Brief Introduction to ML and Applications of ML to (Quantum) Physics

Spring School on Theoretical and Computational Foundations of Quantum Technologies

(24 - 28 October 2023, Alpine Heath Resort, Northern Drakensberg)

Part A: Brief Introduction to ML

Spring School on Theoretical and Computational Foundations of Quantum Technologies

(24 - 28 October 2023, Alpine Heath Resort, Northern Drakensberg)

ML part is based on Lecture Slides CIS 4190/5190: Applied Machine Learning, Penn Engineering, University of Pennsylvania

Outline of ML part:

1. **Introduction**
2. Linear Regression
3. LR: Loss functions
4. LR: feature maps
5. LR: Overfitting vs Underfitting
6. LR: Gradient Decent
7. NN: Basic Structure
8. NN: Backpropagation
9. NN: Tips/Tricks/Problems

What is Machine Learning?

"Learning is any process by which a system improves performance from experience."

Herbert Simon

What is Machine Learning?

“Machine learning … gives computers the ability to learn without being explicitly programmed.”

Arthur Samuel

What is Machine Learning?

- **Tom Mitchell:** Algorithms that
  - improve their **performance** P
  - at **task** T
  - with **experience** E

- A well-defined machine learning task is given by (P, T, E)

Example: Game Playing

- **Tom Mitchell:** Algorithms that
  - improve their **performance** P
  - at **task** T
  - with **experience** E

- $T =$ playing Checkers
- $P =$ win rate against opponents
- $E =$ playing games against itself

Example: Prediction

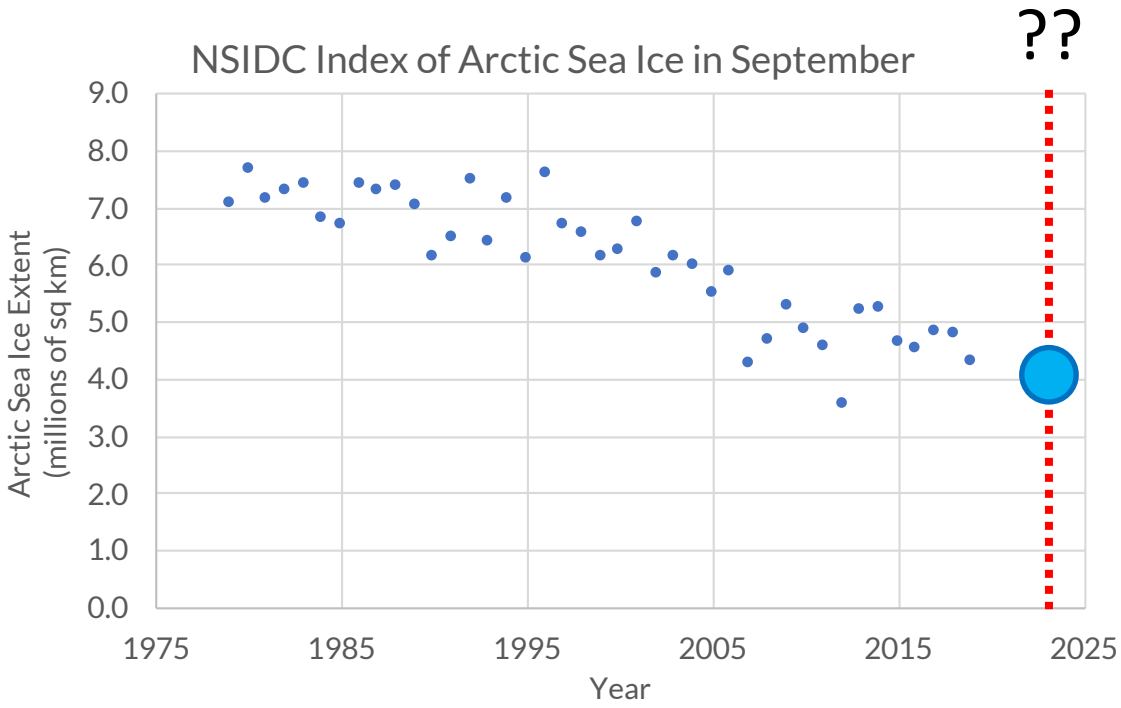
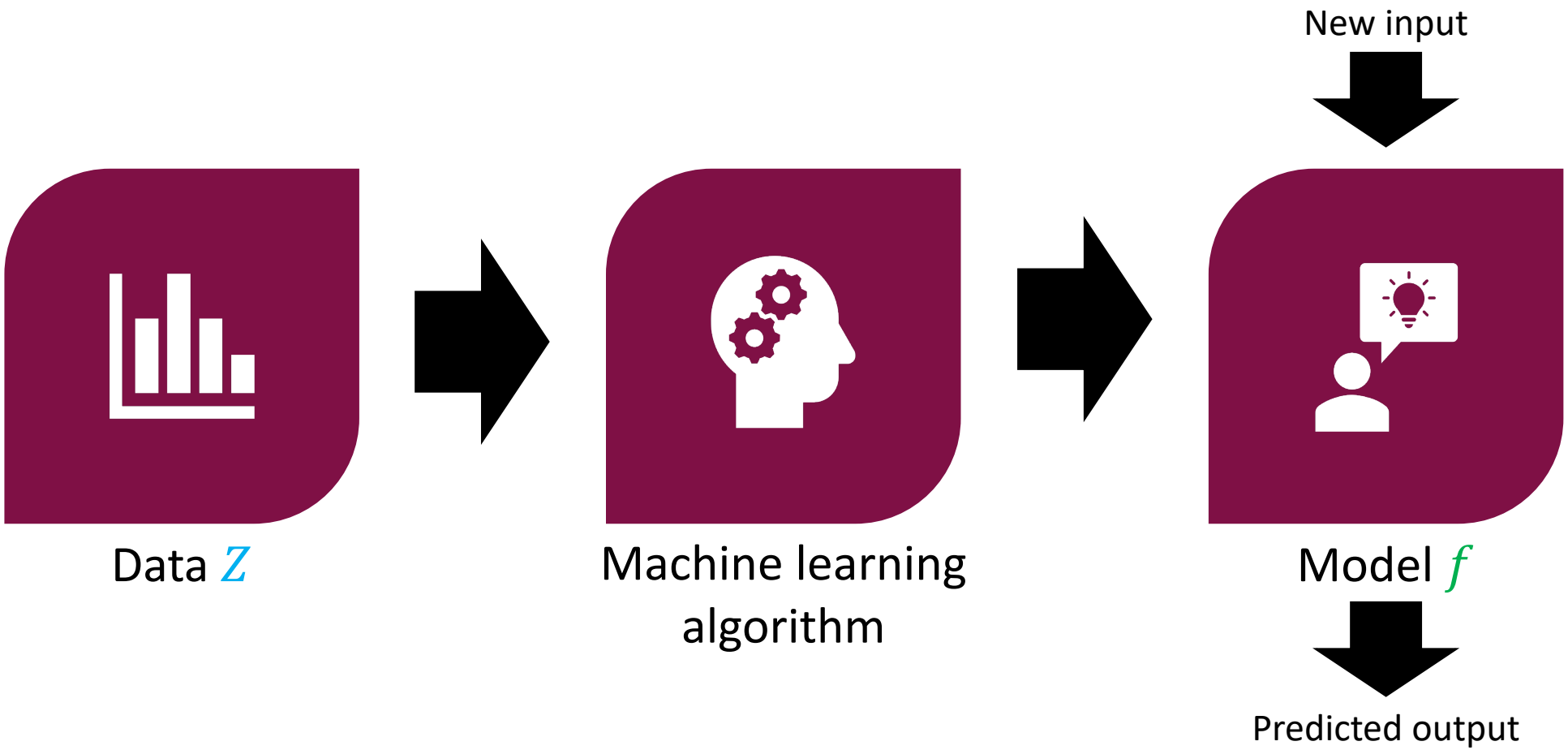


Image: https://www.flickr.com/photos/gsfc/5937599688/
Data from https://nsidc.org/arcticseaicenews/sea-ice-tools/

Outline of ML part:

1. Introduction
2. **Linear Regression**
3. LR: Loss functions
4. LR: feature maps
5. LR: Overfitting vs Underfitting
6. LR: Gradient Decent
7. NN: Basic Structure
8. NN: Backpropagation
9. NN: Tips/Tricks/Problems

Machine Learning for Prediction



Types of Learning

- **Supervised learning**
  - **Input:** Examples of inputs and outputs
  - **Output:** Model that predicts unknown output given a new input

- **Unsupervised learning**
  - **Input:** Examples of some data (no “outputs”)
  - **Output:** Representation of structure in the data

- **Reinforcement learning**
  - **Input:** Sequence of interactions with an environment
  - **Output:** Policy that performs a desired task

Supervised Learning

- Given $(x_1, y_1), \ldots, (x_n, y_n)$, learn a function that predicts y given x
- **Regression:** Labels y are real-valued

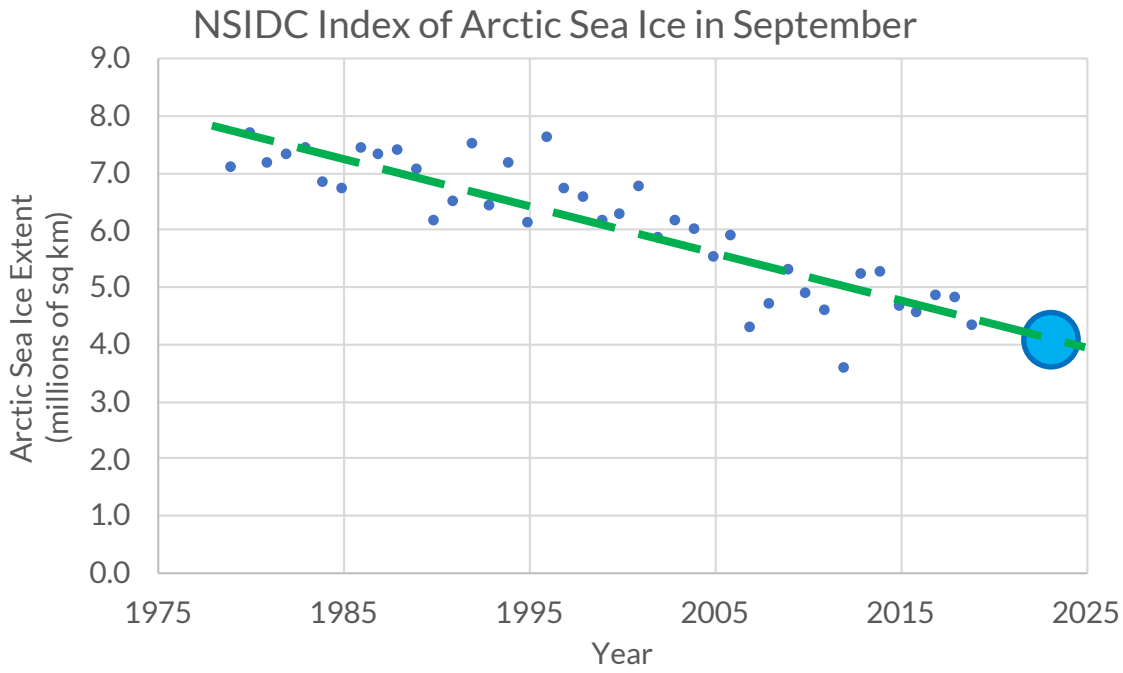
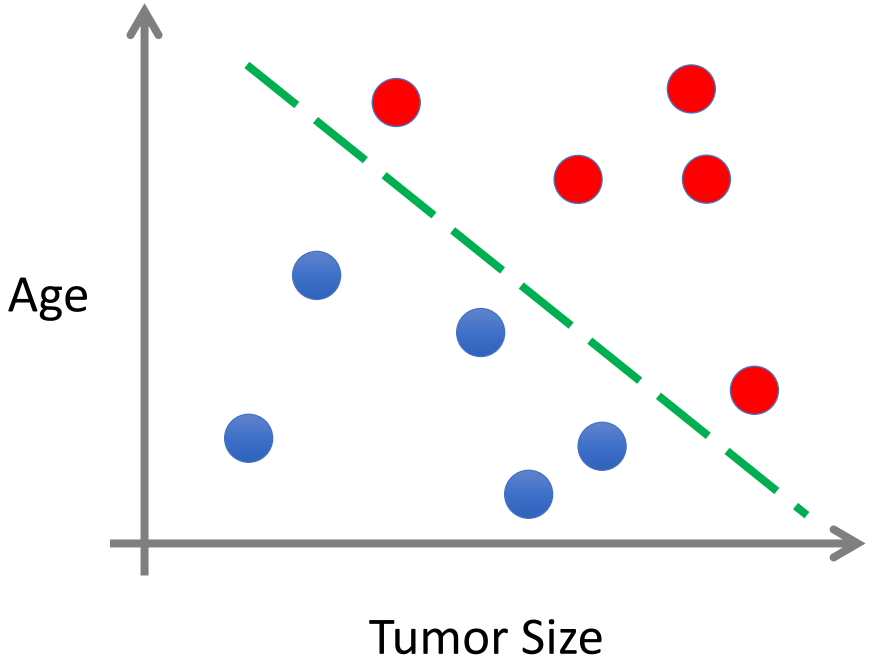


Image: https://www.flickr.com/photos/gsfc/5937599688/
Data from https://nsidc.org/arcticseaicenews/sea-ice-tools/

Supervised Learning

- Given $(x_1, y_1), \ldots, (x_n, y_n)$, learn a function that predicts y given x
- Inputs x can be multi-dimensional



- Patient age
- Clump thickness
- Tumor Color
- Cell type
- …

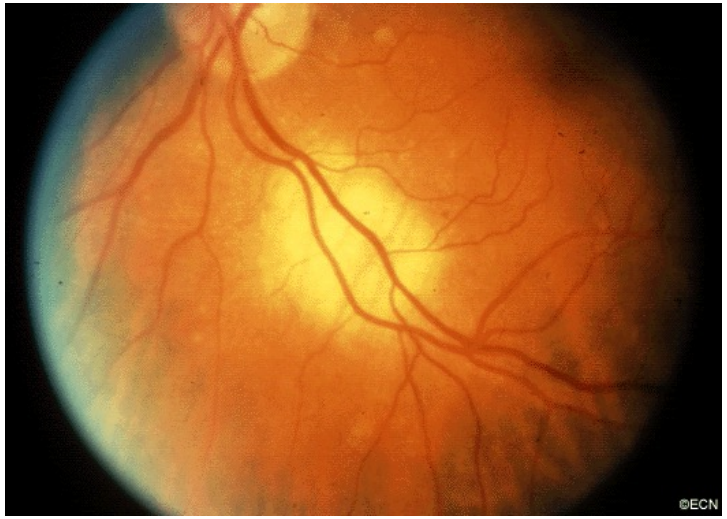
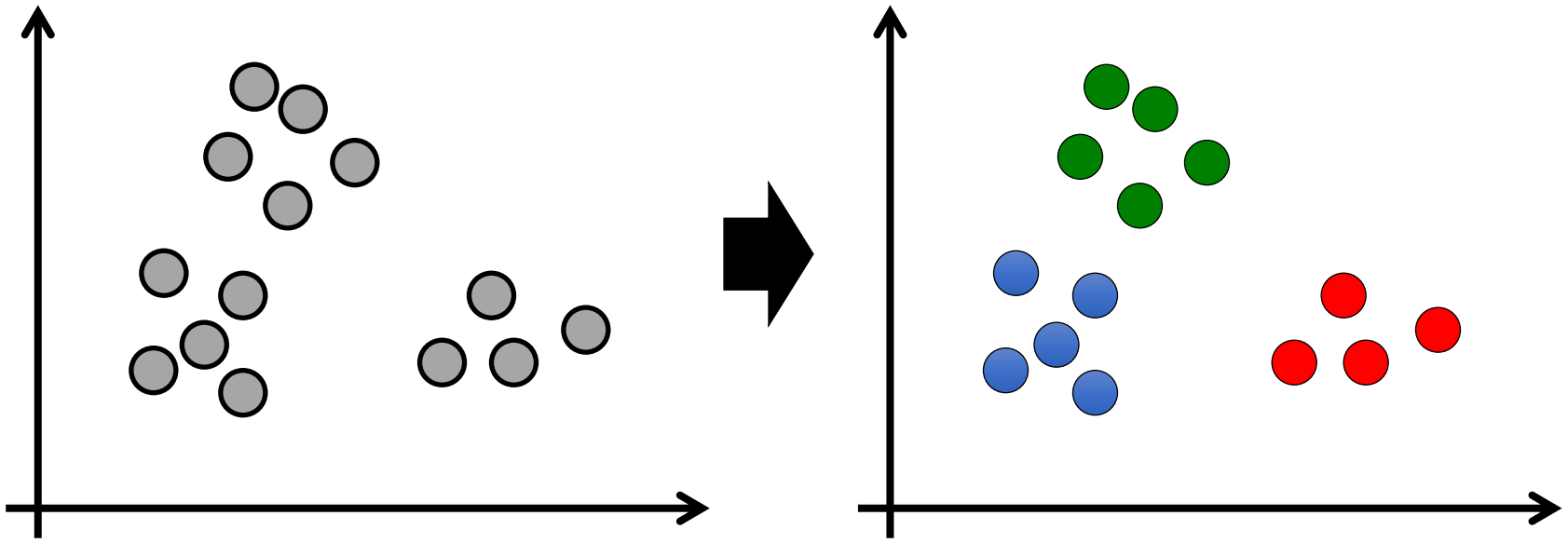


Image: https://eyecancer.com/uncategorized/choroidal-metastasis-test/

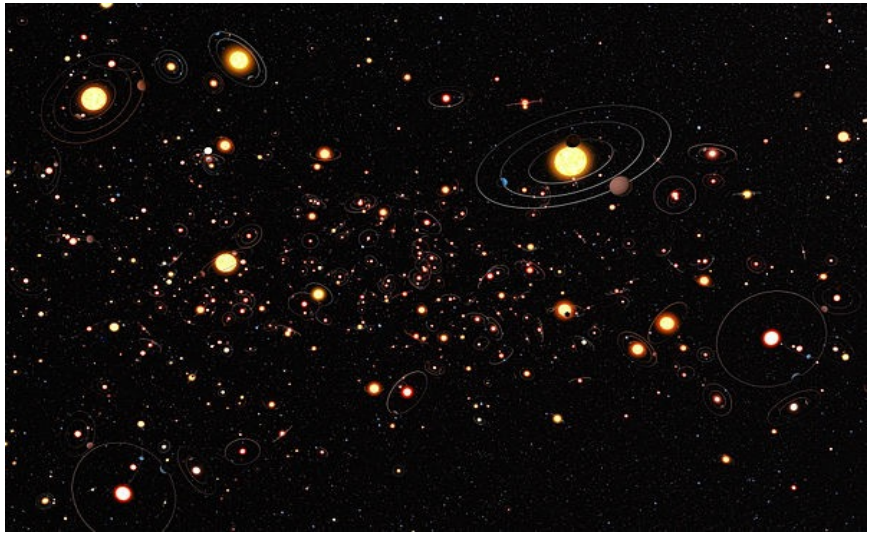
Unsupervised Learning

- Given $x_1, \ldots, x_n$ (no labels), output hidden structure in x's
  - E.g., clustering

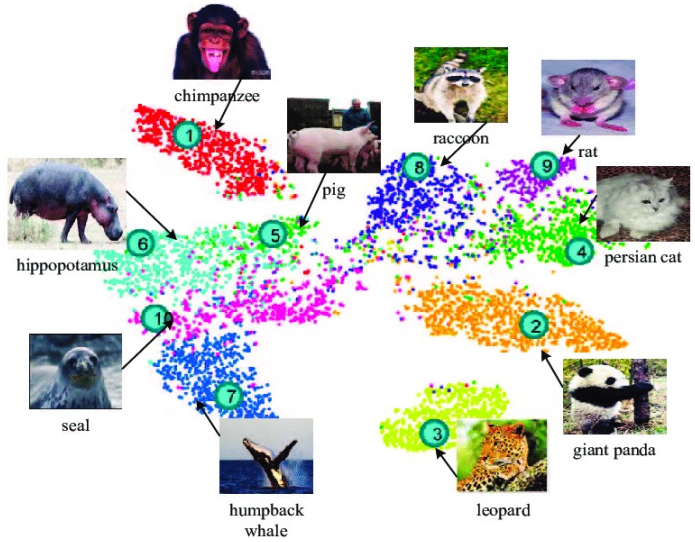


Unsupervised Learning

Find Subgroups in Social Networks



Identify Types of Exoplanets



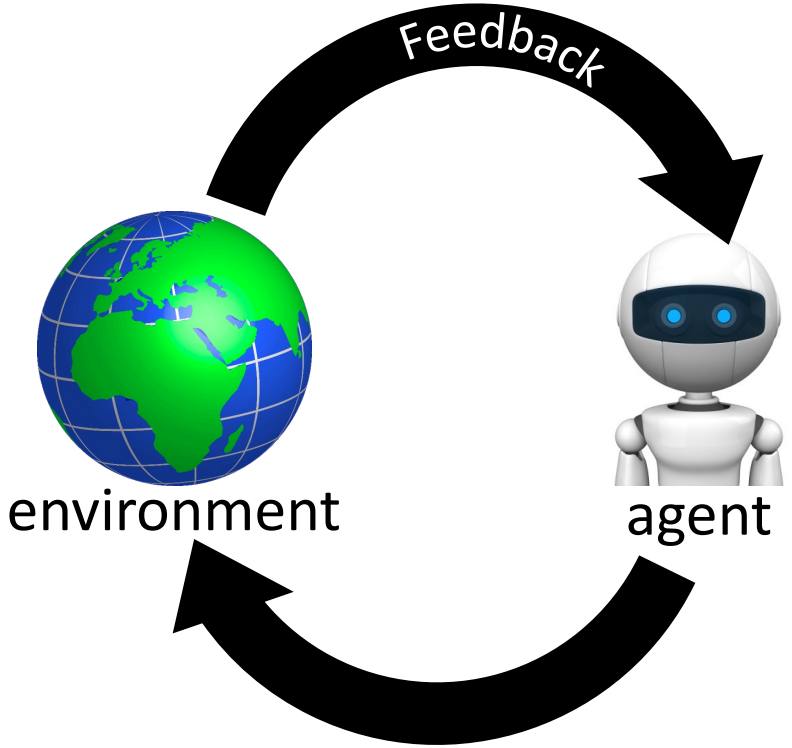
Visualize Data

Image Credits:
https://medium.com/graph-commons/finding-organic-clusters-in-your-complex-data-networks-5c27e1d4645d
https://arxiv.org/pdf/1703.08893.pdf
https://en.wikipedia.org/wiki/Exoplanet

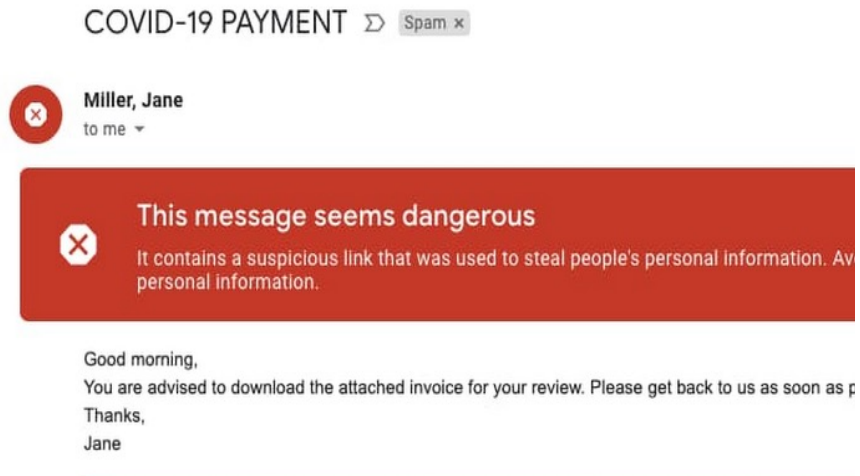
Reinforcement Learning

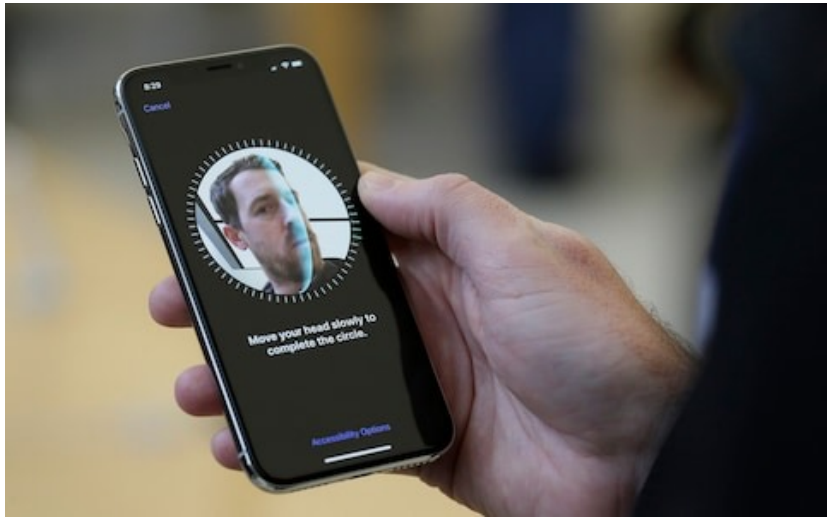
- Learn how to perform a task from interactions with the **environment**

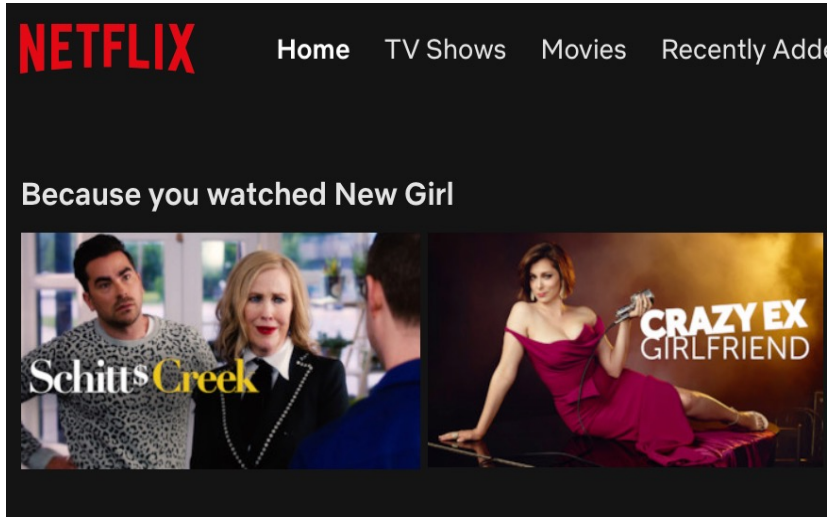
- **Examples:**
  - Playing chess (interact with the game)
  - Robot grasping an object (interact with the object/real world)
  - Optimize inventory allocations (interact with the inventory system)



Everyday Applications

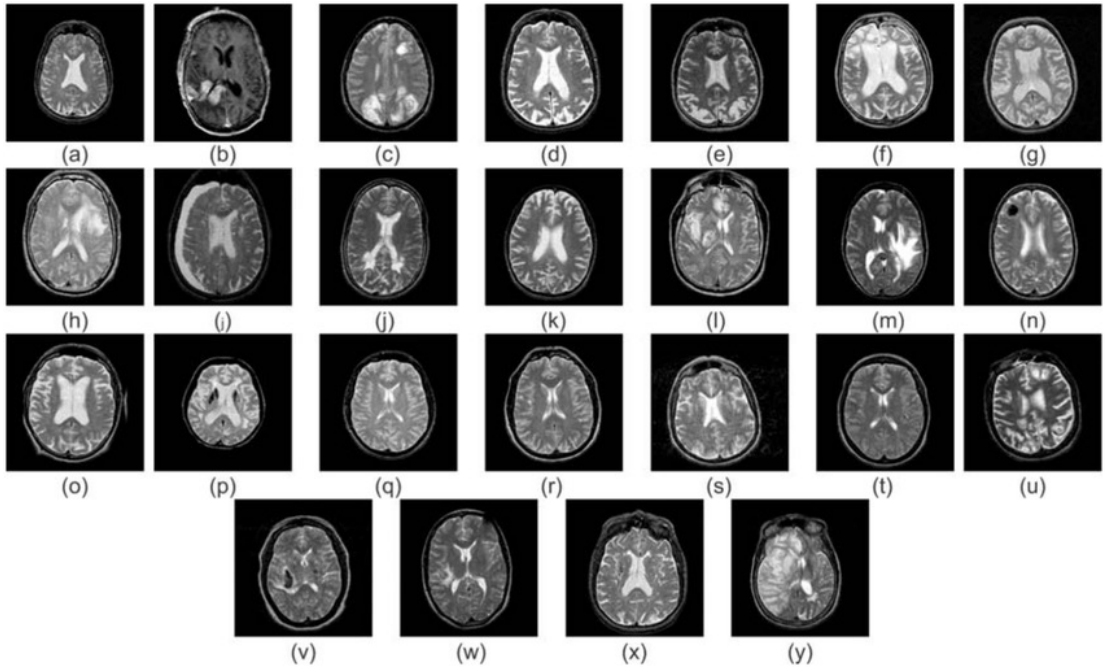






Radiology and Medicine

Input: Brain scans



Output: Neurological disease labels

Machine learning studies on major brain diseases: 5-year trends of 2014–2018

Koji Sakai[1] · Kei Yamada[1]

Applications of machine learning in drug discovery and development

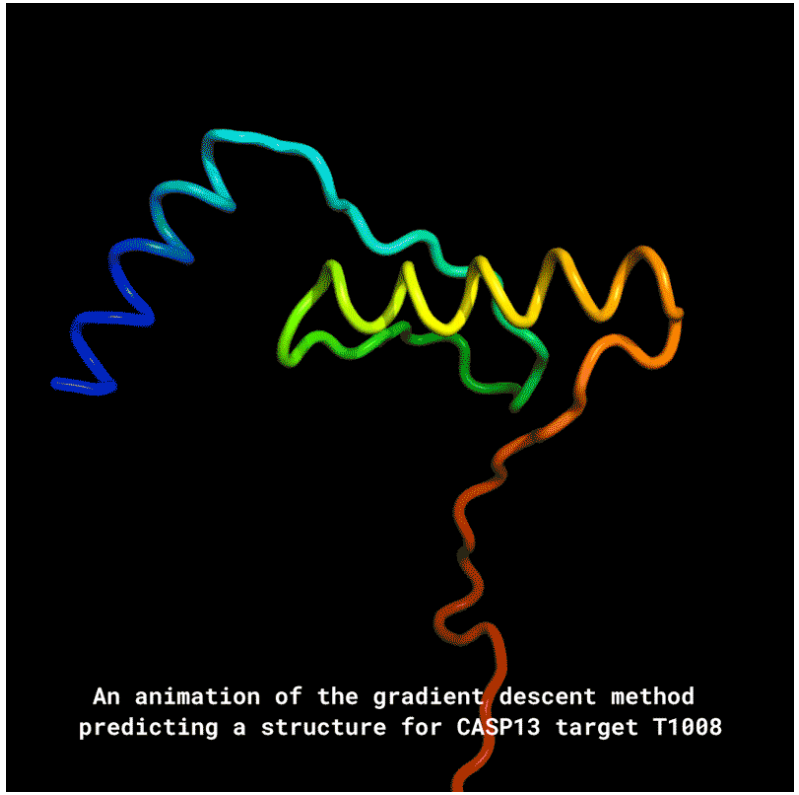
https://www.nature.com/articles/s41573-019-0024-5

Deep learning-enabled medical computer vision

Andre Esteva ✉, Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean & Richard Socher

https://www.nature.com/articles/s41746-020-00376-2

Scientific Discovery



https://deepmind.com/blog/article/AlphaFold-Using-AI-for-scientific-discovery

https://www.jpl.nasa.gov/edu/news/2019/4/19/how-scientists-captured-the-first-image-of-a-black-hole/

Creating Images & Text

https://thispersondoesnotexist.com/

SYSTEM PROMPT (HUMAN-WRITTEN)

Recycling is good for the world.

NO! YOU COULD NOT BE MORE WRONG!!

MODEL COMPLETION (MACHINE-WRITTEN, 25 TRIES)

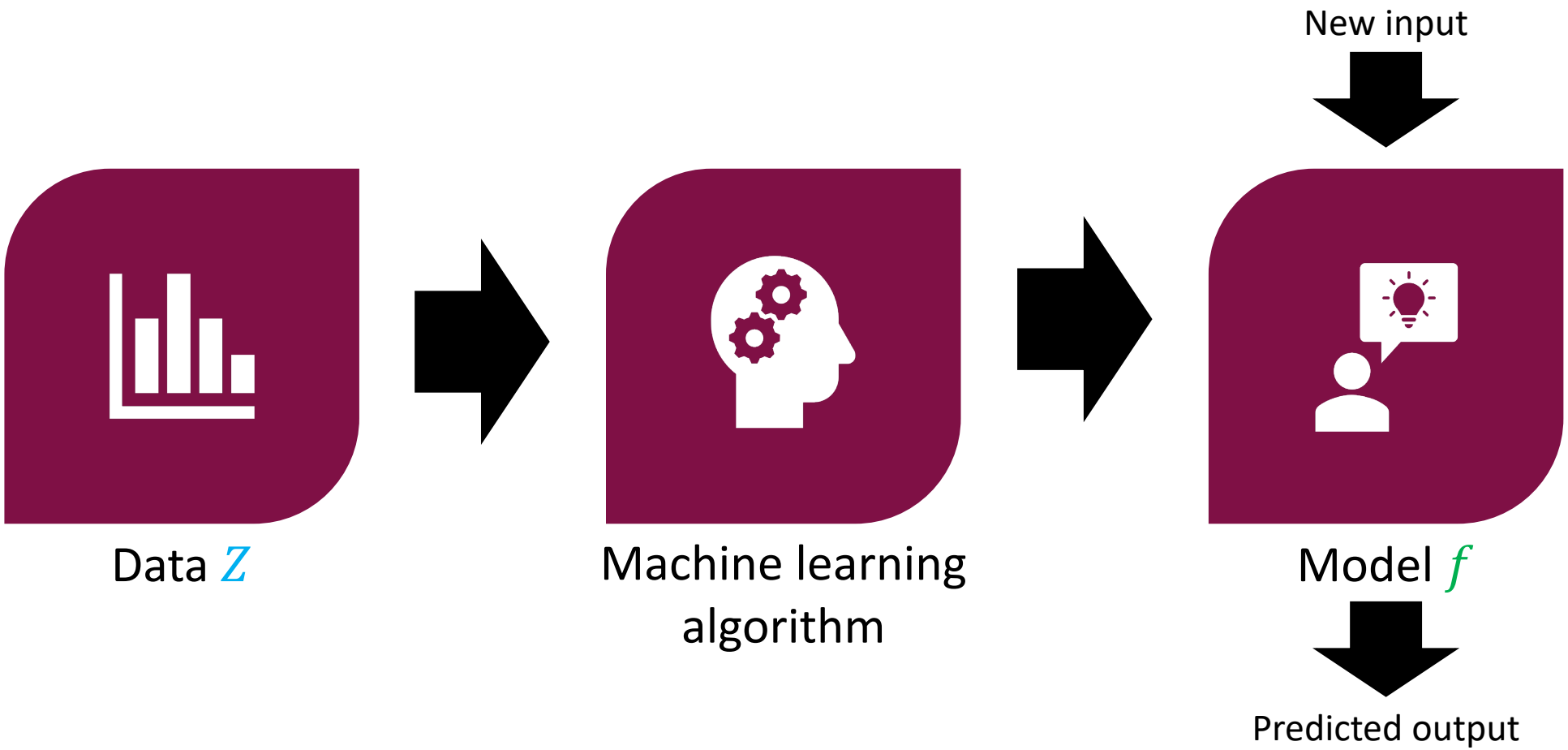
Recycling is NOT good for the world. It is bad for the environment, it is bad for our health, and it is bad for our economy. I'm not kidding. Recycling is not good for the environment. It is destructive to the earth and it is a major contributor to global warming. Recycling is not good for our health. It contributes to obesity and diseases like heart disease and cancer. Recycling is bad for our economy. It increases the cost of a product, and in turn, the price of everything that is made with that product. Recycling is not good for our nation. We pay a

https://transformer.huggingface.co/doc/gpt2-large

Outline of ML part:

1. Introduction
2. **Linear Regression**
3. LR: Loss functions
4. LR: feature maps
5. LR: Overfitting vs Underfitting
6. LR: Gradient Decent
7. NN: Basic Structure
8. NN: Backpropagation
9. NN: Tips/Tricks/Problems

Machine Learning for Prediction



Question: What **model family** (a.k.a. **hypothesis class**) to consider?

Linear Functions

- Consider the space of linear functions $f_\beta(x)$ defined by

$$f_\beta(x) = \beta^\top x = [\beta_1 \quad \cdots \quad \beta_d] \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} = \beta_1 x_1 + \cdots + \beta_d x_d$$

- $x \in \mathbb{R}^d$ is called an **input** (a.k.a. **features** or **covariates**)
- $\beta \in \mathbb{R}^d$ is called the **parameters** (a.k.a. **parameter vector**)
- $y = f_\beta(x)$ is called the **label** (a.k.a. **output** or **response**)

Linear Regression Problem

What does this mean?

- **Input:** Data $Z = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- **Output:** A linear function $f_\beta(x) = \beta^\top x$ such that $y_i \approx \beta^\top x_i$

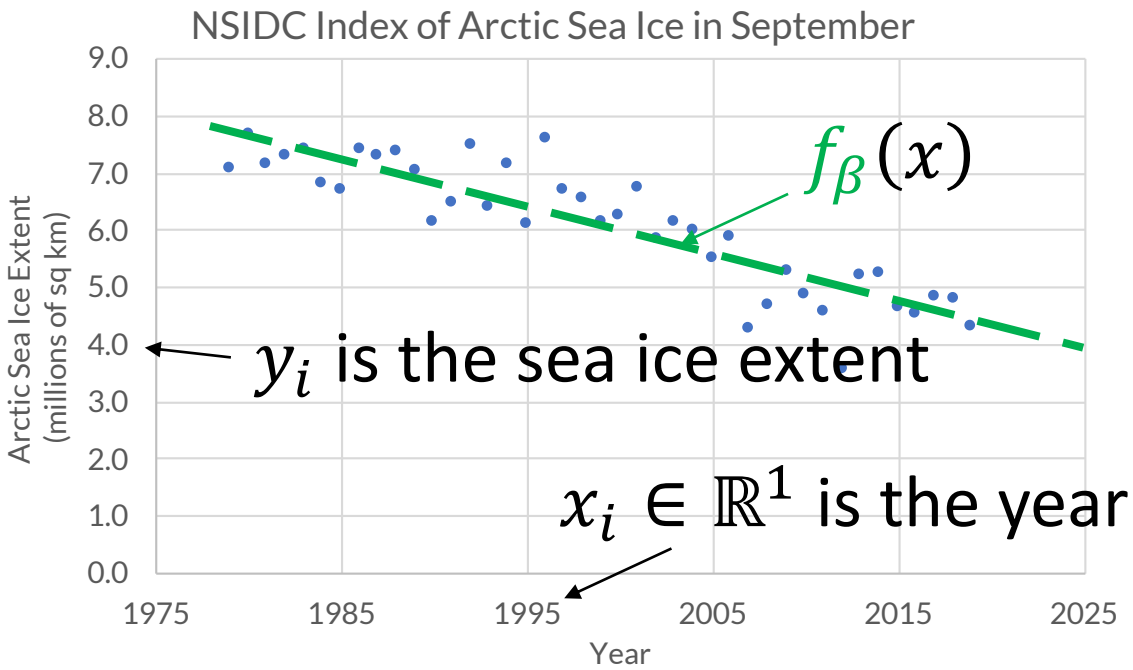


Image: https://www.flickr.com/photos/gsfc/5937599688/
Data from https://nsidc.org/arcticseaicenews/sea-ice-tools/

Outline of ML part:

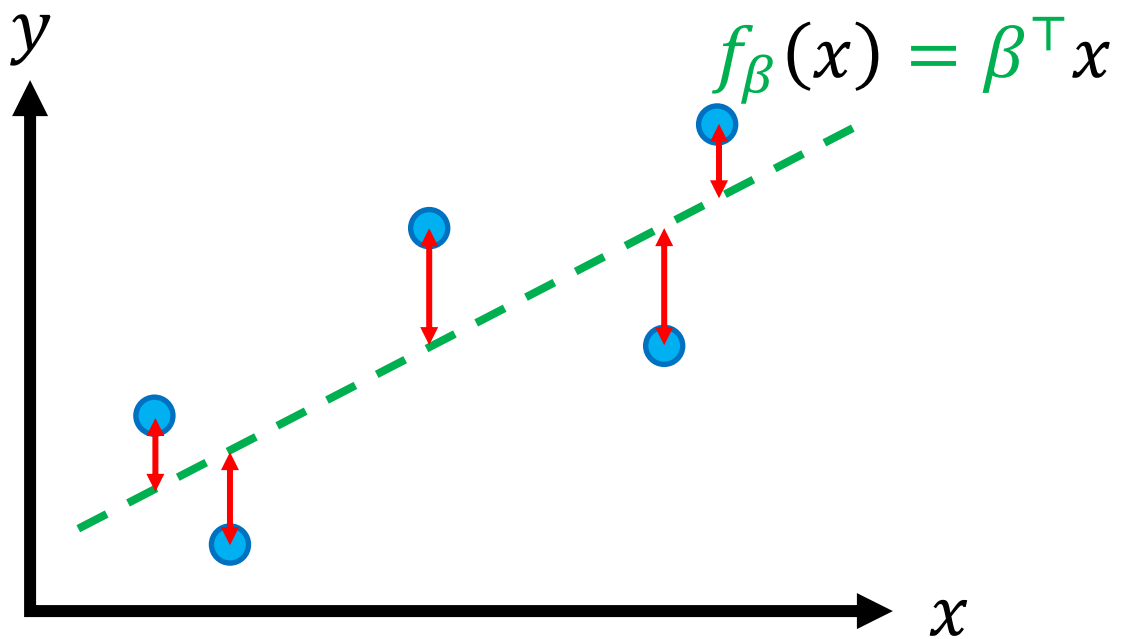
1. Introduction
2. Linear Regression
3. **LR: Loss functions**
4. LR: feature maps
5. LR: Overfitting vs Underfitting
6. LR: Gradient Decent
7. NN: Basic Structure
8. NN: Backpropagation
9. NN: Tips/Tricks/Problems

Choice of Loss Function

- $y_i \approx \beta^\top x_i$ if $(y_i - \beta^\top x_i)^2$ small
- **Mean squared error (MSE):**

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2$$

- Computationally convenient and works well in practice



$$L(\beta; Z) = \frac{\updownarrow^2 + \updownarrow^2 + \updownarrow^2 + \updownarrow^2 + \updownarrow^2}{n}$$

Linear Regression Algorithm

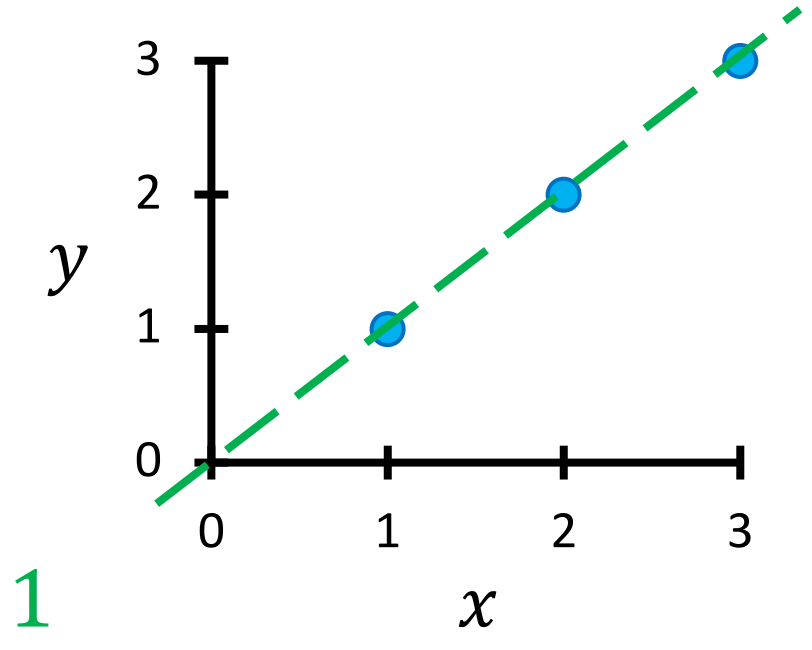
- **Input:** Dataset $Z = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Compute

$$\hat{\beta}(Z) = \arg\min_{\beta \in \mathbb{R}^d} L(\beta; Z)$$
$$= \arg\min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2$$

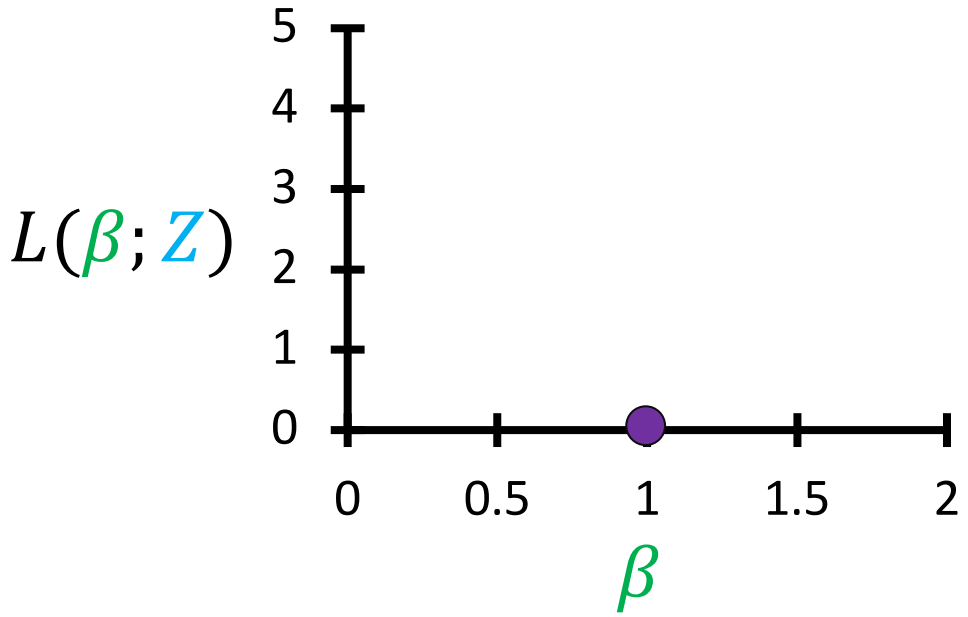
- **Output:** $f_{\hat{\beta}(Z)}(x) = \hat{\beta}(Z)^\top x$
- Discuss algorithm for computing the minimal β later

Intuition on Minimizing MSE Loss

- Consider $x \in \mathbb{R}$ and $\beta \in \mathbb{R}$

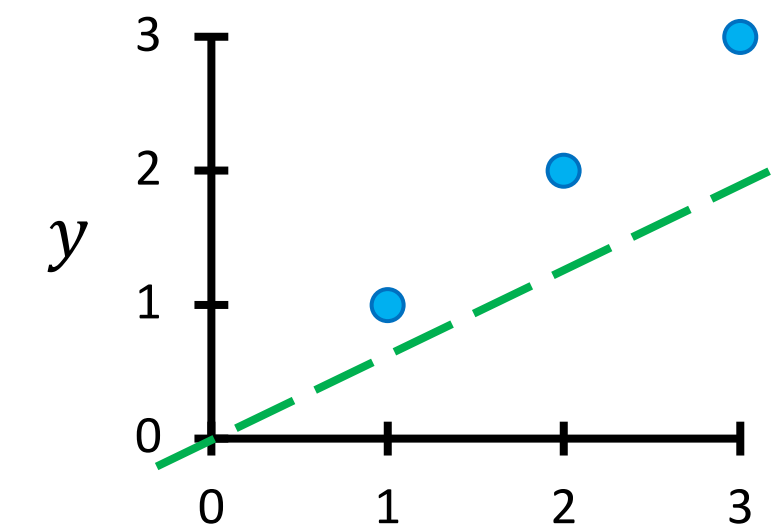


$\beta = 1$

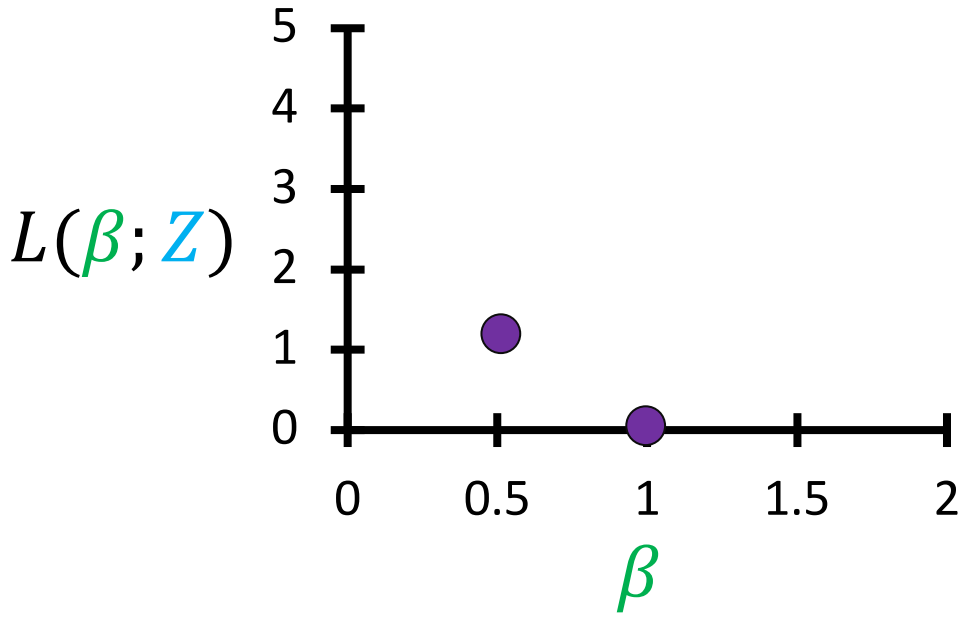


Intuition on Minimizing MSE Loss

- Consider $x \in \mathbb{R}$ and $\beta \in \mathbb{R}$

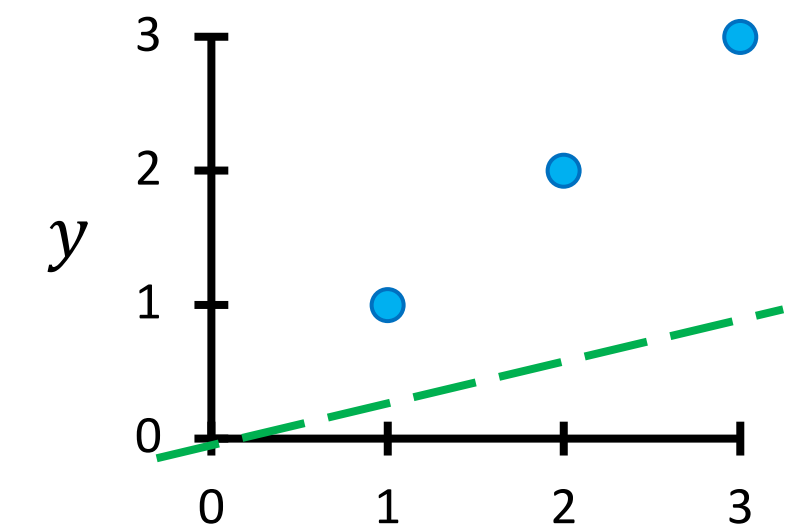


$\beta = 0.5$

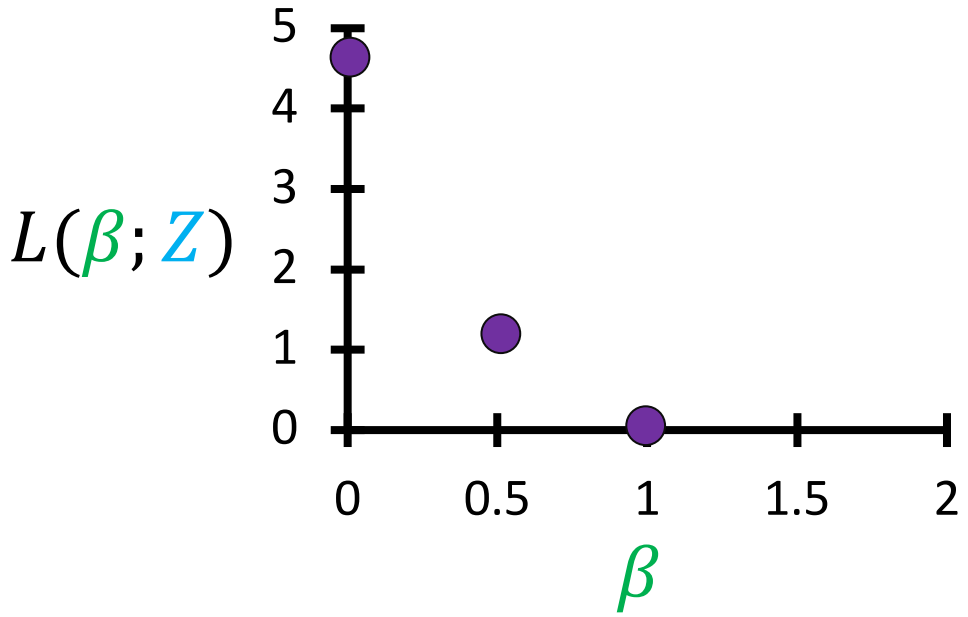


Intuition on Minimizing MSE Loss

- Consider $x \in \mathbb{R}$ and $\beta \in \mathbb{R}$

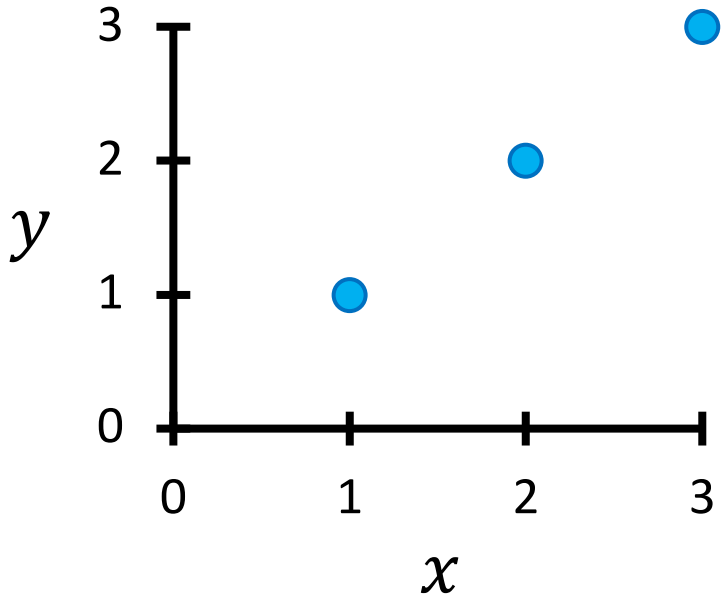


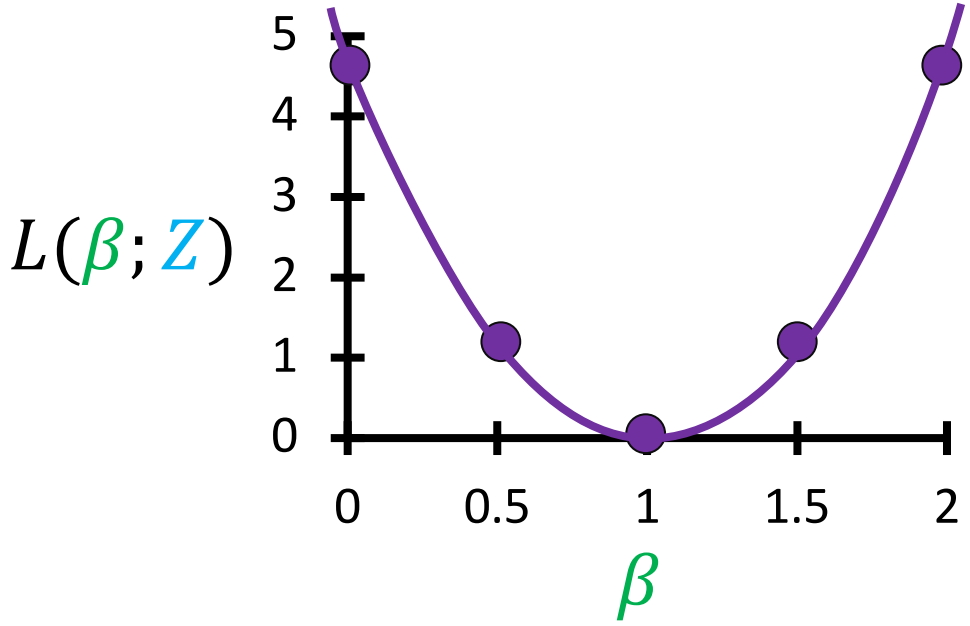
$\beta = 0.25$



Intuition on Minimizing MSE Loss

- Consider $x \in \mathbb{R}$ and $\beta \in \mathbb{R}$

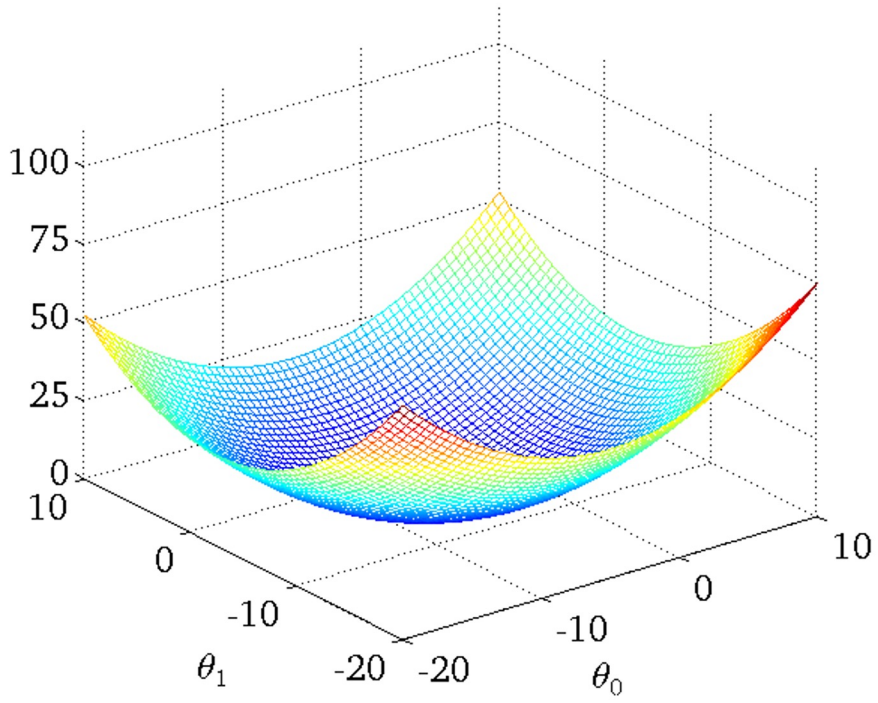


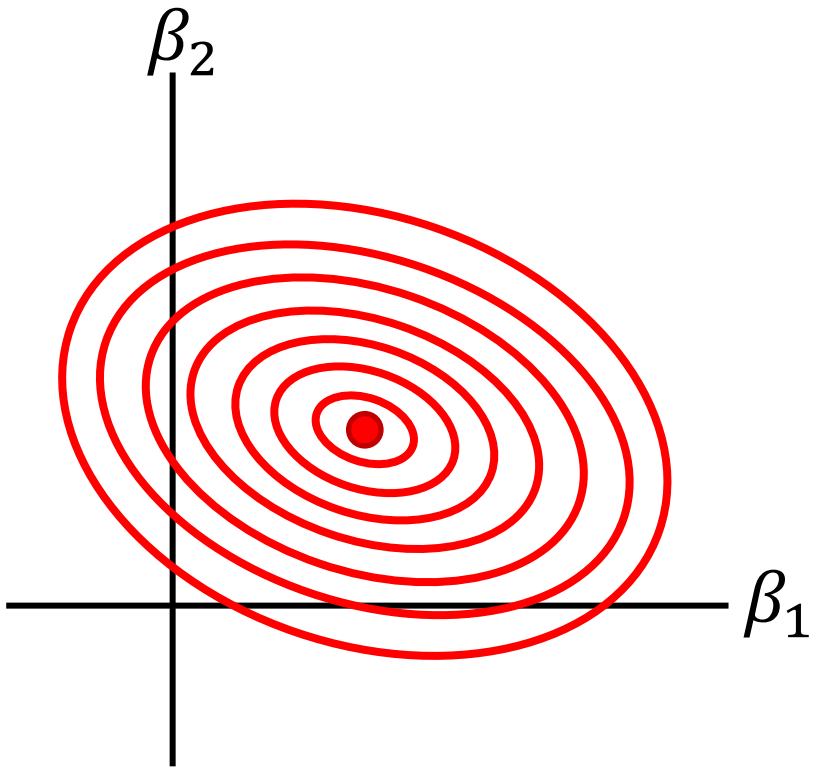


Intuition on Minimizing MSE Loss

- **Convex** (“bowl shaped”) in general

$L(\beta; Z)$





Slide by Andrew Ng

Alternative Loss Functions

- **Mean absolute error:** $\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$
- **Mean relative error:** $\frac{1}{n}\sum_{i=1}^{n}\frac{|\widehat{y_i}-y_i|}{|y_i|}$
- $\boldsymbol{R^2}$ **score:** $1 - \frac{\text{MSE}}{\text{Variance}}$
  - "Coefficient of determination"
  - Higher is better, $R^2 = 1$ is perfect

Outline of ML part:

1. Introduction
2. Linear Regression
3. LR: Loss functions
4. **LR: feature maps**
5. LR: Overfitting vs Underfitting
6. LR: Gradient Decent
7. NN: Basic Structure
8. NN: Backpropagation
9. NN: Tips/Tricks/Problems

Linear Regression

General strategy

- Model family $F = \{f_\beta\}_\beta$
- Loss function $L(\beta; Z)$

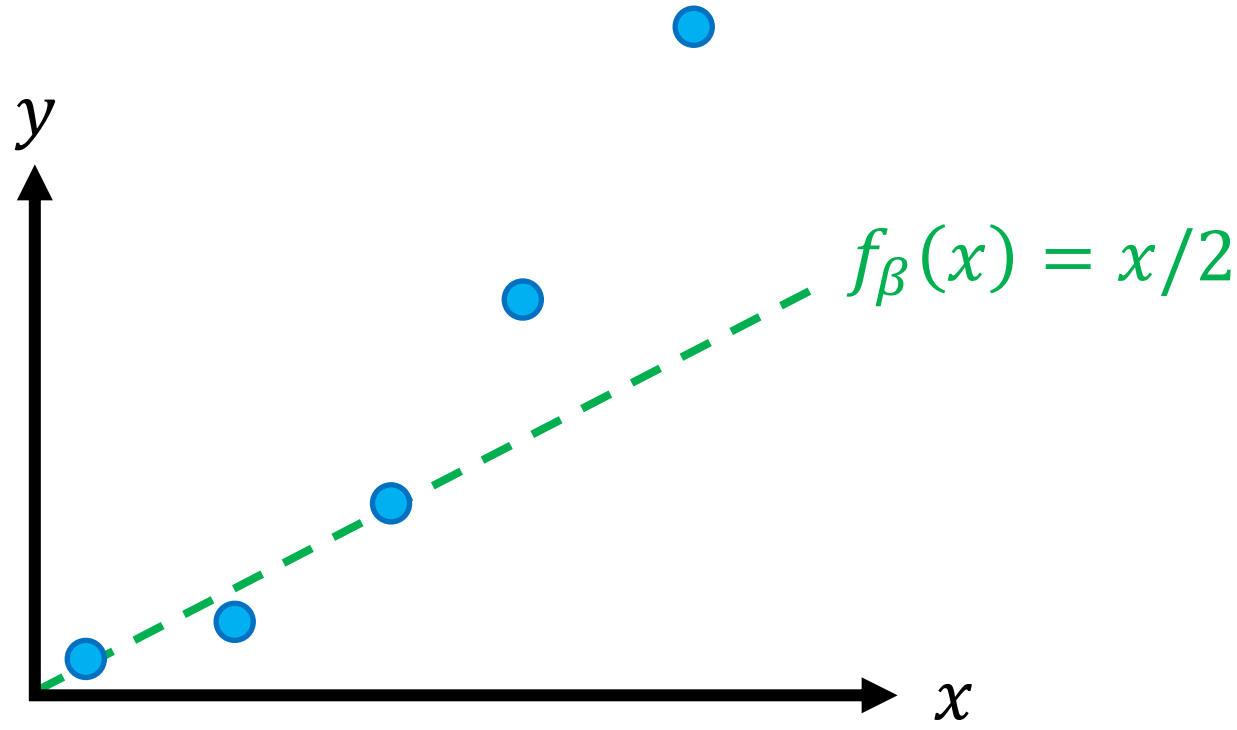
Linear regression strategy

- Linear functions $F = \{f_\beta(x) = \beta^\top x\}$
- MSE $L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2$

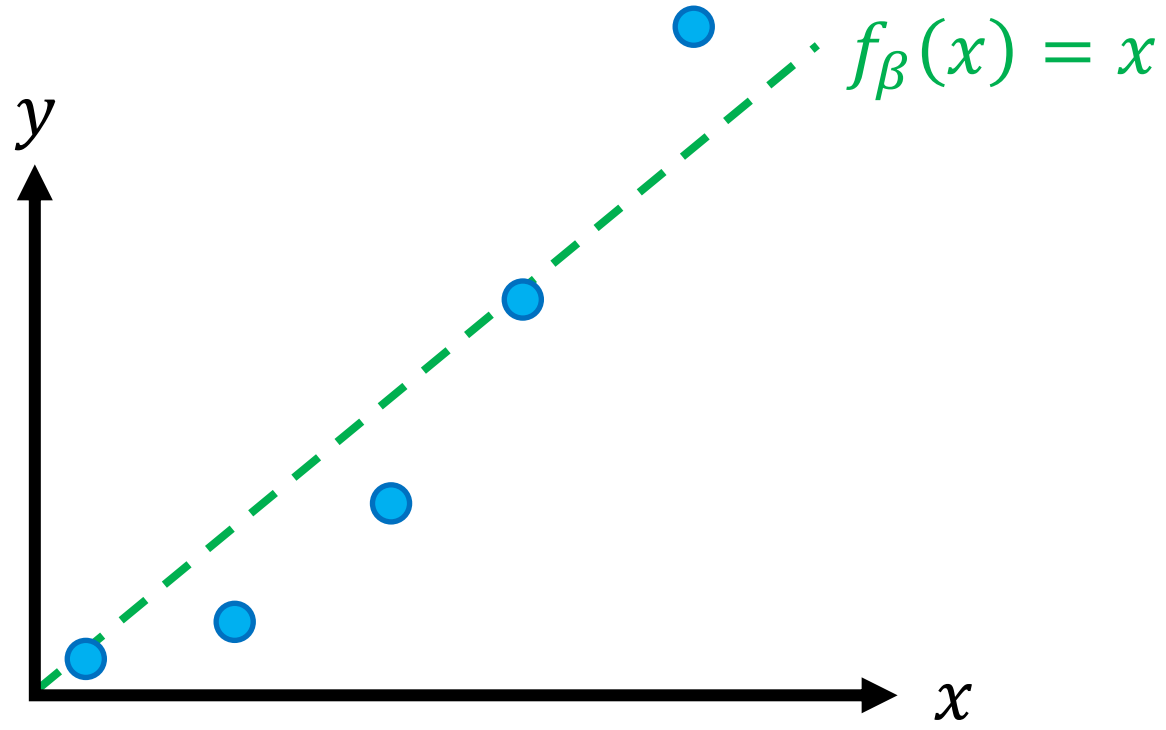
Linear regression algorithm

$$\hat{\beta}(Z) = \arg\min_\beta L(\beta; Z)$$

Example: Quadratic Function



Example: Quadratic Function



Can we get a better fit?

Feature Maps

General strategy

- Model family $F = \{f_\beta\}_\beta$
- Loss function $L(\beta; Z)$

Linear regression with feature map

- Linear functions over a given **feature map** $\phi: X \to \mathbb{R}^d$

$$F = \{f_\beta(x) = \beta^\top \phi(x)\}$$

- MSE $L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta^\top \phi(x_i)\right)^2$

Quadratic Feature Map

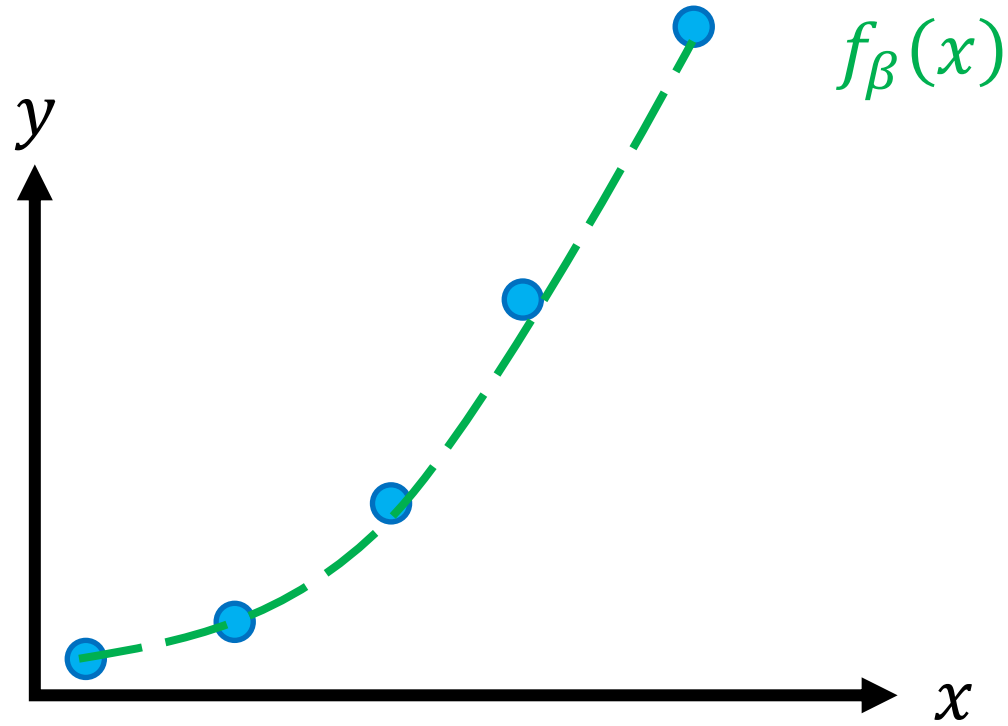
- Consider the feature map $\phi: \mathbb{R} \to \mathbb{R}^2$ given by

$$\phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$$

- Then, the model family is

$$f_\beta(x) = \beta_1 x + \beta_2 x^2$$

Quadratic Feature Map



In our family for $\beta = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$!

Feature Maps

- Powerful strategy for encoding prior knowledge

- **Terminology**
  - x is the **input** and $\phi(x)$ are the **features**
  - Often used interchangeably

Examples of Feature Maps

- **Polynomial features**
  - $\phi(x) = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_1^2 + \beta_5 x_1 x_2 + \beta_6 x_2^2 + \cdots$
  - Quadratic features are very common; capture "feature interactions"
  - Can use other nonlinearities (exponential, logarithm, square root, etc.)

- **Intercept term**
  - $\phi(x) = [1 \quad x_1 \quad \dots \quad x_d]^\top$
  - Almost always used; captures constant effect

- **Encoding non-real inputs**
  - E.g., $x =$ "the food was good" and $y = 4$ stars
  - $\phi(x) = [1(\text{“good”} \in x) \quad 1(\text{“bad”} \in x) \quad \dots]^\top$

Algorithm

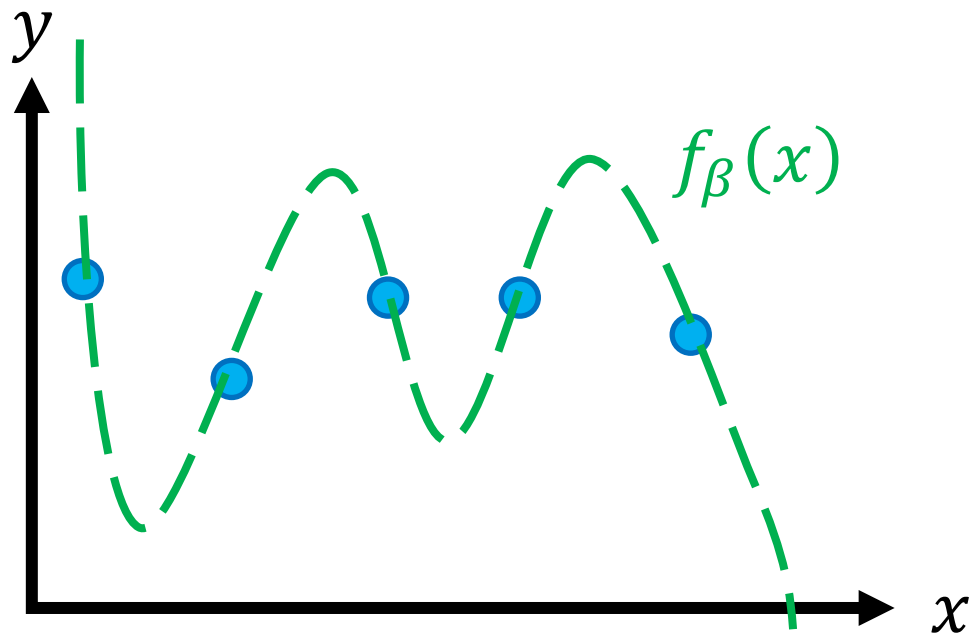
- Reduces to linear regression
- **Step 1:** Compute $\phi_i = \phi(x_i)$ for each x_i in Z
- **Step 2:** Run linear regression with $Z' = \{(\phi_1, y_1), \ldots, (\phi_n, y_n)\}$

Outline of ML part:

1. Introduction
2. Linear Regression
3. LR: Loss functions
4. LR: feature maps
5. **LR: Overfitting vs Underfitting**
6. LR: Gradient Decent
7. NN: Basic Structure
8. NN: Backpropagation
9. NN: Tips/Tricks/Problems

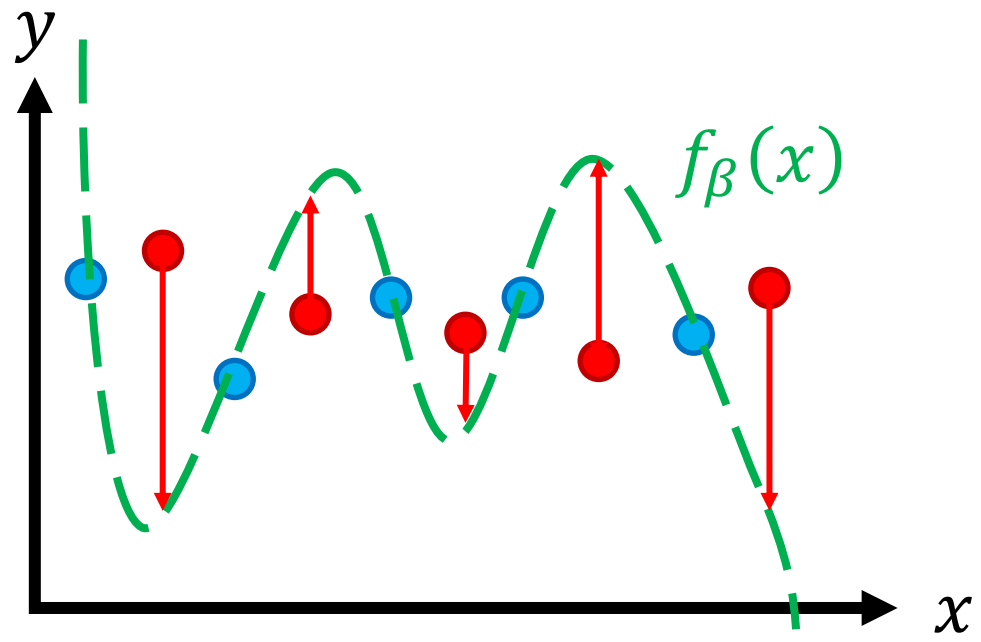
Question

- **Why not throw in lots of features?**
  - $\phi(x) = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_1^2 + \beta_5 x_1 x_2 + \beta_6 x_2^2 + \cdots$
  - Can fit any n points using a polynomial of degree n



Prediction

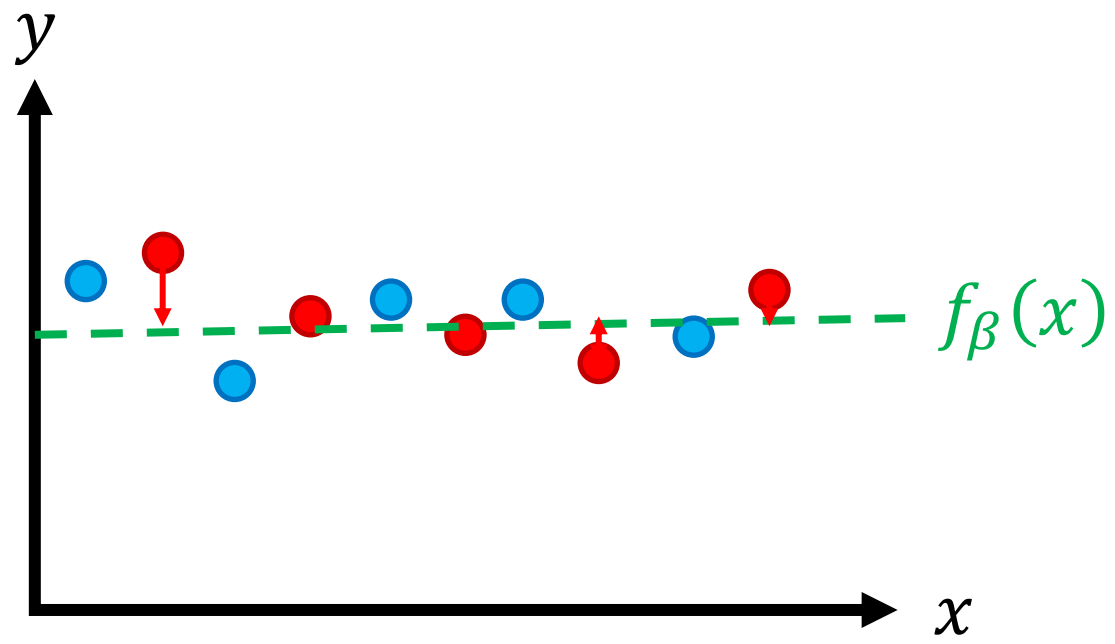
- **Issue:** The goal in machine learning is **prediction**
  - Given a **new** input x, predict the label $\hat{y} = f_\beta(x)$



The errors on new inputs is very large!

Prediction

- **Issue:** The goal in machine learning is **prediction**
  - Given a **new** input x, predict the label $\hat{y} = f_\beta(x)$



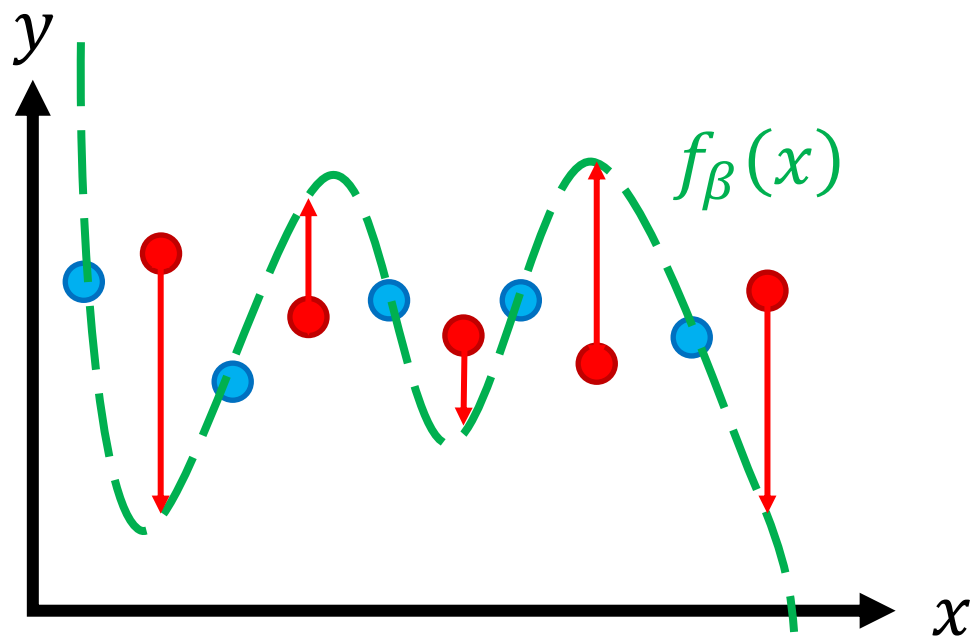
Vanilla linear regression actually works better!

Training vs. Test Data

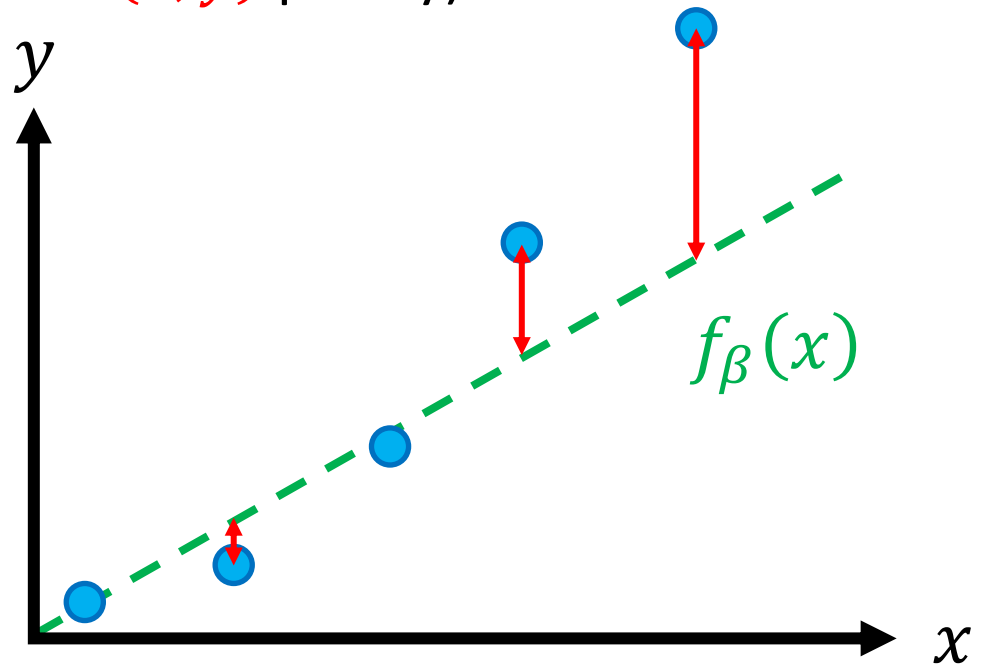
- **Training data:** Examples $Z = \{(x, y)\}$ used to fit our model
- **Test data:** New inputs x whose labels y we want to predict

Overfitting vs. Underfitting

- **Overfitting**
  - Fit the **training data** Z well
  - Fit new **test data** (x, y) poorly

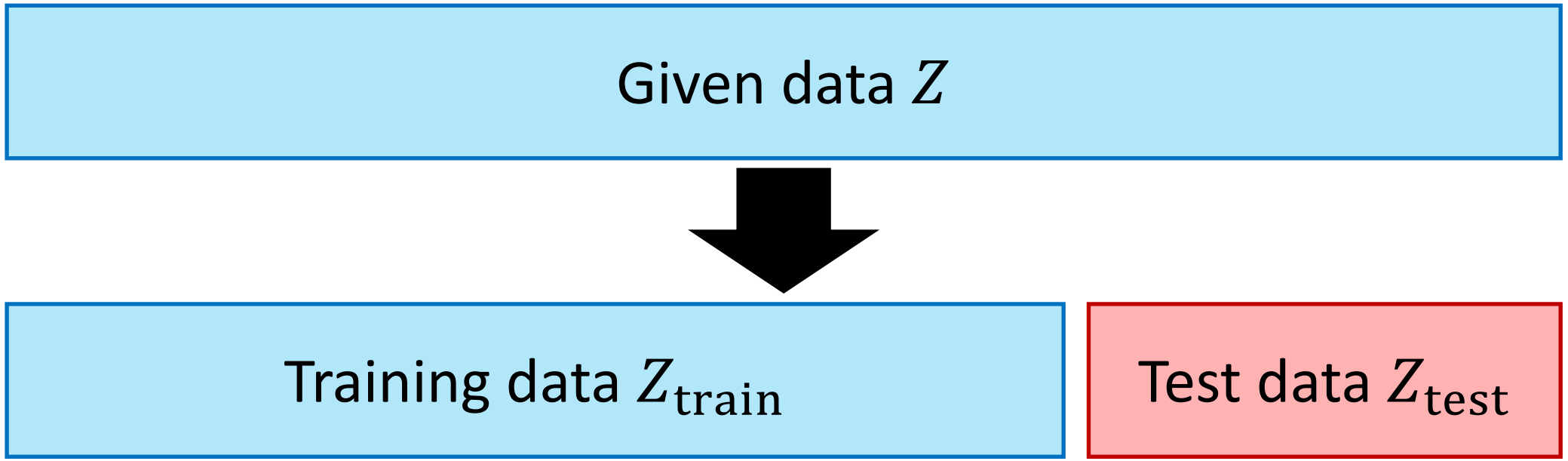


- **Underfitting**
  - Fit the **training data** Z poorly
  - (Necessarily fit new **test data** (x, y) poorly)



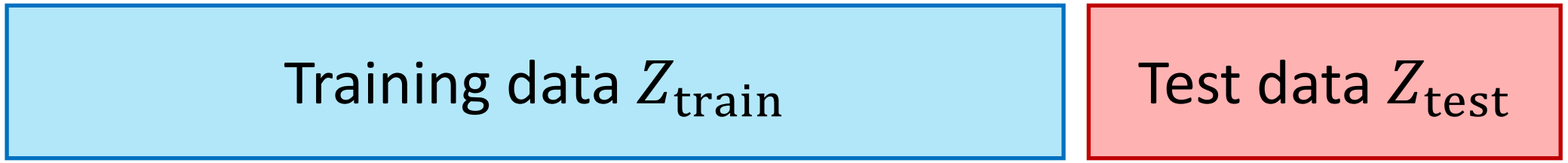
Training/Test Split

- **Issue:** How to detect overfitting vs. underfitting?
- **Solution:** Use **held-out test data** to estimate loss on new data
  - Typically, randomly shuffle data first



Training/Test Split Algorithm

- **Step 1:** Split Z into Z_{train} and Z_{test}

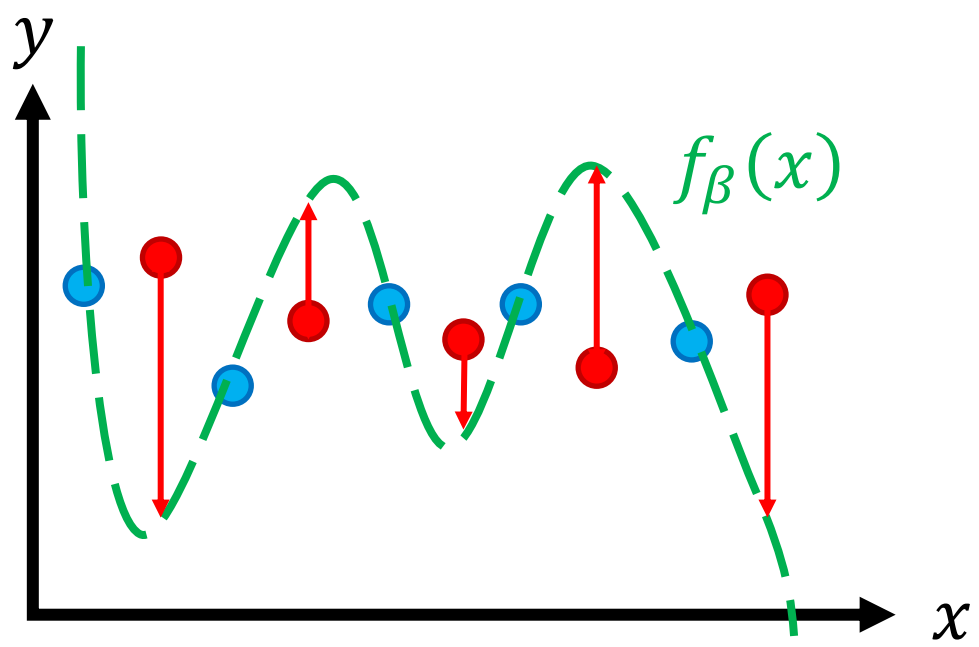


- **Step 2:** Run linear regression with Z_{train} to obtain $\hat{\beta}(Z_{\text{train}})$

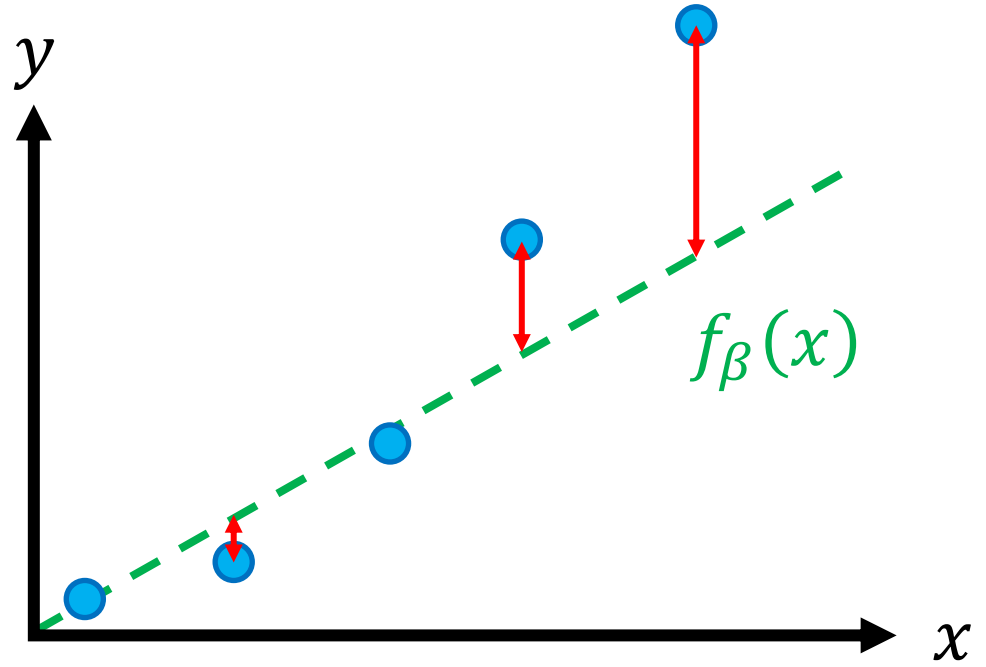
- **Step 3:** Evaluate
  - **Training loss:** $L_{\text{train}} = L\left(\hat{\beta}(Z_{\text{train}}); Z_{\text{train}}\right)$
  - **Test (or generalization) loss:** $L_{\text{test}} = L\left(\hat{\beta}(Z_{\text{train}}); Z_{\text{test}}\right)$

Training/Test Split Algorithm

- **Overfitting**
  - L_{train} is small
  - L_{test} is large



- **Underfitting**
  - L_{train} is large
  - L_{test} is large



How to Fix Underfitting/Overfitting?

- Choose the right model family!

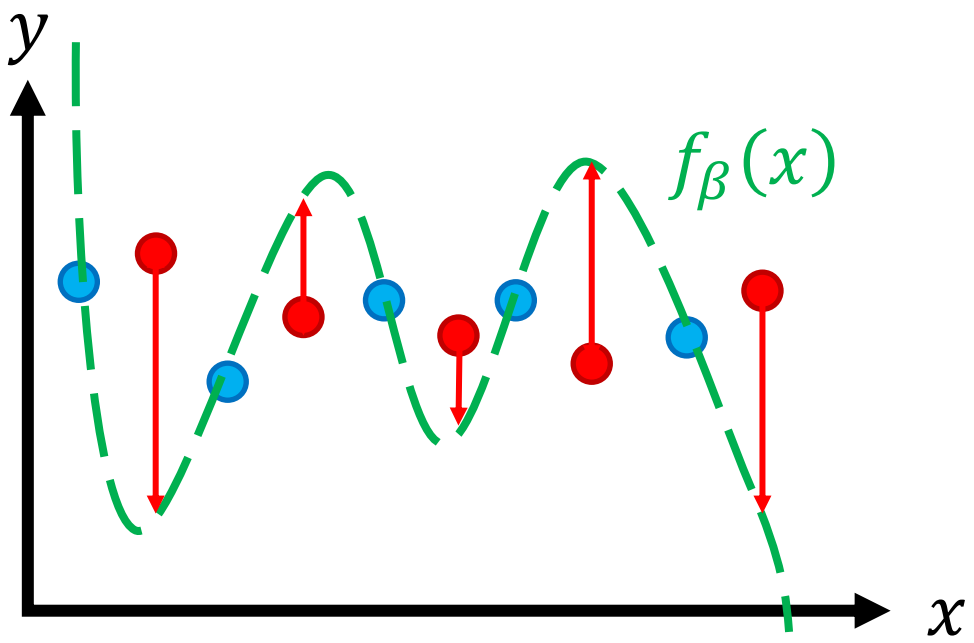
Role of Capacity

- **Capacity** of a model family captures "complexity" of data it can fit
  - Higher capacity → more likely to overfit (model family has high **variance**)
  - Lower capacity → more likely to underfit (model family has high **bias**)

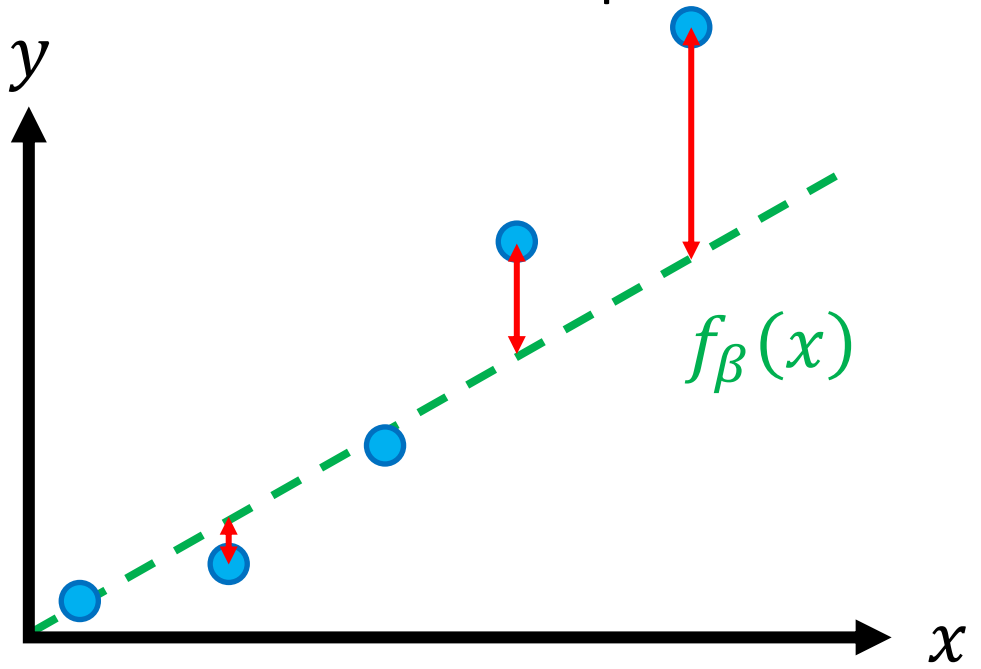
- For linear regression, capacity corresponds to feature dimension d
  - I.e., number of features in $\phi(x)$

Bias-Variance Tradeoff

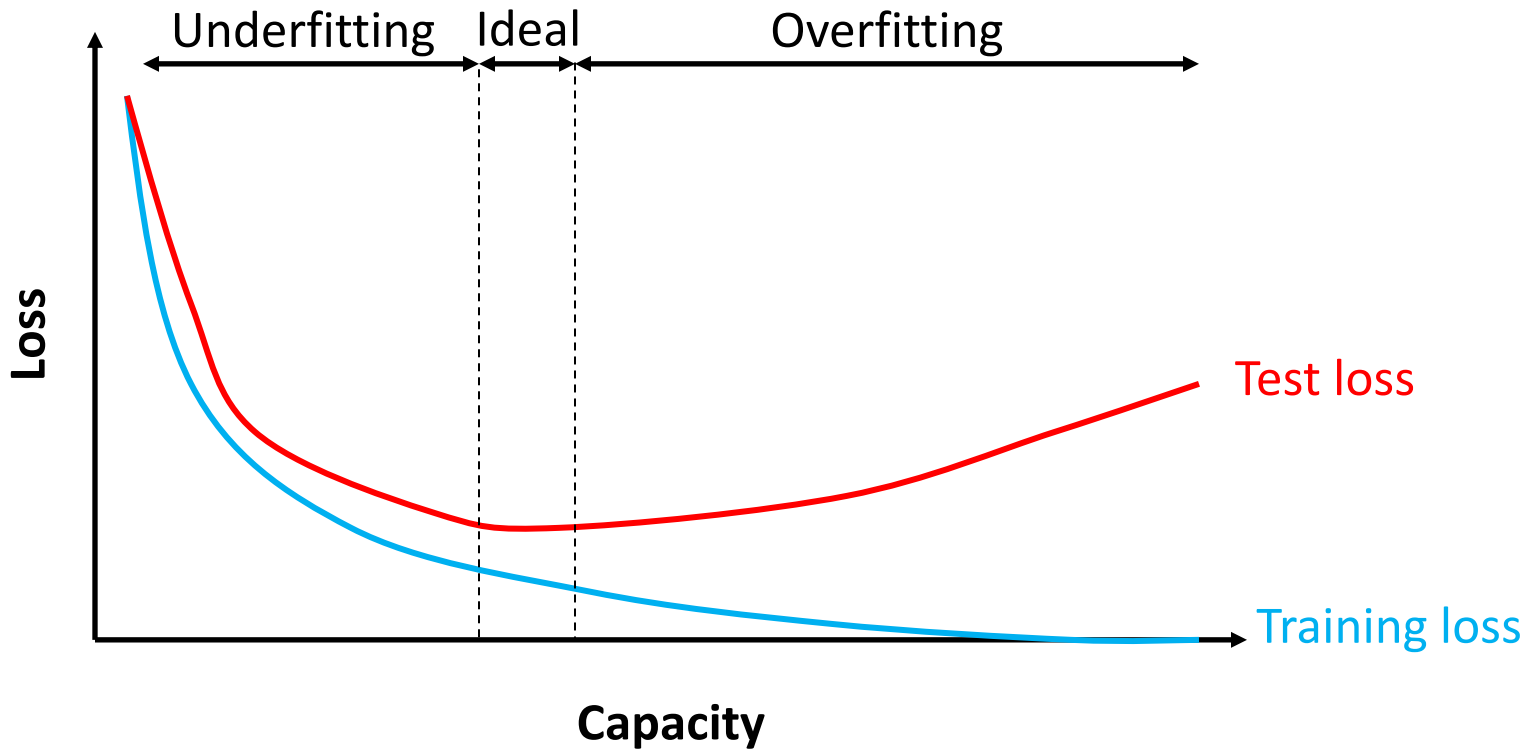
- **Overfitting (high variance)**
  - High capacity model capable of fitting complex data
  - Insufficient data to constrain it



- **Underfitting (high bias)**
  - Low capacity model that can only fit simple data
  - Sufficient data but poor fit



Bias-Variance Tradeoff



Slide by Padhraic Smyth, UCIrvine

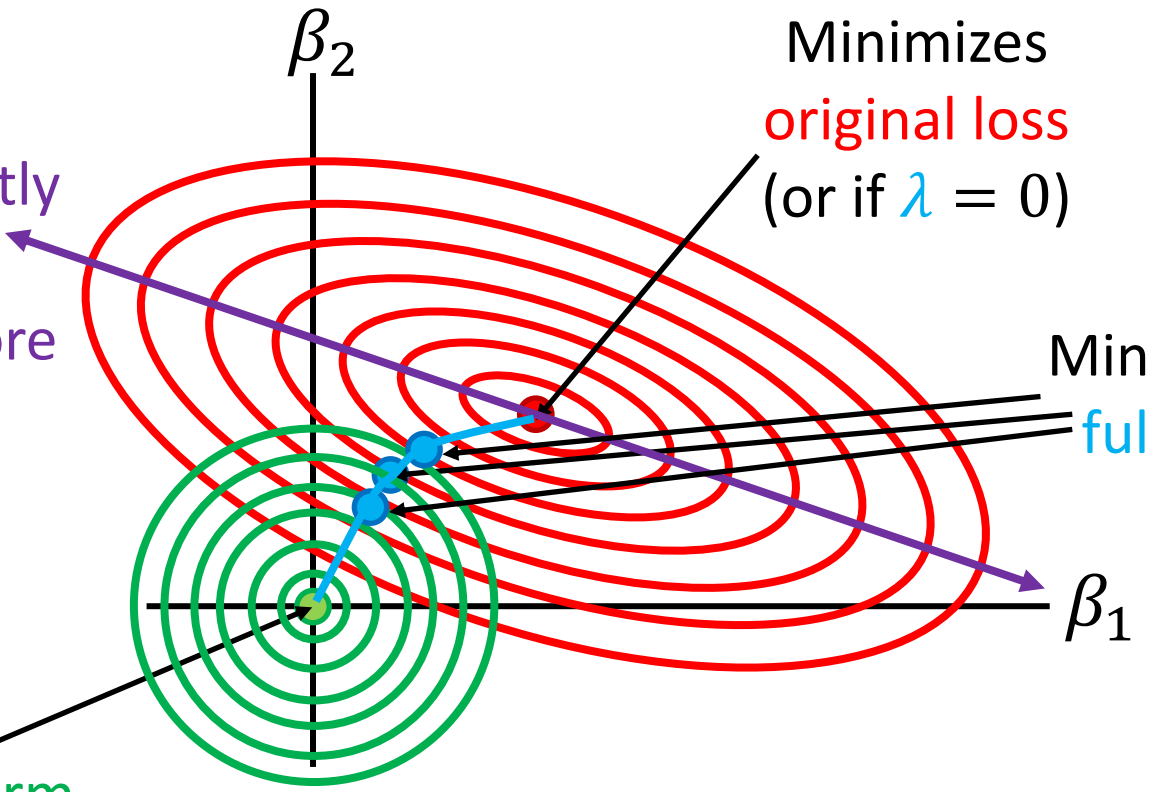
Linear Regression with L_2 Regularization

- **Original loss** + **regularization**:

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \cdot \|\beta\|_2^2$$
$$= \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d}\beta_j^2$$

- λ is a **hyperparameter** that must be tuned (satisfies $\lambda \geq 0$)

Intuition on L_2 Regularization



Minimizes original loss (or if $\lambda = 0$)

Loss varies greatly in this direction → Penalizes more

Minimizes full loss

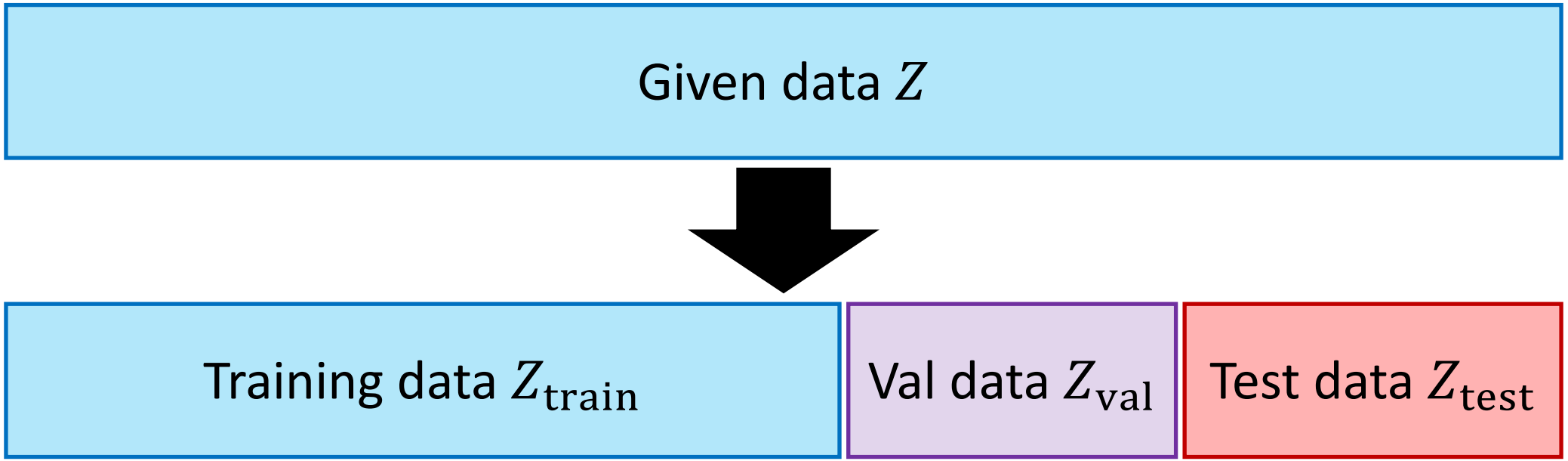
Minimizes regularization term (or if $\lambda \to \infty$)

- At this point, the gradients are **equal** (with opposite sign)
- Tradeoff depends on choice of λ

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d}\beta_j^2$$

Training/Val/Test Split

- **Goal:** Choose best hyperparameter λ
  - Can also compare different model families, feature maps, etc.
- **Solution:** Optimize λ on a **held-out validation data**
  - **Rule of thumb:** 60/20/20 split



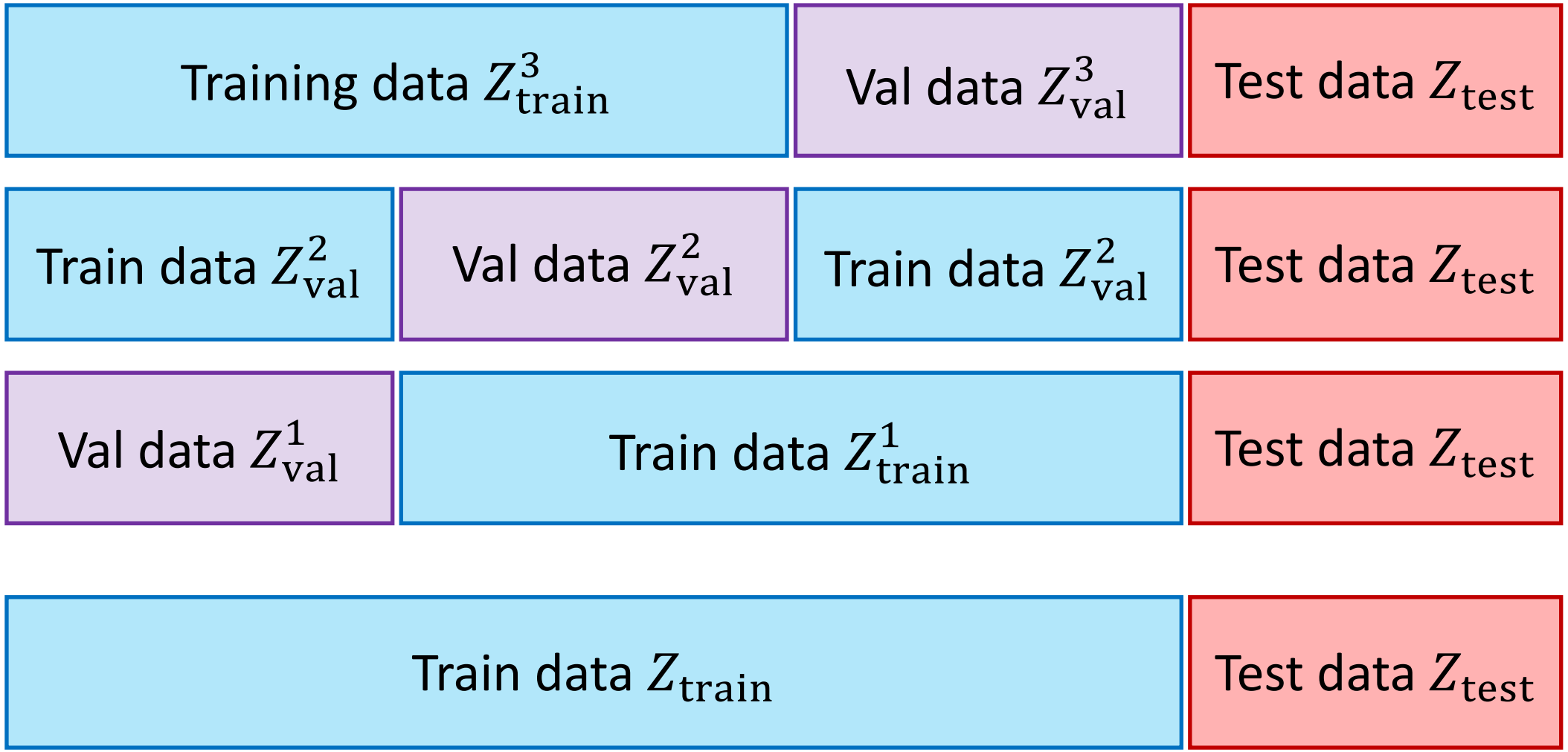
Basic Cross Validation Algorithm

- **Step 1:** Split Z into Z_{train}, Z_{val}, and Z_{test}

- **Step 2:** For $t \in \{1, \ldots, h\}$:
  - **Step 2a:** Run linear regression with Z_{train} and λ_t to obtain $\hat{\beta}(Z_{\text{train}}, \lambda_t)$
  - **Step 2b:** Evaluate validation loss $L_{\text{val}}^t = L\left(\hat{\beta}(Z_{\text{train}}, \lambda_t); Z_{\text{val}}\right)$

- **Step 3:** Use best λ_t
  - Choose $t' = \arg\min_t L_{\text{val}}^t$ with lowest validation loss
  - Re-run linear regression with Z_{train} and $\lambda_{t'}$ to obtain $\hat{\beta}(Z_{\text{train}}, \lambda_{t'})$

Example: 3-Fold Cross Validation



Outline of ML part:

1. Introduction
2. Linear Regression
3. LR: Loss functions
4. LR: feature maps
5. LR: Overfitting vs Underfitting
6. **LR: Gradient Decent**
7. NN: Basic Structure
8. NN: Backpropagation
9. NN: Tips/Tricks/Problems

k-Fold Cross-Validation

- If Z is small, then splitting it can reduce performance
  - Can use $Z_{\text{train}} \cup Z_{\text{val}}$ in Step 3

- **Alternative:** k-fold cross-validation (e.g., $k = 3$)
  - Split Z into Z_{train} and Z_{test}
  - Split Z_{train} into k disjoint sets Z_{val}^{s}, and let $Z_{\text{train}}^{s} = \bigcup_{s' \neq s} Z_{\text{val}}^{s}$
  - Use λ' that works best on average across $s \in \{1, \ldots, k\}$ with Z_{train}
  - Chooses better λ' than above strategy

- **Compute vs. accuracy tradeoff**
  - As $k \rightarrow N$, the model becomes more accurate
  - But algorithm becomes more computationally expensive

Strategy 2: Gradient Descent

- **Gradient descent:** Update β based on **gradient** $\nabla_\beta L(\beta; Z)$ of $L(\beta; Z)$:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- **Intuition:** The gradient is the direction along which $L(\beta; Z)$ changes most quickly as a function of β

- $\alpha \in \mathbb{R}$ is a hyperparameter called the **learning rate**
  - More on this later

Strategy 2: Gradient Descent

- Choose initial value for β
- Until we reach a minimum:
  - Choose a new value for β to reduce $L(\beta; Z)$

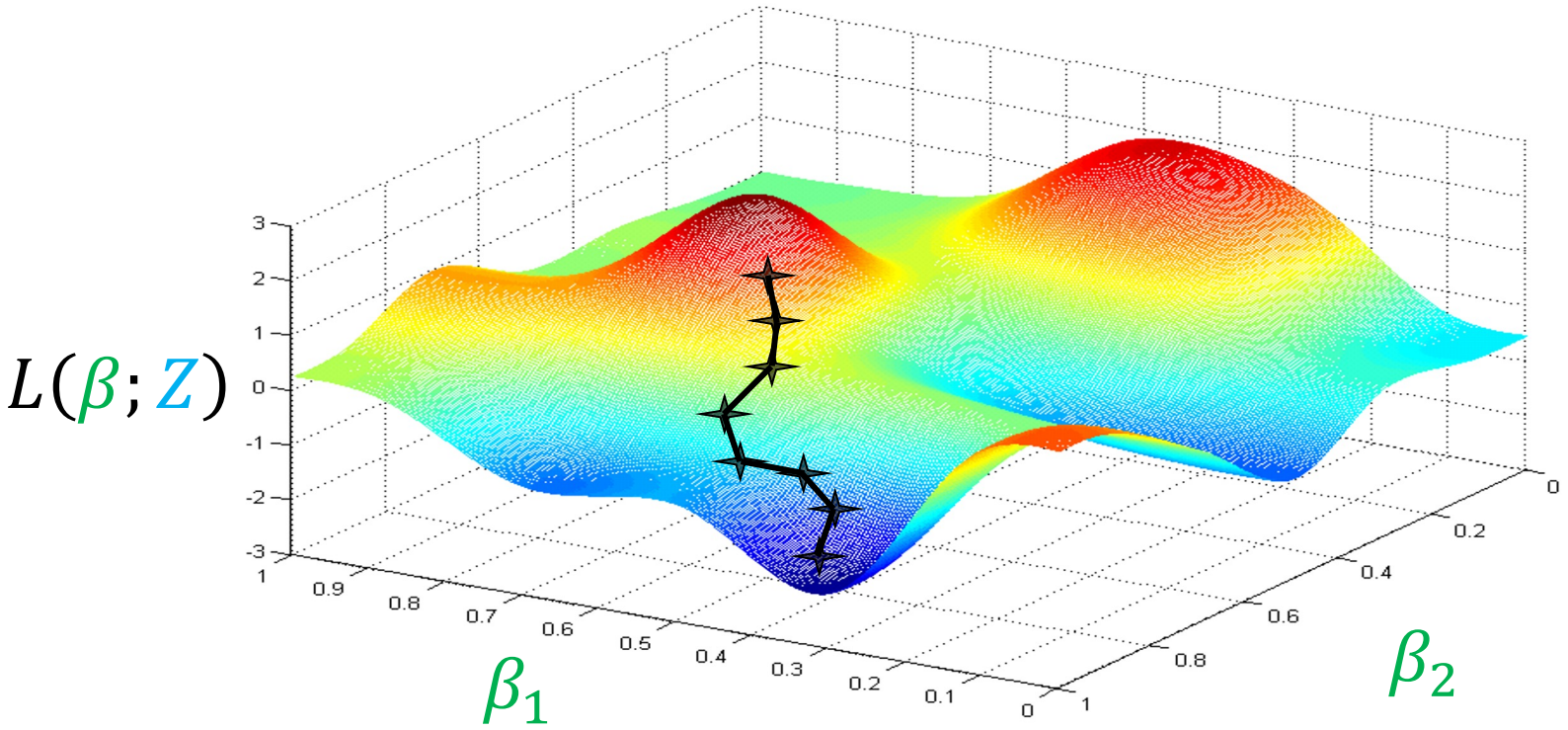


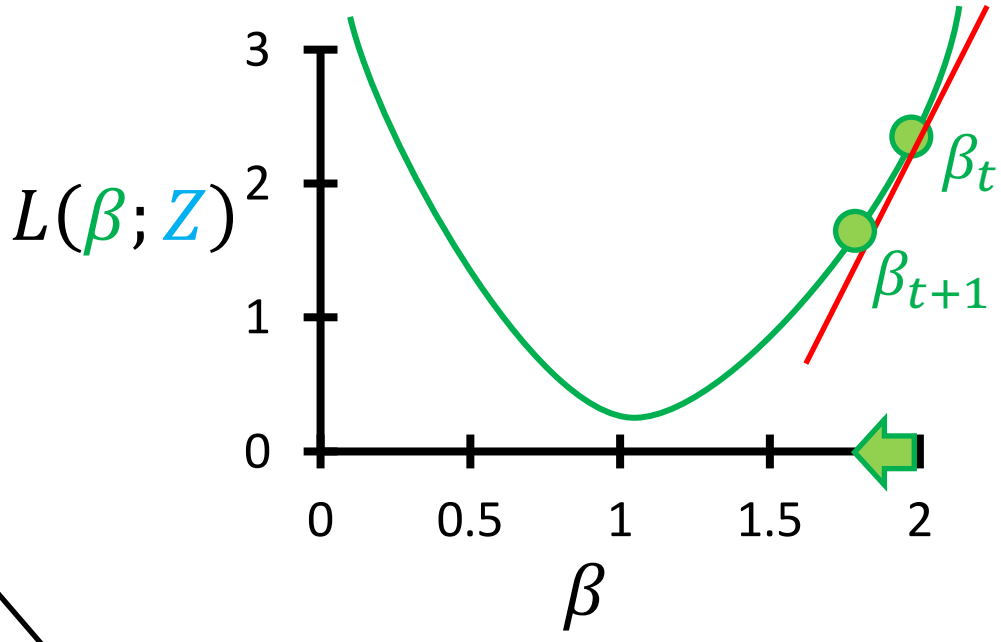
Figure by Andrew Ng

Strategy 2: Gradient Descent

- Initialize $\beta_1 = 0$
- Repeat until convergence:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1



For in-place updates $\beta \leftarrow \beta - \alpha \cdot \nabla_\beta L(\beta; Z)$, compute all components of $\nabla_\beta L(\beta; Z)$ before modifying β

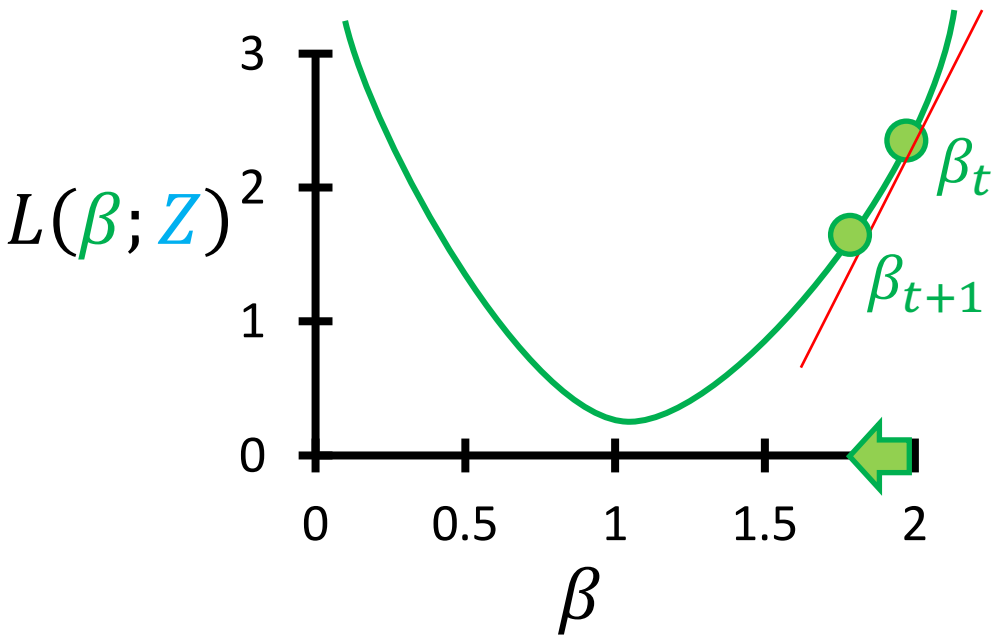
Strategy 2: Gradient Descent

- Initialize $\beta_1 = \vec{0}$
- Repeat until $\|\beta_t - \beta_{t+1}\|_2 \leq \epsilon$:

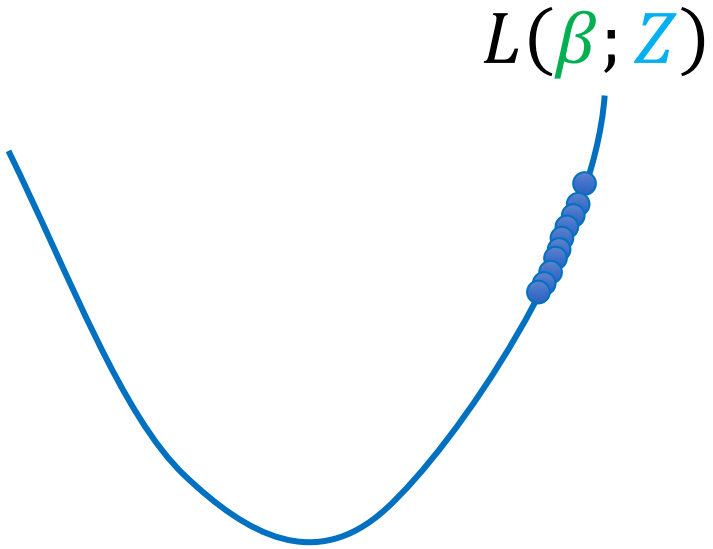
$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1

Hyperparameter defining convergence

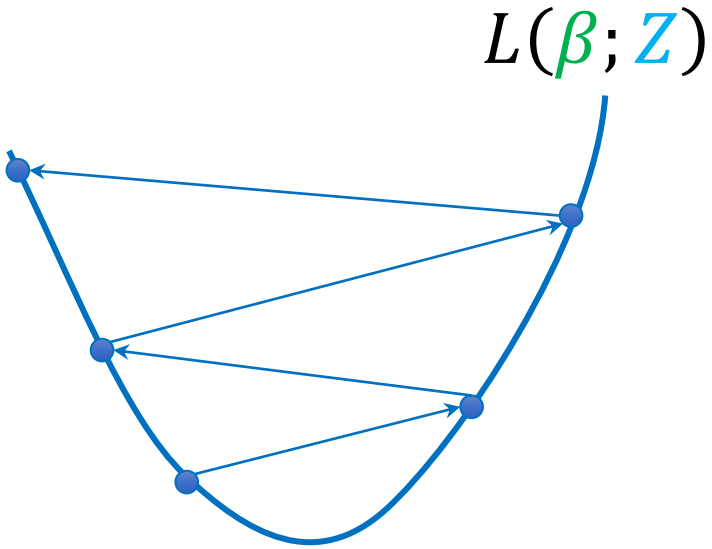


Choice of Learning Rate $\boldsymbol{\alpha}$



Problem: α too small

- $L(\beta; Z)$ decreases slowly



Problem: α too large

- $L(\beta; Z)$ increases!

Plot $L(\beta_t; Z_{\text{train}})$ vs. t to diagnose these problems

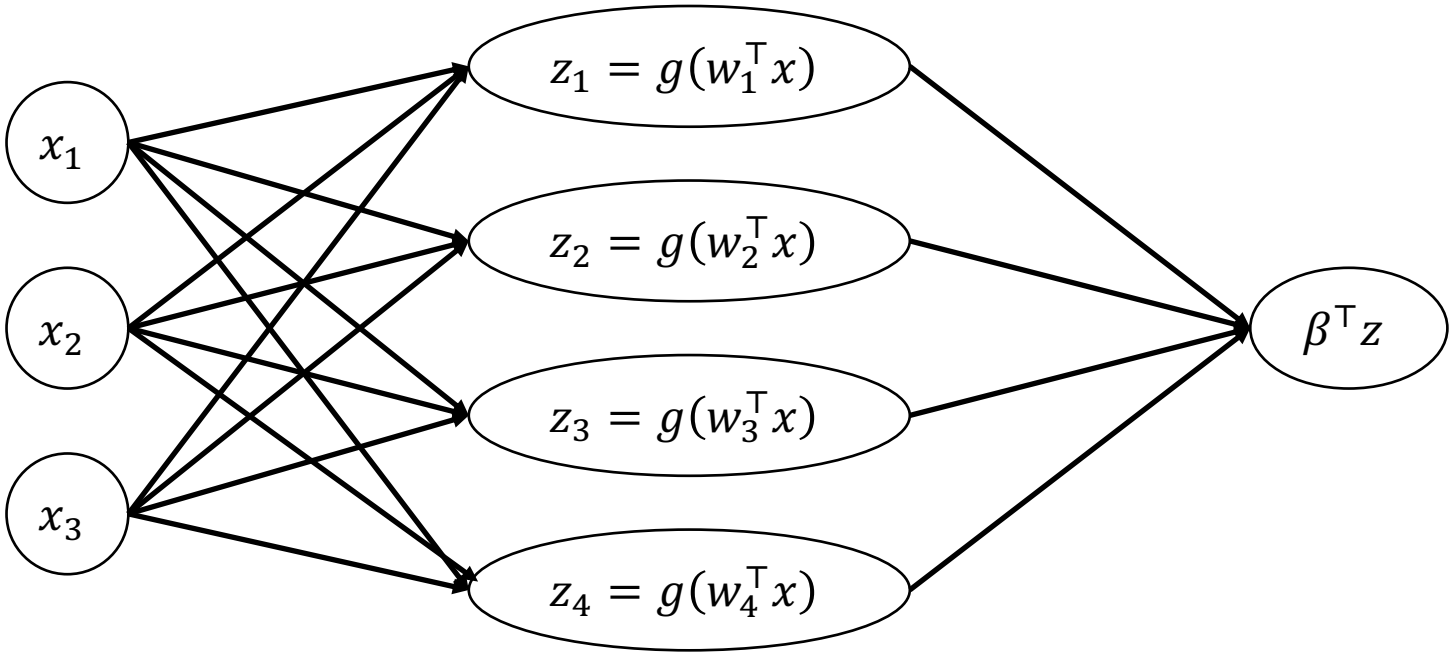
Outline of ML part:

1. Introduction
2. Linear Regression
3. LR: Loss functions
4. LR: feature maps
5. LR: Overfitting vs Underfitting
6. LR: Gradient Decent
7. **NN: Basic Structure**
8. NN: Backpropagation
9. NN: Tips/Tricks/Problems

Modern View

- **Feedforward neural network model family:**

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$



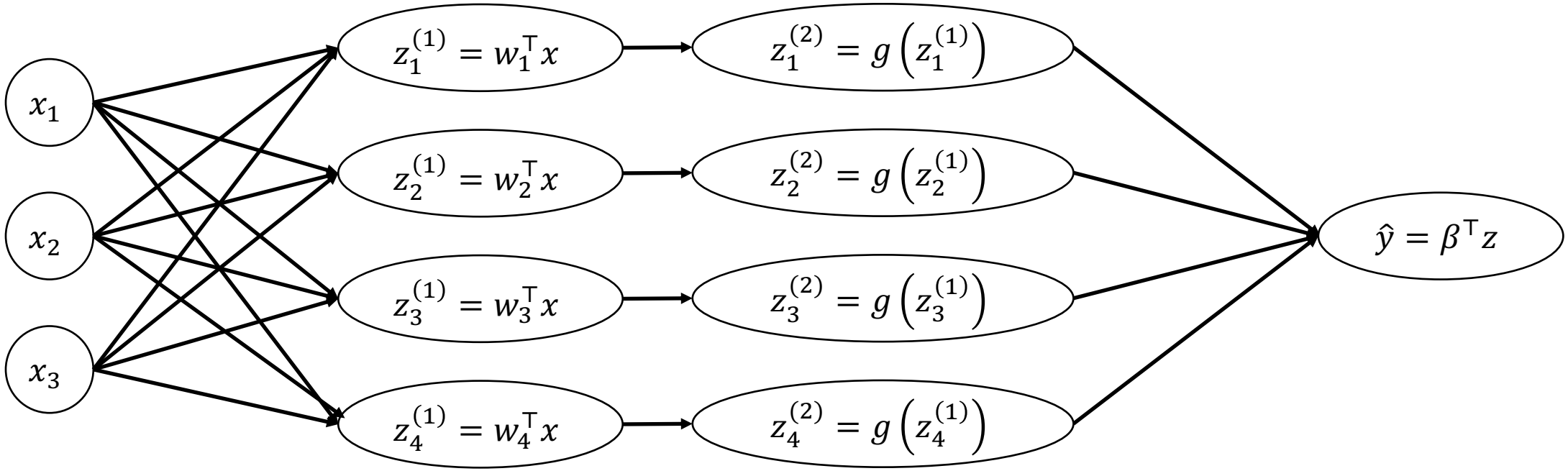
Modern View

Function composition:

$$f \circ g(x) = f\big(g(x)\big)$$

- **Feedforward neural network model family:**

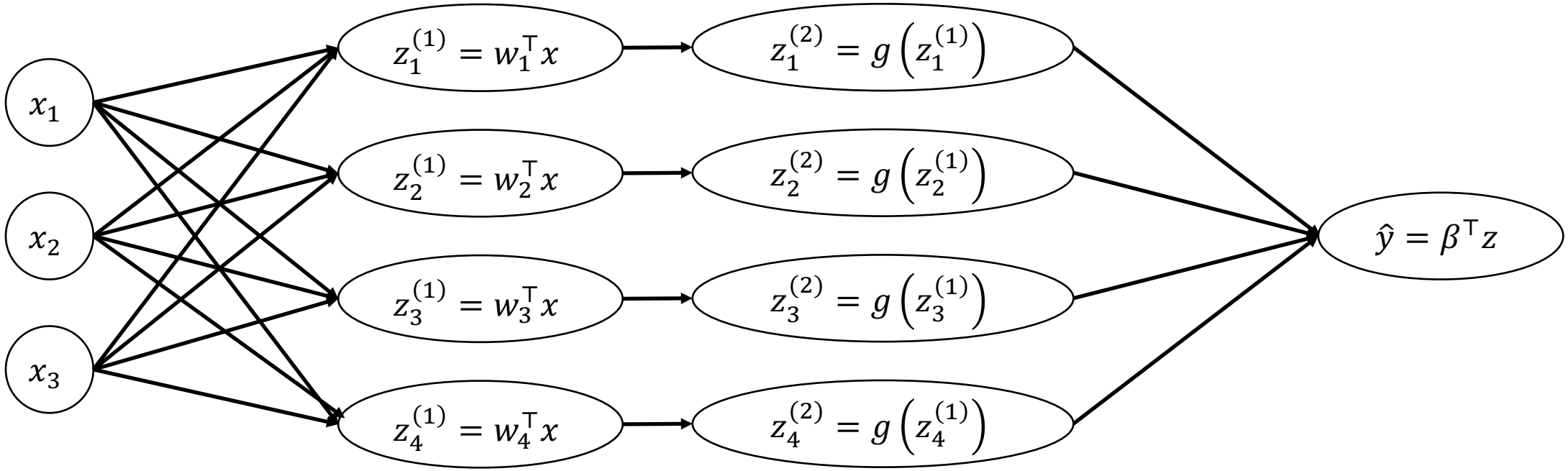
$$f_{W,\beta}(x) = f_\beta\Big(g\big(f_W(x)\big)\Big) = f_\beta \circ g \circ f_W(x)$$



Modern View

- **Feedforward neural network model family (for regression):**

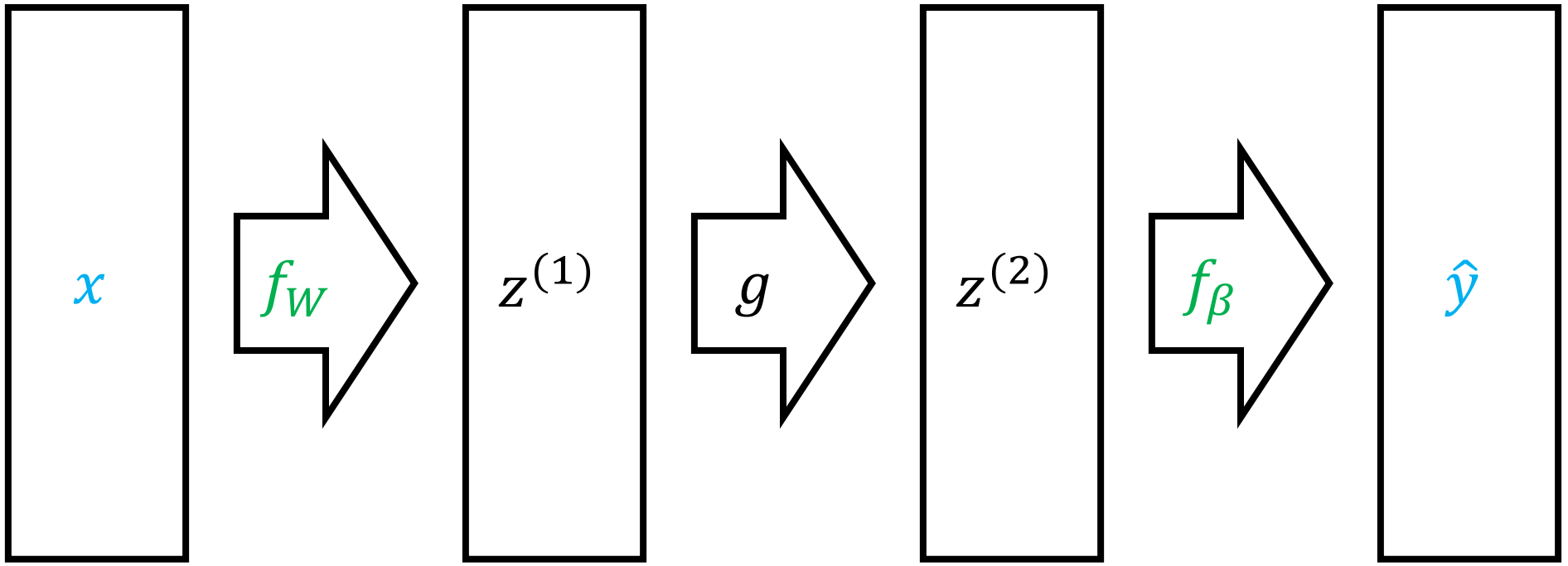
$$f_{W,\beta}(x) = f_\beta \circ g \circ f_W(x)$$



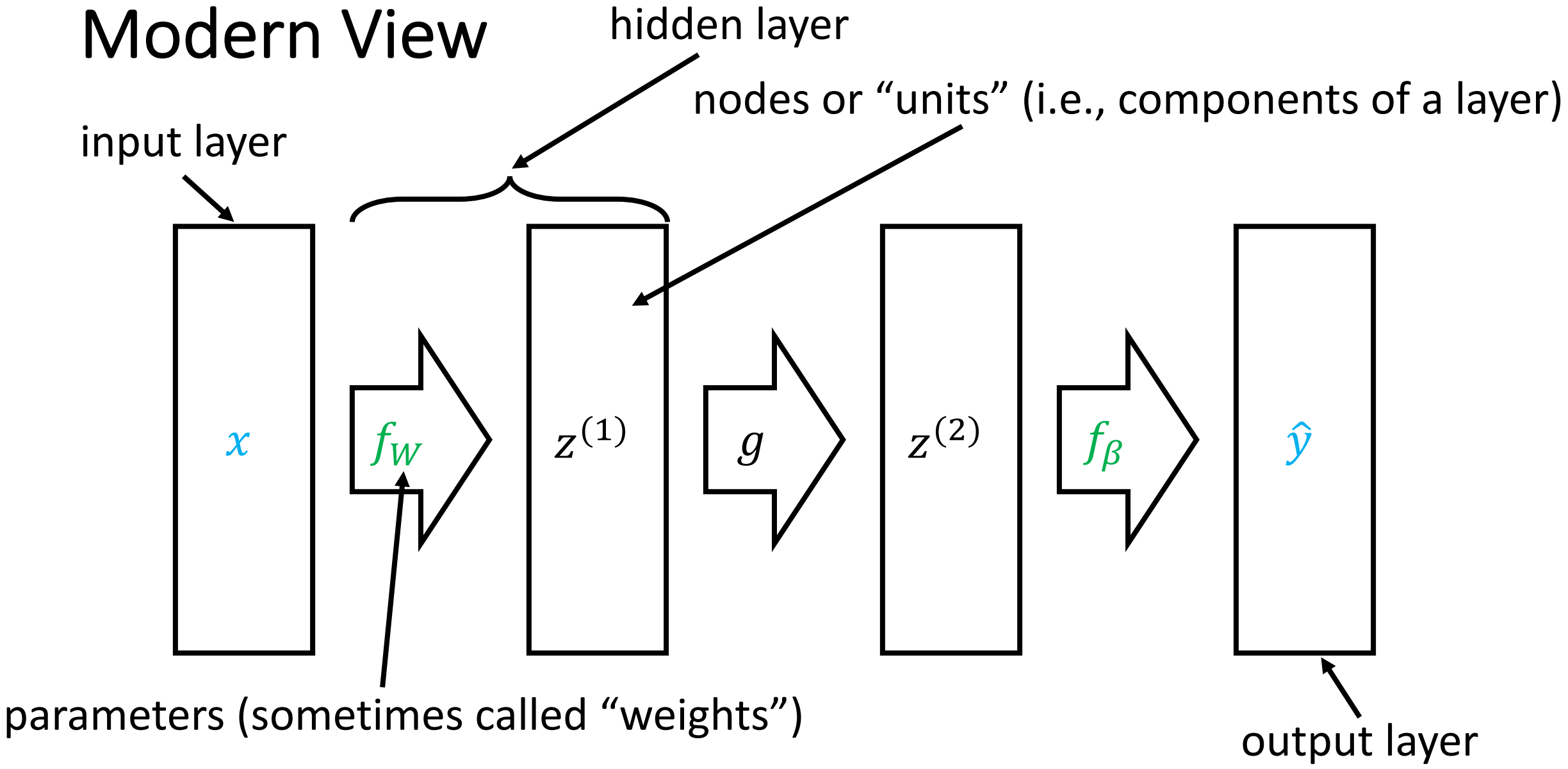
Modern View

- **Feedforward neural network model family (for regression):**

$$f_{W,\beta}(x) = f_\beta \circ g \circ f_W(x)$$



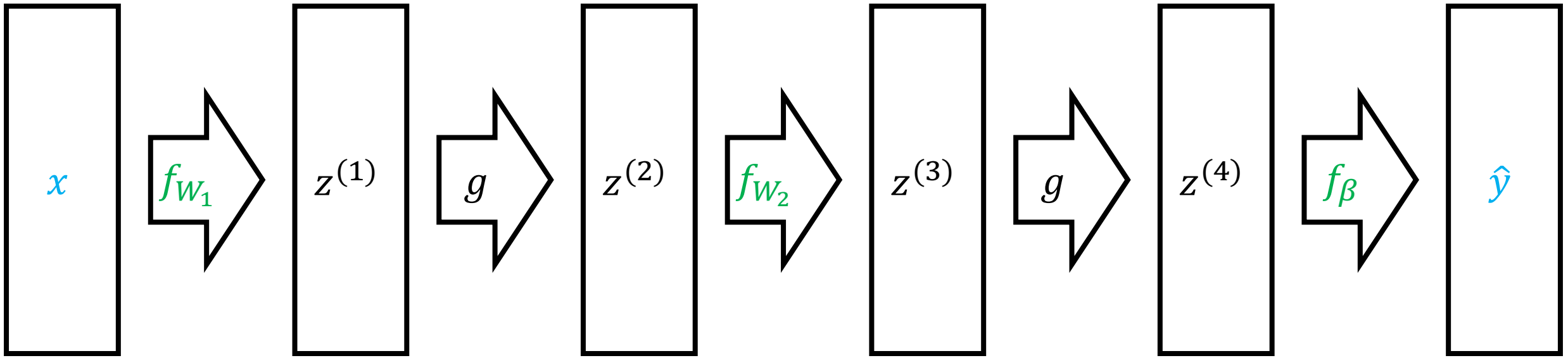
Modern View


hidden layer
nodes or “units” (i.e., components of a layer)
input layer
x
f_W
$z^{(1)}$
g
$z^{(2)}$
f_β
$\hat{y}$
parameters (sometimes called “weights”)
output layer

Modern View

- **Neural network with two hidden linear layers:**

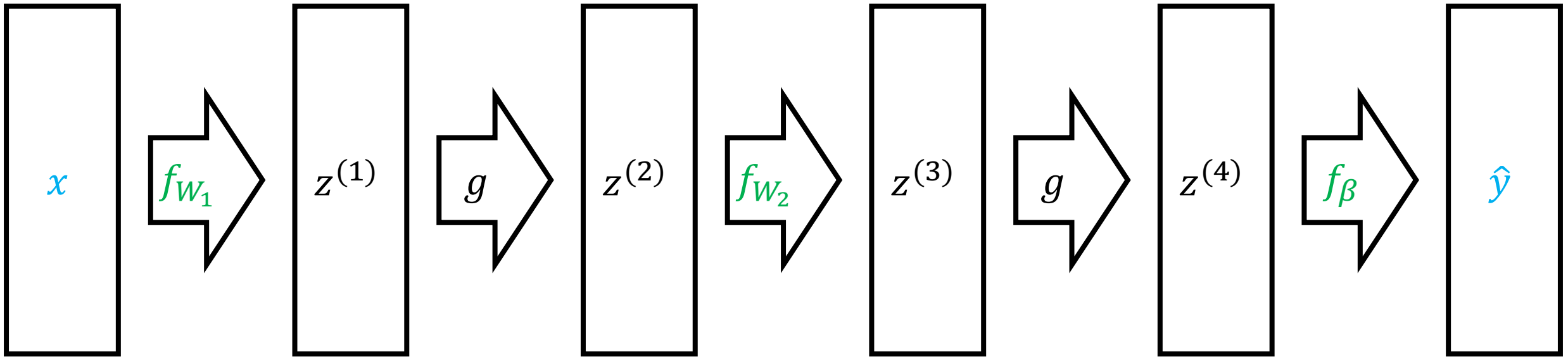
$$f_{W_1,W_2,\beta}(x) = f_\beta \circ g \circ f_{W_2} \circ g \circ f_{W_1}(x)$$



Modern View

- **Neural network with two hidden linear layers:**

$$f_{W_1,W_2,\beta}(x) = f_\beta\left(g\left(f_{W_2}\left(g\left(f_{W_1}(x)\right)\right)\right)\right)$$



Learn successively more “high-level” representations

What About Classification?

- **Recall:** For logistic regression, we choose the likelihood to be

$$p_\beta(Y = 1 \mid x) = \frac{1}{1 + e^{-\beta^\top x}}$$

What About Classification?

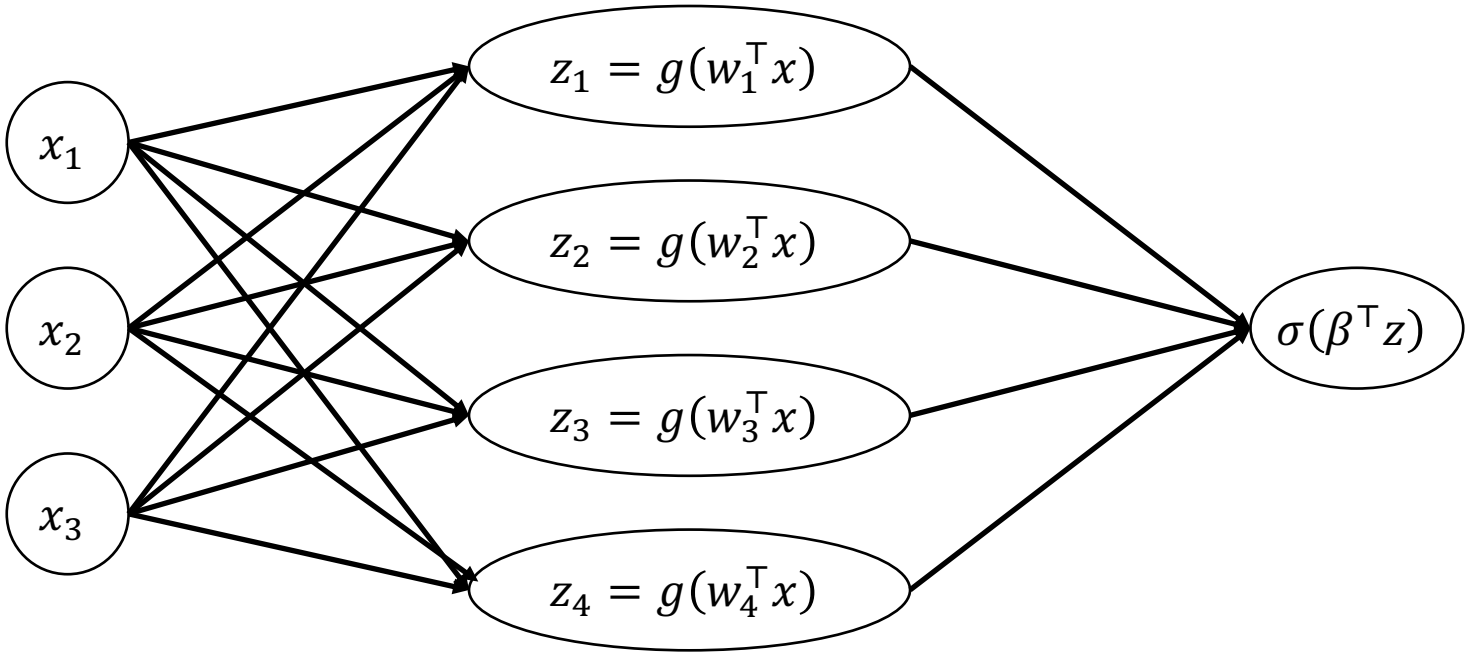
- **Recall:** For logistic regression, we choose the likelihood to be

$$p_\beta(Y = 1 \mid x) = \sigma(\beta^\top x)$$

What About Classification?

- **For binary classification:**

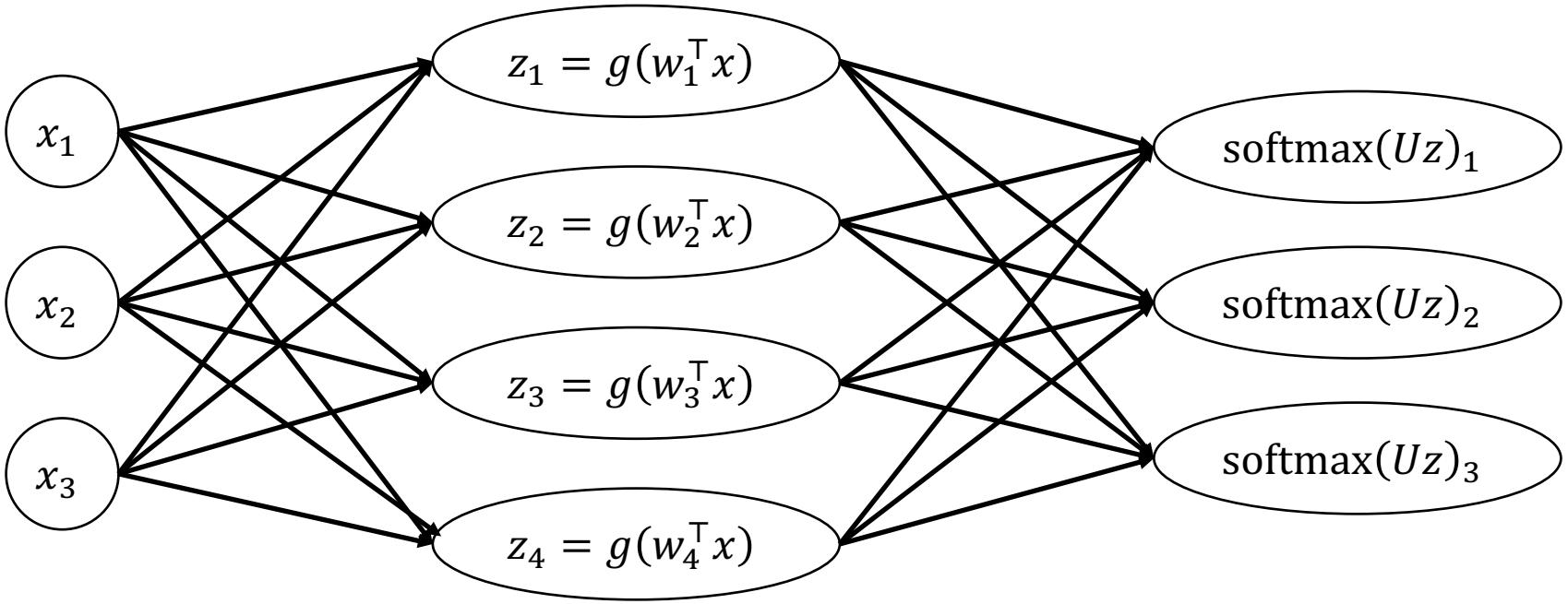
$$p_{W,\beta}(Y = 1 \mid x) = \sigma\left(\beta^\top g(Wx)\right)$$



What About Classification?

- **For multi-class classification:**

$$p_{W,U}(Y = y \mid x) = \text{softmax}\big(Ug(Wx)\big)_y$$



Neural Networks

- **Pros**
  - **“Meta” strategy:** Enables users to **design** model family
  - Design model families that capture **symmetries/structure** in the data (e.g., read a sentence forwards, translation invariance for images, etc.)
  - “Representation learning” (automatically learn features for certain domains)
  - More parameters!

- **Cons**
  - Very hard to train! (Non-convex loss functions)
  - Lots of parameters → need lots of data!
  - Lots of design decisions

Outline of ML part:

1. Introduction
2. Linear Regression
3. LR: Loss functions
4. LR: feature maps
5. LR: Overfitting vs Underfitting
6. LR: Gradient Decent
7. NN: Basic Structure
8. **NN: Backpropagation**
9. NN: Tips/Tricks/Problems

Optimization Algorithm

- Based on gradient descent, with a few tweaks
  - **Note:** Loss is nonconvex, but gradient descent works well in practice

- **Key challenge:** How to compute the gradient?
  - **Strategy so far:** Work out gradient for every model family
  - **New strategy:** Algorithm for computing gradient of an arbitrary programmatic composition of layers
  - This algorithm is called **backpropagation**

Gradient Descent

- $W_1 \leftarrow \text{Initialize}(\,)$
- **for** $t \in \{1,2, \ldots\}$ **until** convergence:

$$W_{t+1,j} \leftarrow W_{t,j} - \frac{\alpha}{n} \cdot \sum_{i=1}^{n} \nabla_{W_j} L\left(f_{W_t}(x_i), y_i\right) \quad \text{(for each } j\text{)}$$

- **return** f_{W_t}

Backpropagation

- **Input**
  - Example-label pair (x, y)
  - Arbitrary model $f_{W_m} \circ \cdots \circ f_{W_1}$
  - Loss $L(\hat{y}, y)$ for predicted label $\hat{y}$ and true label y
  - Derivative $\nabla_{\hat{y}} L(\hat{y}, y)$ (as a function)
  - Derivatives $D_{W_j} f_{W_j}(z)$ and $D_z f_{W_j}(z)$ (e.g., as a function)

- **Output:** $\nabla_{W_j} L(f_W(x), y)$

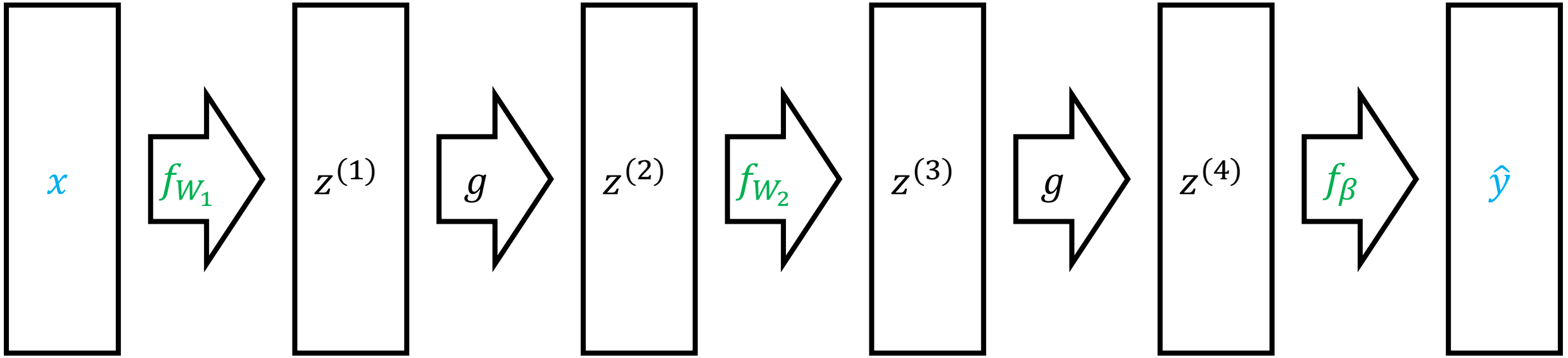
Backpropagation Example

- **Derivative of neural network:**

$$D_\beta f_{W,\beta}(x) = D_\beta\left(f_\beta \circ g \circ f_W\right)(x)$$
$$= D_\beta f_\beta\left(g \circ f_W(x)\right)$$

$$D_W f_{W,\beta}(x) = D_W\left(f_\beta \circ g \circ f_W\right)(x)$$
$$= D_z f_\beta\left(g \circ f_W(x)\right) D_W(g \circ f_W)(x)$$
$$= D_z f_\beta\left(g \circ f_W(x)\right) D_z g\left(f_W(x)\right) D_W f_W(x)$$

Backpropagation



Forward pass: Compute $z^{(j)} = f_{W_j}\left(z^{(j-1)}\right)$

Backward pass: Compute $D^{(j)} = D^{(j+1)} D_z f_{W_{j+1}}\left(z^{(j)}\right)$ and $D_{W_j} f_W(x) = D^{(j)} D_{W_j} f_{W_j}\left(z^{(j-1)}\right)$

Final output: $\nabla_{\hat{y}} L\left(z^{(m)}, y\right)^\top D_{W_j} f_W(x)$

Backpropagation Algorithm

- **Forward pass:** Compute forwards from $j = 0$ to $j = m$
  - $z^{(j)} = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$

- **Backward pass:** Compute backwards from $j = m$ to $j = 1$
  - $D^{(j)} = \begin{cases} 1 & \text{if } j = m \\ D^{(j+1)} D_z f_{W_{j+1}}(z^{(j)}) & \text{if } j < m \end{cases}$
  - $D_{W_j} f_W(x) = D^{(j)} D_{W_j} f_{W_j}(z^{(j-1)})$

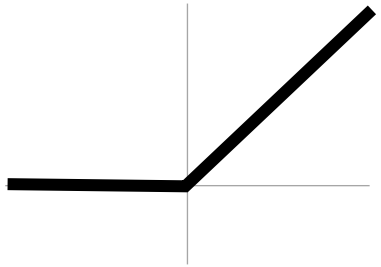
- **Final output:** $\nabla_{W_j} L(f_W(x), y)^\top = \nabla_{\hat{y}} L(z^{(m)}, y)^\top D_{W_j} f_W(x)$ for each j

Outline of ML part:

1. Introduction
2. Linear Regression
3. LR: Loss functions
4. LR: feature maps
5. LR: Overfitting vs Underfitting
6. LR: Gradient Decent
7. NN: Basic Structure
8. NN: Backpropagation
9. **NN: Tips/Tricks/Problems**

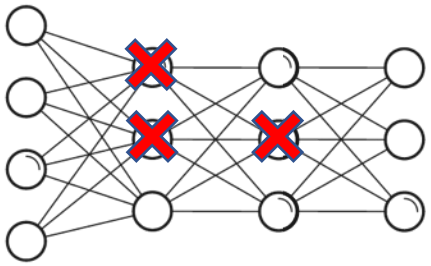
Neural Network Tips & Tricks

Optimization



Activation Functions

Managing Weights



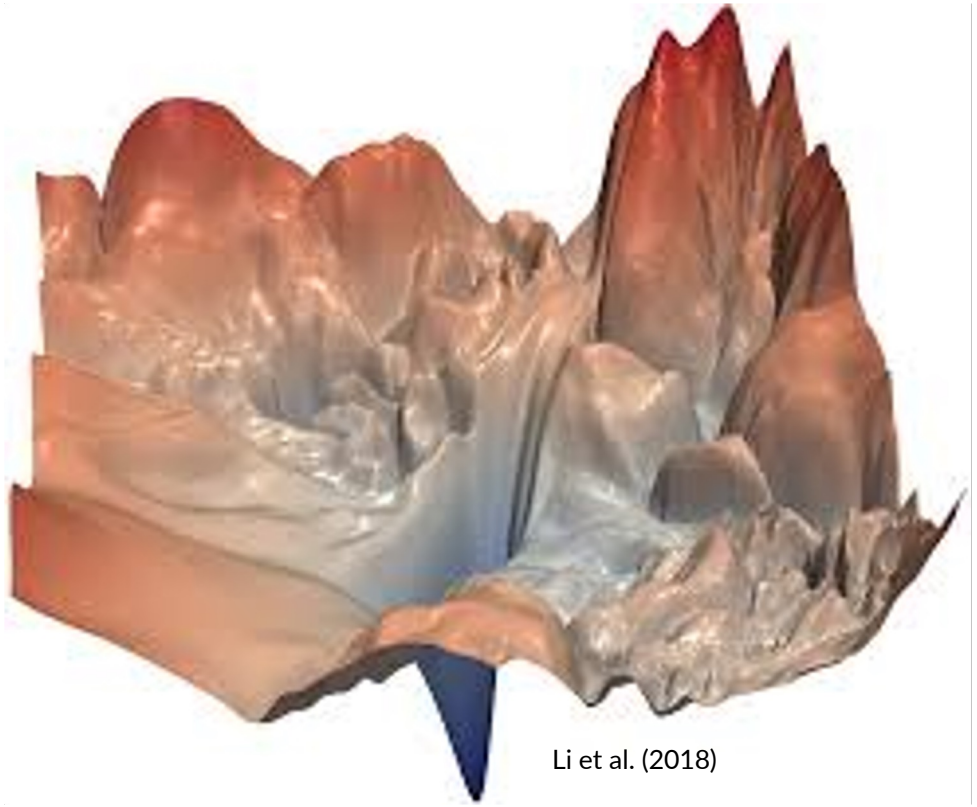
Dropout

Managing Training

Optimization Challenges

- **Challenges**
  - Local minima, saddle points due to non-convex loss
  - Exploding/vanishing gradients
  - Ill-conditioning

- Have heuristics that work in common cases (but not always)



Li et al. (2018)

Minibatch Stochastic Gradient Descent

- $W \leftarrow \text{Initialize}(\,)$
- **for** $t \in \{1,2,\ldots,T\}$:
  - **for** $i' \in \left\{1,2,\ldots,\frac{n}{k}\right\}$:

$$\beta \leftarrow \beta - \frac{\alpha}{k} \cdot \sum_{i=i'k}^{i'(k+1)-1} \nabla_\beta L\big(f_\beta(x_i), y_i\big) \quad \text{(for each } j\text{)}$$

- **return** f_β

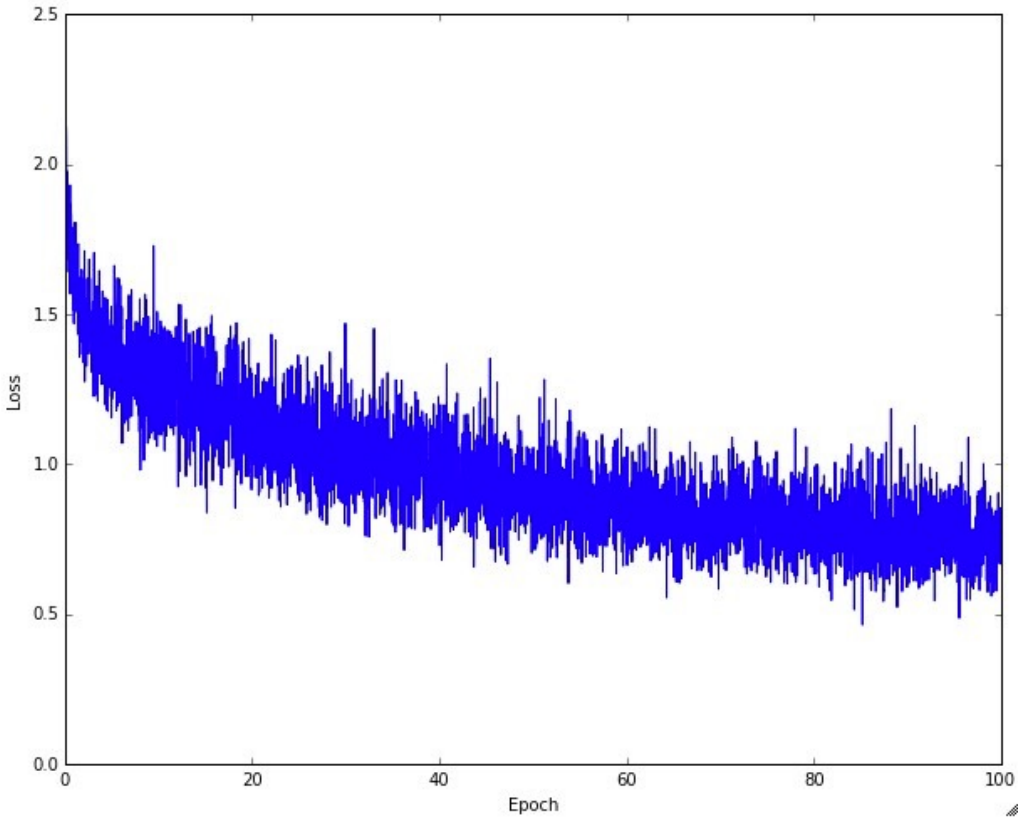
Accelerated Gradient Descent

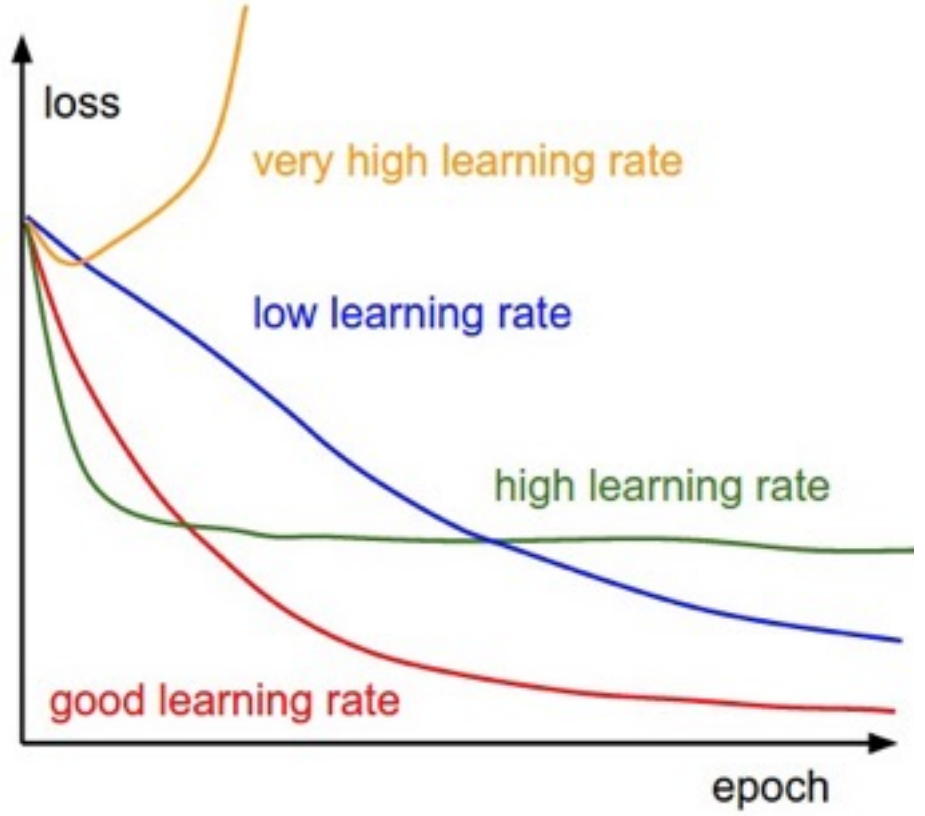
- **Intuition:** ρ holds the previous update $\alpha \cdot \nabla_\beta L\big(f_\beta(x), y\big)$, except it "remembers" where it was heading via momentum

- New hyperparameter μ (typically $\mu = 0.9$ or $\mu = 0.99$)

Learning Rate

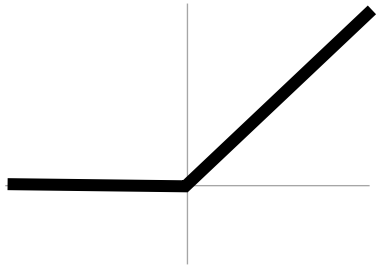
- Most important hyperparameter; tune by looking at training loss





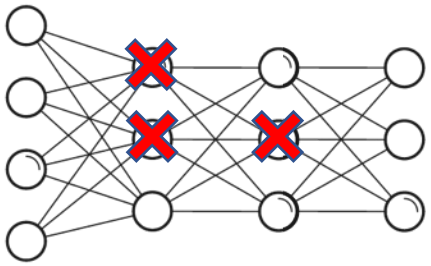
Neural Network Tips & Tricks

Optimization



Activation Functions

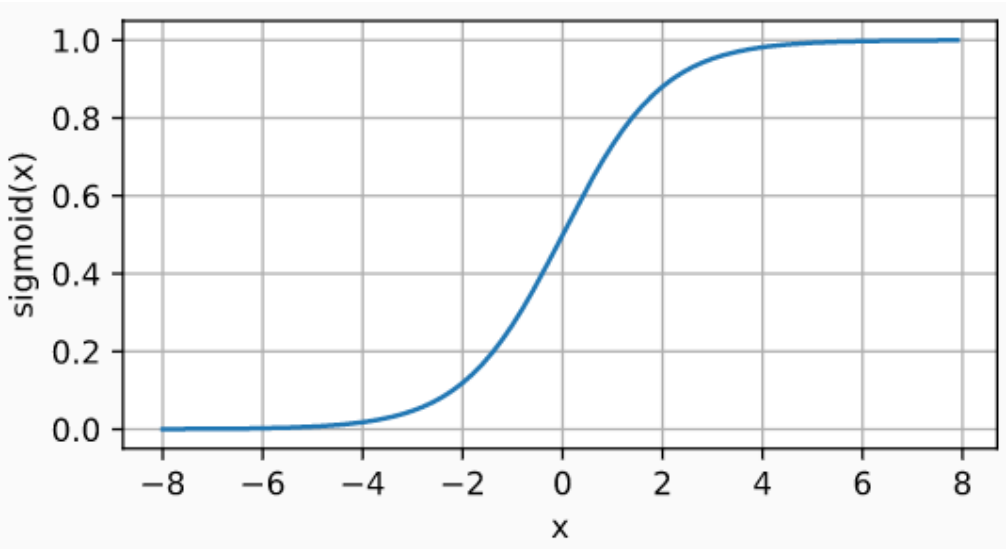
Managing Weights



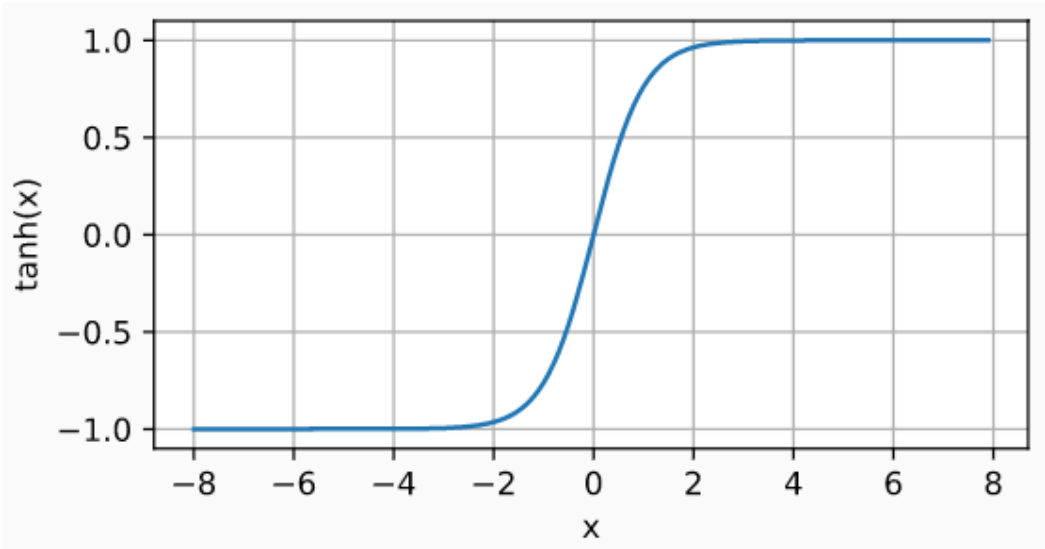
Dropout

Managing Training

Historical Activation Functions



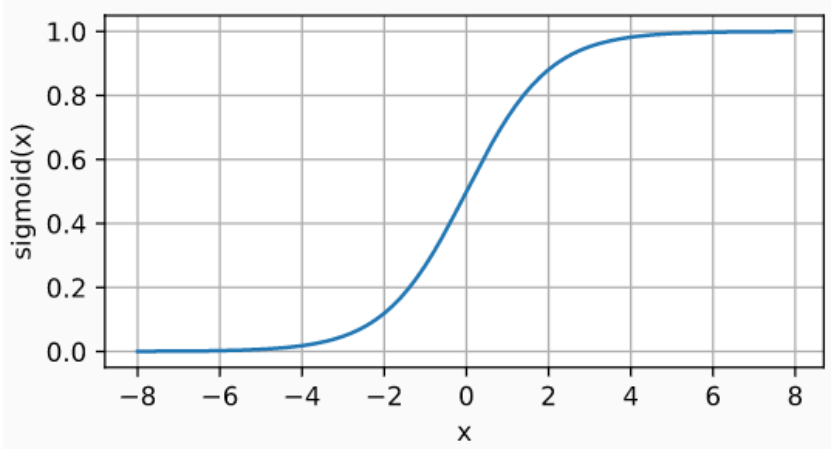
sigmoid



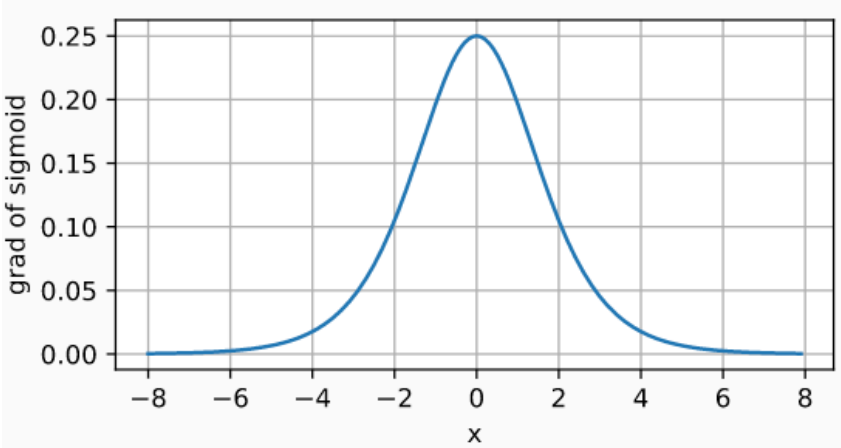
tanh

Vanishing Gradient Problem

- The gradient of the sigmoid function is often nearly zero
- **Recall:** In backpropagation, gradients are products of $\partial_z g\left(z^{(j)}\right)$
- **Quickly multiply to zero!**
  - Early layers update very slowly



sigmoid



sigmoid gradient

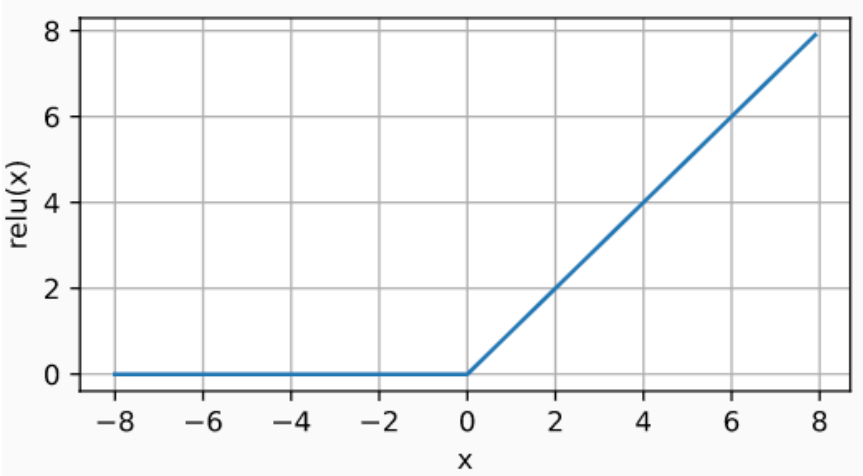
ReLU Activation

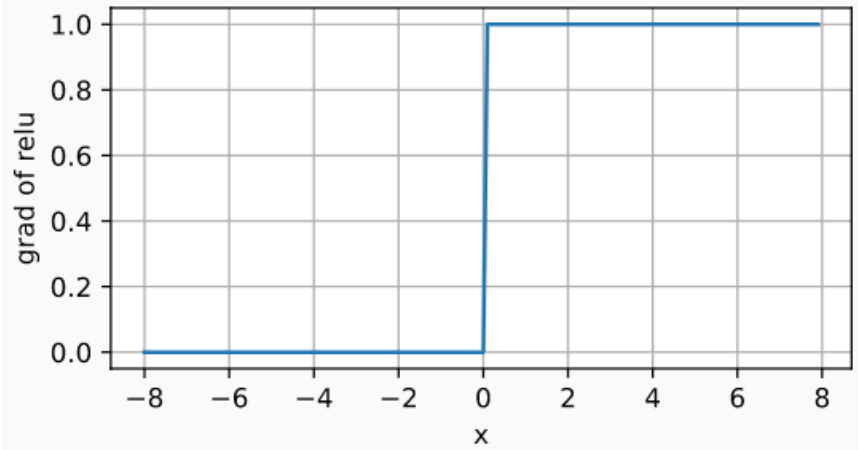
- Activation function

$$g(z) = \max\{0, z\}$$

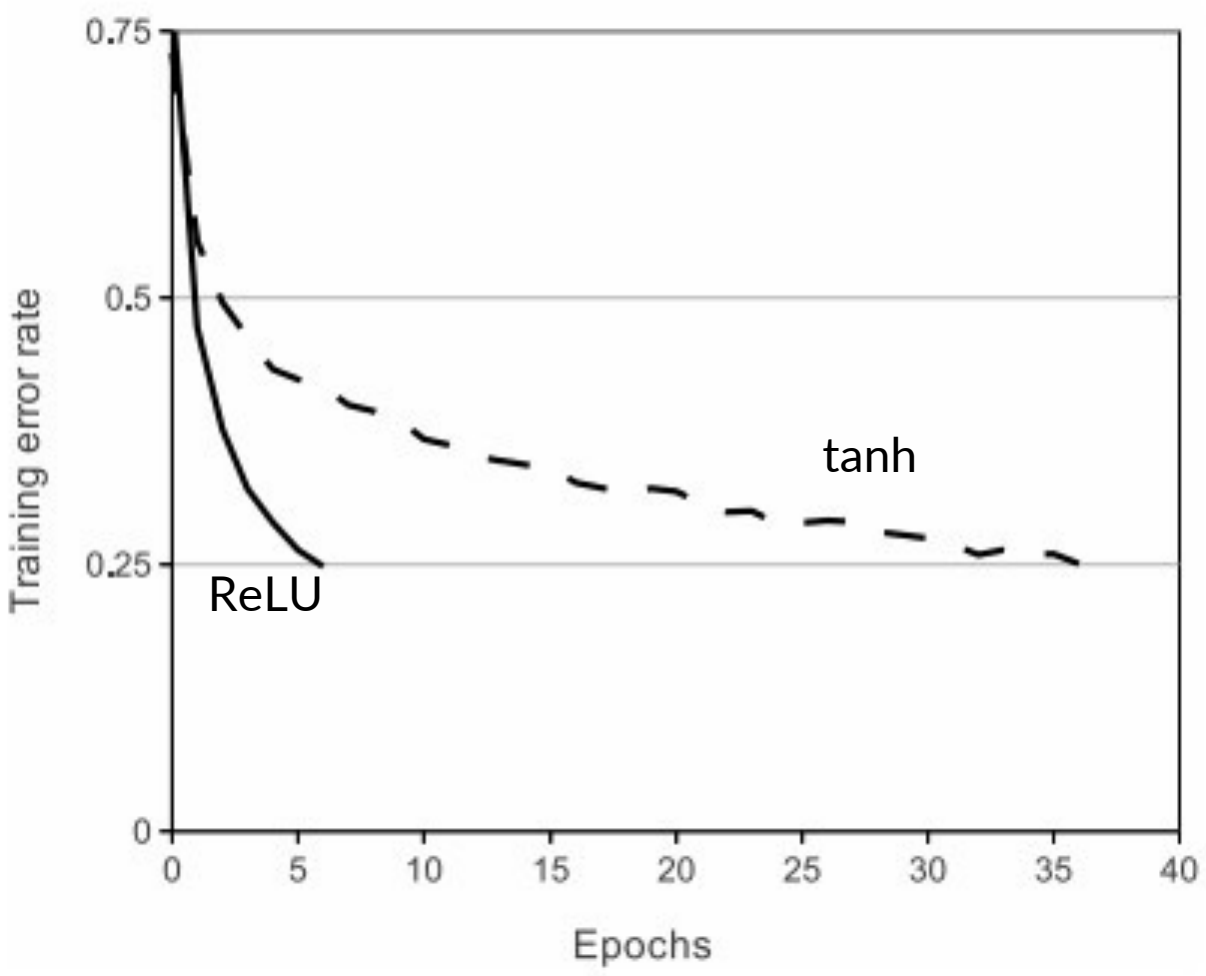
- Gradient now positive on the entire region $z \geq 0$

- Significant performance gains for deep neural networks

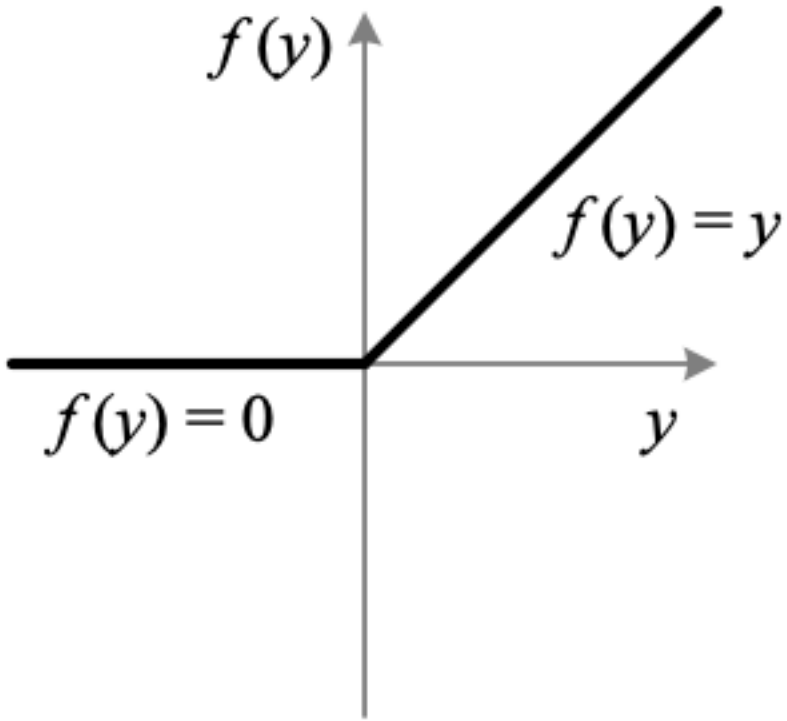


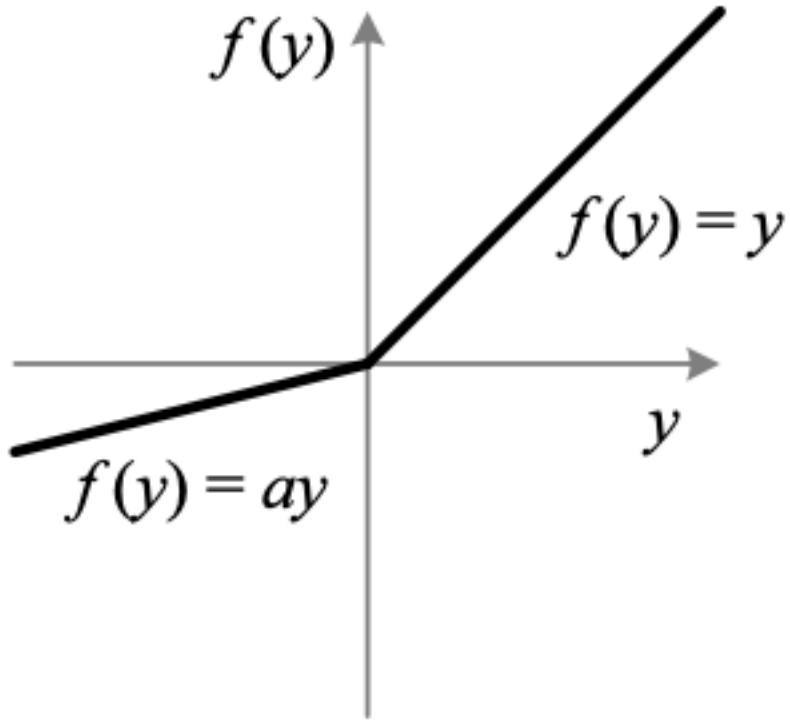


ReLU Activation



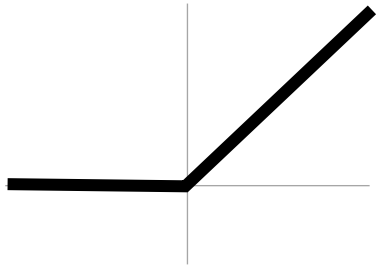
PRReLU Activation





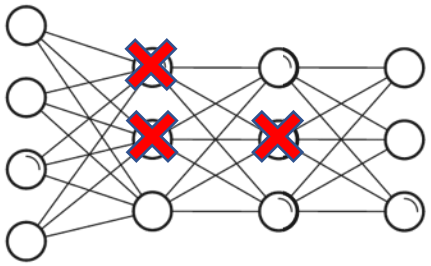
Neural Network Tips & Tricks

Optimization



Activation Functions

Managing Weights



Dropout

Managing Training

Weight Initialization

- Long history of initialization tricks for W_j based on "fan in" d_{in}
  - Here, d_{in} is the dimension of the input of layer W_j
  - **Intuition:** Keep initial layer inputs $z^{(j)}$ in the "linear" part of sigmoid
  - **Note:** Initialize intercept term to 0

- **Kaiming initialization (also called "He initialization")**
  - For ReLU activations, use $W_j \sim N\left(0, \frac{2}{d_{\text{in}}}\right)$

- **Xavier initialization**
  - For tanh activations, use $W_j \sim N\left(0, \frac{1}{d_{\text{in}}+d_{\text{out}}}\right)$ (d_{out} is output dimension)

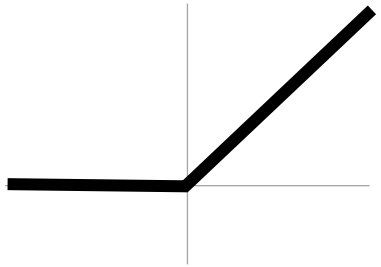
Batch Normalization

- **Problem**
  - During learning, the distribution of inputs to each layer are shifting (since the layers below are also updating)
  - This “covariate shift’’ slows down learning

- **Solution**
  - As with feature standardization, standardize inputs to each layer to $N(0, I)$
  - **Batch norm:** Compute mean and standard deviation of current minibatch and use it to normalize the current layer $z^{(j)}$ (this is differentiable!)
  - **Note:** Needs nontrivial mini-batches or will divide by zero
  - Apply after every layer (before or after activation; after can work better)

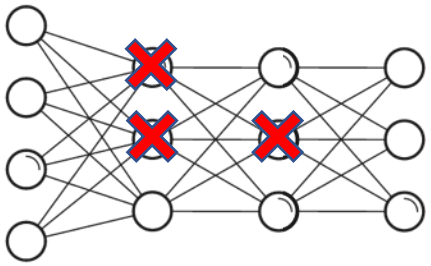
Neural Network Tips & Tricks

Optimization



Activation Functions

Managing Weights



Dropout

Managing Training

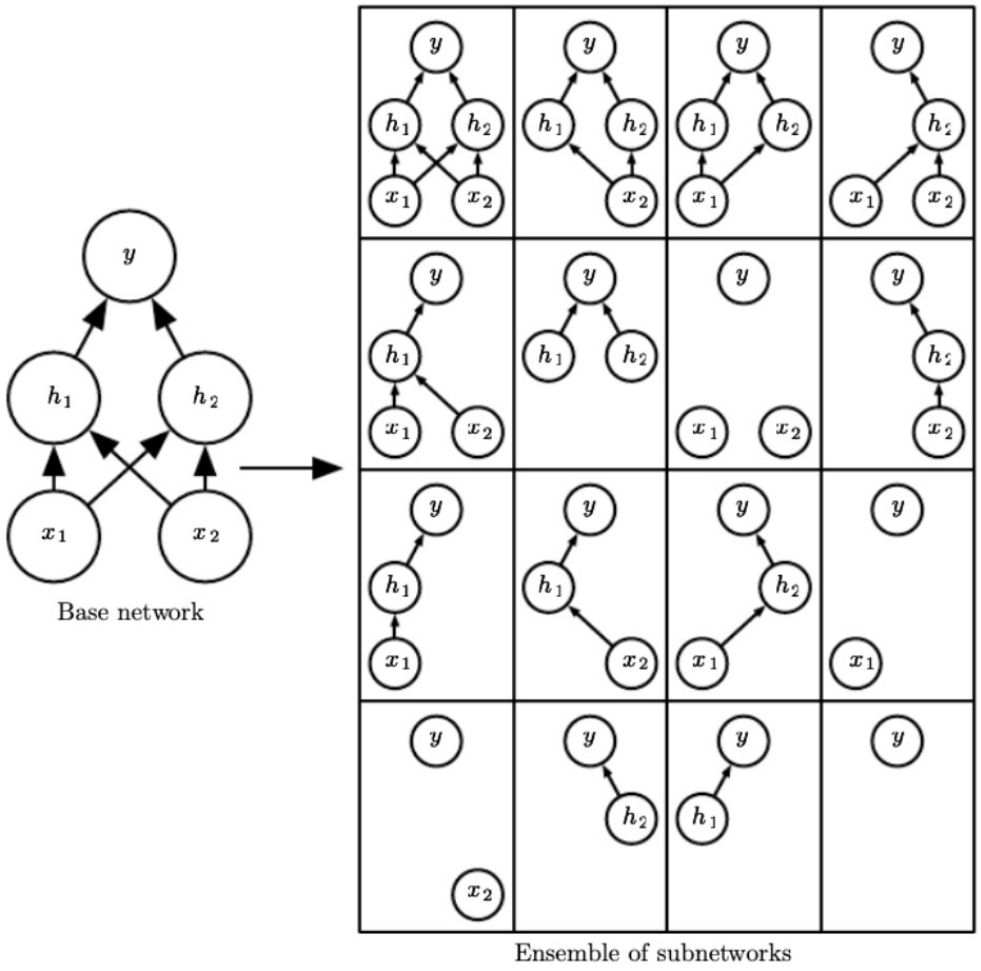
Dropout

- **Idea:** During training, randomly "drop" (i.e., zero out) a fraction p of the neurons $z_i^{(j)}$ (usually take $p = \frac{1}{2}$)

- Implemented as its own layer

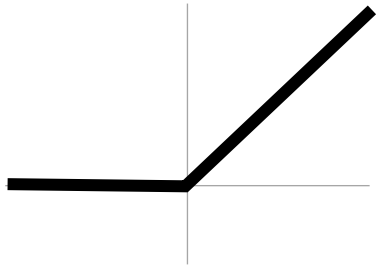
$$\text{Dropout}(z) = \begin{cases} z & \text{with prob. } p \\ 0 & \text{otherwise} \end{cases}$$

- Usually include it at a few layers just before the output layer



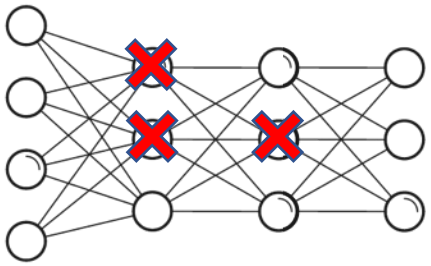
Neural Network Tips & Tricks

Optimization



Activation Functions

Managing Weights

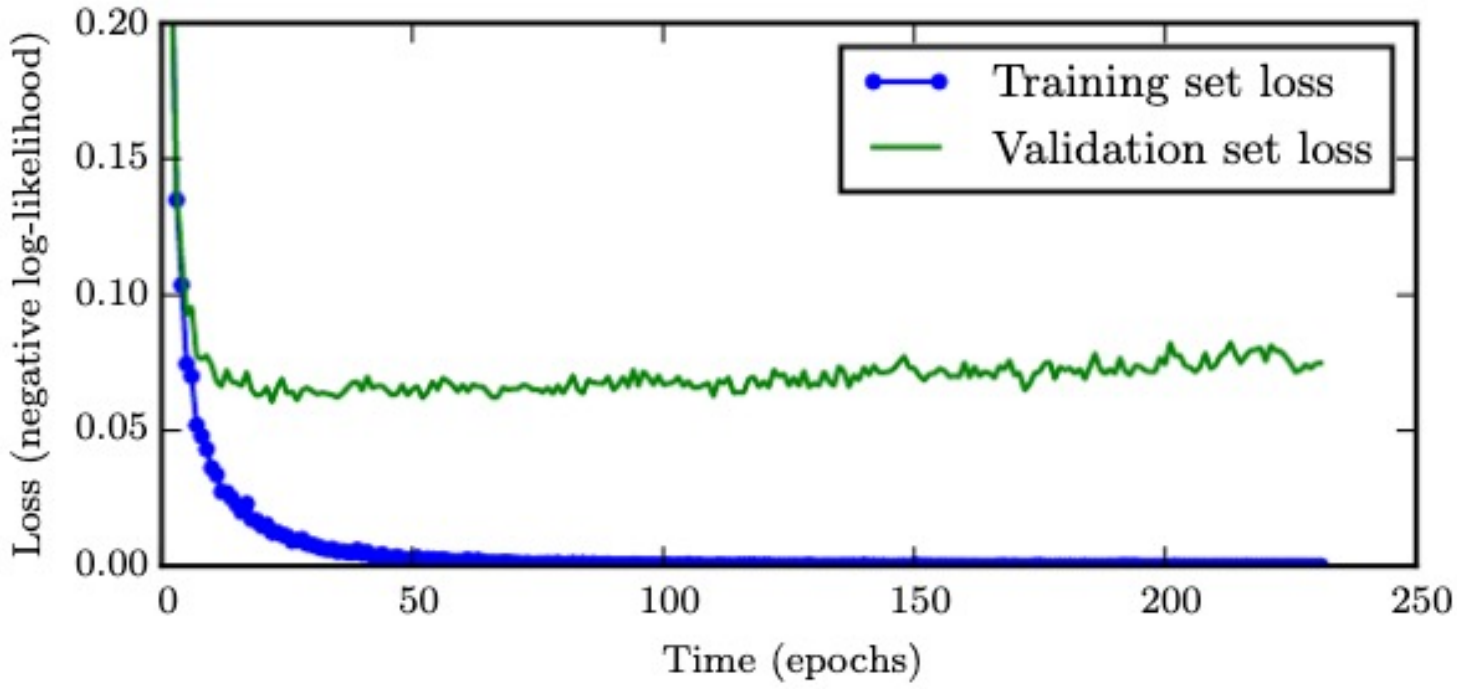


Dropout

Managing Training

Early Stopping

- Stop when your validation loss starts increasing (alternatively, finish training and choose best model on validation set)
  - Simple way to introduce regularization



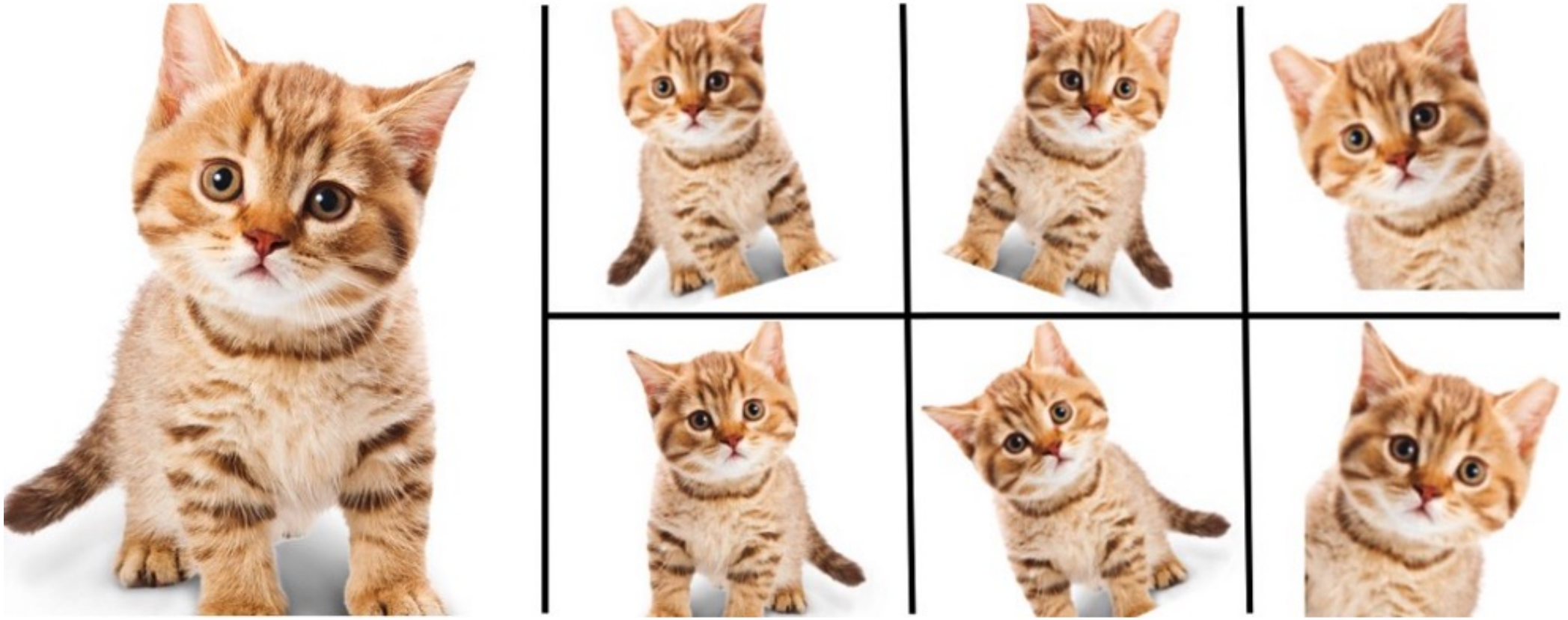
Data Augmentation

- **Data augmentation:** Generate more data by modifying training inputs

- Often used when you know that your output is robust to some transformations of your data
  - **Image domain:** Color shifts, add noise, rotations, translations, flips, crops
  - **NLP domain:** Substitute synonyms, generate examples (doesn't work as well but ongoing research direction)
  - Can combine primitive shifts

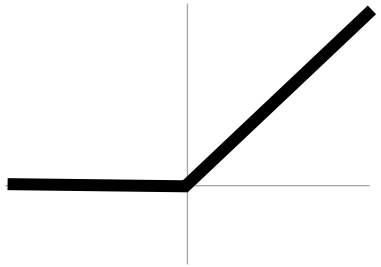
- **Note:** Labels are simply the label of original image

Data Augmentation



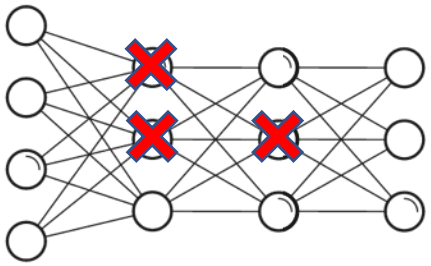
Neural Network Tips & Tricks

Optimization



Activation Functions

Managing Weights



Dropout

Managing Training

Neural Network Tips & Tricks

- **Neural networks**
  - Design the model family
  - Backpropagation to compute gradient

- **Optimization**
  - Gradient descent
  - Momentum
  - Adaptive step sizes
  - Learning rate schedules
  - Initialize weights properly

Neural Network Tips & Tricks

- **Layers**
  - Use ReLU activations to avoid vanishing gradients
  - Use batch normalization at all layers to avoid “covariate shift”
  - Use dropout at last few layers for regularization

- **Regularization**
  - Use early stopping (or choose best model on validation set)
  - Use data augmentation if possible

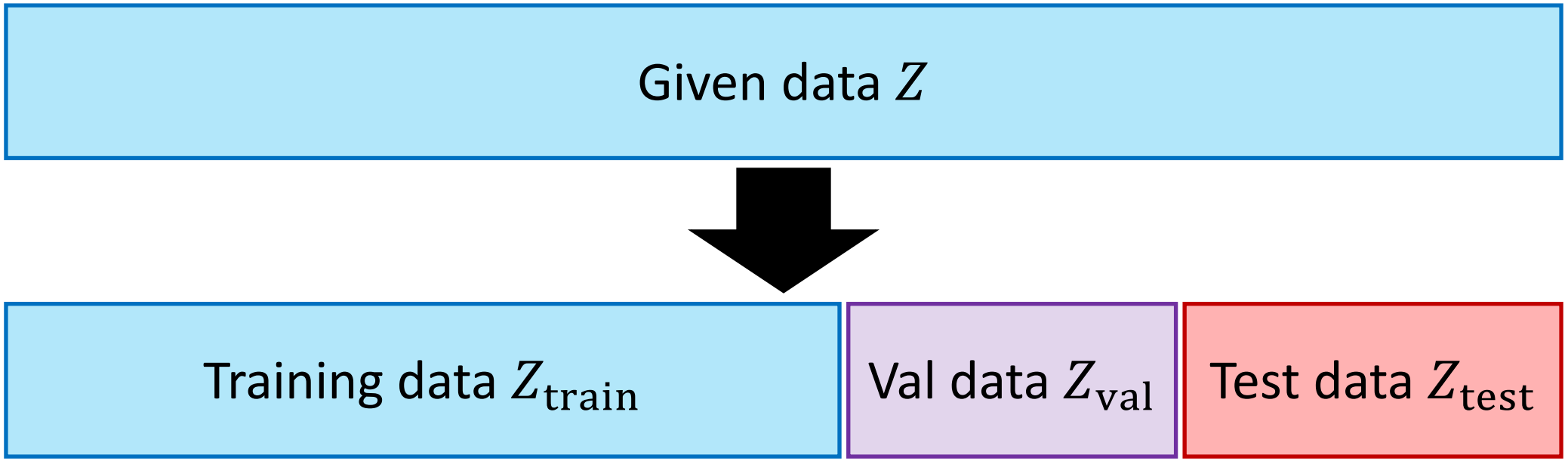
- Lots of hyperparameters! How to tune?

Hyperparameteter Choices

- **Architecture:** Stick close to tried-and-tested architectures (esp. for images)
- **SGD variant:** Adam, second choice SGD + 0.9 momentum
- **Learning rate:** 3e-4 (Adam), 1e-4 (for SGD + momentum)
- **Learning rate schedule**: Divide by 10 every time training loss stagnates
- **Weight initialization**: “Kaiming” initialization (scaled Gaussian)
- **Activation functions**: ReLU
- **Regularization**: BatchNorm (& cousins), L2 regularization + Dropout on some or all fully connected layers
- **Hyperparameter Optimization**: Random sampling (often uniform on log scale), coarse to fine

Hyperparameter Optimization

- **Recall:** Use cross-validation to tune hyperparameters!
  - Typically use one held-out validation set for computational tractability
  - E.g., 60/20/20 split
  - Can use smaller validation/test sets if you have a very large dataset



Hyperparameter Optimization Tips

- Keep the number of hyperparameters as small as possible
  - **Most important:** Learning rate

- **Strategy:** Automatically search over grid of hyperparameters and choose the best one on the validation set
  - Easy to parallelize across many machines
  - Record hyperparameters of all runs carefully!
  - Use the same random seeds for all runs

UNIVERSITY OF KWAZULU-NATAL
INYUVESI YAKWAZULU-NATALI

Quantum Research Group

Stellenbosch UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

NITheCS
National Institute for
Theoretical and Computational Sciences

nicis
NATIONAL INTEGRATED
CYBERINFRASTRUCTURE SYSTEM

SA QuTI
South African
Quantum
Technology
Initiative

Part B: Applications of ML to (Quantum) Physics

Spring School on Theoretical and Computational Foundations of Quantum Technologies

(24 - 28 October 2023, Alpine Heath Resort, Northern Drakensberg)

Physics Reports 810 (2019) 1–124

Contents lists available at ScienceDirect

Physics Reports

journal homepage: www.elsevier.com/locate/physrep

A high-bias, low-variance introduction to Machine Learning for physicists

Pankaj Mehta [a], Marin Bukov [b,*], Ching-Hao Wang [a], Alexandre G.R. Day [a], Clint Richardson [a], Charles K. Fisher [c], David J. Schwab [d]

REVIEWS OF MODERN PHYSICS

Recent Accepted Authors Referees Search Press About Editorial Team

Machine learning and the physical sciences*

Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová

Rev. Mod. Phys. **91**, 045002 – Published 6 December 2019

Article References Citing Articles (712) PDF HTML Export Citation

Quantum Physics

[Submitted on 8 Apr 2022 (v1), last revised 24 Jun 2022 (this version, v2)]

Modern applications of machine learning in quantum sciences

Anna Dawid, Julian Arnold, Borja Requena, Alexander Gresch, Marcin Płodzień, Kaelan Donatella, Kim A. Nicoli, Paolo Stornati, Rouven Koch, Miriam Büttner, Robert Okuła, Gorka Muñoz-Gil, Rodrigo A. Vargas-Hernández, Alba Cervera-Lierta, Juan Carrasquilla, Vedran Dunjko, Marylou Gabrié, Patrick Huembeli, Evert van Nieuwenburg, Filippo Vicentini, Lei Wang, Sebastian J. Wetzel, Giuseppe Carleo, Eliška Greplová, Roman Krems, Florian Marquardt, Micha Tomza, Maciej Lewenstein, Alexandre Dauphin

In these Lecture Notes, we provide a comprehensive introduction to the most recent advances in the application of machine learning methods in quantum sciences. We cover the use of deep learning and kernel methods in supervised, unsupervised, and reinforcement learning algorithms for phase classification, representation of many-body quantum states, quantum feedback control, and quantum circuits optimization. Moreover, we introduce and discuss more specialized topics such as differentiable programming, generative models, statistical approach to machine learning, and quantum machine learning.

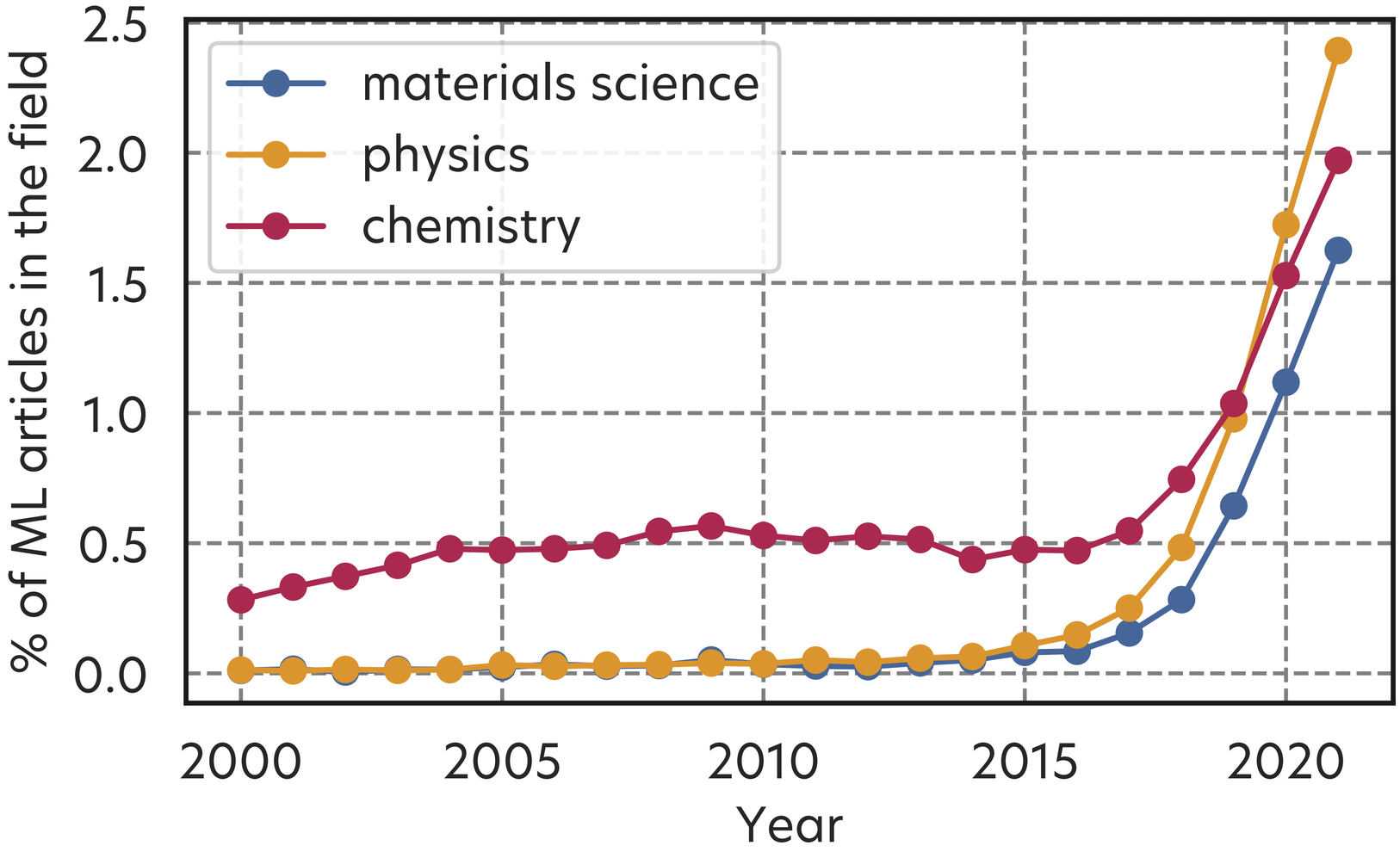
Comments: 283 pages, 91 figures. Comments and feedback are very welcome. Figures and tex files are available at this https URL

Subjects: **Quantum Physics (quant-ph)**; Disordered Systems and Neural Networks (cond-mat.dis-nn); Mesoscale and Nanoscale Physics (cond-mat.mes-hall)

Cite as: arXiv:2204.04198 **[quant-ph]**

(or arXiv:2204.04198v2 **[quant-ph]** for this version)

https://doi.org/10.48550/arXiv.2204.04198



The number of ML-based publications in physics, materials science, and chemistry is growing exponentially. Adapted from B. Blaiszik, Charting ML publications in science, GitHub repository (2021)

Possible application of ML to Quantum Physics

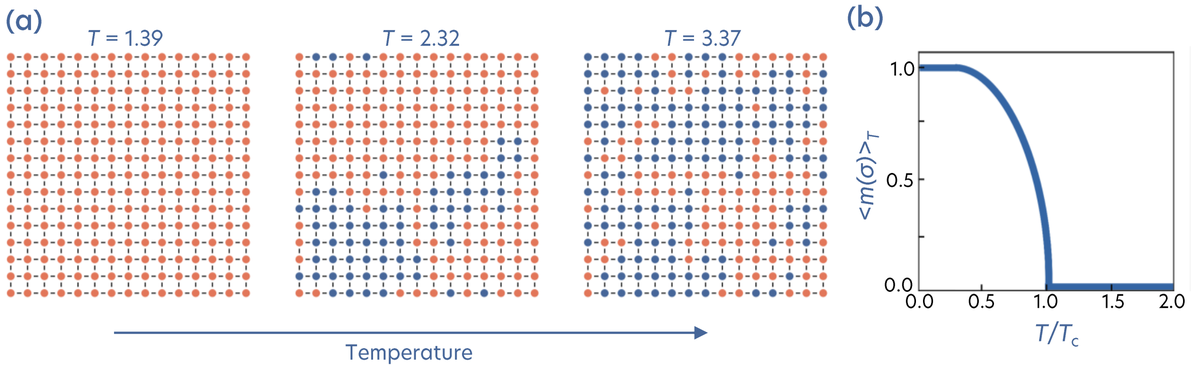
1. Phase classification
2. Neural Network Quantum States
3. Reinforcement Learning (+QECC in between)
4. Some of ML related activities at CQT-Durban

2D Ising Model: Supervised Approach

$$H(\boldsymbol{\sigma}) = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j,$$

$$\sigma_k \in \{+1, -1\}$$

$$T_c = \frac{2J}{k_B \ln\left(1+\sqrt{2}\right)},$$



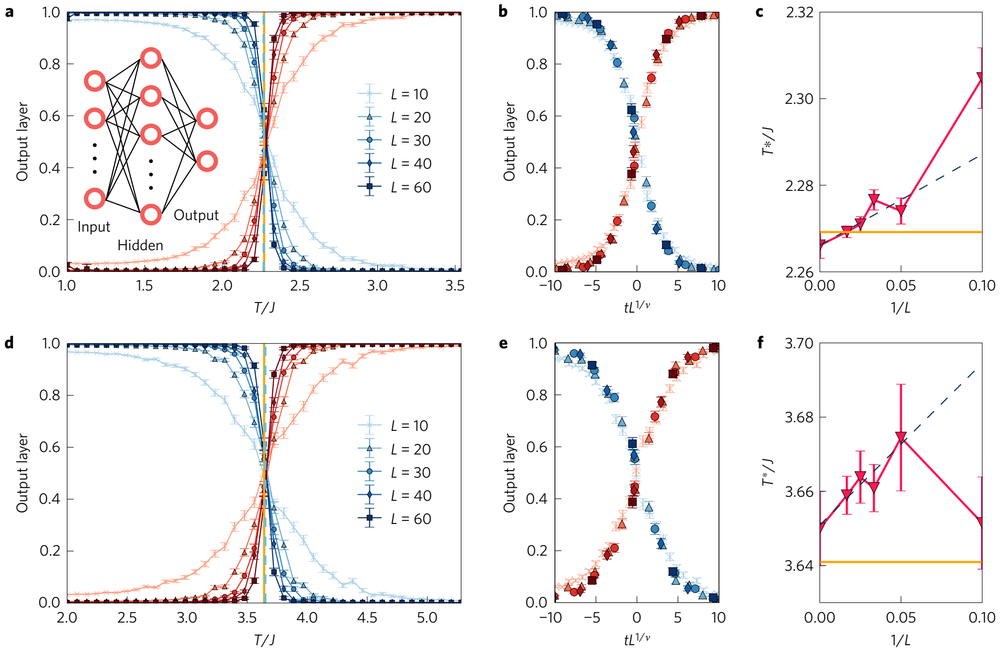


Figure 1 | Machine learning the ferromagnetic Ising model. a, The output layer averaged over a test set as a function of T/J for the square-lattice ferromagnetic Ising model. The inset in **a** displays a schematic of the fully connected neural network used in our simulations. **b**, Plot showing data collapse of the average output layer as a function of $tL^{1/\nu}$, where $t=(T-T_c)/J$ is the reduced temperature. Linear system sizes $L=10, 20, 30, 40$ and 60 are represented by crosses, up triangles, circles, diamonds and squares, respectively. **c**, Plot of the finite-size scaling of the crossing temperature T^*/J (down triangles). **d**–**f**, Analogous data to **a**–**b**, but for the triangular Ising ferromagnet using the neural network trained for the square-lattice model. The vertical orange lines signal the critical temperatures of the models in the thermodynamic limit, $T_c/J=2/\ln(1+\sqrt{2})$ for the square lattice[17] and $T_c/J=4/\ln 3$ for the triangular lattice[19]. The dashed vertical lines represent our estimates of T_c/J from finite-size scaling. The error bars represent one standard deviation statistical uncertainty (see Supplementary Information).

- J. Carrasquilla and R. G. Melko, *Machine learning phases of matter*, Nat. Phys. **13**(5), 431 (2017), doi:10.1038/nphys4035.

Ising Gauge Theory: Supervised Approach

global order parameter $H(\sigma) = -J \sum_{p} \prod_{i \in p} \sigma_i,$

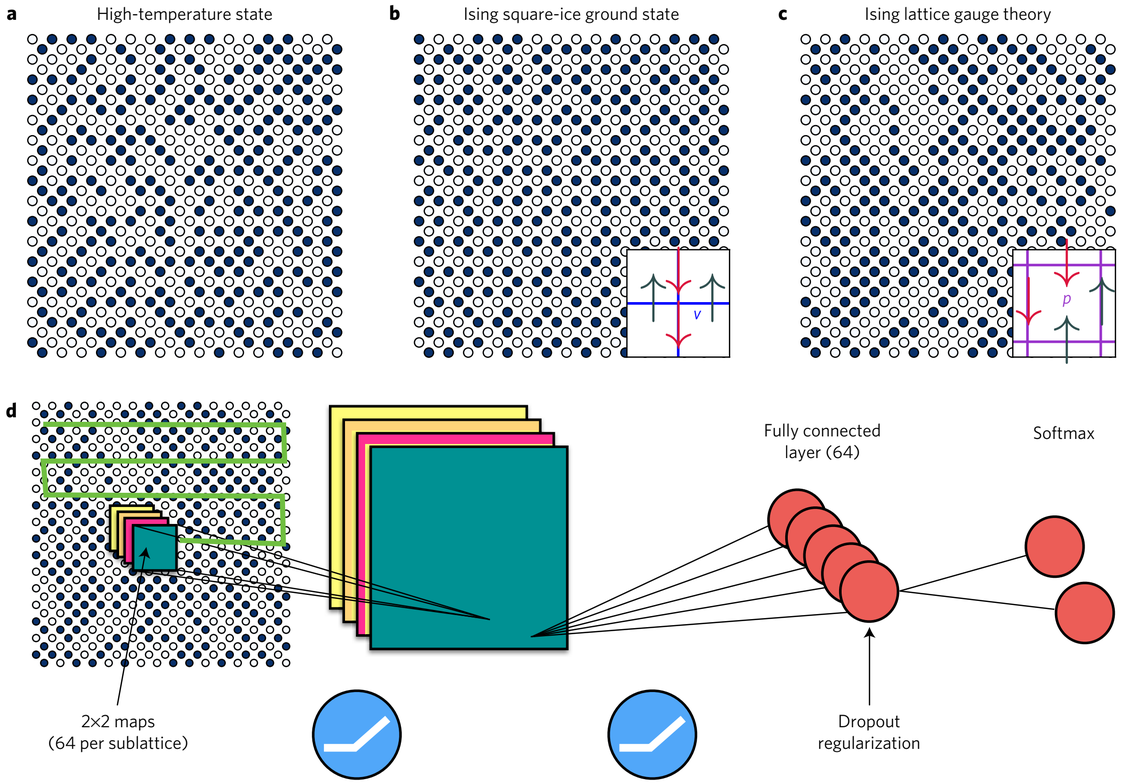


Figure 2 | Typical configurations of square-ice and Ising gauge models. a, A high-temperature state. **b**, A ground state of the square-ice Hamiltonian. **c**, A ground state configuration of the Ising lattice gauge theory. Dark circles represent spins up, while white circles represent spins down. The vertices and plaquettes defining the models are shown in the insets of **b** and **c**. **d**, Illustration of the convolutional neural network of the Ising gauge theory. The convolutional layer applies 64 2×2 filters to the configuration on each sublattice, followed by rectified linear units (ReLu). The outcome is followed by a fully connected layer with 64 units and a softmax output layer. The green line represents the sliding of the maps across the configuration.

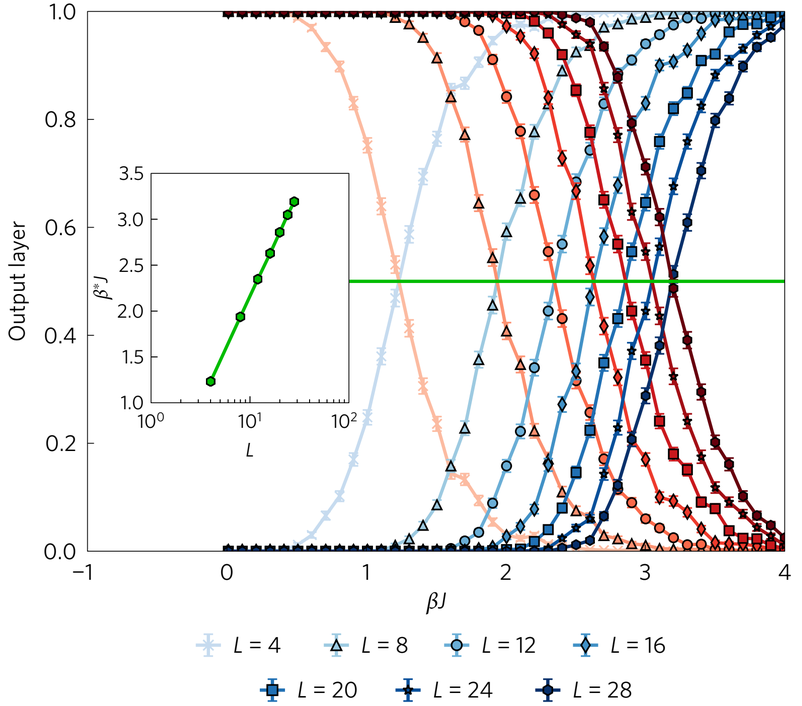


Figure 3 | Detecting the logarithmic crossover temperatures in the Ising gauge theory. Output neurons for different system sizes averaged over test sets versus βJ. Linear system sizes $L = 4, 8, 12, 16, 20, 24$ and 28 are represented by crosses, up triangles, circles, diamonds, squares, stars and hexagons. The inset displays $\beta^* J$ (octagons) versus L in a semilog scale. The error bars represent one standard deviation statistical uncertainty.

- J. Carrasquilla and R. G. Melko, *Machine learning phases of matter*, Nat. Phys. **13**(5), 431 (2017), doi:10.1038/nphys4035.

2D Ising & IGT: Unsupervised Approach

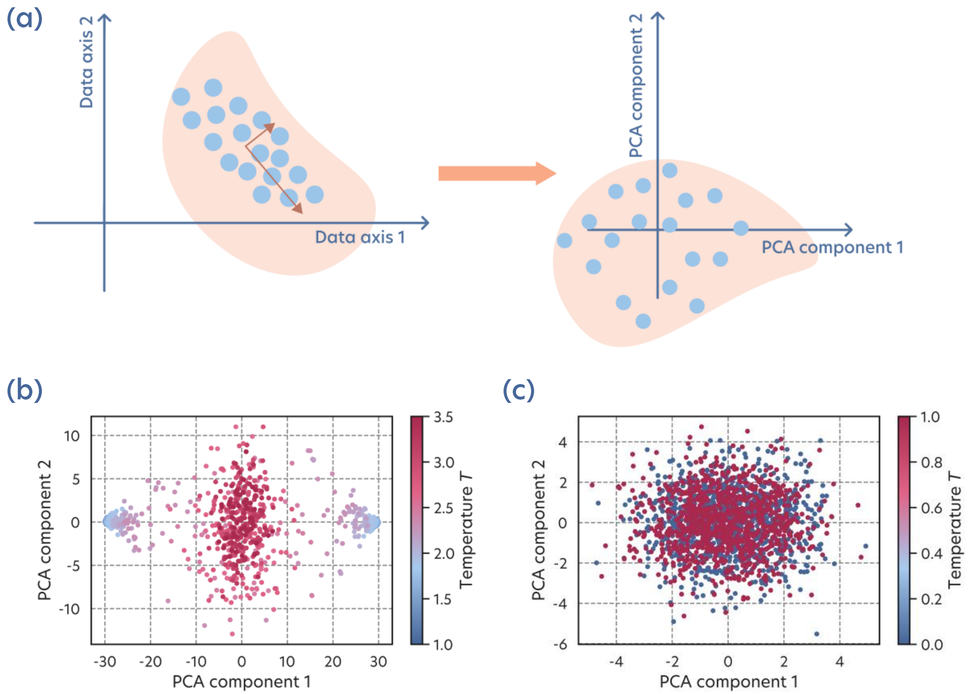


Figure 3.3: (a) Illustration of the principle behind PCA applied to data points (blue) living in a two-dimensional feature space. Orange vectors denote the first two principal components on which we project the data (right panel). PCA applied to the spin configuration samples of (b) the Ising model and (c) IGT with $k = 2$. For the Ising model, the data consists of 50 spin configurations (linear lattice size $L = 30$) sampled using Monte Carlo methods at temperatures T ranging from $T_1 = 1$ to $T_{20} = 3.5$ in equidistant steps. For the IGT, the data consists of 1000 spin configurations (linear lattice size $L = 16$) drawn within the topological phase and the disordered phase at high temperature. Panels (b) and (c) reproduced from [32, Notebook A1].

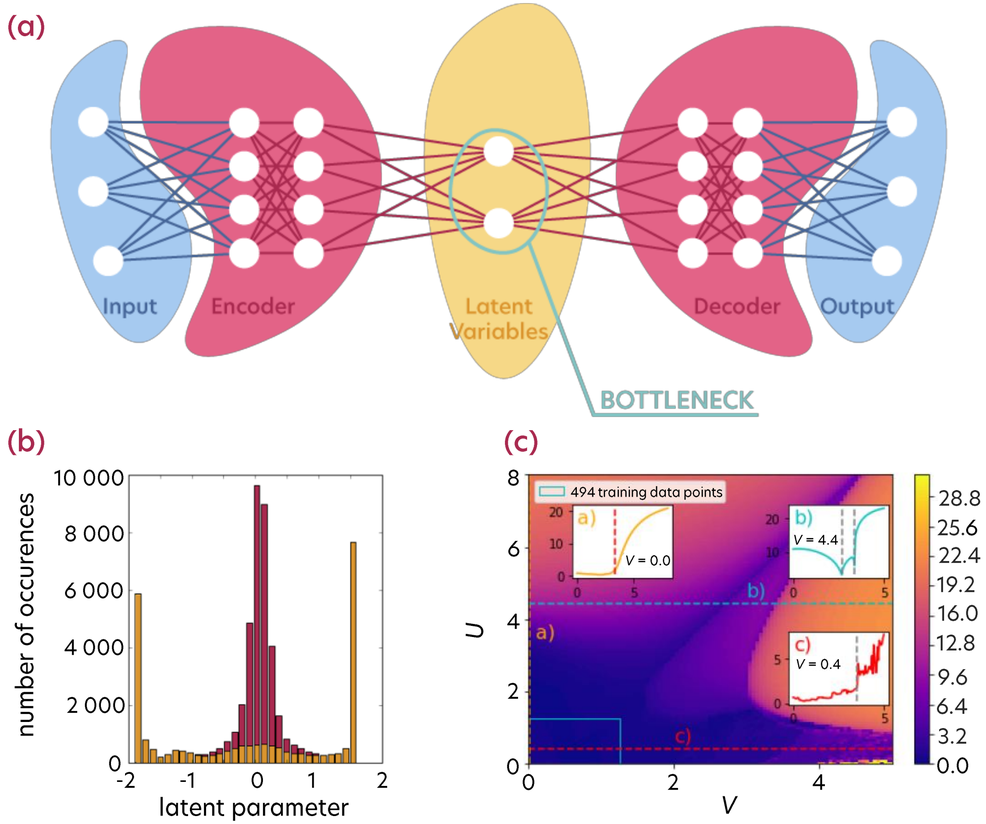


Figure 3.5: (a) Illustration of a natural bottleneck (here two neurons) in an AE architecture. (b) Analysis of bottleneck neurons of an AE trained to reconstruct spin configurations of a two-dimensional Ising model. Latent representation of Ising configurations clusters into two phases visible as a histogram. (c) Anomaly detection scheme allows for the recovery of the phase diagram from the reconstruction loss of an AE trained on one phase (blue box in bottom left). Panel (b) is taken from Ref. [106], (c) from Ref. [116].

Learning by Confusion

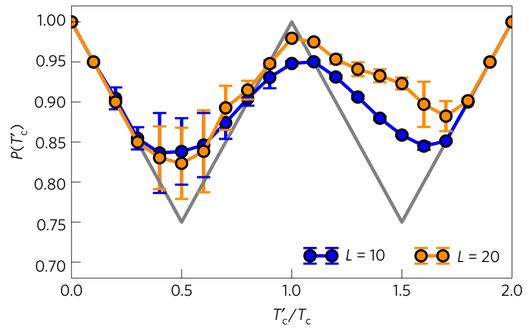


Figure 3 | Learning the Ising transition. The position of the middle peak in the universal W-shape deviates from $T_c' = T_c$ for $L = 10$ due to the finite-size effect. Here $k_B T_c \approx 2.27J$ is the exact transition temperature in the thermodynamic limit. For $L = 20$ the middle peak is located exactly at $T_c' = T_c$. Error bars are obtained by averaging over ten different and independent Monte Carlo runs for obtaining the data. The errors are larger for points that deviate from the expected W-shape. (Parameters for training: batch size $N_b = 100$, learning rate $\alpha = 0.02$ and regularization $l_2 = 0.005$. See the Methods for an explanation of these terms.)

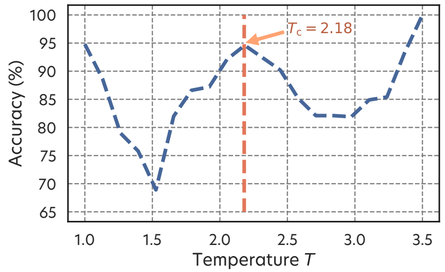


Figure 3.6: Result of the learning by confusion scheme applied to the Ising model. The data consists of 100 spin configurations (linear lattice size $L = 30$) sampled using Monte Carlo methods at temperatures T ranging from $T_1 = 1$ to $T_{20} = 3.5$ in equidistant steps. The data set is split into equally-sized training and test sets (such that 50 spin configurations are present at each sampled temperature). The blue curve shows the classification accuracy on the test set for various choices of bi-partitions. It has a characteristic W-shape whose middle peak is at $T \approx 2.3$, which is in good agreement with the Onsager solution. Reproduced with [32, Notebook A3].

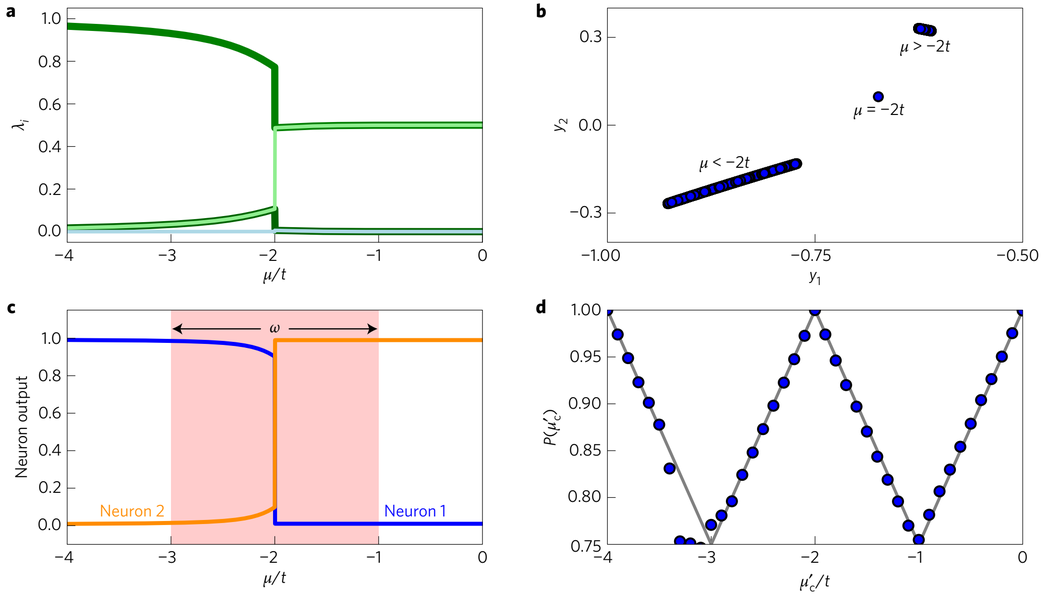


Figure 1 | Learning the topological phase transition in the Kitaev chain. a, Evolution of the entanglement spectrum as a function of the chemical potential μ. Here we plot the largest four eigenvalues of the reduced density matrix ρ_A. The degeneracy structure is clearly observable. **b**, Principal component analysis of the entanglement spectrum. All data points are shown in the plane of the first two principal components y_1 and y_2. **c**, Supervised learning with blanking. The shaded region is blanked out during the training phase, and the NN can still predict the correct transition point $\mu = -2t$. **d**, $P(\mu_c')$, evolution of the accuracy of prediction, as a function of the proposed critical point μ_c', which shows the universal W-shape. See text for more details. (Parameters for training: batch size $N_b = 100$, learning rate $\alpha = 0.075$ and regularization $l_2 = 0.001$. See the Methods for an explanation of these terms.)

- E.P.VanNieuwenburg,Y.-H.LiuandS.D.Huber, *Learning phase transitions by confusion*, Nat. Phys. **13**(5), 435 (2017), doi:10.1038/nphys4037.

Neural Network quantum states

Quantum state of N-spins, 2^N components

$$|\Psi\rangle = C_{\uparrow\uparrow\cdots\uparrow}|\uparrow\uparrow\cdots\uparrow\rangle + C_{\uparrow\uparrow\cdots\downarrow}|\uparrow\uparrow\cdots\downarrow\rangle + \cdots + C_{\downarrow\downarrow\cdots\downarrow}|\downarrow\downarrow\cdots\downarrow\rangle$$
$$= \sum_{\sigma_1,\sigma_2,\cdots,\sigma_N} C_{\sigma_1,\sigma_2,\ldots,\sigma_N}|\sigma_1\rangle \otimes |\sigma_2\rangle \otimes \cdots \otimes |\sigma_N\rangle\,,$$

The main idea behind variational methods is to find a computationally efficient representation of the physically relevant quantum states within the Hilbert space of interest.

$$|\Psi_{\boldsymbol{\theta}}\rangle = \sum_{s=1}^{2^N} \Psi_{\boldsymbol{\theta}}(\boldsymbol{s})|s\rangle,$$

where $\Psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \langle s|\Psi_{\boldsymbol{\theta}}\rangle$ denotes the probability amplitude corresponding to the state $|s\rangle$

Jastrow Wave-function

$$\Psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = e^{-\frac{1}{2}\sum_{i\neq j}\theta_{ij}\sigma_i\sigma_j}$$

Feed-Forward NN

$$\Psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = g^{(D)}\cdot\boldsymbol{W}^{(D)}\ldots g^{(2)}\cdot\boldsymbol{W}^{(2)}g^{(1)}\cdot\boldsymbol{W}^{(1)}\boldsymbol{s}\,.$$

Restricted Boltzmann machines

$$\Psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \sum_{\boldsymbol{h}} e^{\boldsymbol{b}_v^\dagger\boldsymbol{s}+\boldsymbol{b}_h^\dagger\boldsymbol{h}+\boldsymbol{h}^\dagger\boldsymbol{W}\boldsymbol{s}}$$

Autoregressive and recurrent NN

Neural Network quantum states

Restricted Boltzmann machines

$$\Psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \sum_{\boldsymbol{h}} e^{\boldsymbol{b}_v^\dagger \boldsymbol{s} + \boldsymbol{b}_h^\dagger \boldsymbol{h} + \boldsymbol{h}^\dagger \boldsymbol{W} \boldsymbol{s}}$$

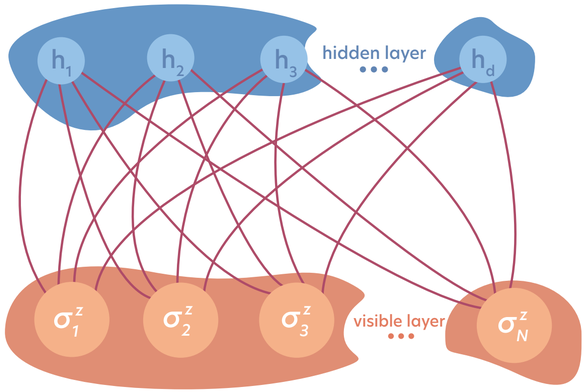


Figure 5.1: Pictorial representation of a restricted Boltzmann machine (RBM) that represents the wave function of an N-spin system, with $\boldsymbol{s} = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ and $\boldsymbol{h} = (h_1, h_2, \ldots, h_d)$ the hidden units.

$$\mathcal{H}_{\text{TFI}} = -h \sum_i \sigma_i^x - \sum_{ij} \sigma_i^z \sigma_j^z$$

$$\mathcal{H}_{\text{AFH}} = \sum_{ij} \sigma_i^x \sigma_j^x + \sigma_i^y \sigma_j^y + \sigma_i^z \sigma_j^z$$

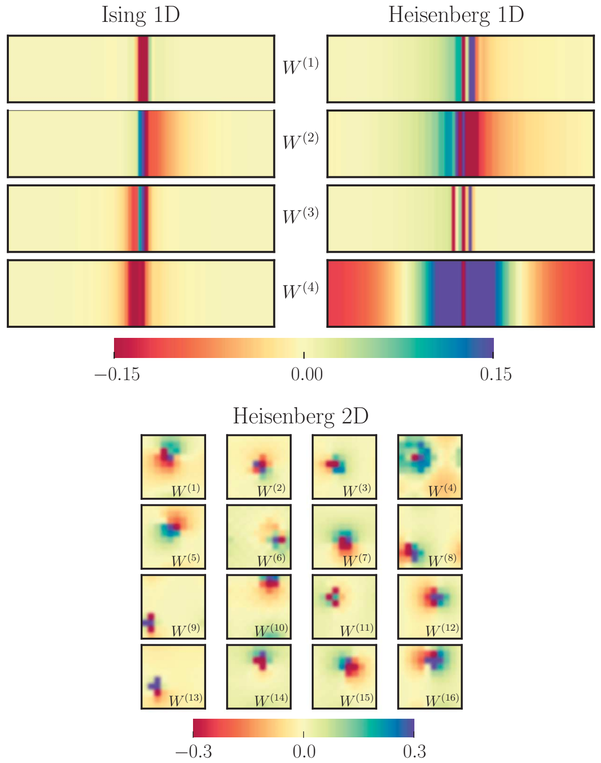


Fig. 2. Neural-network representation of the many-body ground states. Results for prototypical spin models in one and two dimensions are shown. In the top group of panels, we show the feature maps for the 1D transverse-field Ising (TFI) model at the critical point h = 1, as well as for the antiferromagnetic Heisenberg (AFH) model. In both cases, the hidden-unit density is α = 4 and the lattices comprise 80 sites. Each horizontal colormap shows the values that the fth feature map $W_j^{(f)}$ takes on the jth lattice site (horizontal axis, broadened along the vertical direction for clarity). In the bottom group of panels, we show the feature maps for the 2D Heisenberg model on a square lattice, for α = 16. In this case, the horizontal (or vertical) axis of the colormaps corresponds to the x (or y) coordinates on a 10-by-10 square lattice. Each of the feature maps acts as an effective filter on the spin configurations, capturing the most important quantum correlations.

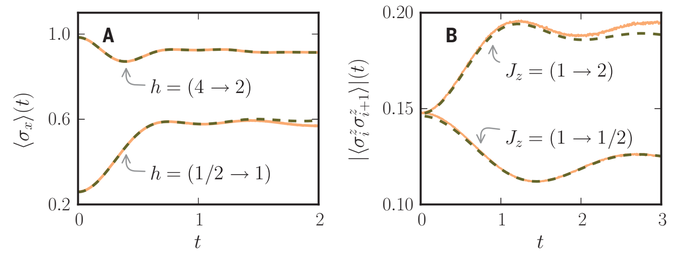


Fig. 4. Many-body unitary time evolution with NQS. NQS results (solid lines) for the time evolution induced by a quantum quench in the microscopic parameters of the models we study (the transverse field h for the TFI model and the coupling constant J_z in the AFH model) are shown. (**A**) Time-dependent transverse spin polarization in the TFI model, compared to exact results (dashed lines). (**B**) Time-dependent nearest-neighbors spin correlations in the AFH model, compared to exact numerical results obtained with t-DMRG (dashed lines). All results refer to 1D chains representative of the thermodynamic limit, with finite-size corrections smaller than the line widths.

The *variational principle* states that given an Hamiltonian $\hat{H}$, the energy $E(\theta)$ of a variational wave function $|\Psi_\theta\rangle$ is greater or equal than the exact ground state energy, i.e.,

$$E(\boldsymbol{\theta}) = \frac{\langle \Psi_{\boldsymbol{\theta}} | \hat{H} | \Psi_{\boldsymbol{\theta}} \rangle}{\langle \Psi_{\boldsymbol{\theta}} | \Psi_{\boldsymbol{\theta}} \rangle} \geq E_0. \tag{5.28}$$

Therefore the energy is a valid loss function, as the *lower* the expectation value of the energy, *the better the approximation is*[a].

$$R(\dot{\mathcal{W}}(t)) = \text{dist}(\partial_t \Psi, -i\mathcal{H}\Psi)$$

- G. Carleo and M. Troyer, *Solving the quantum many-body problem with artificial neural networks*, Science **355**(6325), 602–606 (2017), doi:10.1126/science.aag2302.

Neural Network quantum states

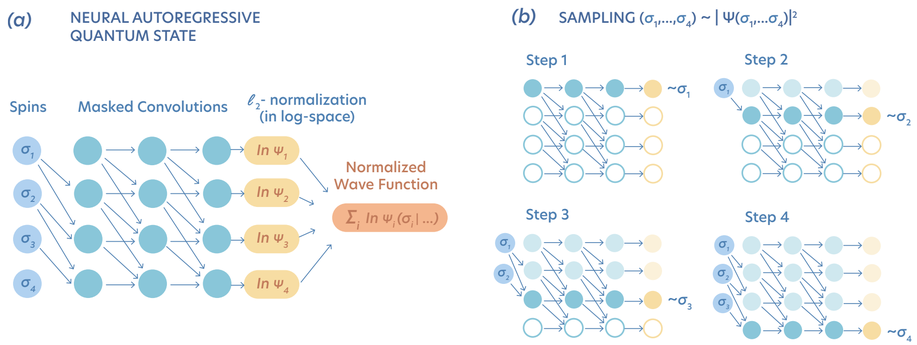


Figure 5.2: Example of an ARNN quantum state for four spins. (a) Pictorial representation of the network. The arrows representing the weights of the model are skewed in order not to break the conditional structure of the output probability distribution. These layers are "masked", due to some connections being deleted. (b) Sampling algorithm. One samples consecutive spins using direct sampling on the conditional probabilities at each step. Adapted from [83].

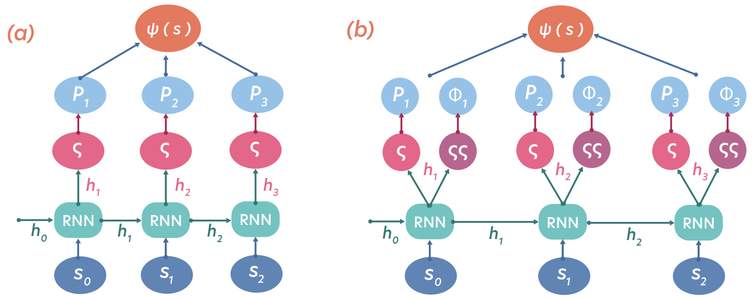


Figure 5.3: Pictorial representation of an RNN architecture for NQS. Panel (a) is for real-valued wave functions, which can be relevant for a certain class of problems, and panel (b) is for complex-valued wave functions. In both schemes, a local spin configuration s_i and a hidden vector h_i are fed into an RNN cell, which performs a nonlinear transformation. Then an activation function (ς, for softmax and/or $\varsigma\varsigma$, for softsign) is applied to obtain the final probability and/or phase corresponding to the configuration. At the end, the probabilities (and phases) are combined to obtain the final wave function amplitudes $\psi(s)$.

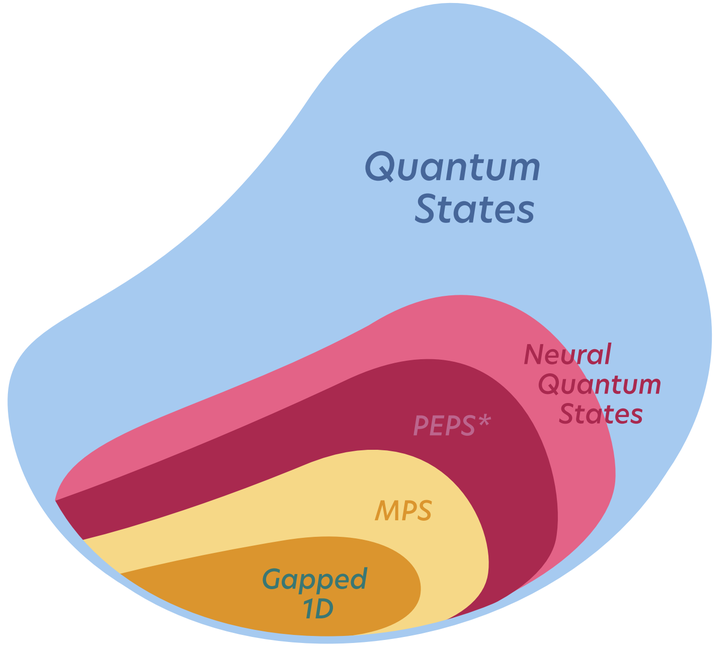


Figure 5.4: Expressive capacity of different classes of variational states, as explicitly proven in Ref. [230] by mapping TNSs to NQS. PEPS* refers to a sub-class of projected entangled pair states, a generalization of MPSs. Adapted from [230].

Neural Network quantum states: OQS

$$\partial_t \hat{\rho} = -\mathrm{i}[\hat{H}, \hat{\rho}] + \sum_{i=1}^{D} \left(\hat{J}_i \hat{\rho} \hat{J}_i^\dagger - \frac{1}{2} \{ \hat{J}_i^\dagger \hat{J}_i, \hat{\rho} \} \right) \equiv \mathbf{L}[\hat{\rho}],$$

$$\left\langle s \middle| \hat{\rho} \middle| s' \right\rangle = \sum_{s'} \Psi(s, s') \Psi^*(s, s'),$$

$$\mathcal{L}(\boldsymbol{\theta}) = \langle \hat{\rho}_{\boldsymbol{\theta}} \mathbf{L}^\dagger \mathbf{L} \hat{\rho}_{\boldsymbol{\theta}} \rangle \, .$$

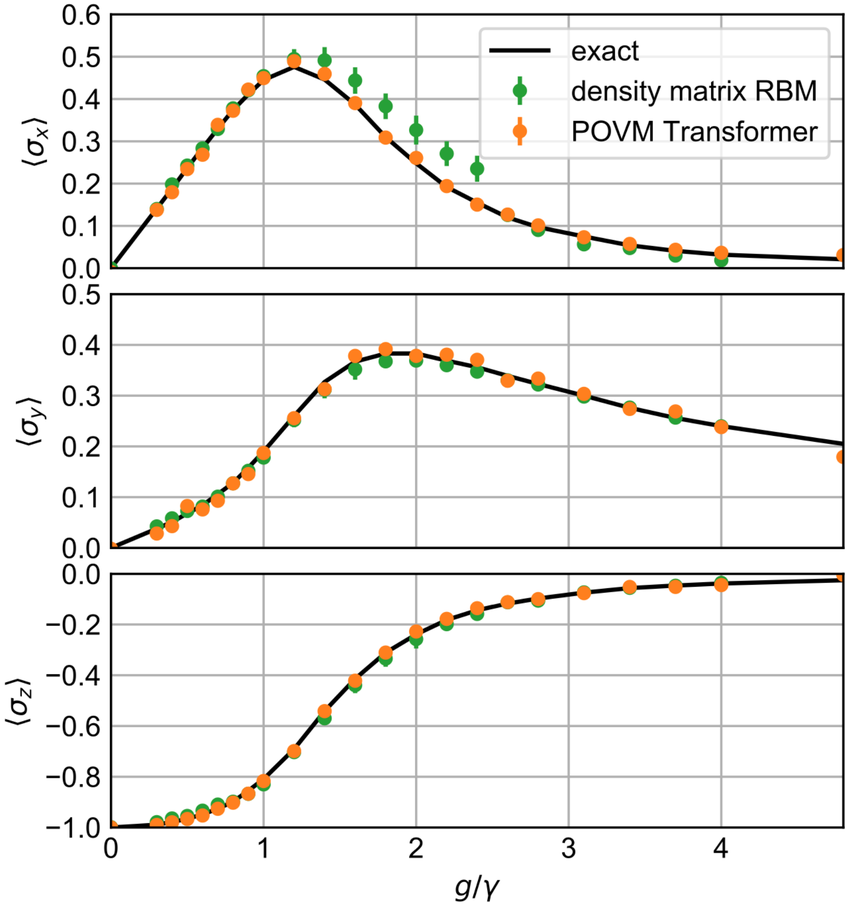


Figure 5.10: Expectation values of observables $\hat{\sigma}_k = 1/N \sum_i \hat{\sigma}_i^k$, $k \in \{x, y, z\}$ at the steady state of the open system described by the transverse-field Ising Hamiltonian and jump operators $\hat{J}_i = \sqrt{\gamma} \hat{\sigma}_i^-$ as a function of g/γ, with g the magnetic field strength. Results are shown for both the purified RBM approach and the POVM approach with a Transformer network. Taken from [268].

Neural Network quantum states: Quantum tomography

Quantum tomography is the task of reconstructing a quantum state from measurement data. Fully reconstructing a state is tedious as it requires an exponential number of measurements in the system size. This is a major problem for experimentalists who want to demonstrate quantum effects and protocols. Neural networks can once again help in this task, and they have been successfully applied to reconstruct non-trivial quantum states from a polynomial number of measurements.

Let us consider the task of reconstructing a wave function $|\psi\rangle$ from a limited number of snapshots $|\psi(s)|^2$ obtained by performing projective measurements in some basis spanned by $|s\rangle = |s_1, s_2, \ldots, s_N\rangle$, with s_i some local quantum numbers and N the size of the system. Then, the task, in the NQS language, is simply to minimize:

$$\min_{\boldsymbol{\theta}} \text{dist}\left(|\psi_{\boldsymbol{\theta}}\rangle, |\psi\rangle\right)$$

Kullback-Leibler (KL) divergence $$\mathrm{D}_{\mathrm{KL}}(p||q) = \sum_{x\in\mathcal{P}} p(x) \frac{\log p(x)}{\log q(x)}$$

$$\mathrm{D}_{\mathrm{KL}}(\boldsymbol{\theta}) = \sum_{B} \sum_{s\in S_B} |\psi^B_{\boldsymbol{\theta}}(s)|^2 \frac{\log |\psi^B_{\boldsymbol{\theta}}(s)|^2}{\log |\psi^B(s)|^2}$$

where S_B is the set of snapshots of the quantum state in basis B, and $\psi^B(s) = \langle s| \hat{U}_B |\psi\rangle$ with $\hat{U}_B$ a unitary operator. Then, gradients are found as usual, either with automatic differentiation or analytically with simple models such as RBMs.

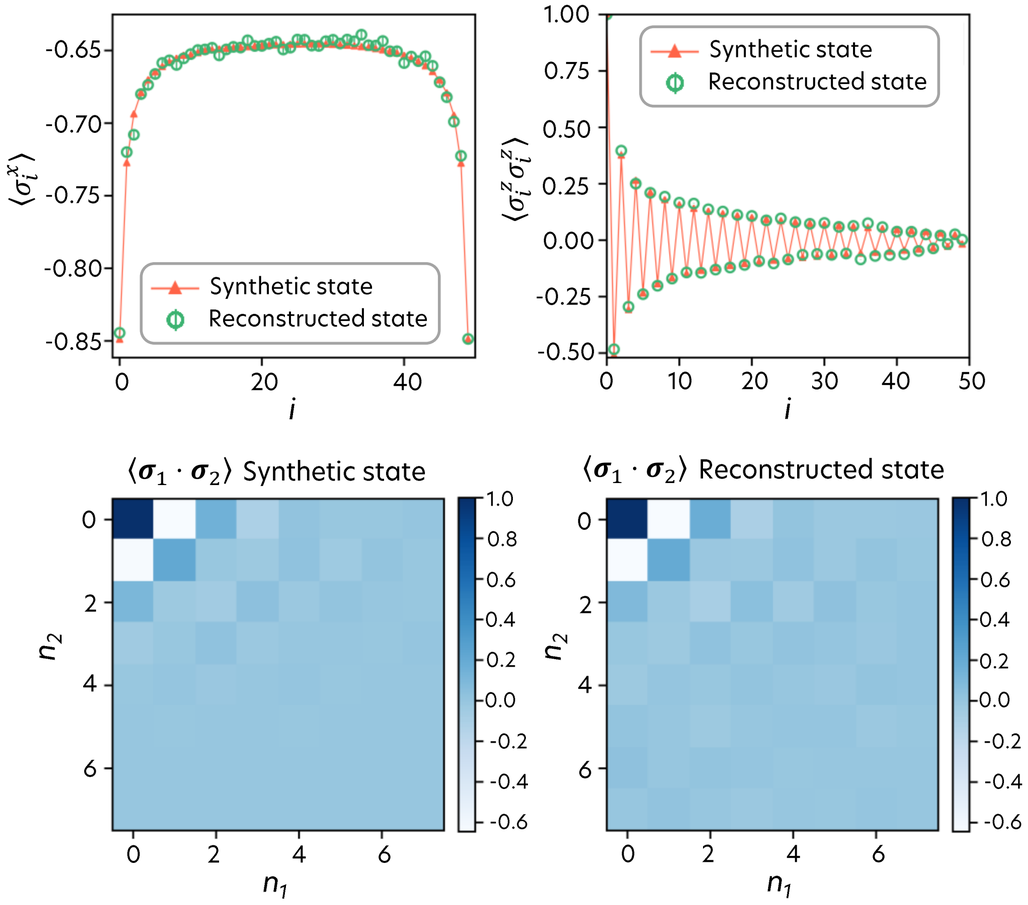


Figure 5.11: Various observables corresponding to the ground state of the Heisenberg model on a triangular lattice with $N = 50$ spins. (a) Average magnetization along x for each spins i. (b) Spin-spin correlation function between the spin at site 1 with spins at site i. (c) and (d) Average spin-spin correlation between the first and the i-th spin. One can see that all observables are reproduced with a very high precision. Adapted from Ref. [82].

Reinforcement Learning

The central objective of any RL problem is to learn the *optimal policy*, π^*, that maximizes the obtained rewards. A policy, π, dictates which actions to take given the observations, and thereby defines the strategy followed by the agent.

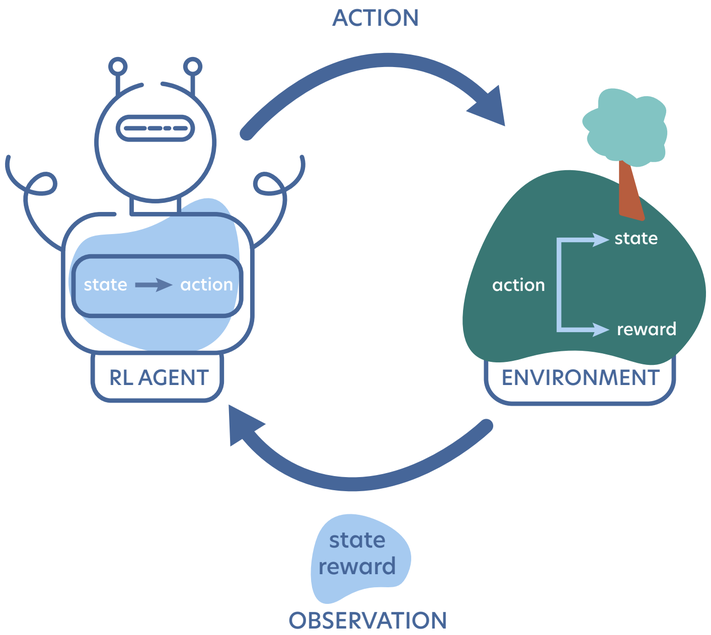


Figure 6.1: Overview of the basic RL setting. The agent receives an observation from the environment. Given the observation, it chooses the next action according to its policy. The environment determines the outcome of the action, and it returns an observation to the agent consisting of the new state and a potential reward.

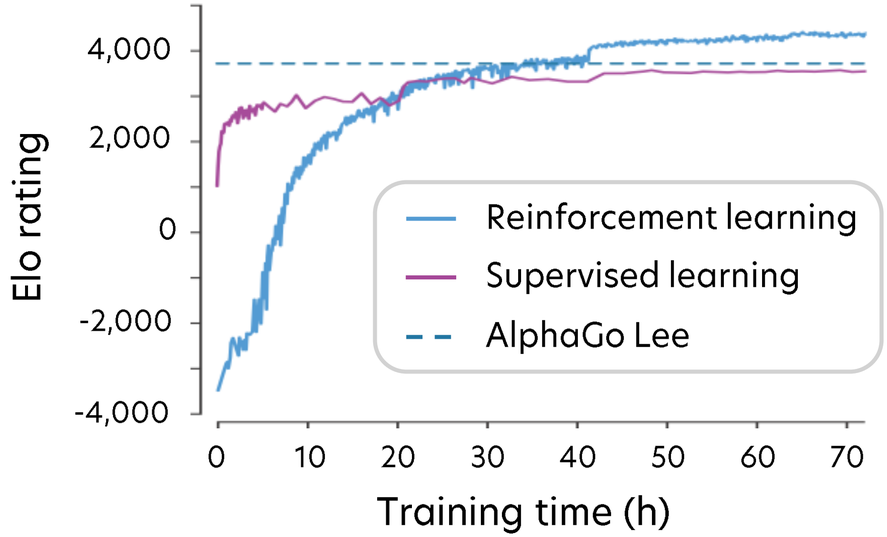


Figure 6.5: Performance comparison between AlphaGo (initial supervised learning) [20], and AlphaGo Zero (pure RL) [325] at the game of Go. Initially, AlphaGo has an advantage thanks to the initial supervised training. However, it limits its capabilities and it is quickly outperformed by AlphaGo Zero. The horizontal dashed line corresponds to the Elo rating of Lee Sedol during his match with AlphaGo being a reference point for the supervised/pure RL performance. Taken from [325].

Reinforcement Learning

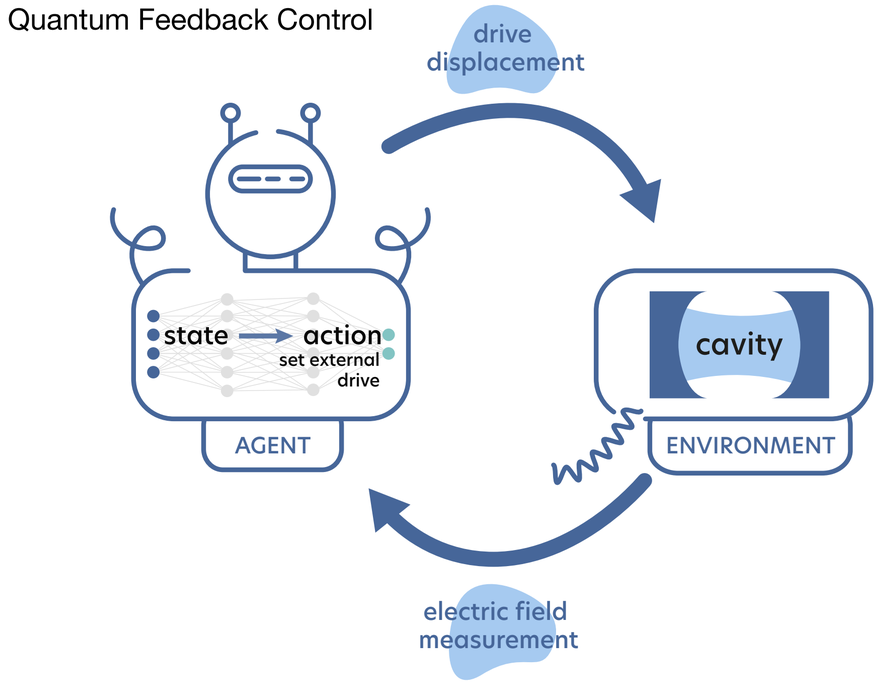


Figure 6.6: Driven single mode microcavity as an RL environment. The mode decays from the cavity. Its measurement serves as the observation for the agent represented by an NN. The network converts a measurement trace into probabilities for all the available actions, which give feedback for the displacement drive of the cavity. Adapted from [330].

The cavity mode is leaking from the cavity and this signal can be measured. Additionally, the cavity mode can be controlled via an external drive. The goal is to adjust the drive amplitude of a beam entering the cavity to create and then stabilize a cavity quantum state with a single photon as depicted in Fig. 6.6.

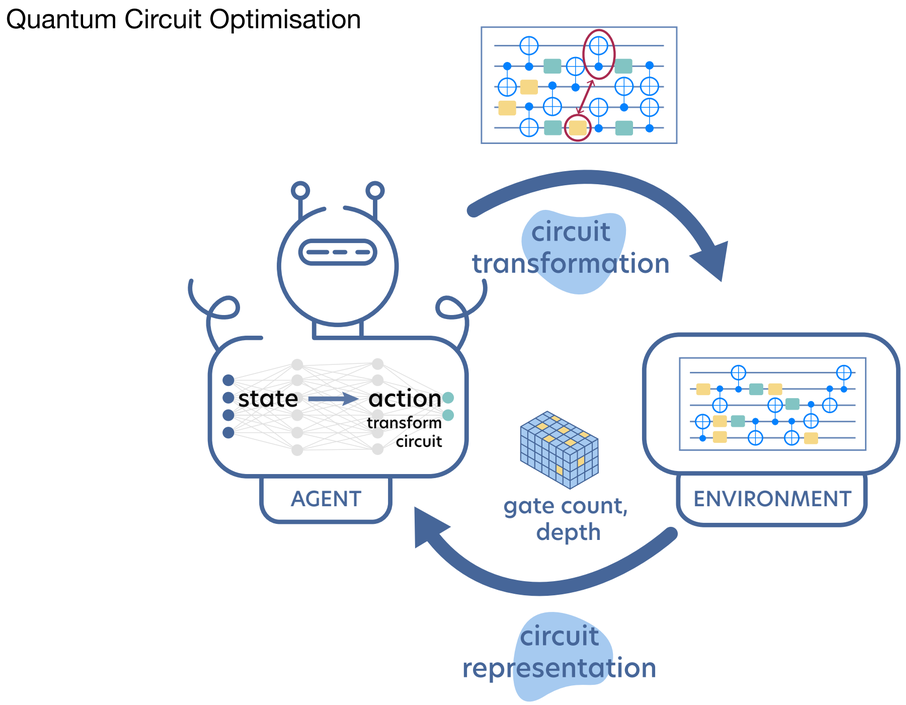


Figure 6.7: Schematic representation of the RL framework for circuit optimization. The agent observes a representation of a quantum circuit given by the environment. Then, it can choose to perform a modification to the circuit. The environment calculates a reward depending on the gate count (or another metric) of the resulting circuit, and it provides the agent with the new circuit and the reward. Adapted from [336].

The reward can account for various aspects, such as the reduction in the total gate count, the reduction in depth (the time needed for the circuit to run), or the combi- nation of both. Additionally, the reward function can also depend on a decoherence estimate for the whole circuit, based on the decoherence that happens on each the gates.

Reinforcement Learning

Quantum Error Correction

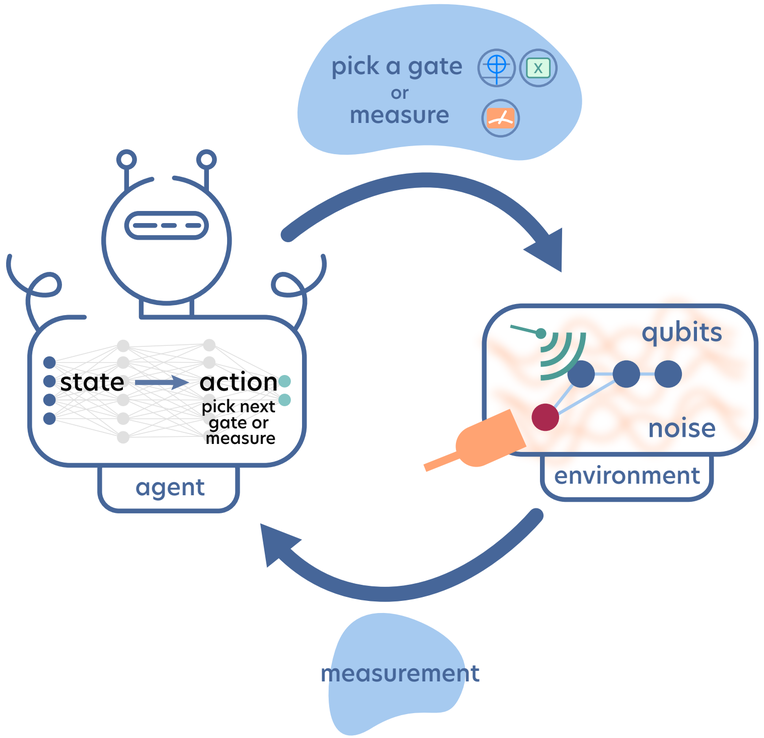
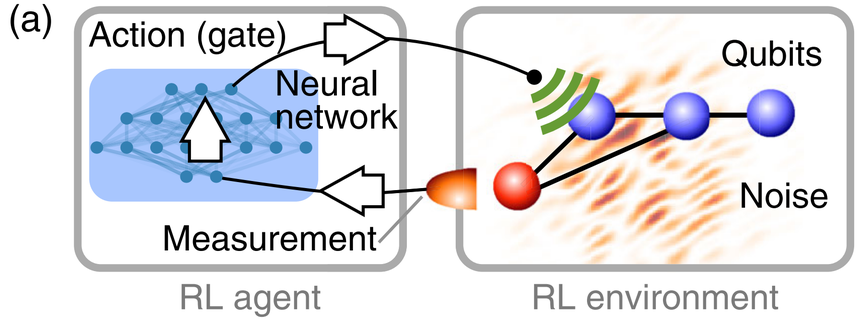


Figure 6.8: Schematic representation of the RL-based error correction framework. The agent can choose the next gate or measurement to be applied to an ensemble of a few, possibly error-affected, qubits in order to protect a single target qubit. Adapted from [332].



the goal is to preserve an arbitrary single-qubit state, $|\phi(0)\rangle = \alpha|0\rangle + \beta|1\rangle$

The agent can choose to apply gates from a given set, or to perform measurements on auxiliary qubits.

Aims at discovering the best error correction scheme from direct qubit interaction!

Reinforcement Learning: QECC

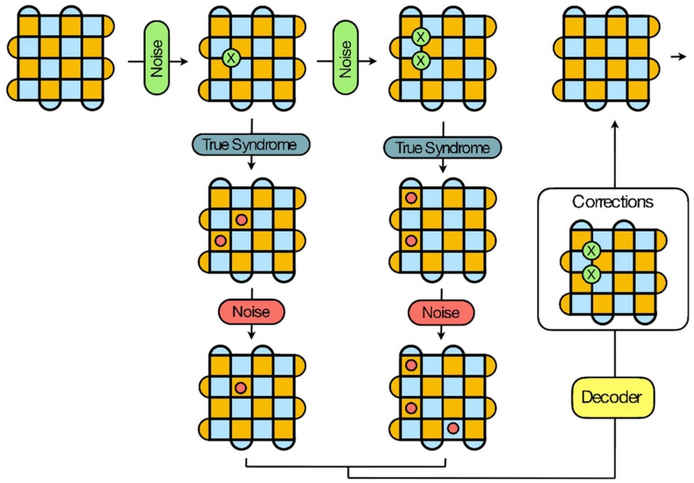


Figure 3. A typical decoding cycle is illustrated for the simplified faulty measurements scenario in which one imagines each time step consisting of an initial physical error process generating errors on the data qubits, followed by a second measurement error process which corrupts the true syndrome. The decoding algorithm then has access to a sequence of potentially faulty syndromes.

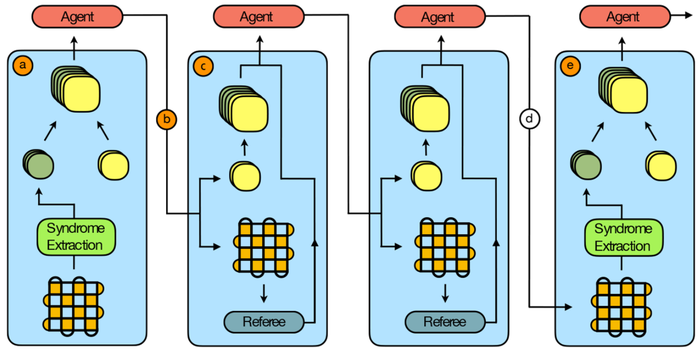


Figure 5. An illustration of the various steps occurring within a single episode. (a) In an initial step, a faulty syndrome volume is extracted from a state initially in the code space. This faulty syndrome volume is combined with an initially empty action history and passed to the agent as the initial state. Given an input state, the agent must decide on an action, which it passes to the environment. If the action is a Pauli flip which is not already in the action history (b), then the procedure illustrated in box (c) occurs, if the agent requests a new syndrome or repeats an action (d), then the procedure illustrated in box (e) occurs. (c) The chosen action is applied to the underlying quantum state. From this state the reward for the applied action is determined, while simultaneously a referee decoder decides whether or not the episode is now over—i.e. whether the underlying quantum state is now in a terminal state. Additionally, the applied action is appended to the action history, which is concatenated with the non-updated syndrome volume and provided to the agent, in conjunction with the reward and terminal state indicator. (e) Once again, the underlying quantum state is first updated, and from this updated state both a reward and terminal state indicator are determined (not shown). However, the trivial or repeated action of the agent triggers a new round of faulty syndrome measurements. After this new round of syndrome extraction, the new syndrome volume is combined with a reset action history and provided to the agent, from which it can once again choose another move.

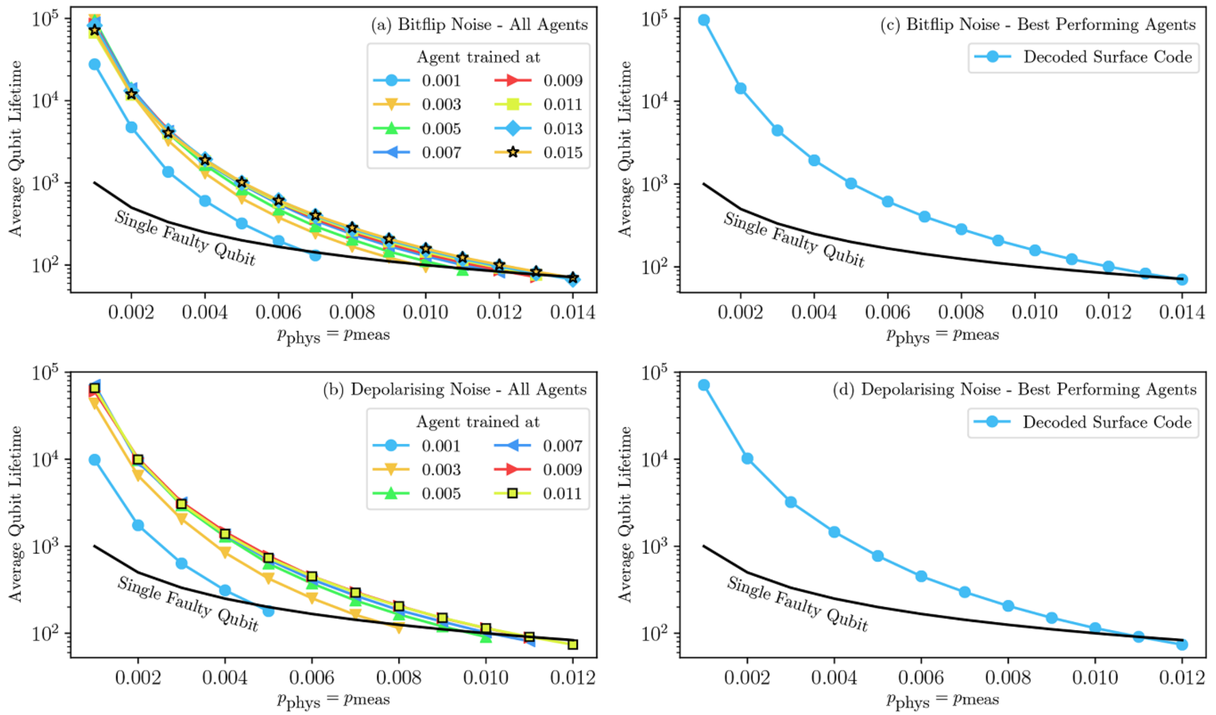


Figure 8. ((a), (b)) Performance of all agents obtained during the iterative training procedure. Each agent was evaluated at increasing error rates, until the average lifetime of the logical qubit actively decoded by the agent became less than the average lifetime of a single faulty qubit. ((c), (d)) Results obtained by using the best performing decoding agent for each error rate.

Ryan Sweke *et al* 2021 *Mach. Learn.: Sci. Technol.* **2** 025005

And more...

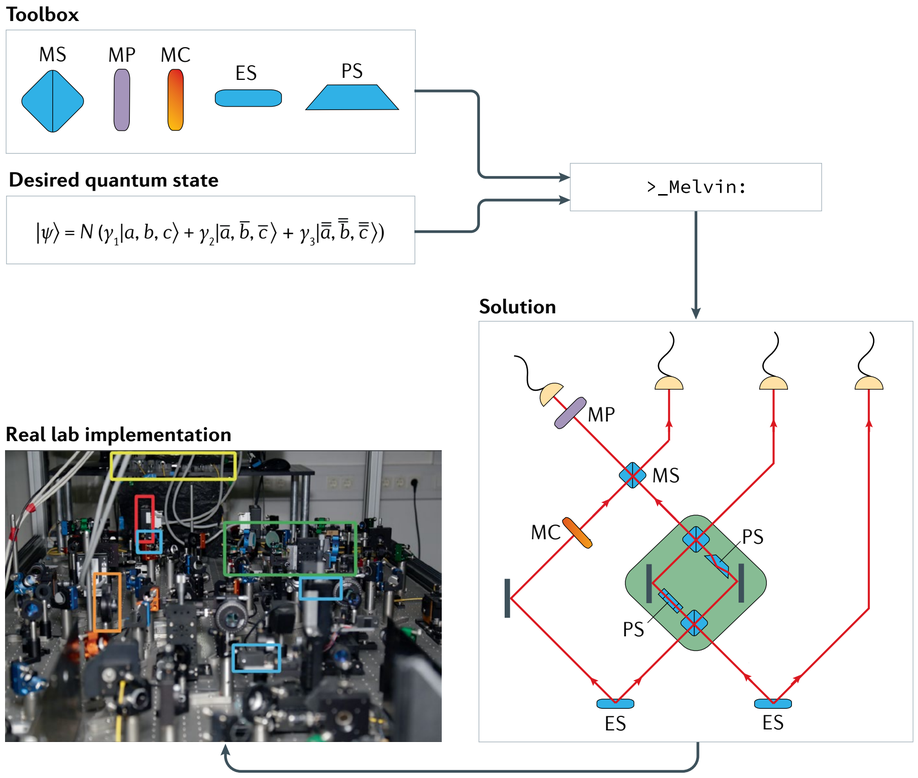


Fig. 3 | **The complexity of computer-inspired quantum experiments.** The MELVIN algorithm discovered the experimental setup for a 3D GHZ state (desired quantum state) using only the toolbox and a generalized quantum state as inputs. The toolbox contains, mode splitters (MS), mode projectors (MP), mode changers (MC), phase shifters (PS) and entanglement sources (ES). Human scientists can then implement the solution in the laboratory[89]. The symbols of the desired quantum state are defined in the main text.

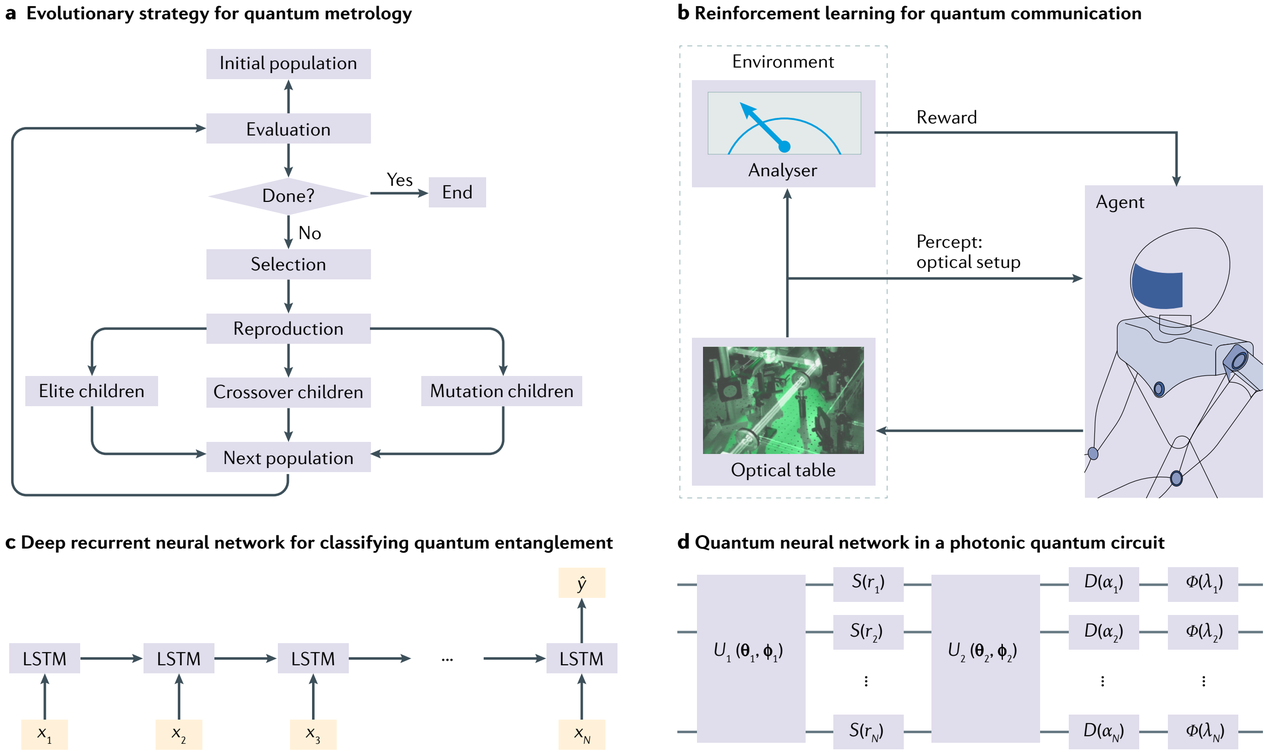


Fig. 6 | **Examples of class I algorithms for computer-inspired quantum experiments using various optimization and machine learning techniques. a** | A genetic algorithm for designing new methods in quantum metrology. An initial, random population undergoes an evolutionary process. The individual setups, which form the population, are selected according to their fitness. The best ones are mutated and form the next generation. **b** | The concept of a reinforcement learning algorithm for designing quantum communication schemes. An agent performs actions in an environment (changing quantum communication scheme), which changes the state of the environment. The agent receives a reward according to the quality of the action. **c** | The concept of a reinforcement learning algorithm for designing quantum optics experiments. An agent performs actions in an environment (changing the setup), which changes the state of the environment. The agent receives a reward according to the quality of the action. Here, x_i stand for optical elements and y is the predicted SRV class. **d** | A photonic quantum circuit can be parametrized continuously, thus gradient-based optimization techniques are possible. The individual elements of the circuits are unitary transformations U_i, squeezing S, displacement D and non-Gaussian gates Φ, with continuous parameters, $r, \alpha, \boldsymbol{\theta}, \boldsymbol{\phi}, \lambda$. LSTM, long short-term memory. Panel **a** reprinted with permission from REF.[120], IOP Publishing; Panel **b** adapted with permission from REF.[121], PNAS. Panel **d** reprinted with permission from REF.[133], IOP Publishing.

Krenn, M., Erhard, M. & Zeilinger, A. Computer-inspired quantum experiments. *Nat Rev Phys* **2**, 649–661 (2020). https://doi.org/10.1038/s42254-020-0230-4

Some of our contributions

Journal of the Optical Society of America B
OPTICAL PHYSICS

Integrating machine learning techniques in quantum communication to characterize the quantum channel

YASEERA ISMAIL,[1,*] ILYA SINAYSKIY,[1,2] AND FRANCESCO PETRUCCIONE[1,2,3]

[1]Quantum Research Group, School of Chemistry and Physics, University of KwaZulu-Natal, University Road, Durban 4000, South Africa
[2]National Institute for Theoretical Physics, South Africa
[3]School of Electrical Engineering, KAIST, Daejeon 34141, South Korea
*Corresponding author: Ismaily@ukzn.ac.za

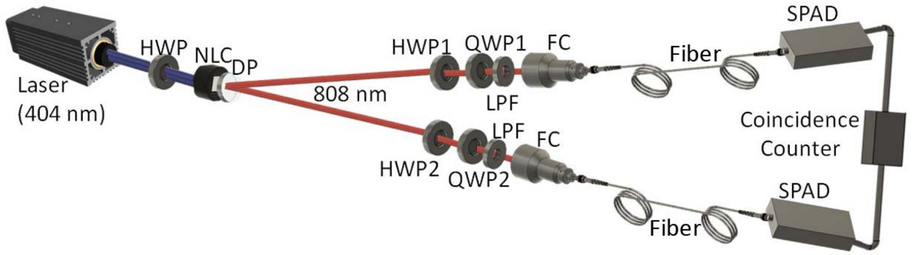


Fig. 1. Experimental setup of the single photon source where the diffractive phase plate (DP) denotes the free-space quantum channel. A pump laser lasing at 404 nm was passed through a half-wave plate (HWP), two type-I concatenated BBO crystals, a HWP, and a quarter-wave plate (QWP) in each arm, followed by a line pass filter (LPF), a fiber coupler connected to a single photon avalanche detector (SPAD), and a coincidence counter.

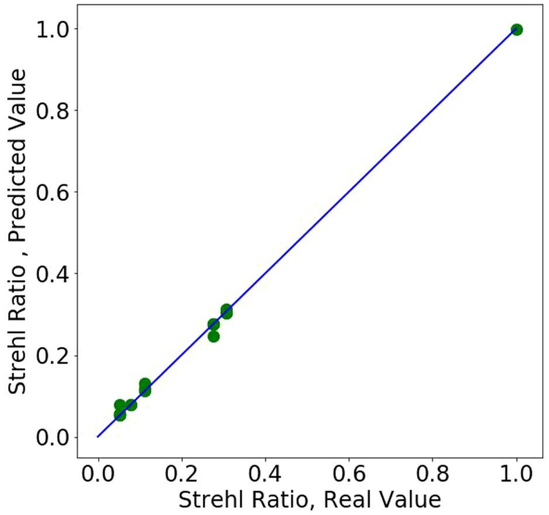


Fig. 2. Predictions of the Strehl ratio using the random forest regressor to train the data set consisting of density matrices for various turbulent strengths. The MAPE on the test set for the random forest regressor was 4.44%.

$$\mathrm{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\frac{\left|y_{\mathrm{pred}}^{(i)} - y_{\mathrm{real}}^{(i)}\right|}{\left|y_{\mathrm{real}}^{(i)}\right|},$$

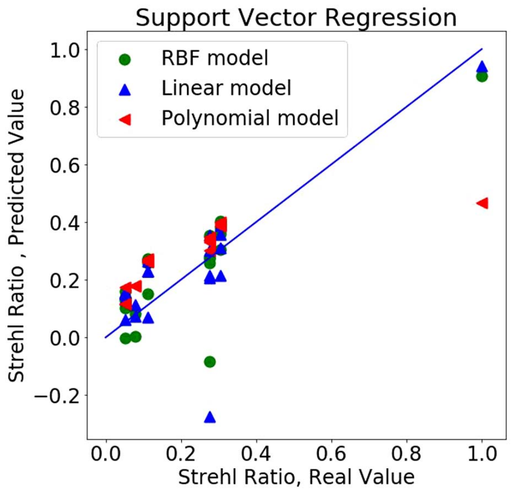
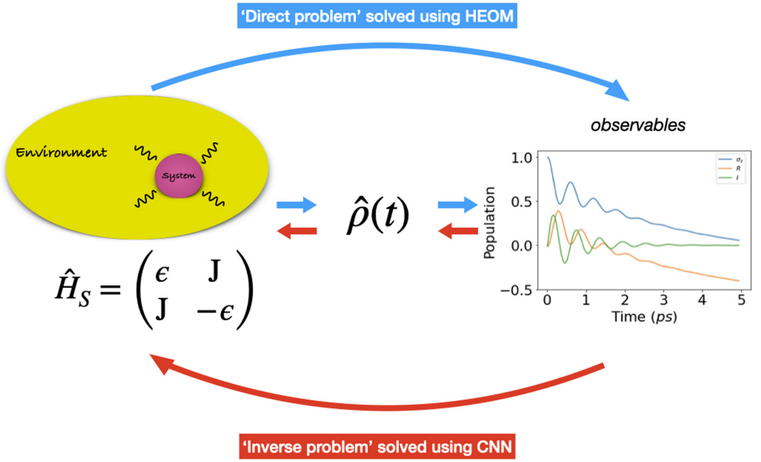


Fig. 6. Predictions of the Strehl ratio using the support vector regressor to train the data set consisting of density matrices for various turbulent strengths. The MAPEs on the test set for the support vector regressors were 74.48%, 75.83%, and 85.93% for the RBF, linear, and polynomial models, respectively.

Table 1. Summary of MAPE for Each of the Supervised Learning Models: Random Forest (RF), Linear Regressor (LR), and Support Vector Regressor (SVR)

Supervised Learning Model	MAPE without Fidelity (%)	MAPE with Fidelity (%)
RF	4.44	3.86
LR	48.96	50.10
SVR : RGB	74.48	77.52
SVR : Linear	75.83	75.56
SVR : Polynomial	85.93	78.97

ML for OQS in the strong coupling regime



$$\frac{\partial}{\partial t}\hat{\sigma}(\mathbf{n},t) = -\left(i\hat{\mathcal{L}}_e + \sum_{j=1}^{N} n_j \gamma_j\right)\hat{\sigma}(\mathbf{n},t) + \sum_{j=1}^{N}[\hat{\Phi}_j\hat{\sigma}(\mathbf{n}_{j+},t) + n_j\hat{\Theta}_j\hat{\sigma}(\mathbf{n}_{j-},t)].$$

$$\hat{\mathcal{L}}_e = [\hat{H}_S, \hat{\rho}_S],$$

$$\hat{\Phi}_j = iV_j^{\times}, \quad V_j^{\times} y = [V_j, y],$$

$$\hat{\Theta}_j = i\left(\frac{2\lambda_j}{\beta\hbar^2}V_j^{\times} - i\frac{\lambda_j}{\hbar}\gamma_j V_j^{\circ}\right), \quad V_j^{\circ} y = \{V_j, y\}.$$

Dataset	$\epsilon_j(\mathrm{cm}^{-1})$	$J_{jk}(\mathrm{cm}^{-1})$
2	$[-100, 100]$	$[-100, 100]$
3	$[-100, 100]$	$[-100, 100]$
4	$[-100, 100]$	$[-100, 100]$

TABLE III. Mean squared error (MSE) and coefficient of determination (R^2 score) of Hamiltonian parameters used in HEOM calculations and predicted by the trained CNNs. For all three datasets, the full time length of 1 ps for all features were input to the model. The results of the training, validation and test sets are shown, separately.

Dataset	Train		Validation		Test	
	MSE	R^2 score	MSE	R^2 score	MSE	R^2 score
2	0.83	99.16	0.65	99.34	0.70	99.28
3	2.92	97.06	2.86	97.11	3.28	96.64
4	6.58	93.42	6.91	93.04	7.31	92.63

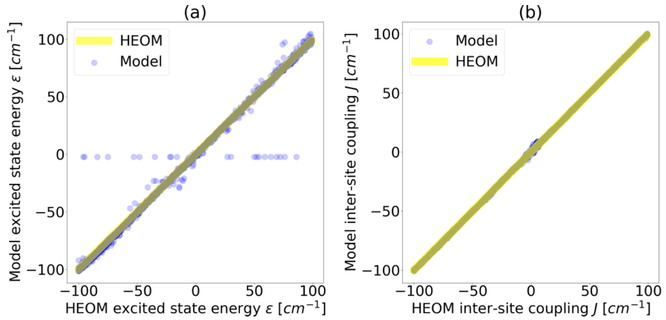
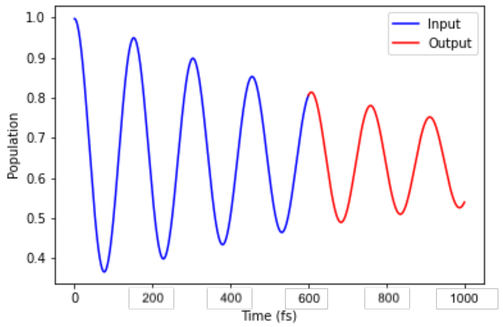


FIG. 3. (a) Excited electronic energy and (b) electronic coupling as computed with the HEOM approach compared to prediction from CNN model for a two-level system. The blue dots represent the model predictions. The yellow line indicates perfect agreement between HEOM results and predictions by the model.

Kimara Naicker, Ilya Sinayskiy, and Francesco Petruccione "Machine learning for excitation energy transfer dynamics" Phys. Rev. Research 4, 033175 2022

series prediction for OQS in the strong coupling regime



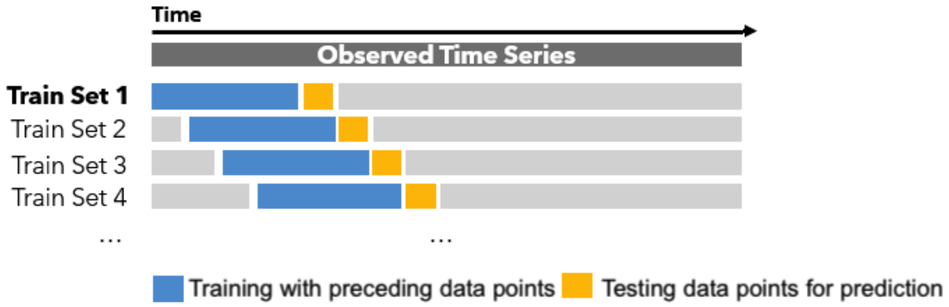


FIG. 1: An example of a sequence generated by the HEOM for a dimer to be split to form the input and output data for the models.

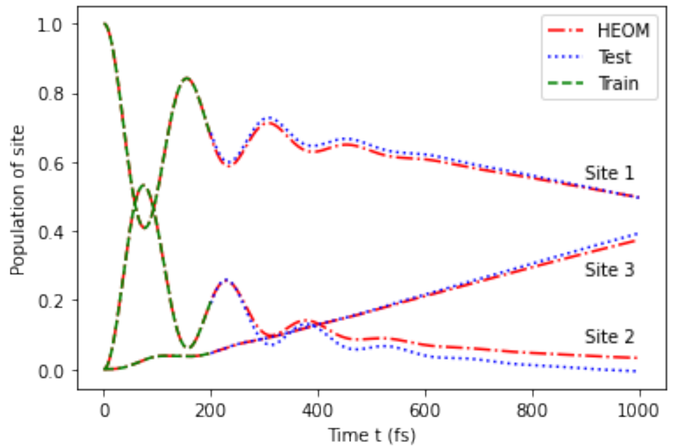


FIG. 5: Comparison of HEOM results (red) with predictions made by SARIMA (blue) based on short-time training data (green) for sites 1-3 of the well-known photosynthetic seven-site pigment protein Fenna–Matthews–Olson complex.

Model	2 level		3 level		4 level	
	MSE (10^{-3})	Time to solve (s)	MSE (10^{-3})	Time to solve (s)	MSE (10^{-3})	Time to solve (s)
SARIMA	0.3116 ± 0.0011	0.10	0.3193 ± 0.0056	0.12	0.3491 ± 0.0002	0.12
CatBoost	3.5634 ± 0.0155	4	2.1060 ± 0.0143	5	2.7800 ± 0.0244	7
Prophet	0.1281 ± 0.0099	23	0.1906 ± 0.0100	26	0.1821 ± 0.0140	27
CNN	1.3012 ± 0.0343	835	12.616 ± 0.0254	900	16.010 ± 0.0233	890
LSTM	0.0461 ± 0.0090	7200	0.0923 ± 0.0109	7400	0.1204 ± 0.0100	7500

TABLE II: The MSE values and time to train/ test for each model to predict 100 time steps ahead i.e. 0.02 fs.

Model	2 level		3 level		4 level	
	MSE (10^{-3})	Time to solve (s)	MSE (10^{-3})	Time to solve (s)	MSE (10^{-3})	Time to solve (s)
SARIMA	9.7242 ± 0.0121	0.27	12.448 ± 0.0021	0.31	11.306 ± 0.0123	0.37
CatBoost	13.596 ± 0.1098	7	19.558 ± 0.1138	9	14.855 ± 0.0198	8
Prophet	236.66 ± 0.3345	24	307.47 ± 0.4341	22	238.76 ± 0.3971	30
CNN	22.681 ± 0.1143	4200	65.123 ± 0.1909	3990	66.321 ± 0.0967	4300

TABLE III: The MSE values and time to train/ test for each model to predict 3000 time steps ahead i.e. 0.6 fs.

Kimara Naicker, Ilya Sinayskiy, Francesco Petruccione “**Statistical and machine learning approaches for prediction of long-time excitation energy transfer dynamics**” arXiv:2210.14160

Thank you!

sinayskiy@ukzn.ac.za

quantum.ukzn.ac.za