EPFL

CS502: Deep Learning in Biomedicine
Deep Learning Basics

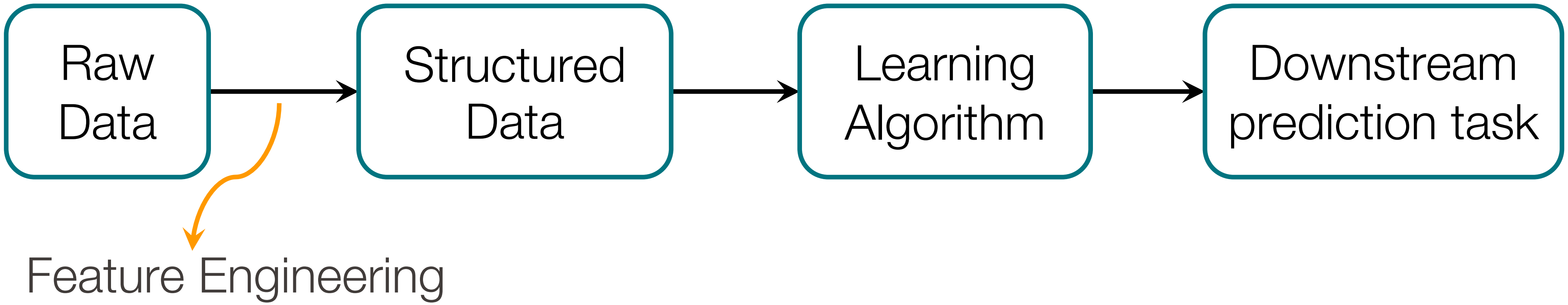
Maria Brbić

Fall 2023

École
polytechnique
fédérale
de Lausanne

Traditional Machine Learning

- Machine learning lifecycle



- Tedious manual effort on feature engineering
- Does not scale in practice and results in suboptimal features

Traditional Machine Learning: Example

DNA sequences

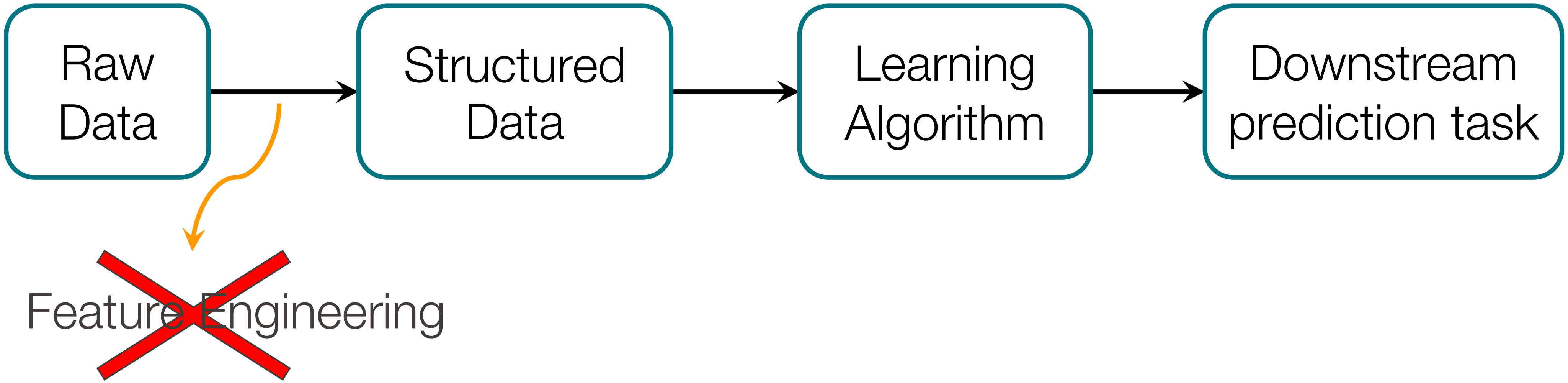
```
> F31.01
tattggacgggattgagtttaacgaacgccgttcctcaatgcgggtggttgcta
atgtag tgctaccgaacccaaagtccatcccgttagtcccttcaaagctttca
acgaggaactgg

> F31.02
aagttttcggcttcaaagggagataccaaatacggcggaaataactttccgcag
ccgttttattgagtgttagcagaatgatctccccccatgttgataaggcttaac
tggcttaccatctcttggggcacttgggtcctgctagacggtacgtgcctcgcc
cgtagctaacgacttgcg agtatgcacacttgtccaataggcaacgcctggct
ttgtg

> F31.03
gcccaaatcggtcatatcccgggactggacaactaaatgtattaaagtctacag
tcagtgtccaccagccaccacagagctaccgcaccgtagattcattaggcgata
tatagacttgggctgagactgacacgggaccataactgattcgatctgtacctc
ctcccacgctattattaagattgtttcgtttgagatgctctacacggcgccatt
gcggtcataaactgaccaaccacaaacttgtcgcccatgcacgccagtaaggca
aactcctcggagatcggaagtcgaagct
```

ID	AT	AC	AG	AA	TA	TC	TG	TT	...
F31.01									
F31.02									
F31.03									

What is the problem with such data representation? 🤔

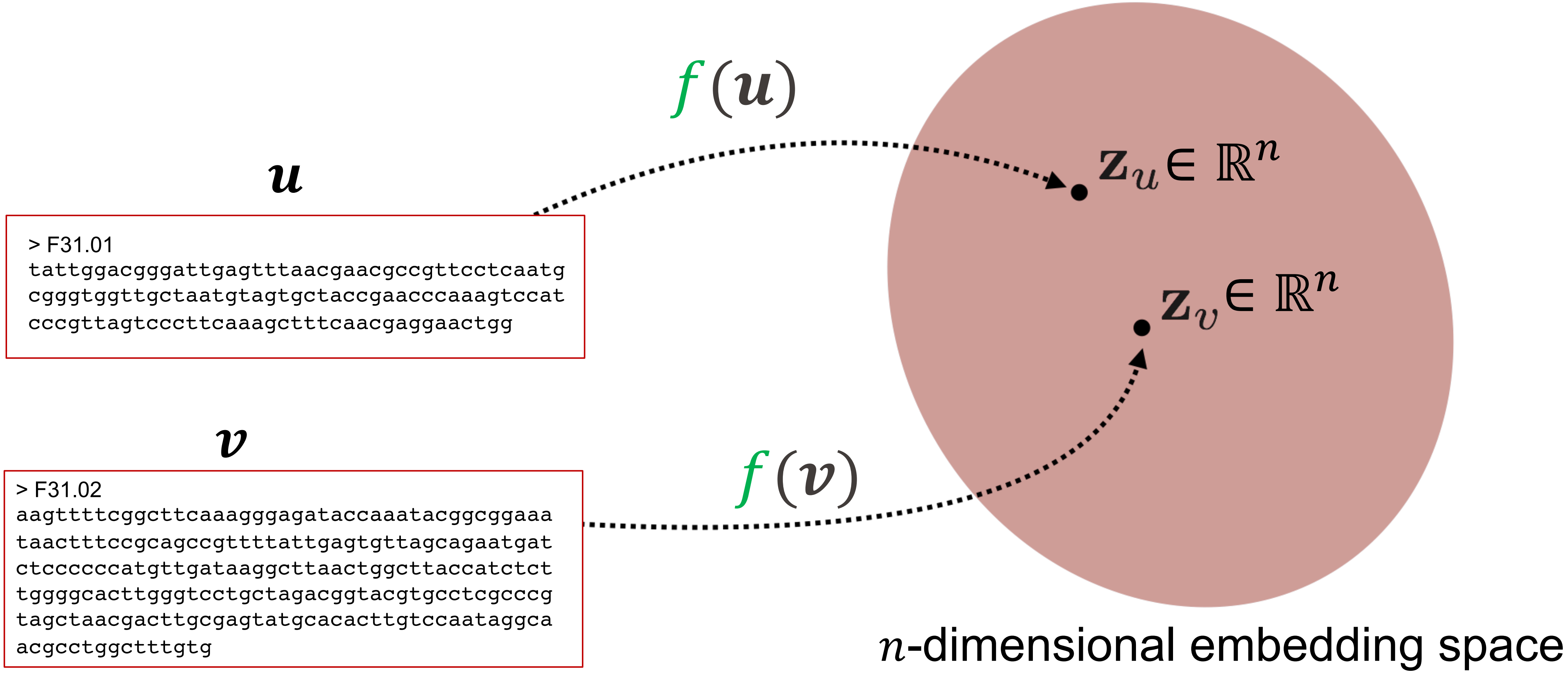
Deep Learning

- Machine learning lifecycle



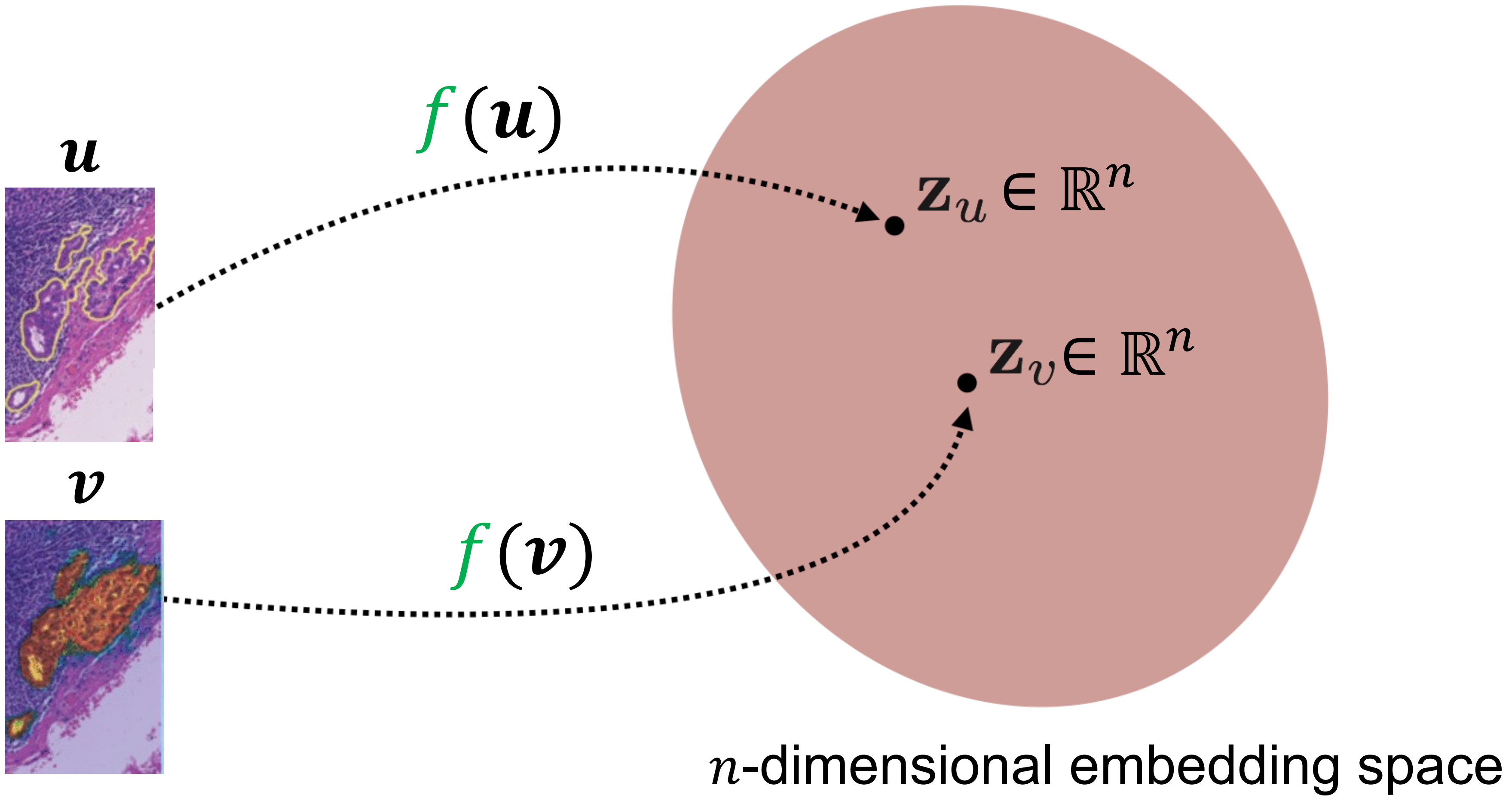
Automatically
learn the features!

Representation Learning



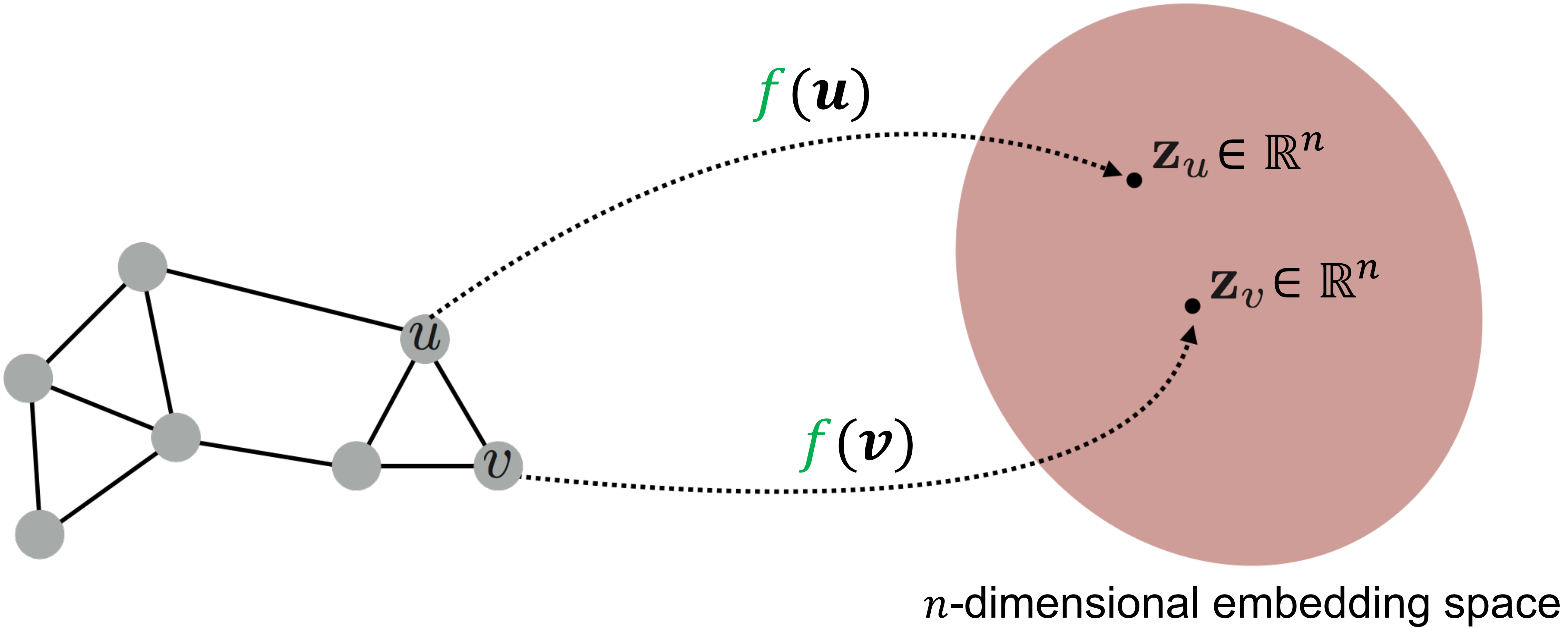
Key idea behind deep learning: Automatically learn underlying representations from the data!

Representation Learning



Key idea behind deep learning: Automatically learn underlying representations from the data!

Representation Learning

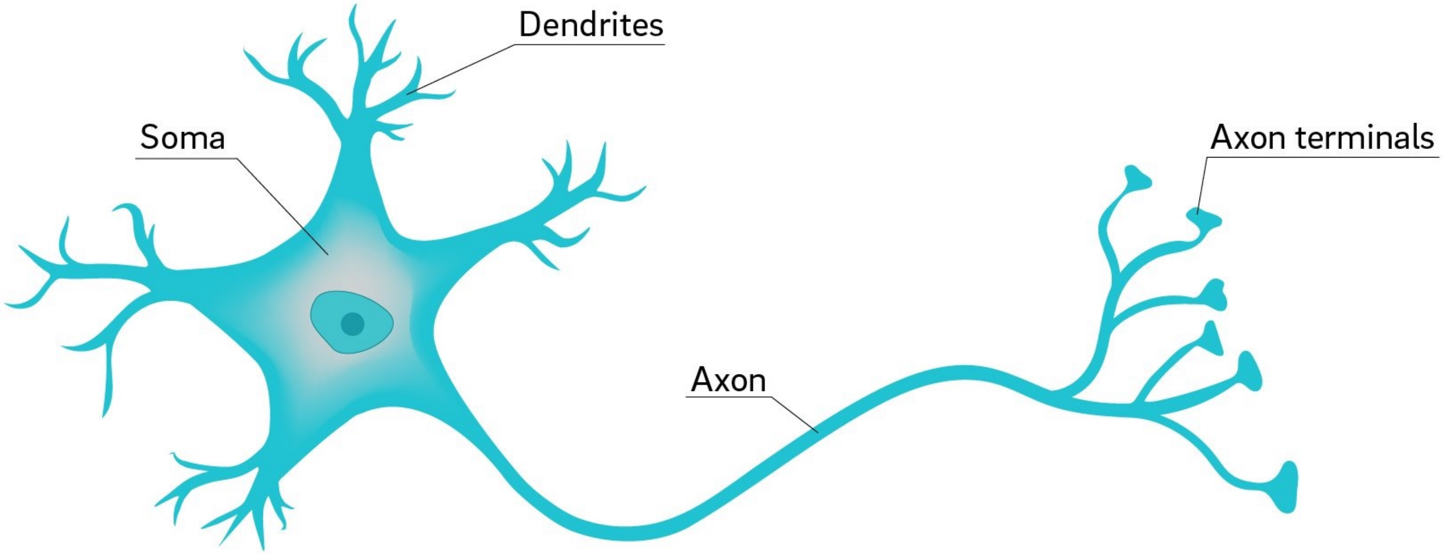


Key idea behind deep learning: Automatically learn underlying representations from the data!

Overview of neural networks

Inspiration

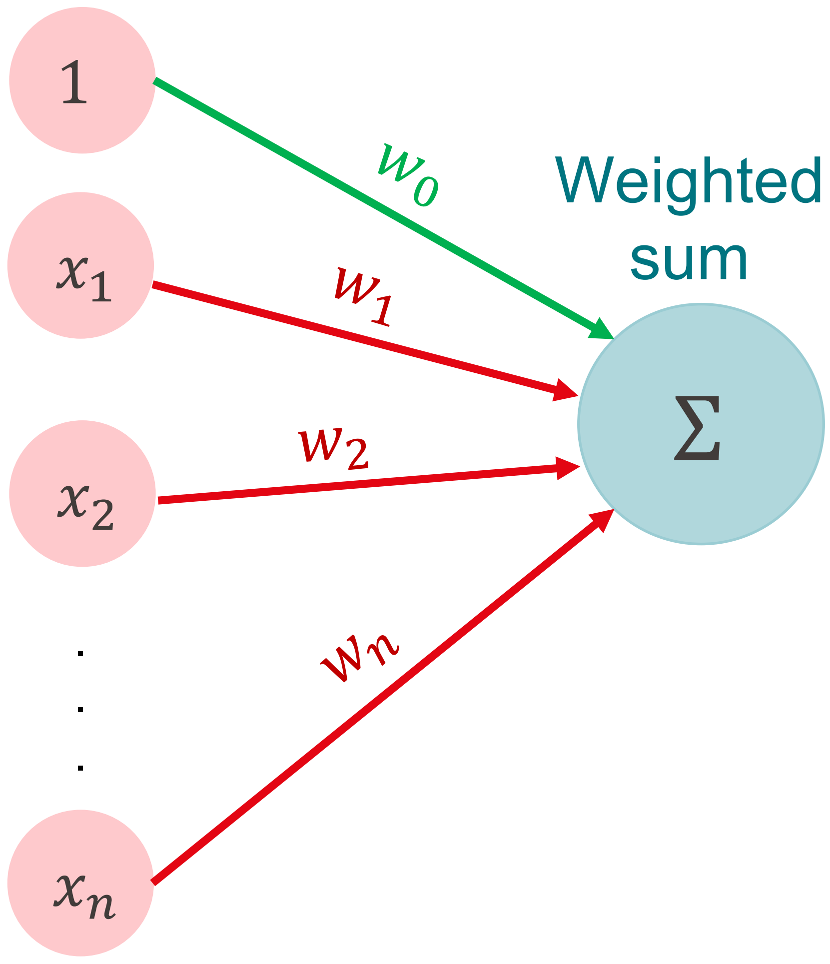
Neuron



How to transfer this idea to a computational model?

The Perceptron

- The structural building block of deep learning

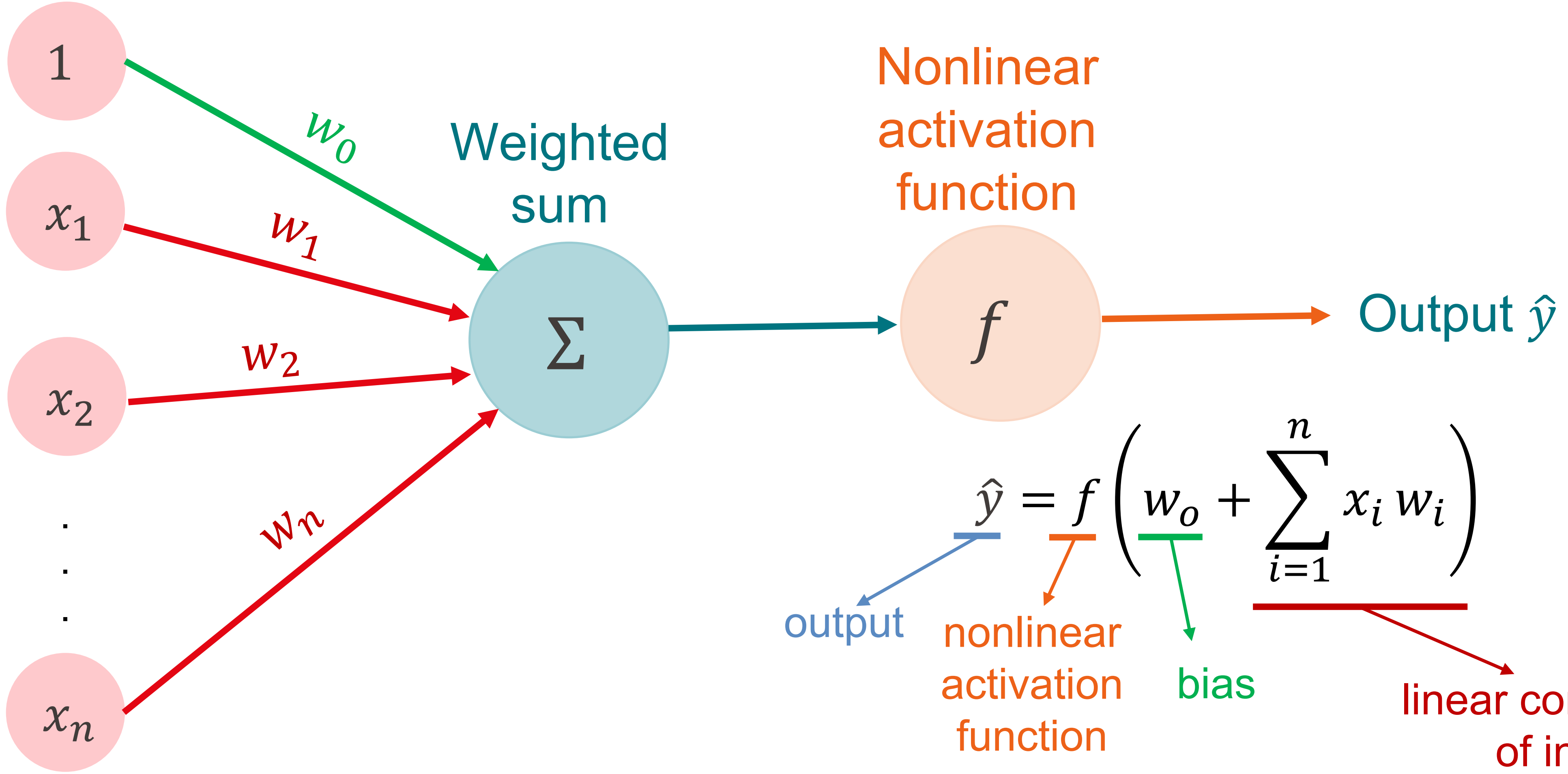


$$\underbrace{w_o}_{\text{bias}} + \underbrace{\sum_{i=1}^{n} x_i \, w_i}_{\text{linear combination of inputs}}$$

EPFL

The Perceptron

- The structural building block of deep learning



$$\underbrace{\hat{y}}_{\text{output}} = \underbrace{f}_{\text{nonlinear activation function}}\left(\underbrace{w_o}_{\text{bias}} + \underbrace{\sum_{i=1}^{n} x_i\, w_i}_{\text{linear combination of inputs}}\right)$$

The Perceptron: Forward Propagation

$$\underbrace{\hat{y}}_{\text{output}} = \underbrace{f}_{\text{nonlinear activation function}}\left(\underbrace{w_o}_{\text{bias}} + \underbrace{\sum_{i=1}^{n} x_i\, w_i}_{\text{linear combination of inputs}}\right)$$

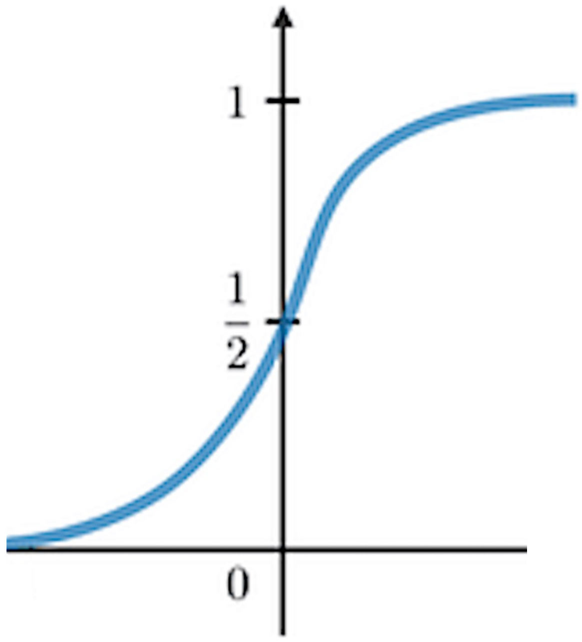
$$\hat{y} = f(w_o + \boldsymbol{x}^T \boldsymbol{w})$$

$$\text{where } \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad and\ \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix}$$

What is activation function f?

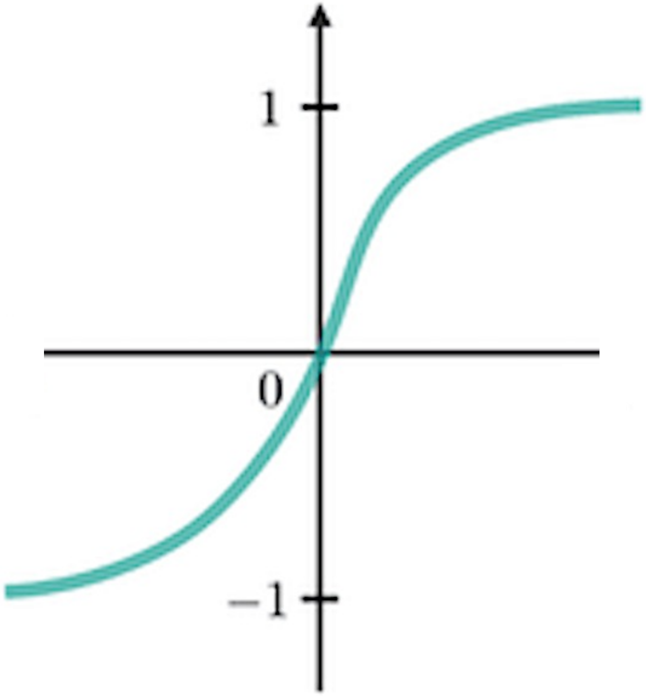
The Perceptron: Activation Functions Examples

Sigmoid



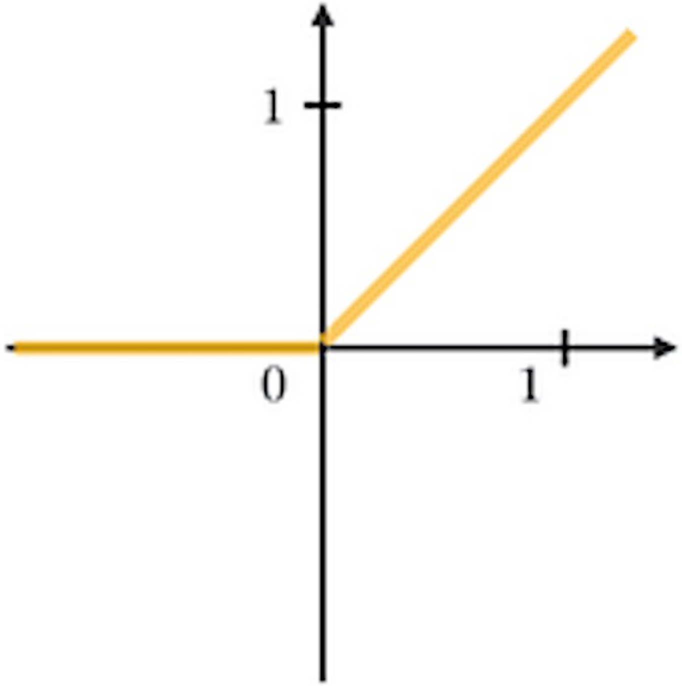
$$f(x) = \frac{1}{1 + e^{-x}}$$

Tanh



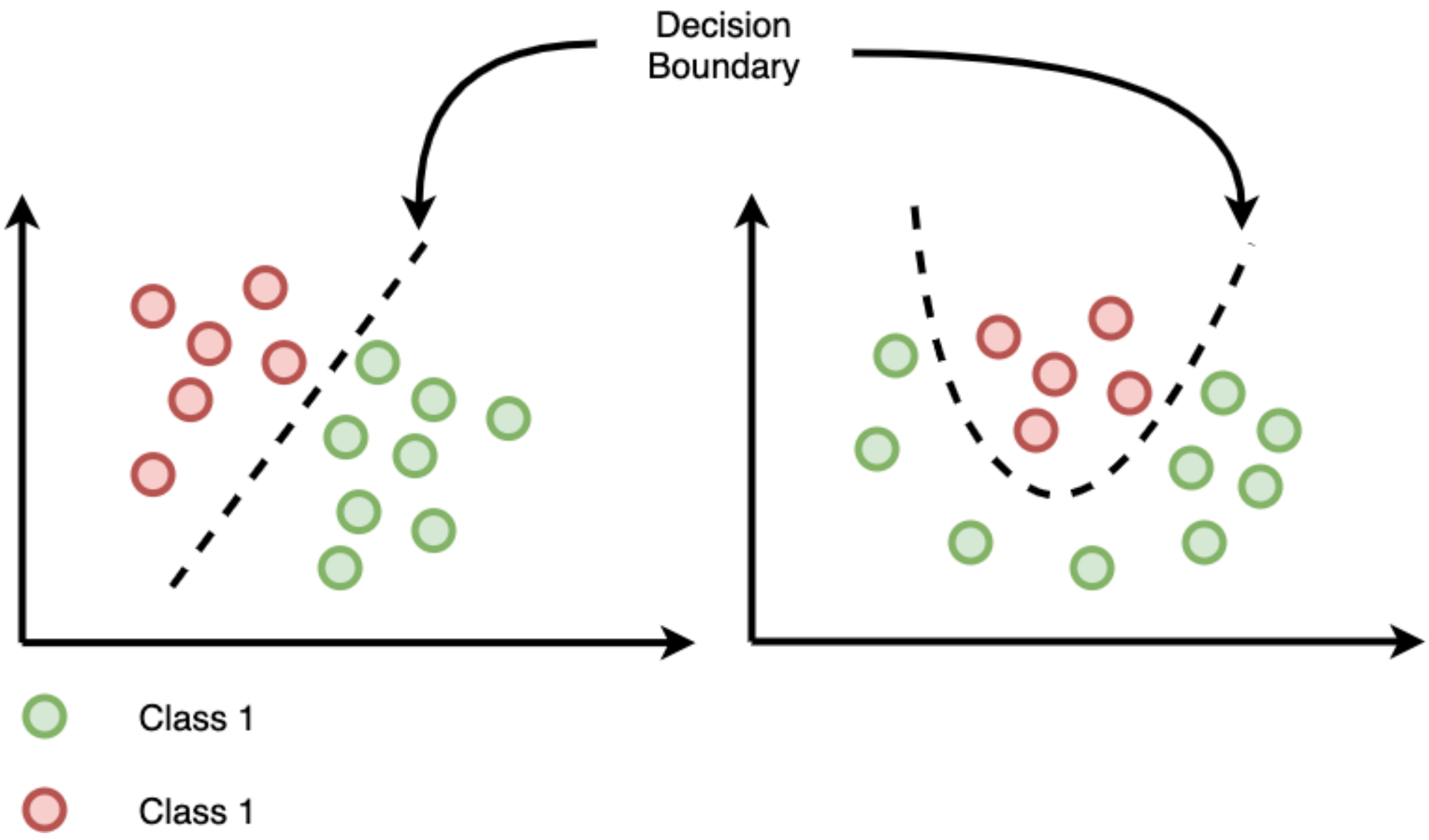
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU

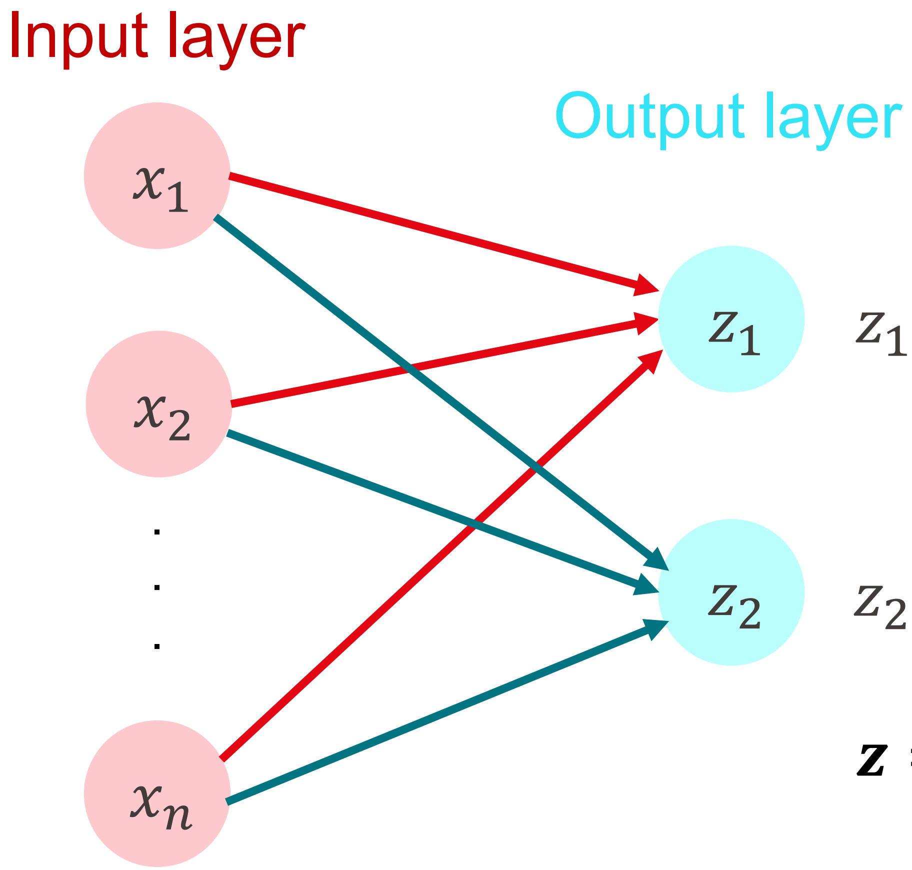


$$f(x) = \max(0, x)$$

Why is Nonlinearity Important?



Multi-dimensional Output



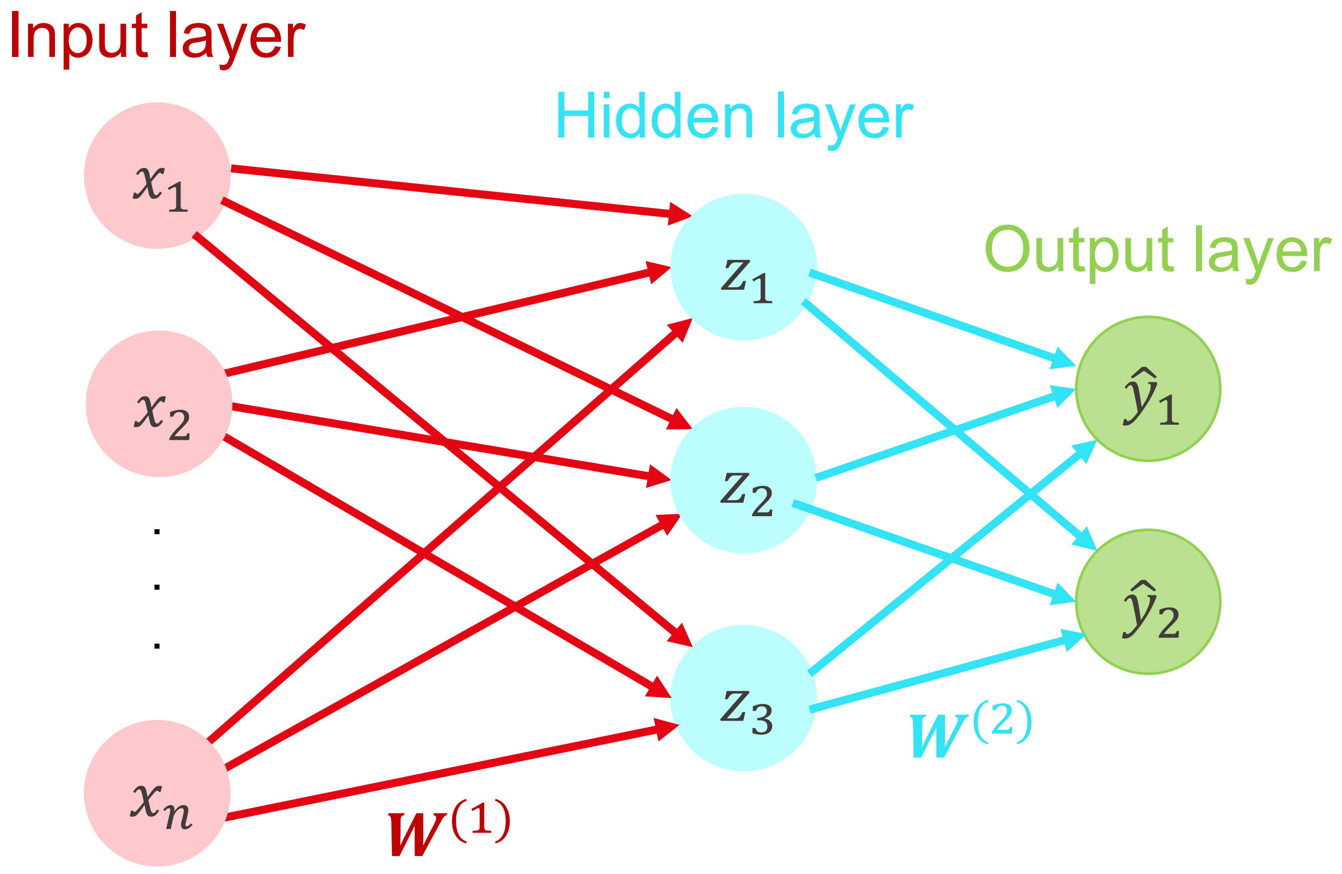
$$z_1 = f(w_{01} + \boldsymbol{x}^T \boldsymbol{w_1})$$

$$z_2 = f(w_{02} + \boldsymbol{x}^T \boldsymbol{w_2})$$

$$\boldsymbol{z} = f(\boldsymbol{b} + \boldsymbol{x}^T \boldsymbol{W})$$

where $\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} w_{01} \\ w_{02} \end{bmatrix}$ and $\boldsymbol{W} = [\boldsymbol{w_1} \boldsymbol{w_2}]$

Fully Connected Neural Network



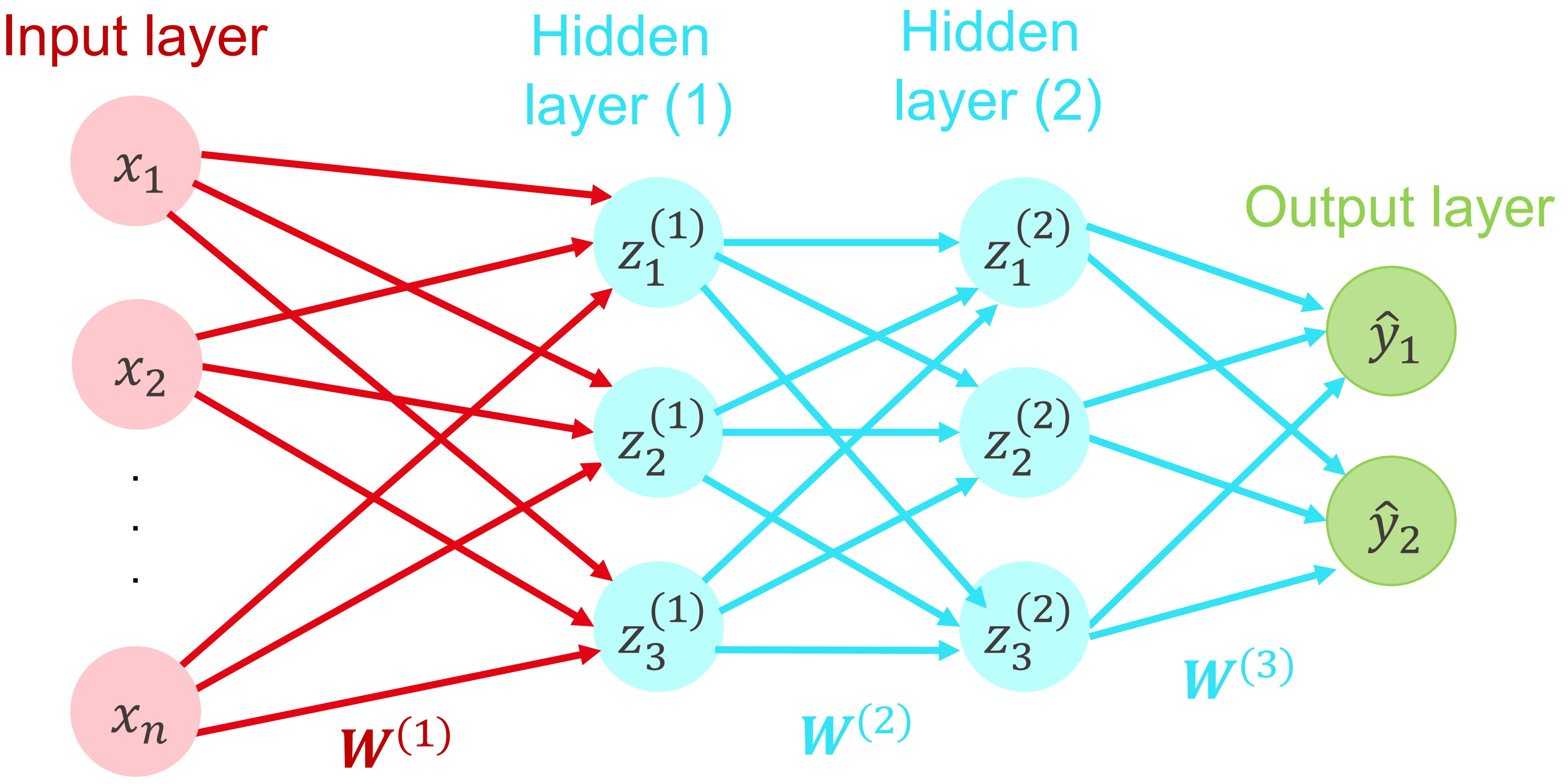
$$\boldsymbol{z} = f\left(\boldsymbol{b}^{(1)} + \boldsymbol{x}^T \boldsymbol{W}^{(1)}\right)$$

$$\hat{\boldsymbol{y}} = f\left(\boldsymbol{b}^{(2)} + \boldsymbol{z}^T \boldsymbol{W}^{(2)}\right)$$

Forward propagation

Deep Neural Network

- **Stacking multiple layers**



Neural Network Learning as Optimization

- **Supervised learning:** we are given input $\boldsymbol{x}$, and the goal is to predict label $\boldsymbol{y}$
- **Input x can be:**
  - Vectors of real numbers
  - Sequences (natural language)
  - Matrices (images)
  - Graphs (potentially with node and edge features)
- **We formulate the task as an optimization problem**

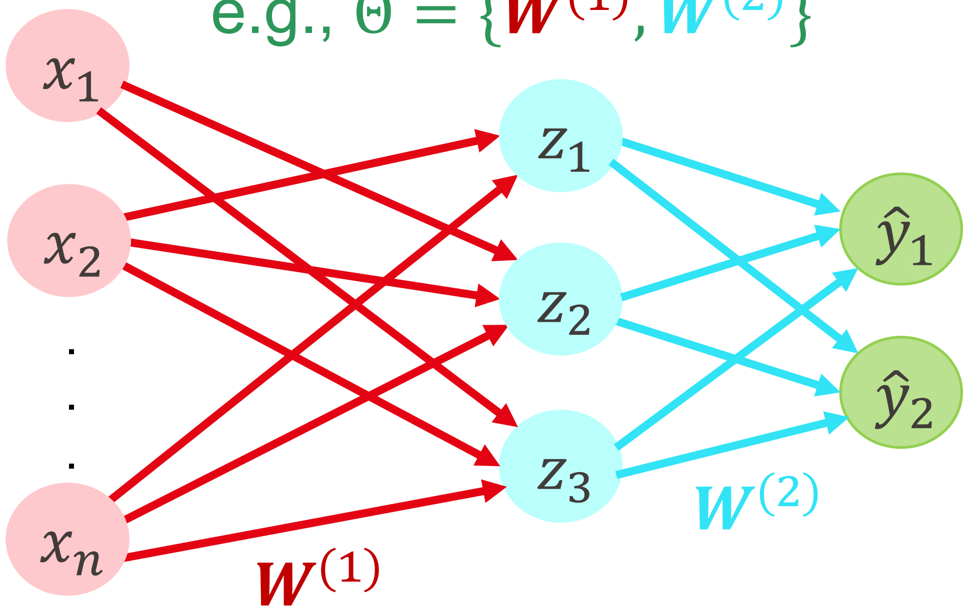
Neural Network Learning as Optimization

- **Formulate the task as an optimization problem:**

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f_{\Theta}(\boldsymbol{x}))$$ objective function

set of parameters we optimize

e.g., $\Theta = \{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}\}$



$f_{\Theta}(\boldsymbol{x})$ … our ML model

Neural Network Learning as Optimization: Loss Function

What is loss function $\mathcal{L}$?

- Depends on a problem
- Quantifies discrepancy between the truth $\boldsymbol{y}$ and prediction $f(\boldsymbol{x})$
- Example regression task: L2 loss

$$\mathcal{L}\big(\boldsymbol{y}, f(\boldsymbol{x})\big) = \|\boldsymbol{y} - f(\boldsymbol{x})\|_2$$

- Common loss for classification: cross entropy (CE)

$$\boldsymbol{CE}\big(\boldsymbol{y}, f(\boldsymbol{x})\big) = -\sum_{i=1}^{C} y_i \log f(x)_i,$$

where C is the number of classes.

labels vector

$\boldsymbol{y} =$	0	0	1	0	0

predictions

$f(\boldsymbol{x}) =$	0.1	0.3	0.4	0.1	0.1

Optimization: Gradient Descent

How to optimize the objective function?

- Iterative algorithm: Repeatedly update weights in the (opposite) direction of gradients until convergence

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} \mathcal{L}$$

learning rate
(controls size of the gradient step)

- Backpropagation: compute gradient using a chain rule
  - We will derive it in the exercise session!
  - Nowadays done using automatic differentiation
- Training: Optimize Θ iteratively
  - Iteration: 1 step of gradient descent

Optimization: Gradient Descent

How to optimize the objective function?

Input: input dataset $(\boldsymbol{X}, \boldsymbol{y})$, loss function $\mathcal{L}$, learning rate η, neural network f

1. Initialize parameters Θ
2. Loop until convergence
3. Compute gradient $\nabla_{\Theta}\mathcal{L}$
4. Update weights $\Theta \leftarrow \Theta - \eta\nabla_{\Theta}\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{X}))$

Output: parameters Θ

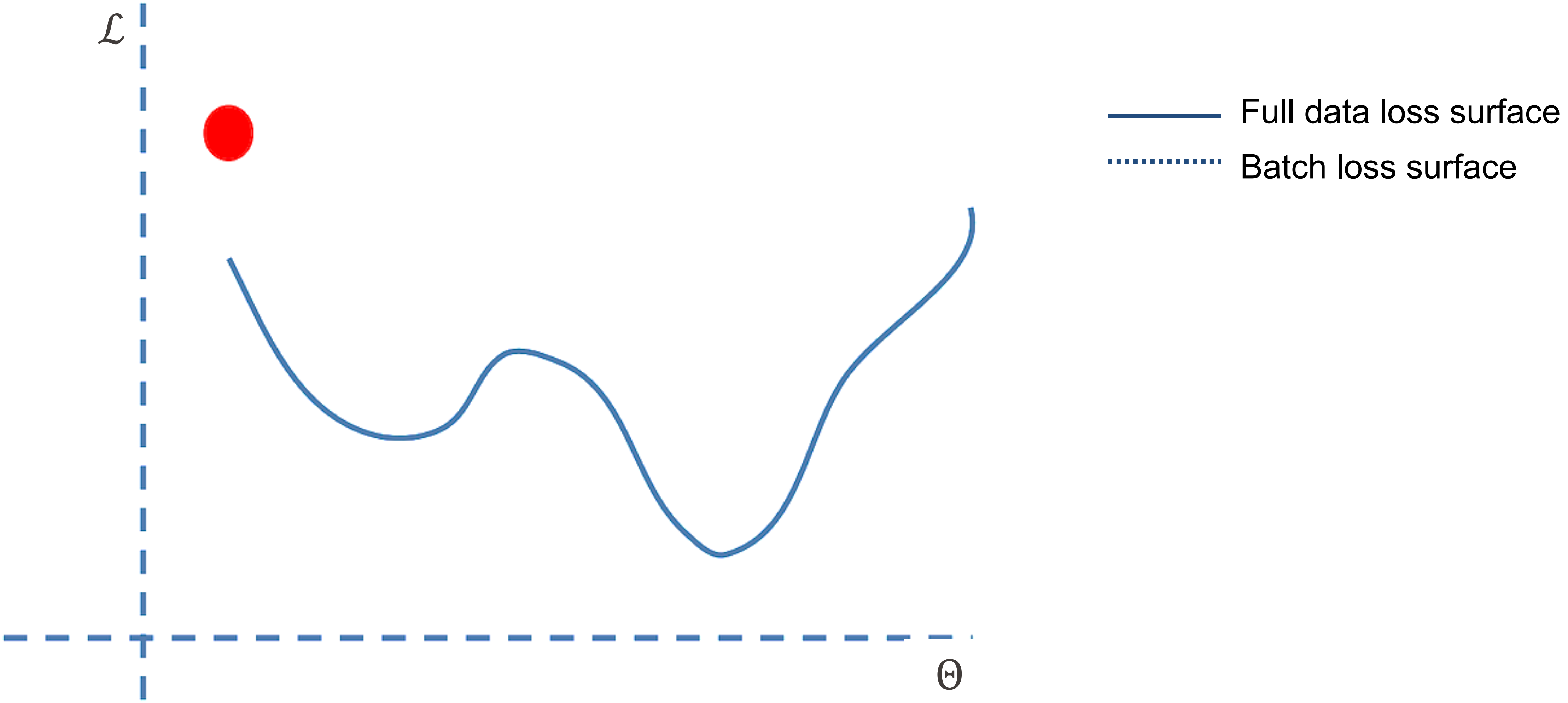
Problem with gradient descent:
Exact gradient requires computing
$\nabla_{\Theta}\mathcal{L}(y, f(X))$,
where X is the entire dataset.

Stohastic Gradient Descent (SGD)

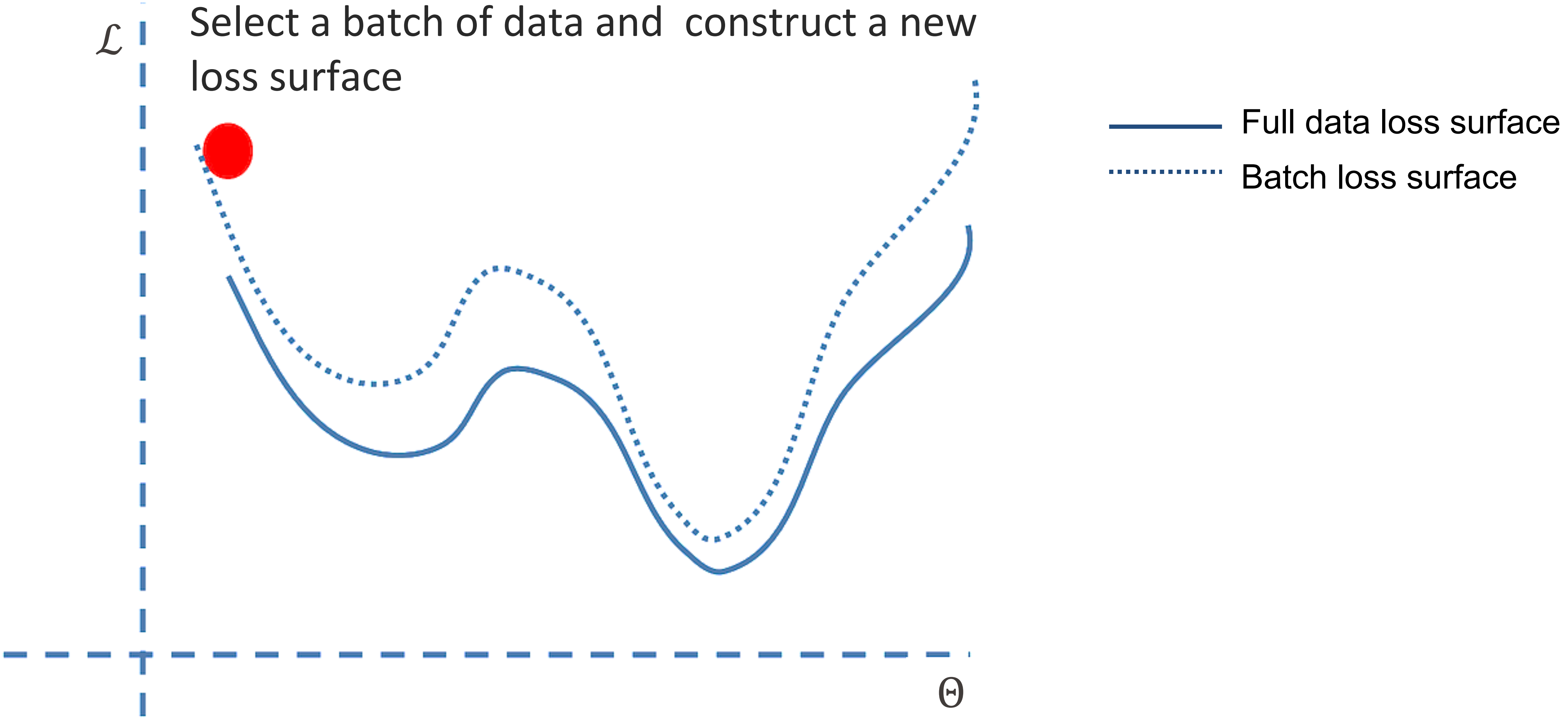
- **Solution: Stochastic gradient descent**
  - At every step, pick a different **minibatch** $\mathcal{B}$ containing a subset of the dataset, use it as input $\boldsymbol{x}$

- **Concepts:**
  - **Batch size:** the number of data points in a minibatch
  - **Iteration:** 1 step of SGD on a minibatch
  - **Epoch:** one full pass over the dataset (# iterations is equal to ratio of dataset size and batch size)

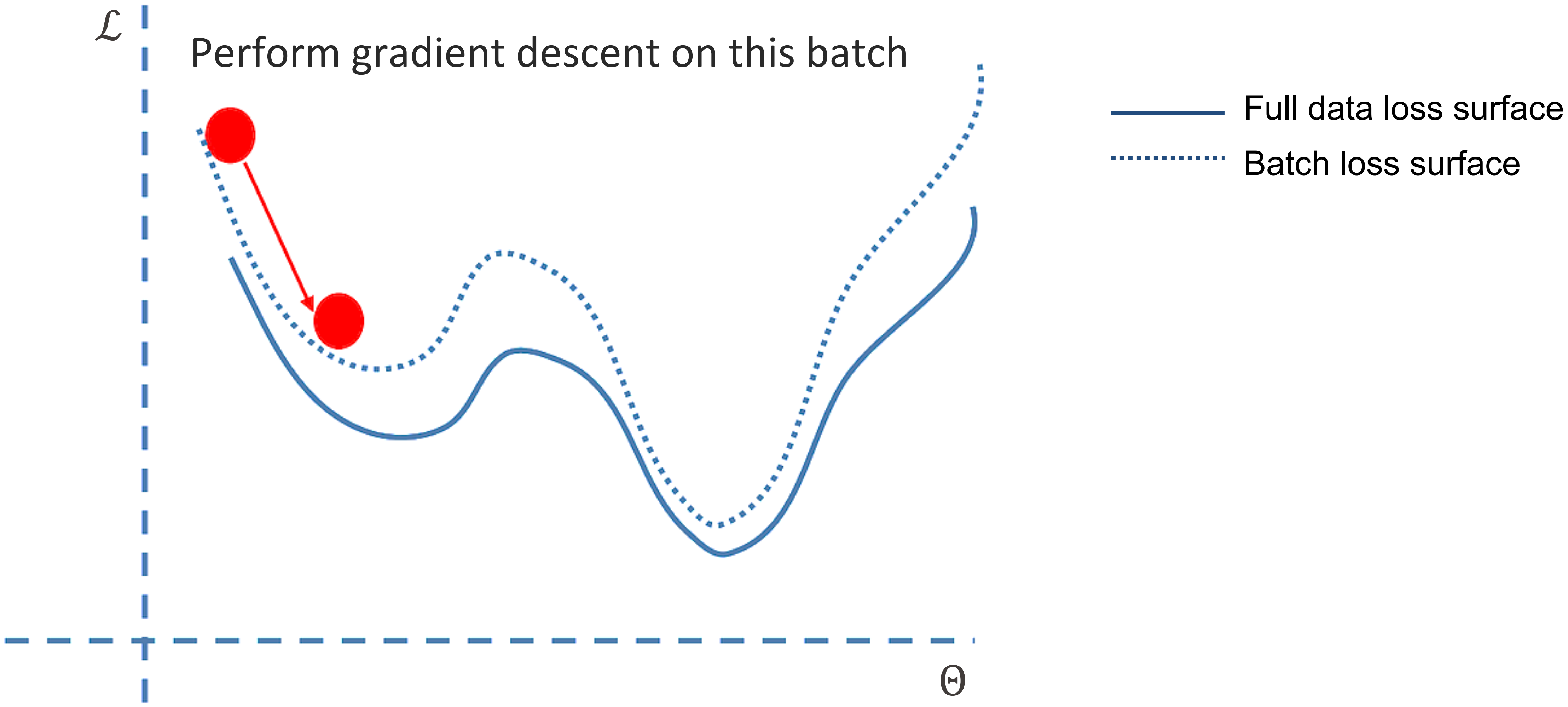
SGD example



SGD example

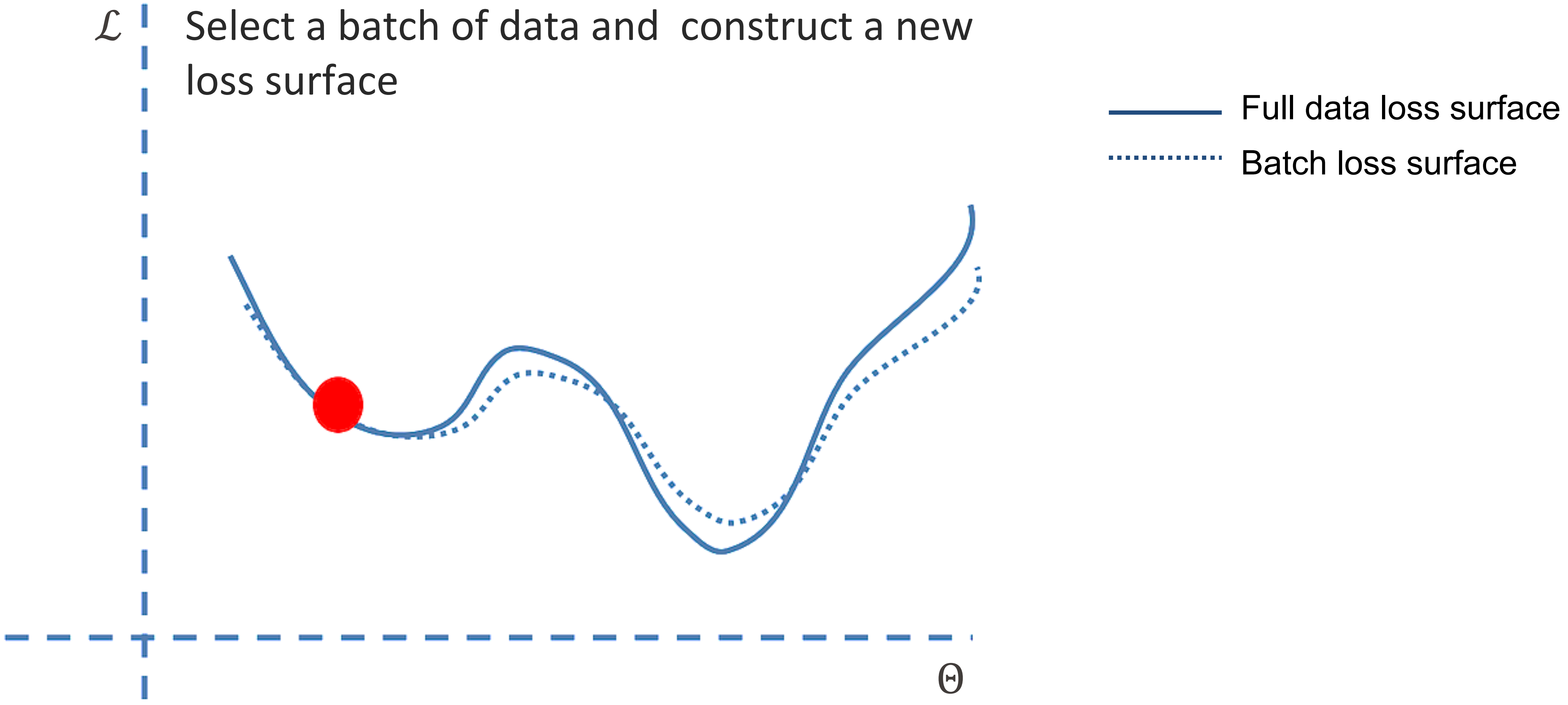

$\mathcal{L}$
Select a batch of data and construct a new loss surface
Θ
Full data loss surface
Batch loss surface

SGD example

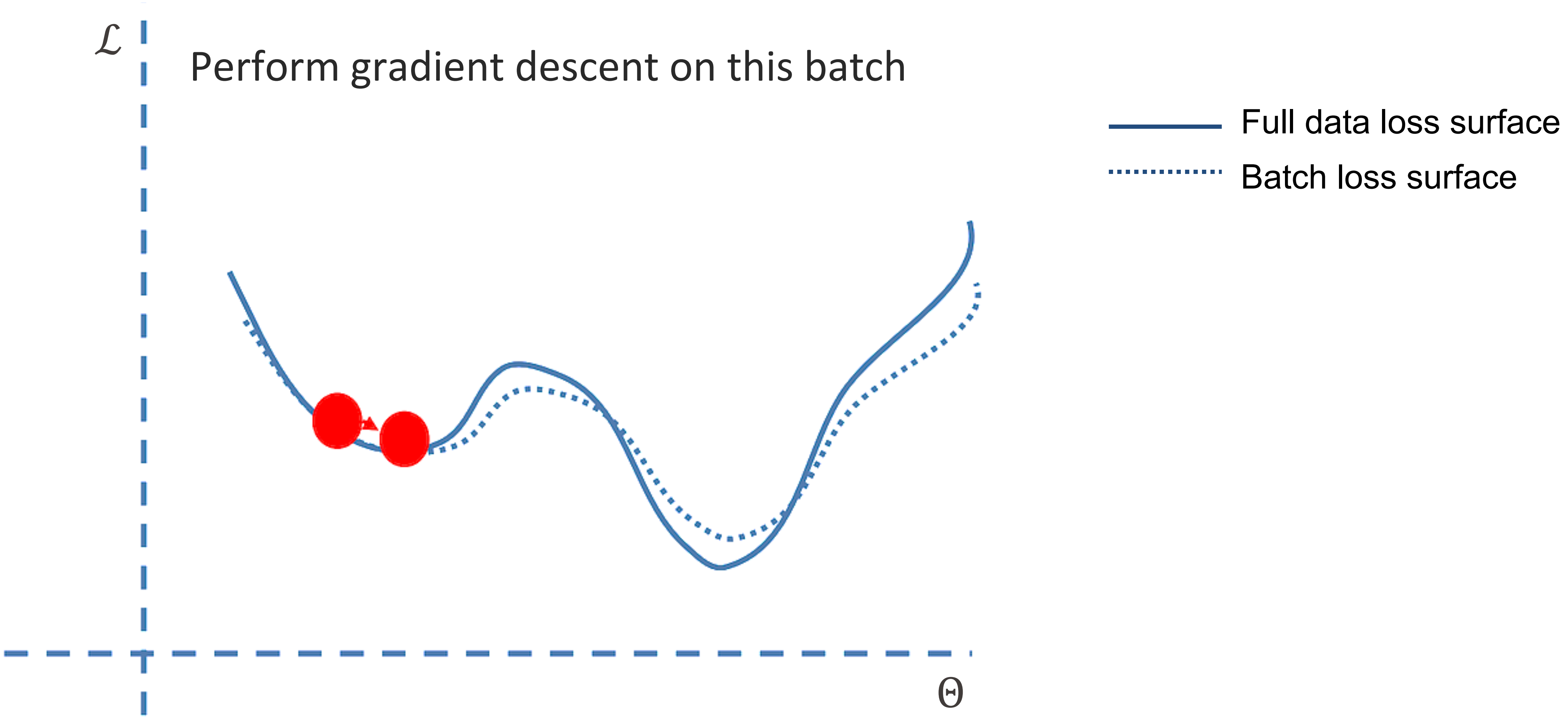

$\mathcal{L}$
Perform gradient descent on this batch
Θ
Full data loss surface
Batch loss surface

SGD example

Select a batch of data and construct a new loss surface

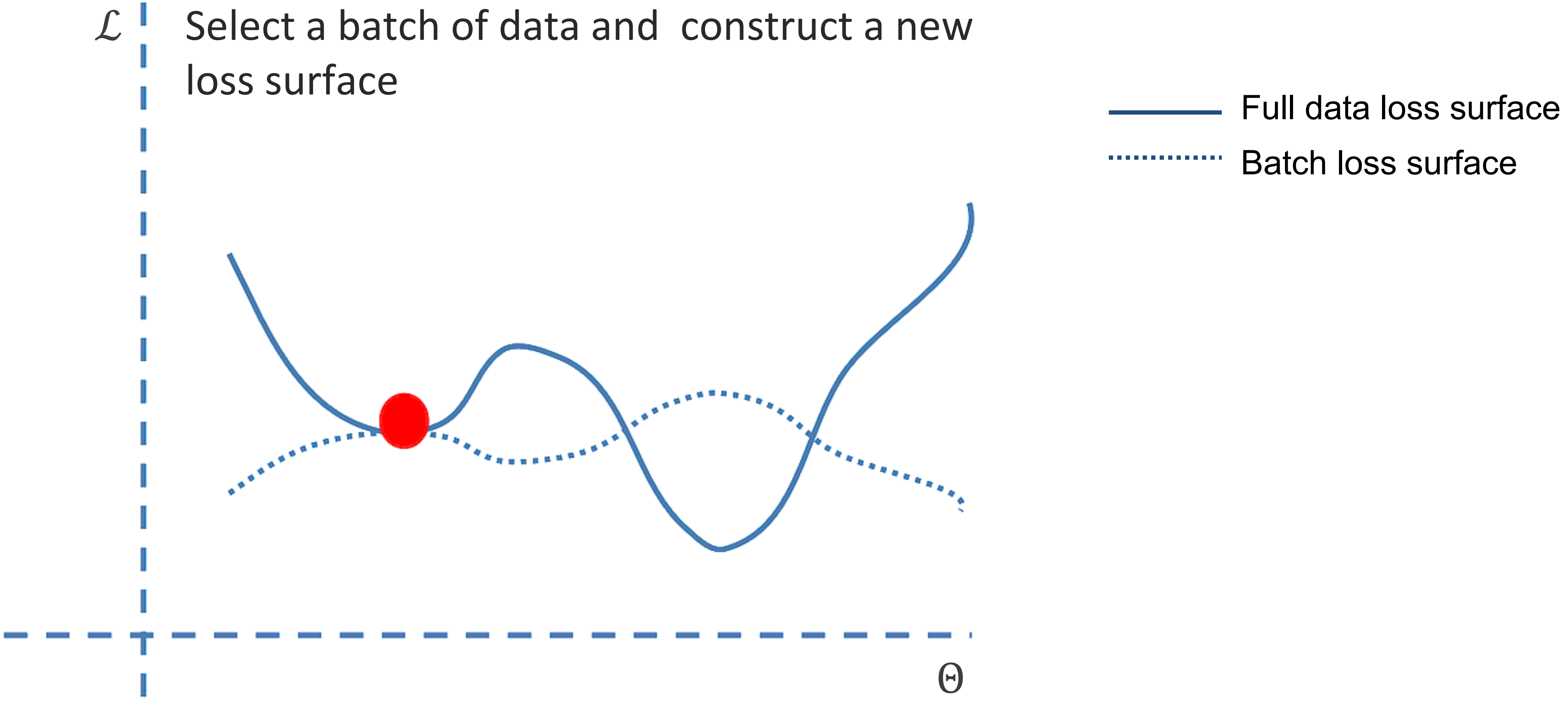


SGD example

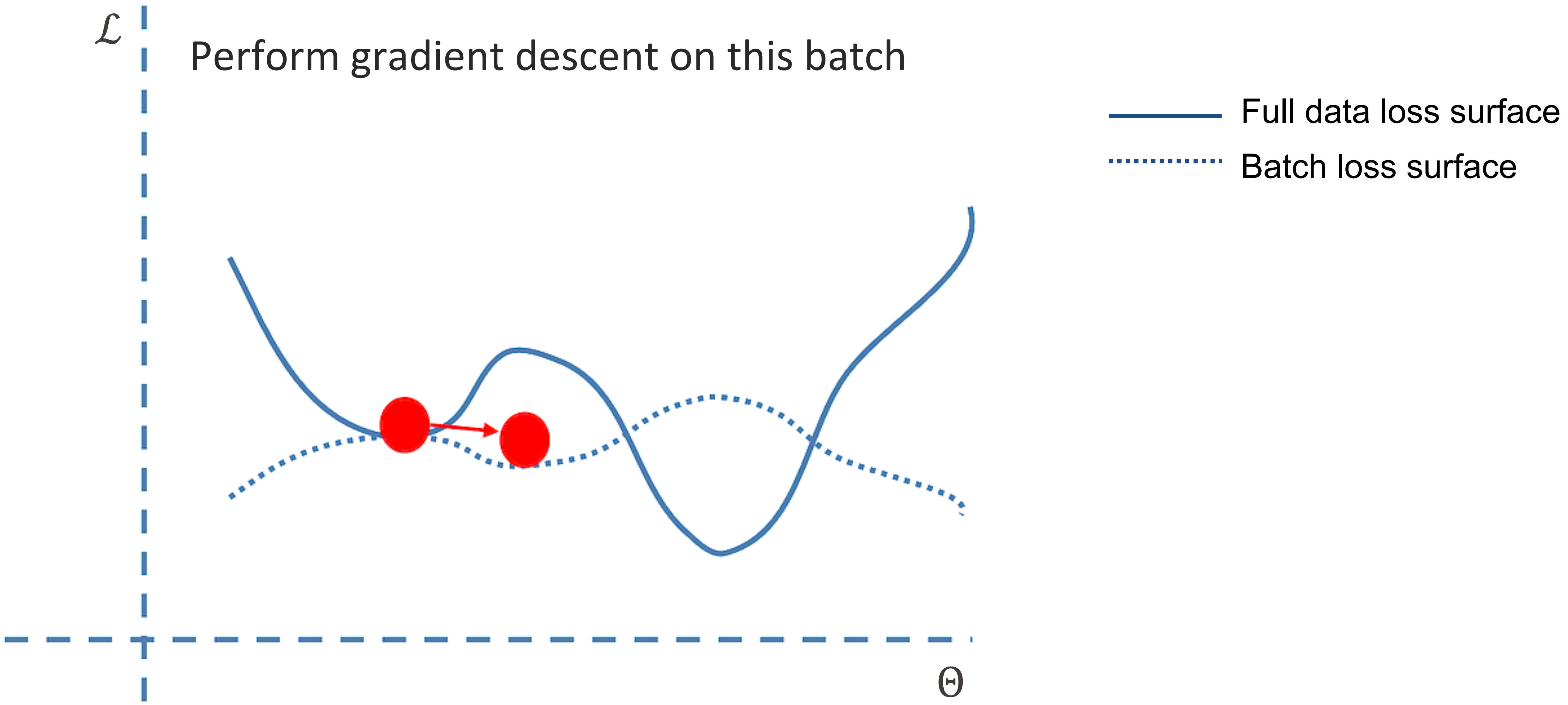

$\mathcal{L}$
Perform gradient descent on this batch
Full data loss surface
Batch loss surface
Θ

SGD example

Select a batch of data and construct a new loss surface

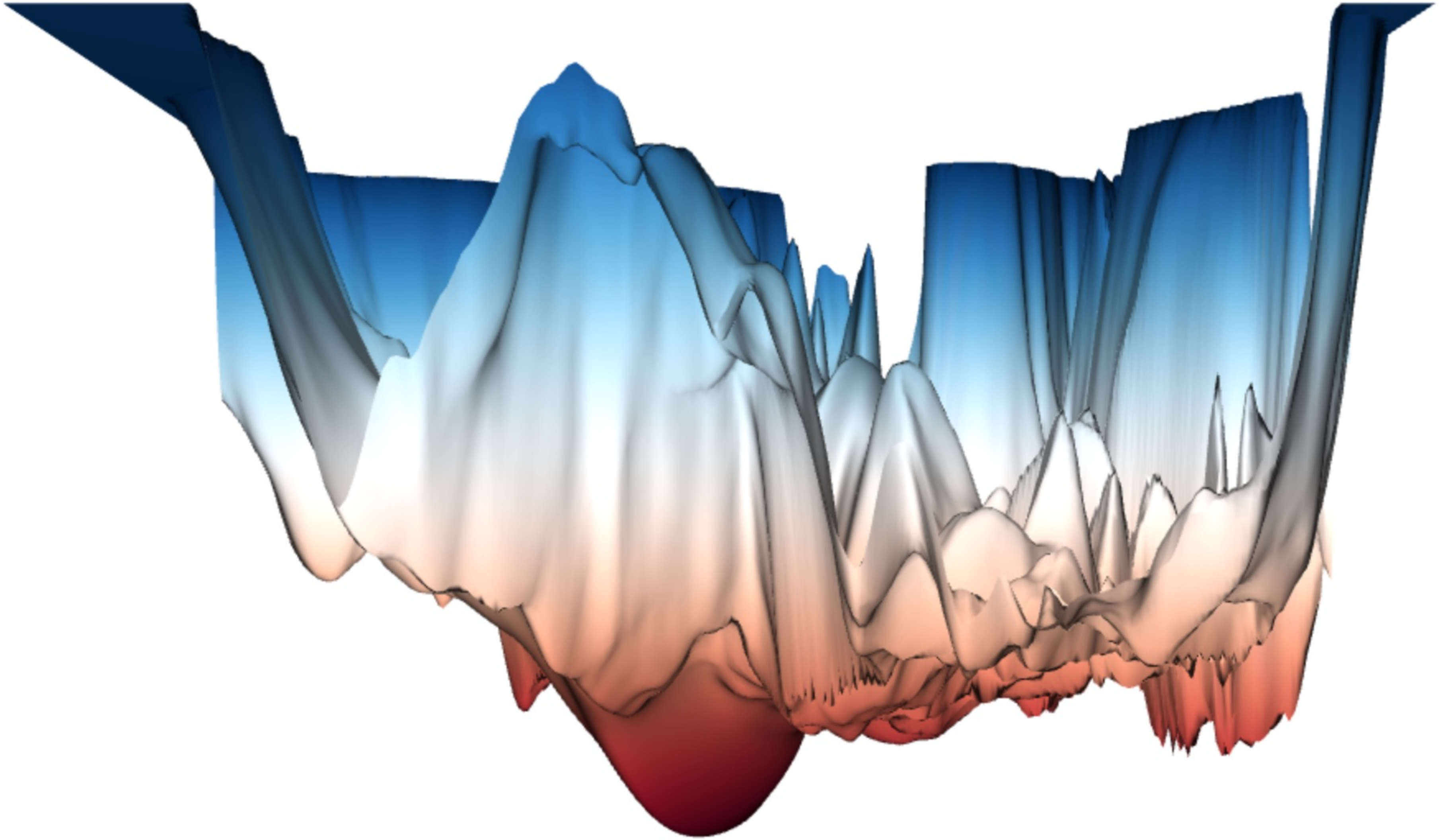


SGD example



$\mathcal{L}$
Perform gradient descent on this batch
Θ
Full data loss surface
Batch loss surface

Loss Surface: Reality



Visualizations from: http://www.telesens.co/loss-landscape-viz/viewer.html

More on https://losslandscape.com/

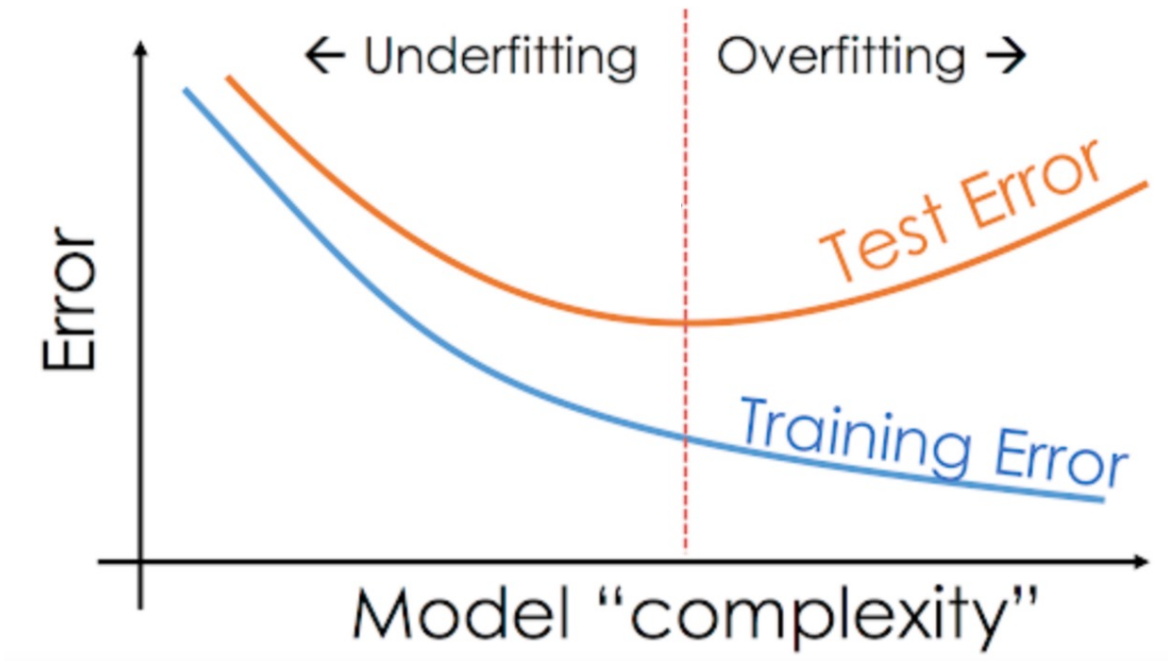
Optimization in Practice

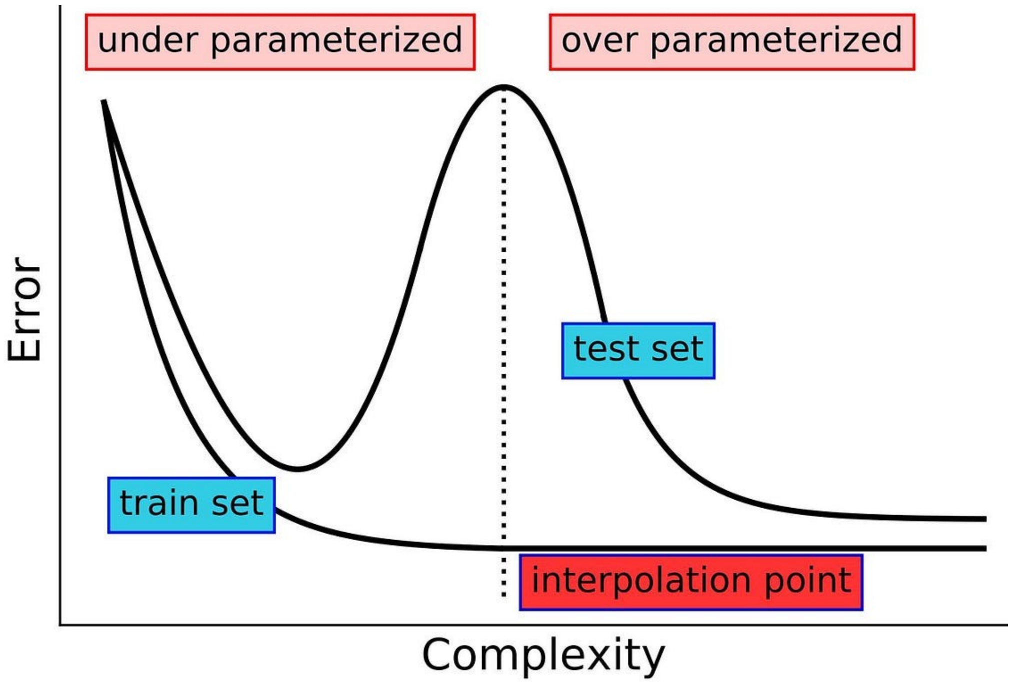
- **How to choose learning rate**
  - Too small → slow convergence
  - Too large → can cause the loss function to fluctuate around the minimum or even diverge
- Common strategies for improvement:
  - Momentum: helps accelerate gradients in the right direction
  - Adaptive learning rate
- **Common optimizers:** Adam, Adagrad, Adadelta, RMSprop…

Nice overview here: https://www.ruder.io/optimizing-gradient-descent/

How to Avoid Overfitting?

Use regularization!

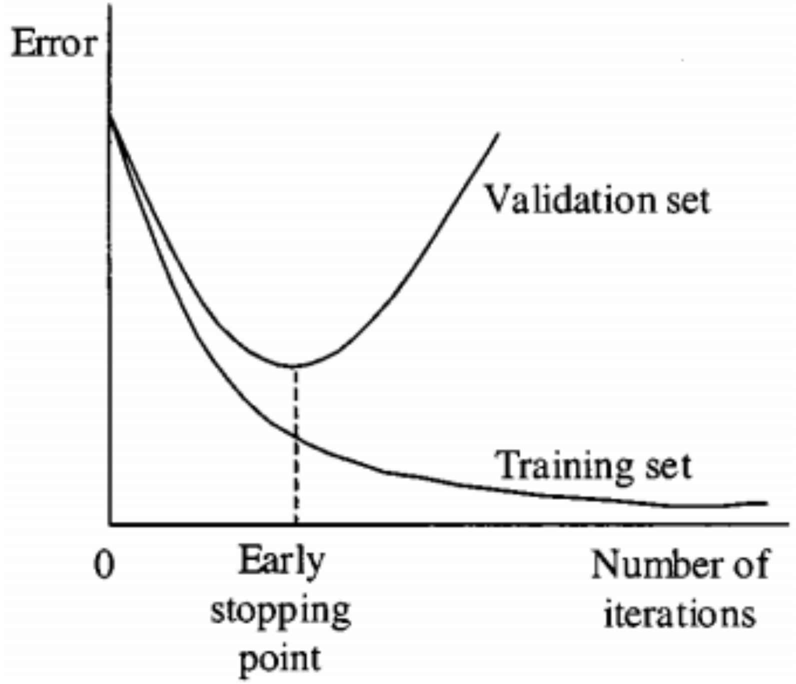




Double descent phenomena

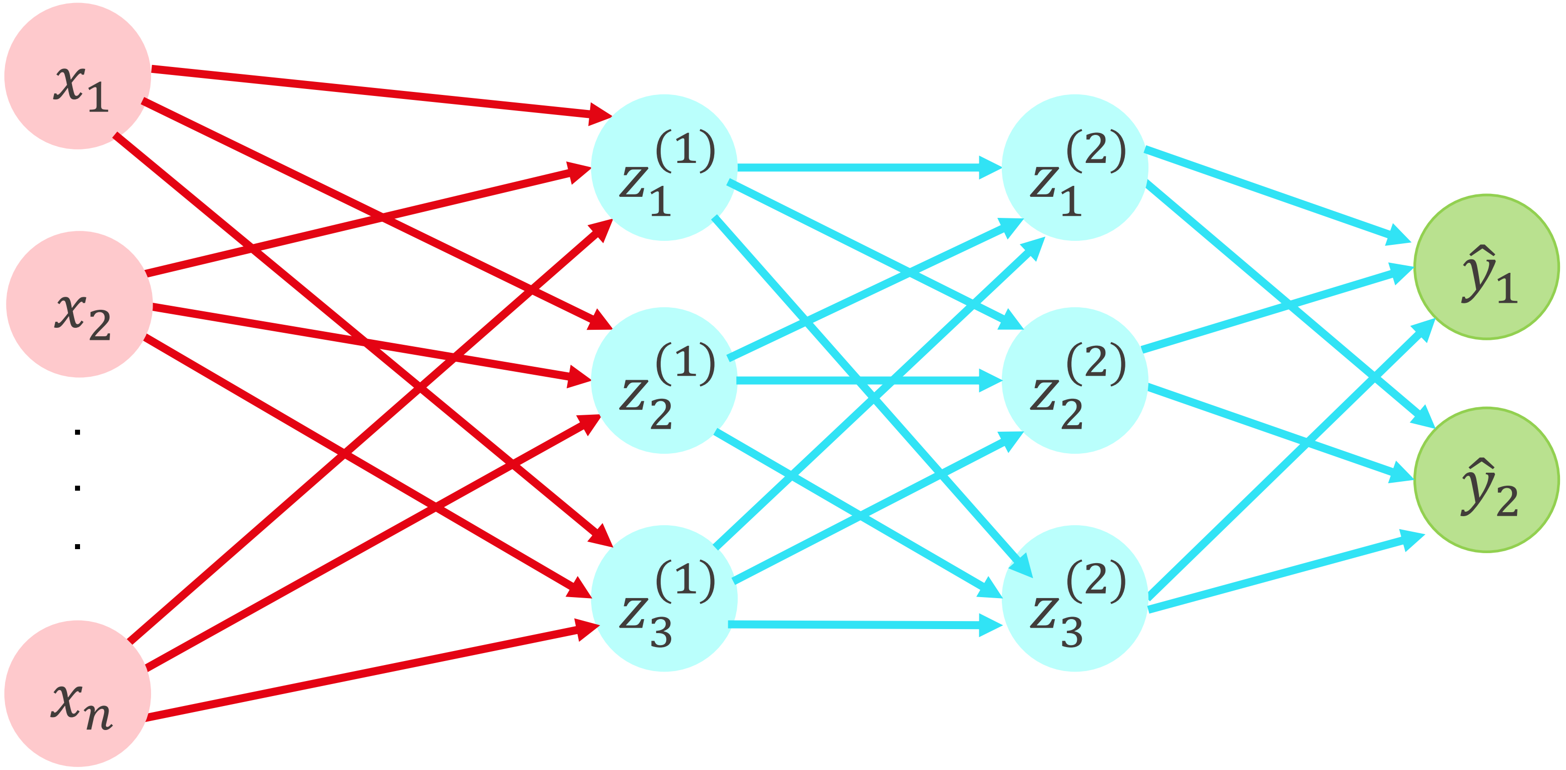
Regularization Techniques

- **Early stopping**
  - Stop training when performance on a validation dataset starts to degrade
- **Weight regularization**
  - Penalizing a network based on the size of the network weights during training
  - E.g., L1 or L2 vector norm penalty



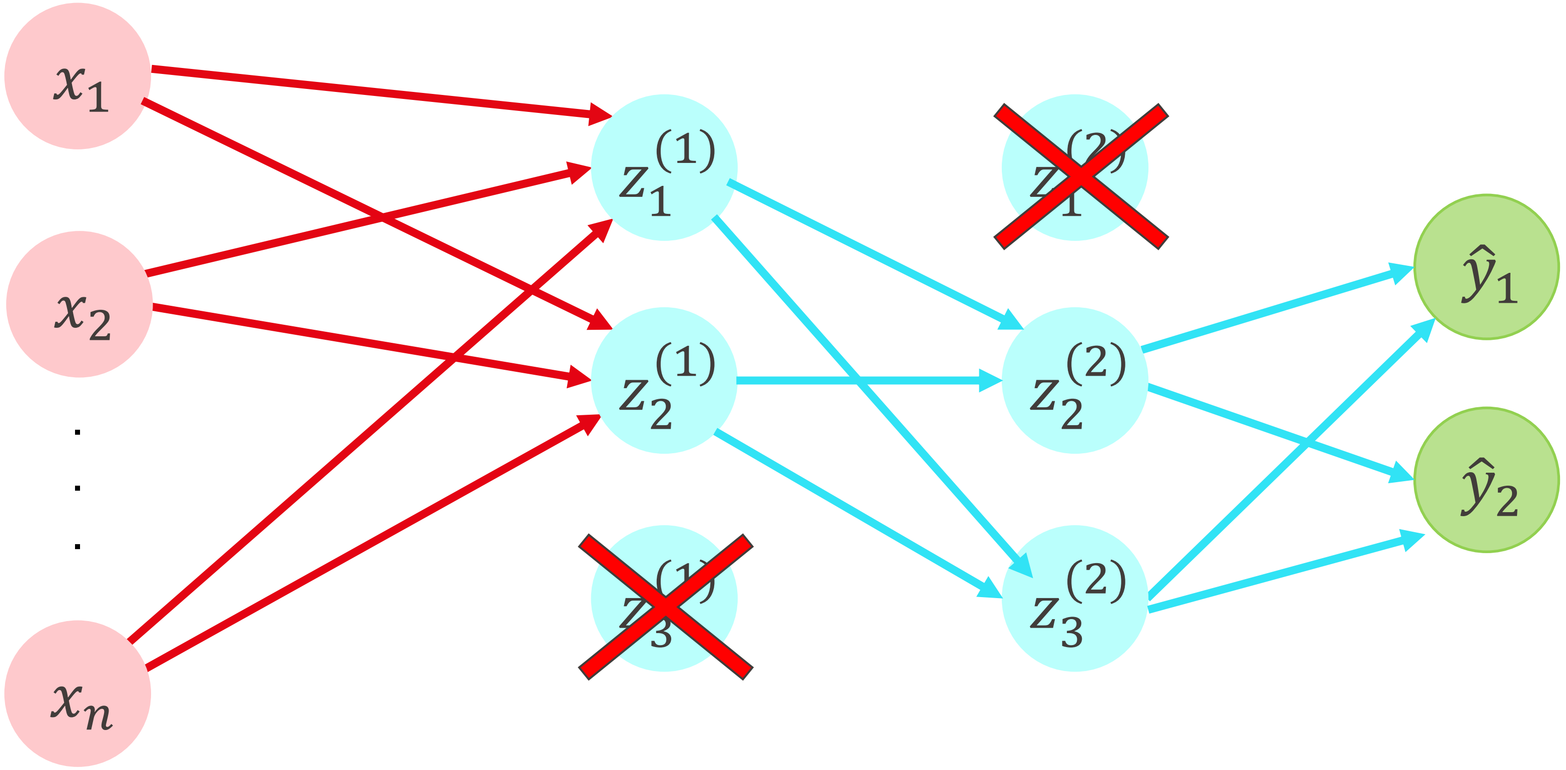
Regularization Techniques

- **Dropout**
  - Randomly drop out neurons during training with some probability



Regularization Techniques

- **Dropout**
  - Randomly drop out neurons during training with some probability



Regularization Techniques

More details in exercise session!

- **Batch normalization**
  - More than “just regularization”
  - Allows higher learning rates
  - Reduces the strong dependence on initialization
  - Usually applied after a fully connected/convolutional layer and before a non-linearity

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

hyperparameters

mean and standard-deviation calculated over the mini-batches

Hyperparameter Optimization

- **Hyperparameters**: Parameters used to control the learning the learning process; they are not learnt
  - E.g,, learning rate, number of neural network layers, mini-batch size
- How to find hyperparameters:
  - Grid search, random search, Bayesian optimization
- **Cross-validation** is often used to estimate generalization performance of the model
  - Partition the data in non-overlapping subsets of train, validation and test data and evaluate model's performance on the held-out data

EPFL

Recap

- **Objective function:**

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$$

- Sample a minibatch of input $\boldsymbol{x}$
- **Forward propagation:** compute $\mathcal{L}$ given $\boldsymbol{x}$
- **Back-propagation:** obtain gradient $\nabla_{\Theta}\mathcal{L}$ using a chain rule
- Use **stochastic gradient descent (SGD)** to optimize for Θ over many iterations
- f can be any neural network

Any Feedback?

Give us feedback on the lecture:

- https://docs.google.com/forms/d/e/1FAIpQLSdgn11OkIAvx9iIKLpviRwJ9gruCdLGUREFWPIpYR0QycY6jA/viewform

Next lectures: Different neural network architectures

- **CNNs** for images
- **GNNs** for graphs
- **Transformers** for sequence data