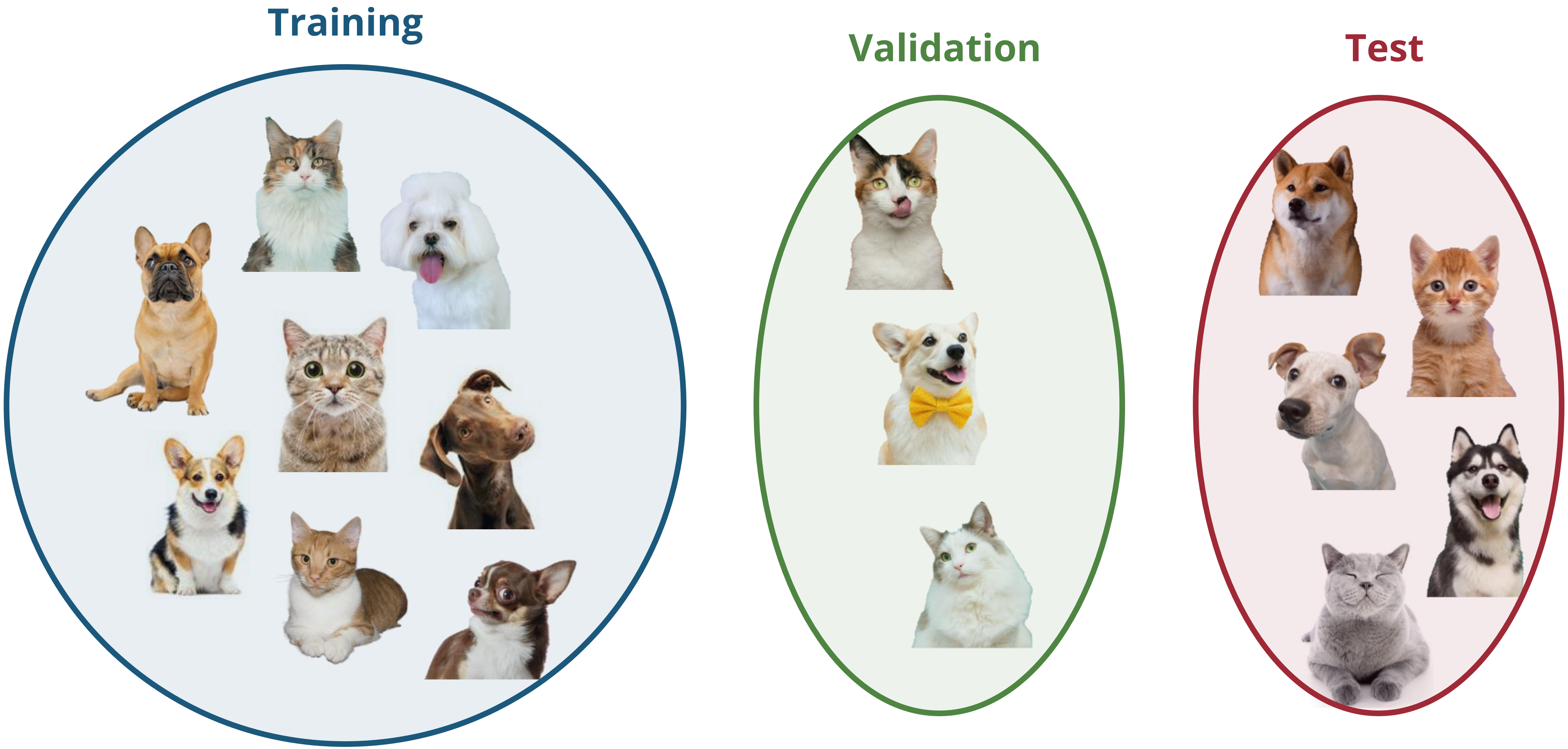


Test Set is an Estimation of the Test Distribution

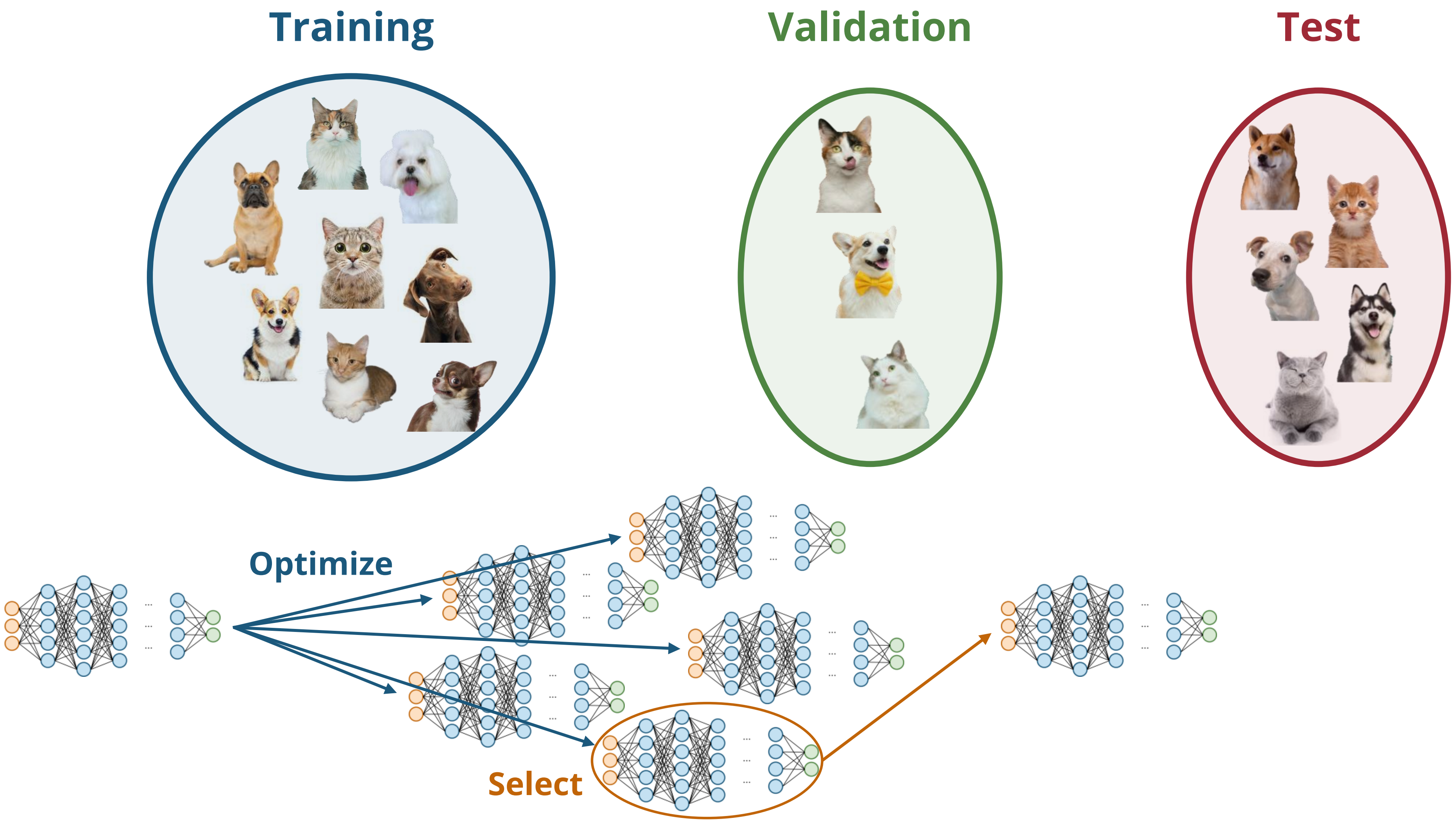
- We create a test set because we want to **estimate the performance when the model is applied to an interested distribution**

Train–Validation–Test Split

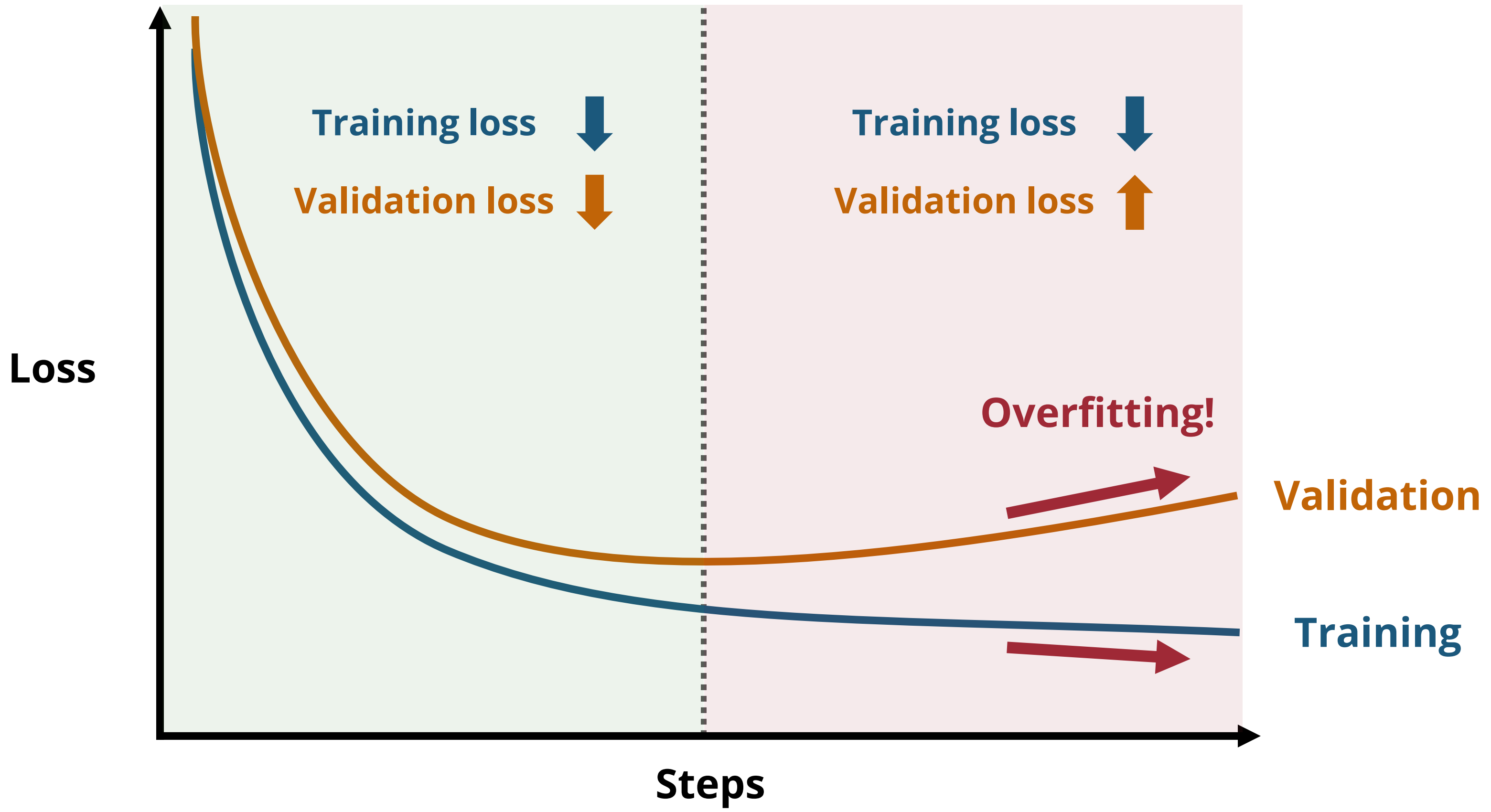
Train–Validation–Test Split



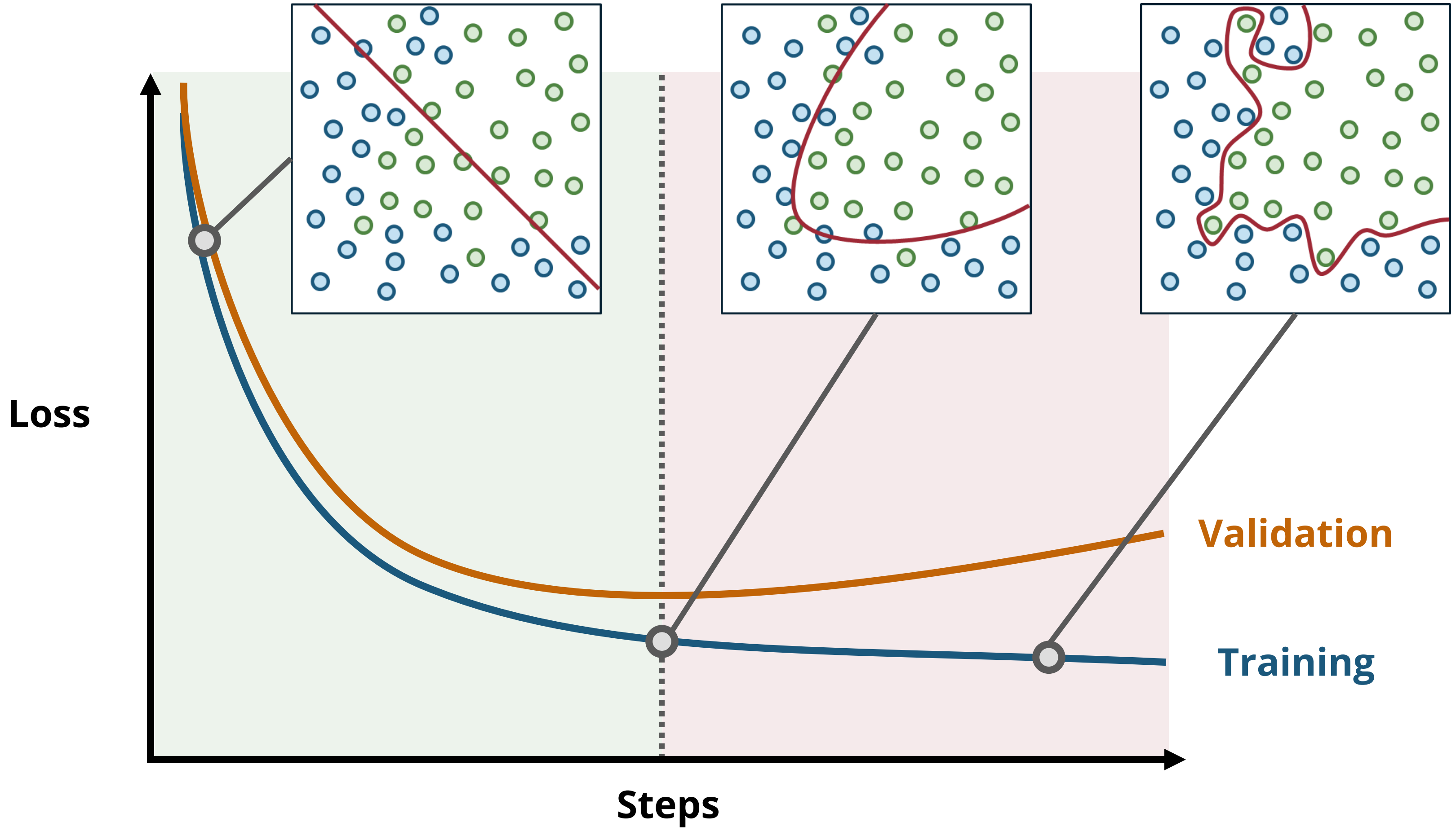
Training–Validation–Test Pipeline



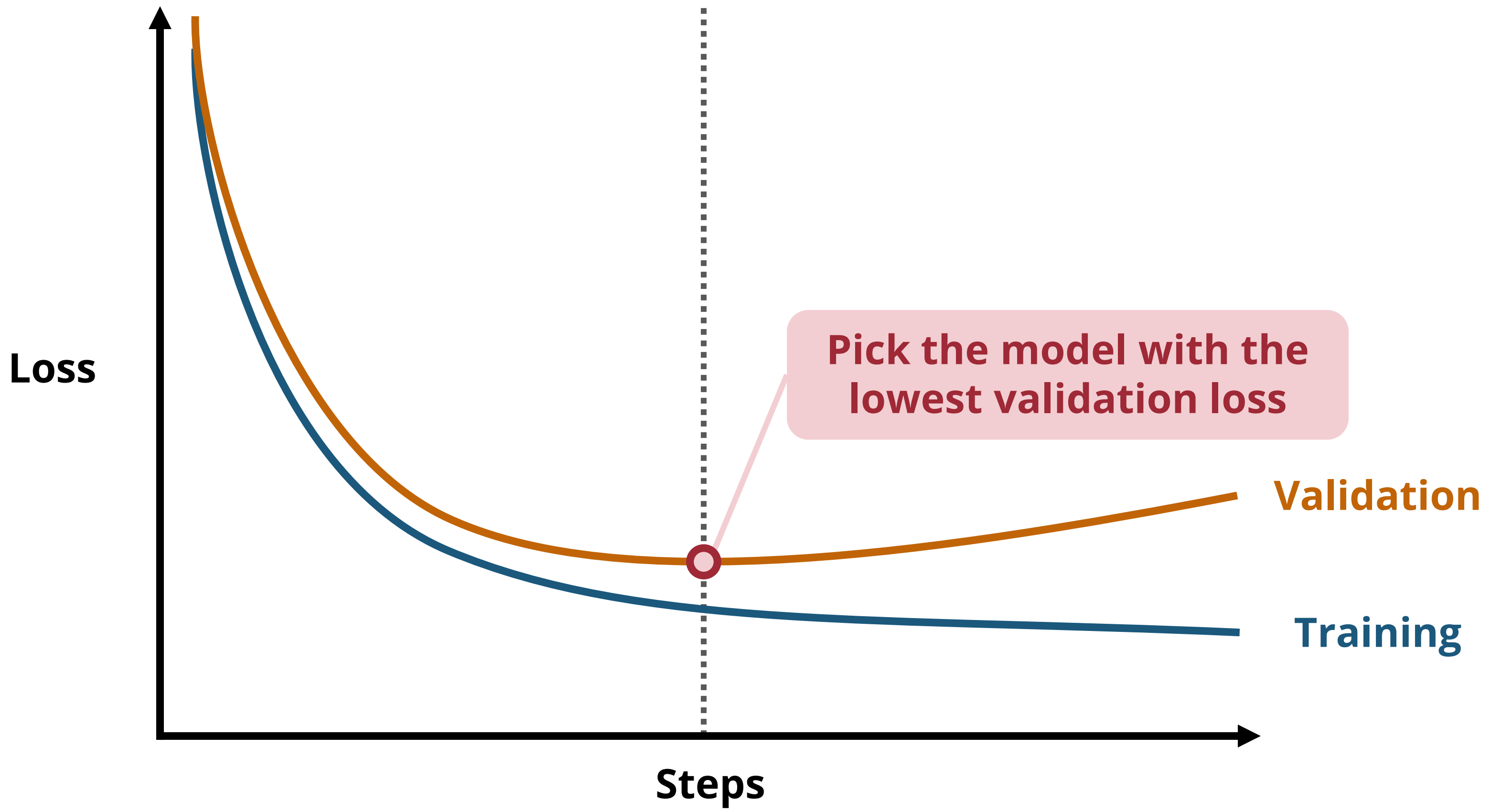
Training vs Validation Losses



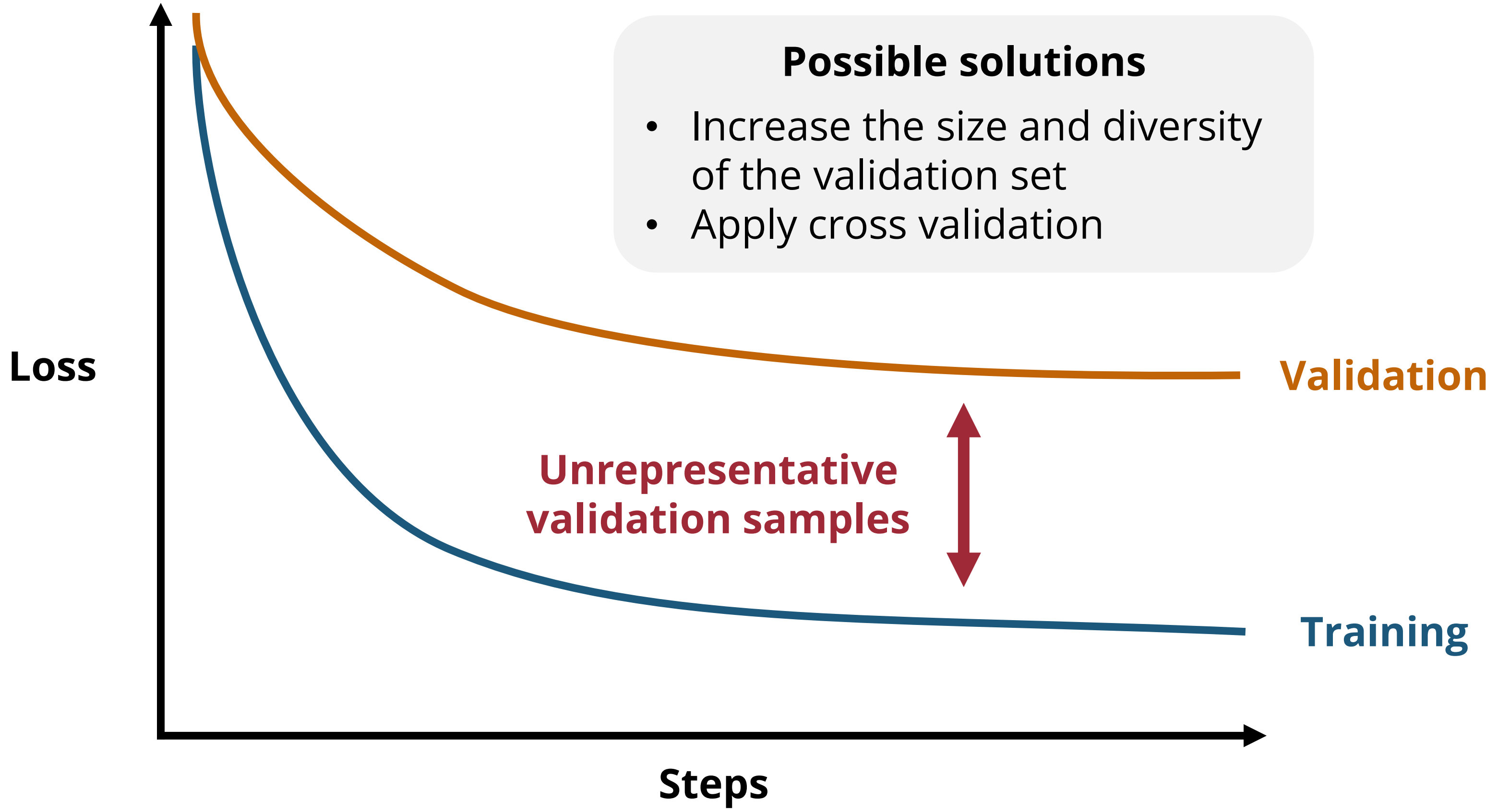
Training vs Validation Losses



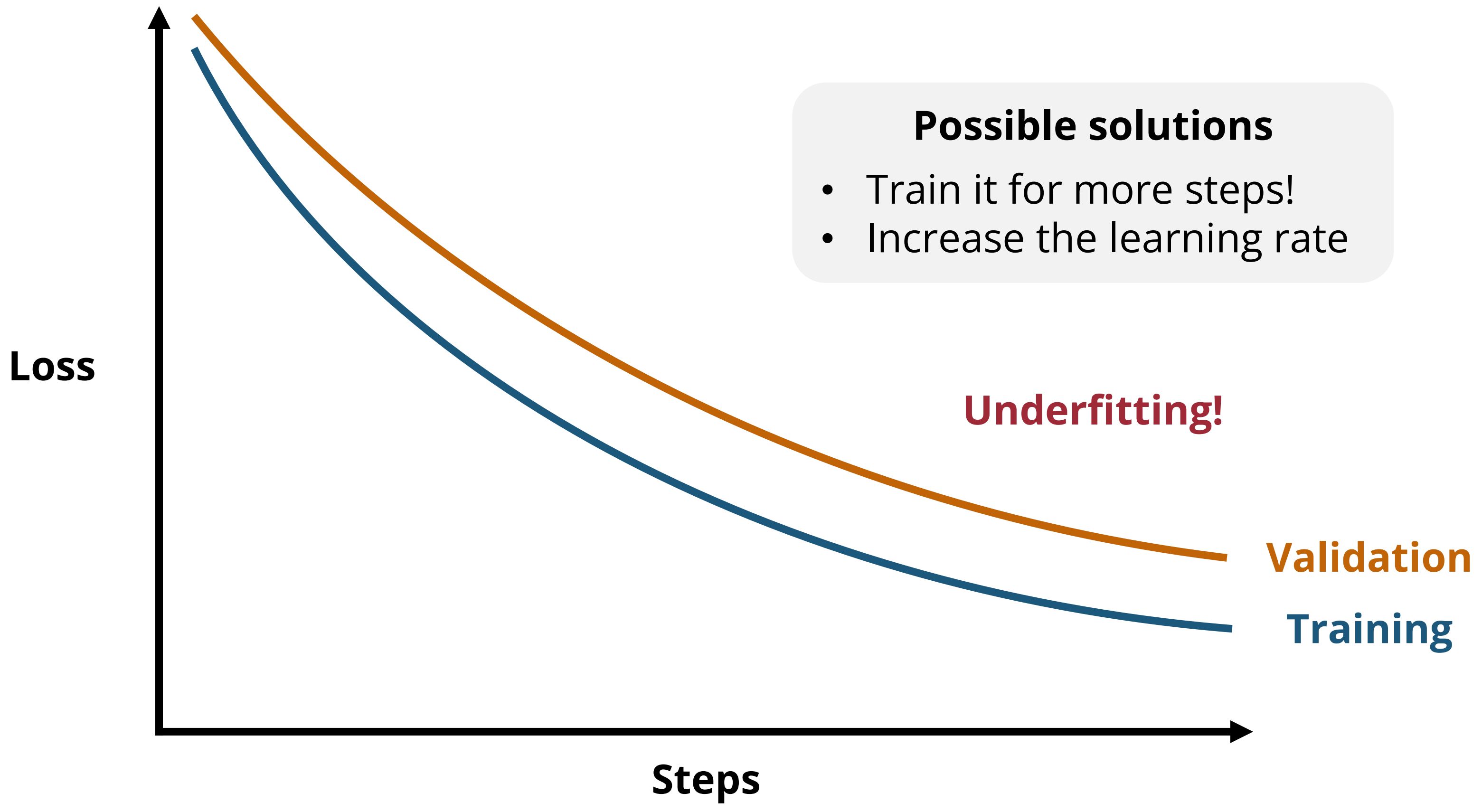
Training vs Validation Losses



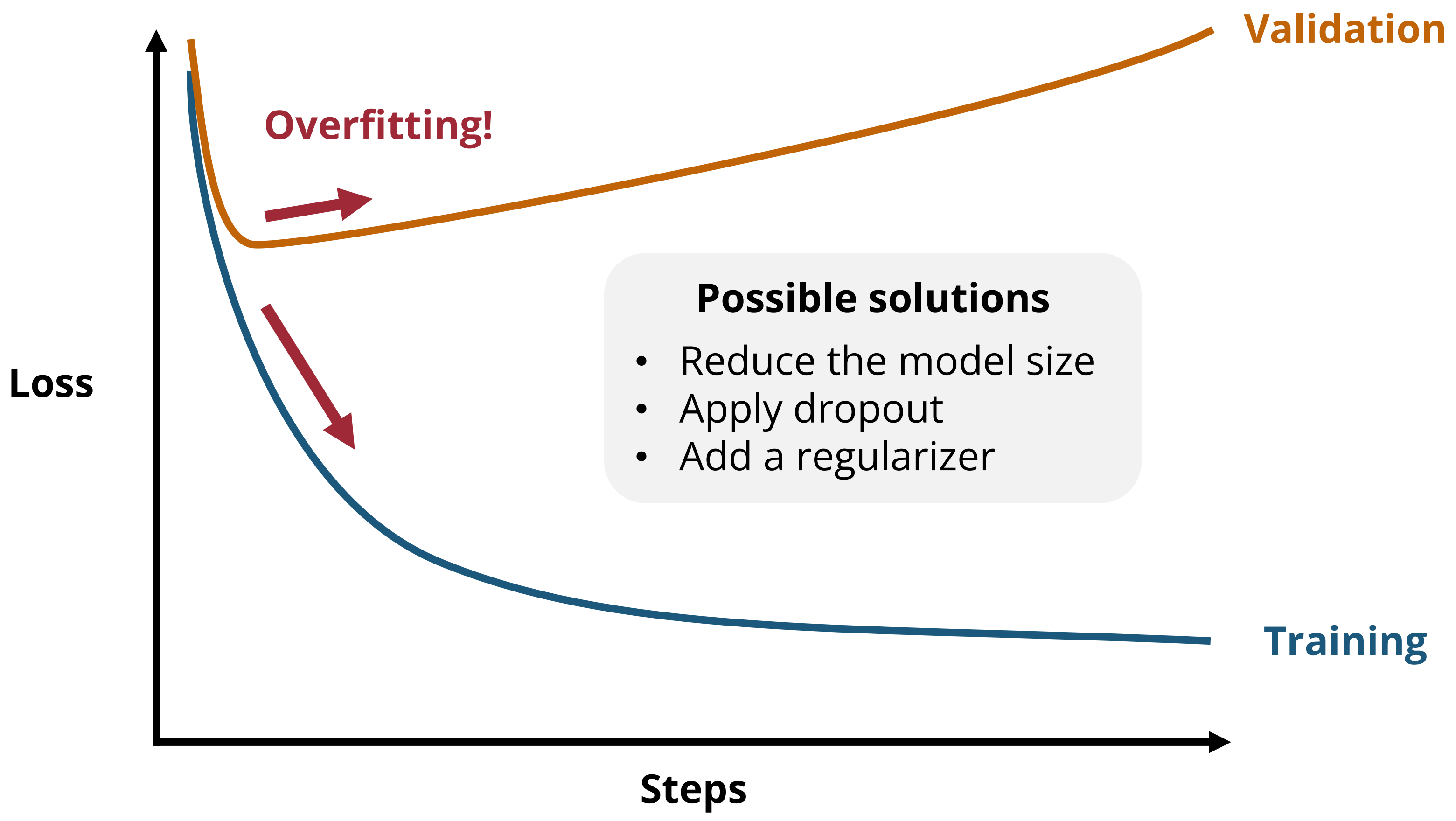
Training vs Validation Losses



Training vs Validation Losses



Training vs Validation Losses



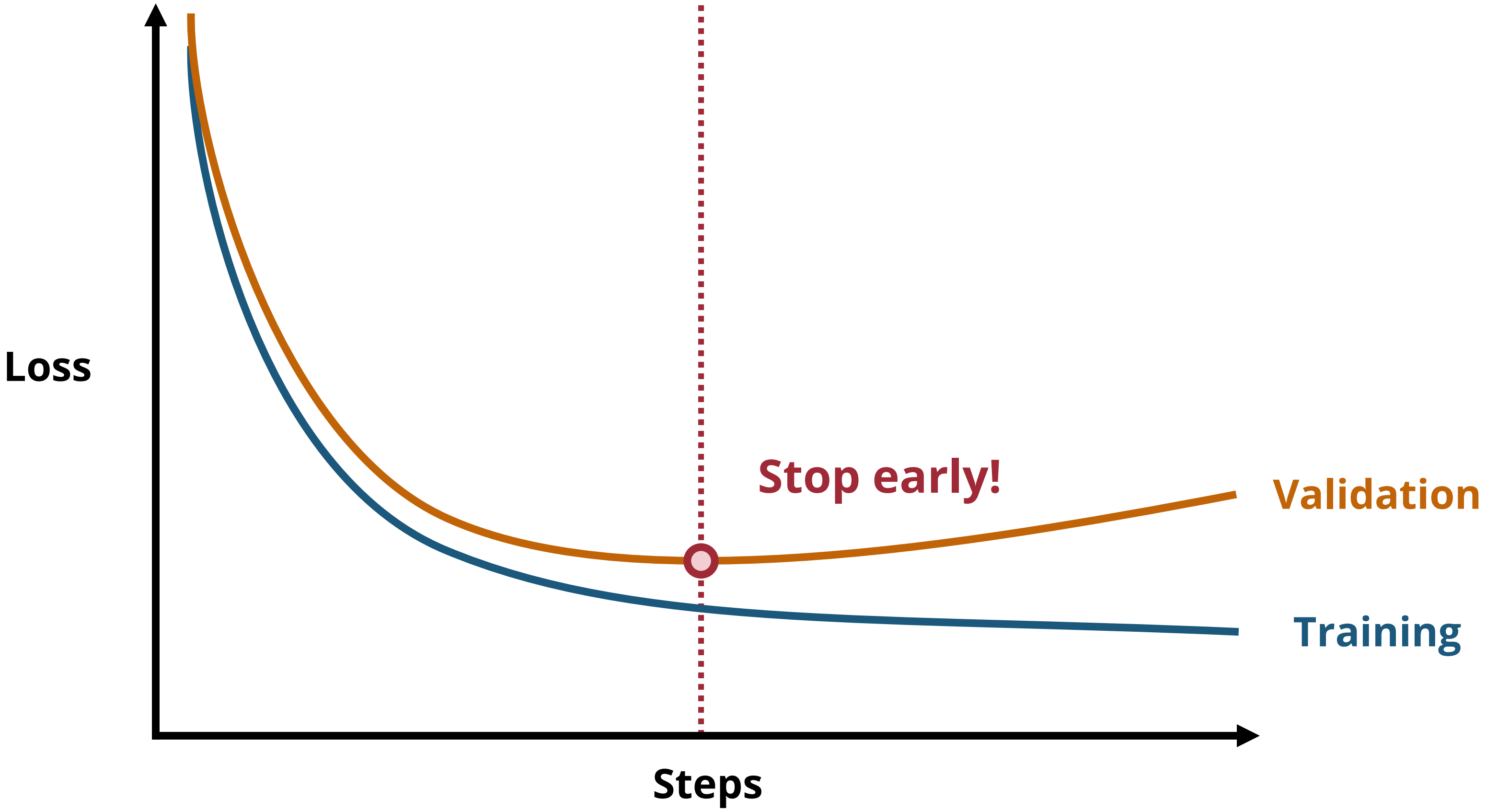
Train–Validation–Test Split

- **Keys**
  - **Never train or select your model on test samples!**
  - Don't over-select your model on the validation set

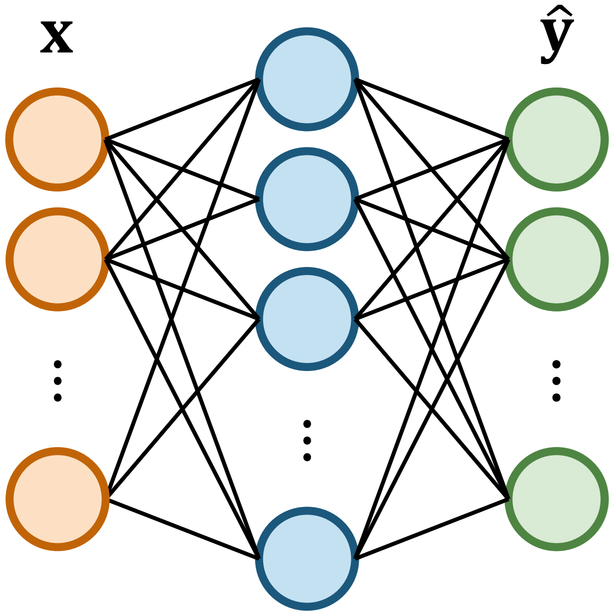
- What's the **best ratio**?
  - Most common: **8:1:1** or 9:0.5:0.5
  - For smaller dataset, you might even want 6:2:2

Overcoming Overfitting

Early Stopping

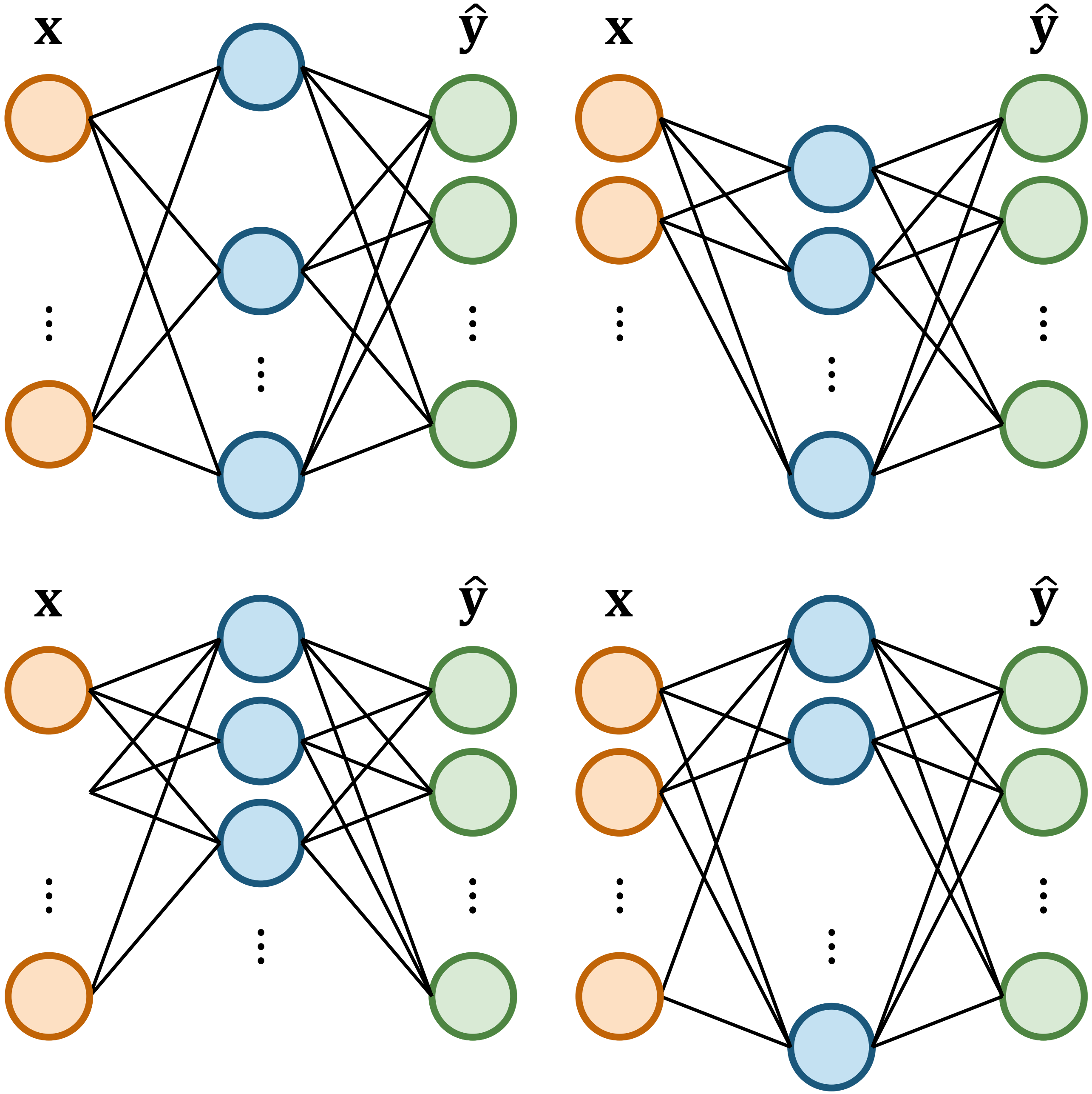


Dropout

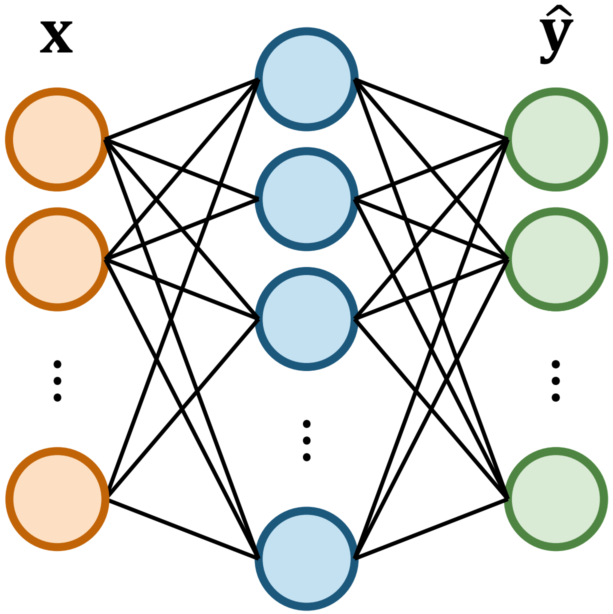


Each neuron may be removed
with probability p during training

Dropout rate

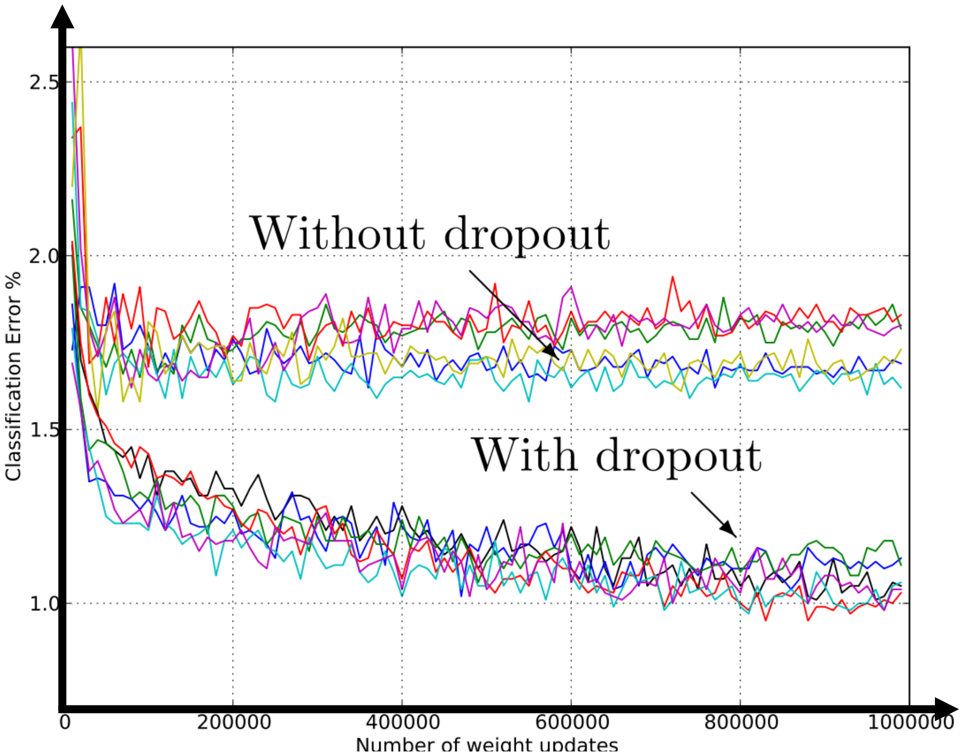


Dropout



Each neuron may be removed with probability p during training

Test error rate



Weight updates

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *JMLR*, 2014.\

Regularization Term

- A regularization term can help alleviate overfitting
  - **L1 regularization** (LASSO)

$$L' = L + \lambda(|w_1| + |w_2| + \cdots + |w_K|)$$

  - **L2 regularization** (ridge regression)

$$L' = L + \lambda\left(w_1^2 + w_2^2 + \cdots + w_K^2\right)$$

Both L1 and L2 regularizations encourage smaller weights

Recap

Training a Neural Network

Build a neural network
(which defines a set of functions)

↓

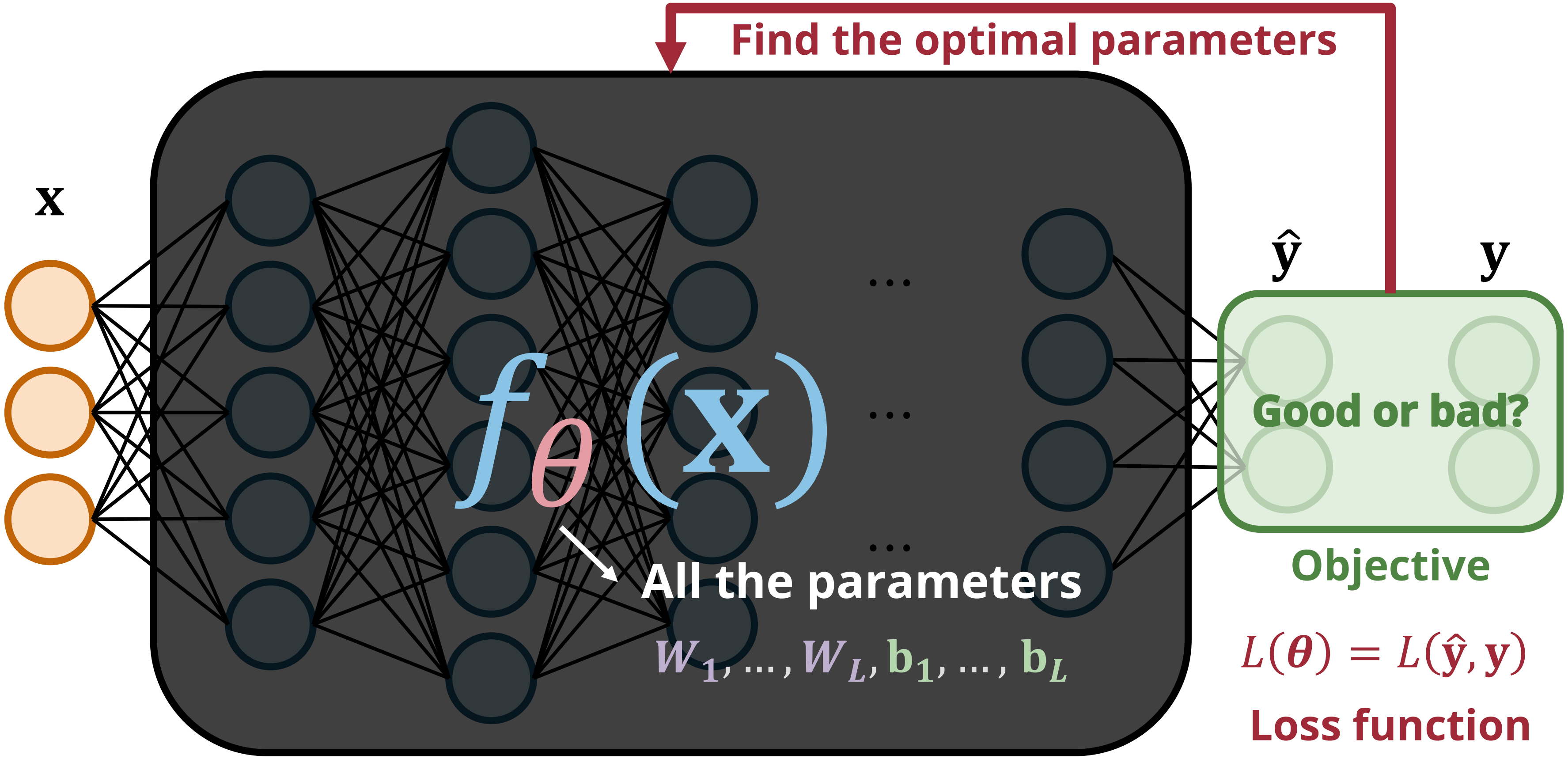
Define the objective
(i.e., what is good for a function)

↓

Find the optimal parameters
(which leads to the best function)

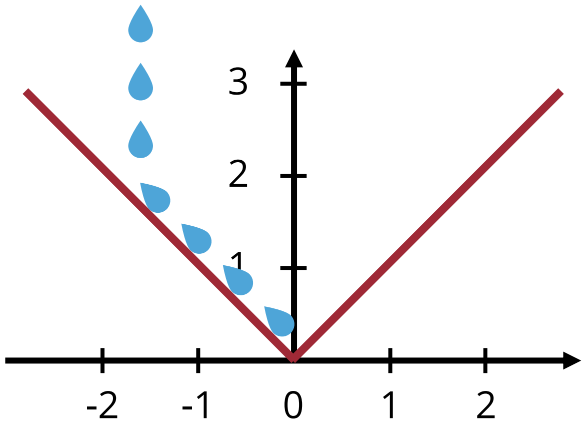
Neural Networks are Parameterized Functions

- A neural network represents **a set of functions**



L1 vs L2 Losses

L1 loss

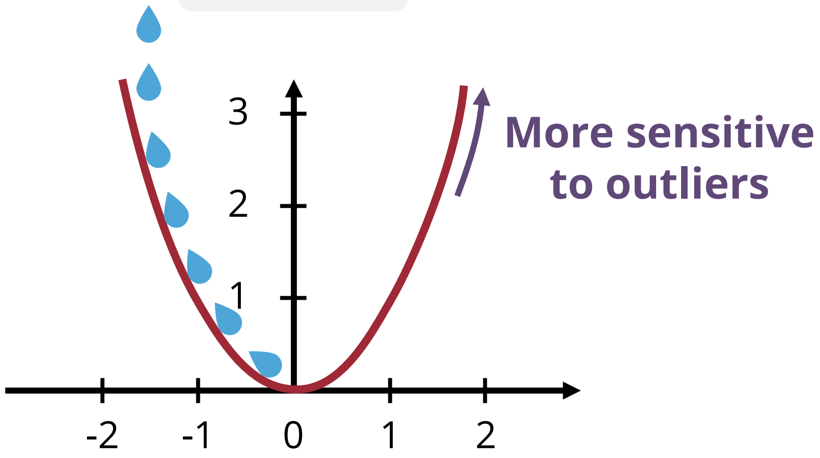


$$L(\hat{y}, y) = |\hat{y} - y|$$

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \mathbf{MAE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Mean Absolute Error (MAE)

L2 loss

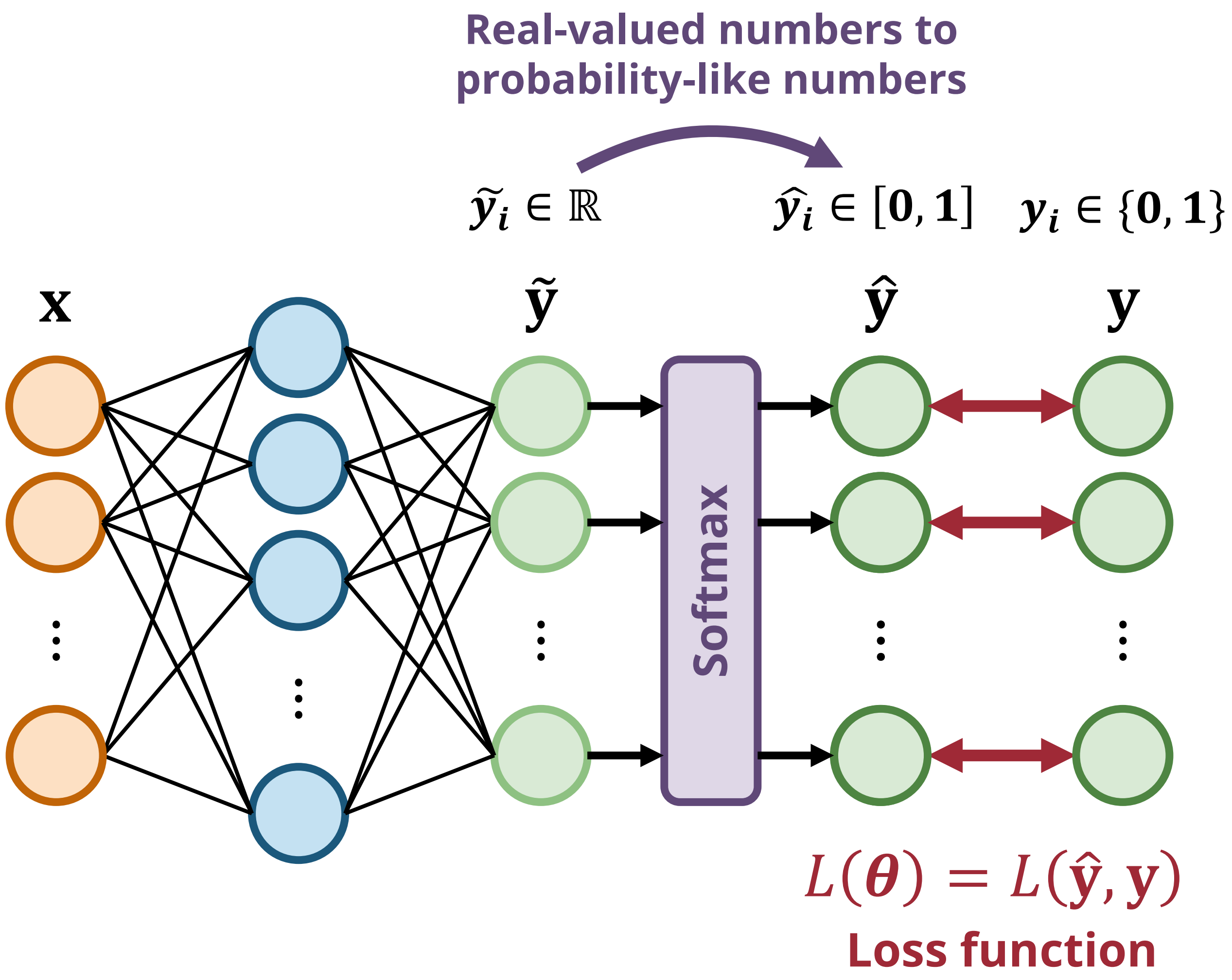


$$L(\hat{y}, y) = (\hat{y} - y)^2$$

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \mathbf{MSE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Mean Squared Error (MSE)

Cross Entropy for Multiclass Classification

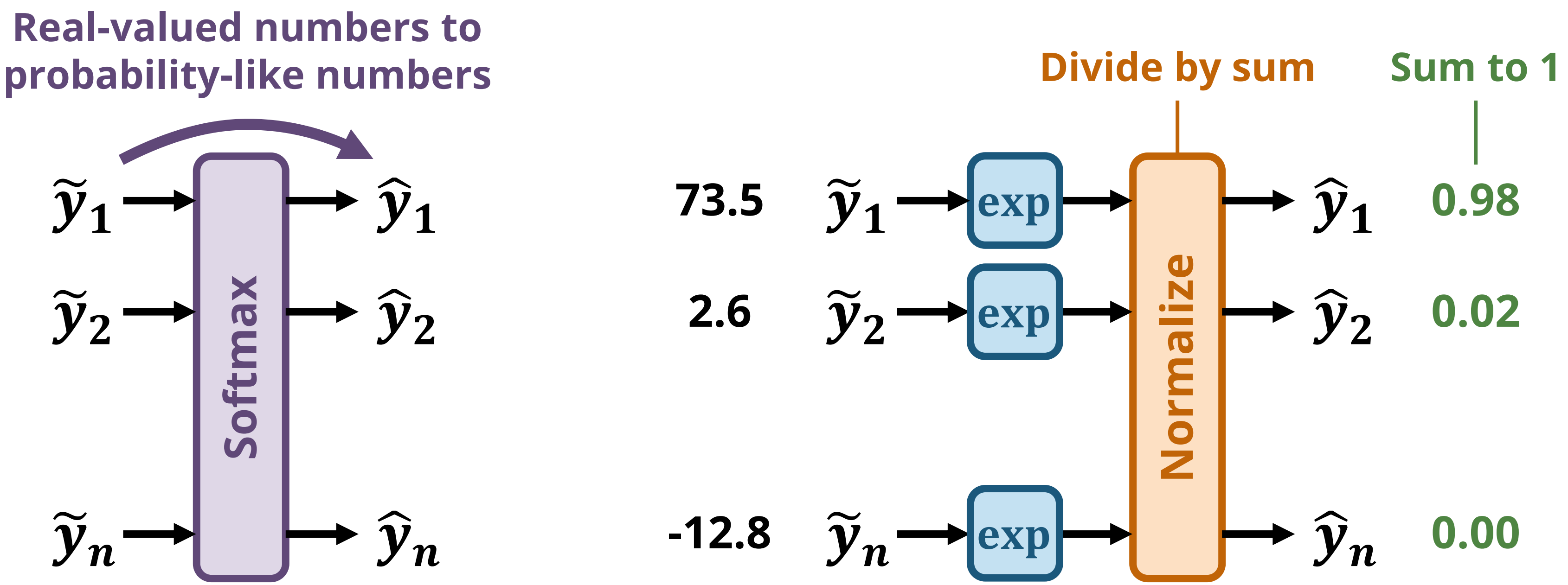


Softmax

$$\hat{y}_i = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n} e^{\tilde{y}_j}}$$

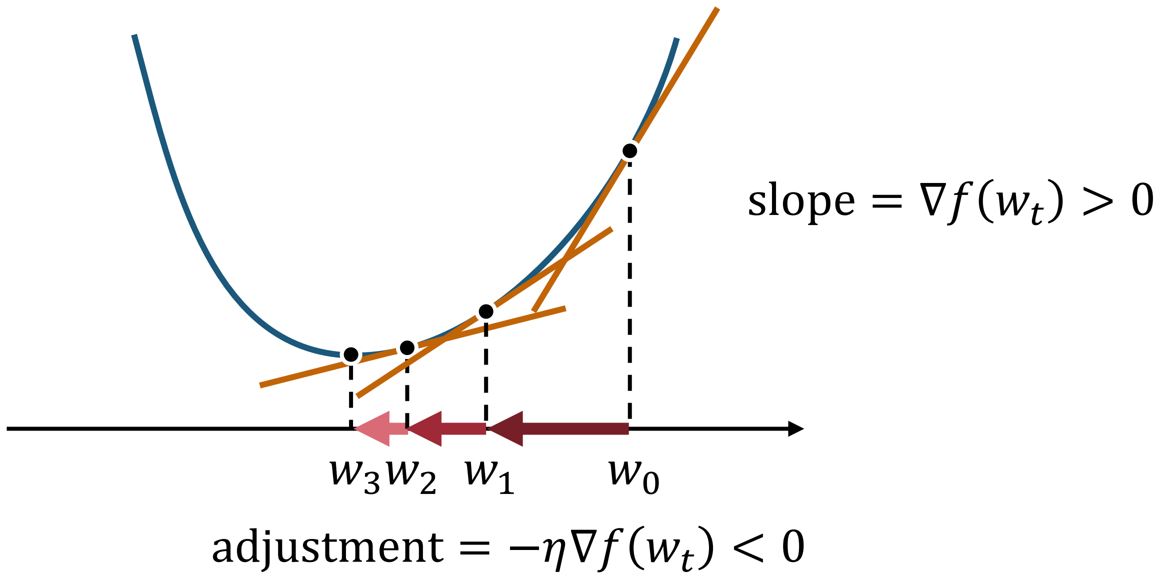
Softmax

- **Intuition**: Map several numbers to $[0, 1]$ while **keeping their relative magnitude**
  - Softmax is like the **multivariate version of sigmoid**

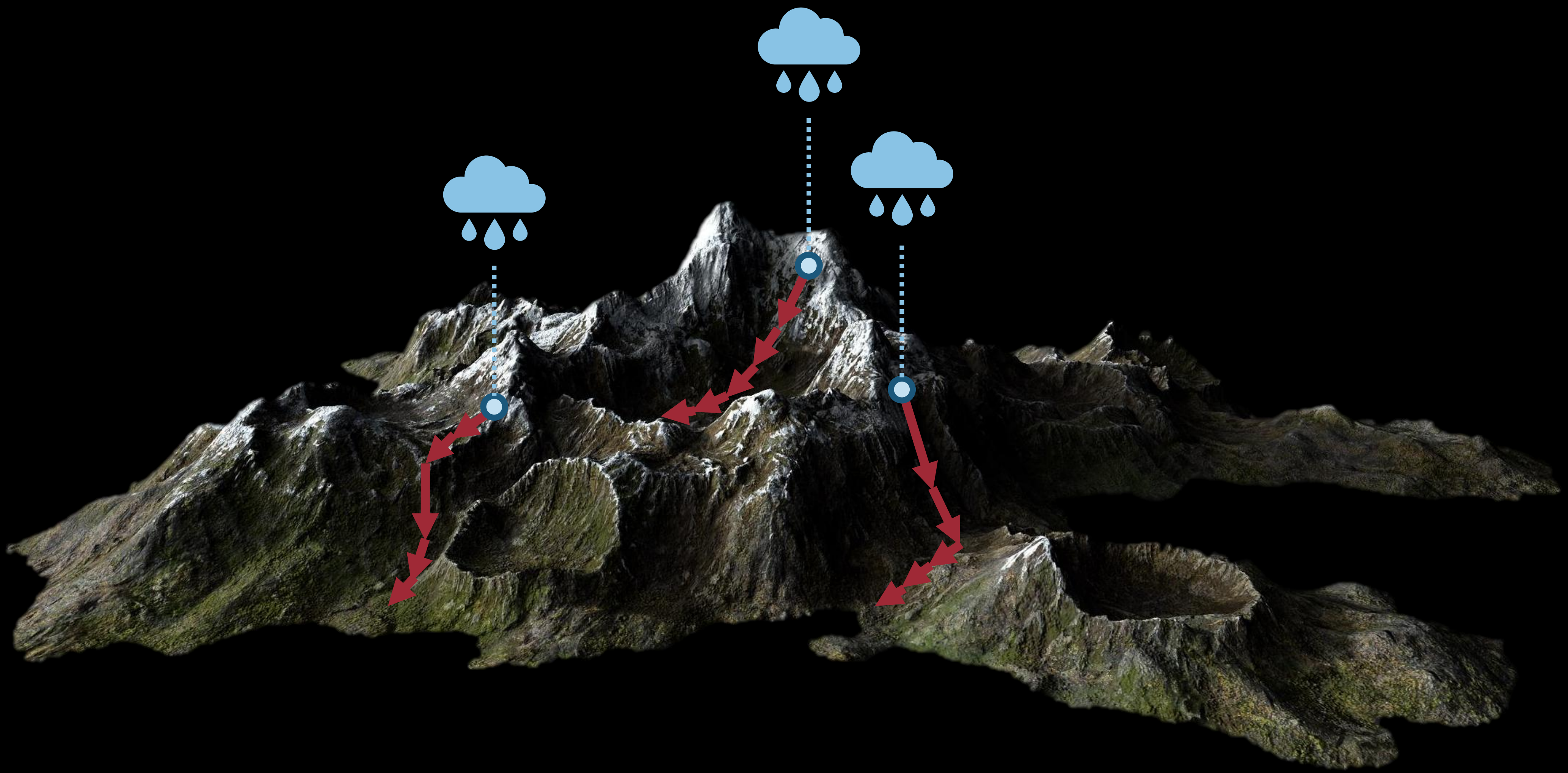


Gradient Descent: Pseudocode

- Pick an initial weight vector w_0 and learning rate η
- Repeat until convergence: $w_{t+1} = w_t - \eta \nabla f(w_t)$



Gradient Descent: 3D Case



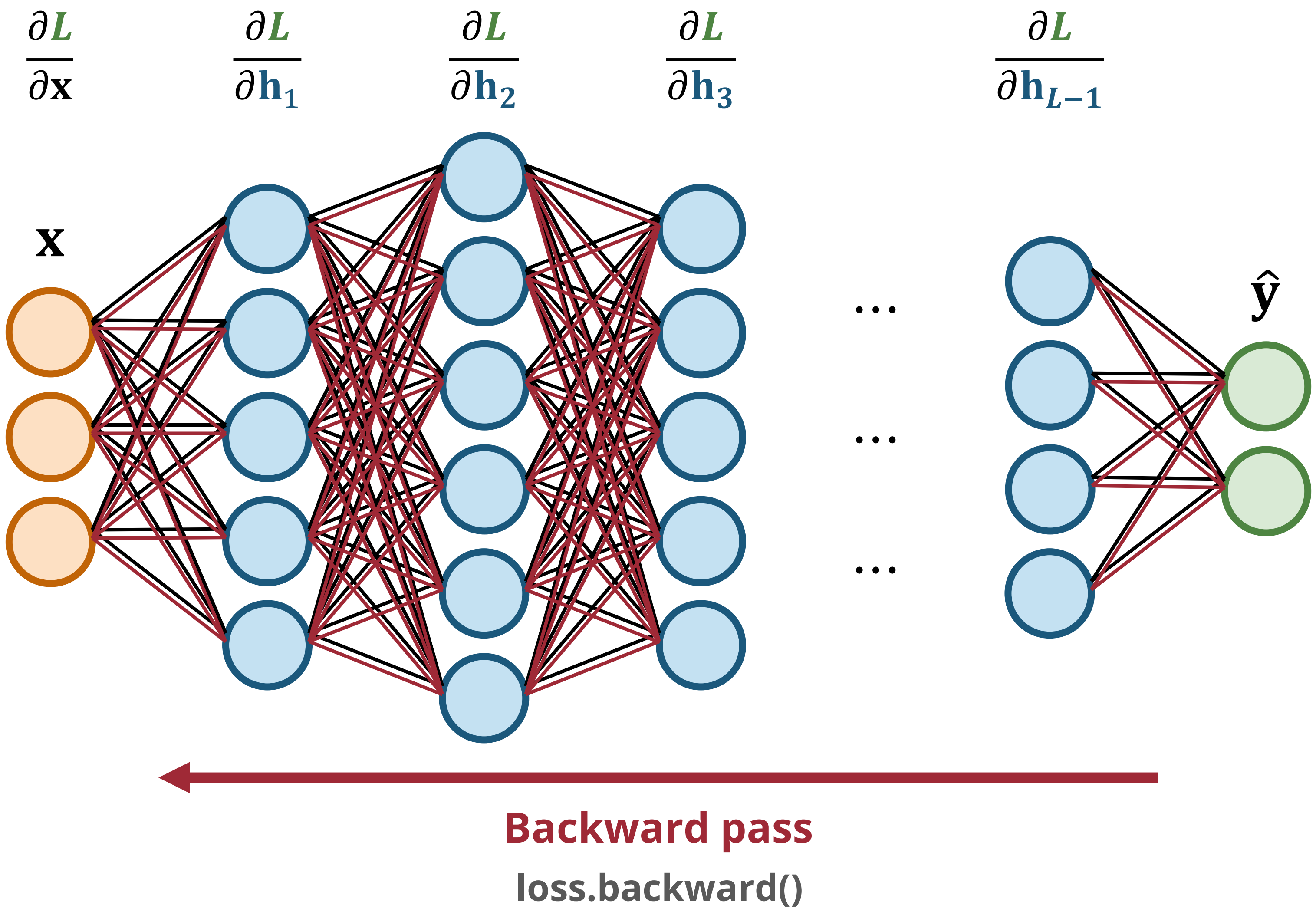
substance3d.adobe.com/community-assets/assets/9fe77d4279e12fe2fbd90b8f7ca18146a565f7ee

Backpropagation: Efficiently Computing the Gradients

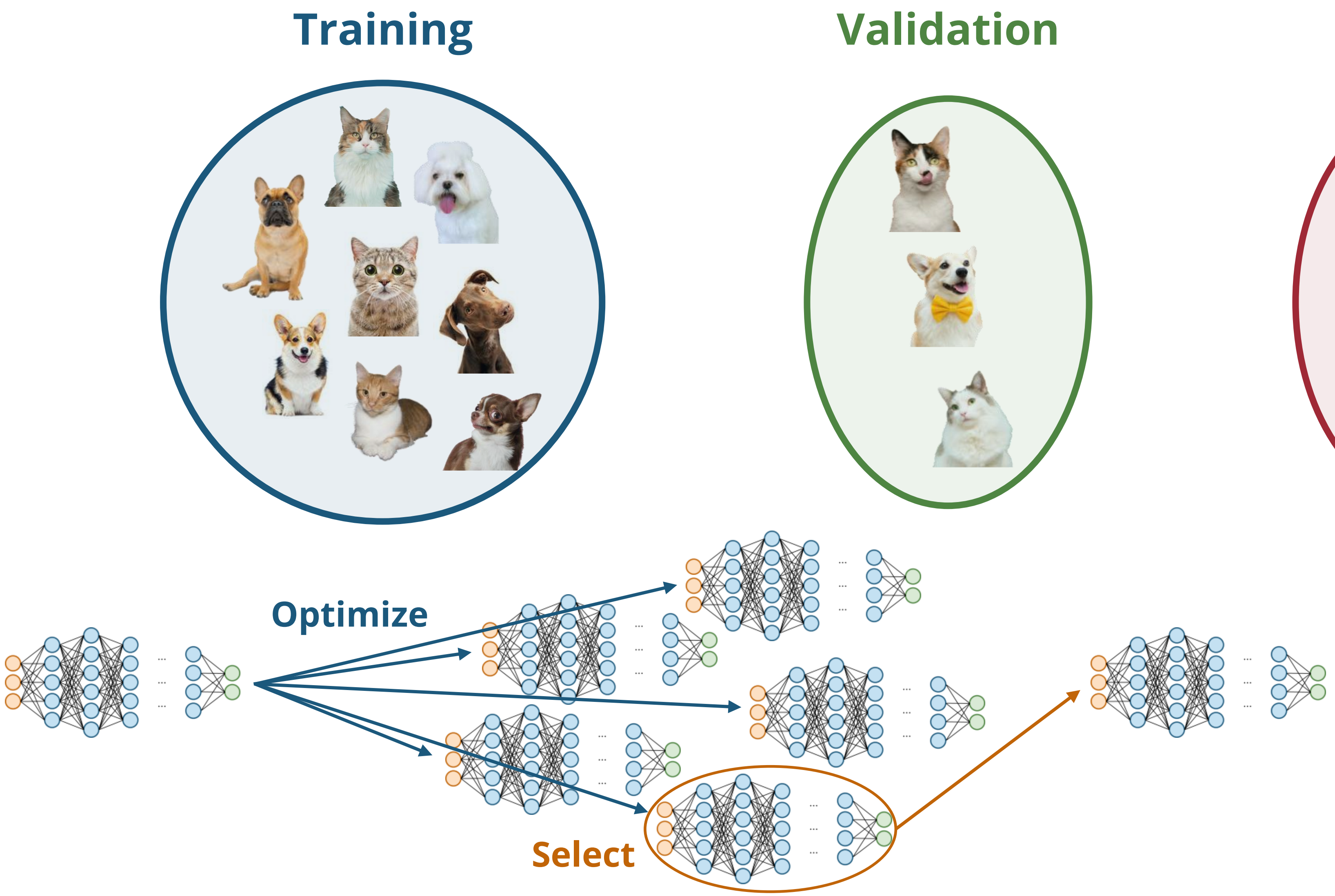
- An efficient way of **computing gradients** using chain rule
- The reason why we want **everything to be differentiable** in deep learning

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

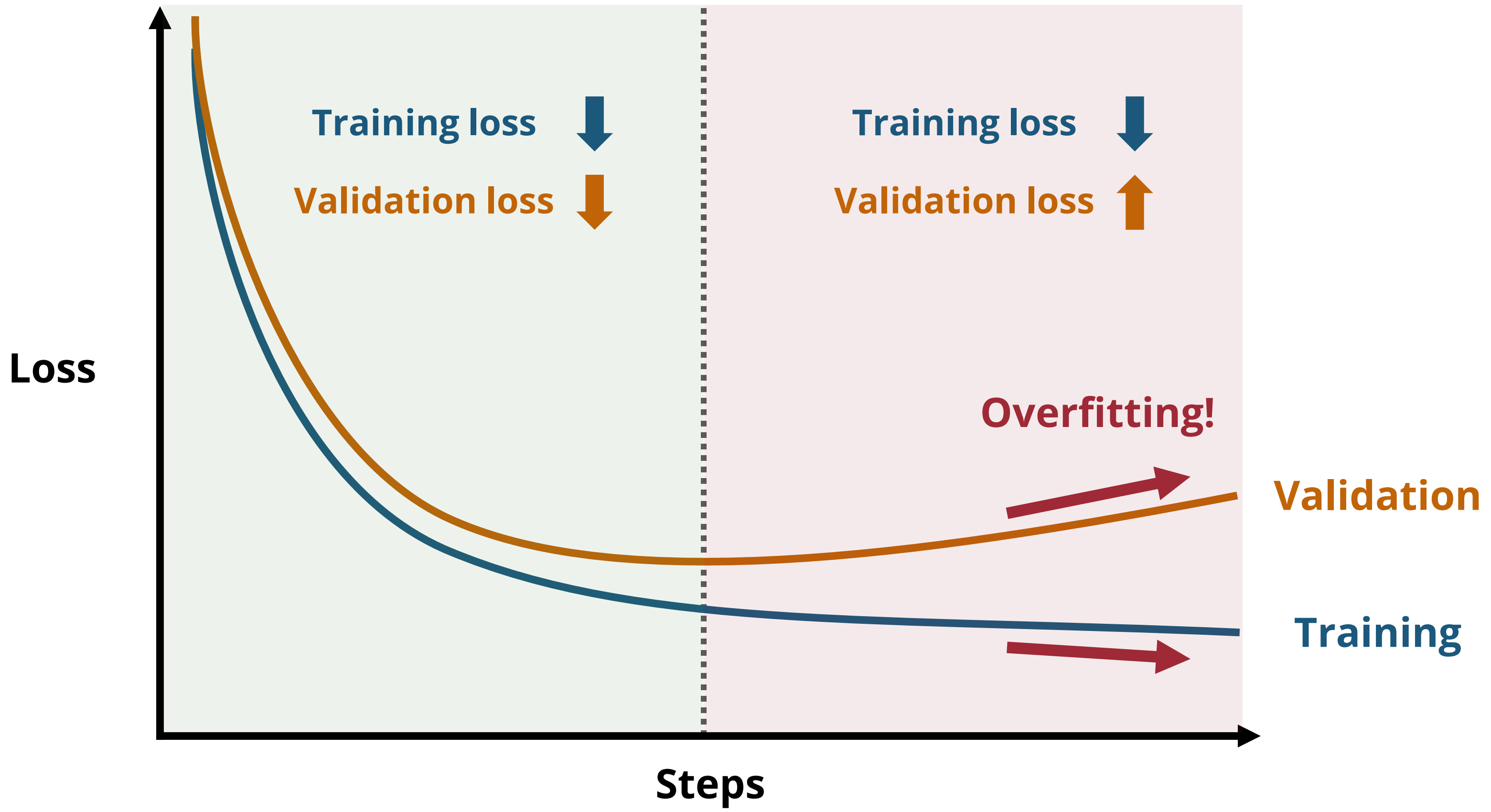
Forward Pass & Backward Pass



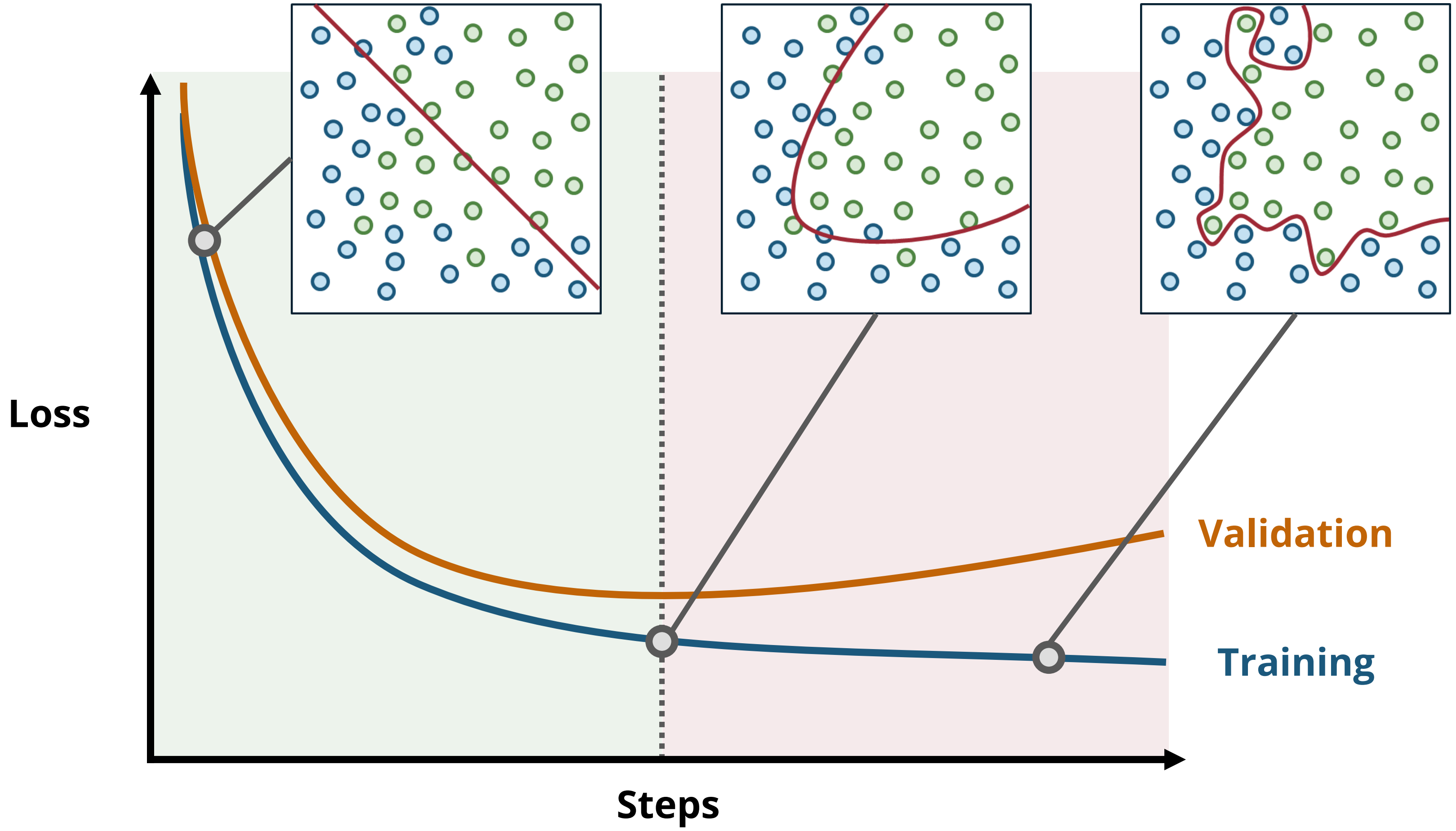
Training–Validation–Test Pipeline



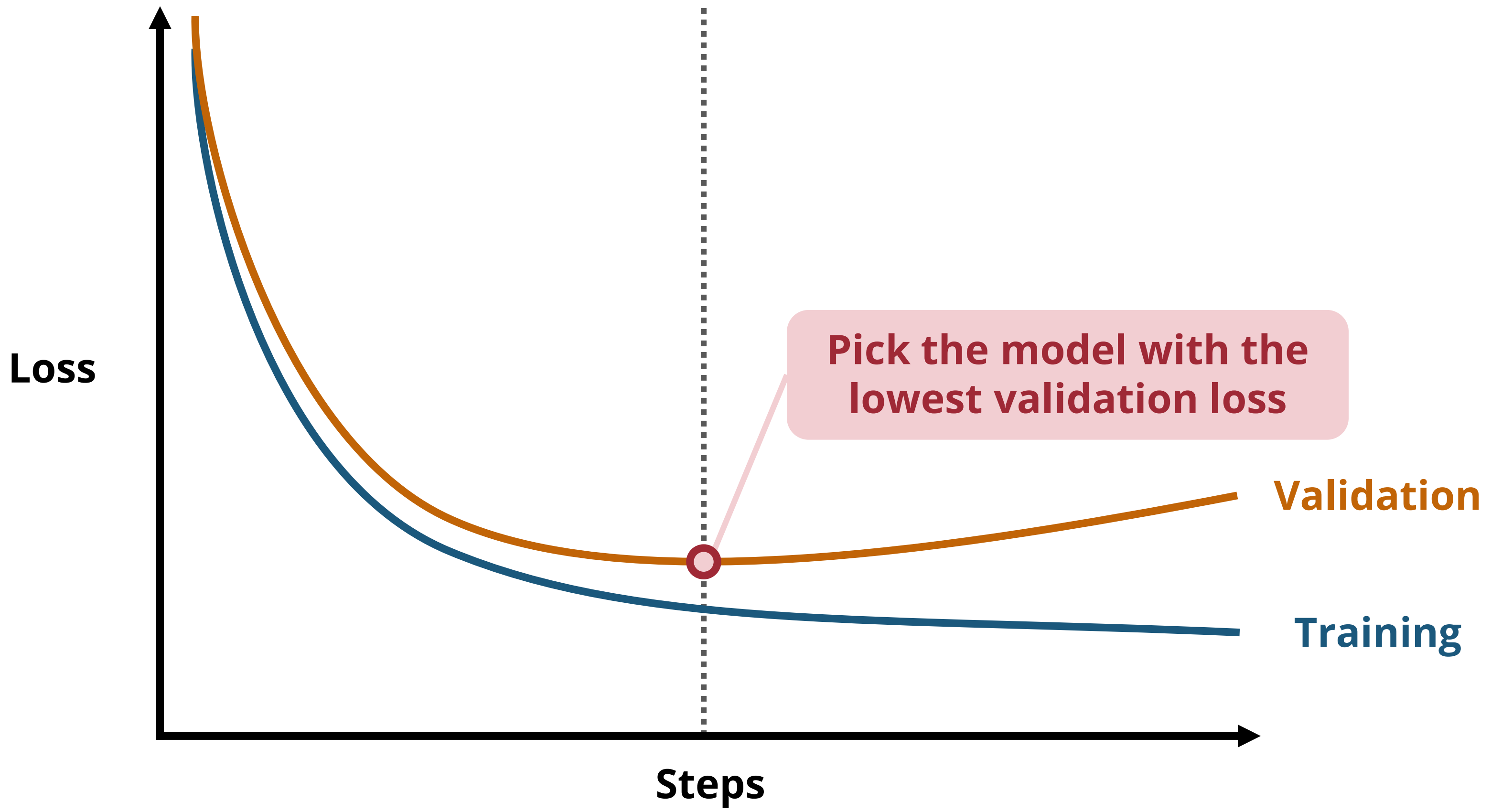
Training vs Validation Losses



Training vs Validation Losses

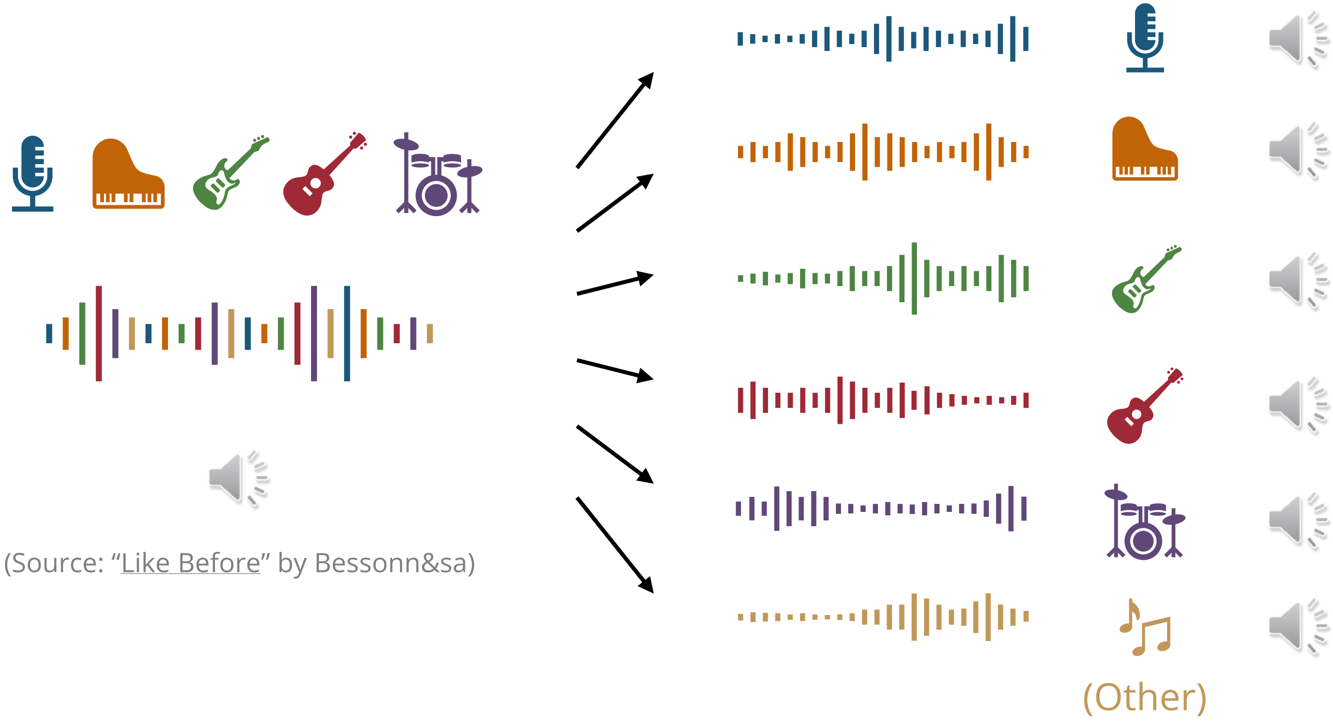


Training vs Validation Losses



Next Lecture

Source Separation



(Source: "Like Before" by Bessonn&sa)