

COGS514 - Cognition and Machine Learning

Introduction

COGS514 - Cognition and Machine Learning

- ODTUClass for course announcements, quizzes, materials, communication, questions...
- Course assistant: [Ali Eren Çetintas](#)

Course Information

Prerequisites:

- Programming experience

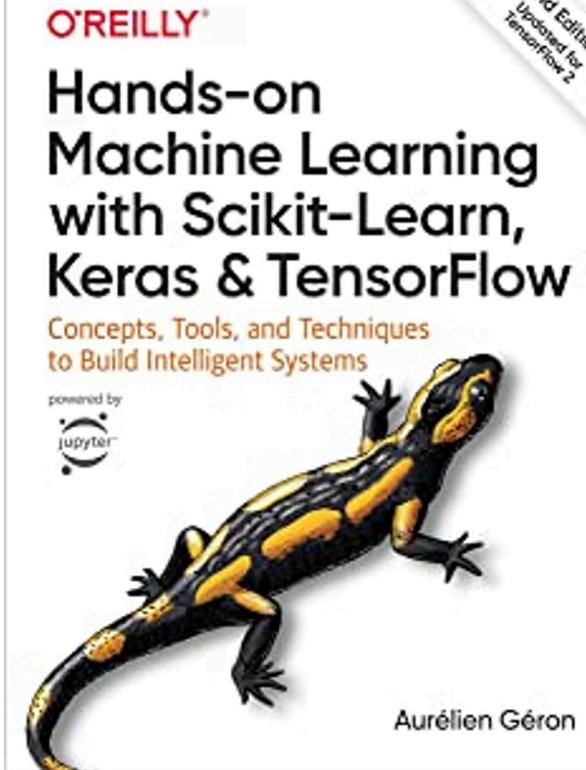
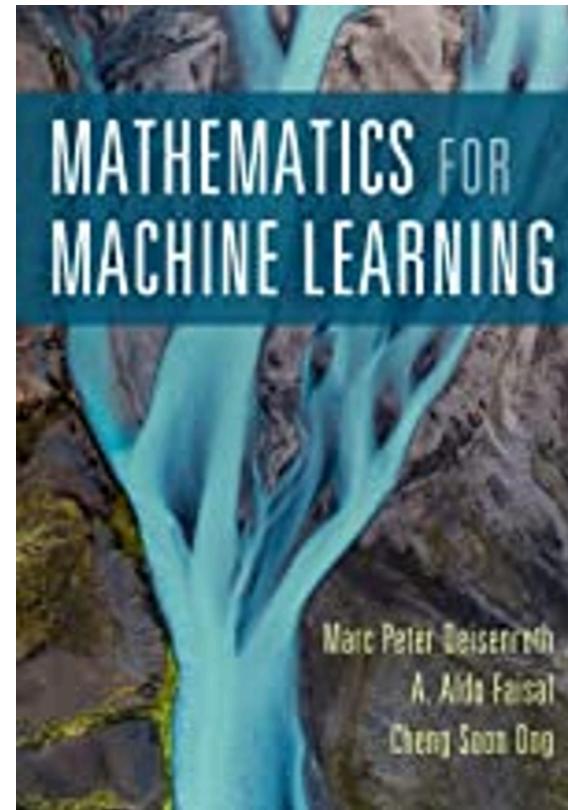
Work and grading (tentative)

- Homework assignments (~4-5) (35%)
- Final project (40%)
- Midterm exam (7.5%)
- Final exam (17.5%)

Course participation (at least 70% of the lectures) is mandatory.

Key Skills

- Programming
- Linear Algebra
- (Basic) Calculus



Course Resources

- Geron A (2019). Hands-on MACHine Learning with Scikit-Learn, Keras & Tensorflow. Second Edition. O'Reilly
- Hardt, M., Recht, B. Patterns, Predictions, and Actions: A story about machine learning, <https://mlstory.org>
- Alpaydin, E. (2010). Introduction to machine learning, Second edition. Cambridge: MIT Press.
- Charniak E. (2018). Introduction to Deep Learning. MIT Press.

COGS514 - Tools

- Python
 - **numpy**
 - **scikit-learn**
 - **tensorflow**
 - **keras**

COG514 - Workload and What to Expect

- Regular programming homeworks and quizzes to help you learn by using/applying
 - You will implement ML algorithms yourself
- Course project

COGS514 - What **not** to Expect

- "It is a graduate course, it will be easier than my undergrad courses." 🤪
- "It is a graduate course, it's OK if I don't attend." 😔
- "It is a graduate course, so it will be like a discussion session." 😕
- "It is a graduate course, there will not be much programming." 😨
- "It is a graduate course, so I don't need to use maths." 🤯

Course Content

- Linear Models
 - Linear classifier
 - Linear regression
- Non-linear models
- Optimization
 - Gradient Descent
- Feature Engineering
- Recommender Systems
 - Collaborative Filtering
- Ensemble Methods
- Neural Networks

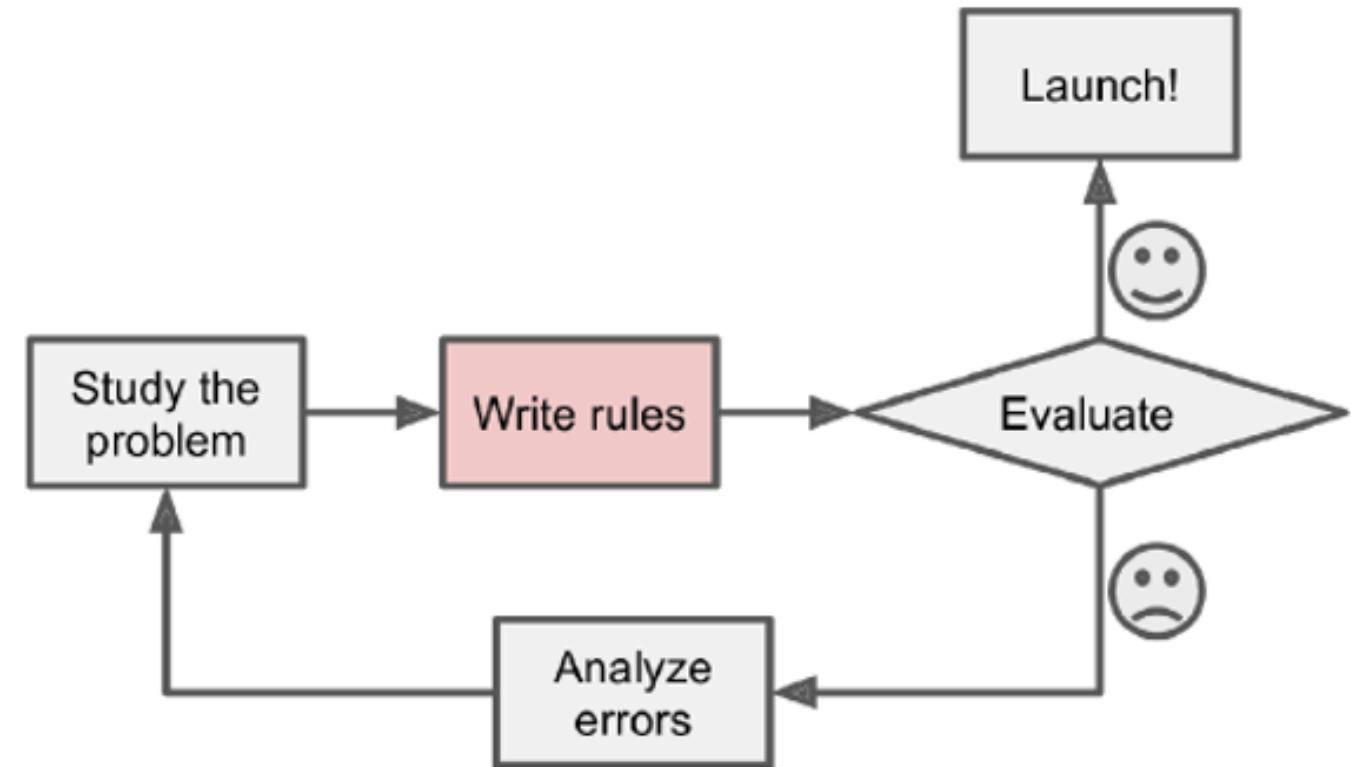
What is Machine Learning?

Artificial Intelligence (AI)

- 1956 - The Dartmouth Summer Research Project on Artificial Intelligence
- John McCarthy, Marvin Minsky, Claude Shannon, Nathaniel Rochester, ...

We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. **We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.**

Good Old Fashioned AI (GOFAI)



AI Spring - 1980's

- Expert Systems
- Industry interest
 - AI becomes a billion dollar industry
 - AI division in nearly every major company





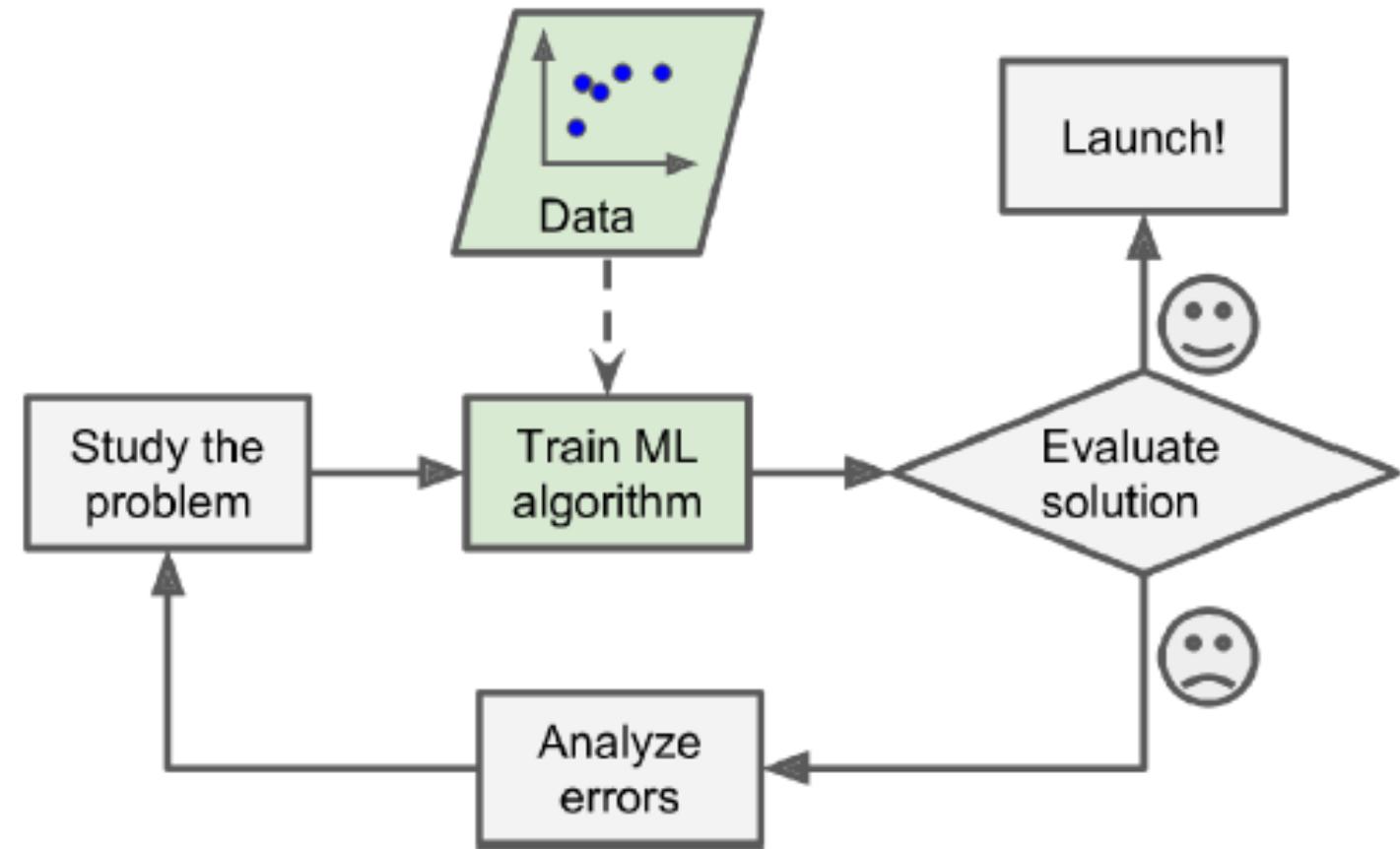
AI Winter - 1990's

Problems with expert systems:

- Brittleness
- Learning

Machine Learning

Computer programs that improve with experience



Machine Learning and Prediction

Most of machine learning is focussed on prediction

AI Spring

- Plays games better than humans (Go, Chess, Starcraft 2, DOTA ...)
- Similar or better prediction performance for detecting some diseases (radiography, dermatology, ophthalmology)
- More than 30 million kilometers of autonomous driving in roads with traffic

Metin

Dokümanlar

DİLİ ALGILA

ENDONEZYA DİLİ

TÜRKÇE

RUSÇA



İSVEÇÇE

İNGİLİZCE

TÜRKÇE



O güzel

O akıllı

O okur

O bulaşık yıkıyor

O araştırma yapıyor

O profesör

O dikiş dikiyor

She is beautiful

He is smart

He reads

She's washing the dishes

He is doing research

He is professor

She is sewing



89 / 5000



Learning

Supervised Learning

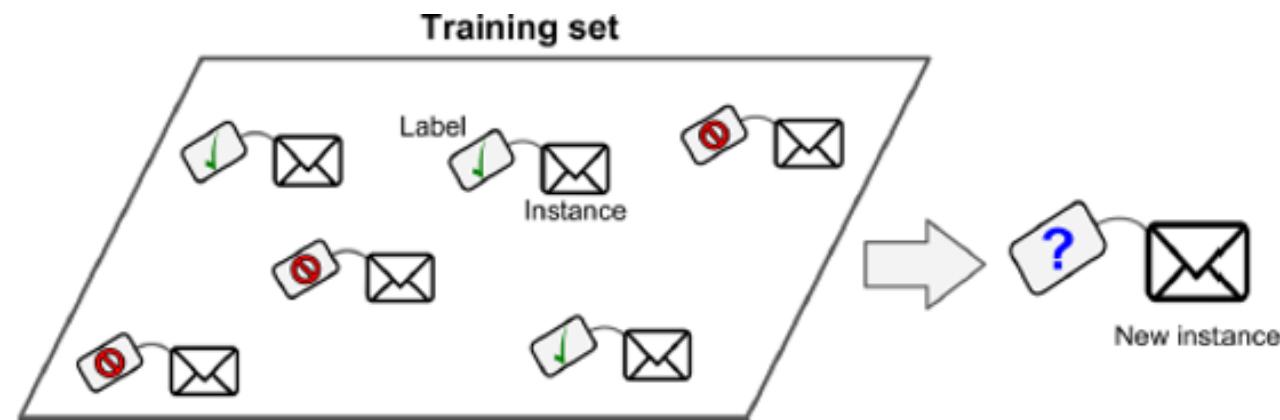
Unsupervised Learning

Reinforcement Learning

:

Supervised Learning

- Learn from labelled data
- Learn a function that map inputs X to outputs Y



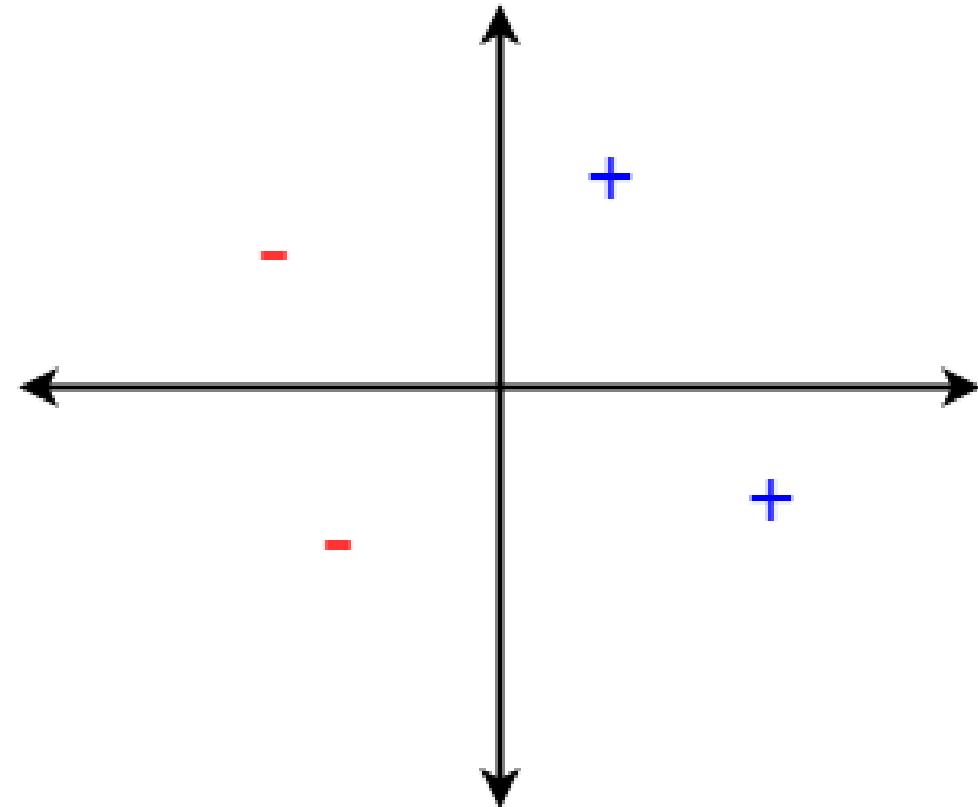
Supervised Learning - Classification

- Predicted values (Y) are discrete

fruit	width	height	weight	class (Y)
1	38	165	172	banana
2	39	218	230	banana
3	80	76	145	orange
4	35	145	150	banana
5	88	90	160	orange
...

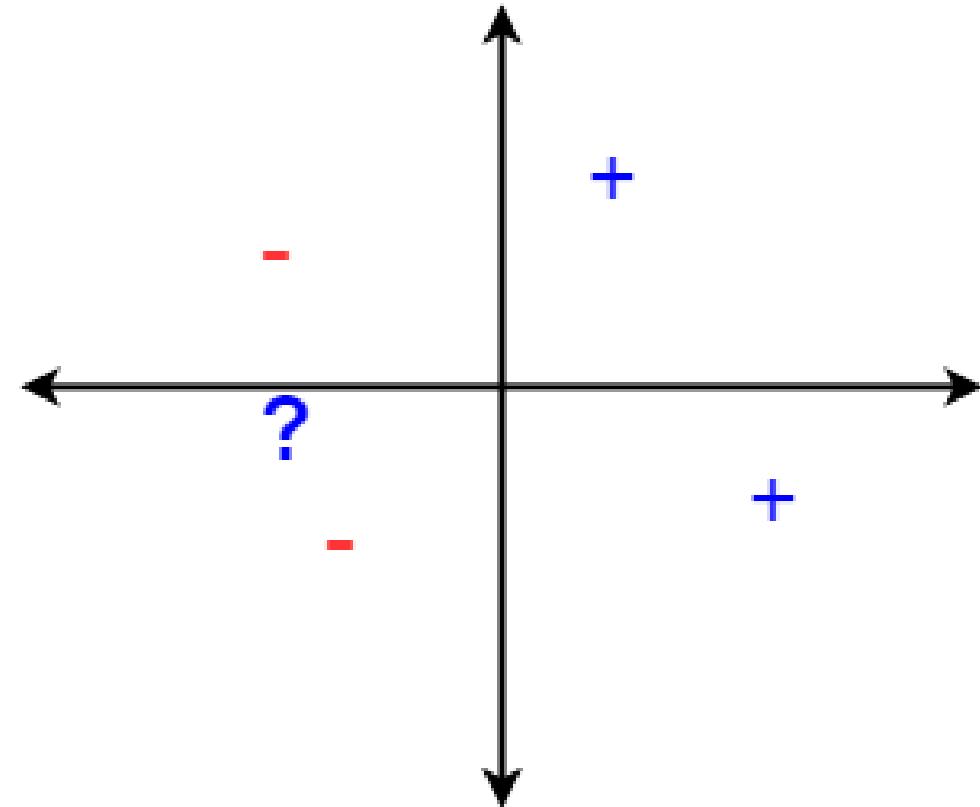
Supervised Learning - Example

item	power	portability	like
	3	1	+
	-2	2	+
	-2.1	-1	-
	2.5	-2	-

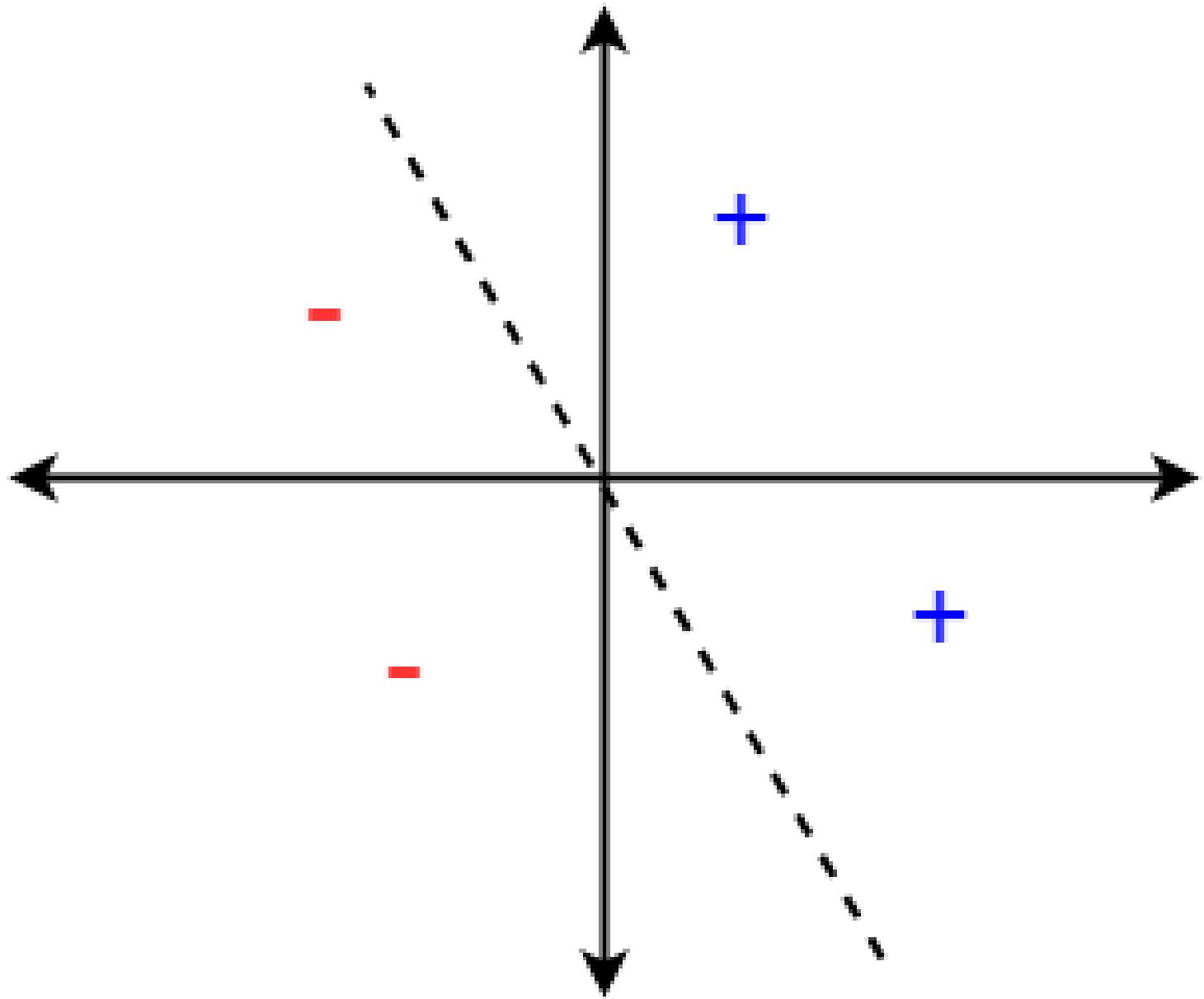


Supervised Learning - Example

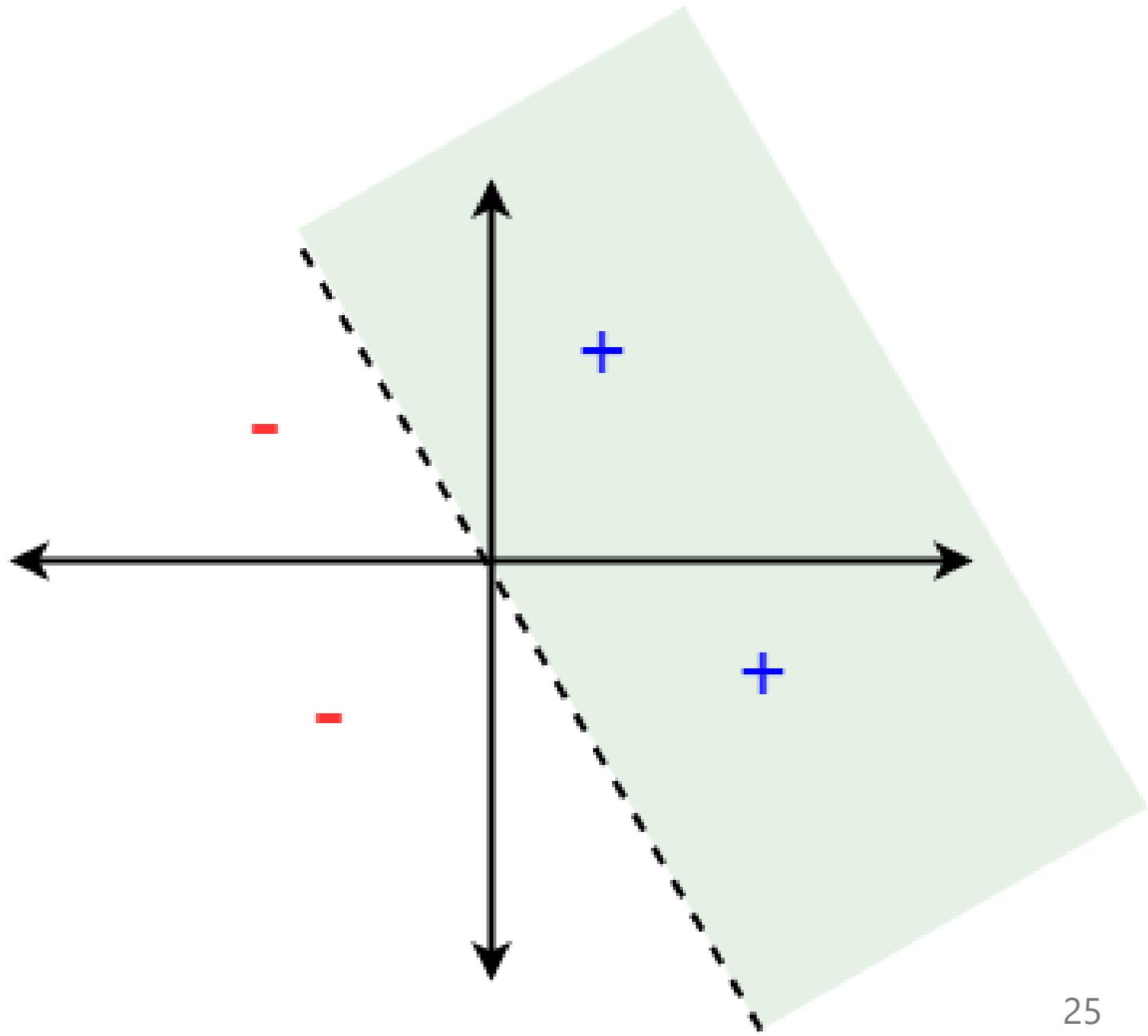
item	power	portability	like
	3	1	+
	-2	2	+
	-2.1	-1	-
	2.5	-2	-
	-0.5	-2	?



Linear Classifiers

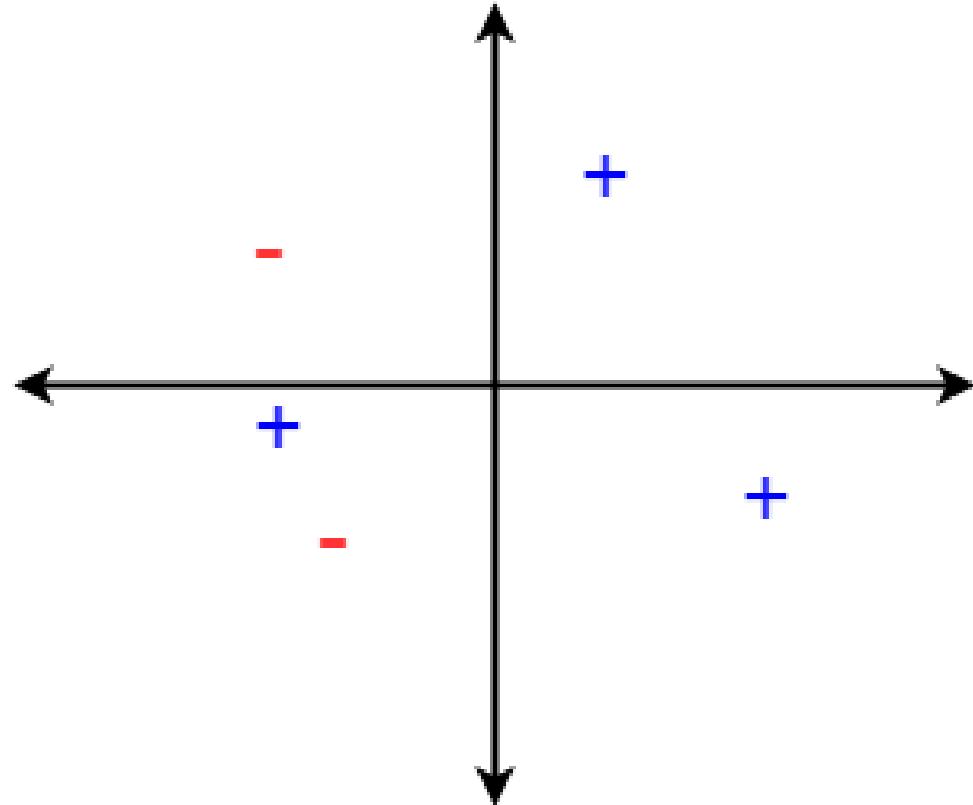


Linear Classifiers



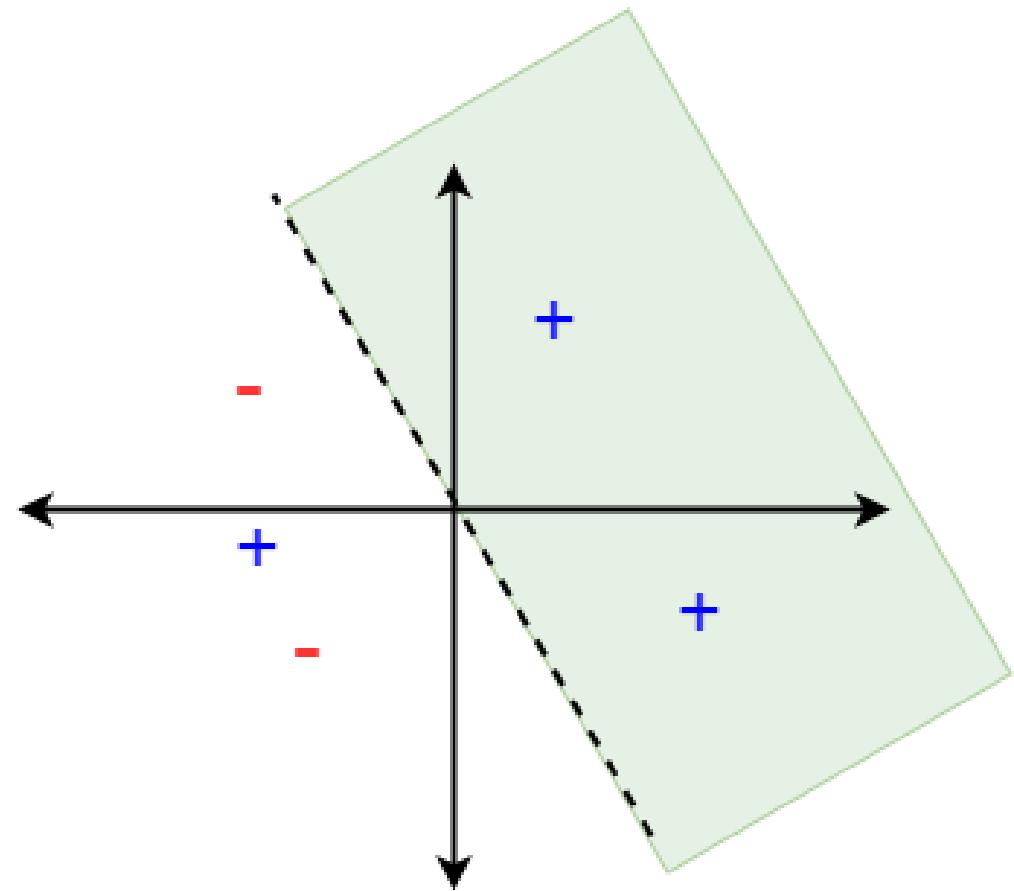
Linear Classifiers

item	power	portability	like
	3	1	+
	-2	2	+
	-2.1	-1	-
	2.5	-2	-
	-0.5	-2	+

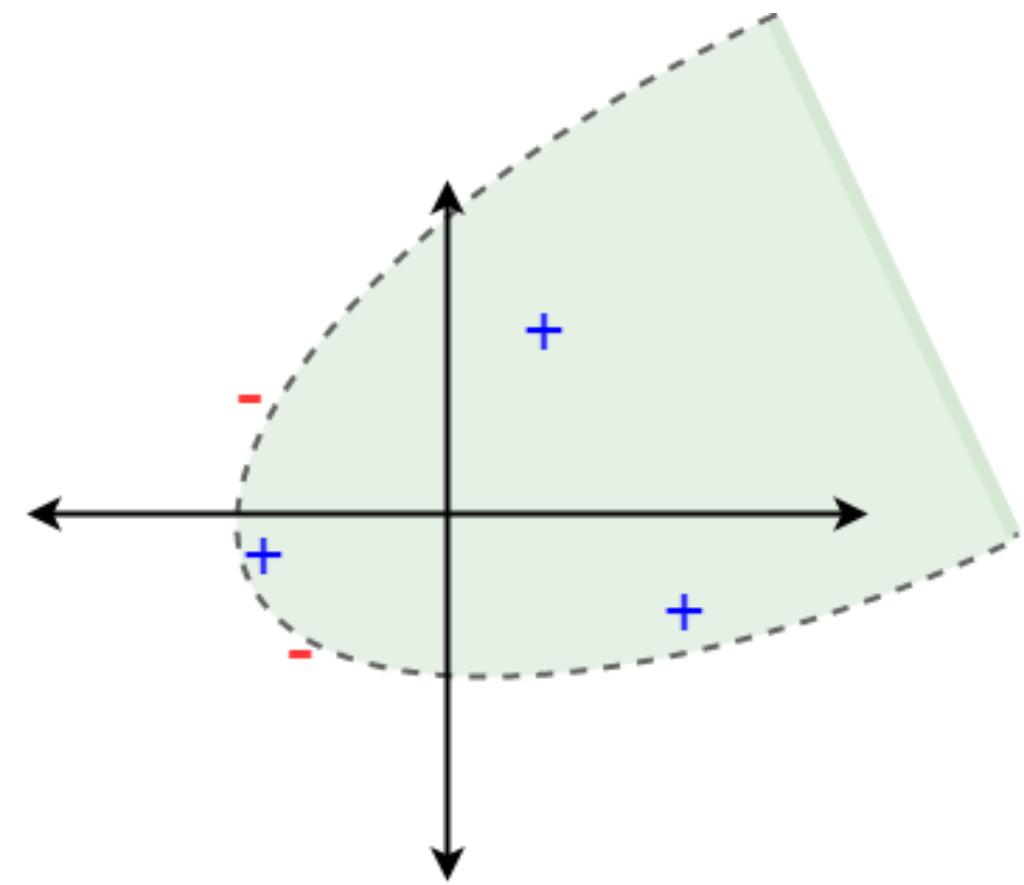


Linear Classifiers

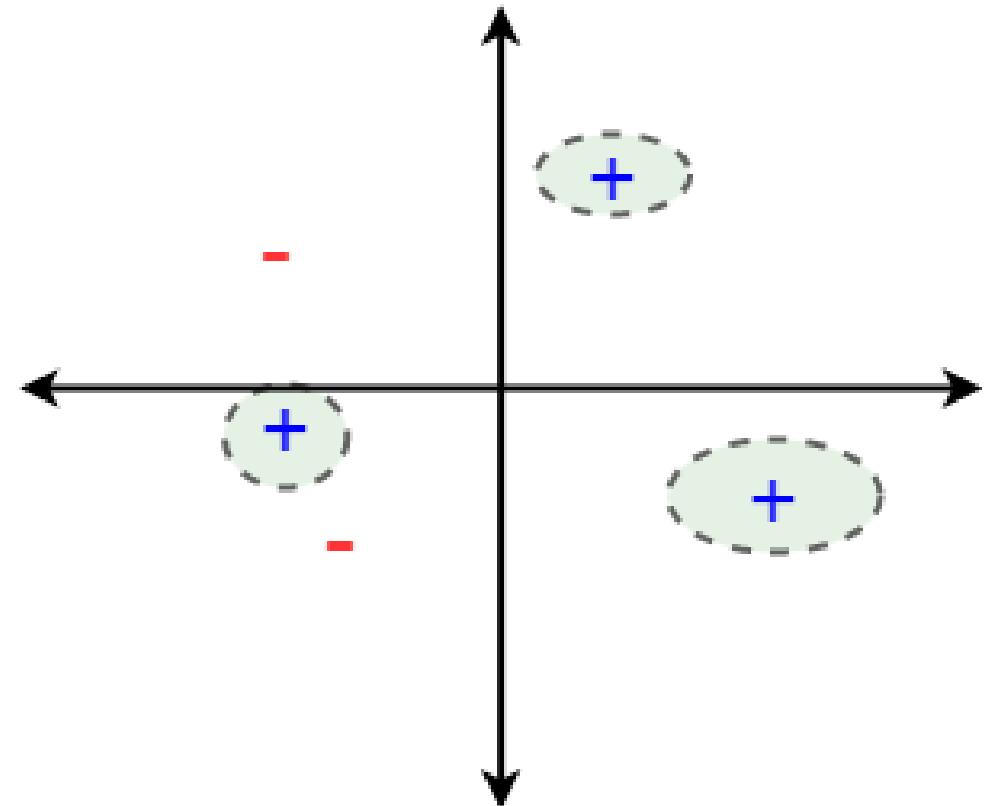
Error



Non-Linear Classifiers



Non-Linear Classifiers



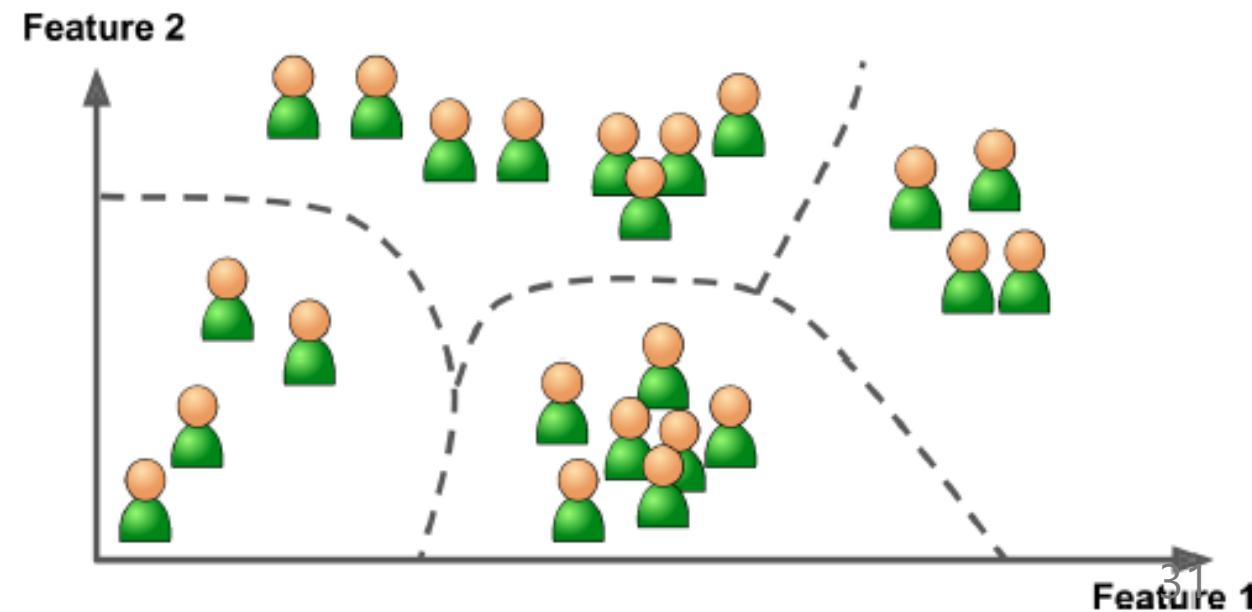
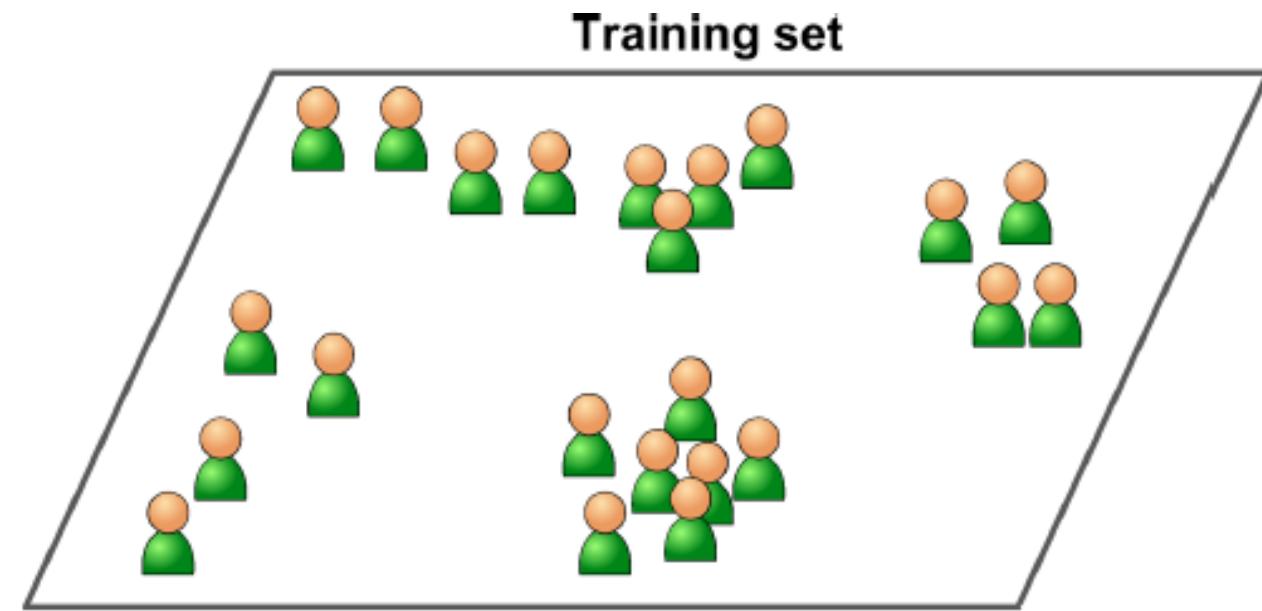
Supervised Learning - Regression

- Predicted values (Y) are continuous

property	room	bathroom	area m^2	location	price (Y)
1	2	1	100	Çayyolu	₺4,000,000
2	3	1	140	Yıldız	₺2,500,000
3	3	2	180	Oran	₺5,000,000
...

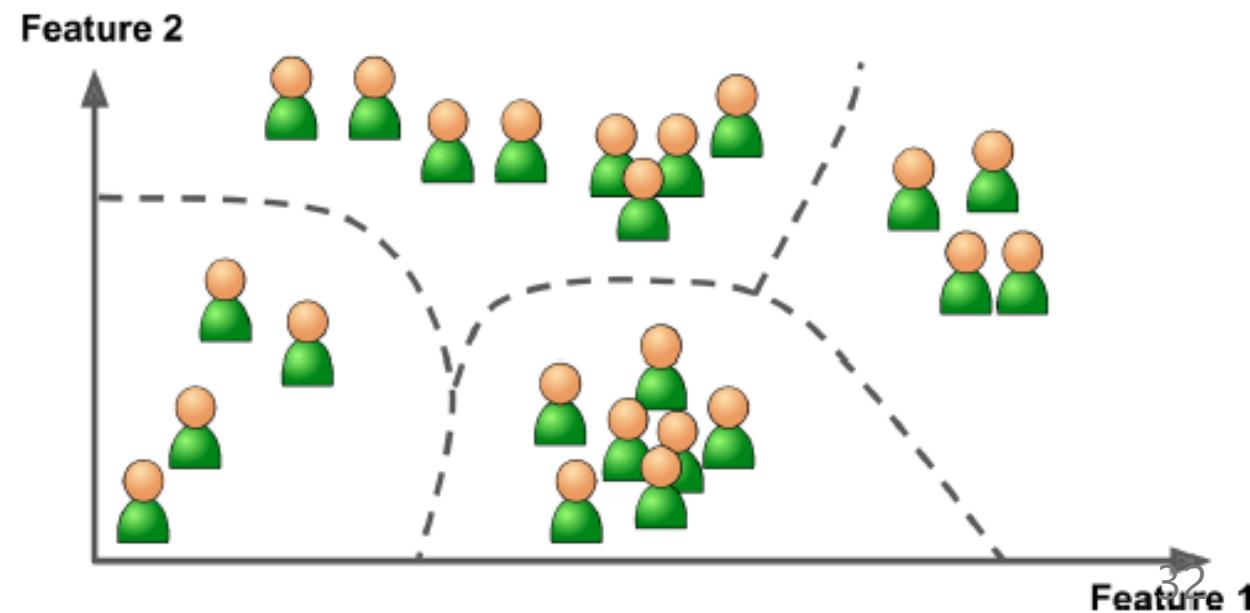
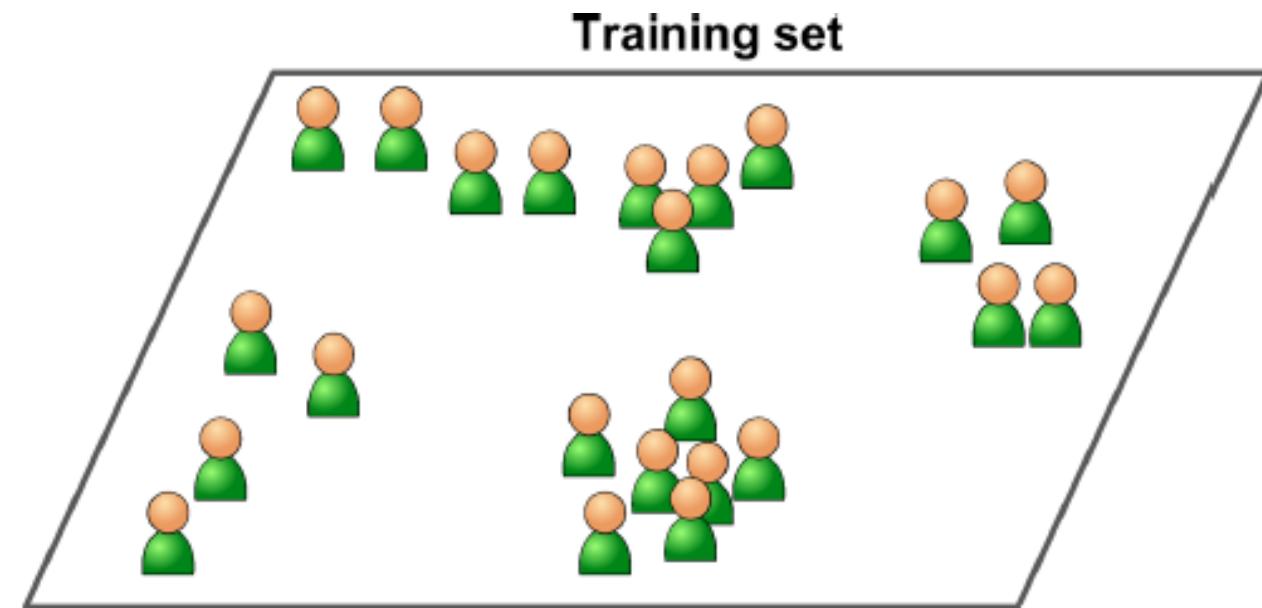
Unsupervised Learning

- Learn from unlabelled data
- Learn patterns among X without any Y



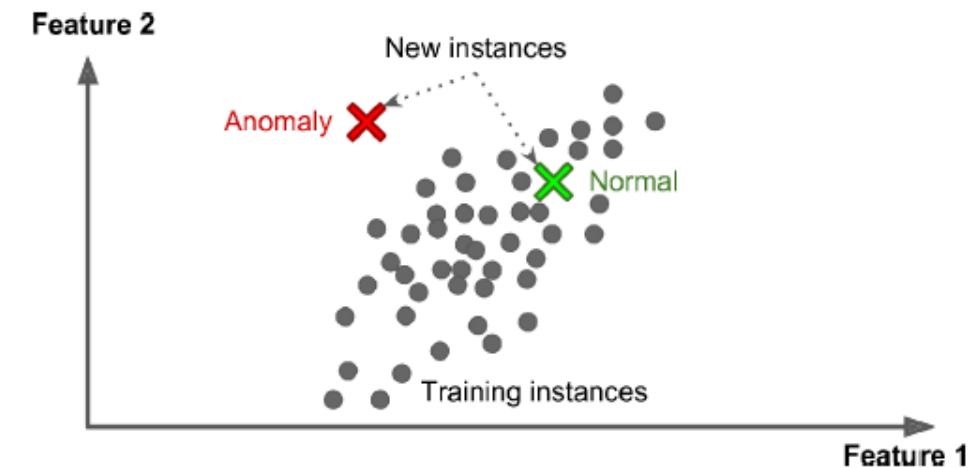
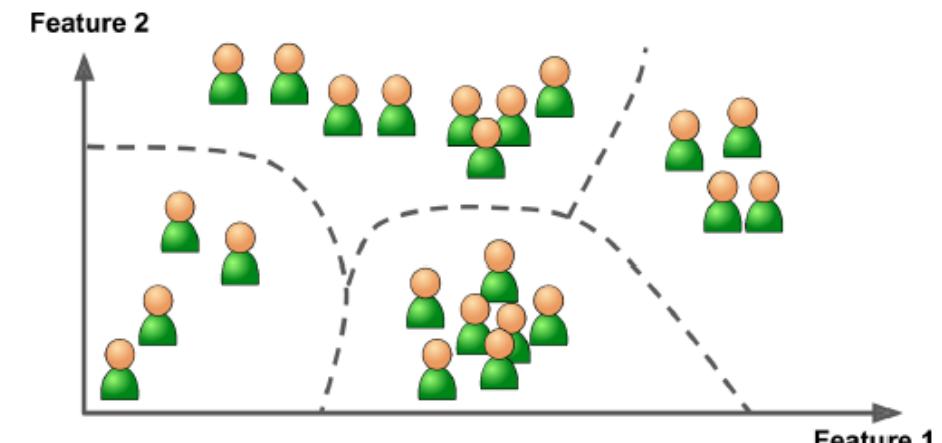
Unsupervised Learning

fruit	width	height	weight
1	38	165	172
2	39	218	230
3	80	76	145
4	35	145	150
5	88	90	160
...



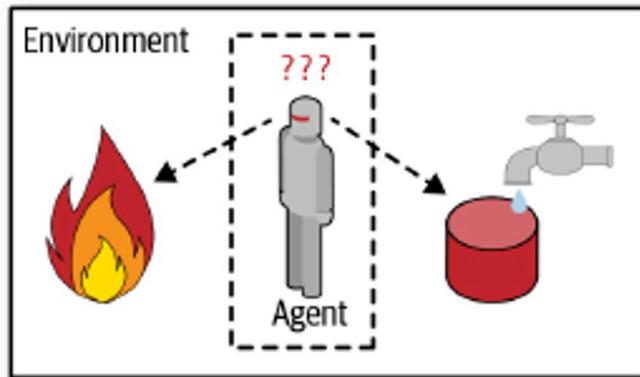
Unsupervised Learning

- Clustering
 - Organize a set of objects into groups such that similar objects tend to be in the same group
- Matrix Completion
 - Complete missing information
 - Collaborative Filtering
- Anomaly Detection

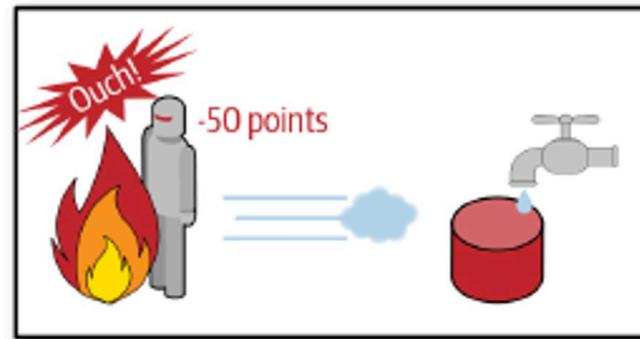


Reinforcement Learning

- Learn what **actions** to make given rewards or punishments for previous action outcomes



- 1 Observe
- 2 Select action using policy



- 3 Action!
- 4 Get reward or penalty



- 5 Update policy (learning step)
- 6 Iterate until an optimal policy is found

Three main problems of Machine Learning

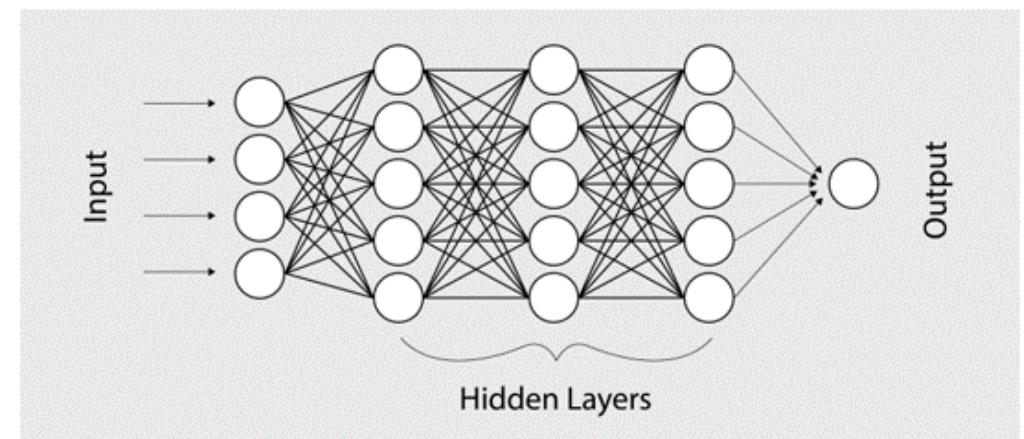
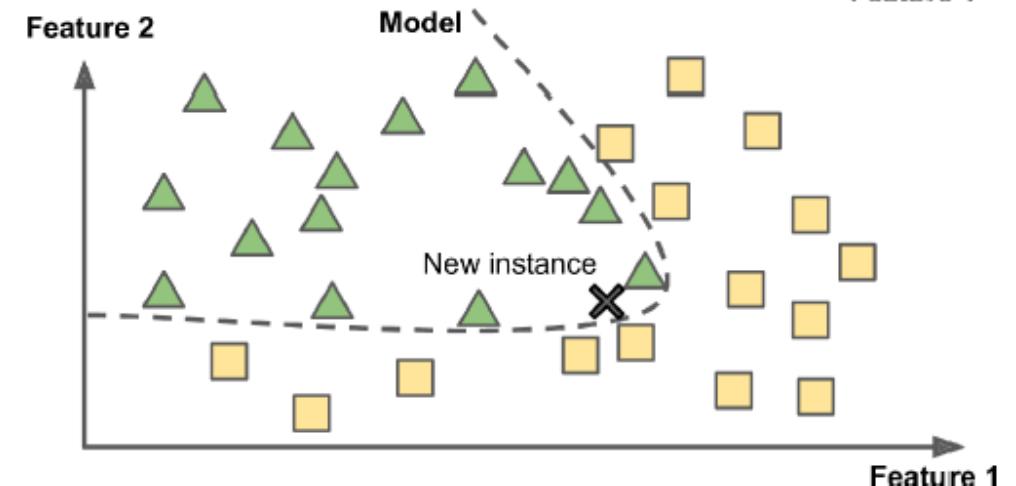
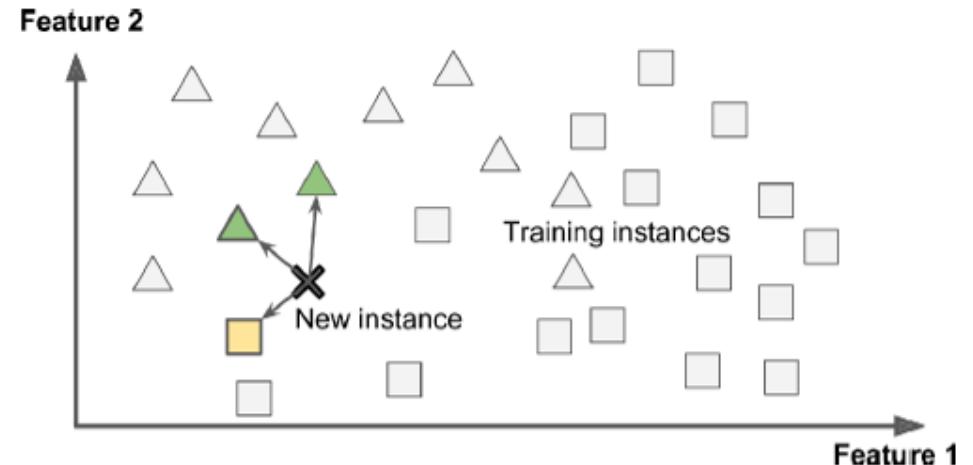
1. Representation

2. Optimization

3. Generalization

1. Representation

1. Linear models
2. Neural networks
3. SVM
4. Random forests
5. Bayesian networks
6. Instance-based learning
7. ...



"No free lunch" theorem

"All models are wrong but some are useful."

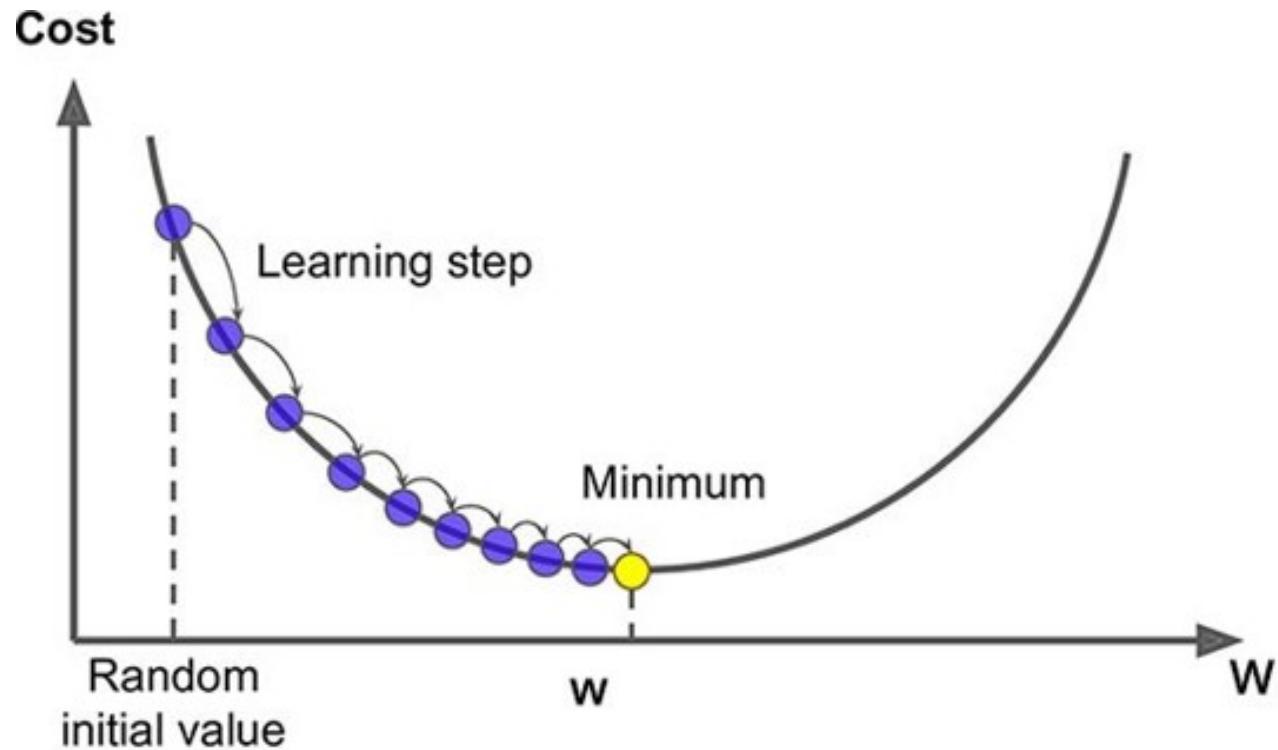
George box

- Models are simplifications
- They capture some useful mechanism about the world but not all
- There is no universally best model
- You need to make assumptions regarding what to discard and what to keep in your model.
- Set of assumptions that works well in some domain may work poorly in another

2. Optimization

After choosing representation, we optimize a performance measure

- We minimize loss function
- the Gradient Descent Algorithm



3. Generalization

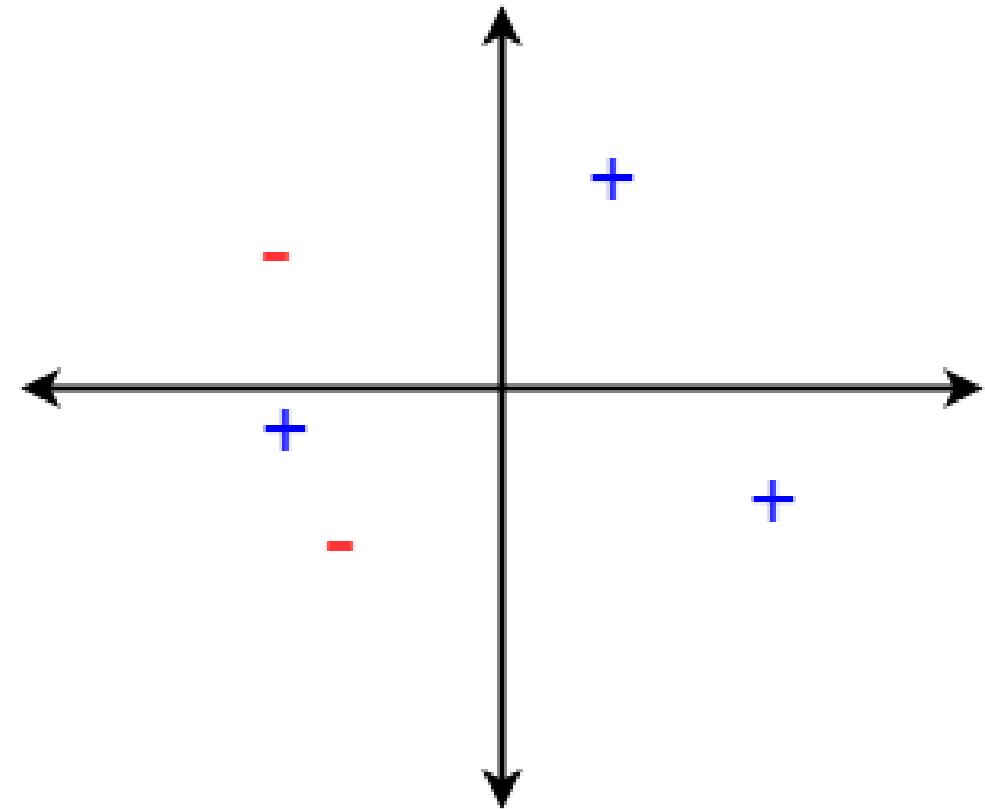
- Training dataset: the data you used to train your model
- Will the model be able to reproduce its performance on a new dataset (data it has not seen before)?
 - Overfitting vs. underfitting problem

Overgeneralization (humans) & Overfitting (machines)

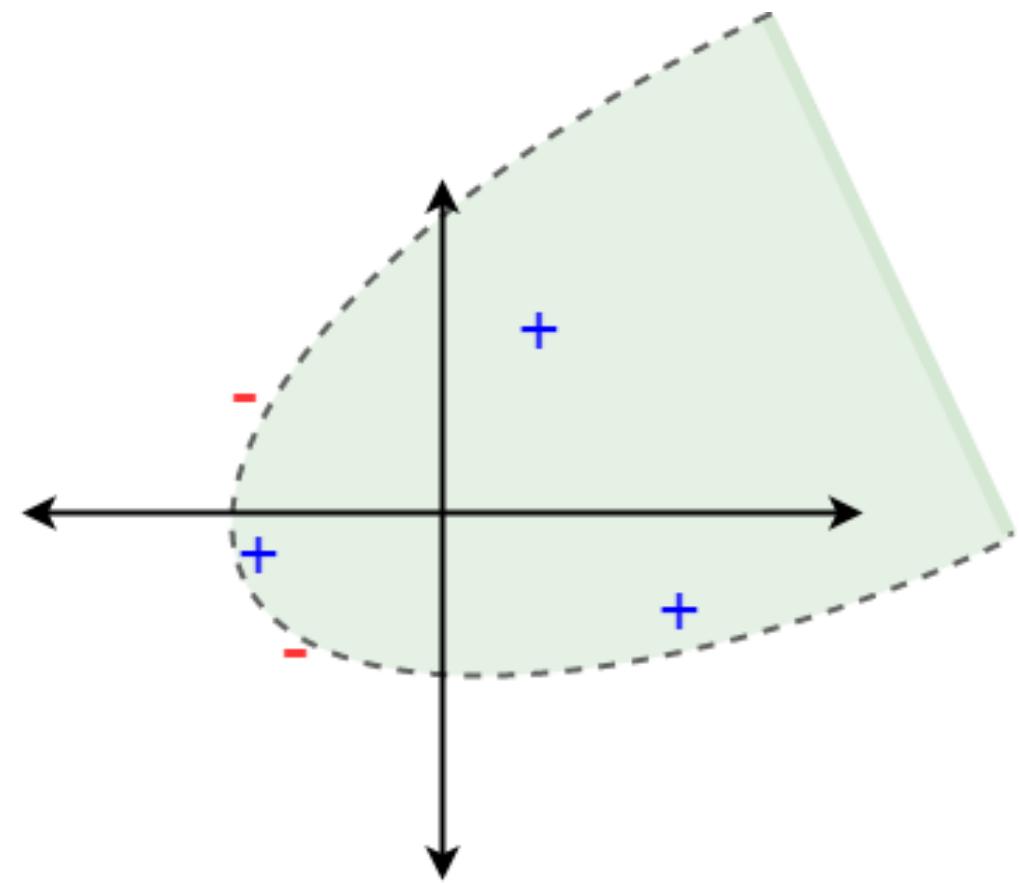
- Generalization from a few examples
- "All taxi drivers scam!" 😠



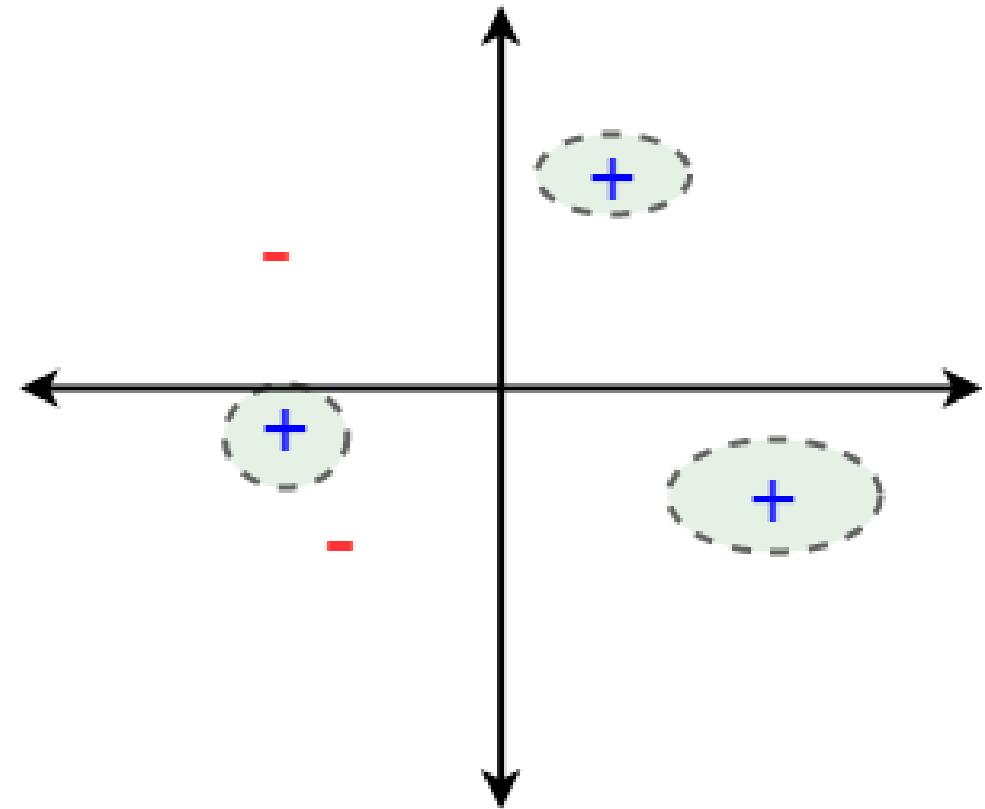
Overfitting



Overfitting

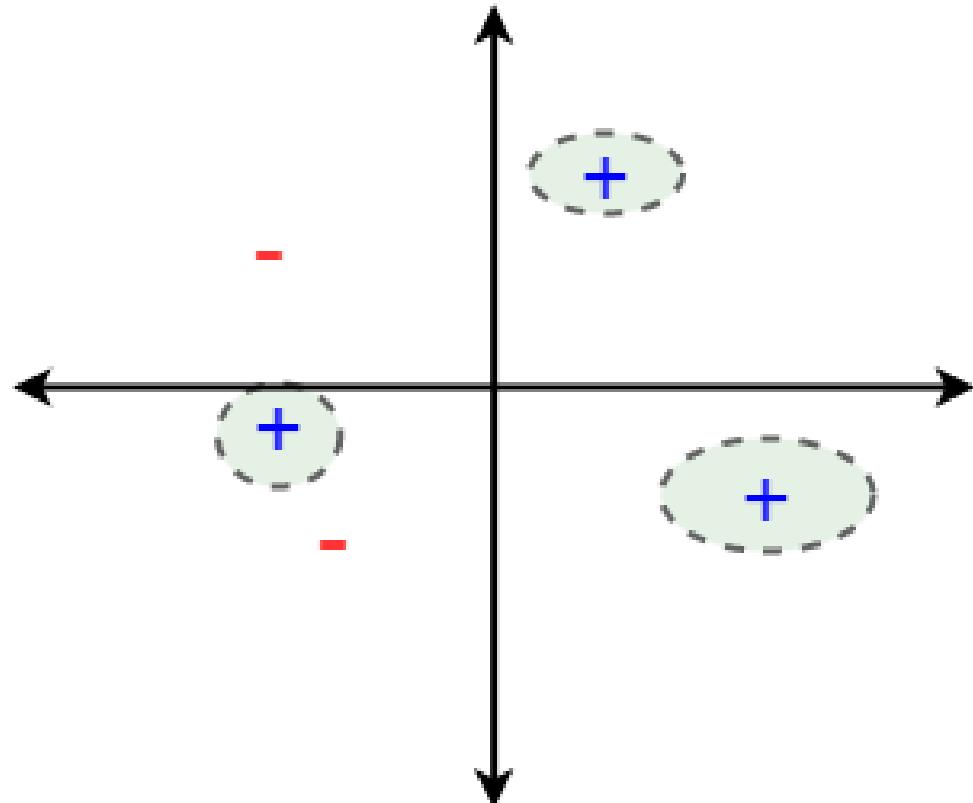


Overfitting



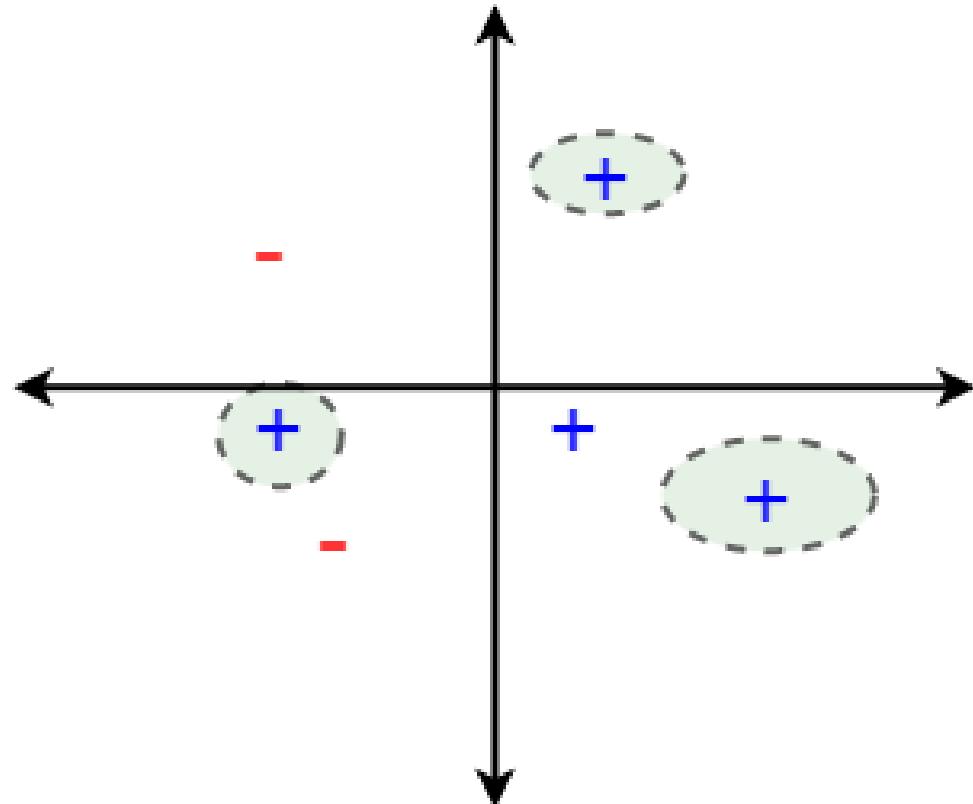
Overfitting

- Algorithm performs well on the training data
- The performance does not generalize when predicting new examples



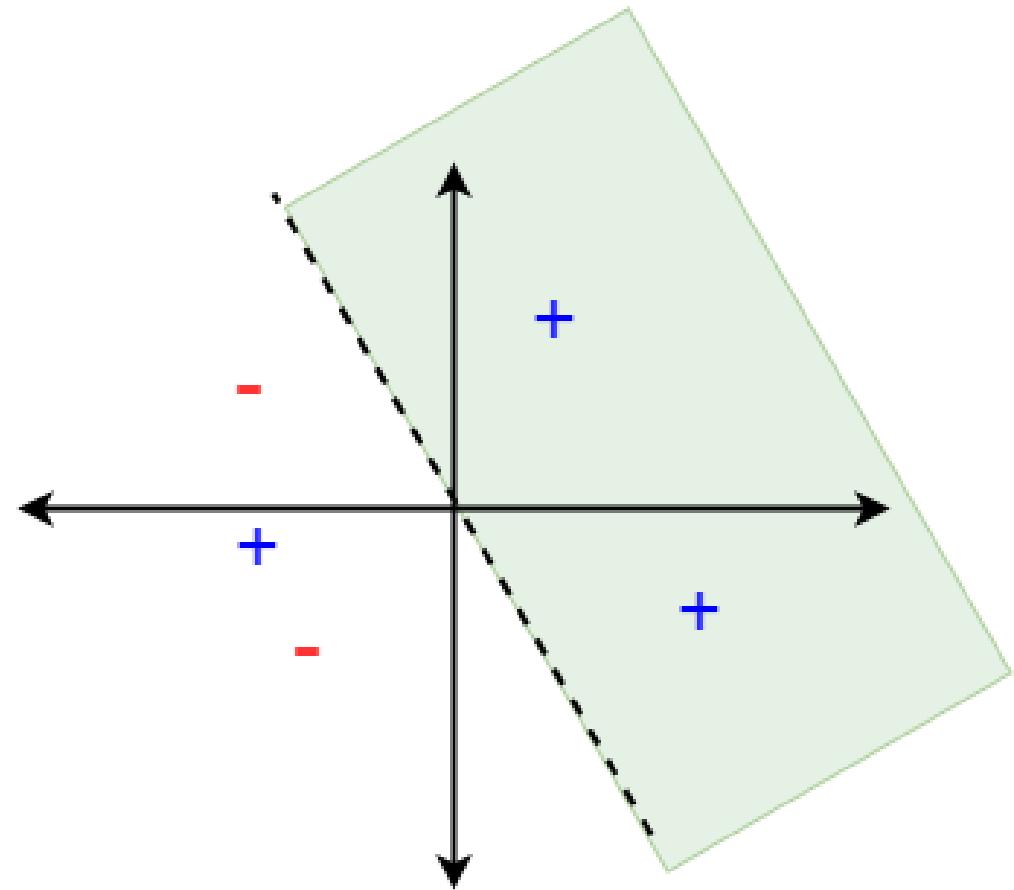
Overfitting

- Algorithm performs well on the training data
- The performance does not generalize when predicting new examples



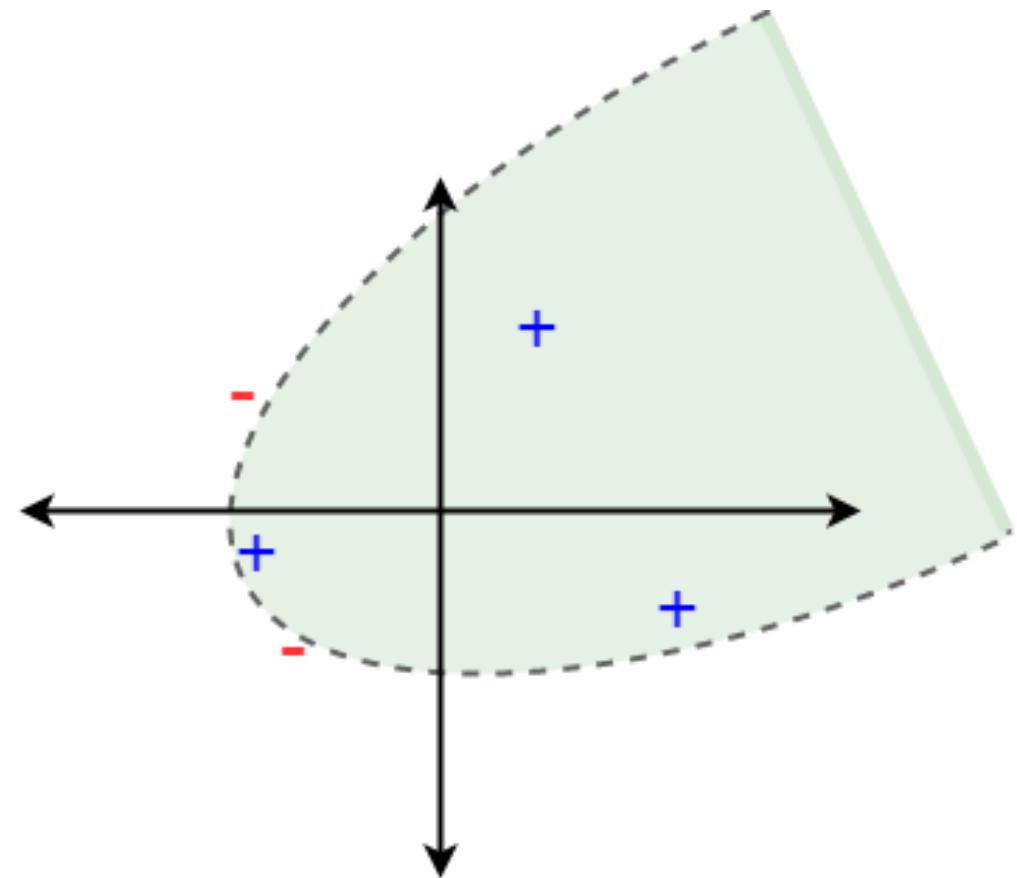
Underfitting

- Model is too simple to learn the underlying patterns in the data
 - Opposite of overfitting



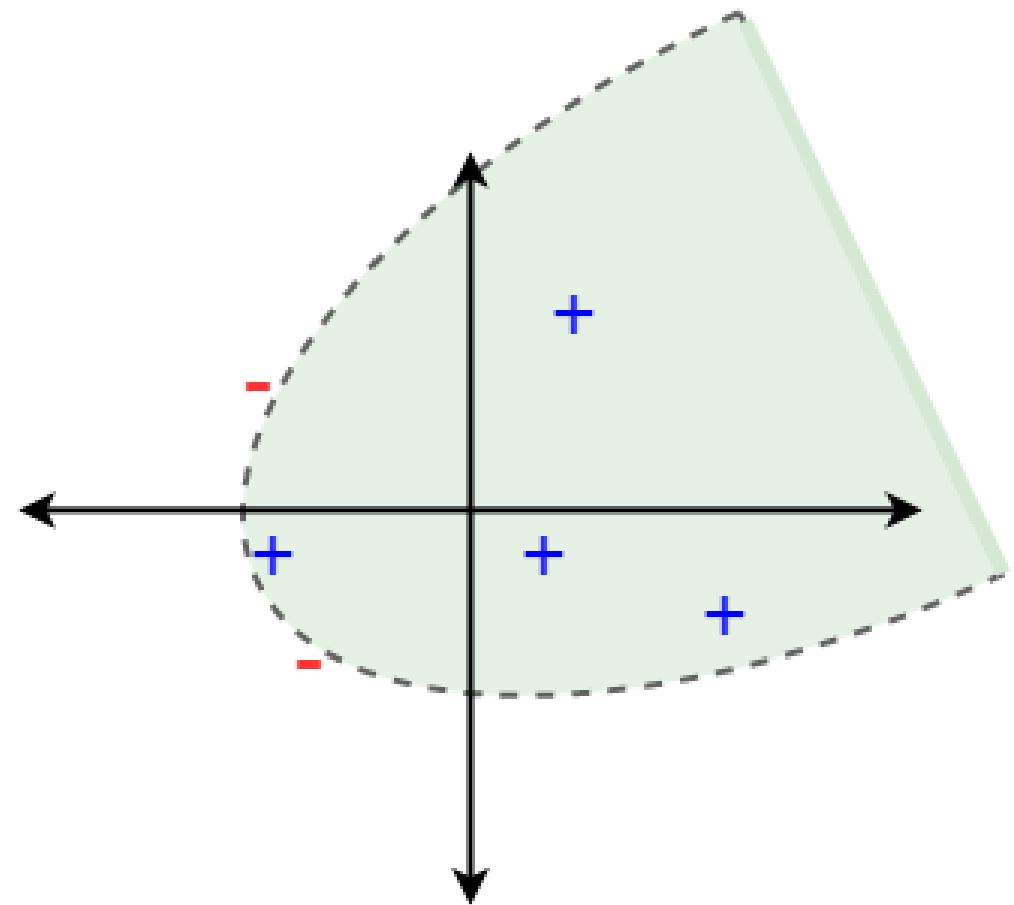
over vs. under fitting

- We need to determine the right degree of model complexity considering overfitting and underfitting



over vs. under fitting

- We need to determine the right degree of model complexity considering overfitting and underfitting



Training - Validation - Test Datasets

- It's not a good idea to evaluate how well your model performs after you release it to customers/users.

Training - Validation - Test Datasets

- It's not a good idea to evaluate how well your model performs after you release it to customers/users.
- We divide the model into a train set and validation set.
 - A common division choice 80% for training – 20% for validation



Training - Validation - Test Datasets

- It's not a good idea to evaluate how well your model performs after you release it to customers/users.
- We divide the model into a train set and validation set.
 - A common division choice 80% for training – 20% for validation
- Learn the model with the data in training set, evaluate the model with the validation set (the data that model have not used for training)



Training - Validation - Test Datasets

- It's not a good idea to evaluate how well your model performs after you release it to customers/users.
- We divide the model into a train set and validation set.
 - A common division choice 80% for training – 20% for validation
- Learn the model with the data in training set, evaluate the model with the validation set (the data that model have not used for training)
- After you finalise your model, you can use a third test set to estimate the final performance of your model.
 - Test set must never be used for further tuning your model.



Other Challenges

- Data quantity
 - Most ML algorithms require large datasets to achieve a sufficient performance (unlike children who can learn complex things from a few examples)

Other Challenges

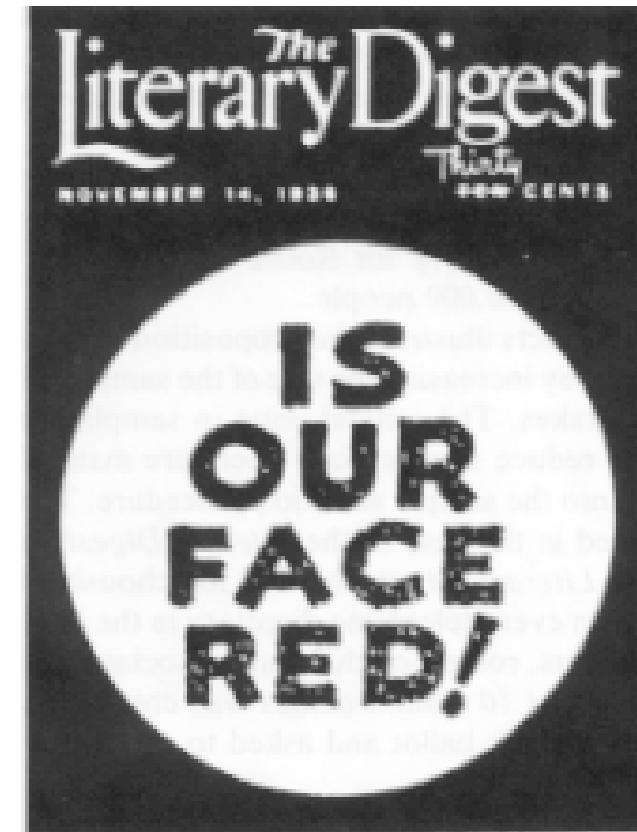
- Data quantity
 - Most ML algorithms require large datasets to achieve a sufficient performance (unlike children who can learn complex things from a few examples)
- Data quality
 - Missing data, outliers, errors will make it difficult to learn the underlying patterns

Other Challenges

- Data quantity
 - Most ML algorithms require large datasets to achieve a sufficient performance (unlike children who can learn complex things from a few examples)
- Data quality
 - Missing data, outliers, errors will make it difficult to learn the underlying patterns
- Nonrepresentative training data

Other Challenges - Non-representative Training Data

- In order to generalize, training data should be representative of the new cases that you want to generalize
- Sampling bias



Other Challenges

- Data quantity
 - Most ML algorithms require large datasets to achieve a sufficient performance (unlike children who can learn complex things from a few examples)
- Data quality
 - Missing data, outliers, errors will make it difficult to learn the underlying patterns
- Nonrepresentative training data
- Irrelevant features
 - Garbage-in, garbage-out

COGS514 - Cognition and Machine Learning

Linear Algebra Review

Linear Algebra is for essential for ML.

Example Problem - Iris Flower Dataset

iris setosa



petal

sepal

iris versicolor



petal

sepal

iris virginica



petal

sepal

Example Problem - Iris Flower Dataset

Sepal Length	Sepal Width	Petal Length	Petal Width	Flower
5.1	3.5	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
:	:	:	:	:



Features as vectors

$$x^{(0)} = \begin{bmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix}$$

Sepal Length	Sepal Width	Petal Length	Petal Width	Flower
5.1	3.5	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
⋮	⋮	⋮	⋮	⋮

Features as vectors

$$x^{(0)} = \begin{bmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix} \quad y^{(0)} = -1$$

Sepal Length	Sepal Width	Petal Length	Petal Width	Flower
5.1	3.5	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
⋮	⋮	⋮	⋮	⋮

Features as vectors

$$x^{(1)} = \begin{bmatrix} 7.0 \\ 3.2 \\ 4.7 \\ 2.4 \end{bmatrix} \quad y^{(1)} = +1$$

Sepal Length	Sepal Width	Petal Length	Petal Width	Flower
5.1	3.5	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
⋮	⋮	⋮	⋮	⋮

Classifier models as vectors

$$w = \begin{bmatrix} 1.4 \\ 2.1 \\ -0.7 \\ 0.5 \end{bmatrix} \quad w_0 = -18.1$$

Prediction with classifiers as dot product

$$\hat{y}^{(i)} = \begin{cases} +1 & \text{if } w \cdot x^{(i)} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

Prediction with classifiers as dot product

$$\hat{y}^{(i)} = \begin{cases} +1 & \text{if } w \cdot x^{(i)} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

e.g.

$$w = \begin{bmatrix} 1.4 \\ 2.1 \\ -0.7 \\ 0.5 \end{bmatrix}, \quad w_0 = -18.1$$

Prediction with classifiers as dot product

$$\hat{y}^{(i)} = \begin{cases} +1 & \text{if } w \cdot x^{(i)} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$w = \begin{bmatrix} 1.4 \\ 2.1 \\ -0.7 \\ 0.5 \end{bmatrix}, \quad w_0 = -18.1, \quad x^{(1)} = \begin{bmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix}, \quad y^{(1)} = -1$$

Prediction with classifiers as dot product

$$\hat{y}^{(i)} = \begin{cases} +1 & \text{if } w \cdot x^{(i)} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$w = \begin{bmatrix} 1.4 \\ 2.1 \\ -0.7 \\ 0.5 \end{bmatrix}, \quad w_0 = -18.1, \quad x^{(1)} = \begin{bmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix}, \quad y^{(1)} = -1$$

$$w \cdot x^{(i)} + w_0 = 1.4 \times 5.1 + 2.1 \times 3.5 + -0.7 \times 1.4 + 0.5 \times 0.2 - 18.1 = -4.49$$

Prediction with classifiers as dot product

$$\hat{y}^{(i)} = \begin{cases} +1 & \text{if } w \cdot x^{(i)} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$w = \begin{bmatrix} 1.4 \\ 2.1 \\ -0.7 \\ 0.5 \end{bmatrix}, \quad w_0 = -18.1, \quad x^{(1)} = \begin{bmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix}, \quad y^{(1)} = -1$$

$$w \cdot x^{(i)} + w_0 = 1.4 \times 5.1 + 2.1 \times 3.5 + -0.7 \times 1.4 + 0.5 \times 0.2 - 18.1 = -4.49$$

$$-4.49 \leq 0, \quad \text{so}$$

$$\hat{y}^{(i)} = -1$$

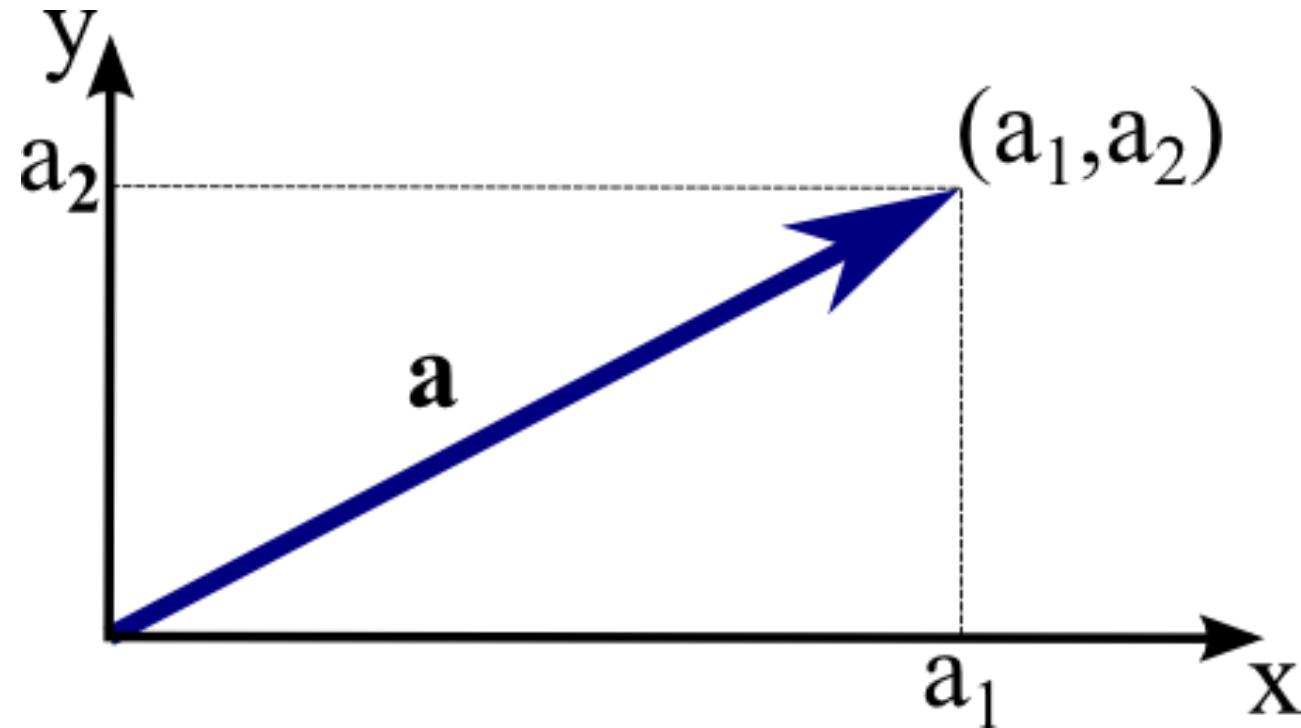
Vectors

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2$$

$$r = \begin{bmatrix} 10 \\ 50 \\ 1000 \end{bmatrix} \in \mathbb{R}^3$$

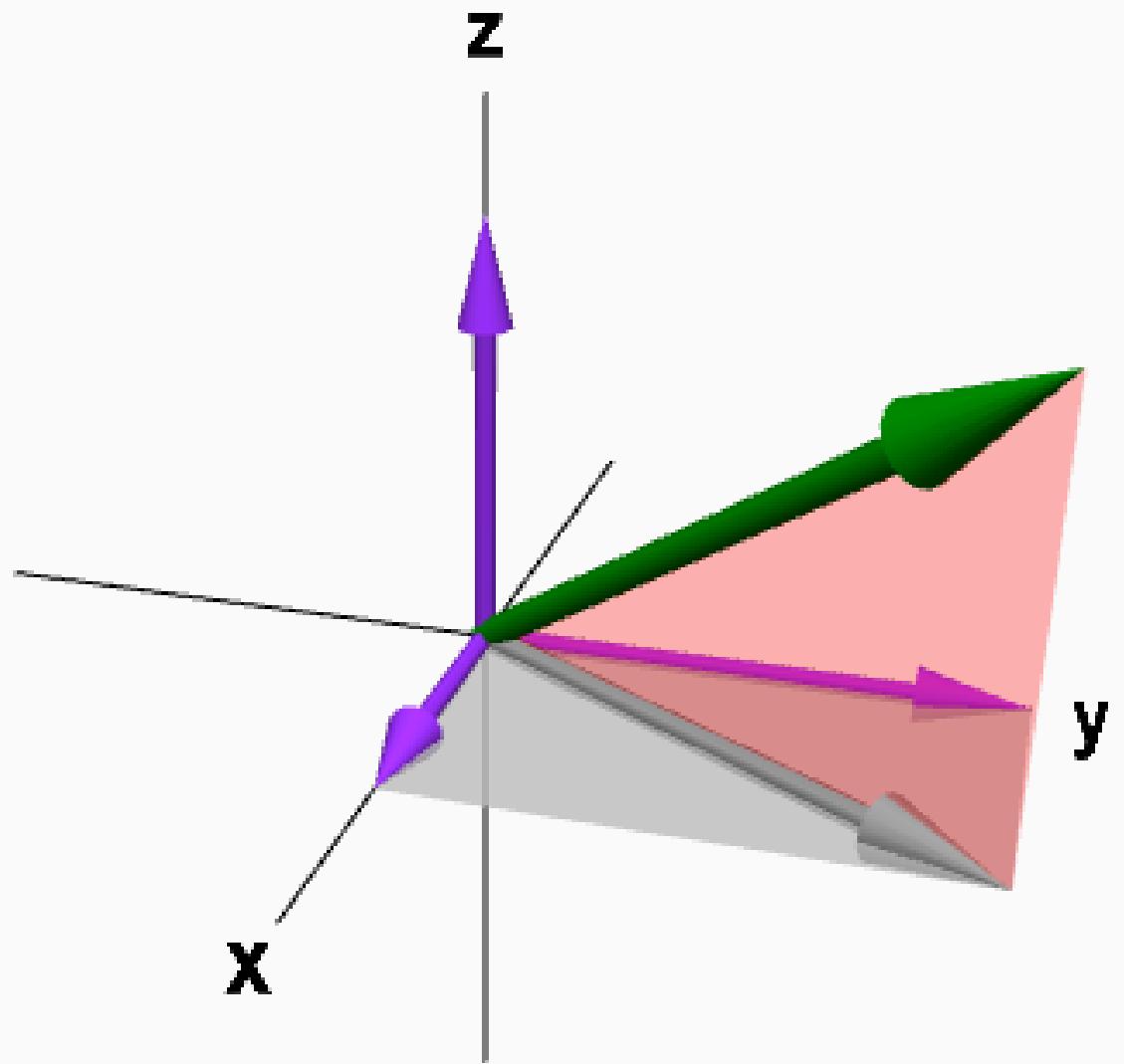
Vectors

$$a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \in \mathbb{R}^2$$



Vectors

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \in \mathbb{R}^3$$



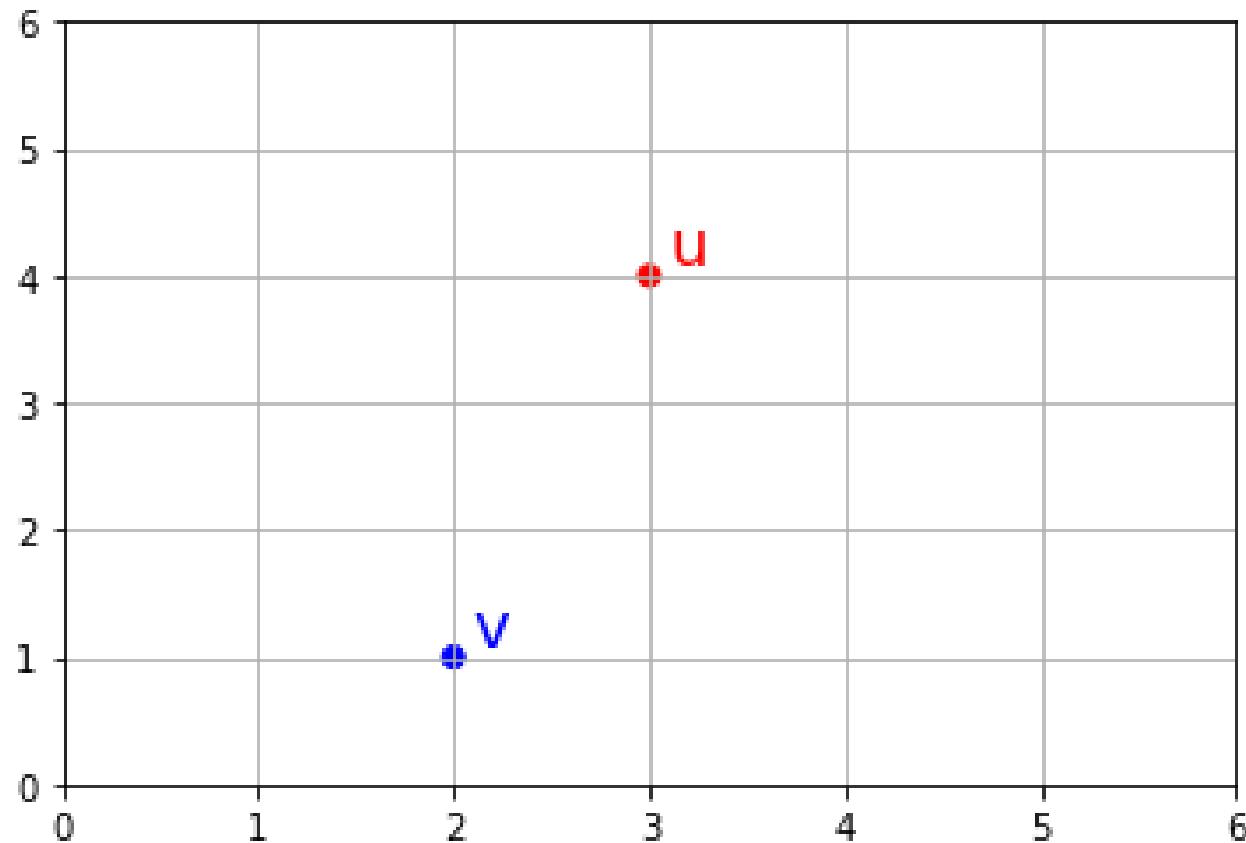
Vectors

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^n$$

Vectors

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2$$

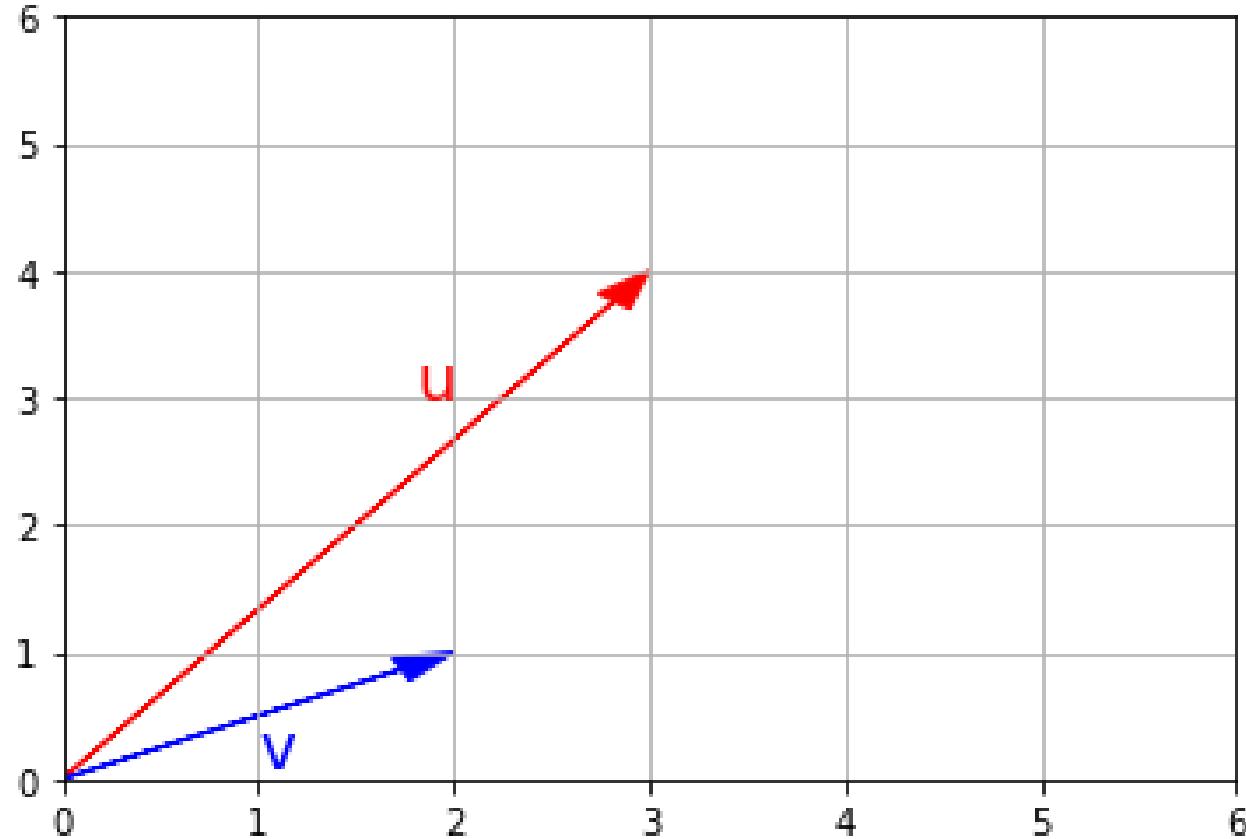
$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$



Vectors

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2$$

$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$



Vector - Summation

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2$$

$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

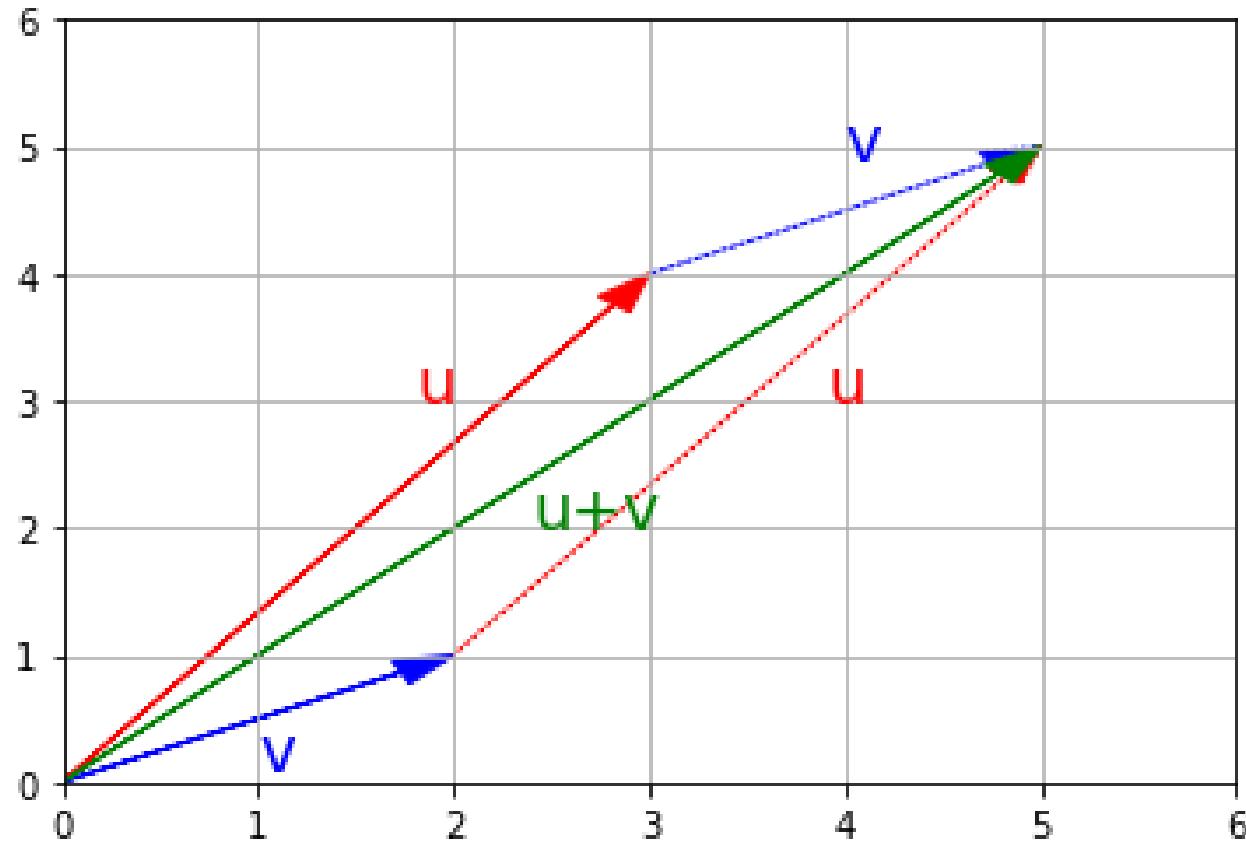
$$u + v =$$

Vector - Summation

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2$$

$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

$$u + v = z = \begin{bmatrix} 3 + 2 \\ 4 + 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \in \mathbb{R}^2$$



Vector - Summation

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^n \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{bmatrix} \in \mathbb{R}^n$$

$$a + b = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ \vdots \\ a_n + b_n \end{bmatrix} \in \mathbb{R}^n$$

Vector - Multiplication by a Scalar

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2 \quad v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

$$\lambda = 1.5 \in \mathbb{R}$$

$$\lambda u =$$

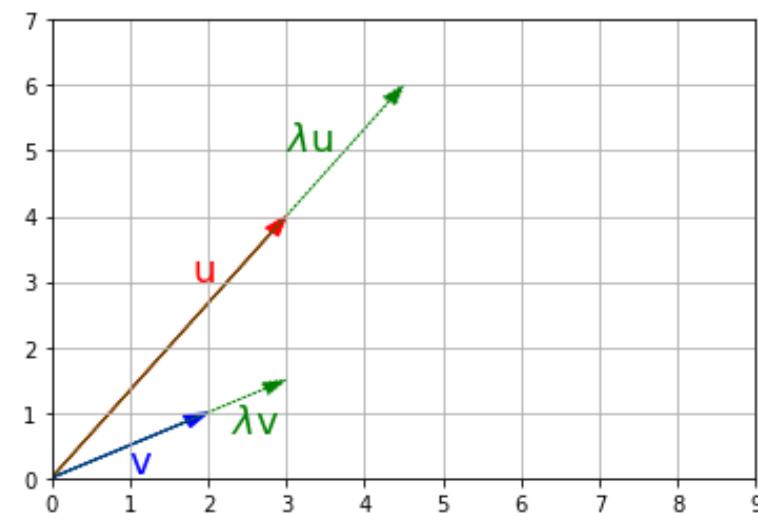
Vector - Multiplication by a Scalar

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2$$

$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

$$\lambda = 1.5 \in \mathbb{R}$$

$$\lambda u = \begin{bmatrix} 1.5 \times 3 \\ 1.5 \times 4 \end{bmatrix} = \begin{bmatrix} 4.5 \\ 6 \end{bmatrix} \quad b = \begin{bmatrix} 3 \\ 1.5 \end{bmatrix}$$



Vector - Multiplication by a Scalar

$$\lambda \in \mathbb{R}, \quad a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^n$$

$$\lambda a = \begin{bmatrix} \lambda a_1 \\ \lambda a_2 \\ \vdots \\ \lambda a_n \end{bmatrix} \in \mathbb{R}^n$$

Dot Product (Inner Product)

Dot Product (Inner Product) of Two Vectors

$$a, b \in \mathbb{R}^n$$

$$a \cdot b = \sum_{i=1}^n a_i b_i = c \in \mathbb{R}$$

Dot Product (Inner Product) of Two Vectors

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2 \qquad v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

$$u \cdot v = 3 \times 2 + 4 \times 1 = 10$$

Dot Product (Inner Product) of Two Vectors

Throughout the course we will show dot product with $a \cdot b$ and $a^T b$. They mean the same thing.

$$a, b \in \mathbb{R}^n$$

$$a \cdot b = a^T b = \sum_{i=1}^n a_i b_i = c \in \mathbb{R}$$

Dot Product (Inner Product) of Two Vectors

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2 \qquad v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

$$u \cdot v \qquad \text{or} \qquad u^T v$$

Dot Product (Inner Product) of Two Vectors

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2, \quad v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

$$\begin{aligned} u \cdot v &= u^T v = \begin{bmatrix} 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \\ &= 3 \times 2 + 4 \times 1 = 10 \end{aligned}$$

Length of a Vector (Norm)

Length of a Vector $\|x\|$

(Norm of a vector)

$$\|a\|_2 = \sqrt{\sum_{i=1}^n a_i^2} = \sqrt{a^T a}$$

- Squared root of dot product of a vector with itself

Length of a Vector $\|x\|$

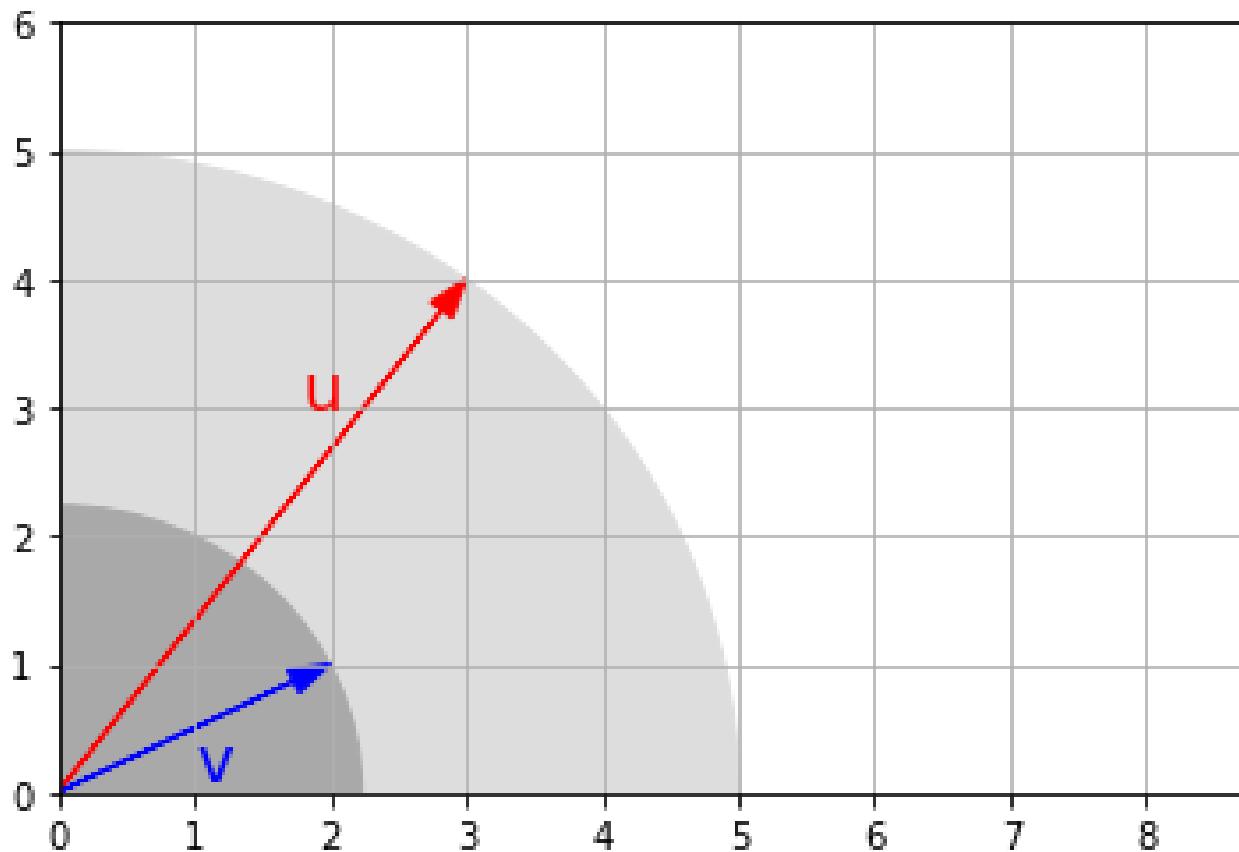
L2 Norm $\|x\|_2$

$$\|a\|_2 = \sqrt{\sum_{i=1}^n a_i^2} = \sqrt{a^T a}$$

Length (L2 Norm) Example

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2 \quad \|u\|_2 = \sqrt{3^2 + 4^2} = 5$$

$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2 \quad \|v\|_2 = \sqrt{2^2 + 1^2} = 2.24$$



Norm of a Vector $\|x\|$

L1 Norm

$$\|a\|_1 = \sum_{i=1}^n |a_i|$$

Norm of a Vector $\|x\|$

L1 Norm

$$\|a\|_1 = \sum_{i=1}^n |a_i|$$

L2 Norm

$$\|a\|_2 = \sqrt{\sum_{i=1}^n a_i^2} = \sqrt{a^T a}$$

Norm of a Vector $\|x\|$

L1 Norm

$$\|a\|_1 = \sum_{i=1}^n |a_i|$$

L2 Norm

$$\|a\|_2 = \sqrt{\sum_{i=1}^n a_i^2} = \sqrt{a^T a}$$

L ∞ Norm

$$\|a\|_\infty = \max_i |a_i|$$

Norm of a vector $\|x\|$

Norm is any function $\|\cdot\| : \mathbb{R}^n \rightarrow \mathbb{R}$ that:

1. is positive definite $\|x\| \geq 0, \|x\| = 0 \iff x = 0$
2. is homogeneous $\|\lambda x\| = |\lambda| \|x\|$
3. satisfies triangle inequality $\|x + y\| \leq \|x\| + \|y\|$

where $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$

Zero vector

Zero vector is a vector of 0s

$$0 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^n$$

Unit vector

Unit vector is a vector with a L2 norm that is equal to 1

For example:

$$\hat{i} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^3 \quad \hat{p} = \begin{bmatrix} 12/13 \\ -3/13 \\ -4/13 \end{bmatrix} \in \mathbb{R}^3$$

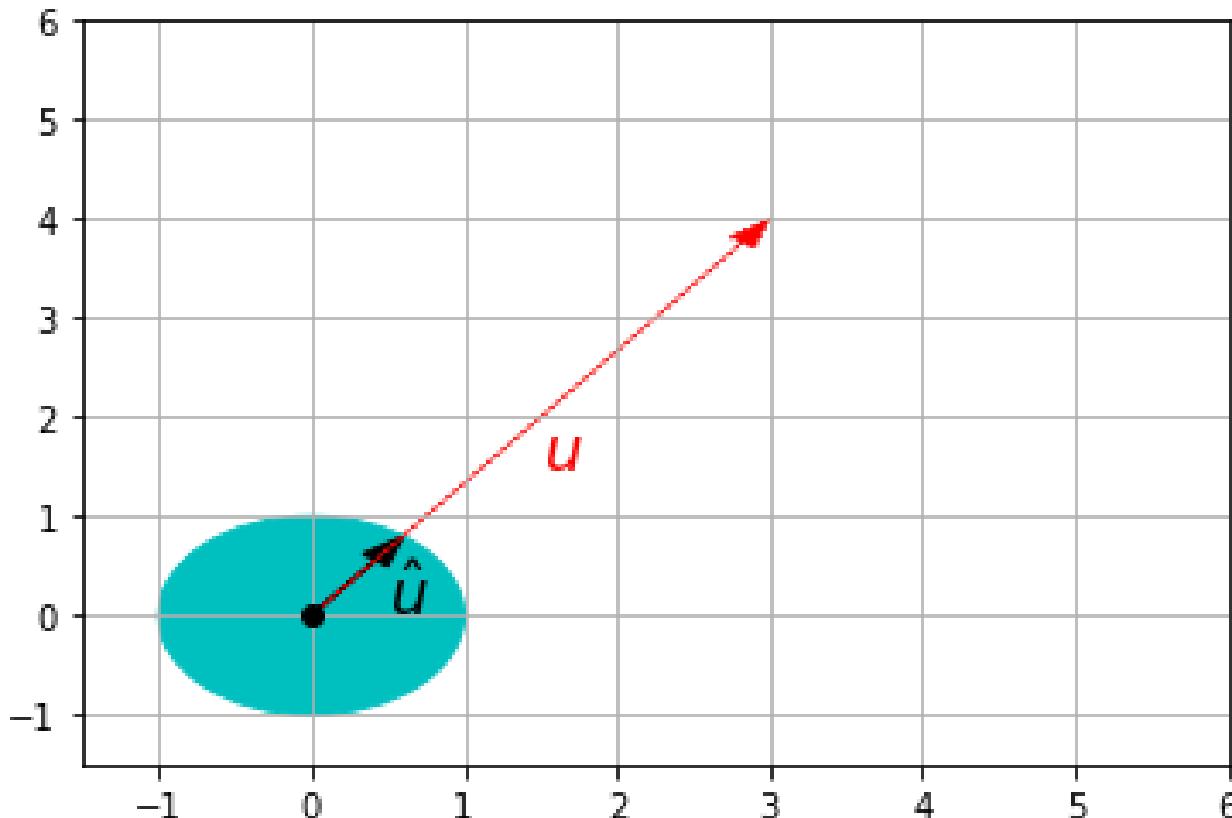
Normalized Vector

Normalized vector \hat{u} of vector u is the unit vector that points the same direction as u

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2$$

$$\hat{u} = \frac{u}{\|u\|_2}$$

$$u = \begin{bmatrix} 3/5 \\ 4/5 \end{bmatrix} \in \mathbb{R}^2$$



Angle Between Vectors

Dot Product and Angle Between Vectors

$$u \cdot v = \|u\|_2 \|v\|_2 \cos(\theta)$$

Dot Product and Angle Between Vectors

$$u \cdot v = \|u\|_2 \|v\|_2 \cos(\theta)$$

$$\cos(\theta) = \frac{u \cdot v}{\|u\|_2 \|v\|_2}$$

Dot Product and Angle Between Vectors

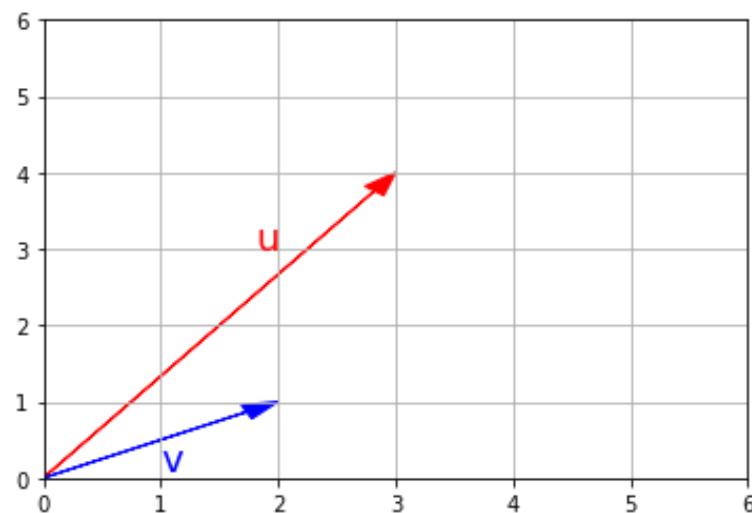
$$u \cdot v = \|u\|_2 \|v\|_2 \cos(\theta)$$

$$\theta = \arccos \left(\frac{u \cdot v}{\|u\|_2 \|v\|_2} \right)$$

Dot Product and Angle Between Vectors

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2, \quad v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

$$\theta = \arccos \left(\frac{10}{5 \times 2.24} \right) = 0.46 \text{ rad} = 26.6^\circ$$



Orthogonal Vectors

Orthogonal Vectors

- Two vectors $a, b \in \mathbb{R}^n$ are orthogonal if $a \cdot b = 0$

Recap:

- A vector is normalized if $\|a\|_2 = 1$

Orthonormal Vectors

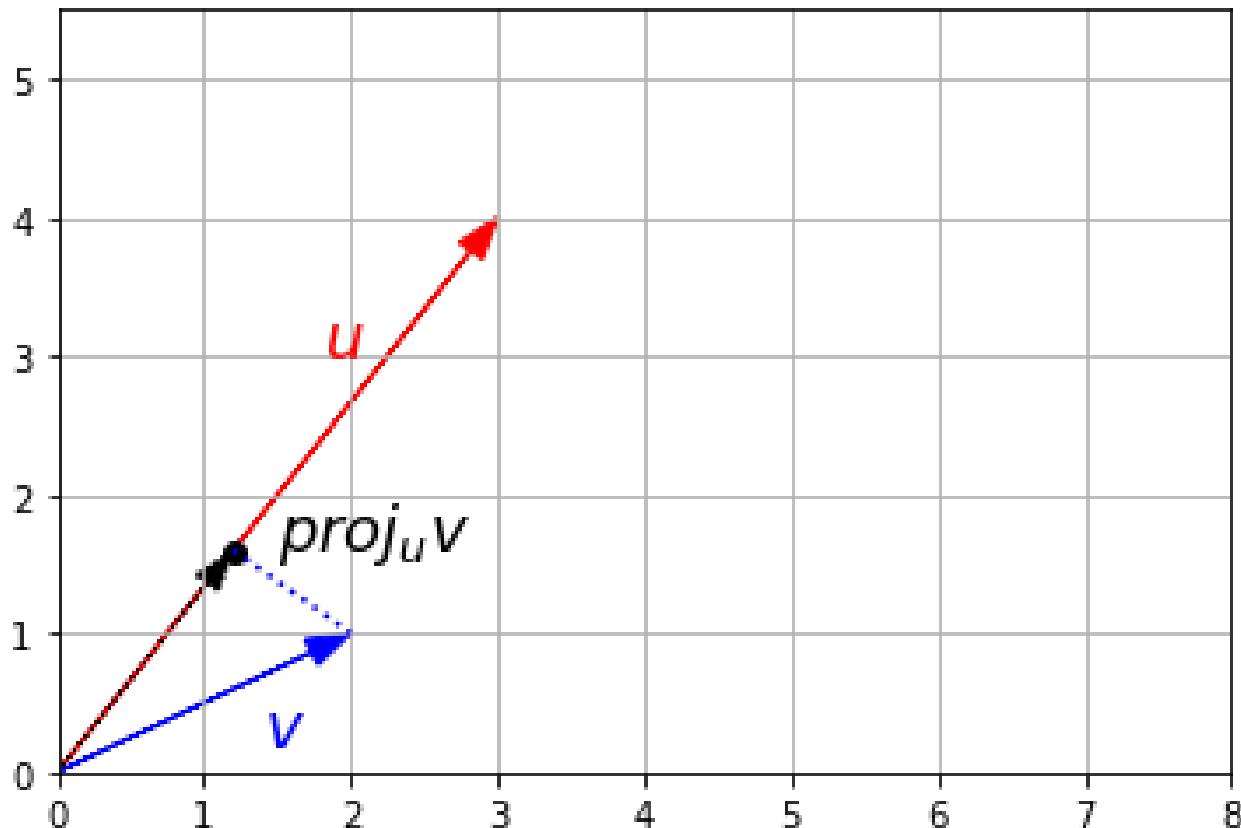
- Two vectors are orthonormal if they are *orthogonal* and *normalized*

Projecting onto an axis

$$\text{proj}_u v = \frac{u \cdot v}{\|u\|^2} u$$

$$\text{proj}_u v = (\hat{u} \cdot v) \hat{u}$$

$$\text{proj}_u v = \frac{10}{5^2} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.6 \end{bmatrix}$$



Matrix

Matrices

$$A \in \mathbb{R}^{m \times n}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \quad a_{ij} \in \mathbb{R}$$

Matrices - Multiplication by a Scalar

$$A \in \mathbb{R}^{m \times n}, \quad \lambda \in \mathbb{R}$$

$$\lambda A = \begin{bmatrix} \lambda a_{11} & \lambda a_{12} & \dots & \lambda a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \dots & \lambda a_{2n} \\ \vdots & & & \vdots \\ \lambda a_{m1} & \lambda a_{m2} & \dots & \lambda a_{mn} \end{bmatrix}$$

Matrices - Summation

$$A \in \mathbb{R}^{m \times n}, \quad B \in \mathbb{R}^{m \times n}$$

$$A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \dots & a_{2n} + b_{2n} \\ \vdots & & & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \dots & a_{mn} + b_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

Matrix times Vector

Consider three vectors

$$u = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad v = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} \quad w = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

multiplied by three scalars x_1, x_2, x_3 .

$$x_1 \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} + x_2 \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} + x_3 \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Matrix multiplication Ax is a combination of the columns

$$Ax = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 2 \\ 5 & 6 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} + x_2 \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} + x_3 \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Matrix multiplication Ax is a combination of the columns

$$Ax = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 2 \\ 5 & 6 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} + x_2 \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} + x_3 \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Matrix multiplication Ax is a combination of the columns

$$Ax = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 2 \\ 5 & 6 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} =$$

Matrix multiplication Ax is a combination of the columns

$$Ax = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 2 \\ 5 & 6 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 * 1 + 0 * 2 + 2 * 3 \\ 1 * 3 + 0 * 4 + 2 * 2 \\ 1 * 5 + 0 * 6 + 2 * 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \\ 7 \end{bmatrix}$$

Matrix-Matrix Multiplication

$$AB = C \in \mathbb{R}^{m \times k}$$

$$c_{ij} = \sum_{l=1}^n a_{il} b_{lj}$$

$$AB = \begin{bmatrix} \sum_{l=1}^n a_{1l} b_{l1} & \sum_{l=1}^n a_{1l} b_{l2} & \dots & \sum_{l=1}^n a_{1l} b_{lk} \\ \sum_{l=1}^n a_{2l} b_{l1} & \sum_{l=1}^n a_{2l} b_{l2} & \dots & \sum_{l=1}^n a_{2l} b_{lk} \\ \vdots & & & \vdots \\ \sum_{l=1}^n a_{ml} b_{l1} & \sum_{l=1}^n a_{ml} b_{l2} & \dots & \sum_{l=1}^n a_{ml} b_{lk} \end{bmatrix} \in \mathbb{R}^{m \times k}$$

Matrix-Matrix Multiplication

$$AB = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 2 \\ 5 & 6 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 2 & 3 \end{bmatrix} =$$

$$AB = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 2 \\ 5 & 6 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 1*1 + 0*2 + 2*3 & -1*1 + 2*2 + 3*3 \\ 1*3 + 0*4 + 2*2 & -1*3 + 2*4 + 3*2 \\ 1*5 + 0*6 + 2*1 & -1*5 + 2*6 + 3*1 \end{bmatrix} = \begin{bmatrix} 7 & 12 \\ 7 & 11 \\ 7 & 10 \end{bmatrix}$$

Matrix-Matrix Multiplication

$$A \in \mathbb{R}^{m \times n}, \quad B \in \mathbb{R}^{n \times k}$$

$$AB = C \in \mathbb{R}^{m \times k}$$

Matrix Multiplication

$$A \in \mathbb{R}^{m \times n}, \quad B \in \mathbb{R}^{n \times k}$$

$$AB = C \in \mathbb{R}^{m \times k}$$

- In order to multiply AB , the number of columns of A must be equal to the number of rows of B

$$A \in \mathbb{R}^{m \times \boxed{n}}, \quad B \in \mathbb{R}^{\boxed{n} \times k}$$

Matrix Multiplication

$$A \in \mathbb{R}^{m \times n}, \quad B \in \mathbb{R}^{n \times k}$$

$$AB = C \in \mathbb{R}^{m \times k}$$

- In order to multiply AB , the number of columns of A must be equal to the number of rows of B

$$A \in \mathbb{R}^{m \times n}, \quad B \in \mathbb{R}^{n \times k}$$

- The size of the resulting matrix $AB = C$ is $\mathbb{R}^{m \times k}$ i.e.

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$AB = ?$$

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$AB = ?$$

1. Can we calculate AB ?

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$AB = ?$$

1. Can we calculate AB ? 

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$AB = ?$$

1. Can we calculate AB ? 
2. What will be the size of the resulting matrix?

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$AB = ?$$

1. Can we calculate AB ? 
2. What will be the size of the resulting matrix? $\mathbb{R}^{2 \times 2}$

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3}, \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$AB = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 2 & 5 \end{bmatrix} \in \mathbb{R}^{2 \times 2}$$

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$BA = ?$$

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

1. Can we calculate BA ?

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

1. Can we calculate BA ? 

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

1. Can we calculate BA ? 
2. What will be the size of the resulting matrix?

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

1. Can we calculate BA ? 
2. What will be the size of the resulting matrix? $\mathbb{R}^{3 \times 3}$

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

Matrix Multiplication - Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3}$$

$$B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$BA = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 4 & 2 \\ -2 & 0 & 2 \\ 3 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 3}$$

Matrix Multiplication Properties

Associative

$$\forall A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}, C \in \mathbb{R}^{p \times q} : (AB)C = A(BC)$$

Matrix Multiplication Properties

Associative

$$\forall A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}, C \in \mathbb{R}^{p \times q} : (AB)C = A(BC)$$

Distributive

$$\forall A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times n}, C \in \mathbb{R}^{n \times p}, D \in \mathbb{R}^{n \times p} : (A + B)C = (AC) + (BC)$$

$$A(C + D) = (AC) + (AD)$$

Matrix Multiplication Properties

NOT commutative

$$AB \neq BA$$

Square Matrix

$$A \in \mathbb{R}^{n \times n} \quad A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

$$X \in \mathbb{R}^{3 \times 3} \quad X = \begin{bmatrix} 9 & 4 & 2 \\ 7 & 1 & 5 \\ 8 & 1 & 6 \end{bmatrix}$$

Diagonal Matrix

$$D \in \mathbb{R}^{n \times n} \quad D = \begin{bmatrix} d_{11} & 0 & \dots & 0 \\ 0 & d_{22} & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & d_{nn} \end{bmatrix}$$

$$D \in \mathbb{R}^{3 \times 3} \quad D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Upper Triangular Matrix

$$U \in \mathbb{R}^{n \times n} \quad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & u_{nn} \end{bmatrix}$$

$$U \in \mathbb{R}^{3 \times 3} \quad U = \begin{bmatrix} 9 & 4 & 2 \\ 0 & 1 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

Lower Triangular Matrix

$$L \in \mathbb{R}^{n \times n} \quad L = \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \vdots & & & & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix}$$

$$L \in \mathbb{R}^{3 \times 3} \quad L = \begin{bmatrix} 6 & 0 & 0 \\ 4 & 5 & 0 \\ 7 & 2 & 6 \end{bmatrix}$$

Identity Matrix

The identity matrix is a square matrix where the diagonal elements are **ones** and the other elements are **zeros**. For example $I_n \in \mathbb{R}^{n \times n}$:

$$I_n = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

Identity Matrix

The identity matrix is a square matrix where the diagonal elements are **ones** and the other elements are **zeros**. For example $I_n \in \mathbb{R}^{n \times n}$.

$$I_n = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

$$\forall A \in \mathbb{R}^{m \times n}, \quad AI_n = I_m A = A$$

Transpose X^T

Transpose of a matrix is the results of changing rows with columns

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \quad a_{ij} \in \mathbb{R}$$

Transpose X^T

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \quad a_{ij} \in \mathbb{R}$$

$$\mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & & & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix} \in \mathbb{R}^{n \times m}, \quad a_{ij} \in \mathbb{R}$$

Transpose - Example

$$B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

Transpose - Example

$$B = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 2}$$

$$B^T = \begin{bmatrix} 0 & 1 & 0 \\ 2 & -1 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3}$$

Transpose

$$(\mathbf{A}^T)^T = \mathbf{A}$$

Transpose

$$(\mathbf{A}^T)^T = \mathbf{A}$$

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$$

Transpose

$$(\mathbf{A}^T)^T = \mathbf{A}$$

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$$

$$(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$$

Symmetric Matrices

A square matrix $\mathbf{A} \in \mathbf{R}^{n \times n}$ is:

- Symmetric if $(\mathbf{A}^T) = \mathbf{A}$
- Anti-symmetric if $\mathbf{A} = -(\mathbf{A}^T)$

Matrix multiplication and Linear Transformation

Any linear transformation f that map n -dimensional vectors to m -dimensional vectors can be represented as an $m \times n$ matrix.

$$\mathbf{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3$$

$$f(\mathbf{u}) = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \end{bmatrix} \in \mathbb{R}^2$$

Matrix multiplication and Linear Transformation

Any linear transformation f that map n -dimensional vectors to m -dimensional vectors can be represented as an $m \times n$ matrix.

$$\mathbf{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3$$

$$f(\mathbf{u}) = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \end{bmatrix} \in \mathbb{R}^2$$

$$\mathbf{F} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \in \mathbb{R}^{2 \times 3}$$

$$f(\mathbf{u}) = \mathbf{F}\mathbf{u}$$

Matrix Multiplication - Linear Transformation

$$\mathbf{G} = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_m] \in \mathbb{R}^{3 \times m} \quad \mathbf{u}_i = \begin{bmatrix} u_{1i} \\ u_{2i} \\ u_{3i} \end{bmatrix} \in \mathbb{R}^3$$

Matrix Multiplication - Linear Transformation

$$\mathbf{G} = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_m] \in \mathbb{R}^{3 \times m} \quad \mathbf{u}_i = \begin{bmatrix} u_{1i} \\ u_{2i} \\ u_{3i} \end{bmatrix} \in \mathbb{R}^3$$
$$\mathbf{F}\mathbf{G} = [f(\mathbf{u}_1) \quad f(\mathbf{u}_2) \quad \dots \quad f(\mathbf{u}_m)]$$

Example - Shear Mapping

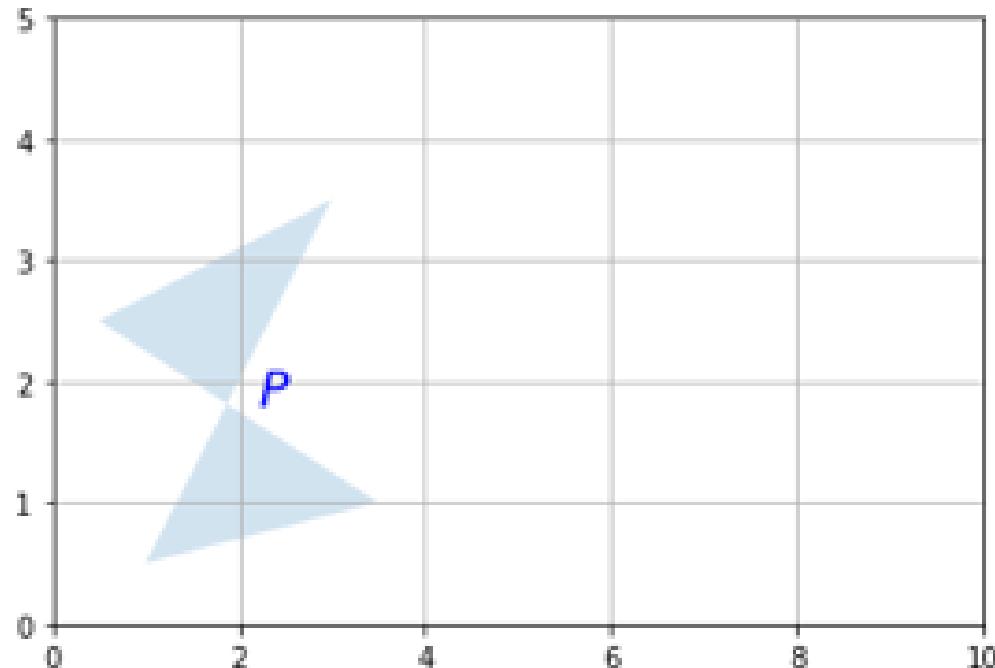
$$\mathbf{F}_{shear} = \begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix}$$

Example - Shear Mapping

$$\mathbf{F}_{shear} = \begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix}$$

Example:

$$\mathbf{P} = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$



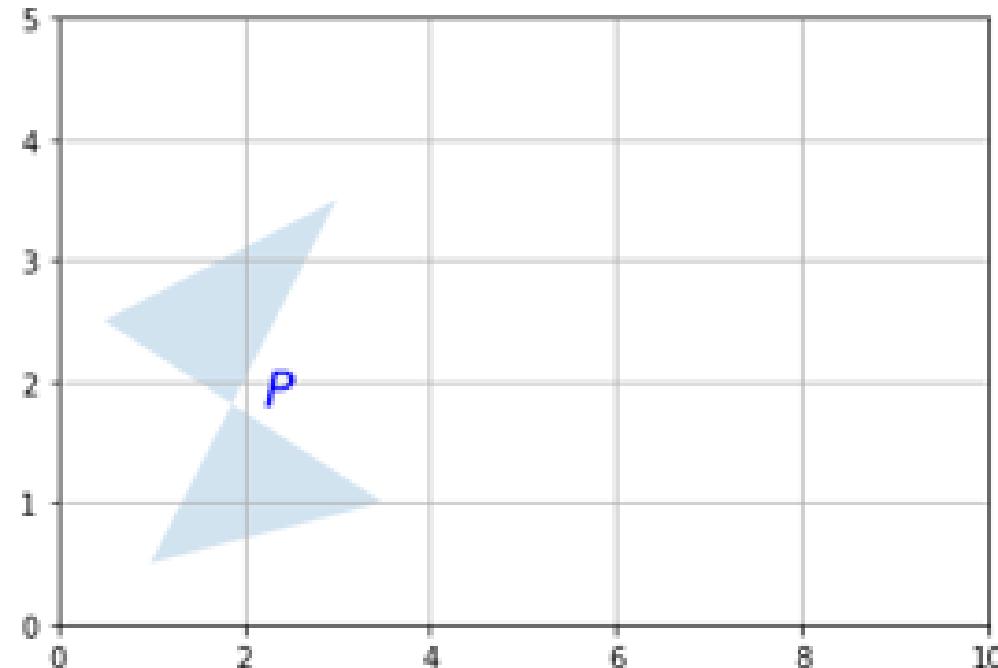
Example - Shear Mapping

$$\mathbf{F}_{shear} = \begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix}$$

Example:

$$\mathbf{P} = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{shear} \mathbf{P} = ?$$



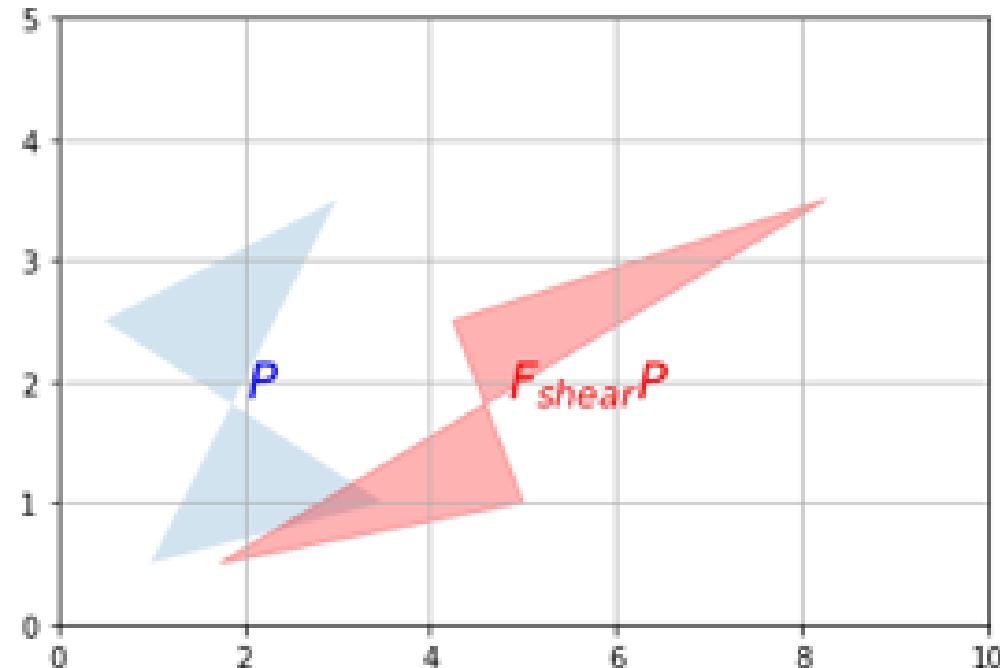
Example - Shear Mapping

$$\mathbf{F}_{shear} = \begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix}$$

Example:

$$\mathbf{P} = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{shear} \mathbf{P} = \begin{bmatrix} 1.75 & 8.25 & 4.25 & 5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix}$$



Example - Shear Mapping

$$\mathbf{F}_{shear} = \begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix}$$

Example:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$



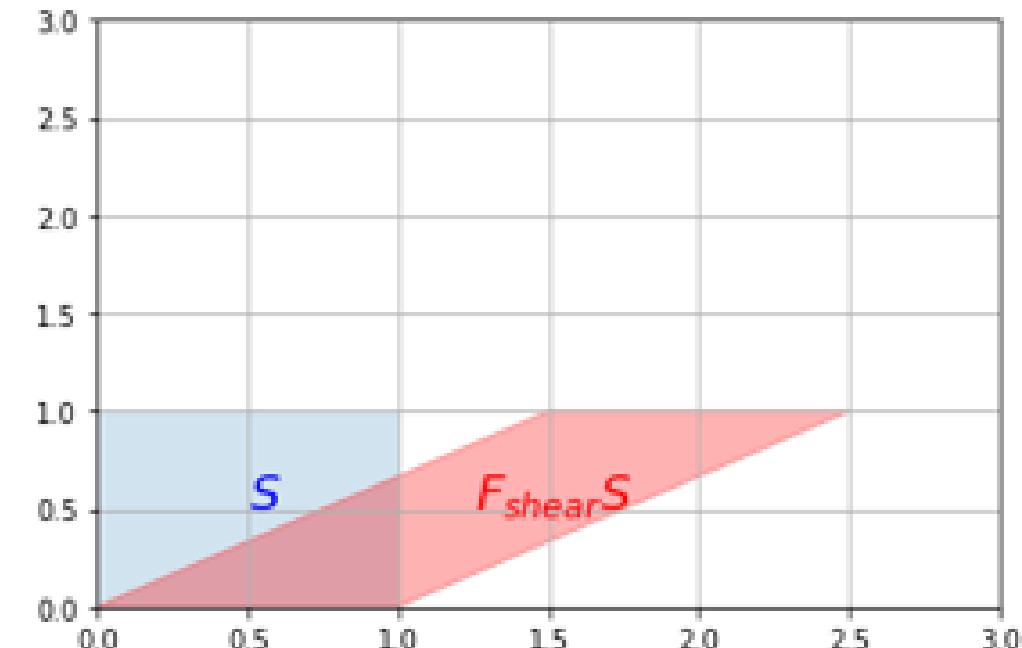
Example - Shear Mapping

$$\mathbf{F}_{shear} = \begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix}$$

Example:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{shear} \mathbf{S} = \begin{bmatrix} 0 & 1.5 & 2.5 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$



Example - Squeeze Mapping

$$\mathbf{F}_{squeeze} = \begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

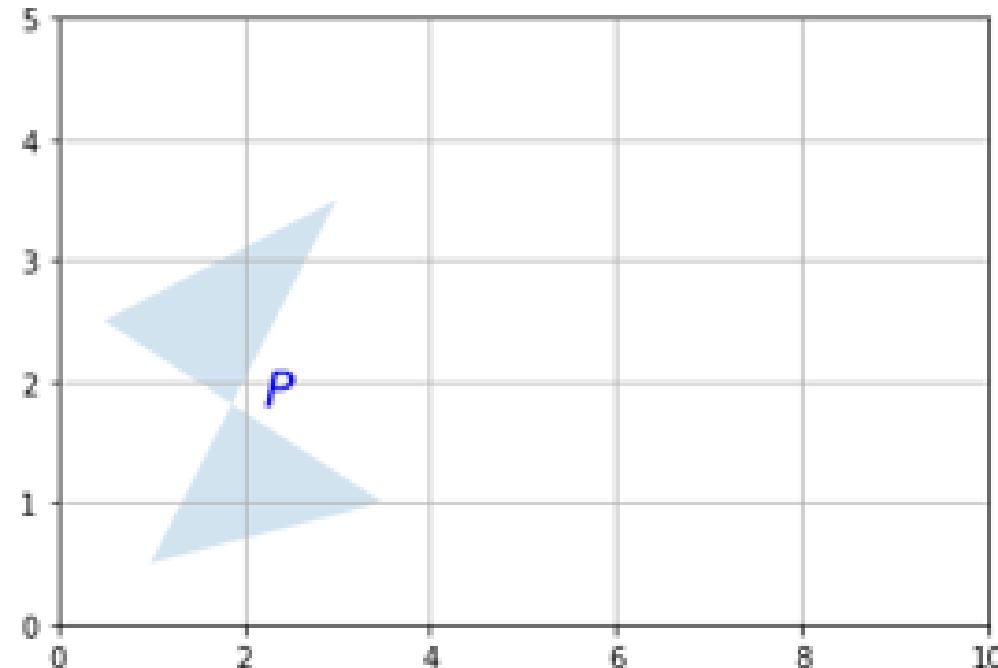
Example - Squeeze Mapping

$$\mathbf{F}_{squeeze} = \begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

Example:

$$\mathbf{P} = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{squeeze} \mathbf{P} = ?$$



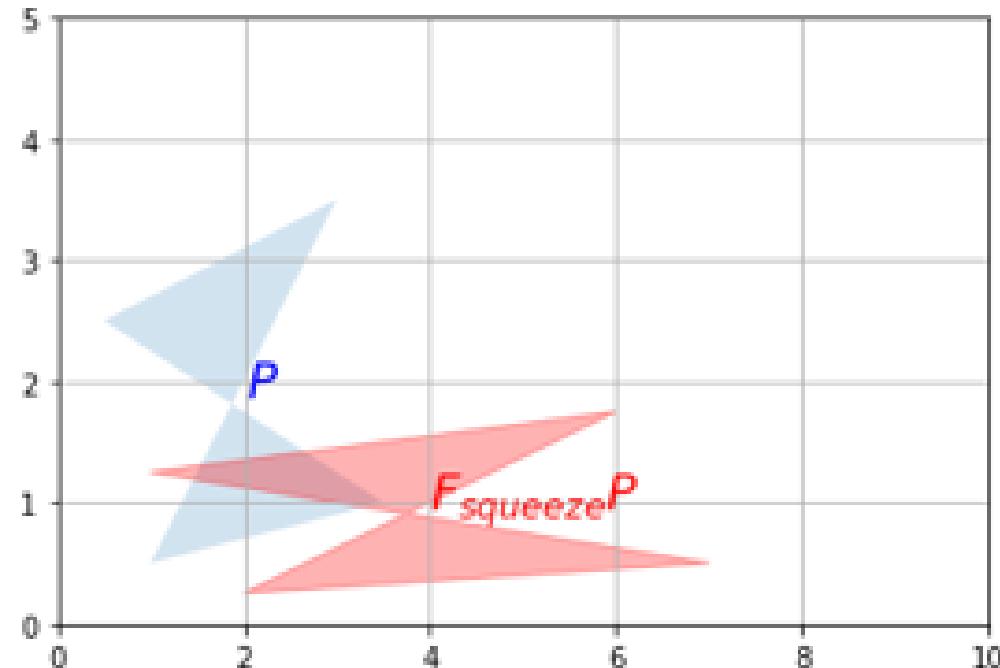
Example - Squeeze Mapping

$$\mathbf{F}_{squeeze} = \begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

Example:

$$\mathbf{P} = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{squeeze} \mathbf{P} = \begin{bmatrix} 2 & 6 & 1 & 7 \\ 0.25 & 1.75 & 1.25 & 0.5 \end{bmatrix}$$



Example Squeeze Mapping

$$\mathbf{F}_{squeeze} = \begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

Example:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$



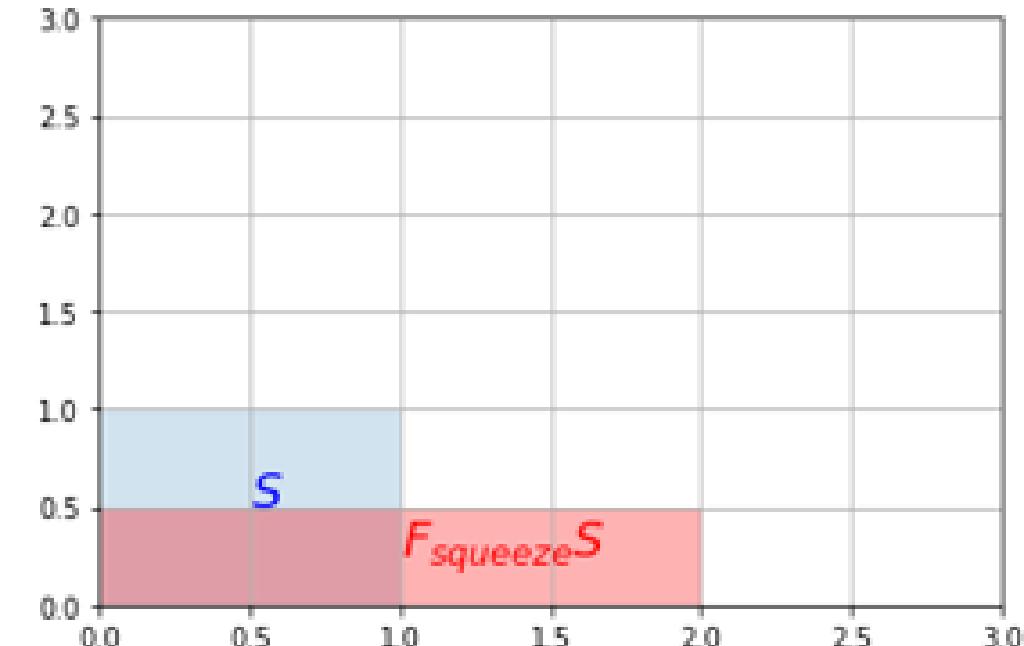
Example Squeeze Mapping

$$\mathbf{F}_{squeeze} = \begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

Example:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{squeeze} \mathbf{S} = \begin{bmatrix} 0 & 0 & 2 & 2 \\ 0 & .5 & .5 & 0 \end{bmatrix}$$



Rotation

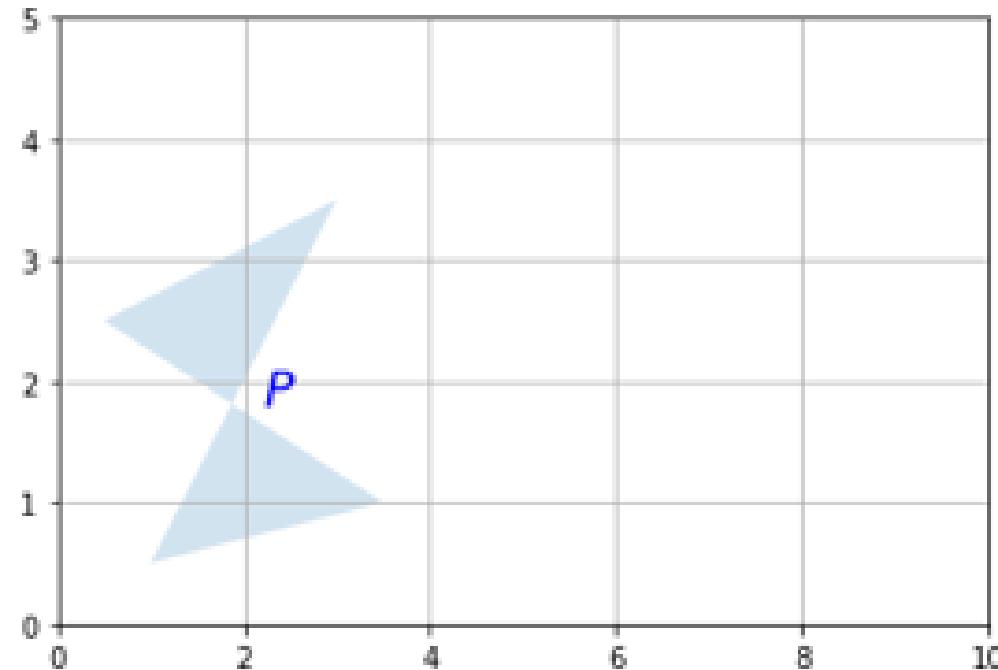
$$V = \begin{bmatrix} \cos 20^\circ & \sin 20^\circ \\ -\sin 20^\circ & \cos 20^\circ \end{bmatrix}$$

Rotation

$$V = \begin{bmatrix} \cos 20^\circ & \sin 20^\circ \\ -\sin 20^\circ & \cos 20^\circ \end{bmatrix}$$

Example:

$$P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$



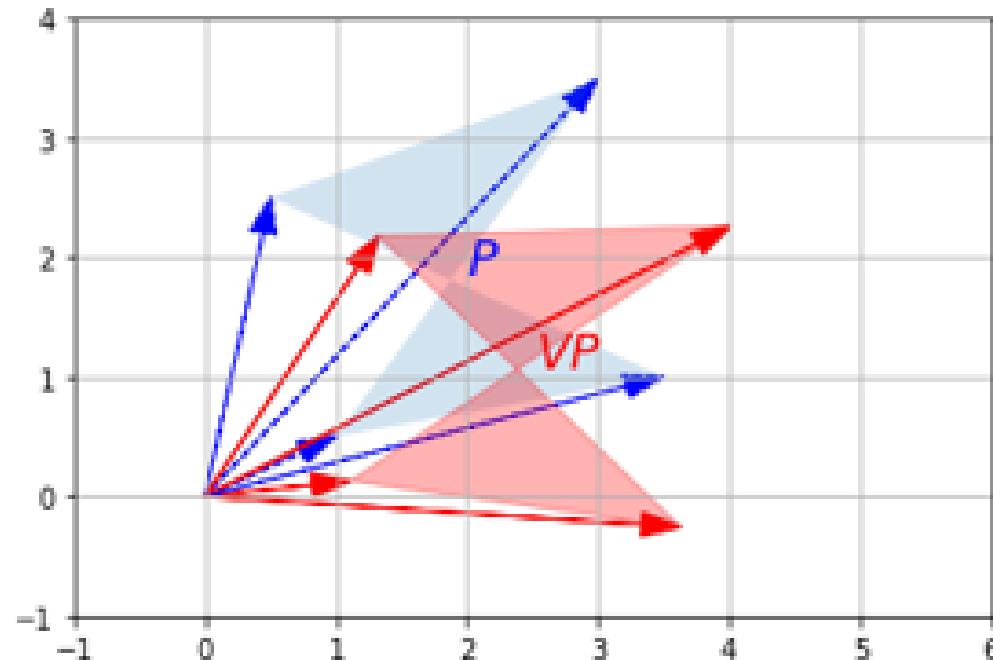
Rotation

$$V = \begin{bmatrix} \cos 20^\circ & \sin 20^\circ \\ -\sin 20^\circ & \cos 20^\circ \end{bmatrix}$$

Example:

$$P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

VP

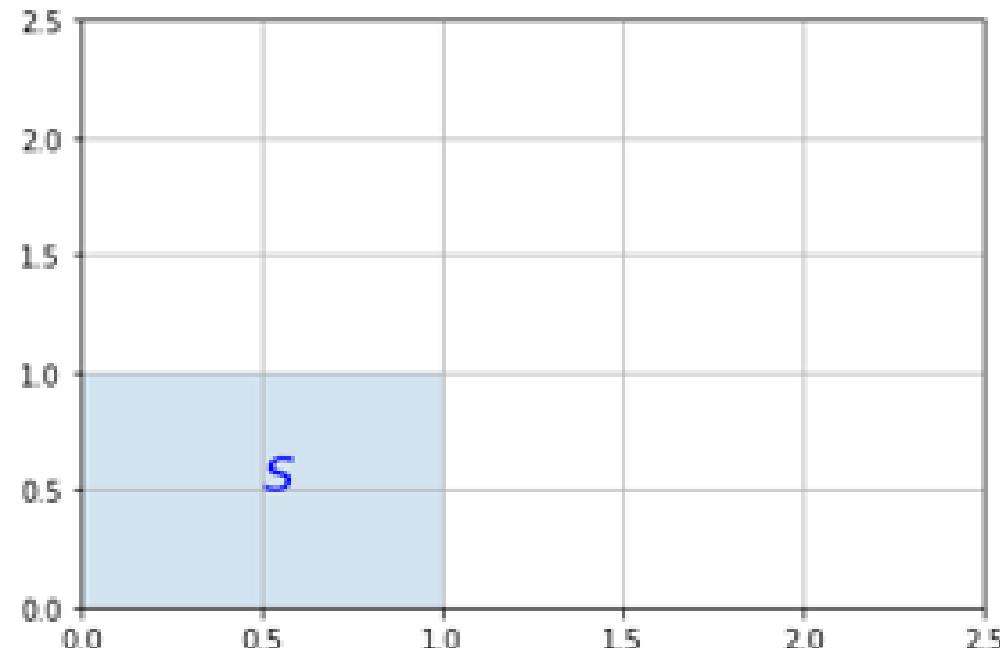


Rotation

$$V = \begin{bmatrix} \cos 20^\circ & \sin 20^\circ \\ -\sin 20^\circ & \cos 20^\circ \end{bmatrix}$$

Example:

$$S = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$



Rotation

$$V = \begin{bmatrix} \cos 20^\circ & \sin 20^\circ \\ -\sin 20^\circ & \cos 20^\circ \end{bmatrix}$$

Example:

$$S = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

VS



Inverse of a Matrix

- Inverse of a **square matrix** A is denoted by A^{-1}

$$A^{-1}A = AA^{-1} = I$$

Inverse of a Matrix - Properties

- $(A^{-1})^{-1} = A$
- $(AB)^{-1} = B^{-1}A^{-1}$
- $(A^{-1})^T = (A^T)^{-1}$

Inverse of a Matrix - Properties

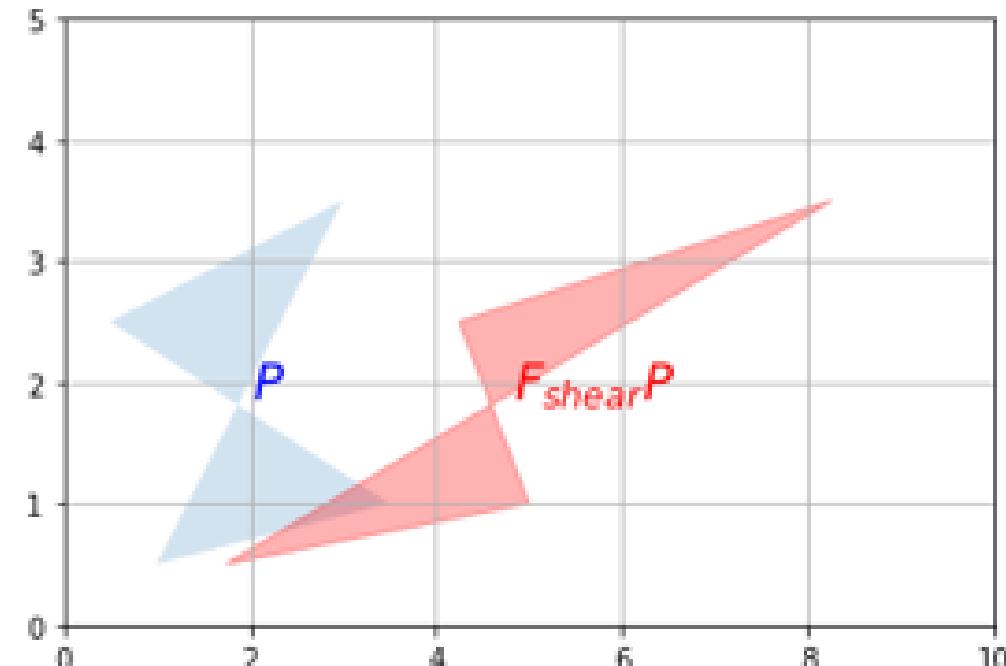
- Not all matrices have inverse. If matrix A^{-1} has inverse, we call A invertible or non-singular
- Non-square matrices do not have inverse
- Having an inverse means that the transformation you applied it is reversible
- For example,
 - shear mapping have an inverse
 - If you project a matrix onto a line, some information is lost, so this does not have an inverse...

Inverse Example - Shear Mapping

$$\mathbf{F}_{shear} = \begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{shear} \mathbf{P} = \begin{bmatrix} 1.75 & 8.25 & 4.25 & 5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} = \mathbf{Q}$$



Inverse Example - Shear Mapping

$$\mathbf{F}_{shear}^{-1} = \begin{bmatrix} 1 & -1.5 \\ 0 & 1 \end{bmatrix}$$

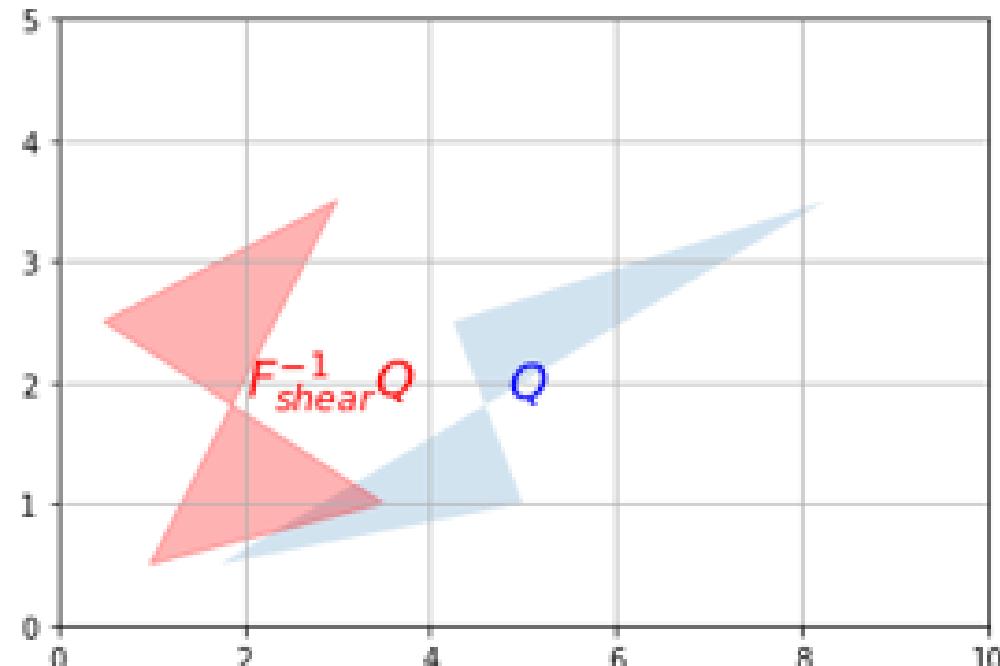
$$Q = \begin{bmatrix} 1.75 & 8.25 & 4.25 & 5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix}$$

Inverse Example - Shear Mapping

$$\mathbf{F}_{shear}^{-1} = \begin{bmatrix} 1 & -1.5 \\ 0 & 1 \end{bmatrix}$$

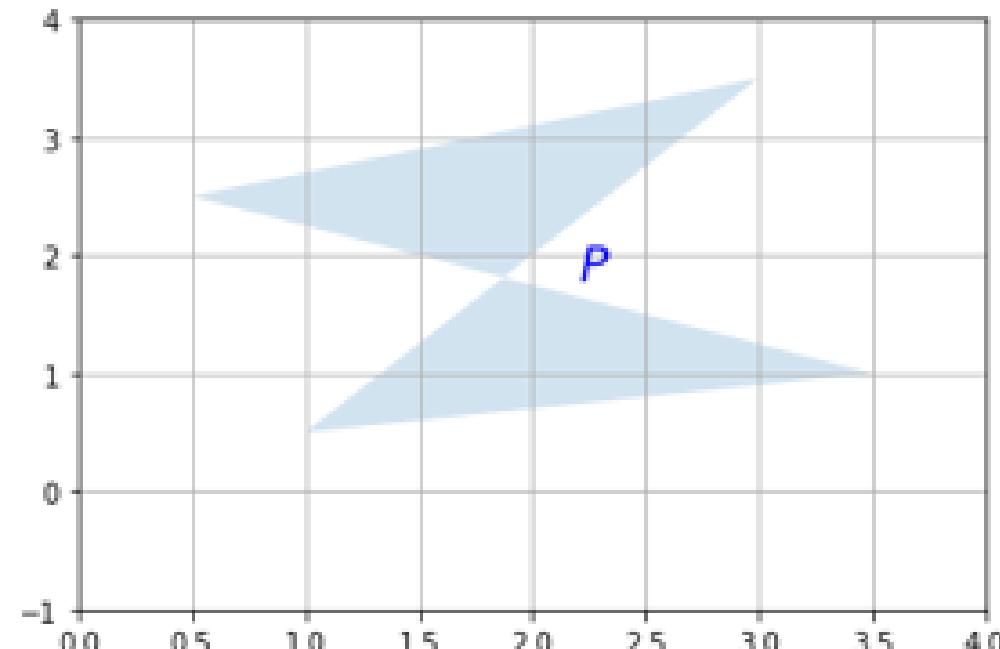
$$Q = \begin{bmatrix} 1.75 & 8.25 & 4.25 & 5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix}$$

$$\mathbf{F}_{shear}^{-1} Q$$



Inverse Example - Project onto a vector

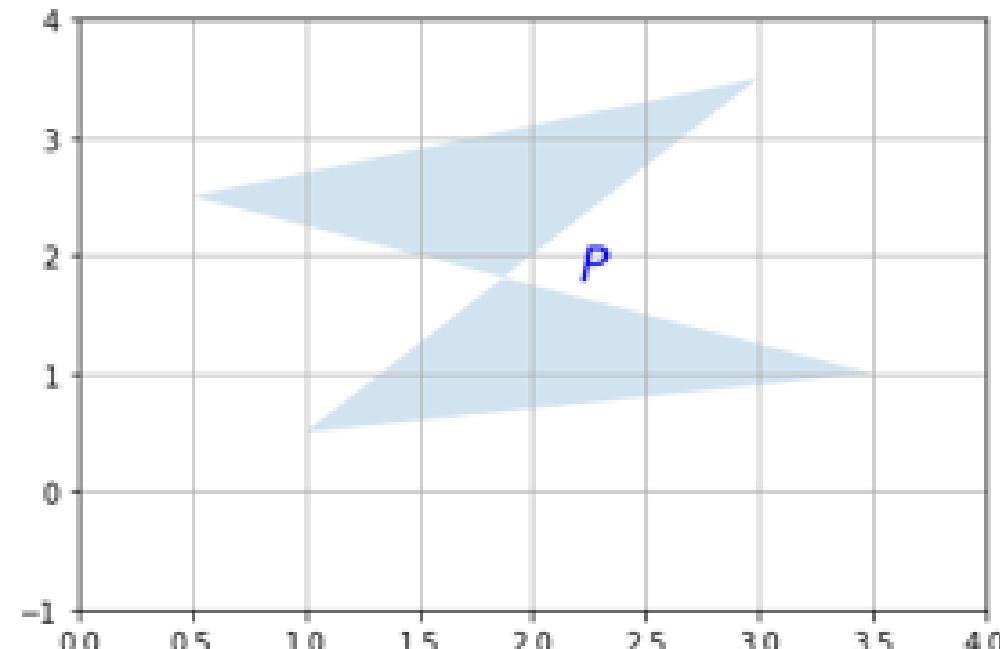
$$P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$



Inverse Example - Project onto a vector

$$\mathbf{P} = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{project} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

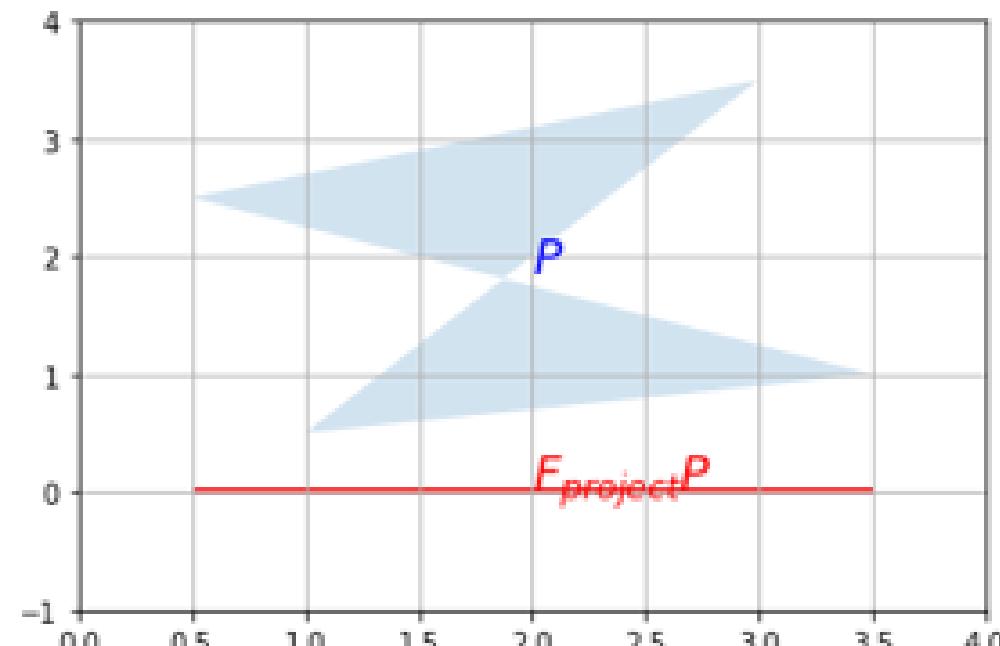


Inverse Example - Project onto a vector

$$P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$F_{project} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$F_{project}P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



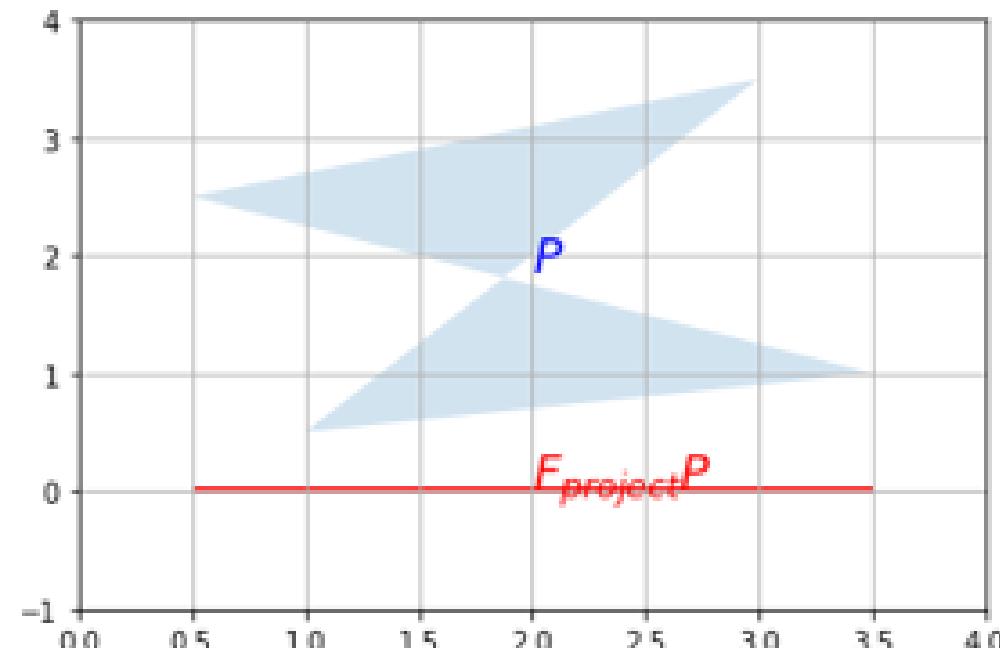
Inverse Example - Project onto a vector

$$P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$F_{project} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$F_{project}P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

There is no $F_{project}^{-1}$



Other Topics

Orthogonal Matrices

Recap Orthogonal Vectors:

- Two **vectors** $a, b \in \mathbb{R}^n$ are orthogonal if $a \cdot b = 0$
- A vector is normalized if $\|a\|_2 = 1$
- Two **vectors** are orthonormal if they are *orthogonal* and *normalized*

Orthogonal Matrices

- A square matrix $U \in \mathbb{R}^{n \times n}$ is orthogonal if all its *columns* are *orthonormal* to each other.

Orthogonal Matrices

- A square matrix $U \in \mathbb{R}^{n \times n}$ is orthogonal if all its *columns* are *orthonormal* to each other.
- Inverse of an orthogonal matrix is its transpose

$$U^T U = I = UU^T$$

- Operating on a vector with an orthogonal matrix does not change its norm

$$\|U\mathbf{x}\|_2 = \|\mathbf{x}\|_2$$

Eigenvalue and Eigenvector

For a square matrix $A \in \mathbb{R}^{n \times n}$, λ is an eigenvalue of A and x is an eigenvector if

$$Ax = \lambda x, \quad x \neq 0$$

- Multiplying A by vector x results in a new vector that is in the same direction as x but scaled by a factor of λ

Trace of a Matrix

Trace of a *square* matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the sum of its diagonal elements

$$\text{tr} \mathbf{A} = \sum_{i=1}^n a_{ii}$$

Trace of a Matrix

Trace of a *square* matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the sum of its diagonal elements

$$\text{tr} \mathbf{A} = \sum_{i=1}^n a_{ii}$$

Trace has useful properties

$$\text{tr} \mathbf{A} + \mathbf{B} = \text{tr} \mathbf{A} + \text{tr} \mathbf{B}$$

$$\text{tr} \mathbf{A} \mathbf{B} = \text{tr} \mathbf{B} \mathbf{A}$$

$$\text{tr} \mathbf{A} \mathbf{B} \dots \mathbf{Y} \mathbf{Z} = \text{tr} \mathbf{Z} \mathbf{A} \mathbf{B} \dots \mathbf{Y}$$

$$\text{tr} \mathbf{A}^T \mathbf{B} = \text{tr} \mathbf{A} \mathbf{B}^T = \text{tr} \mathbf{B} \mathbf{A}^T = \text{tr} \mathbf{B}^T \mathbf{A}$$

Determinant

$$\det A \quad |A|$$

$$\det A : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}$$

Determinant

$$\det A \quad |A|$$

$$\det A : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}$$

- If determinant of a matrix is 0 then the matrix is singular (it cannot be inversed), if it is not 0 the matrix can be inversed
- Consider the set of points formed by taking all possible linear combinations of row vectors of a matrix, where the coefficients of those combinations are between 0 and 1. Absolute value of the determinant is a measure of the volume of this set.

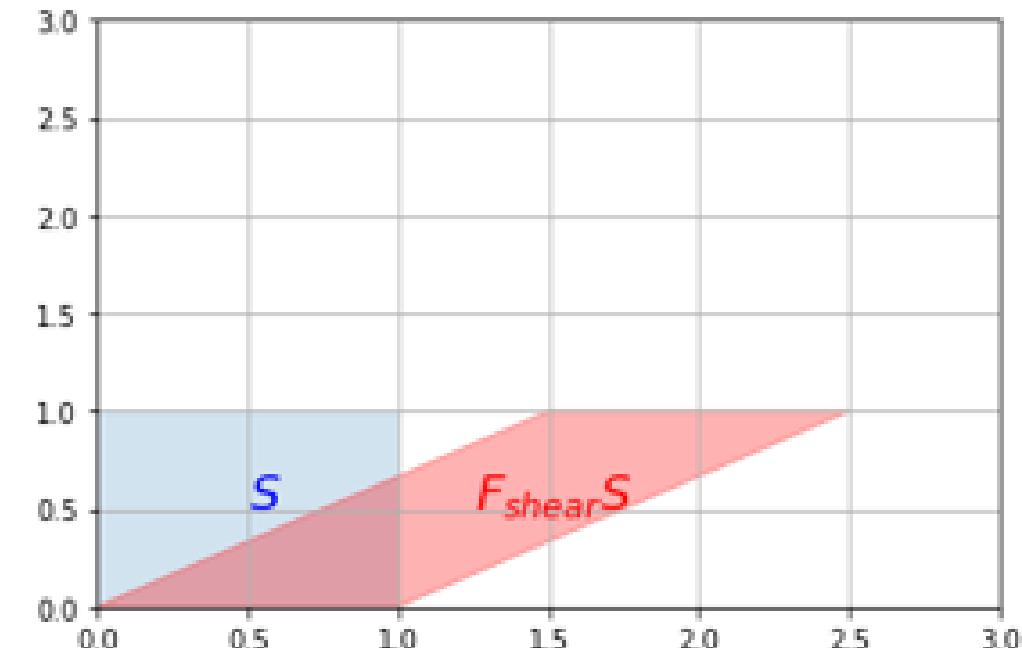
Example - Shear Mapping

$$\mathbf{F}_{shear} = \begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix}$$

Example:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$\mathbf{F}_{shear} \mathbf{S} = \begin{bmatrix} 0 & 1.5 & 2.5 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

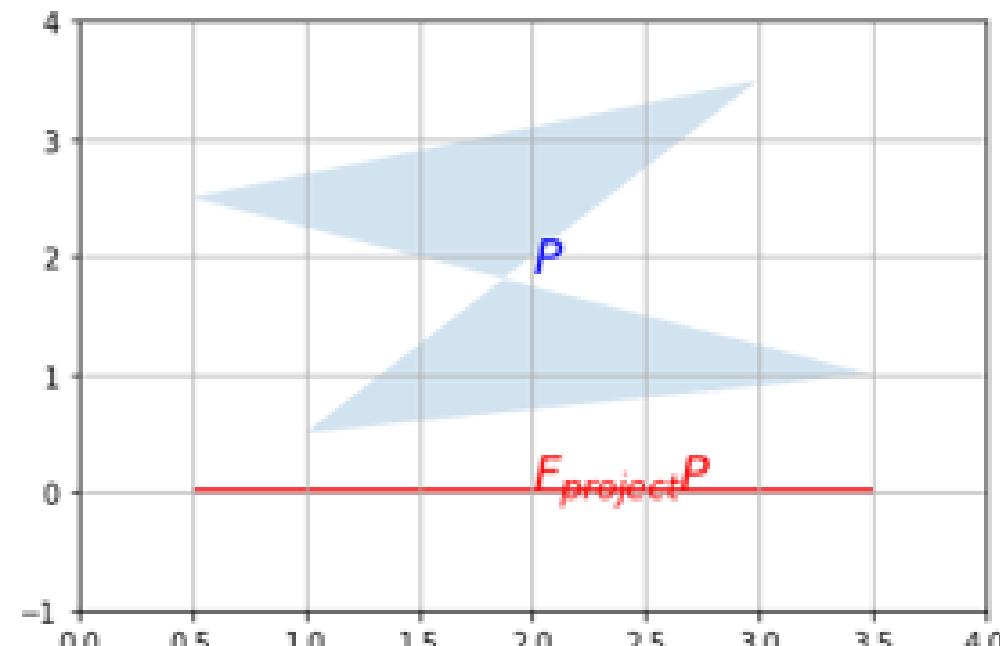


Inverse Example - Project onto a vector

$$P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0.5 & 3.5 & 2.5 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

$$F_{project} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$F_{project}P = \begin{bmatrix} 1 & 3 & 0.5 & 3.5 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



Determinant - Properties

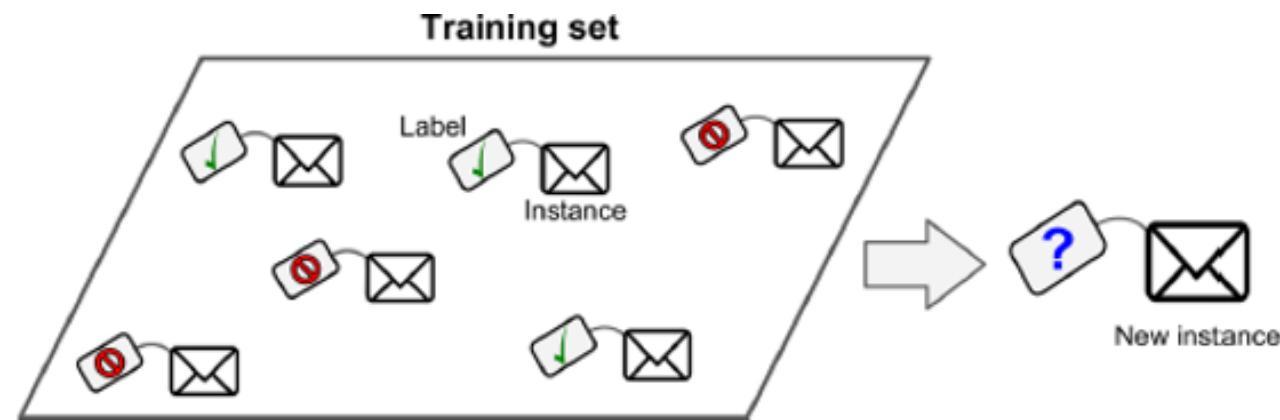
- Determinant of I is 1
- If we multiply a single row of A by a scalar λ , the determinant of the resulting matrix is $\lambda|A|$
- If we exchange any two rows of A , the determinant of the resulting matrix is $-|A|$
- $|A| = |A^T|$
- For $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$, $|AB| = |A||B|$
- If determinant is 0 then the matrix is singular (it cannot be inversed), if it is not 0 the matrix can be inversed

COGS514 - Cognition and Machine Learning

Classification

Supervised Learning

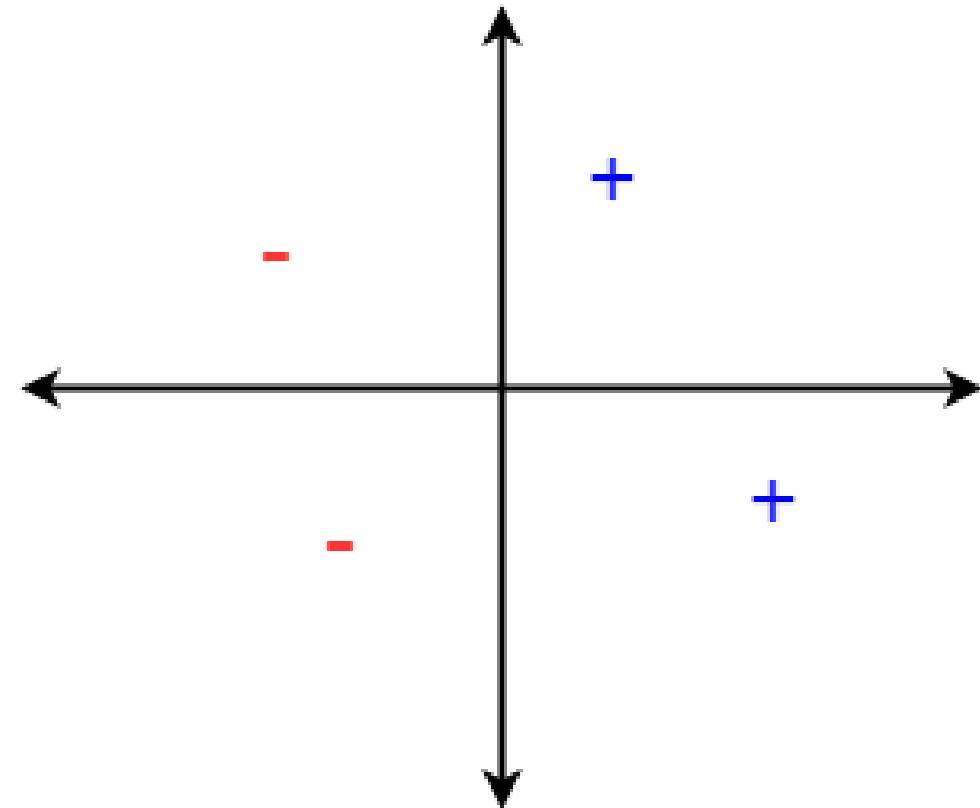
- Learn from labelled data
- Learn a function that map inputs X to outputs Y



Supervised Learning - Classification

- Predicted values (Y) are discrete

item	power	portability	like
	3	1	+
	-2	2	+
	-2.1	-1	-
	2.5	-2	-



Binary Classification

- Suppose we work on a classification problem with labels:

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

and features

$$x \in \mathbb{R}^d$$

Binary Classification

- Suppose we work on a classification problem with labels:

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

and features

$$x \in \mathbb{R}^d$$

In supervised learning we will have a training data of form

$$\mathcal{D} \in \left\{ (x^{(0)}, y^{(0)}), (x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \right\}$$

Example

item	power	portability	like	
	3	1	+	
	-2	2	+	
	-2.1	-1	-	 
	2.5	-2	-	

$$x^{(2)} = \begin{bmatrix} -2.1 \\ -1 \end{bmatrix}$$
$$y^{(2)} = -1$$

Binary Classification

- Labels $y \in \{-1, +1\}$, features $x \in \mathbb{R}^d$
- A binary classifier h is a mapping $\mathbb{R}^d \rightarrow \{-1, +1\}$

$$x \rightarrow \boxed{h} \rightarrow y$$

Binary Classification Training Error

Binary classifier

$$x \rightarrow \boxed{h} \rightarrow y$$

Predicted class

$$h(x^{(i)})$$

Example

item	power	portability	like	
	3	1	+	
	-2	2	+	
	-2.1	-1	-	
	2.5	-2	-	

$$x^{(2)} = \begin{bmatrix} -2.1 \\ -1 \end{bmatrix}$$
$$y^{(2)} = -1$$

Suppose h is a linear classifier with $w = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$

$$h(x^{(2)}) = \begin{cases} +1 & \text{if } w \cdot x^{(2)} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Example

item	power	portability	like	
	3	1	+	
	-2	2	+	
	-2.1	-1	-	
	2.5	-2	-	

$$x^{(2)} = \begin{bmatrix} -2.1 \\ -1 \end{bmatrix}$$

$$y^{(2)} = -1$$

Suppose h is a linear classifier with $w =$
 $\begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$

$$h(x^{(2)}) = \begin{cases} +1 & \text{if } w \cdot x^{(2)} > 0 \\ -1 & \text{otherwise} \end{cases}$$
$$= +1$$

Binary Classification Training Error

Binary classifier

$$x \rightarrow \boxed{h} \rightarrow y$$

Loss function

$$\ell(h(\mathbf{x}^{(i)}), y^{(i)}) = \begin{cases} 1 & h(\mathbf{x}^{(i)}) \neq y^{(i)} \\ 0 & \text{otherwise} \end{cases}$$

Example

item	power	portability	like	
	3	1	+	
	-2	2	+	
	-2.1	-1	-	 
	2.5	-2	-	

$$\mathbf{x}^{(2)} = \begin{bmatrix} -2.1 \\ -1 \end{bmatrix}, \quad y^{(2)} = -1$$

Suppose h is a linear classifier with $w = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$

$$h(\mathbf{x}^{(2)}) = \begin{cases} +1 & \text{if } w \cdot \mathbf{x}^{(2)} > 0 \\ -1 & \text{otherwise} \end{cases} = +1$$

$$\ell(h(\mathbf{x}^{(2)}), y^{(2)}) = 1$$

Binary Classification Training Error

Binary classifier

$$x \rightarrow \boxed{h} \rightarrow y$$

Loss function

$$\ell(h(\mathbf{x}^{(i)}), y^{(i)}) = \begin{cases} 1 & h(\mathbf{x}^{(i)}) \neq y^{(i)} \\ 0 & \text{otherwise} \end{cases}$$

Empirical Risk

$$\mathbb{E}_{\mathcal{D}}[\ell(h(\mathbf{x}), y)] = \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}), y^{(i)})$$

Example

item	power	portability	like	
	3	1	+	
	-2	2	+	
	-2.1	-1	—	 
	2.5	-2	—	

Exercise

Compute the empirical risk of linear classifier h with $w = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$ on this \mathcal{D}

Hypothesis Class

- Hypothesis class \mathcal{H} is the set of possible classifiers that represents a mapping $\mathbb{R}^d \rightarrow \{-1, 1\}$

Learning Algorithm

Learning algorithm takes a dataset \mathcal{D} and returns $h \in \mathcal{H}$

$$\mathcal{D} \rightarrow \boxed{\text{learning alg}(\mathcal{H})} \rightarrow h$$

Linear classifier

Linear classifiers is a hypothesis class that represents a linear separator for the data by using coefficients (parameters) $w \in \mathbb{R}^d$

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Linear classifier

Linear classifiers is a hypothesis class that represents a linear separator for the data by using coefficients (parameters) $\mathbf{w} \in \mathbb{R}^d$

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

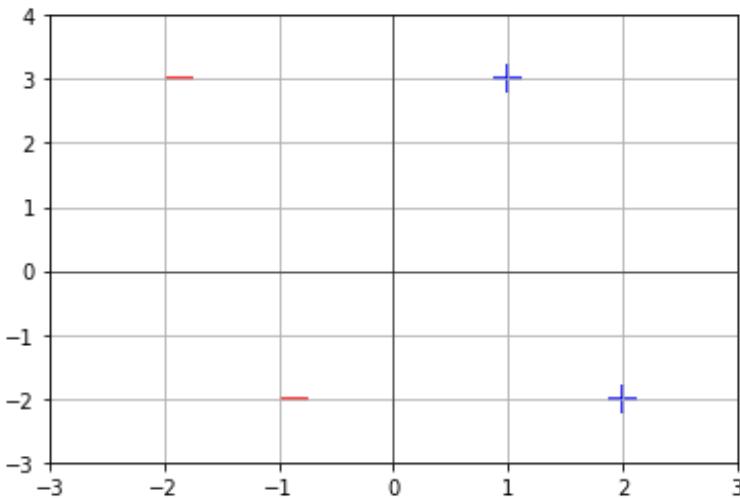
- i.e. a hyperplane specified by coefficients $\mathbf{w} \in \mathbb{R}^d$

Note that: $\mathbf{w}^T \mathbf{x} = \mathbf{w} \cdot \mathbf{x} = \sum_{j=1}^n w_j x_j$

Example - Binary Classification

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} & \begin{bmatrix} 2 \\ -2 \end{bmatrix} & \begin{bmatrix} -2 \\ 3 \end{bmatrix} & \begin{bmatrix} -1 \\ -2 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & [-1] \end{bmatrix}$$

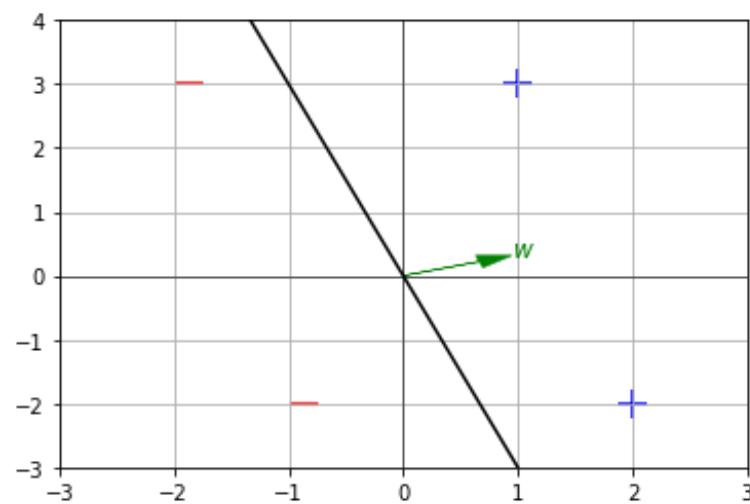


Example - Linear Classifier

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} & \begin{bmatrix} 2 \\ -2 \end{bmatrix} & \begin{bmatrix} -2 \\ 3 \end{bmatrix} & \begin{bmatrix} -1 \\ -2 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & [-1] \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$



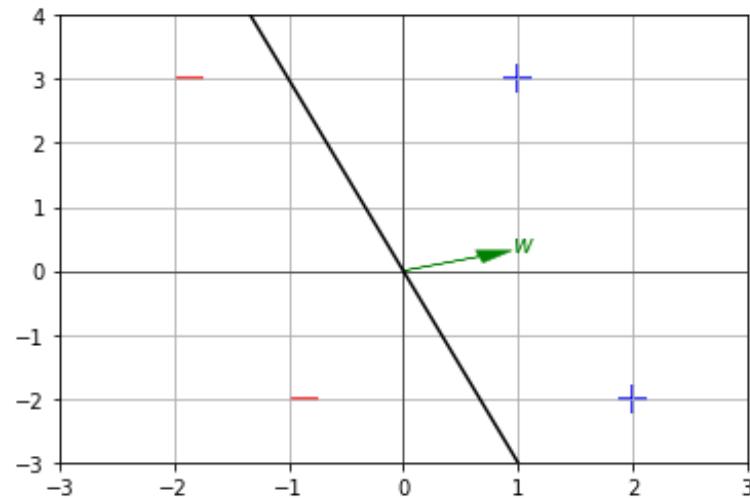
Example - Linear Classifier

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & [-1] \\ [3] & [-2] & [3] & [-2] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$\mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } 3x_1 + 1x_2 > 0 \\ -1 & \text{otherwise} \end{cases}$$



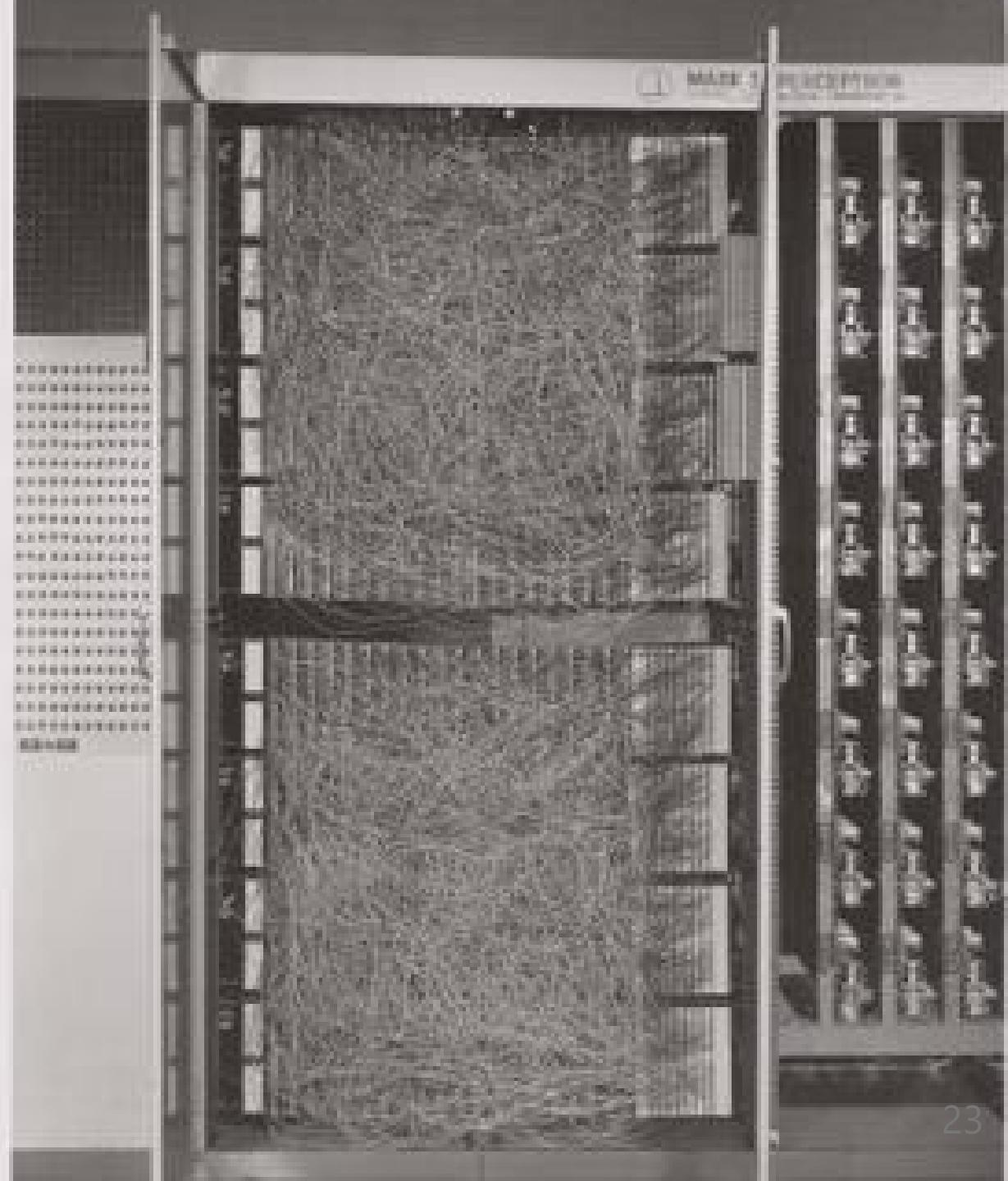
How to learn a linear classifier?

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo
of Computer Designed to
Read and Grow Wiser

WASHINGTON, July 7 (UPI)
—The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

Image¹ Mark 1 Perceptron Machine (New York Times (1958))



Perceptron Algorithm

- Start $\mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$

Perceptron Algorithm

- Start $\mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$
- For $t = 0, 1, 2, \dots$:
 - Select a random index $i \in \{1, \dots, n\}$

Perceptron Algorithm

- Start $\mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$
- For $t = 0, 1, 2, \dots$:
 - Select a random index $i \in \{1, \dots, n\}$
 - If: $y(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

Perceptron Algorithm

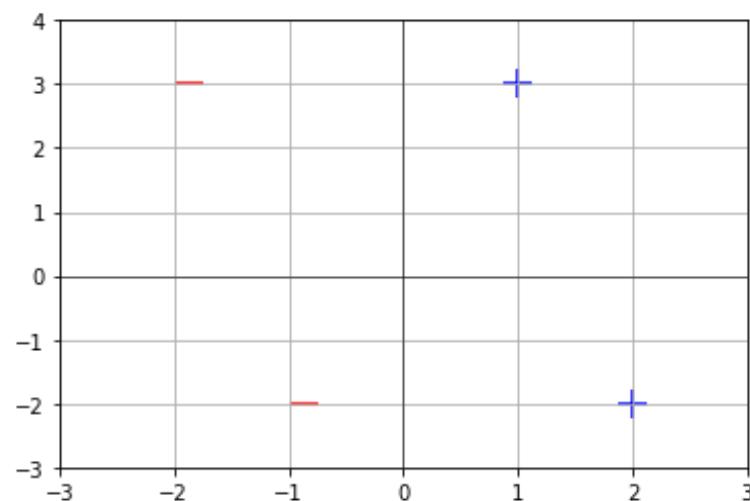
- Start $\mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$
- For $t = 0, 1, 2, \dots$:
 - Select a random index $i \in \{1, \dots, n\}$
 - If: $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$
$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$
 - Else:
$$\mathbf{w} = \mathbf{w}$$

Example - Perceptron

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} & \begin{bmatrix} 2 \\ -2 \end{bmatrix} & \begin{bmatrix} -2 \\ 3 \end{bmatrix} & \begin{bmatrix} -1 \\ -2 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



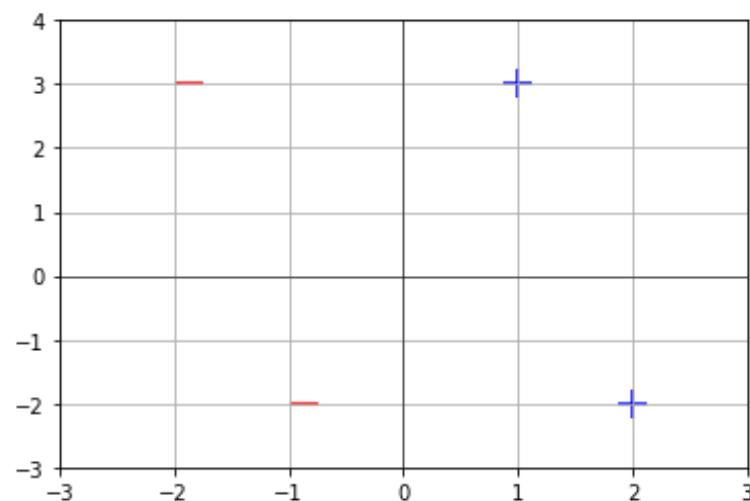
Example - Perceptron

Select a random index

$$\mathbf{X} = \begin{bmatrix} \overbrace{\begin{bmatrix} 1 \\ 3 \end{bmatrix}} \\ \begin{bmatrix} 2 \\ -2 \end{bmatrix} \quad \begin{bmatrix} -2 \\ 3 \end{bmatrix} \quad \begin{bmatrix} -1 \\ -2 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} \underbrace{\begin{bmatrix} +1 \end{bmatrix}} \\ \begin{bmatrix} +1 \end{bmatrix} \quad \begin{bmatrix} -1 \end{bmatrix} \quad \begin{bmatrix} -1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

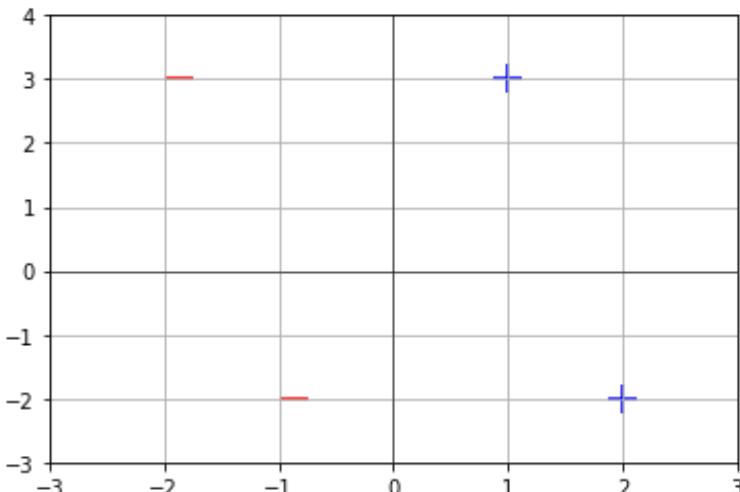


Example - Perceptron

$$\mathbf{x} = \begin{bmatrix} \overbrace{[1]} \\ [3] \end{bmatrix} \quad \begin{bmatrix} 2 \\ -2 \end{bmatrix} \quad \begin{bmatrix} -2 \\ 3 \end{bmatrix} \quad \begin{bmatrix} -1 \\ -2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} \overbrace{[+1]} \\ [+1] \end{bmatrix} \quad \begin{bmatrix} [-1] \\ [-1] \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$



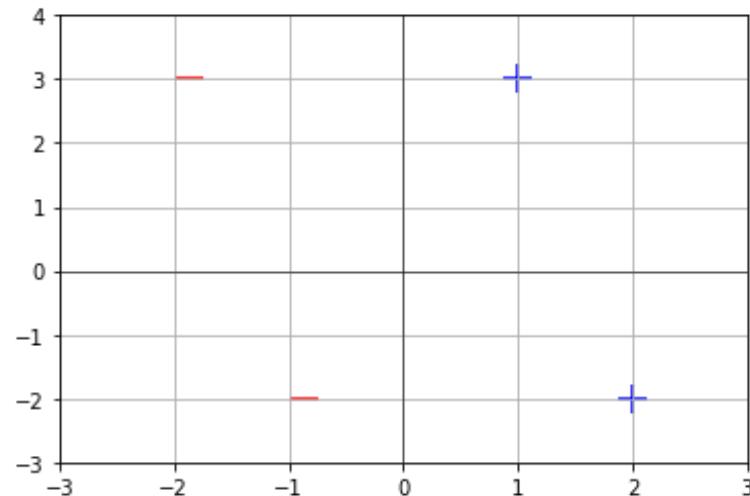
Example - Perceptron

$$\mathbf{X} = \left[\begin{matrix} \widehat{[1]} \\ [2] \\ [-2] \\ [3] \end{matrix} \quad \begin{matrix} [2] \\ [-2] \\ [3] \\ [-2] \end{matrix} \quad \begin{matrix} [-2] \\ [3] \\ [-1] \\ [-2] \end{matrix} \quad \begin{matrix} [-1] \\ [-2] \\ [-1] \\ [-1] \end{matrix} \right] \quad \mathbf{Y} = \left[\begin{matrix} \widehat{[+1]} \\ [+1] \\ [-1] \\ [-1] \end{matrix} \right] \quad \mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

$$\begin{aligned} y^{(0)}(\mathbf{w}^T \mathbf{x}^{(0)}) &= 1 \left([0 \quad 0] \begin{bmatrix} 1 \\ 3 \end{bmatrix} \right) \\ &= 1(0 \times 1 + 0 \times 3) \\ &= 0 \end{aligned}$$



Example - Perceptron

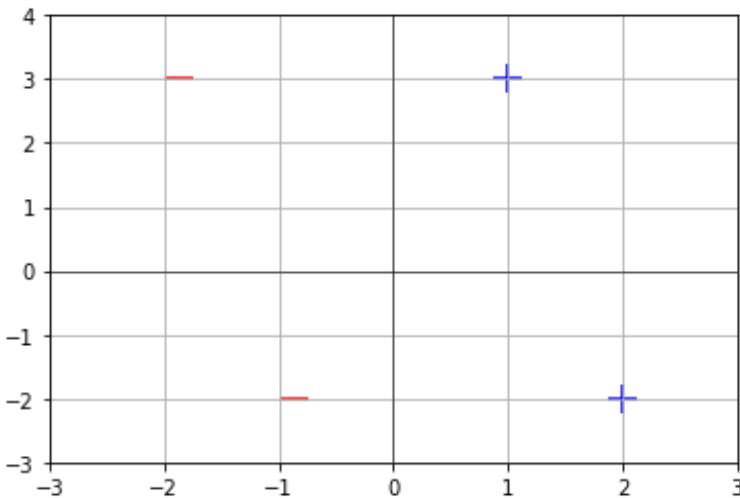
$$\mathbf{x} = \begin{bmatrix} \overbrace{[1]} \\ [2] \\ [-2] \\ [3] \\ [3] \\ [-2] \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} \overbrace{[+1]} \\ [+1] \\ [-1] \\ [-1] \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

$y^{(0)}(\mathbf{w}^T \mathbf{x}^{(0)}) = 0$:

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$



Example - Perceptron

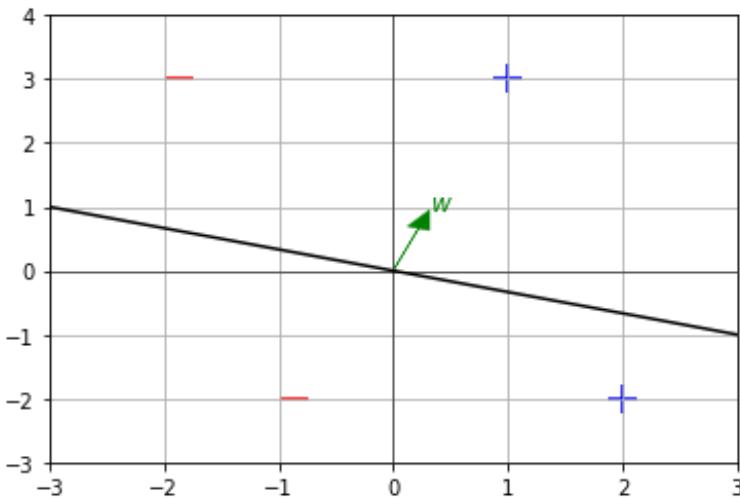
$$\mathbf{x} = \begin{bmatrix} \overbrace{[1]} \\ [2] \\ [-2] \\ [3] \\ [3] \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} \overbrace{[+1]} \\ [+1] \\ [-1] \\ [-1] \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

$y^{(0)}(\mathbf{w}^T \mathbf{x}^{(0)}) = 0$:

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$



Example - Perceptron

Select a random index

$$\mathbf{X} = \begin{bmatrix} [1] & \overbrace{[2]} & [-2] & [-1] \\ [3] & [-2] & 3 & -2 \end{bmatrix}$$

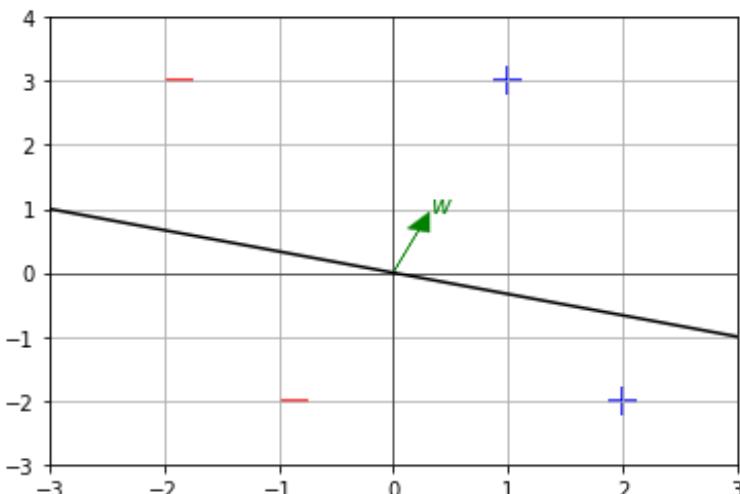
$$\mathbf{Y} = \begin{bmatrix} [+1] & \overbrace{[+1]} & [-1] & [-1] \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

Example - Perceptron

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} & \overbrace{\begin{bmatrix} 2 \\ -2 \end{bmatrix}} & \begin{bmatrix} -2 \\ 3 \end{bmatrix} & \begin{bmatrix} -1 \\ -2 \end{bmatrix} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} [+1] & \overbrace{\begin{bmatrix} +1 \\ -1 \end{bmatrix}} & \begin{bmatrix} -1 \\ -1 \end{bmatrix} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$



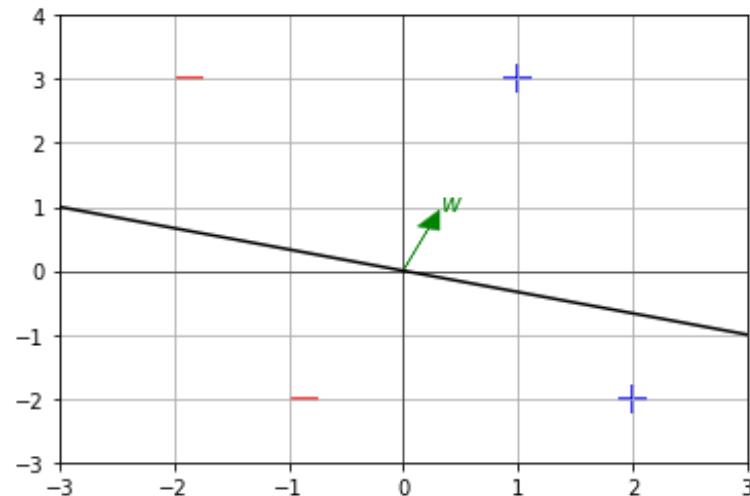
Example - Perceptron

$$\mathbf{x} = \begin{bmatrix} 1 & \overbrace{2} \\ 3 & -2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} +1 & \overbrace{+1} \\ -1 & -1 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

$$\begin{aligned} y^{(1)}(\mathbf{w}^T \mathbf{x}^{(1)}) &= 1 \left([1 \quad 3] \begin{bmatrix} 2 \\ -2 \end{bmatrix} \right) \\ &= -4 \end{aligned}$$



Example - Perceptron

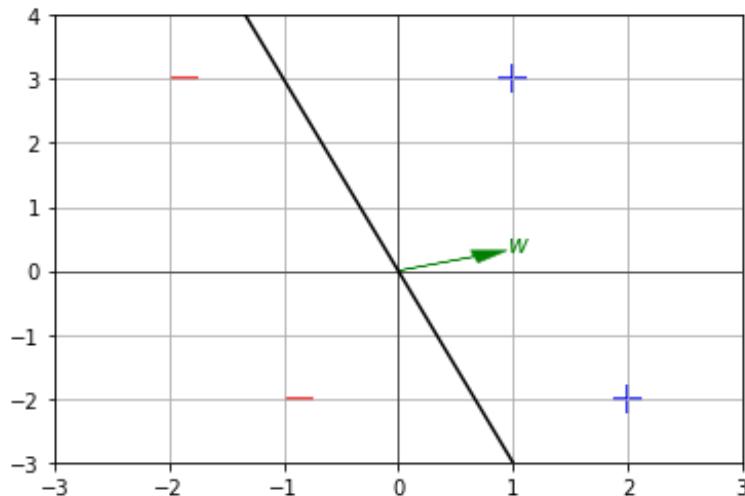
$$\mathbf{X} = \begin{bmatrix} [1] & \overbrace{[2]} \\ [3] & [-2] \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} [+1] & \overbrace{[+1]} \\ [-1] & [-1] \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

$$y^{(1)}(\mathbf{w}^T \mathbf{x}^{(1)}) = -4$$

$$\mathbf{w} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 1 \begin{bmatrix} 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$



Example - Perceptron

Select a random index

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & \overbrace{[-1]} \\ [3] & [-2] & [3] & [-2] \end{bmatrix}$$

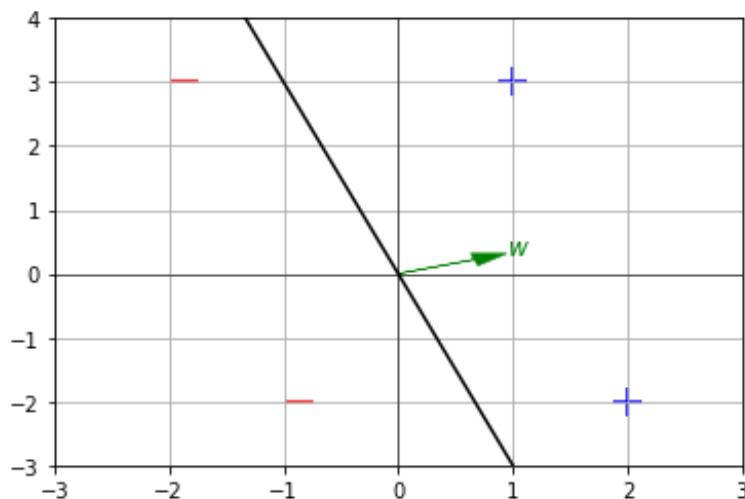
$$\mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & \overbrace{[-1]} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

Example - Perceptron

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} & \begin{bmatrix} 2 \\ -2 \end{bmatrix} & \begin{bmatrix} -2 \\ 3 \end{bmatrix} & \overbrace{\begin{bmatrix} -1 \\ -2 \end{bmatrix}} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & \overbrace{[-1]} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$



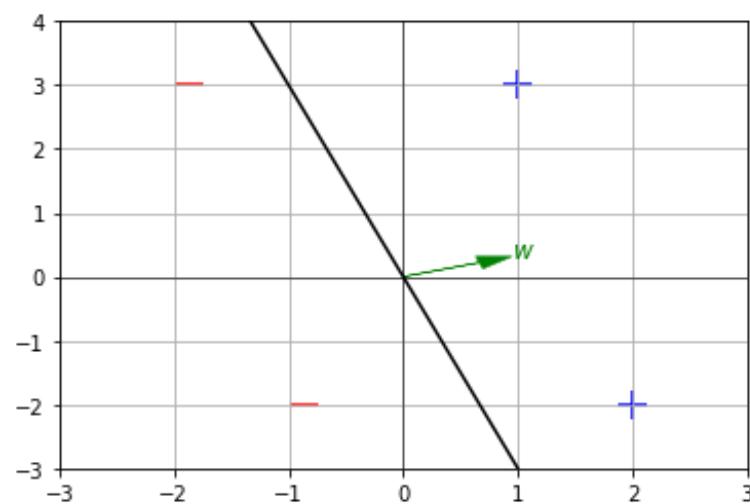
Example - Perceptron

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & \widetilde{[-1]} \\ [3] & [-2] & [3] & \widetilde{[-2]} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & \widetilde{[-1]} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

$$y^{(3)}(\mathbf{w}^T \mathbf{x}^{(3)}) = -1 \left([3 \quad 1] \begin{bmatrix} -1 \\ -2 \end{bmatrix} \right) = 5$$



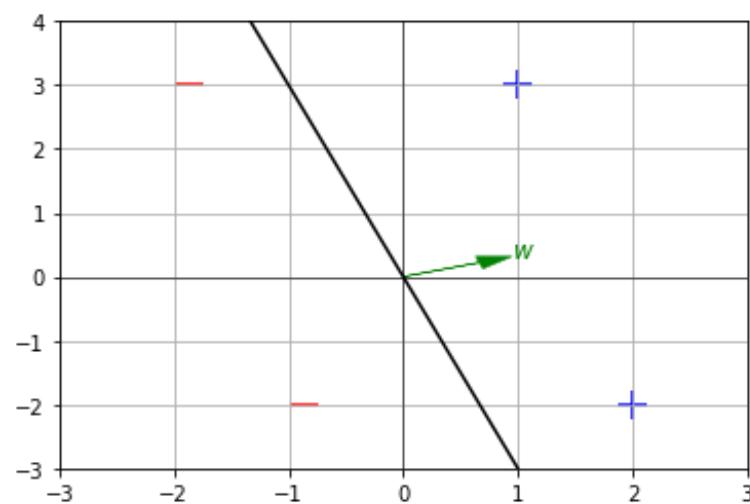
Example - Perceptron

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & \widetilde{[-1]} \\ [3] & [-2] & [3] & \widetilde{[-2]} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & \widetilde{[-1]} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

$$y^{(3)}(\mathbf{w}^T \mathbf{x}^{(3)}) = -1 \left([3 \quad 1] \begin{bmatrix} -1 \\ -2 \end{bmatrix} \right) = 5 \not\leq 0$$



Example - Perceptron

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & \overbrace{[-1]} \\ [3] & [-2] & [3] & [-2] \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & \overbrace{[-1]} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

If $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$:

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

Else :

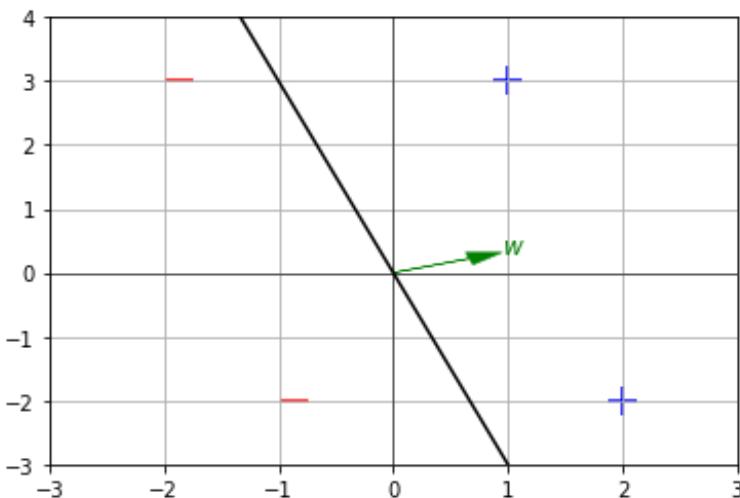
$$\mathbf{w} = \mathbf{w}$$

We don't update as $y^{(3)}(\mathbf{w}^T \mathbf{x}^{(3)}) = 5 \not\leq 0$

Offset Term

Note that the linear classifiers in the example always passed through the origin

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$



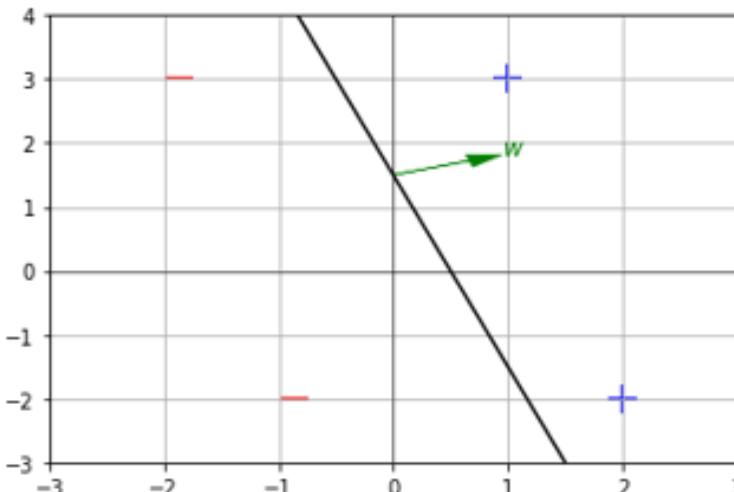
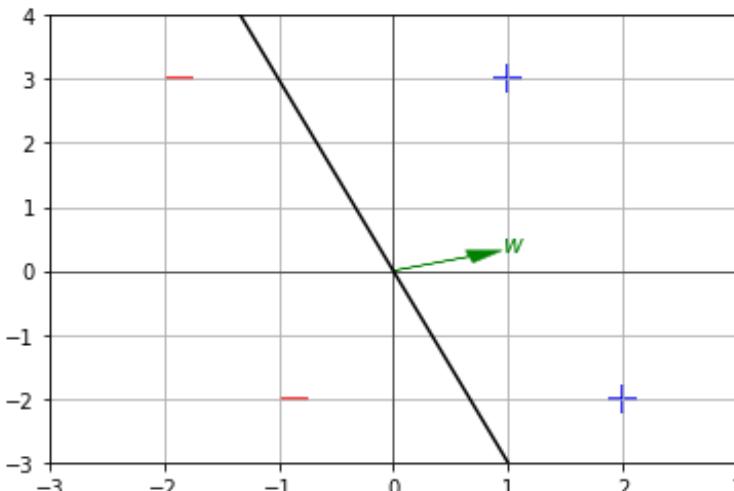
Offset Term

Note that the linear classifiers in the example always passed through the origin

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

We need an **offset term** w_0 for classifiers that do not pass through the origin

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$



Adding Offset Term

We can convert any linear classifier with offset into one with no offset by **adding one more dimension**

Adding Offset Term

We can convert any linear classifier with offset into one with no offset by **adding one more dimension**

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \in \mathbb{R}^d \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d \quad w_0 \in \mathbb{R}$$

Adding Offset Term

We can convert any linear classifier with offset into one with no offset by **adding one more dimension**

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \in \mathbb{R}^d \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d \quad w_0 \in \mathbb{R}$$

$$h(\mathbf{x}, \mathbf{w}_{new}) = \begin{cases} +1 & \text{if } \mathbf{w}_{new}^T \mathbf{x}_{new} > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\mathbf{w}_{new} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ w_0 \end{bmatrix} \in \mathbb{R}^{d+1} \quad \mathbf{x}_{new} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{bmatrix} \in \mathbb{R}^{d+1}$$

Adding Offset Term

We can convert any linear classifier with offset into one with no offset by **adding one more dimension**

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \in \mathbb{R}^d \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d \quad w_0 \in \mathbb{R}$$

$$h(\mathbf{x}, \mathbf{w}_{new}) = \begin{cases} +1 & \text{if } \mathbf{w}_{new}^T \mathbf{x}_{new} > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\mathbf{w}_{new} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ w_0 \end{bmatrix} \quad \mathbf{x}_{new} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{bmatrix}$$

$$\mathbf{w}_{new}^T \mathbf{x}_{new} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_0$$

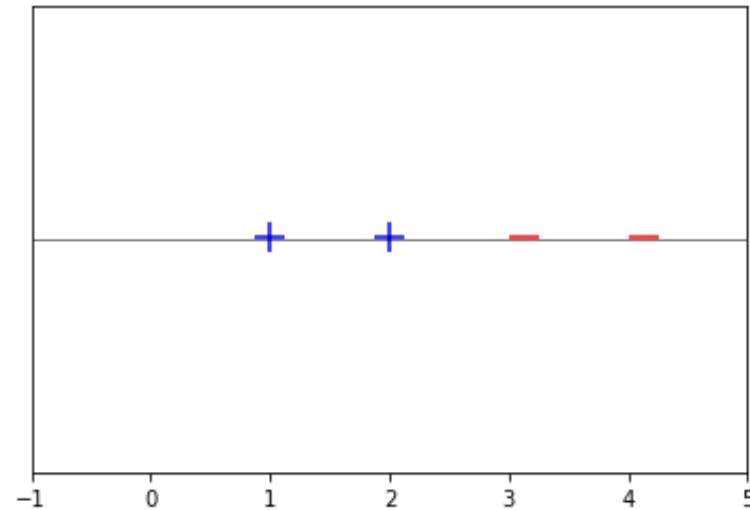
Example - 1D Linear Separator

$$\mathbf{X} = [[1] \quad [2] \quad [3] \quad [4]]$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$w = [w_1]$$

$$h(\mathbf{x}, w) = \begin{cases} +1 & \text{if } w_1 \cdot x > 0 \\ -1 & \text{otherwise} \end{cases}$$



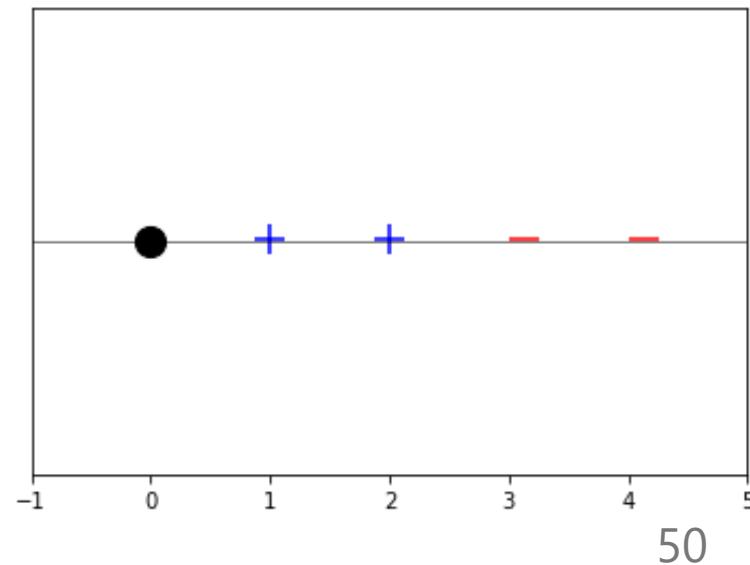
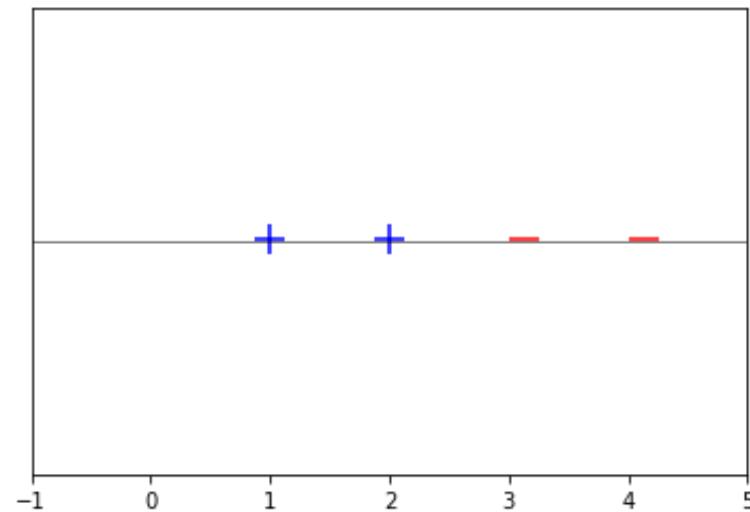
Example - 1D Linear Separator

$$\mathbf{X} = [[1] \quad [2] \quad [3] \quad [4]]$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$w = [w_1]$$

$$h(\mathbf{x}, w) = \begin{cases} +1 & \text{if } w_1 \cdot x > 0 \\ -1 & \text{otherwise} \end{cases}$$

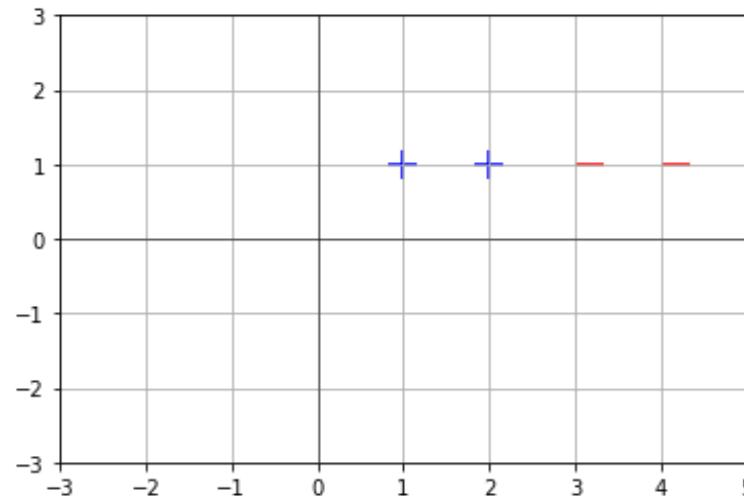
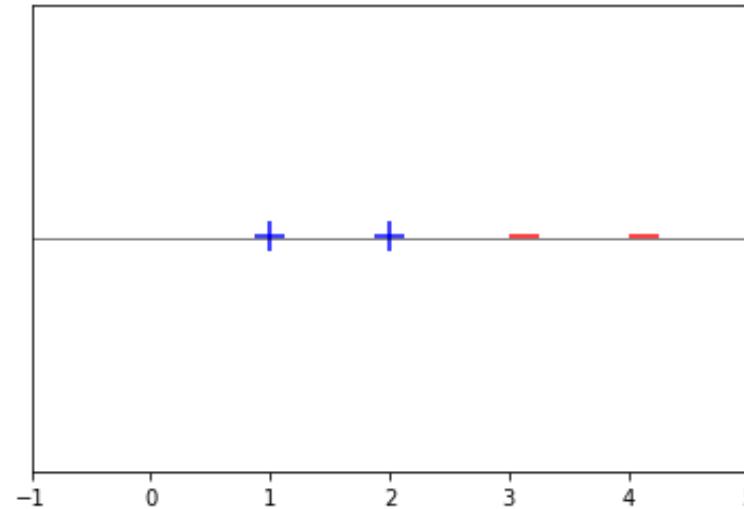


Example - Add one more dimension for offset

$$\mathbf{X}_{new} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 2 \\ 1 \end{bmatrix} & \begin{bmatrix} 3 \\ 1 \end{bmatrix} & \begin{bmatrix} 4 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$w_{new} = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix}$$

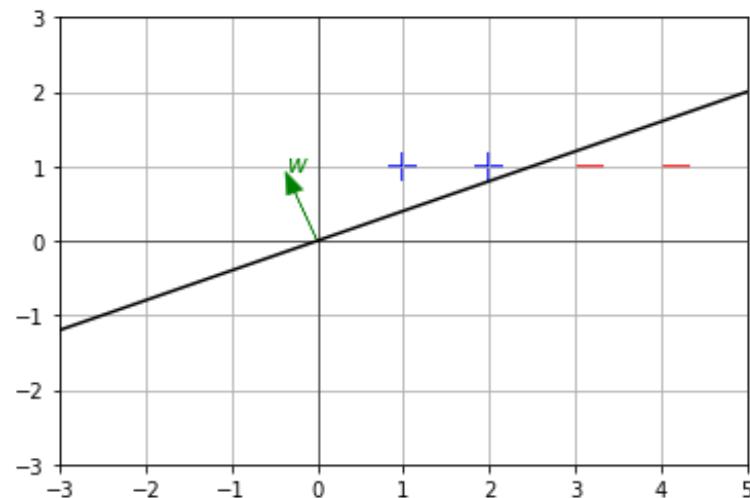


Example - There is a linear separator passing through the origin in 2D

$$\mathbf{X}_{new} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 2 \\ 1 \end{bmatrix} & \begin{bmatrix} 3 \\ 1 \end{bmatrix} & \begin{bmatrix} 4 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & [-1] \end{bmatrix}$$

$$w_{new} = \begin{bmatrix} -2 \\ 5 \end{bmatrix}$$



Linear Separability

A training set is linearly separable if there exists w such that for all $i = 1, 2, \dots, n$

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) > 0$$

Linear Separability

- A training set is linearly separable if there exists w such that for all $i = 1, 2, \dots, n$

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) > 0$$

- or that all predictions on the dataset are correct

$$h(\mathbf{x}^{(i)}; \mathbf{w}) = y^{(i)}$$

- or that the training error (empirical risk) is zero

$$R_{\mathcal{D}}[h] = 0$$

Convergence theorem

If training dataset \mathcal{D} is linearly separable, the perceptron algorithm is guaranteed to find a linear separator 

Distance

Distance of a point x to hyperplane w is

$$\text{dist}(\mathbf{x}, \mathbf{w}) = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|}$$

Distance - Example

$$\text{dist}(\mathbf{x}, \mathbf{w}) = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|}$$

$$\mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \quad \mathbf{x}^{(2)} = \begin{bmatrix} 2 \\ -2 \end{bmatrix} \quad y^{(2)} = +1$$

Distance - Example

$$\text{dist}(\mathbf{x}, \mathbf{w}) = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|}$$

$$\mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \quad \mathbf{x}^{(2)} = \begin{bmatrix} 2 \\ -2 \end{bmatrix} \quad y^{(2)} = +1$$

$$\mathbf{w}^T \mathbf{x} = 4$$

$$\|\mathbf{w}\|_2 = \sqrt{\mathbf{w}^T \mathbf{w}} = \sqrt{3^2 + 1^1} = 3.16$$

Distance - Example

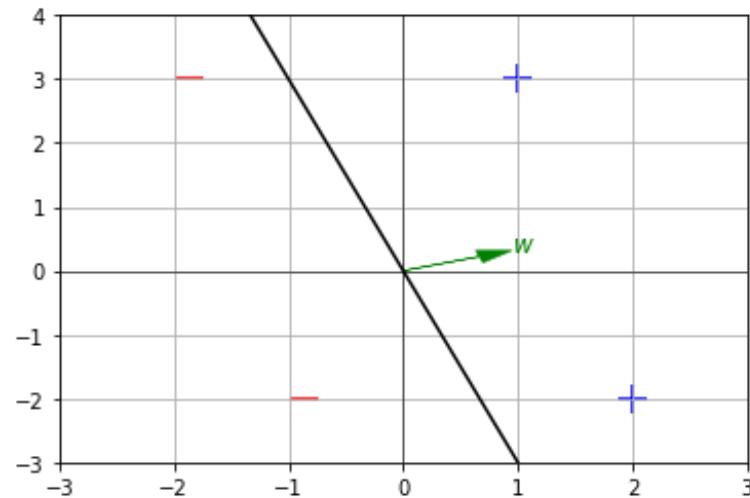
$$\text{dist}(\mathbf{x}, \mathbf{w}) = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|}$$

$$\mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \quad \mathbf{x}^{(2)} = \begin{bmatrix} 2 \\ -2 \end{bmatrix} \quad y^{(2)} = +1$$

$$\mathbf{w}^T \mathbf{x} = 4$$

$$\|\mathbf{w}\|_2 = \sqrt{\mathbf{w}^T \mathbf{w}} = \sqrt{3^2 + 1^1} = 3.16$$

$$\text{dist}(\mathbf{x}, \mathbf{w}) = 4/3.16 = 1.26$$



Margin

Margin of a point (x, y) with respect to hyperplane w is

$$y \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = y \cdot \text{dist}(\mathbf{x}, \mathbf{w})$$

Margin

Margin of a point (x, y) with respect to hyperplane w is

$$y \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = y \cdot \text{dist}(\mathbf{x}, \mathbf{w})$$

Note that margin is positive if the sign of y is classified correctly by $\mathbf{w}^T \mathbf{x}$

Margin

Margin of a point (x, y) with respect to hyperplane w is

$$y \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = y \cdot \text{dist}(\mathbf{x}, \mathbf{w})$$

Note that margin is positive if the sign of y is classified correctly by $\mathbf{w}^T \mathbf{x}$

- e.g. margin of

$$(\mathbf{x}^{(2)}, y^{(2)}) \quad \mathbf{x}^{(2)} = \begin{bmatrix} 2 \\ -2 \end{bmatrix} \quad y^{(2)} = +1$$

$$1 \cdot \text{dist}(\mathbf{x}, \mathbf{w}) = 1 \times 1.26 = 1.26$$

Margin of a Dataset

Margin γ of a dataset with respect to hyperplane \mathbf{w} is the minimum margin of any point with respect to \mathbf{w}

$$\gamma = \min_i \left(y^{(i)} \frac{\mathbf{w}^T \mathbf{x}^{(i)}}{\|\mathbf{w}\|} \right)$$

Margin of a Dataset

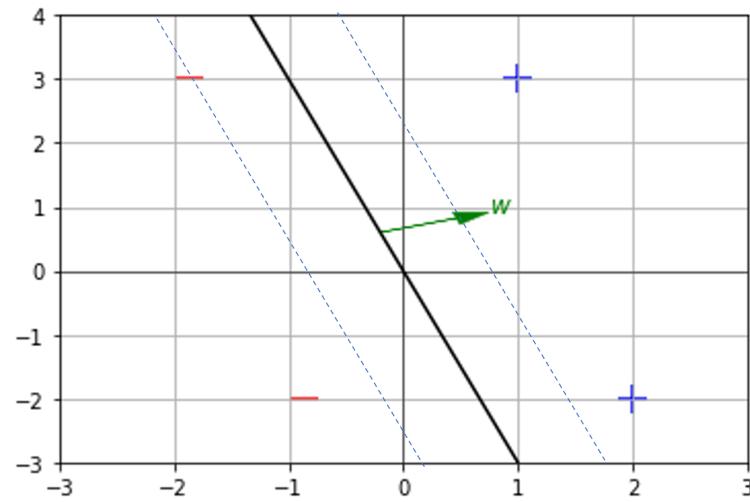
Margin γ of a dataset with respect to hyperplane w is the minimum margin of any point with respect to w

$$\gamma = \min_i \left(y^{(i)} \frac{\mathbf{w}^T \mathbf{x}^{(i)}}{\|\mathbf{w}\|} \right)$$

Margin is positive if and only if all points are classified correctly. In that case, it represents the distance from the hyperplane to the closest point.

Margin Example

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & [-1] \\ [3] & [-2] & [3] & [-2] \end{bmatrix} \quad \mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]] \quad \mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$



Margin Example

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & [-1] \\ [3] & [-2] & [3] & [-2] \end{bmatrix} \quad \mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]] \quad \mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

All points are classified correctly. Their margins are:

$$(\mathbf{x}^{(1)}, y^{(1)}) = 1.90$$

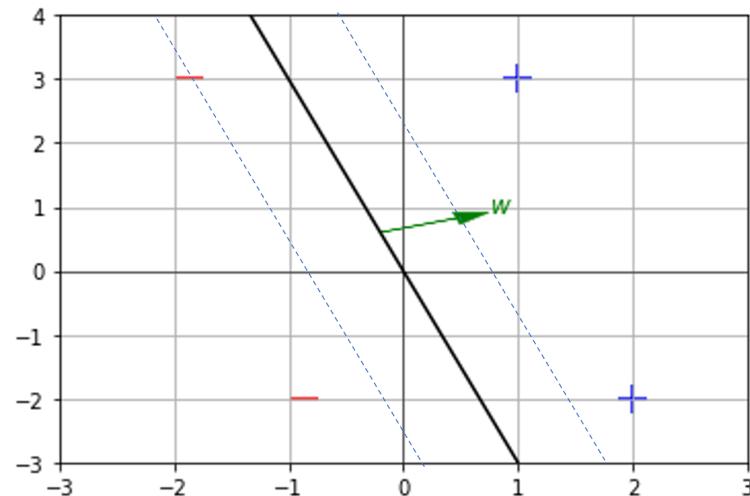
$$(\mathbf{x}^{(2)}, y^{(2)}) = 1.26$$

$$(\mathbf{x}^{(3)}, y^{(3)}) = 0.95$$

$$(\mathbf{x}^{(4)}, y^{(4)}) = 1.58$$

Margin of the dataset is minimum of those

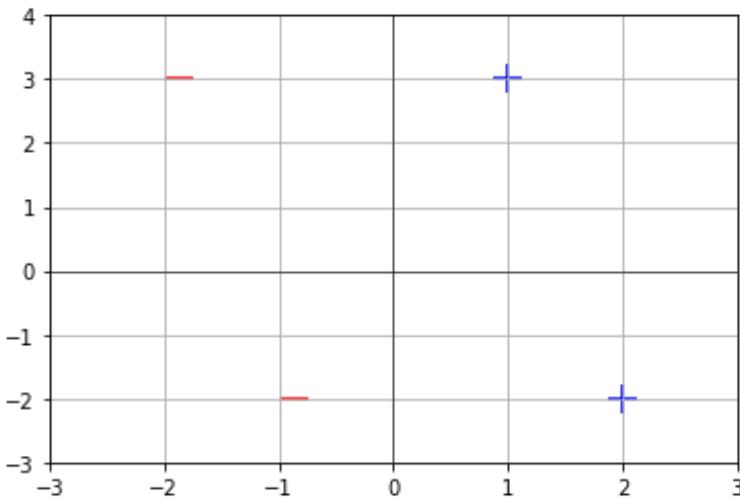
$$\gamma = 0.95$$



Diameter of the dataset

Diameter of the dataset is the maximum norm of the data points

$$R = \max_i \|\mathbf{x}^{(i)}\|$$



Diameter of the dataset

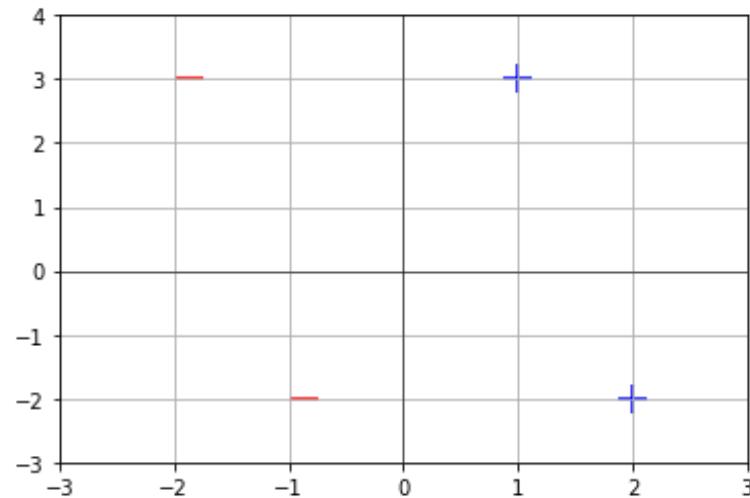
Diameter of the dataset is the maximum norm of the data points

$$R = \max_i \|\mathbf{x}^{(i)}\|$$

Example

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} & \begin{bmatrix} 2 \\ -2 \end{bmatrix} & \begin{bmatrix} -2 \\ 3 \end{bmatrix} & \begin{bmatrix} -1 \\ -2 \end{bmatrix} \end{bmatrix}$$

$$\begin{aligned} \|\mathbf{x}^{(1)}\| &= 3.16 & \|\mathbf{x}^{(2)}\| &= 2.83 \\ \|\mathbf{x}^{(3)}\| &= 3.61 & \|\mathbf{x}^{(4)}\| &= 2.24 \end{aligned}$$



Diameter of the dataset

Diameter of the dataset is the maximum norm of the data points

$$R = \max_i \|\mathbf{x}^{(i)}\|$$

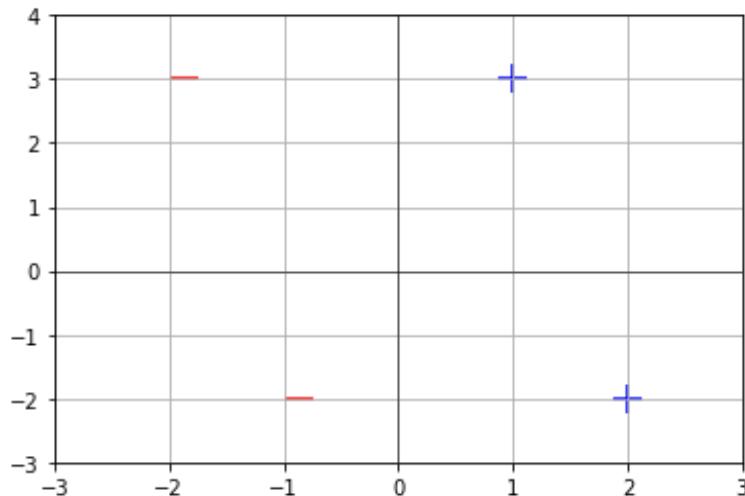
Example

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} & \begin{bmatrix} 2 \\ -2 \end{bmatrix} & \begin{bmatrix} -2 \\ 3 \end{bmatrix} & \begin{bmatrix} -1 \\ -2 \end{bmatrix} \end{bmatrix}$$

$$\|\mathbf{x}^{(1)}\| = 3.16 \quad \|\mathbf{x}^{(2)}\| = 2.83$$

$$\|\mathbf{x}^{(3)}\| = 3.61 \quad \|\mathbf{x}^{(4)}\| = 2.24$$

$$R = 3.61$$



Perceptron convergence

If the following conditions hold:

$$\overbrace{\mathbf{w}^*^T \mathbf{x}^{(i)}}^{\text{margin}}$$

- there exists \mathbf{w}^* such that $y^{(i)} \frac{\mathbf{w}^*^T \mathbf{x}^{(i)}}{\|\mathbf{w}^*\|} \geq \gamma$ for some $\gamma > 0$ and for all $i = 1, \dots, n$
- all examples have bounded magnitude $\|\mathbf{x}^{(i)}\| \leq R$ for all $i = 1, \dots, n$

then the perceptron algorithm will make at most $\left(\frac{R}{\gamma}\right)^2$ mistakes.

At this point the hypothesis h will be a linear separator of the data.

Proof of Perceptron Convergence

We initialize with $w^{(0)} = 0$ and let $w^{(k)}$ define our hyperplane after perceptron algorithm has made k mistakes

$$\mathbf{u}^T \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos(\theta)$$

Proof of Perceptron Convergence

We initialize with $w^{(0)} = 0$ and let $w^{(k)}$ define our hyperplane after perceptron algorithm has made k mistakes

$$\mathbf{u}^T \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos(\theta)$$

$$\mathbf{w}^{(k)T} \mathbf{w}^{(*)} = \|\mathbf{w}^{(k)}\| \|\mathbf{w}^{(*)}\| \cos(\theta)$$

Proof of Perceptron Convergence

We initialize with $\mathbf{w}^{(0)} = 0$ and let $\mathbf{w}^{(k)}$ define our hyperplane after perceptron algorithm has made k mistakes

$$\mathbf{u}^T \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos(\theta)$$

$$\mathbf{w}^{(k)T} \mathbf{w}^{(*)} = \|\mathbf{w}^{(k)}\| \|\mathbf{w}^{(*)}\| \cos(\theta)$$

$$\cos(\theta) = \frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(k)}\| \|\mathbf{w}^{(*)}\|} = \frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} \cdot \frac{1}{\|\mathbf{w}^{(k)}\|}$$

Proof of Perceptron Convergence

$$\frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} = \frac{(\mathbf{w}^{(k-1)} + y^{(i)} \mathbf{x}^{(i)})^T \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|}$$

Proof of Perceptron Convergence

$$\begin{aligned}\frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} &= \frac{(\mathbf{w}^{(k-1)} + y^{(i)} \mathbf{x}^{(i)})^T \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} \\ &= \frac{\mathbf{w}^{(k-1)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} + \frac{(y^{(i)} \mathbf{x}^{(i)})^T \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|}\end{aligned}$$

Proof of Perceptron Convergence

$$\begin{aligned}\frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} &= \frac{(\mathbf{w}^{(k-1)} + y^{(i)} \mathbf{x}^{(i)})^T \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} \\ &= \frac{\mathbf{w}^{(k-1)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} + \underbrace{\frac{(y^{(i)} \mathbf{x}^{(i)})^T \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|}}_{\text{margin}}\end{aligned}$$

Proof of Perceptron Convergence

$$\begin{aligned}\frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} &= \frac{(\mathbf{w}^{(k-1)} + y^{(i)} \mathbf{x}^{(i)})^T \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} \\ &= \frac{\mathbf{w}^{(k-1)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} + \frac{(y^{(i)} \mathbf{x}^{(i)})^T \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} \\ &\geq \frac{\mathbf{w}^{(k-1)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} + \gamma \\ &\geq k\gamma\end{aligned}$$

Proof of Perceptron Convergence

$$\cos(\theta) = \frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(k)}\| \|\mathbf{w}^{(*)}\|} = \underbrace{\frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|}}_{\geq k\gamma} \cdot \underbrace{\frac{1}{\|\mathbf{w}^{(k)}\|}}_{?}$$

Proof of Perceptron Convergence

Lets take square of $\|w^{(k)}\|$

$$\|w^{(k)}\|^2 = \|w^{(k-1)} + y^{(i)}x^{(i)}\|^2$$

Proof of Perceptron Convergence

Lets take square of $\|w^{(k)}\|$

$$\begin{aligned}\|w^{(k)}\|^2 &= \|w^{(k-1)} + y^{(i)}x^{(i)}\|^2 \\ &= \|w^{(k-1)}\|^2 + 2y^{(i)}w^{(k-1)^T}x^{(i)} + \|x^{(i)}\|^2\end{aligned}$$

Proof of Perceptron Convergence

Lets take square of $\|w^{(k)}\|$

$$\|w^{(k)}\|^2 = \|w^{(k-1)} + y^{(i)}x^{(i)}\|^2$$

$$= \|w^{(k-1)}\|^2 + \underbrace{2y^{(i)}w^{(k-1)^T}x^{(i)}}_{\text{Negative as algorithm made a mistake}} + \underbrace{\|x^{(i)}\|^2}_{\text{Diameter } R^2}$$

Proof of Perceptron Convergence

Lets take square of $\|w^{(k)}\|$

$$\|w^{(k)}\|^2 = \|w^{(k-1)} + y^{(i)}x^{(i)}\|^2$$

$$= \|w^{(k-1)}\|^2 + 2y^{(i)}w^{(k-1)^T}x^{(i)} + \|x^{(i)}\|^2$$

$$\leq \|w^{(k-1)}\| + R^2$$

$$\leq kR^2$$

Proof of Perceptron Convergence

$$\|w^{(k)}\|^2 \leq kR^2$$

$$\frac{1}{\|w^{(k)}\|} \geq \frac{1}{\sqrt{kR}}$$

Proof of Perceptron Convergence

$$\begin{aligned}\cos(\theta) &= \frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(k)}\| \|\mathbf{w}^{(*)}\|} \\ &= \frac{\mathbf{w}^{(k)T} \mathbf{w}^{(*)}}{\|\mathbf{w}^{(*)}\|} \cdot \frac{1}{\|\mathbf{w}^{(k)}\|} \\ &\geq (k\gamma) \frac{1}{\sqrt{kR}} \\ &\geq \sqrt{k} \frac{\gamma}{R}\end{aligned}$$

Proof of Perceptron Convergence

$$\cos(\theta) \geq \sqrt{k} \frac{\gamma}{R}$$

Value of cosine is at most 1 so

$$1 \geq \sqrt{k} \frac{\gamma}{R}$$

$$k \leq \left(\frac{R}{\gamma} \right)^2$$

Perceptron algorithm will at most make $(R/\gamma)^2$ mistakes

COGS514 - Cognition and Machine Learning

Nonlinear Classification & Feature Representation

Revision

- Classifier
- Linear classifier
 - Learning linear classifiers: perceptron

Revision - Binary Classification

- Suppose we work on a classification problem with labels:

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

and features

$$x \in \mathbb{R}^d$$

- In supervised learning we will have a training data of form

$$\mathcal{D} \in \left\{ (x^{(0)}, y^{(0)}), (x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \right\}$$

Revision - Binary Classification

- Suppose we work on a classification problem with labels:

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

and features

$$x \in \mathbb{R}^d$$

- In supervised learning we will have a training data of form

$$\mathcal{D} \in \left\{ (x^{(0)}, y^{(0)}), (x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \right\}$$

- A binary classifier h is a mapping $\mathbb{R}^d \rightarrow \{-1, +1\}$

$$x \rightarrow \boxed{h} \rightarrow y$$

Revision - Linear classifier

Linear classifiers is a hypothesis class that represents a linear separator for the data by using coefficients (parameters) $w \in \mathbb{R}^d$

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

- i.e. a hyperplane specified by coefficients $w \in \mathbb{R}^d$

Revision - Perceptron Algorithm

$$\bullet \text{ Start } \mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

- For $t = 0, 1, 2, \dots$:
 - Select a random index $i \in \{1, \dots, n\}$
 - If: $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

- Else:

$$\mathbf{w} = \mathbf{w}$$

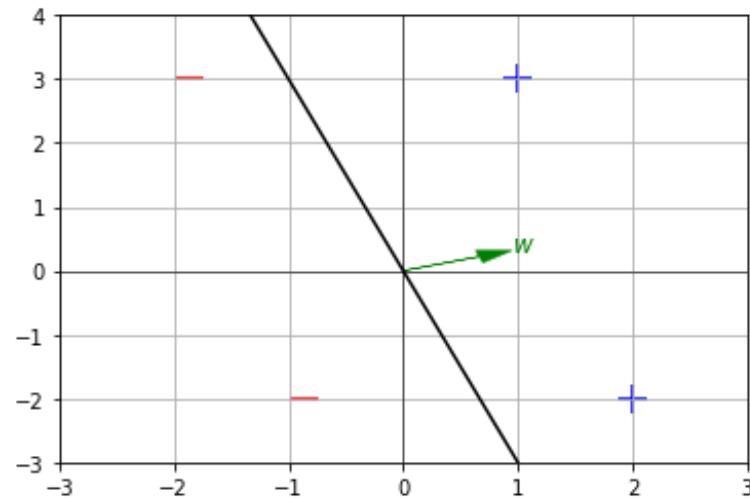
Example - Linear Classifier

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & [-1] \\ [3] & [-2] & [3] & [-2] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$\mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } 3x_1 + 1x_2 > 0 \\ -1 & \text{otherwise} \end{cases}$$

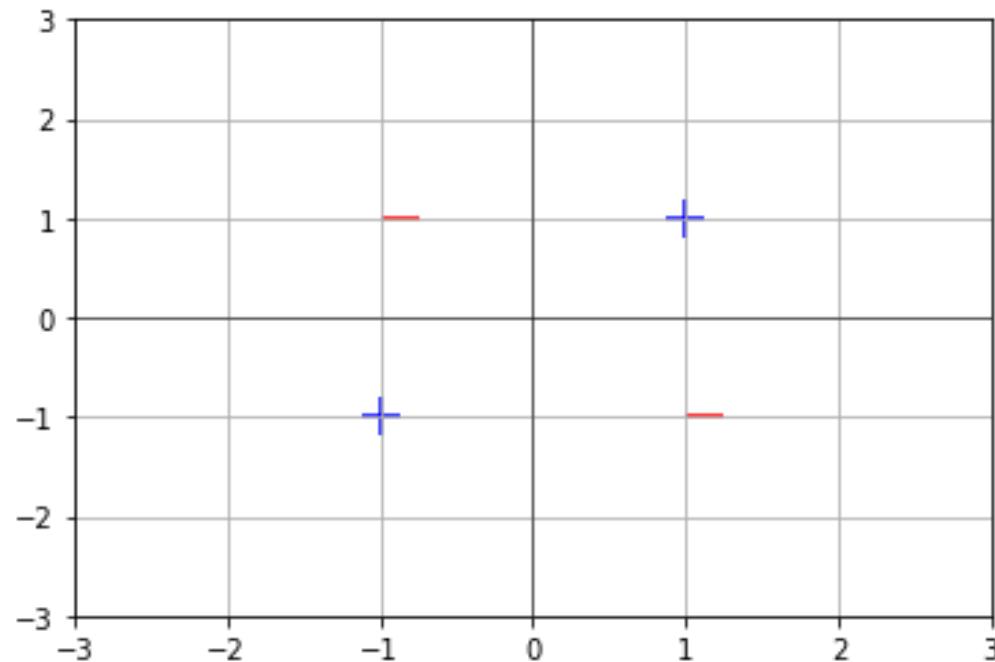


Non-linear Classifiers

"XOR" - Linearly Separable?

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & [-1] \end{bmatrix}$$

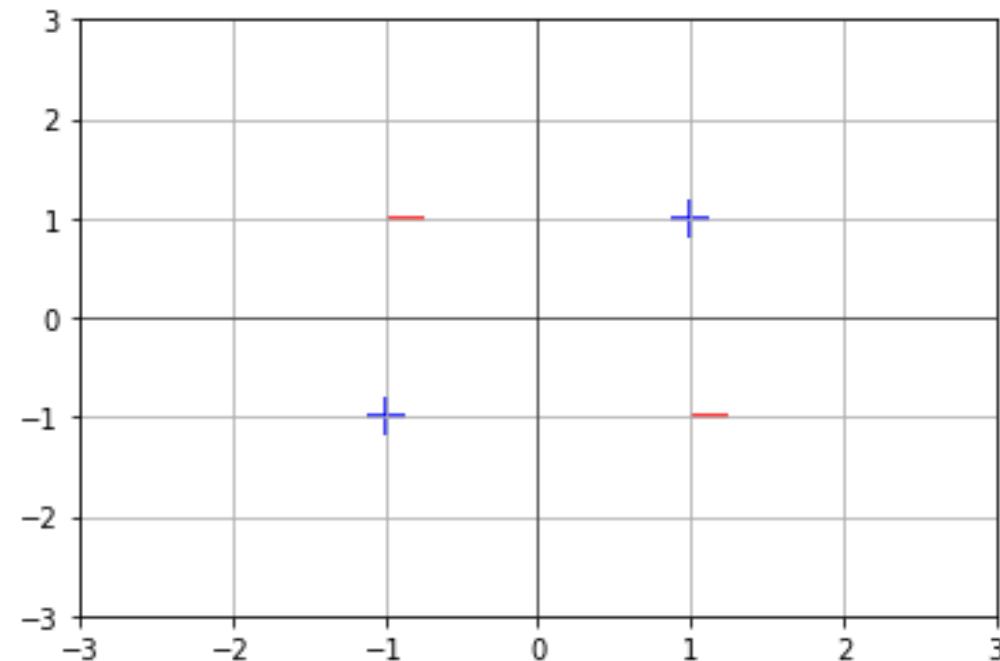


"XOR" - Linearly Separable?

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & [-1] \end{bmatrix}$$

- No linear separator for XOR in two-dimensional space

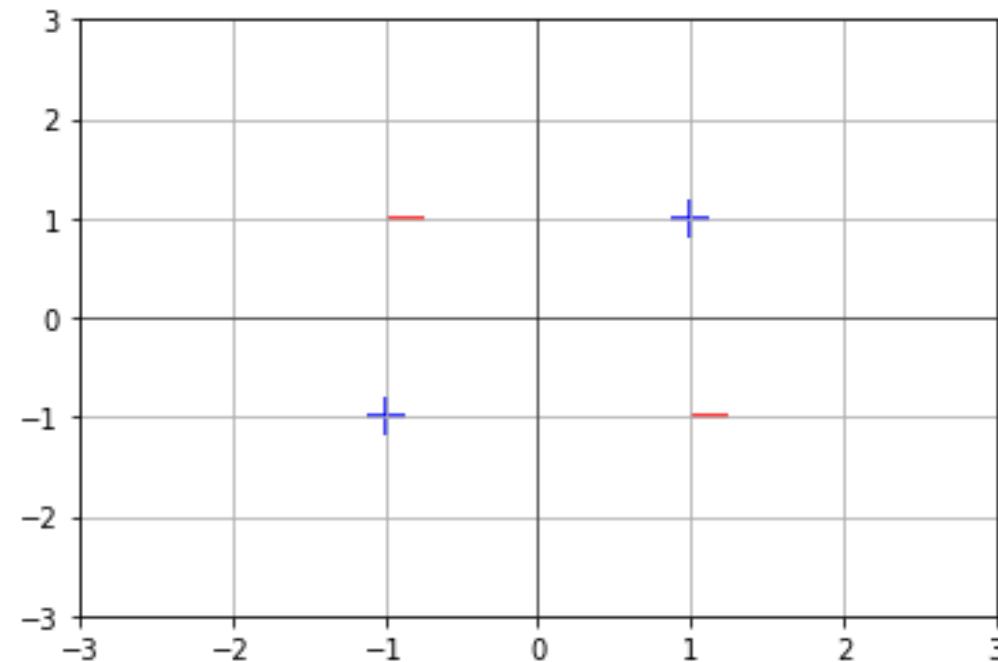


"XOR" - Linearly Separable?

$$\mathbf{X} = \begin{bmatrix} [1] & [-1] & [-1] & [1] \\ [1] & [-1] & [1] & [-1] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

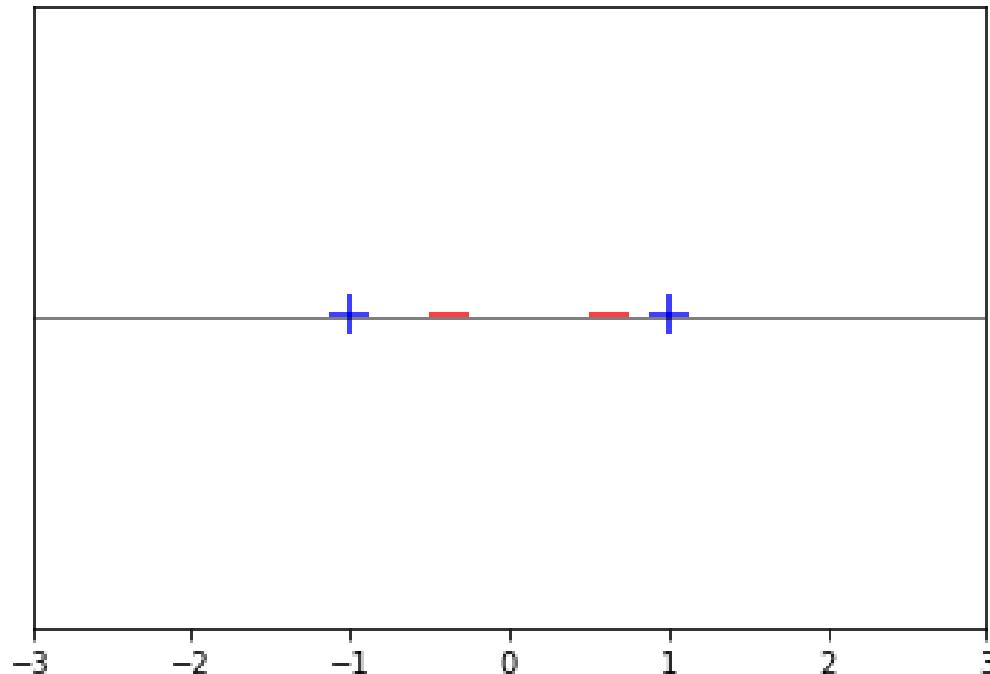
- No linear separator for XOR in two-dimensional space
- **But** we can move the data in a **higher-dimensional space** and find a linear separator there



1-D Example

$$\mathbf{X} = [[-1] \quad [-0.5] \quad [0.5] \quad [1]]$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [+1]]$$

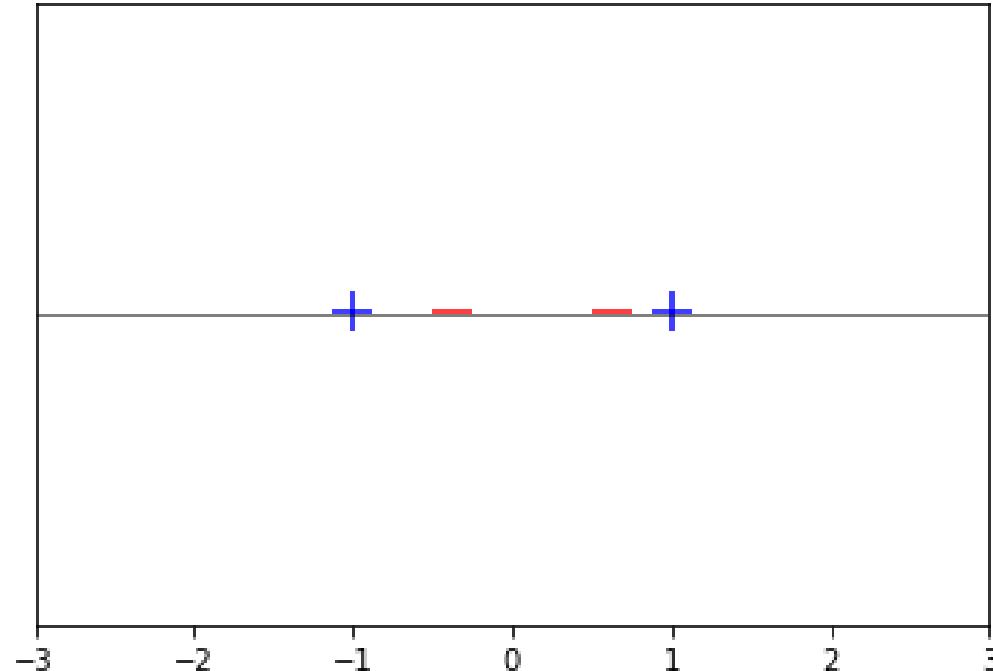


1-D Example

$$\mathbf{X} = [[-1] \quad [-0.5] \quad [0.5] \quad [1]]$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [+1]]$$

$$\phi(\mathbf{x}) = [x \quad x^2]^T$$



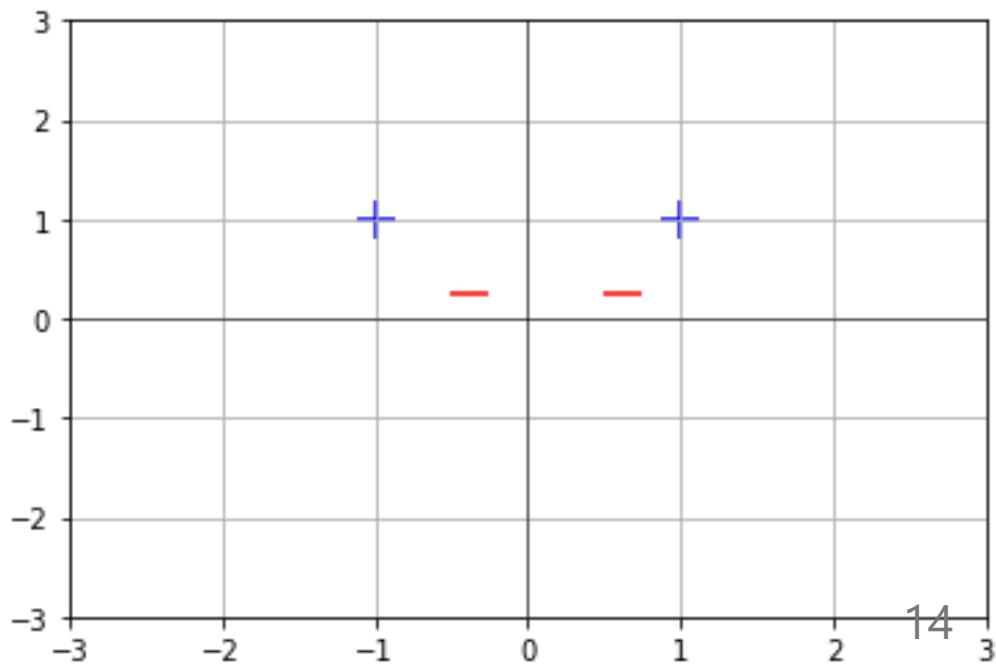
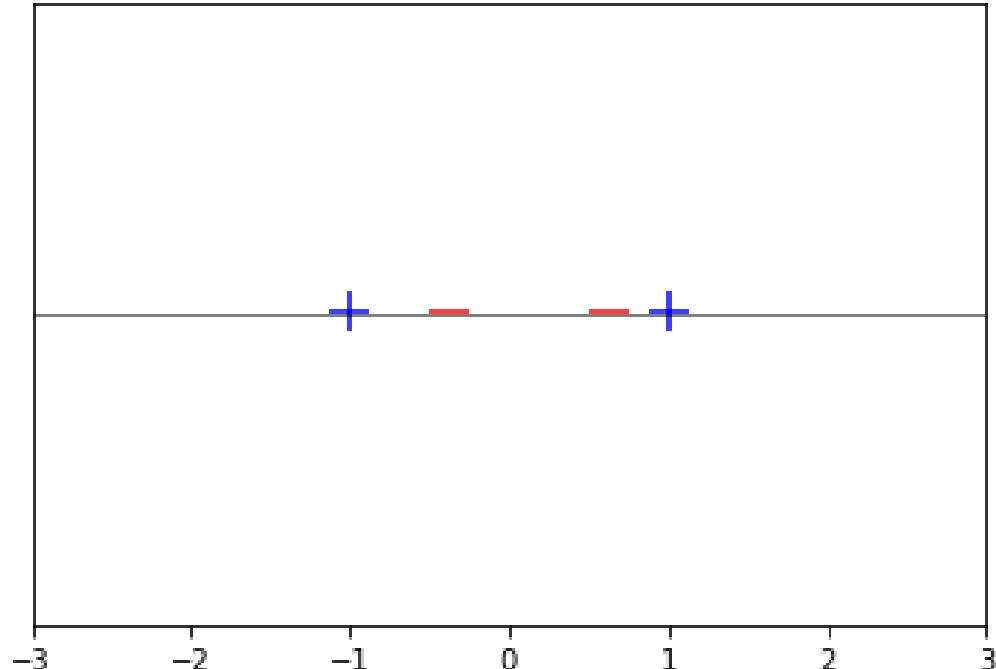
1-D Example - Lifting Function

$$\mathbf{X} = [[-1] \quad [-0.5] \quad [0.5] \quad [1]]$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [+1]]$$

$$\phi(\mathbf{x}) = [x \quad x^2]^T$$

$$\phi(\mathbf{X}) = \begin{bmatrix} [-1] & [-0.5] & [0.5] & [1] \\ [1] & [0.25] & [0.25] & [1] \end{bmatrix}$$



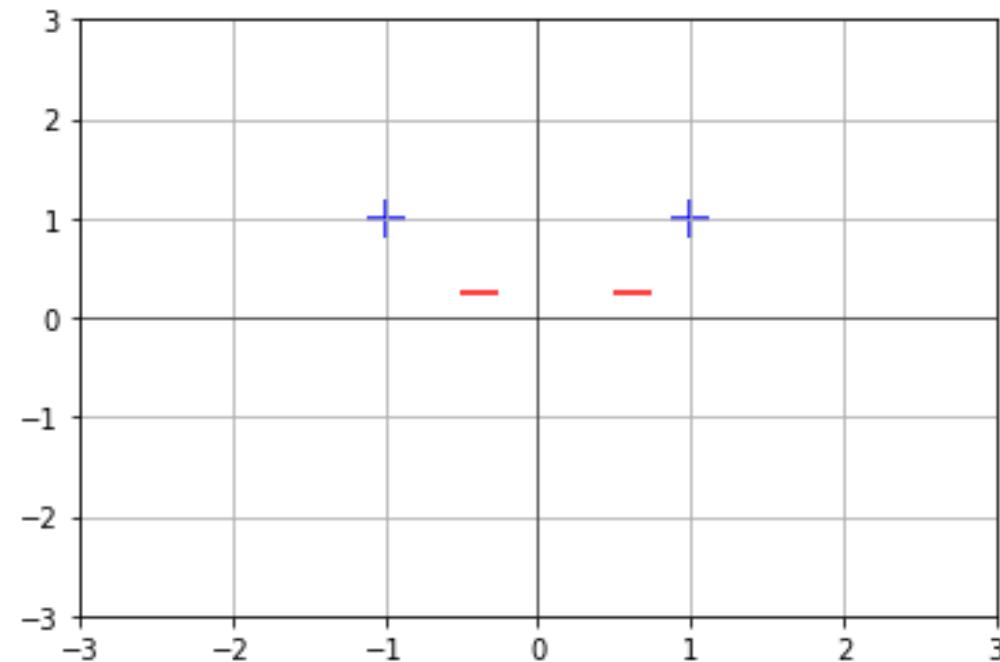
1-D Example - with Offset Term

$$\mathbf{X} = [[-1] \quad [-0.5] \quad [0.5] \quad [1]]$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [+1]]$$

$$\phi(\mathbf{x}) = [1 \quad x \quad x^2]^T$$

$$\phi(\mathbf{X}) = \begin{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -0.5 \\ 0.25 \end{bmatrix} & \begin{bmatrix} 1 \\ 0.5 \\ 0.25 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \end{bmatrix}$$



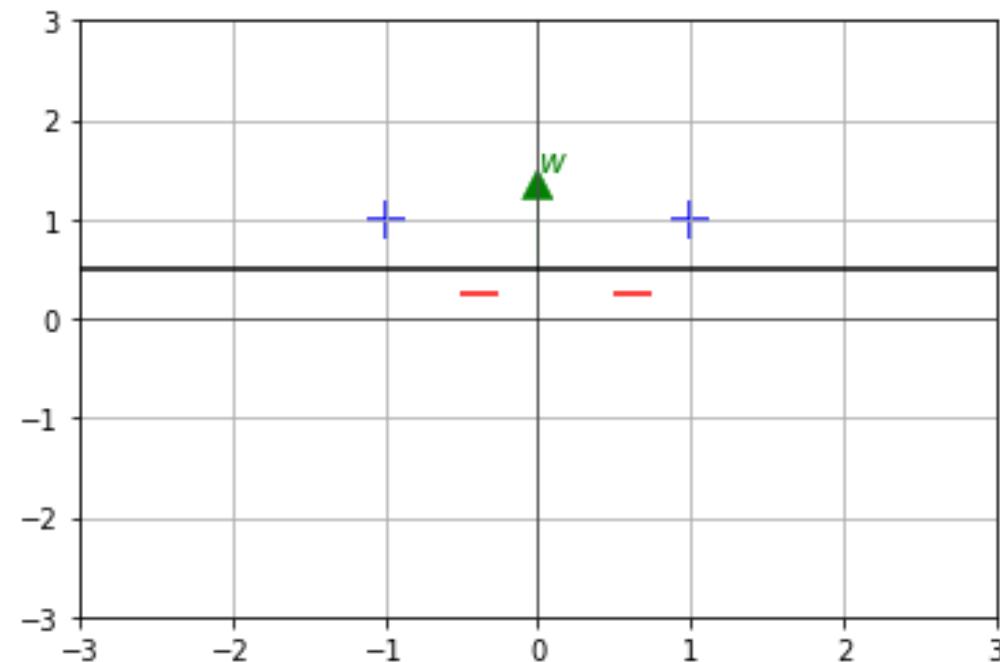
1-D Example

$$\phi(\mathbf{X}) = \begin{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -0.5 \\ 0.25 \end{bmatrix} & \begin{bmatrix} 1 \\ 0.5 \\ 0.25 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [+1]]$$

- Now they are linearly separable

$$\mathbf{w} = \begin{bmatrix} -1 \\ 0 \\ 2 \end{bmatrix}$$



Lifting function $\phi(x)$

- $\phi(x)$ is a *lifting function* that transforms a set of features into a more expressive set of features
 - Selected based on domain knowledge or systematically

Polynomial Basis

k^{th} order basis uses every possibly product of k different dimensions in the original input

Example XOR $k = 2$

$$\phi([x_1 \quad x_2]^T) = [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_1x_2 \quad x_2^2]^T$$

Polynomial Basis

k	1 dimension	General Case
0	$[1]$	$[1]$
1	$[1 \ x]$	$[1 \ x_1 \ \dots \ x_d]$
2	$[1 \ x \ x^2]$	$[1 \ x_1 \ \dots \ x_d \ x_1^2 \ x_1x_2 \ x_1x_3 \ \dots \ x_2^2 \ \dots]$
3	$[1 \ x \ x^2 \ x^3]$	$[1 \ x_1 \ \dots \ x_d \ x_1^2 \ x_1x_2 \ \dots \ x_1^3 \ x_1x_2x_3 \ \dots]$
\vdots	\vdots	\vdots

Example - Apply $k = 2$ polynomial basis to XOR data

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & [-1] \end{bmatrix}$$

$$\phi(\begin{bmatrix} x_1 & x_2 \end{bmatrix}^T) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 \end{bmatrix}^T$$

Example - Apply $k = 2$ polynomial basis to XOR data

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{bmatrix} \quad \mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$\phi(\begin{bmatrix} x_1 & x_2 \end{bmatrix}^T) = [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_1 x_2 \quad x_2^2]^T$$

$$\phi(\mathbf{X}) = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \end{bmatrix}$$

A linear classifier for lifted features

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & [-1] \end{bmatrix}$$

$$\phi(\begin{bmatrix} x_1 & x_2 \end{bmatrix}^T) = [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_1 x_2 \quad x_2^2]^T$$

A linear separator is:

$$\phi(\mathbf{X}) = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$

A linear classifier for lifted features

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} [+1] & [+1] & [-1] & [-1] \end{bmatrix}$$

$$\phi(\begin{bmatrix} x_1 & x_2 \end{bmatrix}^T) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 \end{bmatrix}^T$$

A linear separator is:

$$\phi(\mathbf{X}) = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$

$$0 + 0x_1 + 0x_2 + 0x_1^2 + 2x_1 x_2 + 0x_2^2 = 0$$

XOR

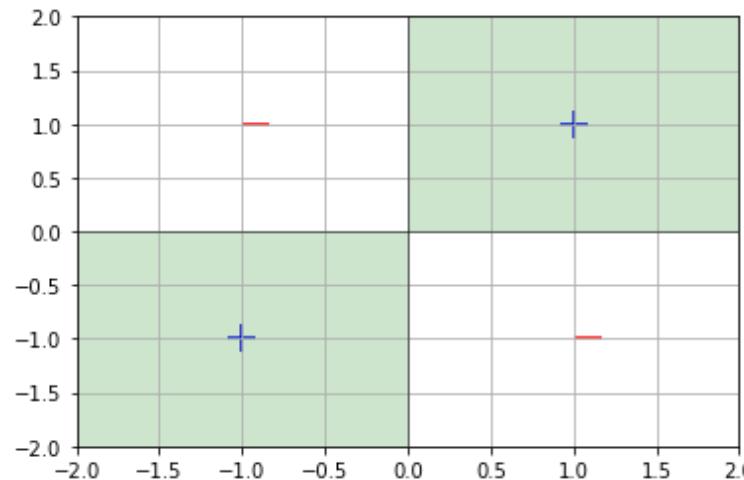
$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$0 + 0x_1 + 0x_2 + 0x_1^2 + 2x_1x_2 + 0x_2^2 = 0$$

That is a linear classifier in 6-D space shown below:

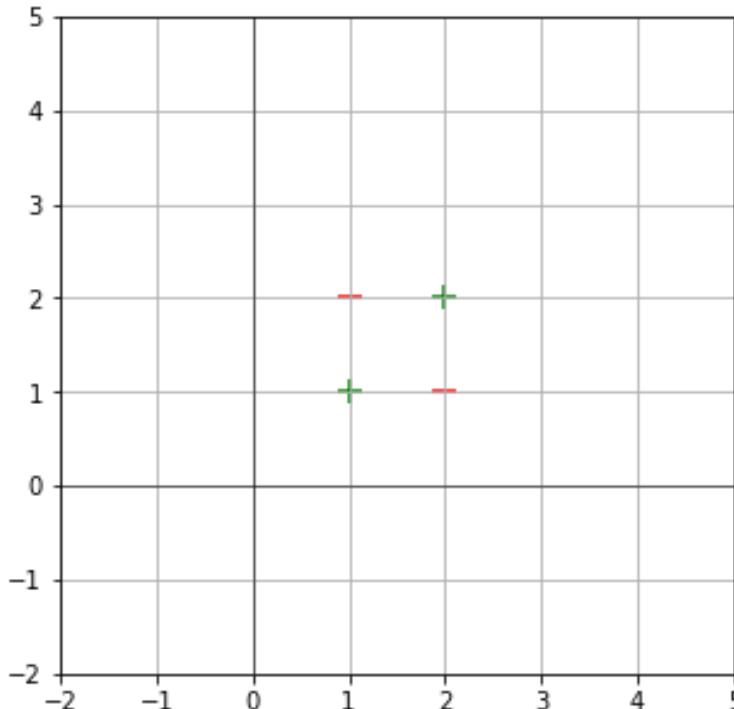
$$\phi(\mathbf{X}) = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$



Example XOR but the data are in a different place

$$\mathbf{X} = \begin{bmatrix} [2] & [1] & [1] & [2] \\ [2] & [1] & [2] & [1] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$



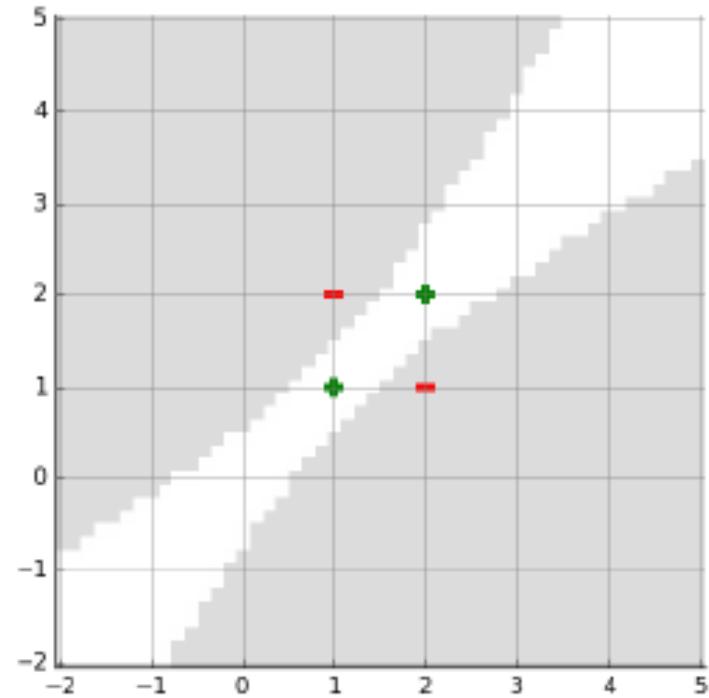
Example XOR but the data are in a different place

$$\mathbf{X} = \begin{bmatrix} [2] & [1] & [1] & [2] \\ [2] & [1] & [2] & [1] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

After ~ 60 revisions

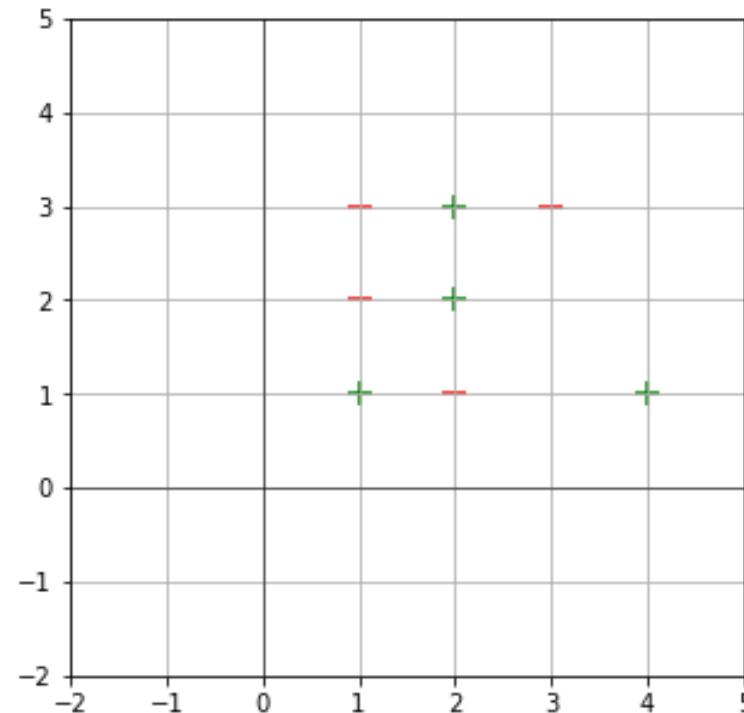
$$\mathbf{w} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -5 \\ 11 \\ -5 \end{bmatrix}$$



Example - A harder dataset

$$\mathbf{X} = \begin{bmatrix} [1] & [1] & [1] & [2] & [2] & [2] & [3] & [4] \\ [1] & [2] & [3] & [1] & [2] & [3] & [3] & [1] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [-1] \quad [+1] \quad [+1] \quad [-1] \quad [+1]]$$

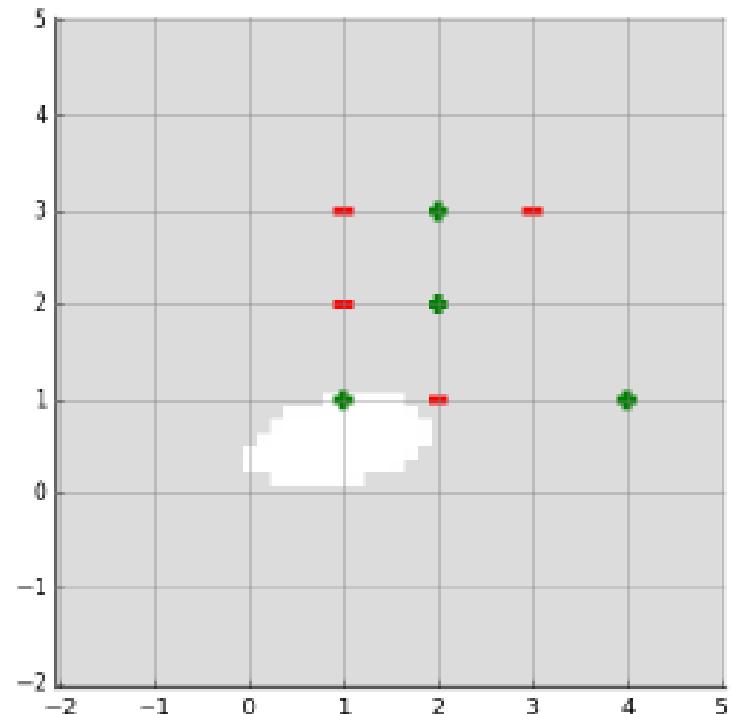


$k = 2$ after 200 iterations

$$k = 2 \quad [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_1 x_2 \quad x_2^2]^T$$

$$\mathbf{X} = \begin{bmatrix} [1] & [1] & [1] & [2] & [2] & [2] & [3] & [4] \\ [1] & [2] & [3] & [1] & [2] & [3] & [3] & [1] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [-1] \quad [+1] \quad [+1] \quad [-1] \quad [+1]]$$

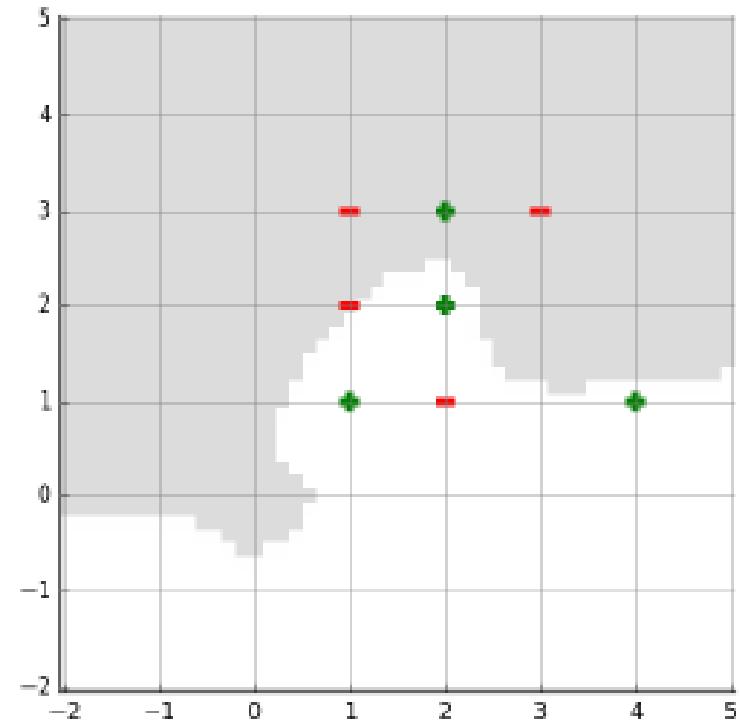


$k = 3$ after 200 iterations

$$k = 3 \quad [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_1x_2 \quad \dots \quad x_1^3 \quad x_1^2x_2 \quad \dots]$$

$$\mathbf{X} = \begin{bmatrix} [1] & [1] & [1] & [2] & [2] & [2] & [3] & [4] \\ [1] & [2] & [3] & [1] & [2] & [3] & [3] & [1] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [-1] \quad [+1] \quad [+1] \quad [-1] \quad [+1]]$$

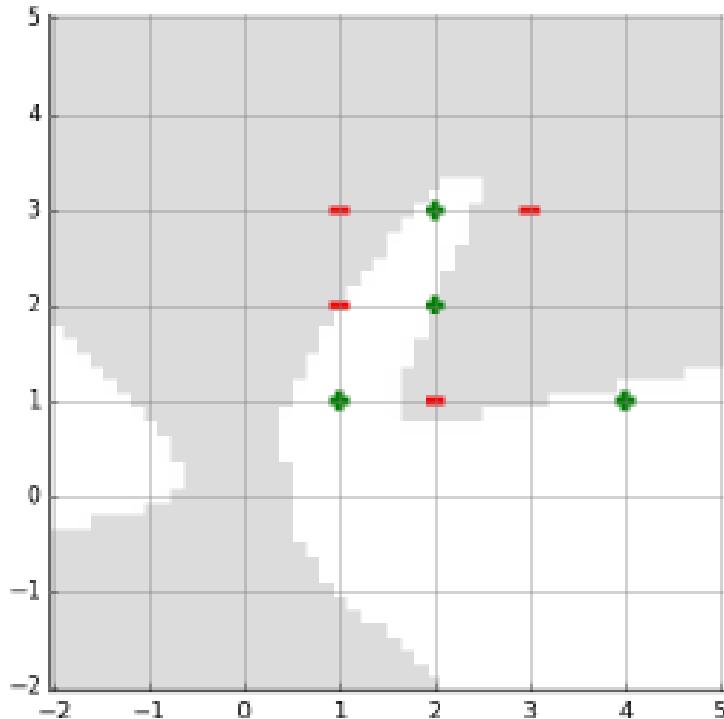


$k = 4$ after 200 iterations

$$k = 4 \quad [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_1x_2 \quad \dots \quad x_1^3 \quad x_1^2x_2 \quad \dots \quad x_1^4 \quad x_1^3x_2 \dots]$$

$$\mathbf{X} = \begin{bmatrix} [1] & [1] & [1] & [2] & [2] & [2] & [3] & [4] \\ [1] & [2] & [3] & [1] & [2] & [3] & [3] & [1] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [-1] \quad [+1] \quad [+1] \quad [-1] \quad [+1]]$$

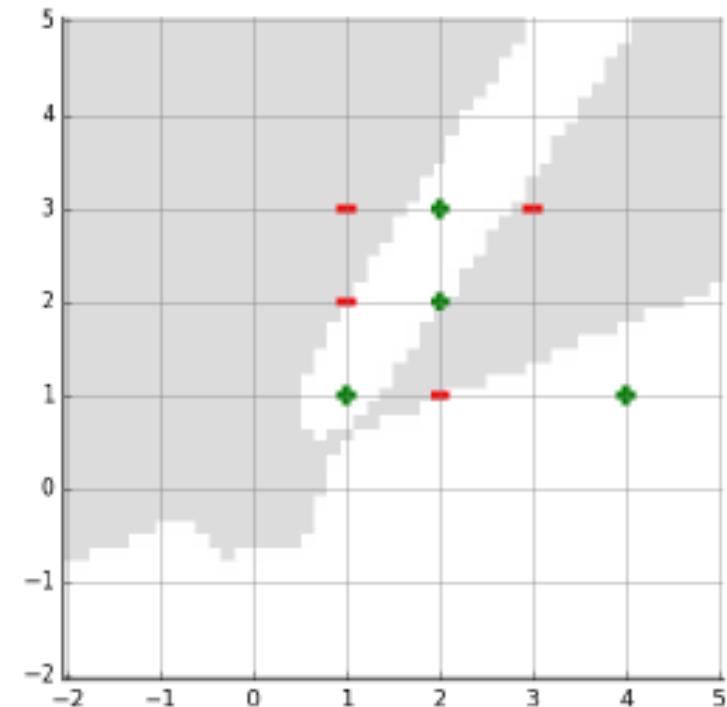


$k = 5$ after 200 iterations

$$k = 5 \quad [1 \ x_1 \ x_2 \ x_1^2 \ x_1x_2 \ \dots \ x_1^3 \ x_1^2x_2 \ \dots \ x_1^4 \ x_1^3x_2 \dots \ x_1^5 \ \dots]$$

$$\mathbf{X} = \left[\begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad \begin{bmatrix} 3 \\ 3 \end{bmatrix} \quad \begin{bmatrix} 4 \\ 1 \end{bmatrix} \right]$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [-1] \quad [+1] \quad [+1] \quad [-1] \quad [+1]]$$



How many terms do you need?

- The goal is to transform the data into a high dimensional space where there is a *linear separator*
- **Overparameterizing:** models with more parameter than data points.

Convergence theorem

If training dataset \mathcal{D} is linearly separable, the perceptron algorithm is

guaranteed to find a linear separator by making at most $\left(\frac{R}{\gamma}\right)^2$ mistakes

How many terms do you need?

- In perceptron the number of steps required depends on the maximum norm of data and the norm of w^* . The dimension of the data plays no role.
- We need to design features where w^* has a controlled norm (but this is not related with the number of dimensions)

Kernel Functions

Kernel Functions

Suppose

- you work with two dimensional data
 - $x = [x_1 \quad x_2], x' = [x'_1 \quad x'_2]$

Kernel Functions

Suppose

- you work with two dimensional data

- $x = [x_1 \quad x_2], x' = [x'_1 \quad x'_2]$

- you lift $\phi(x)$ and $\phi(x')$ to $k = 2$ polynomial

- $\phi(x) = [x_1^2 \quad x_1x_2 \quad x_1x_2 \quad x_2^2], \phi(x') = [x'_1{}^2 \quad x'_1x'_2 \quad x'_1x'_2 \quad x'_2{}^2]$

- You want to find the dot product of $\phi(x) \cdot \phi(x')$

Kernel Functions

Suppose

- you work with two dimensional data

- $x = [x_1 \ x_2], x' = [x'_1 \ x'_2]$

- you lift $\phi(x)$ and $\phi(x')$ to polynomial

- $\phi(x) = [x_1^2 \ x_1x_2 \ x_1x_2 \ x_2^2], \phi(x') = [x'^2_1 \ x'_1x'_2 \ x'_1x'_2 \ x'^2_2]$

- you want to find the dot product of $\phi(x) \cdot \phi(x')$

$$\phi(x) \cdot \phi(x') = [x_1^2 \ x_1x_2 \ x_2x_1 \ x_2^2] \cdot [x'^2_1 \ x'_1x'_2 \ x'_2x'_1 \ x'^2_2]$$

Kernel Functions

- With kernel function we can make this operation in the original space and get results in lifted space

$$K(x, x') = (x \cdot x')^2$$

- This gives the same results as $\phi(x) \cdot \phi(x')$

$$K(x, x') = \phi(x) \cdot \phi(x')^2$$

Example

$$x = [3 \quad 5], x' = [2 \quad 1]$$

$$\phi(x) = [x_1^2 \quad x_1x_2 \quad x_1x_2 \quad x_2^2]$$

Example

$$x = [3 \ 5], x' = [2 \ 1]$$

$$\phi(x) = [x_1^2 \ x_1x_2 \ x_1x_2 \ x_2^2]$$

$$\phi(x) = [9 \ 15 \ 15 \ 25]$$

$$\phi(x') = [4 \ 2 \ 2 \ 1]$$

$$\phi(x) \cdot \phi(x') = [9 \ 15 \ 15 \ 25]^T \cdot [4 \ 2 \ 2 \ 1]$$

$$= 36 + 30 + 30 + 25 = 121$$

Example

$$x = [3 \ 5], x' = [2 \ 1]$$

$$\phi(x) = [x_1^2 \ x_1x_2 \ x_1x_2 \ x_2^2]$$

$$\phi(x) = [9 \ 15 \ 15 \ 25]$$

$$\phi(x') = [4 \ 2 \ 2 \ 1]$$

$$\phi(x) \cdot \phi(x') = [9 \ 15 \ 15 \ 25]^T \cdot [4 \ 2 \ 2 \ 1]$$

$$= 36 + 30 + 30 + 25 = 121$$

$$K(x_i, x_j) = ([3 \ 5]^T \cdot [2 \ 1])^2 = (11)^2 = 121$$

Example

$$x = [3 \ 5], x' = [2 \ 1]$$

$$\phi(x) = [x_1^2 \ x_1x_2 \ x_1x_2 \ x_2^2]$$

$$\phi(x) = [9 \ 15 \ 15 \ 25]$$

$$\phi(x') = [4 \ 2 \ 2 \ 1]$$

$$\phi(x) \cdot \phi(x') = [9 \ 15 \ 15 \ 25]^T \cdot [4 \ 2 \ 2 \ 1]$$

$$= 36 + 30 + 30 + 25 = \boxed{121}$$

$$K(x_i, x_j) = ([3 \ 5]^T \cdot [2 \ 1])^2 = (11)^2 = \boxed{121}$$

Kernel Function

All linear methods rely on dot product so Kernel Function turns out to be very useful

Kernel Perceptron

Perceptron

- Start $\mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$
- For $t = 0, 1, 2, \dots$:
 - Select a random index $i \in \{1, \dots, n\}$
 - If: $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$
$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$
 - Else:
$$\mathbf{w} = \mathbf{w}$$

Perceptron with lifting

e.g. $\phi(x) = [x_1^2 \quad x_1x_2 \quad x_1x_2 \quad x_2^2]$

⋮

- If: $y^{(i)}(\mathbf{w}^T \phi(\mathbf{x}^{(i)})) \leq 0$

$$\mathbf{w} = \mathbf{w} + y^{(i)} \phi(\mathbf{x}^{(i)})$$

- Else:

$$\mathbf{w} = \mathbf{w}$$

- If: $y^{(i)}(\mathbf{w}^T \phi(\mathbf{x}^{(i)})) \leq 0$

$$\mathbf{w} = \mathbf{w} + y^{(i)} \phi(\mathbf{x}^{(i)})$$

- Else:

$$\mathbf{w} = \mathbf{w}$$

Let α_j be the number of mistakes made for example j . The final weights are

$$w = \sum_{j=1}^n \alpha_j y^{(j)} \phi(x^{(j)})$$

$$w = \sum_{j=1}^n \alpha_j y^{(j)} \phi(x^{(j)})$$

When we make a prediction for example i i.e. $w \cdot \phi(x^{(i)})$

$$w = \sum_{j=1}^n \alpha_j y^{(j)} \phi(x^{(j)})$$

When we make a prediction for example i i.e. $w \cdot \phi(x^{(i)})$

$$w\phi(x^{(i)}) = \sum_{j=1}^h \alpha_j y^{(j)} \phi(x^{(j)}) \phi(x^{(i)})$$

$$w = \sum_{j=1}^n \alpha_j y^{(j)} \phi(x^{(j)})$$

When we make a prediction for example i i.e. $w \cdot \phi(x^{(i)})$

$$w\phi(x^{(i)}) = \sum_{j=1}^h \alpha_j y^{(j)} \overbrace{\phi(x^{(j)})\phi(x^{(i)})}^{K(x^{(j)}, x^{(i)})}$$

Therefore predictions can be described by

$$w\phi(x^{(i)}) = \sum_{j=1}^h \alpha_j y^{(j)} K(x^{(j)}, x^{(i)})$$

Kernel Perceptron Algorithm

$$\bullet \text{ Start } \boldsymbol{\alpha} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

- For $t = 0, 1, 2, \dots$:
 - Select a random index $i \in \{1, \dots, n\}$
 - If: $y^{(i)} \sum_{j=1}^n \alpha_j y^{(j)} K(x^{(j)}, x^{(i)}) \leq 0$

$$\alpha_i = \alpha_i + 1$$

- Else:

$$\alpha_i = \alpha_i$$

Kernels are powerful!

- Polynomial kernel of degree- d : $K(x, x') = (x \cdot x' + 1)^d$
 - e.g. $d = 2$ lifts to $\phi(x) = [1 \quad \sqrt{2}x_1 \quad \sqrt{2}x_2 \quad x_1^2 \quad \sqrt{2}x_1x_2 \quad x_2^2]$
- Radial basis kernel: $K(x, x') = e^{-\frac{1}{2}\|x-x'\|^2}$

Editing and Encoding Features

Editing Features

A large amount of effort goes into selecting and editing features

The choice of features determines

1. error rates for decision boundaries
2. whether training error can be optimized with them
3. to what extent generalization will be achieved

Encoding Features

- Most machine learning techniques uses numeric features.
- Useful strategies are required to turn discrete values into real numbers:
 - Numeric
 - Thermometer code (dummy variables)
 - Factored code
 - One-hot code

Numeric Encoding

- If you have k discrete values, turn them into numbers by $1.0/k, 2.0/k \dots 1.0$

Numeric Encoding

- If you have k discrete values, turn them into numbers by $1.0/k, 2.0/k \dots 1.0$
- Numeric encoding should only be used when the discrete values signify some numeric quantity so these numerical values are meaningful
 - e.g. $\{None, OneThird, TwoThirds, Whole\}$ to $\{0.0, 0.33, 0.66, 1.0\}$

Thermometer Encoding (Dummy Variables)

- Used when discrete variables have a ordering from $1, \dots, k$ but not a mapping into real numbers
 - This is often the case

Thermometer Encoding (Dummy Variables)

- Used when discrete variables have a ordering from $1, \dots, k$ but not a mapping into real numbers
 - This is often the case
- Use a vector of length k , convert discrete input value $0 < j \leq k$ into a vector where the first j values are 1.0 and the remaining values are 0.0

Thermometer Encoding (Dummy Variables)

- Used when discrete variables have a ordering from $1, \dots, k$ but not a mapping into real numbers
- Use a vector of length k , convert discrete input value $0 < j \leq k$ into a vector where the first j values are 1.0 and the remaining values are 0.0

e.g. $Pain = \{None, Low, Medium, High\}$

$$None = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad Low = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad Medium = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad High = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Thermometer Encoding (Dummy Variables)

Note: usually a vector of $k - 1$ length is used for a variable with k values e.g.

$$None = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad Low = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad Medium = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad High = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

One-hot Encoding

- Used when there is no obvious numeric, ordering or factorial structure
- Use a vector of length k , convert discrete input value $0 < j \leq k$ to a vector where j^{th} value is 1.0 and all other values are 0.0

One-hot Encoding - Example

- Blood type $\{A+, A-, B+, B-, 0+, 0-, AB+, AB-\}$

One-hot Encoding - Example

- Blood type $\{A+, A-, B+, B-, 0+, 0-, AB+, AB-\}$
- One hot encoding, Blood Type is $B+:$

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Factored Encoding

- It is used when discrete values can be decomposed into two parts
 - e.g. Maker and model of a car (Toyota Corolla)
 - e.g. Blood Type $\{A, B, AB, 0\}$ $\{+, -\}$

Factored Encoding

- It is used when discrete values can be decomposed into two parts
 - e.g. Maker and model of a car (Toyota Corolla)
 - e.g. Blood Type $\{A, B, AB, 0\}$ $\{+, -\}$
 - We can factor features $\{A, B, AB, 0\}$ into $\{A, \text{not } A\}$ $\{B, \text{not } B\}$
 - 0 can be considered not A and not B
 - As a result we can encode it into a 3-D vector

$$B- = \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix} \quad 0+ = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}$$

Numeric features: scaling - standardization

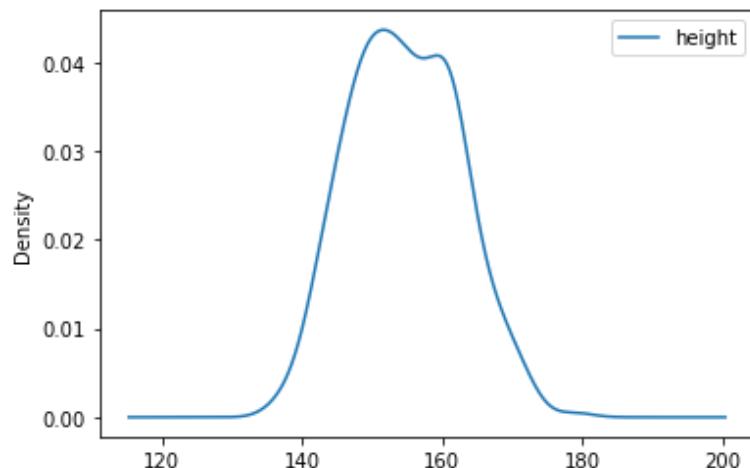
- Most ML techniques do not work well on numeric features with different scales
- It is useful to scale the numeric features
- A typical scaling is **standardization** where we transform the numeric feature x as

$$\frac{x - \mu}{\sigma}$$

where μ is the average of x and σ is the standard deviation of x

Standardization

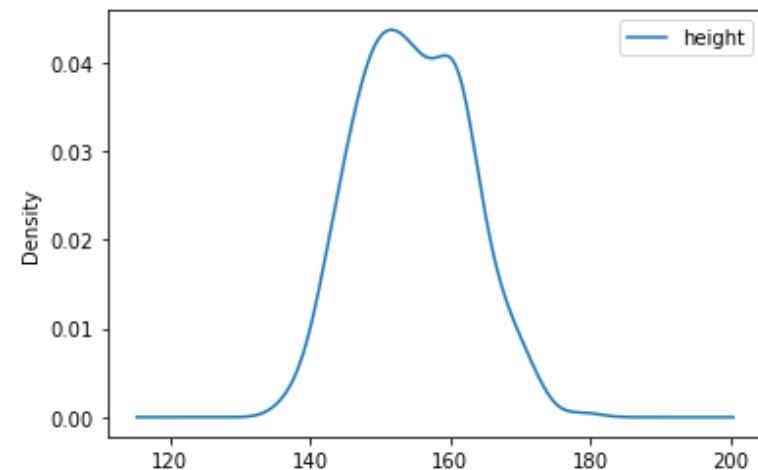
```
height [158, 140, 137, 159, ...]
```



Standardization

```
height [152, 140, 137, 156, ...]
```

- Average height `np.mean(height)` 154.6
- Standard deviation of height `np.std(height)` 7.7



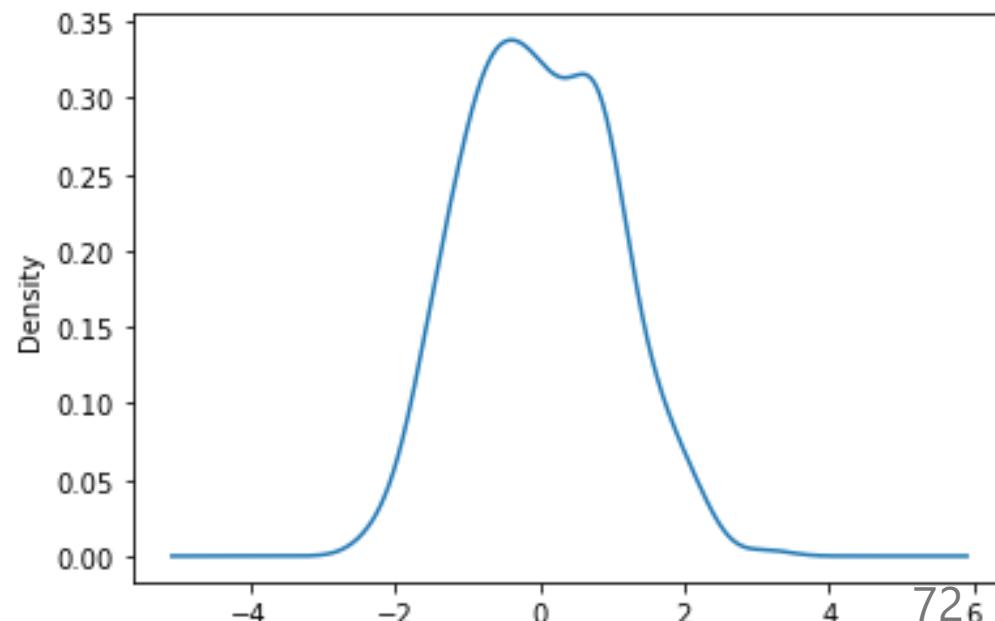
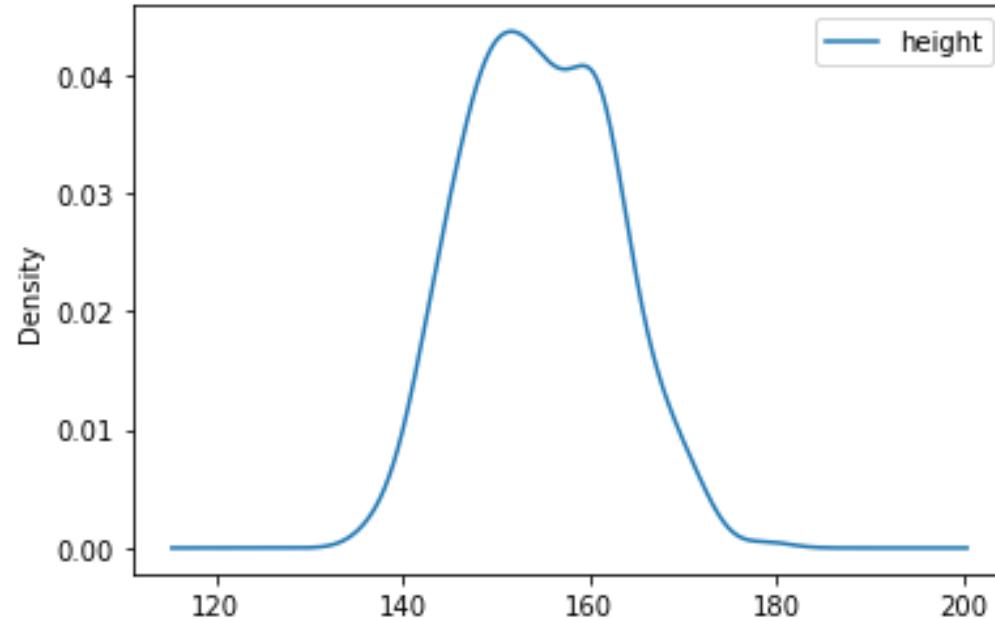
Standardization

```
height [158, 140, 137, 159, ...]
```

- Average height `np.mean(height)` 154.6
- Standard deviation of height

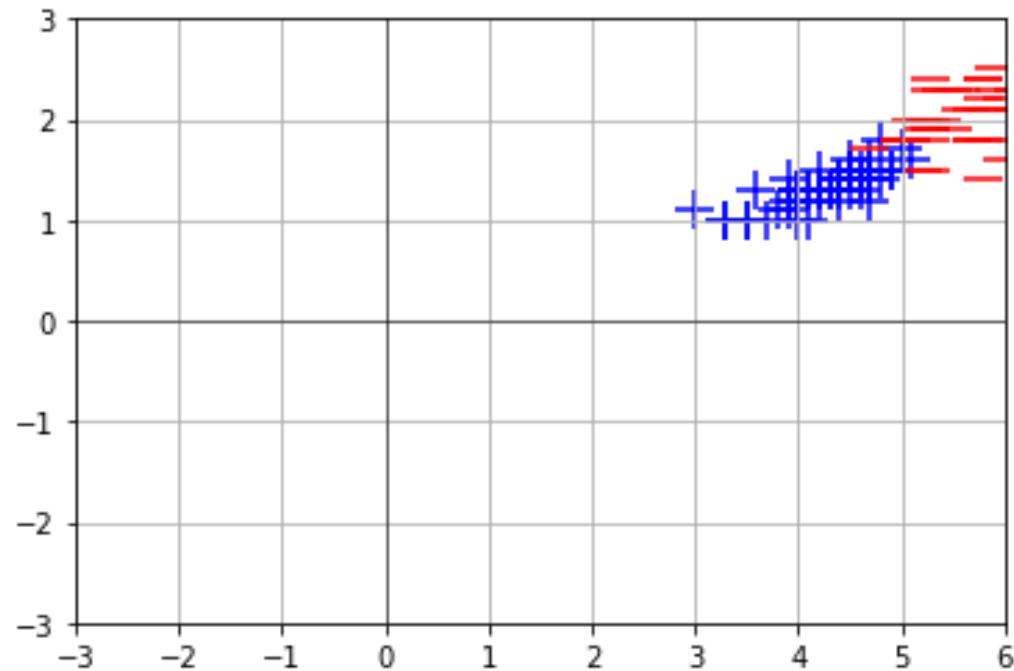
```
np.std(height) 7.7
```

```
(height - mu)/ sigma [-0.37, -1.92, -2.33,  
0.29, ...]
```



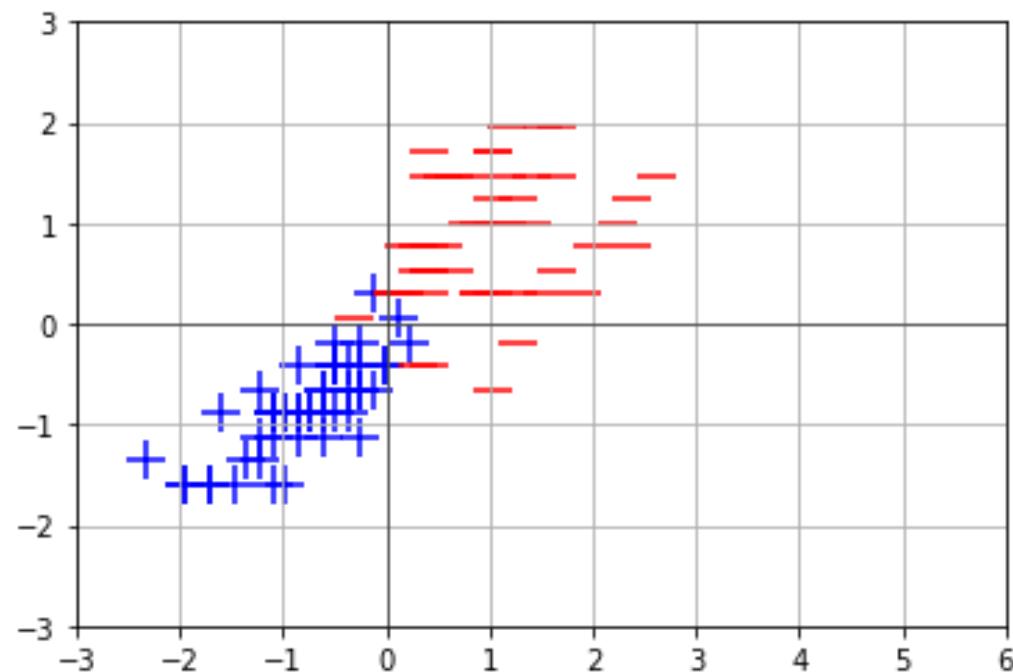
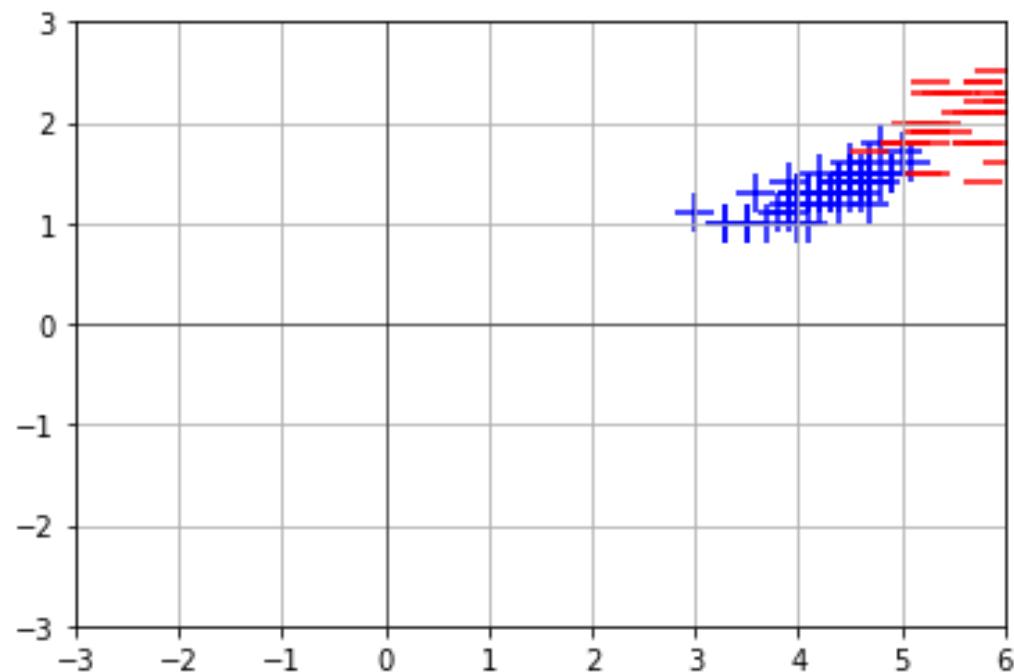
Standardization - IRIS Dataset

Virginica vs Versicolor



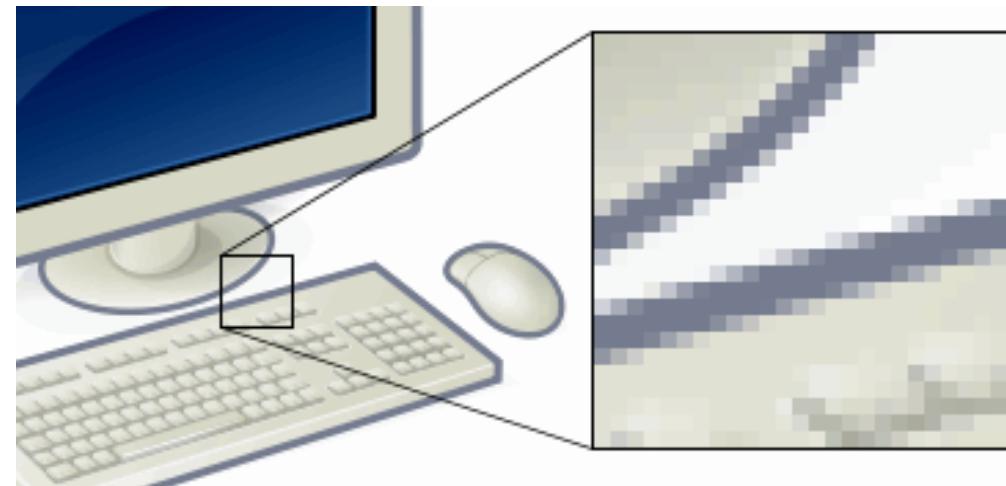
Standardization - IRIS Dataset

Virginica vs Versicolor



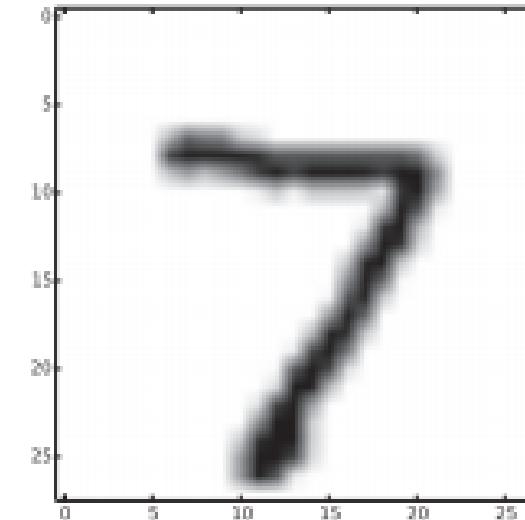
Images

- Bitmap file is an array indexed by three coordinates (i.e. tensor of order 3)
- Each pixel x_{ijk} denotes intensity at row i , column j and color channel k
 - Quantity of each three primary colors at each grid location
- Useful for displaying on screen but not useful for prediction
- Small modifications on an image can be far away in pixel representation



Grayscale images

- In grayscale images, each pixel takes a value between 0 and P .
- In this example, between 0 and 255



	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	185	159	151	60	36	0	0	0	0	0	0	0	0	0
8	254	254	254	254	241	198	198	198	198	198	198	198	198	170
9	114	72	114	163	227	254	225	254	254	254	250	229	254	254
10	0	0	0	0	17	66	14	67	67	67	59	21	236	254
11	0	0	0	0	0	0	0	0	0	0	0	83	253	209
12	0	0	0	0	0	0	0	0	0	0	22	233	255	83
13	0	0	0	0	0	0	0	0	0	0	129	254	238	44
14	0	0	0	0	0	0	0	0	0	59	249	254	62	0
15	0	0	0	0	0	0	0	0	0	133	254	187	5	0
16	0	0	0	0	0	0	0	0	9	205	248	58	0	0
17	0	0	0	0	0	0	0	0	126	254	182	0	0	0
18	0	0	0	0	0	0	0	75	251	240	57	0	0	0
19	0	0	0	0	0	0	19	221	254	166	0	0	0	0
20	0	0	0	0	0	3	203	254	219	35	0	0	0	0
21	0	0	0	0	38	254	254	77	0	0	0	0	0	0
22	0	0	0	0	31	224	254	115	1	0	0	0	0	0
23	0	0	0	0	133	254	254	52	0	0	0	0	0	0
24	0	0	0	61	242	254	254	52	0	0	0	0	0	0
25	0	0	0	121	254	254	219	40	0	0	0	0	0	0
26	0	0	0	121	254	207	18	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Text

- How to embed a document as a vector?

Text - One-hot encoding

"He wrote a book"

he [1, 0, 0, 0, 0, 0, 0, 0, ...]

wrote [0, 1, 0, 0, 0, 0, 0, 0, ...]

a [0, 0, 1, 0, 0, 0, 0, 0, ...]

book [0, 0, 0, 1, 0, 0, 0, 0, ...]

Text - One-hot encoding

- A large vector of the same size as the dictionary for every word in the document.
- Each component of the vector corresponds to a one dictionary word. If a word occurs, the corresponding value in the vector is 1. All values corresponding to words that are not in the document are 0
- No order of words
- Sparse vectors
- Sequential models (features inputted word-by-word or character-by-character)

Template Matching

Template matching aims to extract higher level patterns which can be more useful for discriminative tasks

- Convolution
 - Small templates that count the number of occurrence of some pattern
 - Multiple convolutions can be combined
 - e.g. A template for heads, legs, tails for a system to detect animals in images
- Fourier transforms
 - Sinusoid patterns in spatial and temporal data
 - Measure the amount of oscillation in a vector

Summarization

Summarization techniques can be used to remove noise and reduce dimensionality of input

Summarization

Bag of words

- Summarizes one-hot encoding
- Sum up one-hot encoding of each word occurring in the text
- Each component is the number of times the associated word appears in the document

He wrote a book. She read his book and wrote a commentary.

One-hot encoding

he [1, 0, 0, 0, 0, 0, 0, 0, ...]

wrote [0, 1, 0, 0, 0, 0, 0, 0, ...]

a [0, 0, 1, 0, 0, 0, 0, 0, ...]

book [0, 0, 0, 1, 0, 0, 0, 0, ...]

she [0, 0, 0, 0, 1, 0, 0, 0, ...]

read [0, 0, 0, 0, 0, 1, 0, 0, ...]

his [0, 0, 0, 0, 0, 0, 1, 0, ...]

book [0, 0, 0, 1, 0, 0, 0, 0, ...]

and [0, 0, 0, 0, 0, 0, 0, 1, ...]

...

He wrote a book. She read his book and wrote a commentary.

Bag of words

```
{"he":1, "wrote":2, "a": 2, "book": 2, she: "1", "read": 1, "his": 1, "and", : 1,  
"commentary": 1}
```

Summarization

Pooling (downsampling)

- Locally summarize pixels of image
- For example, break an image into 3×3 grids, take the maximum in each grid. This summarizes local variability and reduces size.

Output from the convolutional layer & ReLU:

0	0	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	0

COGS514 - Cognition and Machine Learning

Logistic Regression and Support Vector Machine (SVM)

Three Important Questions for Machine Learning

Representation

- What class of \mathcal{H} functions (classifiers) we should choose?

Optimization

- How can we efficiently solve the optimization problem?

Generalization

- Will the performance of a classifier generalize from training data to unseen data?

Classification

- Suppose we work on a classification problem with labels:

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

and features

$$\mathbf{x} \in \mathbb{R}^d$$

- In supervised learning we will have a training data of form

$$\mathcal{D} \in \left\{ (\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}) \right\}$$

- A binary classifier h is a mapping $\mathbb{R}^d \rightarrow \{-1, +1\}$

$$\mathbf{x} \rightarrow \boxed{h} \rightarrow y$$

Perceptron Algorithm

- Start $\mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$

- **for** $t = 0, 1, 2, \dots$:

 - Select a random index $i \in \{1, \dots, n\}$
 - **If**: $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \leq 0$

$$\mathbf{w} = \mathbf{w} + y^{(i)} \mathbf{x}^{(i)}$$

- **Else**:

$$\mathbf{w} = \mathbf{w}$$

Loss function

$$\ell(i, j)$$

loss function of declaring $y = i$ when in reality $y = j$

Classifier

$$\mathbf{x} \rightarrow \boxed{h} \rightarrow y$$

Risk

$$R[h] = \mathbb{E}[\ell(h((\mathbf{x}), y)]$$

Classifier

$$\mathbf{x} \rightarrow \boxed{h} \rightarrow y$$

Empirical Risk

$$R_{\mathcal{D}}[h] = \mathbb{E}_{\mathcal{D}}[\ell(h(\mathbf{x}), y)]$$

$$= \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}), y^{(i)})$$

Empirical Risk Minimization - Supervised Learning

- Empirical Risk

$$R_{\mathcal{D}}[h] = \mathbb{E}_{\mathcal{D}}[\ell(h(\mathbf{x}), y)]$$

$$= \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}), y^{(i)})$$

- In supervised learning we aim to find a good classifier from data points by optimizing empirical risk

$$\min_{h \in \mathcal{H}} R_{\mathcal{D}}[h]$$

- Empirical risk $R_{\mathcal{D}}[h]$ serves as a proxy for risk $R[h]$

Optimization

minimize $_{\mathbf{w}}$ $\Phi(\mathbf{w})$:

where $\Phi(\mathbf{w}) : \mathbb{R}^d \rightarrow \mathbb{R}$ is a real valued function over the domain of \mathbb{R}^d

Typical Machine Learning Objective Function

$$\Phi(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) + \lambda R(\mathbf{w})$$

Typical Machine Learning Objective Function

$$\Phi(\mathbf{w}) = \underbrace{\frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)})}_{\text{Empirical risk of classifier } h} + \lambda R(\mathbf{w})$$

Typical Machine Learning Objective Function

$$\Phi(\mathbf{w}) = \underbrace{\frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)})}_{\text{Empirical risk of classifier } h: \mathbb{E}_{\mathcal{D}}[\ell(h(\mathbf{x}; \mathbf{w}), y)]} + \lambda R(\mathbf{w})$$

Typical Machine Learning Objective Function

$$\Phi(\mathbf{w}) = \underbrace{\frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)})}_{\text{Empirical risk of classifier } h} + \lambda \underbrace{R(\mathbf{w})}_{\text{Regularizer}}$$

Typical Machine Learning Objective Function

$$\Phi(\mathbf{w}) = \underbrace{\frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)})}_{\text{Empirical risk of classifier } h} + \underbrace{\lambda R(\mathbf{w})}_{\substack{\text{Constant} \\ \text{Regularizer}}}$$

Regularization

- If we purely optimize the performance on the training dataset, we **overfit**

Regularization

- If we purely optimize the performance on the training dataset, we **overfit**
- Regularizer enables us to determine the trade-off between model complexity and training performance

Regularization

- If we purely optimize the performance on the training dataset, we **overfit**
- Regularizer enables us to determine the trade-off between model complexity and training performance
- Regularizers in the following form are commonly used

$$R(\mathbf{w}) = \|\mathbf{w}\|^2$$

Regularization

- ℓ_1 regularization uses L1 norm of the weights

$$\|\mathbf{w}\|_1$$

- ℓ_2 regularization uses L2 norm of the weights

$$\|\mathbf{w}\|_2^2$$

Optimizing Perceptron

Up to now, we have used the following loss function

$$\ell_s(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & h(\mathbf{x}^{(i)}; \mathbf{w}) = y^{(i)} \\ 1 & \text{otherwise} \end{cases}$$

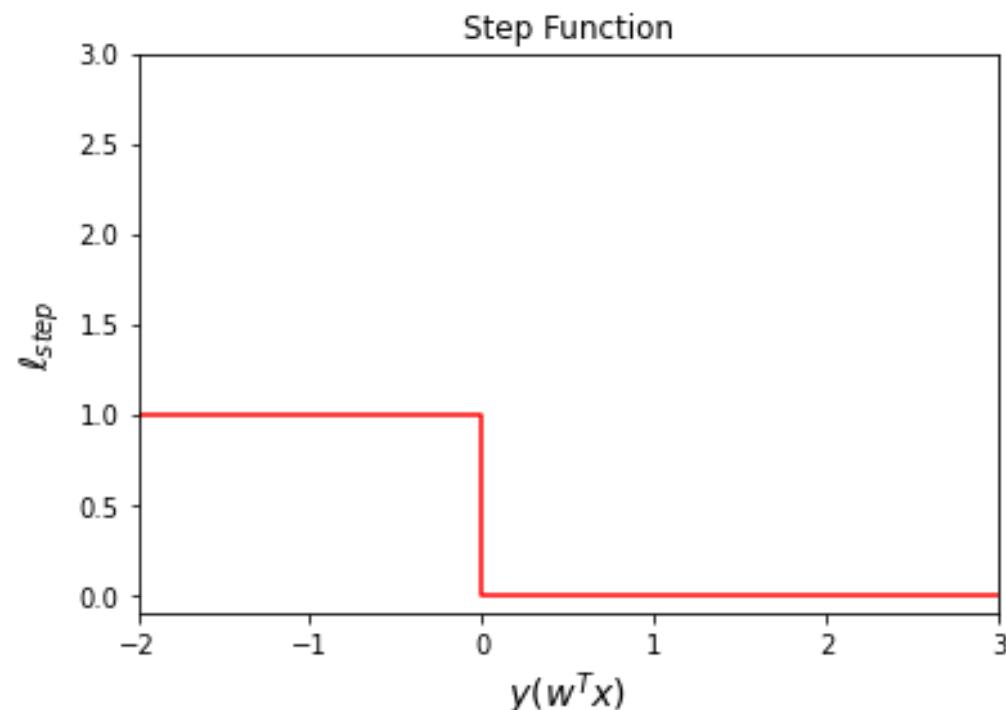
Optimizing Perceptron

Up to now, we have used the following loss function

$$\ell_s(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & h(\mathbf{x}^{(i)}; \mathbf{w}) = y^{(i)} \\ 1 & \text{otherwise} \end{cases}$$

In perceptron this becomes

$$\ell_s(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & y(\mathbf{w}^T \mathbf{x}) > 0 \\ 1 & \text{otherwise} \end{cases}$$



Optimizing Perceptron - Step Function

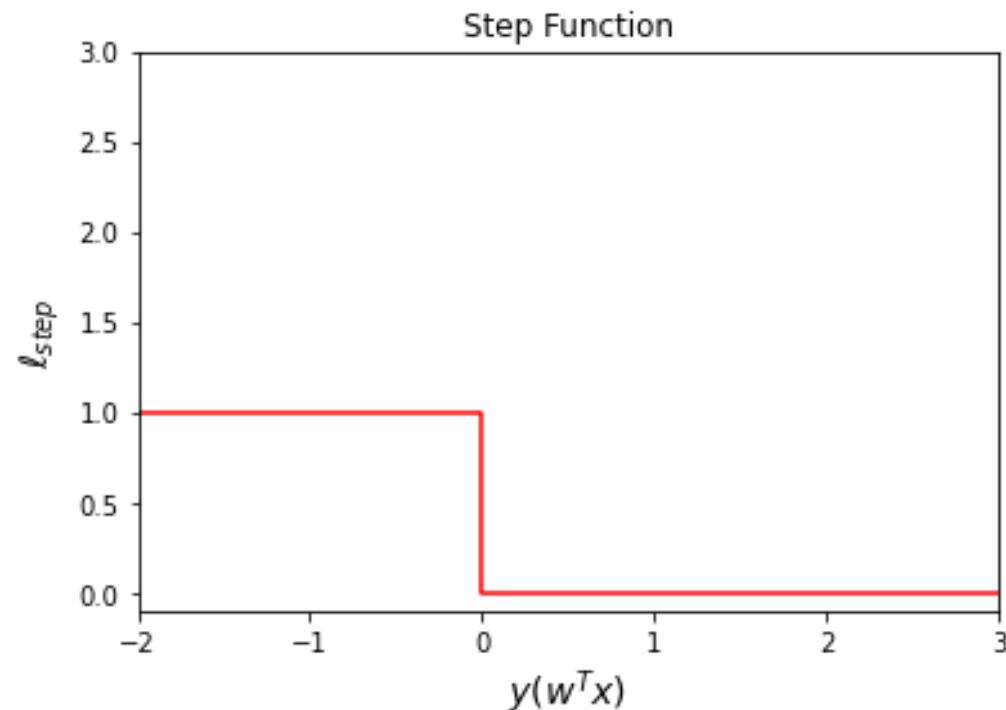
Up to now, we have used the following loss function

$$\ell_s(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & h(\mathbf{x}^{(i)}; \mathbf{w}) = y^{(i)} \\ 1 & \text{otherwise} \end{cases}$$

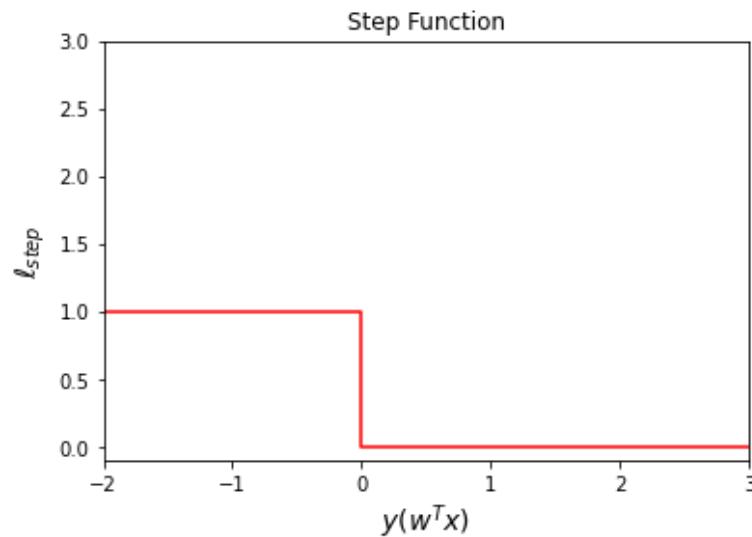
In perceptron this becomes

$$\ell_s(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & y(\mathbf{w}^T \mathbf{x}) > 0 \\ 1 & \text{otherwise} \end{cases}$$

This is called the *step function* and it is *hard* to optimize

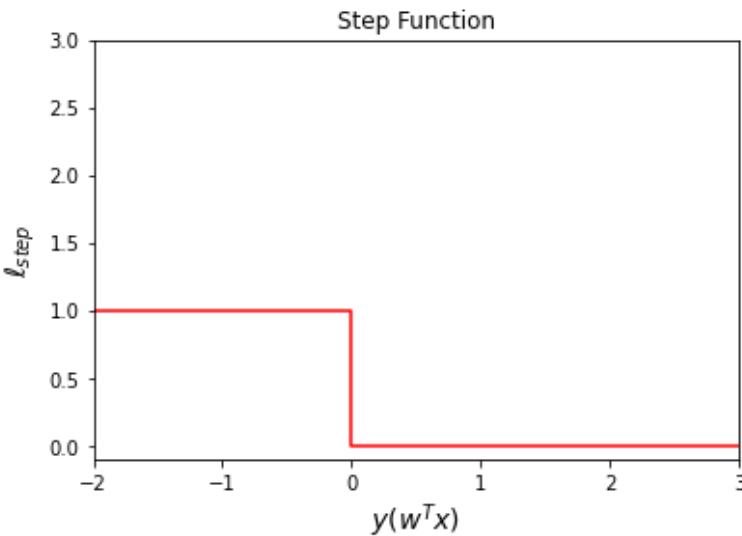


Why *Step Function* is hard to optimize?



Why *Step Function* is hard to optimize?

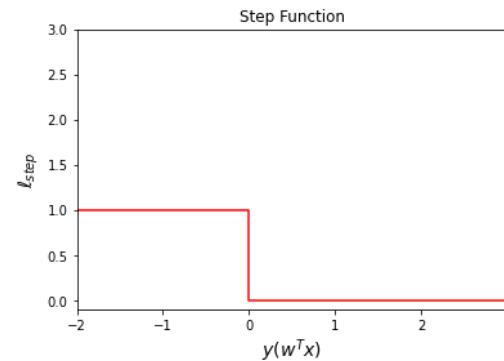
Because it is not a "smooth" function



Why *Step Function* is hard to optimize?

Because it is not a "*smooth*" function

- There can be two classifiers \mathbf{w} and \mathbf{w}' . One can be closer in parameter space to the optimal values \mathbf{w}^* , but they can make the same number of misclassifications, so they have the same loss
- Classifier can't express a degree of certainty about whether a particular input \mathbf{w} should have an associated value y .

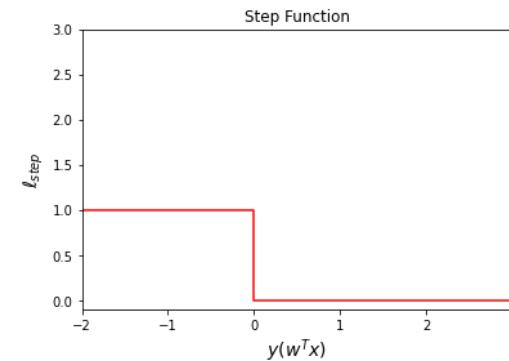


Why *Step Function* is hard to optimize?

Because it is not a "*smooth*" function

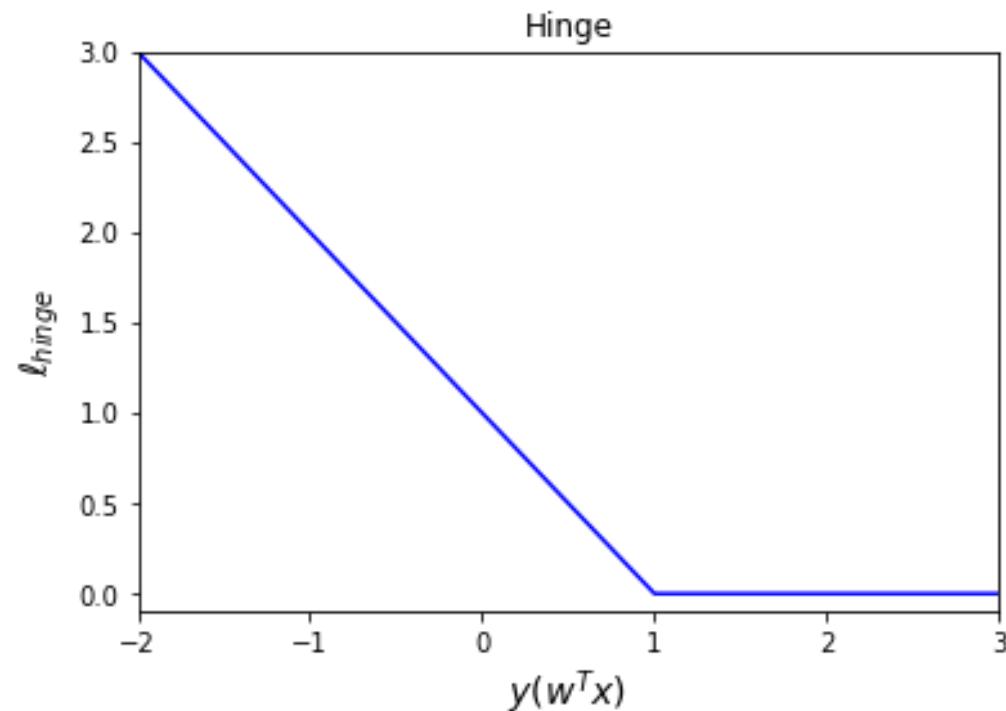
- There can be two classifiers \mathbf{w} and \mathbf{w}' . One can be closer in parameter space to the optimal values \mathbf{w}^* , but they can make the same number of misclassifications, so they have the same loss
- Classifier can't express a degree of certainty about whether a particular input \mathbf{w} should have an associated value y .

Can we use a smooth function that is easier to optimize instead of the step function?



Hinge Loss Function

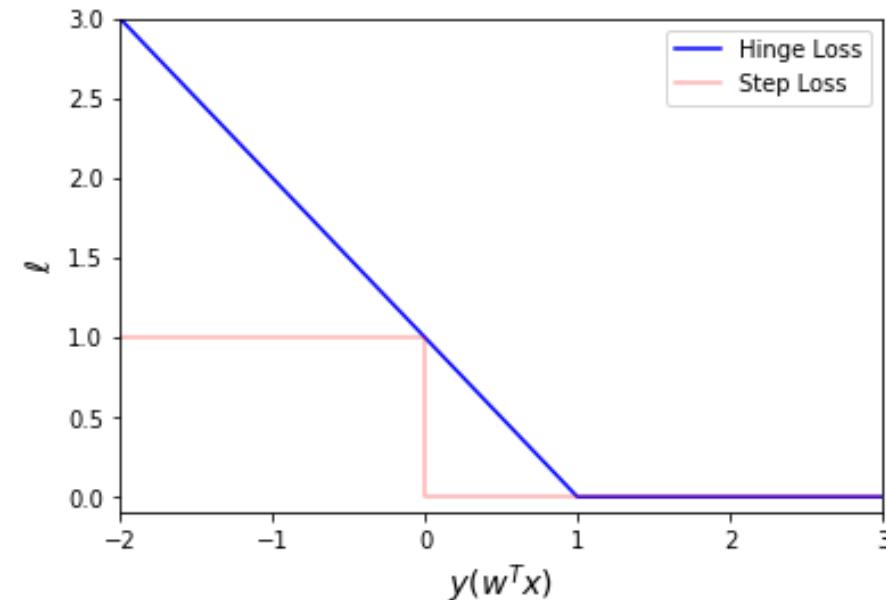
$$\ell_h(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) > 1 \\ 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) & \text{otherwise} \end{cases}$$



Hinge Loss Function

$$\ell_h(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) > 1 \\ 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) & \text{otherwise} \end{cases}$$

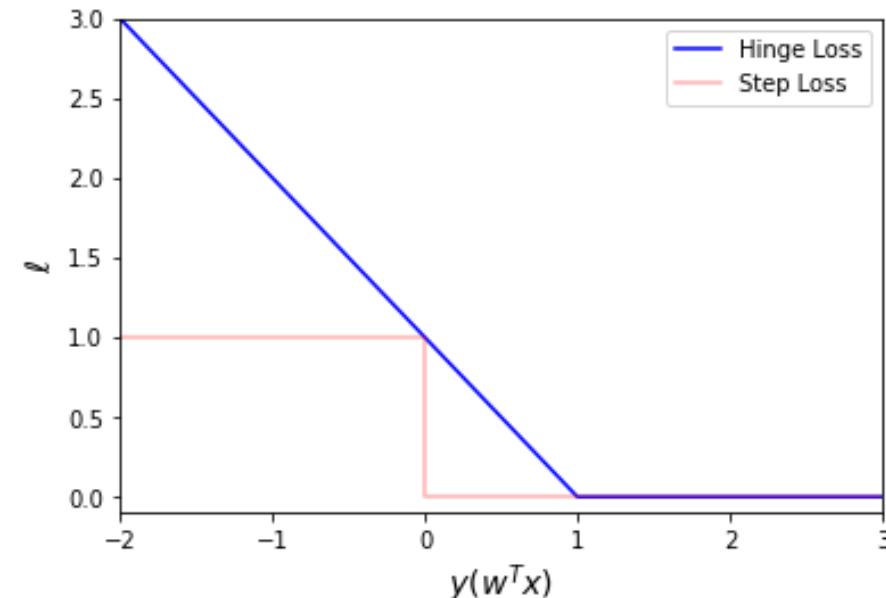
- Hinge loss is zero if $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) > 1$



Hinge Loss Function

$$\ell_h(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) > 1 \\ 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) & \text{otherwise} \end{cases}$$

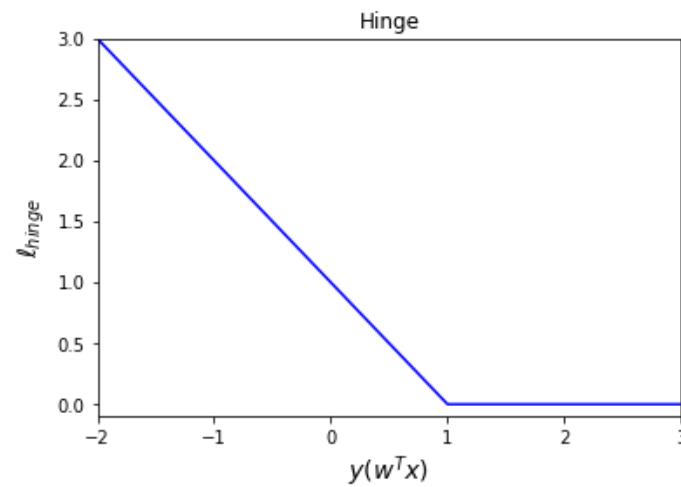
- Hinge loss penalizes incorrect predictions, penalization increase as the difference between the true value and prediction increase



Hinge Loss Function can also be written as

$$\max(1 - y(\mathbf{w}^T \mathbf{x}), 0)$$

$$\begin{aligned}\ell_h(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) &= \begin{cases} 0 & y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) > 1 \\ 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) & \text{otherwise} \end{cases} \\ &= \max(1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}), 0)\end{aligned}$$



Support Vector Machine (SVM)

- SVM is empirical risk minimization with **hinge loss** and ℓ_2 regularization

$$\text{minimize}_{\mathbf{w}} \Phi(\mathbf{w}) = \frac{1}{n} \sum_i^n \max(1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}), 0) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

<small>Note: $\frac{1}{2}$ in $\frac{\lambda}{2}$ is for convenience when we later optimize this. When we take the derivative of $\|\mathbf{w}\|^2$ this will cancel out</small>

Logistic Classifiers (Logistic Regression)

- Logistic classifiers generate real-valued outputs in the interval of $[0, 1]$
- They are also parameterized by a vector of d components

$$\mathbf{w} \in \mathbb{R}^d$$

Logistic Classifiers (Logistic Regression)

- Logistic classifiers generate real-valued outputs in the interval of $[0, 1]$
- They are also parameterized by a d -dimensional vector

$$\mathbf{w} \in \mathbb{R}^d$$

- It is in the form

$$h(\mathbf{x}; \mathbf{w}) = S(\mathbf{w}^T \mathbf{x})$$

Logistic Classifiers (Logistic Regression)

- Logistic classifiers generate real-valued outputs in the interval of $[0, 1]$
- They are also parameterized by a d -dimensional vector

$$\mathbf{w} \in \mathbb{R}^d$$

- It is in the form

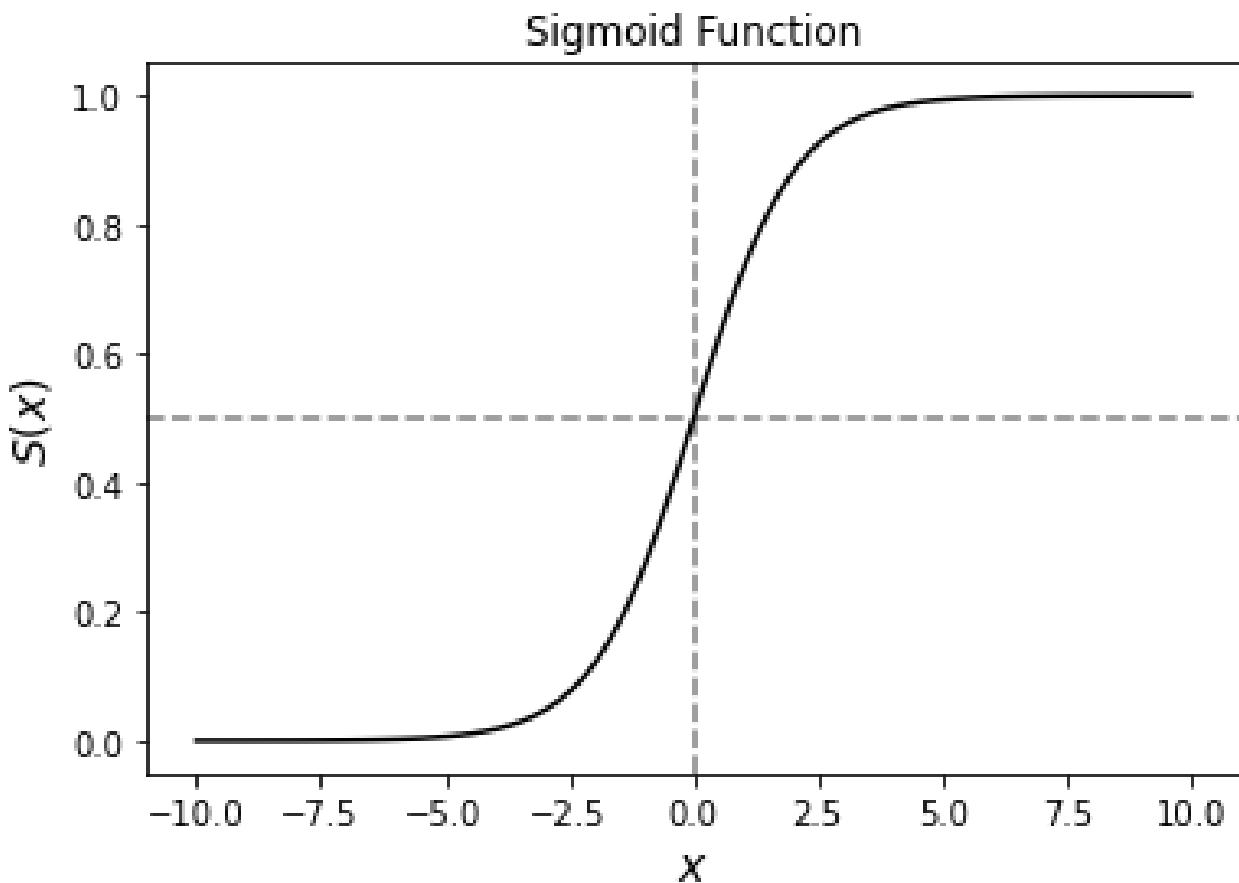
$$h(\mathbf{x}; \mathbf{w}) = S(\mathbf{w}^T \mathbf{x})$$

- S is the sigmoid function (logistic function)

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

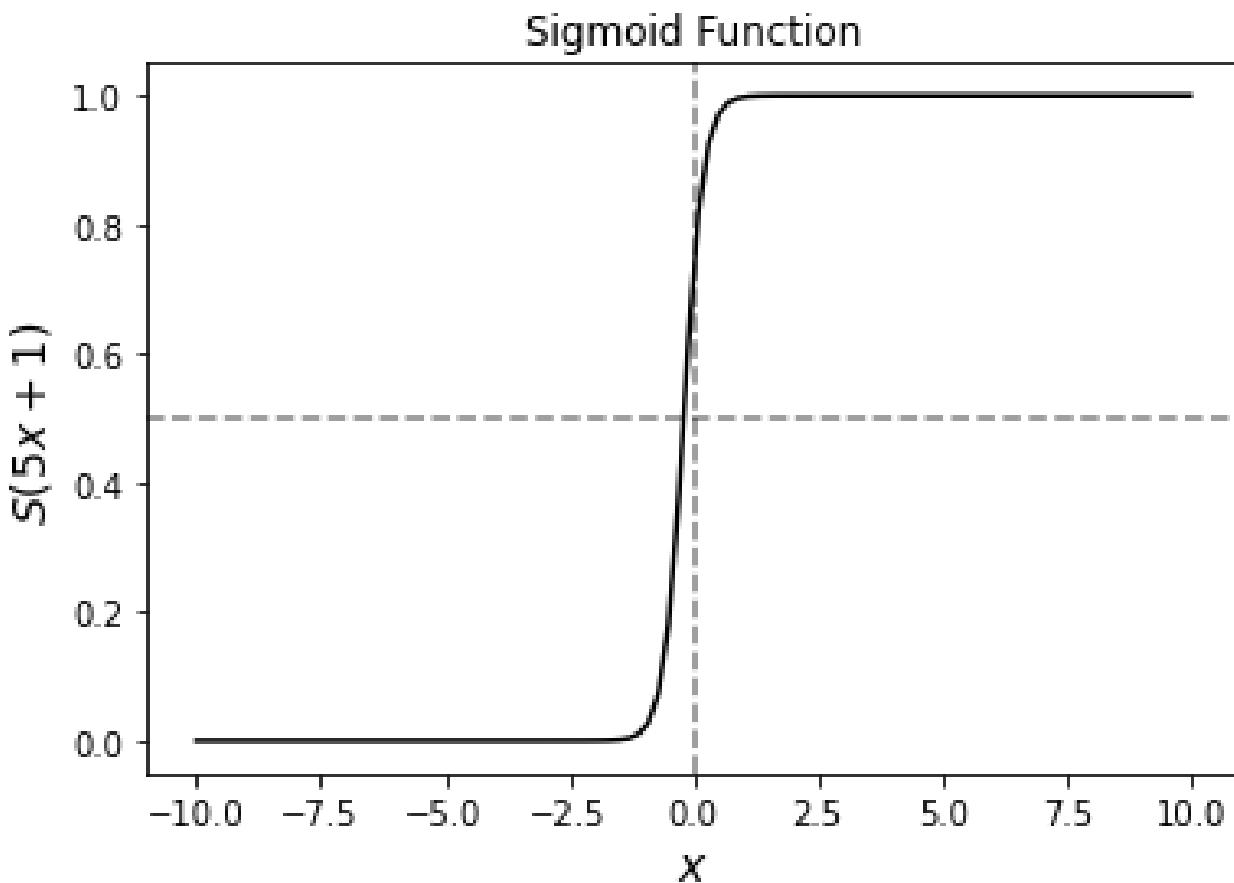
Sigmoid Function

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



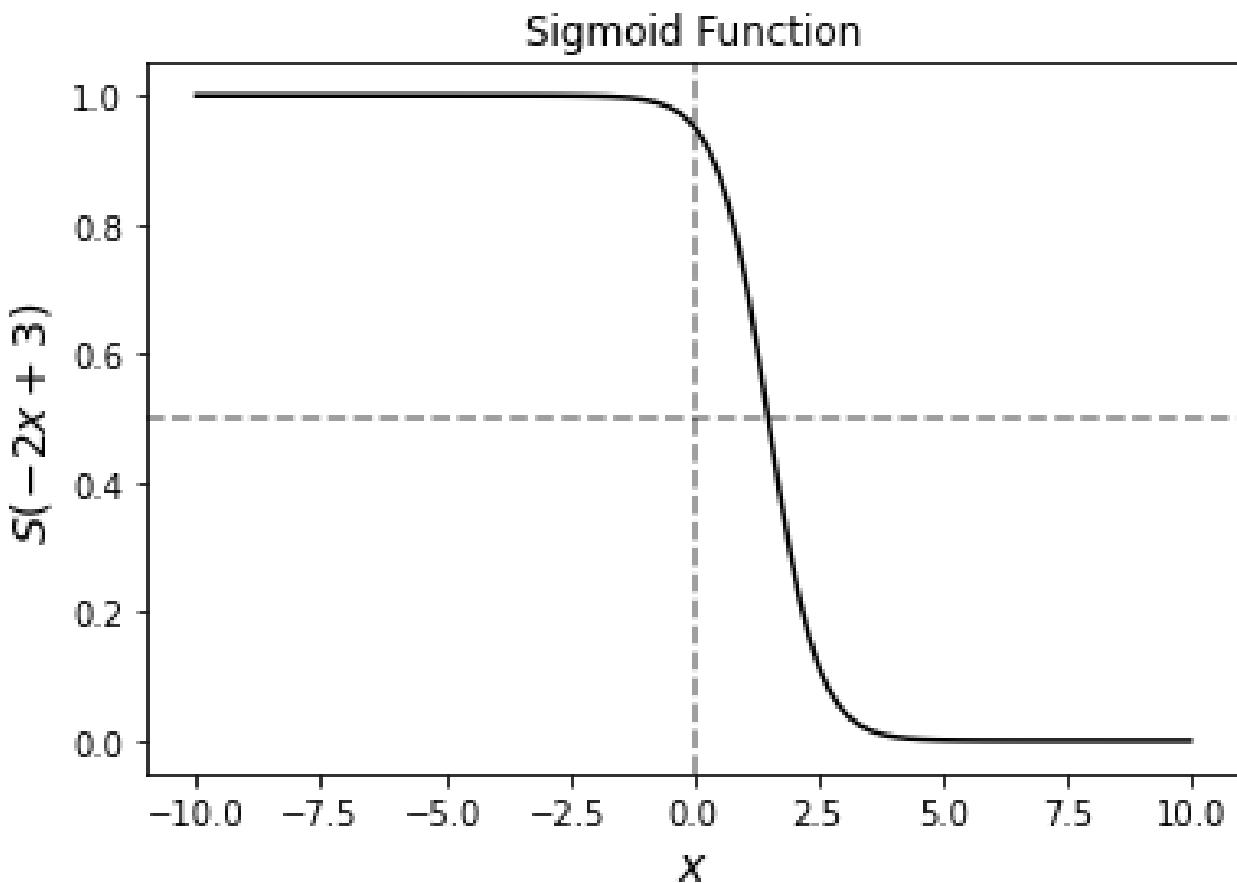
Different 1-D Logistic Classifier Examples

$$S(5x + 1) = \frac{1}{1 + e^{-(5x+1)}}$$



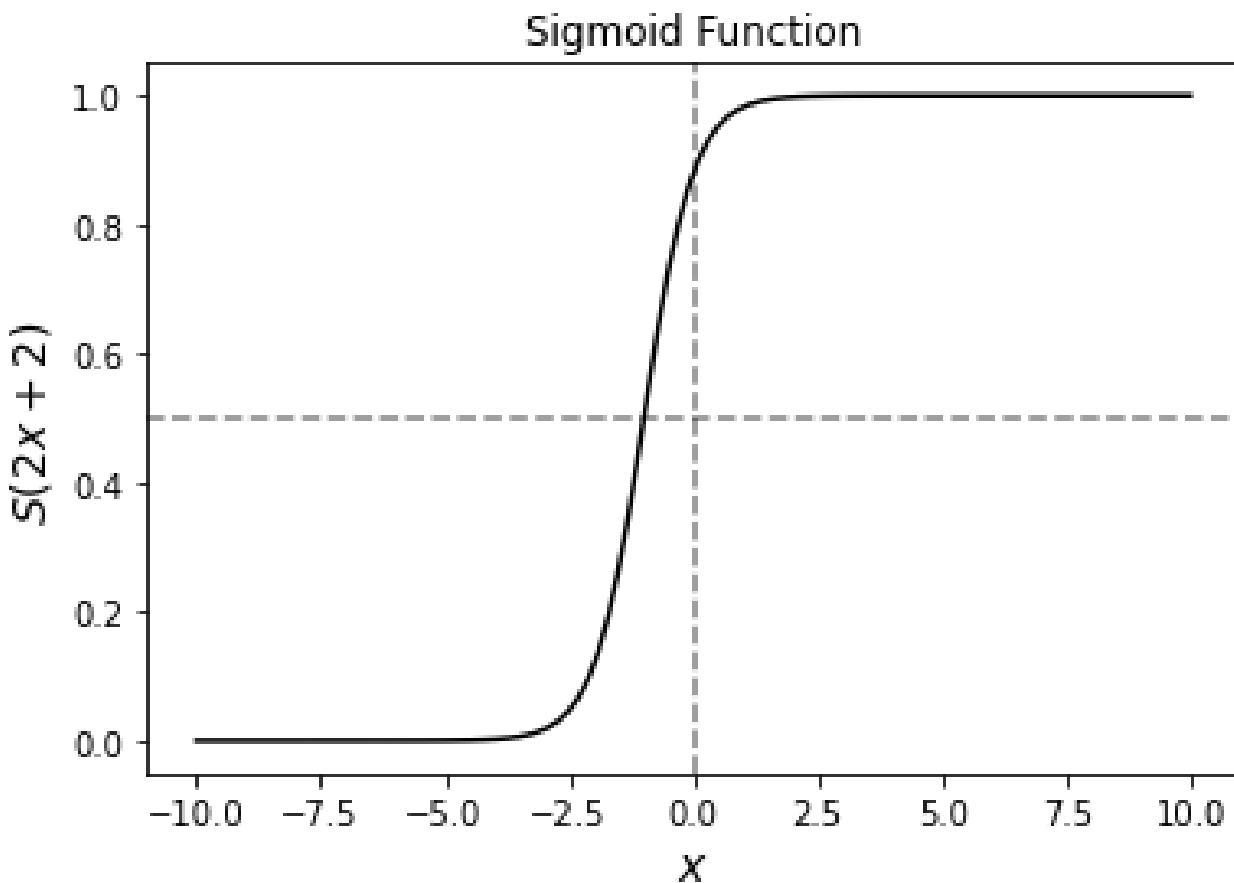
Different 1-D Logistic Classifier Examples

$$S(-2x + 3) = \frac{1}{1 + e^{-(2x+3)}}$$



Different 1-D Logistic Classifier Examples

$$S(2x + 2) = \frac{1}{1 + e^{-(2x+2)}}$$



Classification with Logistic Classifiers

Classifiers definition is

$$\mathbb{R}^d \rightarrow \{-1, +1\}$$

Logistic classifier return a number between 0 and 1 i.e.

$$\mathbb{R}^d \rightarrow [0, 1]$$

Classification with Logistic Classifiers

Classifiers definition is

$$\mathbb{R}^d \rightarrow \{-1, +1\}$$

Logistic classifier return a number between 0 and 1 i.e.

$$\mathbb{R}^d \rightarrow [0, 1]$$

This number $[0, 1]$ is interpreted as the probability of belonging to $+1$ class.

Classification with Logistic Classifiers

If you want to make a classification you need to pick a prediction threshold for probability of belonging to $+1$ class.

- Choosing a threshold considering cost of false positive and false negative (Bayesian Decision Theory)

Classification with Logistic Classifiers

If you want to make a classification you need to pick a prediction threshold for probability of belonging to $+1$ class.

- e.g. using a threshold of 0.5

$$h(\mathbf{x}; \mathbf{w}) = \begin{cases} +1 & S(\mathbf{w}^T \mathbf{x}) > 0.5 \\ -1 & \text{otherwise} \end{cases}$$

Loss Function for Logistic Classifier

Loss Function for Logistic Classifier

IMPORTANT:

Up to now we have used the labels

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

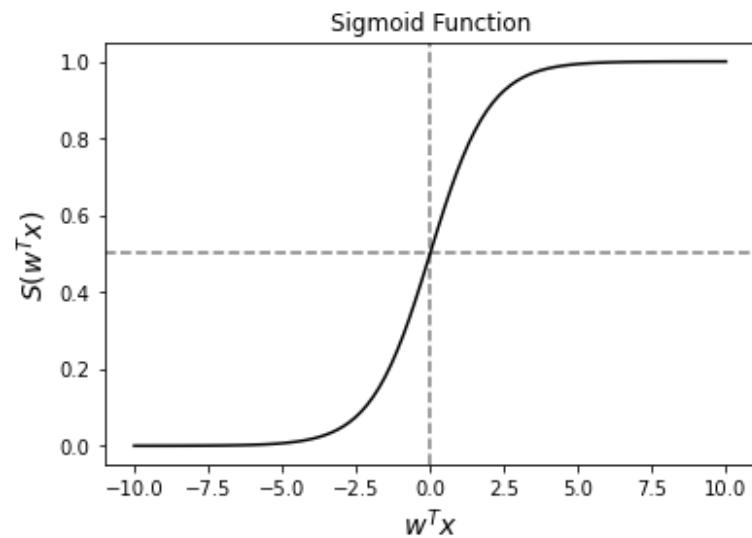
but for logistic loss function we will use the labels

$$y \in \{0, 1\}$$

<small>e.g. banana = 0, orange = 1</small>

Loss Function for Logistic Classifier

- Let guess $g^{(i)} = S(\mathbf{w}^T \mathbf{x}^{(i)})$
- This is interpreted as probability of belonging to class +1.



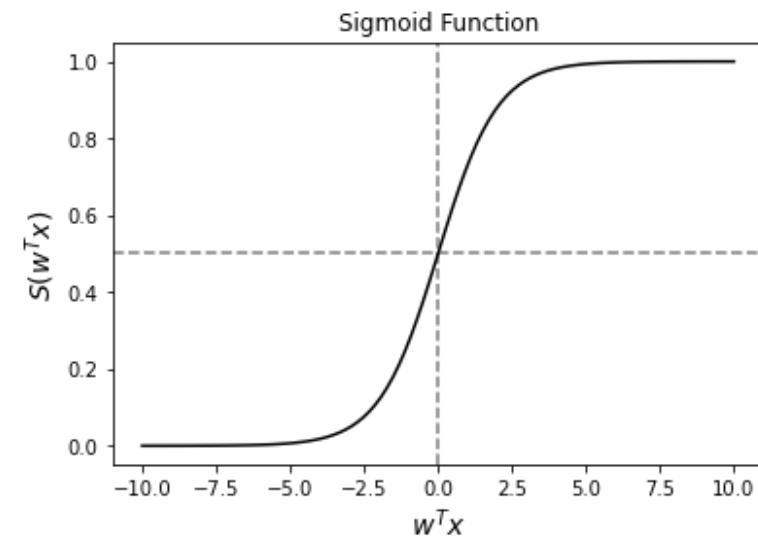
Loss Function for Logistic Classifier

- Let guess $g^{(i)} = S(\mathbf{w}^T \mathbf{x}^{(i)})$
- This is interpreted as probability of belonging to class +1.
- Dataset

$$\mathcal{D} \in \left\{ (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}) \right\}$$

- The probability predictions for the whole data are

$$\prod_{i=1}^n \begin{cases} g^{(i)} & \text{if } y^{(i)} = 1 \\ 1 - g^{(i)} & \text{otherwise} \end{cases}$$



$$\begin{cases} g^{(i)} & \text{if } y^{(i)} = 1 \\ 1 - g^{(i)} & \text{otherwise} \end{cases}$$

can also be written as

$$g^{(i)^{y^{(i)}}} (1 - g^{(i)})^{1-y^{(i)}}$$

Note: $y^{(i)} \in \{0, 1\}$

$$\begin{cases} g^{(i)} & \text{if } y^{(i)} = 1 \\ 1 - g^{(i)} & \text{otherwise} \end{cases}$$

can also be written as

$$g^{(i)^{y^{(i)}}} (1 - g^{(i)})^{1-y^{(i)}}$$

Note: $y^{(i)} \in \{0, 1\}$

$$\text{guess}^{\text{actual}} \cdot (1 - \text{guess})^{1-\text{actual}}$$

Likelihood

$$y^{(i)} \in \{0, 1\}$$

We would like to assign a high probability for the correct class and low probability for the incorrect class.

$$g^{(i)^{y^{(i)}}} (1 - g^{(i)})^{1-y^{(i)}}$$

Likelihood

$$y^{(i)} \in \{0, 1\}$$

We would like to assign a high probability for the correct class and low probability for the incorrect class.

$$g^{(i)^{y^{(i)}}} (1 - g^{(i)})^{1-y^{(i)}}$$

$$\text{guess}^{\text{actual}} \cdot (1 - \text{guess})^{1-\text{actual}}$$

Likelihood

$$y^{(i)} \in \{0, 1\}$$

We would like to assign a high probability for the correct class and low probability for the incorrect class.

$$g^{(i)^{y^{(i)}}} (1 - g^{(i)})^{1-y^{(i)}}$$

Gets high values when:

- $y^{(i)} = 1$ and $g^{(i)}$ is large or
- $y^{(i)} = 0$ and $g^{(i)}$ is small

Likelihood

- So we want to **maximize**: it for the whole training data

$$\prod_{i=1}^n g^{(i)y^{(i)}} (1 - g)^{(i)1-y^{(i)}}$$

Log-likelihood

- We often deal with log-probabilities rather than probabilities as they are more numerically stable, so we want to **maximize** logarithm of the likelihood

$$\log \left(\prod_{i=1}^n g^{(i)y^{(i)}} (1 - g)^{(i)1-y^{(i)}} \right)$$

Log-likelihood

- We often deal with log-probabilities rather than probabilities as they are more numerically stable, so we want to **maximize** logarithm of the likelihood

$$\log \left(\prod_{i=1}^n g^{(i)y^{(i)}} (1 - g)^{(i)1-y^{(i)}} \right)$$

$$= \sum_{i=1}^n \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

Log-Likelihood

$$\sum_{i=1}^n \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

- We have a function that we want to **maximize**.
- Recall that in machine learning we work with loss functions that we want to **minimize**.

Log-Likelihood

$$\sum_{i=1}^n \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

- We have a function that we want to **maximize**.
- Recall that in machine learning we work with loss functions that we want to **minimize**.
- By taking the negative of the function above, we turn it into a **minimization** problem

Negative Log-Likelihood

$$\sum_{i=1}^n \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

- We have a function that we want to **maximize**.
- Recall that in machine learning we work with loss functions that we want to **minimize**.
- By taking the negative of the function above, we turn it into a **minimization** problem

$$\sum_{i=1}^n - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

Negative Log-Likelihood Loss Function ℓ_{nll}

$$\ell_{nll}(g^{(i)}, y^{(i)}) = - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

Negative Log-Likelihood Loss Function ℓ_{nll}

$$\ell_{nll}(g^{(i)}, y^{(i)}) = - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

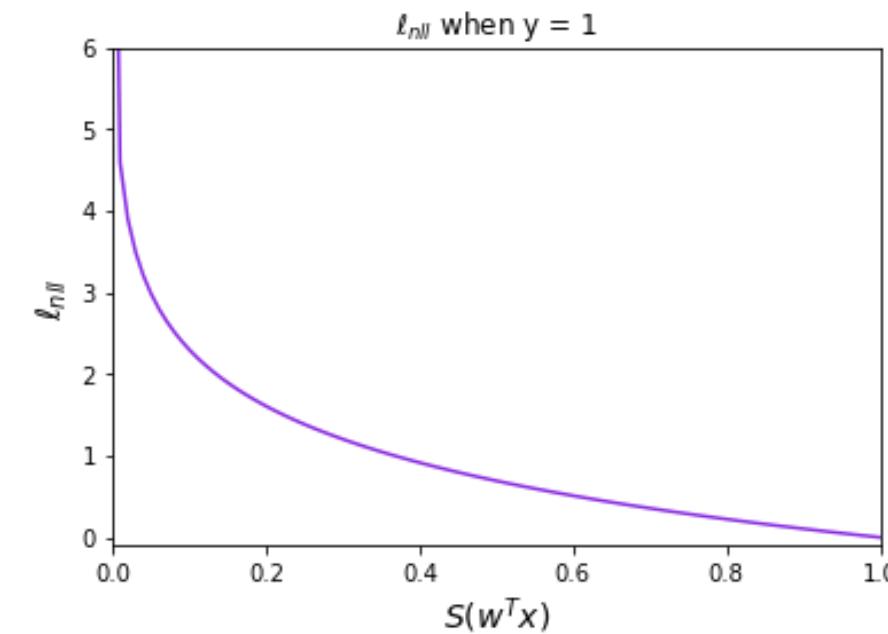
$$\ell_{nll}(\text{guess, actual}) = - (\text{actual} \cdot \log(\text{guess}) + (1 - \text{actual}) \cdot \log(1 - \text{guess}))$$

Negative Log-Likelihood Loss Function

ℓ_{nll}

$$\ell_{nll}(g^{(i)}, y^{(i)}) = - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

$$\ell_{nll}(\text{guess, actual}) = - (\text{actual} \cdot \log(\text{guess}) + (1 - \text{actual}) \cdot \log(1 - \text{guess}))$$



Negative Log-Likelihood Loss Function ℓ_{nll}

$$\ell_{nll}(g^{(i)}, y^{(i)}) = - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

$$\ell_{nll}(\text{guess}, \text{actual}) = - (\text{actual} \cdot \log(\text{guess}) + (1 - \text{actual}) \cdot \log(1 - \text{guess}))$$

- This is negative log-likelihood loss function
 - Also called *log loss* or *cross entropy*
- Extends to cases with more than two outputs
- Optimizing negative log-likelihood is called **logistic regression**

Regularizer for Logistic Regression

To avoid overfitting, we need a regularizer term for the logistic regression

Ridge Regression

If we add a $\ell 2$ regularizer to logistic regression, it's called **ridge regression**

$$\begin{aligned}\Phi(\mathbf{w}) &= \mathbb{E}_{\mathcal{D}}[\ell_{nll}(h(\mathbf{x}; \mathbf{w}), y)] + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \\ &= \frac{1}{n} \sum_{i=1}^n - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2\end{aligned}$$

Regularizer for Logistic Regression

Lasso Regression

If we add a ℓ_1 regularizer to logistic regression, it's called a **lasso regression**

$$\begin{aligned}\Phi(\mathbf{w}) &= \mathbb{E}_{\mathcal{D}}[\ell_{nll}(h(\mathbf{x}; \mathbf{w}), y)] + \lambda \|\mathbf{w}\|_1 \\ &= \frac{1}{n} \sum_{i=1}^n - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right) + \lambda \|\mathbf{w}\|_1\end{aligned}$$

COGS514 - Cognition and Machine Learning

Optimization - Gradient Descent

Optimizing Loss and Regularizer Functions

$$\text{minimize } \Phi(w)$$

Support Vector Machine

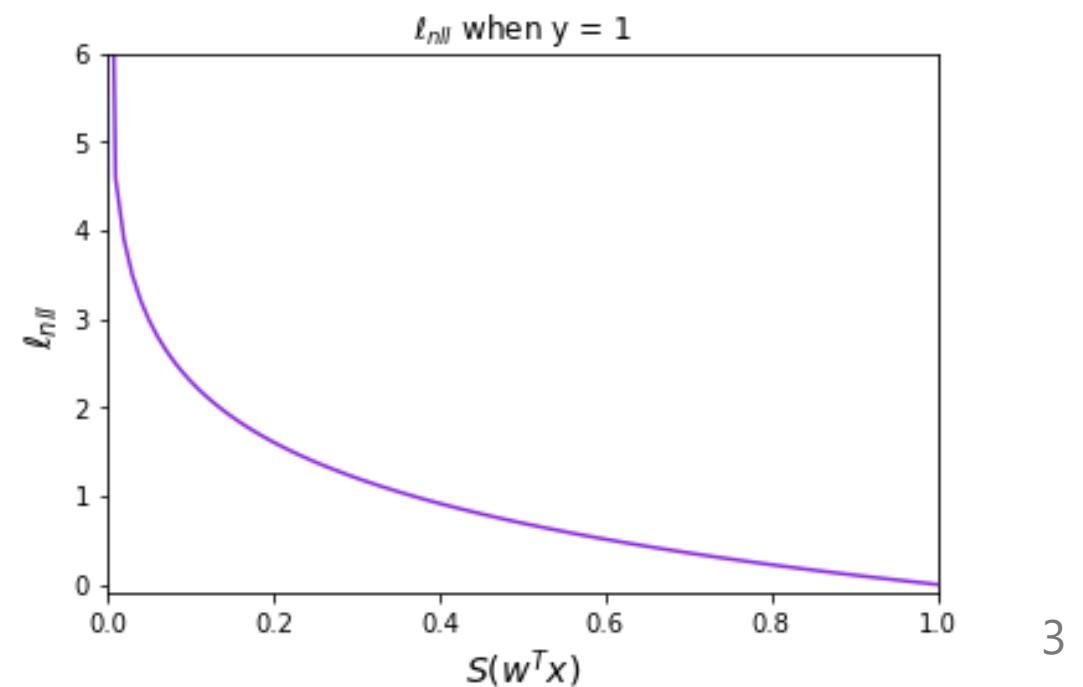
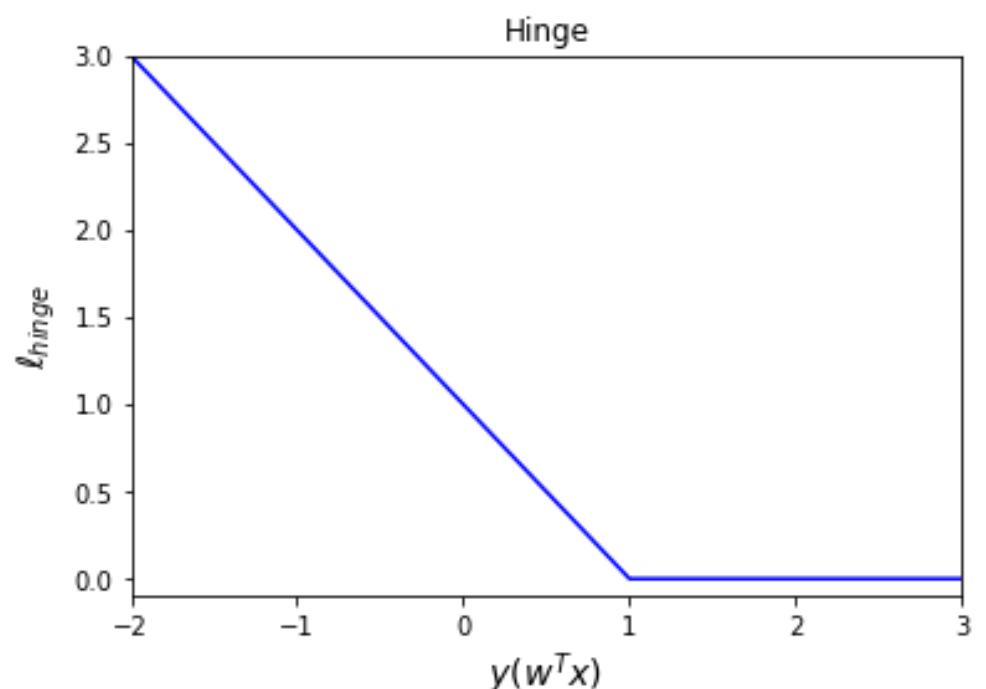
$$\Phi(w) = \sum_{i=1}^n \max(1 - y^{(i)}(w^T x^{(i)}), 0) + \frac{\lambda}{2} \|w\|_2^2$$

Ridge Logistic Regression

$$\Phi(w) = \sum_{i=1}^n \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right) + \frac{\lambda}{2} \|w\|_2^2$$

How to optimize $\Phi(w)$

- A simple way to minimize a *differentiable* function $\Phi(w)$ is to find a direction that the function Φ is decreasing, and change w towards that direction

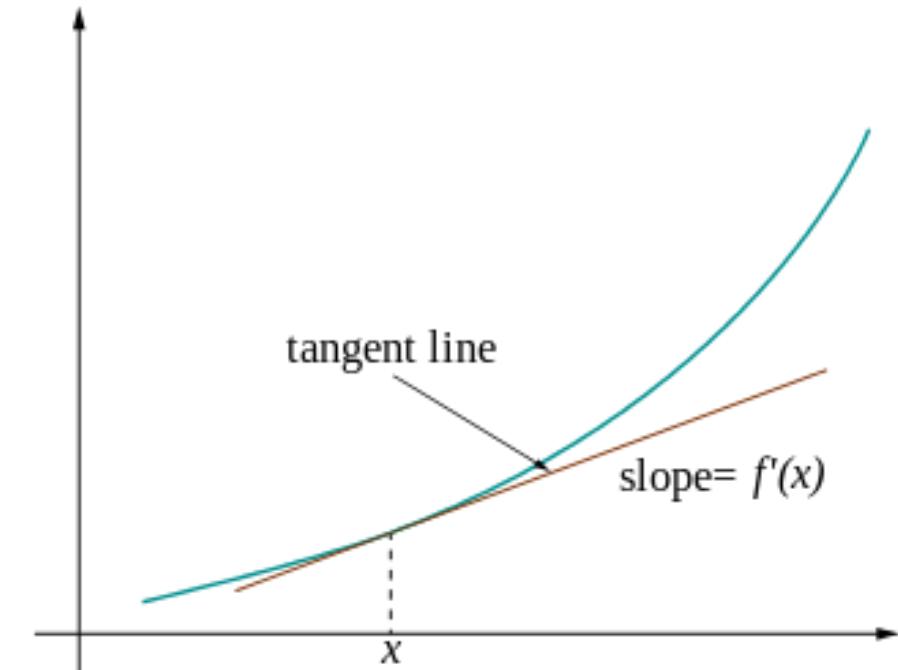


How to find the direction that the function is decreasing?

Derivative

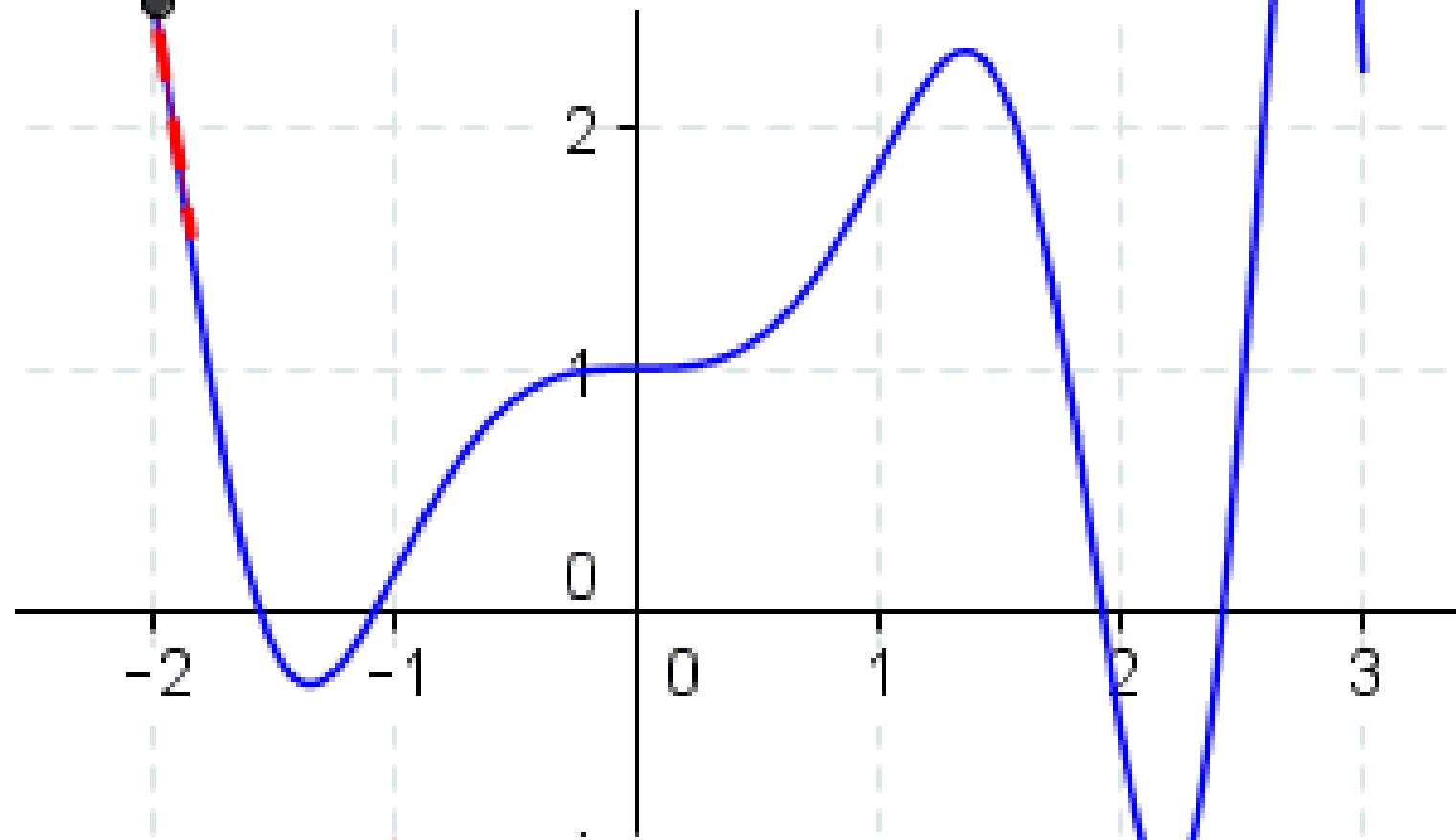
$$f'(x) = \frac{df}{dx}$$

- Derivative of a function is its sensitivity to change w.r.t to its inputs
 - df i.e. tiny change in $f(x)$ with respect dx i.e. to tiny change in x
- $f'(x)$ can be visualised as the slope of a tangent line to the function $f(x)$ at x



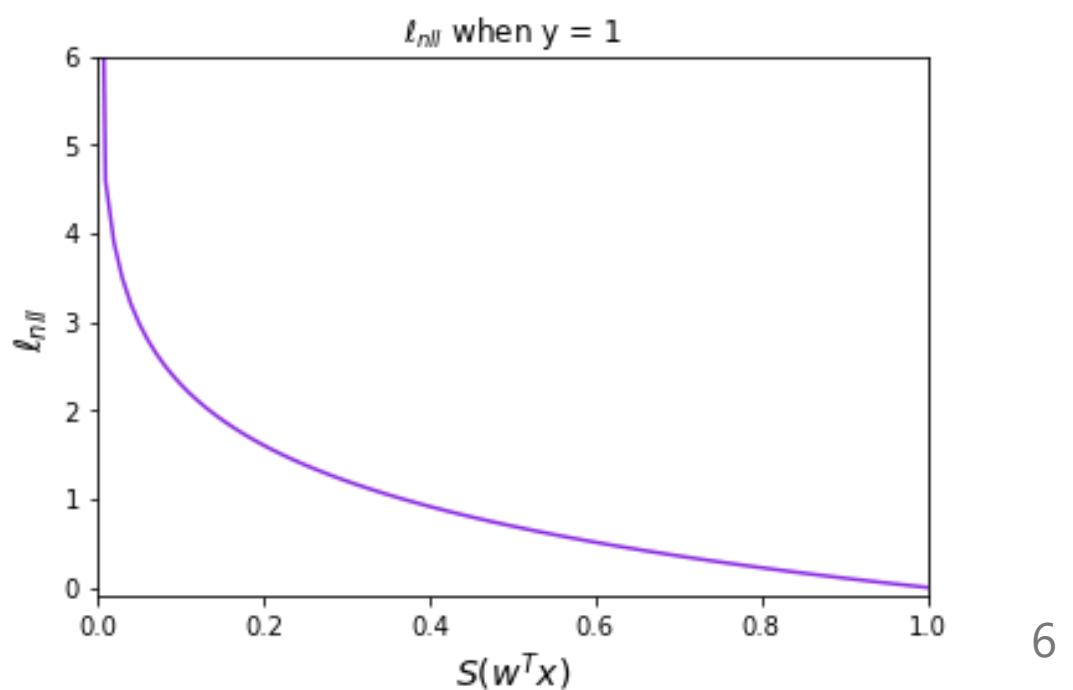
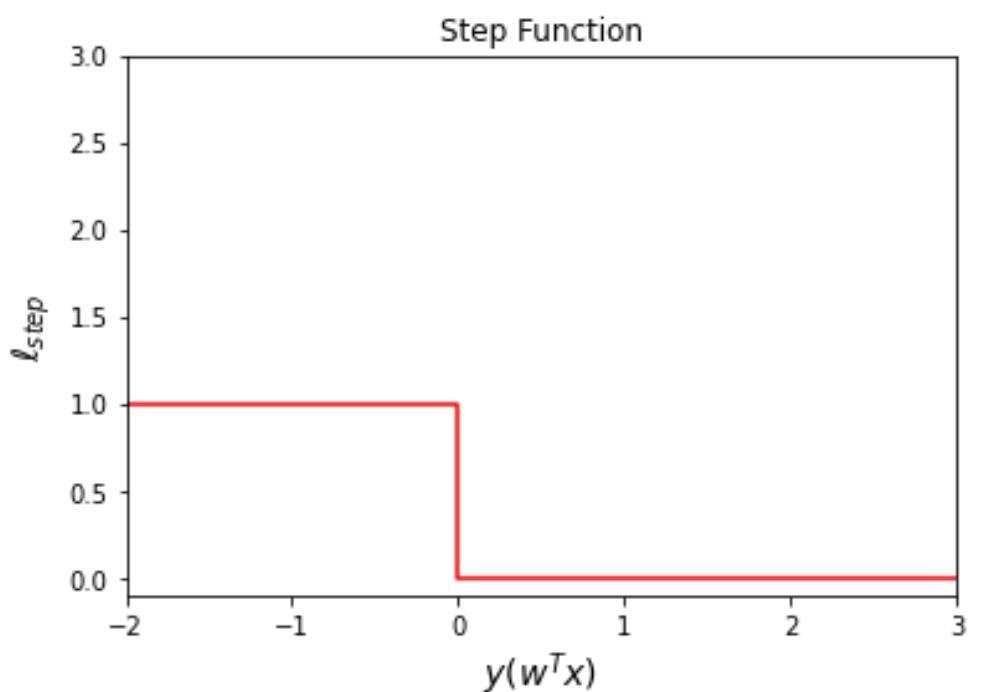
$$f(x) = x \sin(x^2) + 1$$

$$A = (-2, 2.51)$$



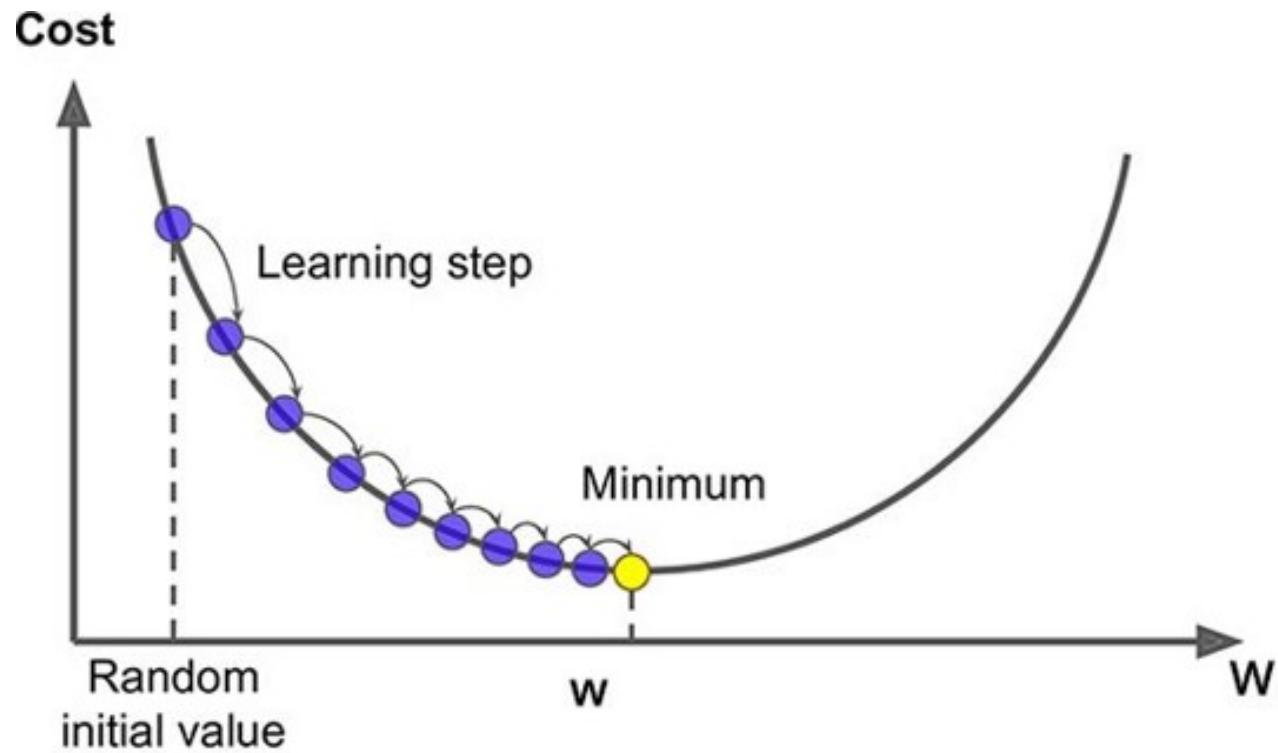
Differentiable (smooth) vs. Non-differentiable functions

- Step loss: non differentiable
- Negative log likelihood
differentiable



Gradient Descent Algorithm

- *Gradient Descent* is an iterative algorithm that moves towards the decreasing direction at each step
- It is guaranteed to find an optimal solution for *convex functions*



Gradient Descent

Start from an initial point $w^{(0)} \in \mathbb{R}^d, t = 0$

while

- $t = t + 1$
- $w^{(t)} = w^{(t-1)} - \alpha f'(w^{(t-1)})$

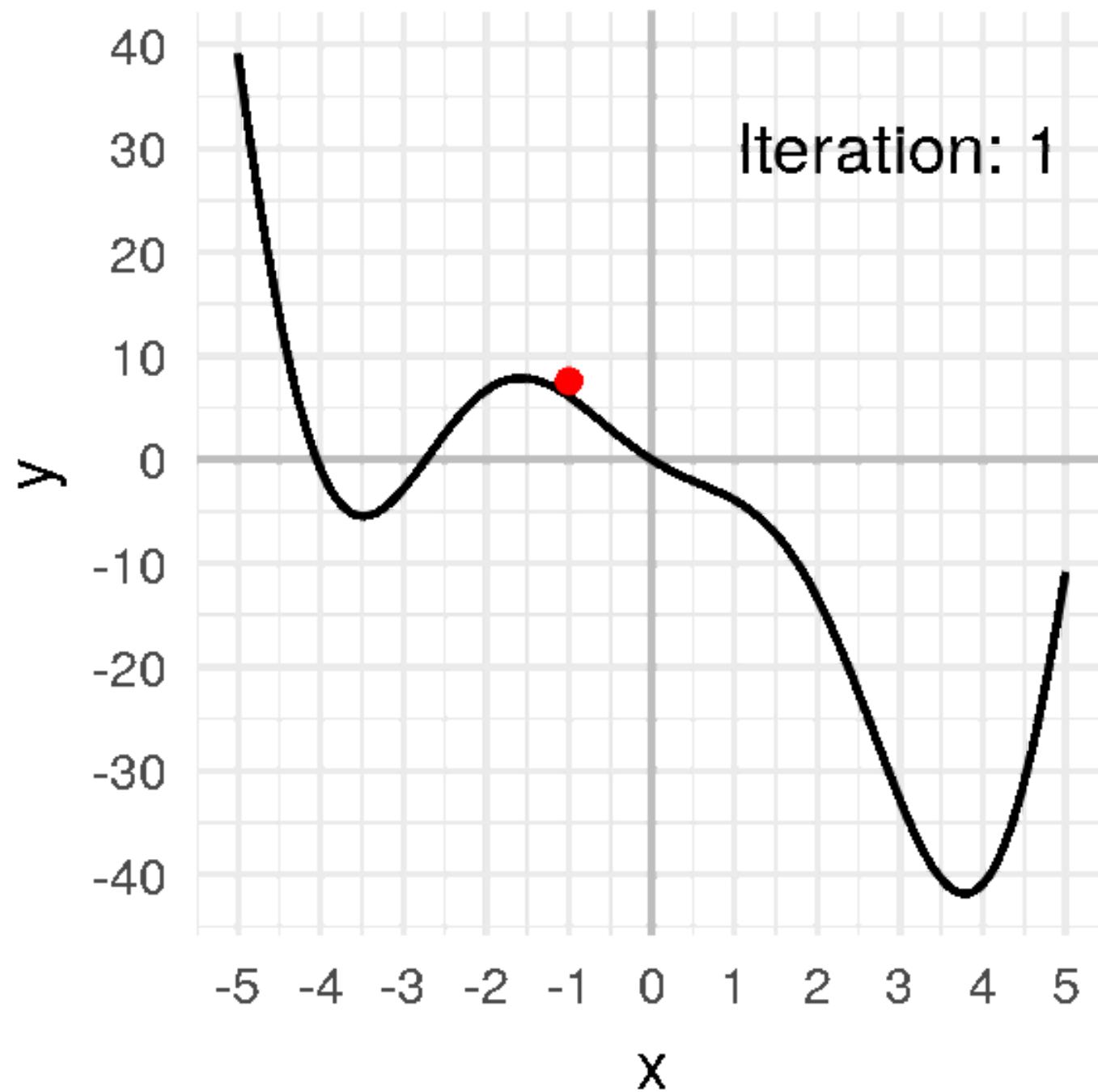
until $|f(w^{(t)}) - f(w^{(t-1)})| < \epsilon$

return $w^{(t)}$

- α is a constant (*step-size parameter*)
- ϵ is a constant (accuracy parameter)

Gradient Descent Stopping rules

1. Stop after a fixed number of iterations (stop when $t = T$)
2. Stop when the change in the value of the parameter is small (stop when $|f(w^{(t)}) - f(w^{(t-1)})| < \epsilon$)
3. Stop when the derivative f' is sufficiently small (stop when $|f'(w^{(t)})| < \epsilon$)



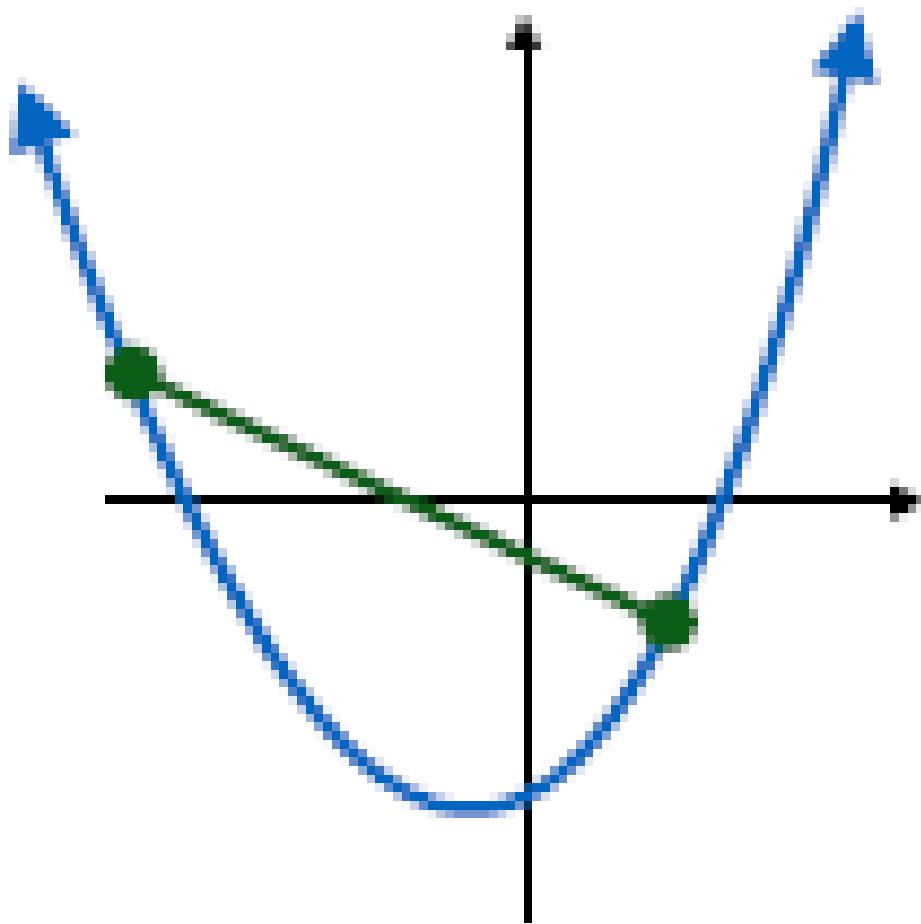
Gradient descent finds an optimal solution in convex functions

If Φ is convex, for any desired accuracy ϵ , there is some step size α that gradient descent will converge to within ϵ of the optimal Φ

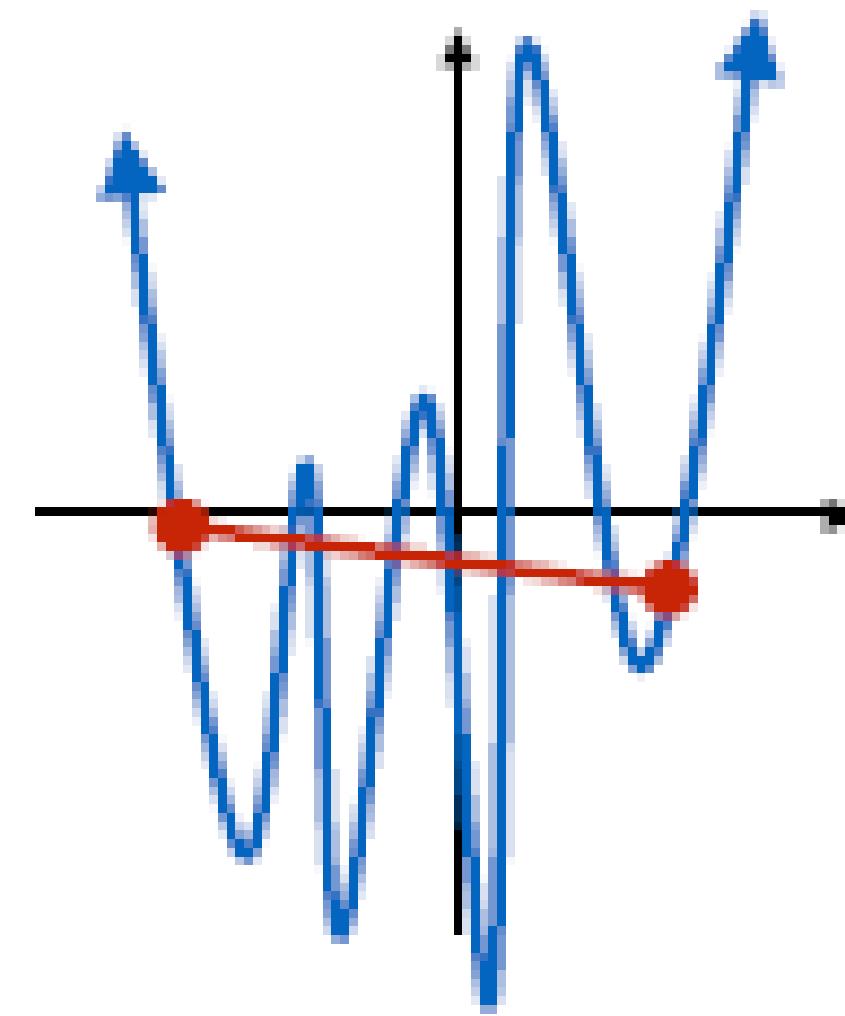
Gradient descent finds an optimal solution in convex functions

If Φ is convex, for any desired accuracy ϵ , there is some step size α that gradient descent will converge to within ϵ of the optimal Φ

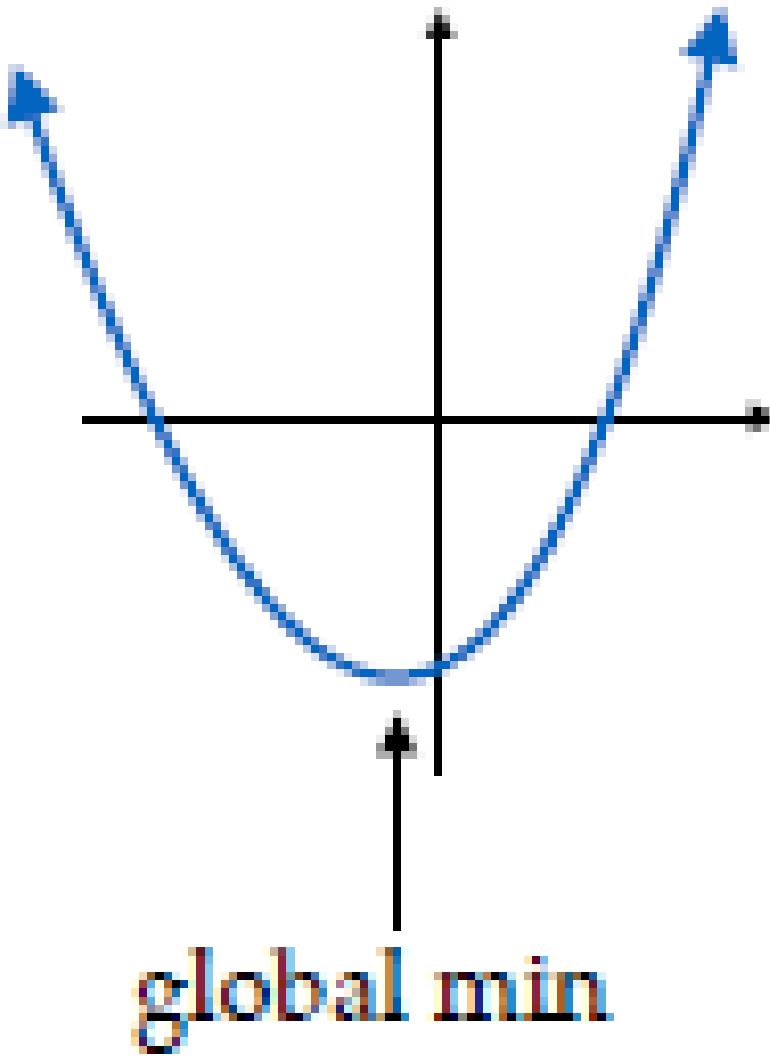
Convex function: A function Φ is convex if for all w_1, w_2 in \mathbb{R}^d and $\alpha \in [0, 1]$,
$$\Phi(\alpha w_1 + (1 - \alpha) w_2) \leq \alpha \Phi(w_1) + (1 - \alpha) \Phi(w_2)$$

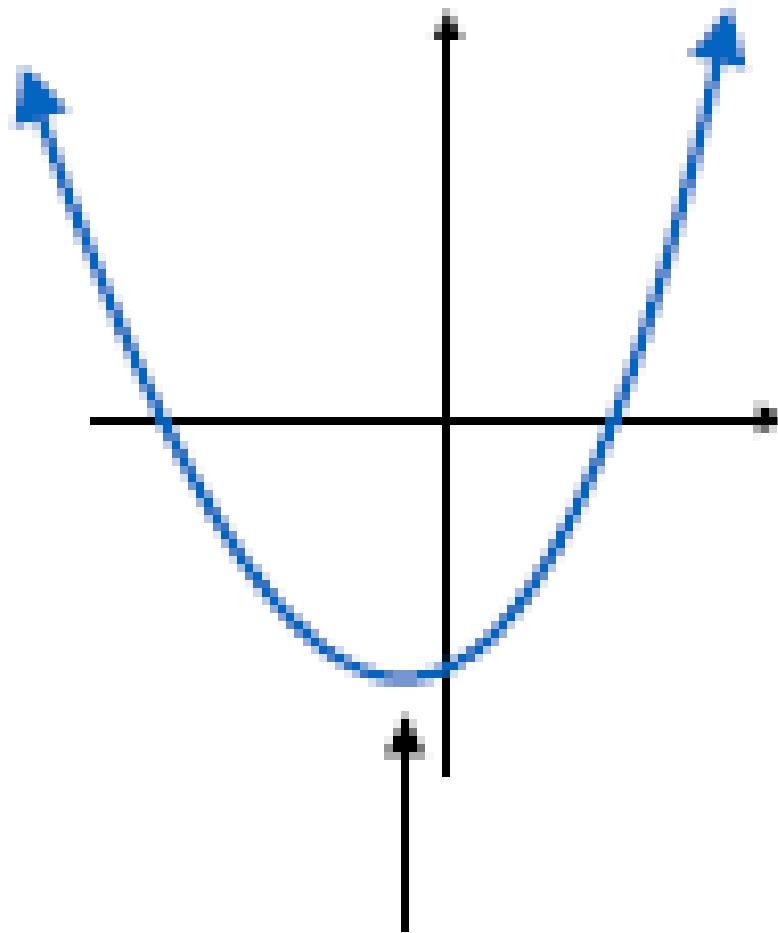


convex

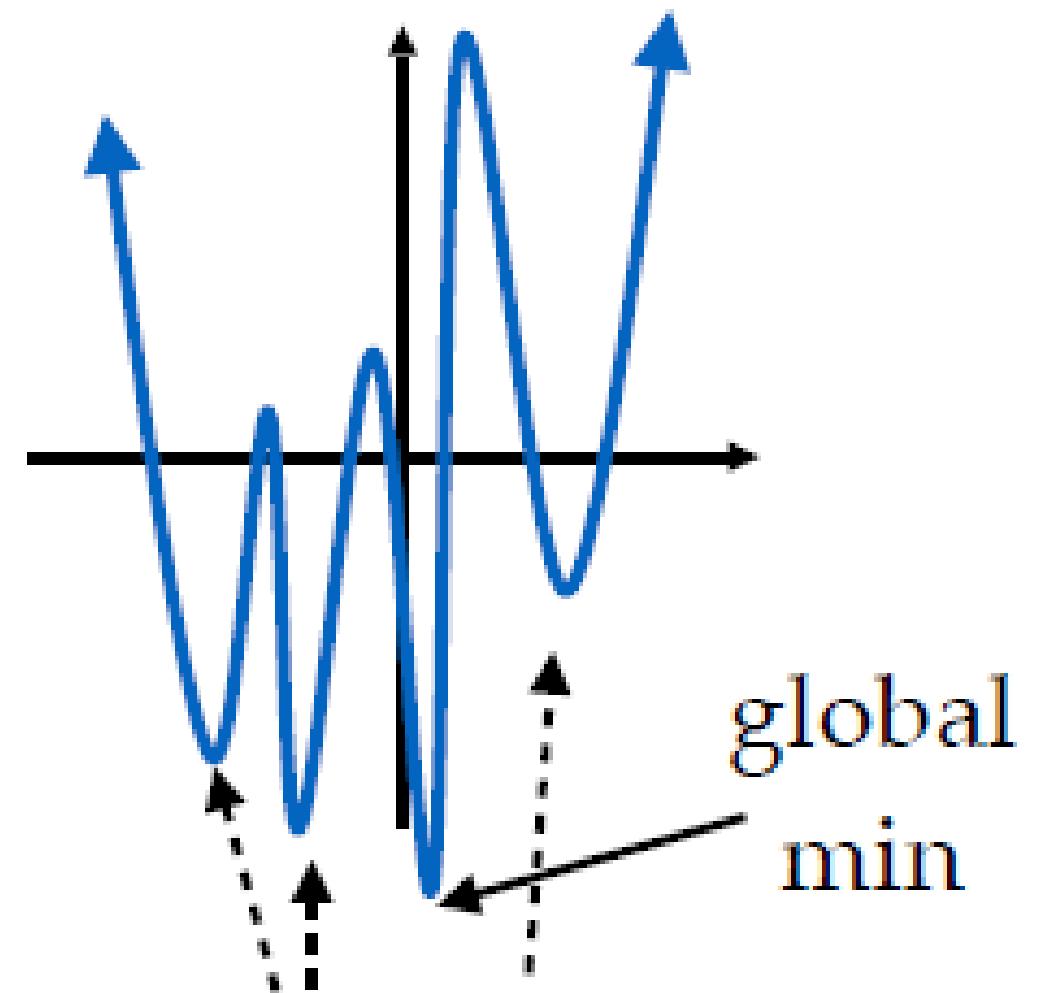


nonconvex





global min

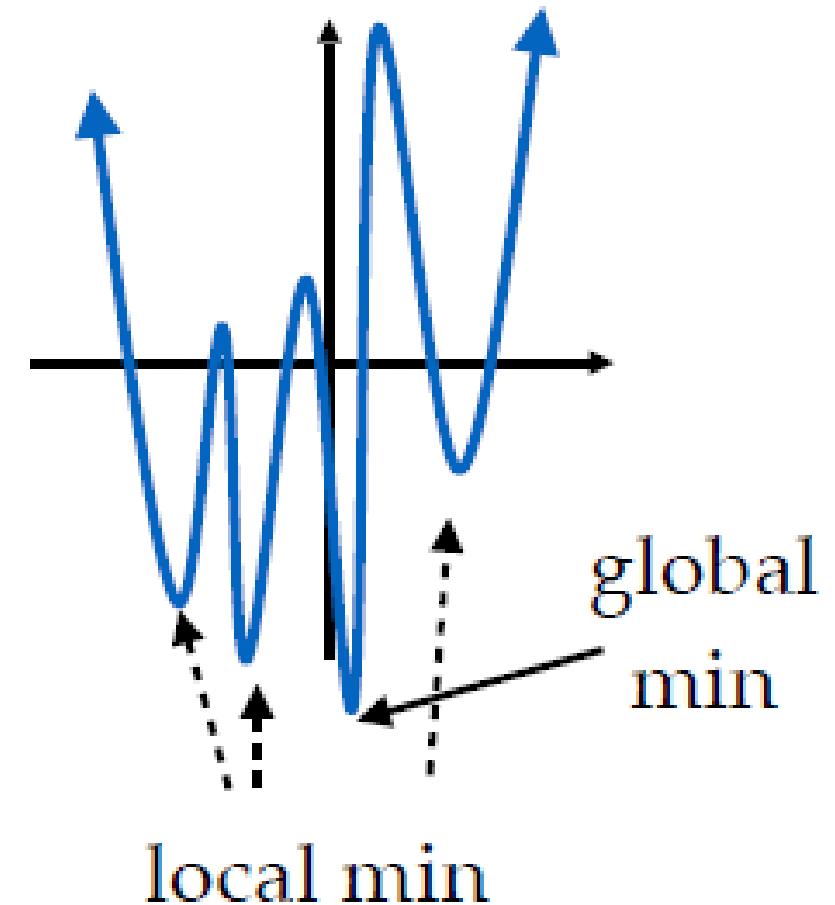


local min

global
min

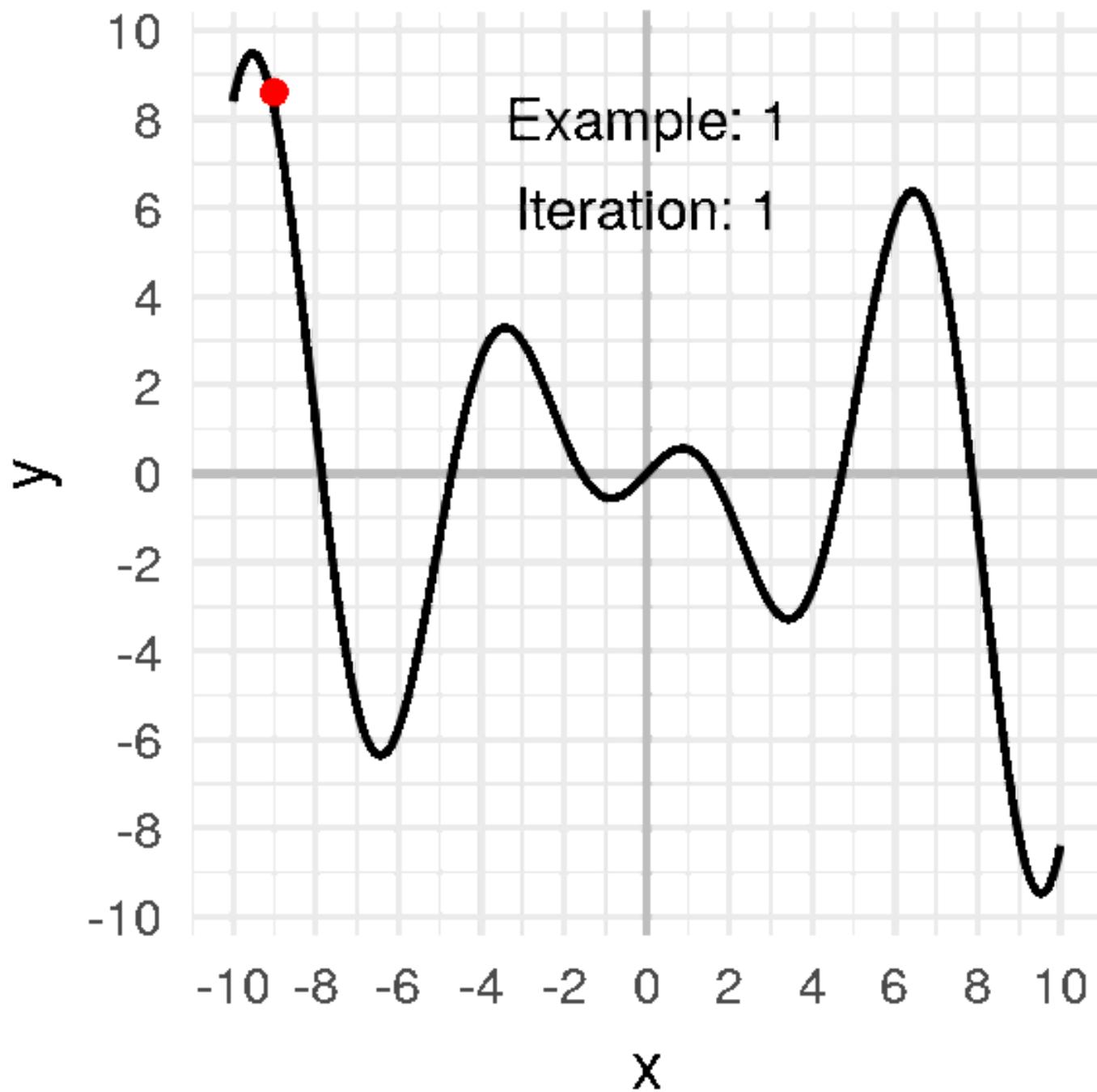
Nonconvex Functions - Local Optima

- In non-convex functions the point that the gradient descent converges depends on the starting point.
 - It may reach a point where $f'(w) = 0$ but is not a minimum of the function. It can be a local minimum

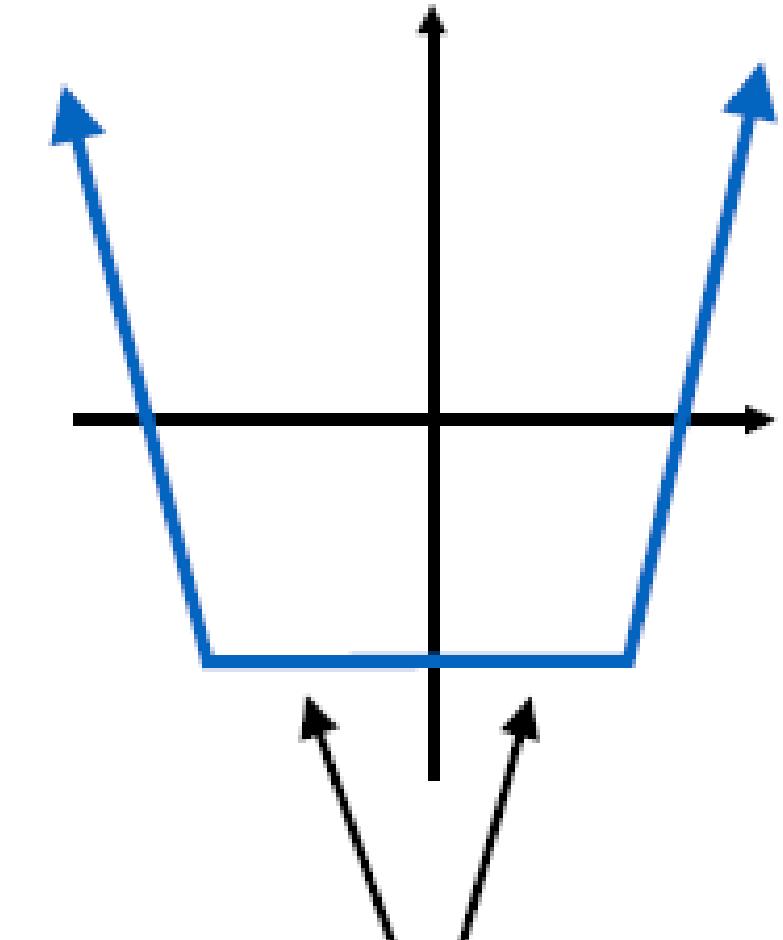


Nonconvex Functions - Local Optima

- In non-convex functions the point that the gradient descent converges depends on the starting point.
 - It may reach a point where $f'(w) = 0$ but is not a minimum of the function. It can be a local minimum
- Points where the gradient vanishes are the stationary points. Not all stationary points are minimizers.



Global optima may not be unique

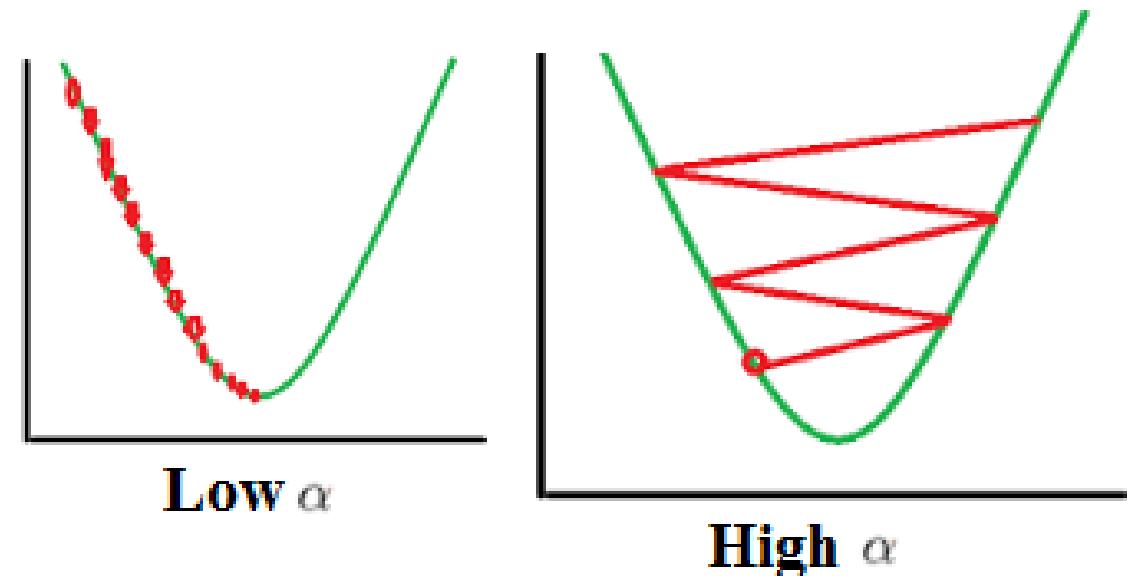


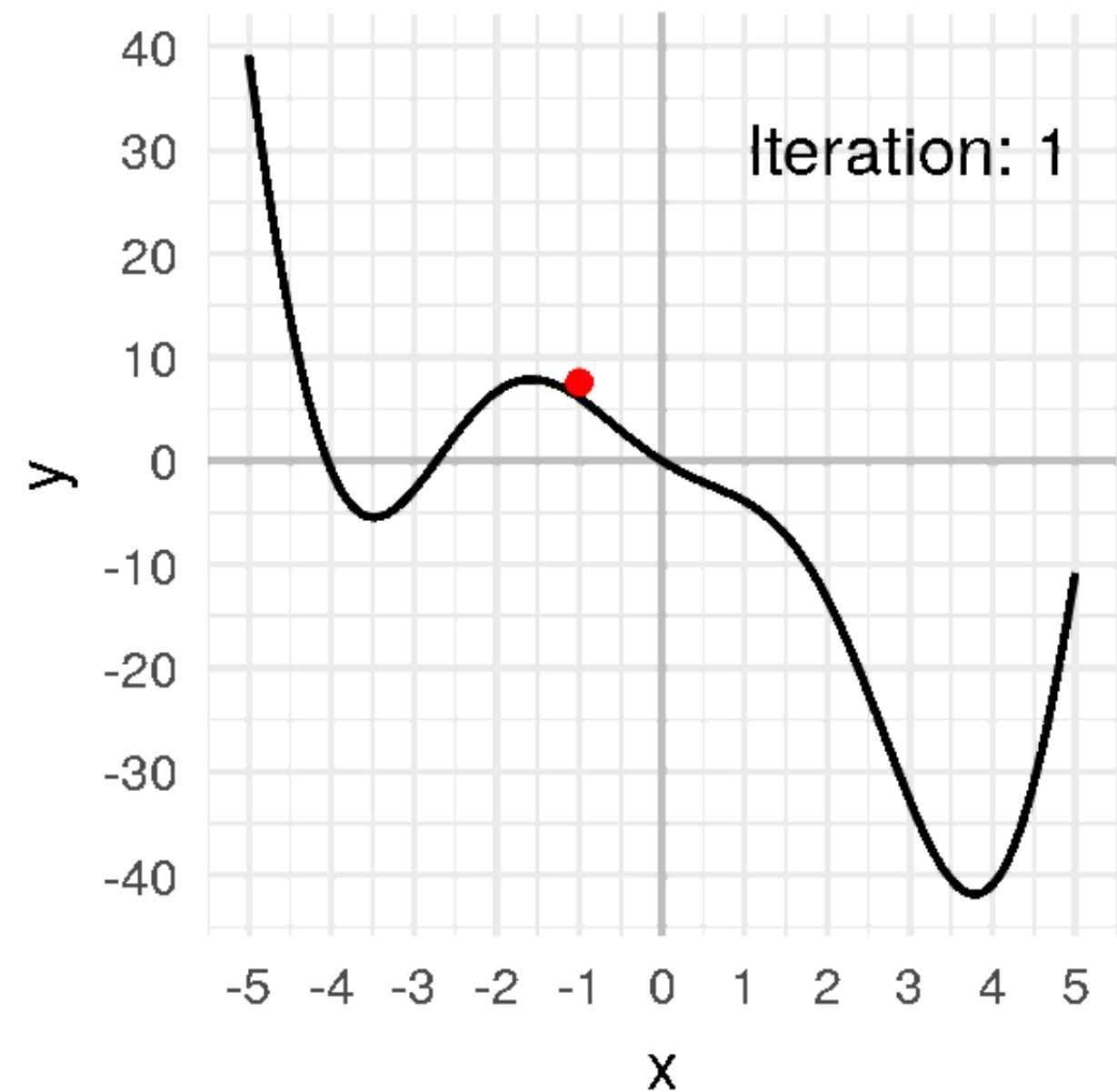
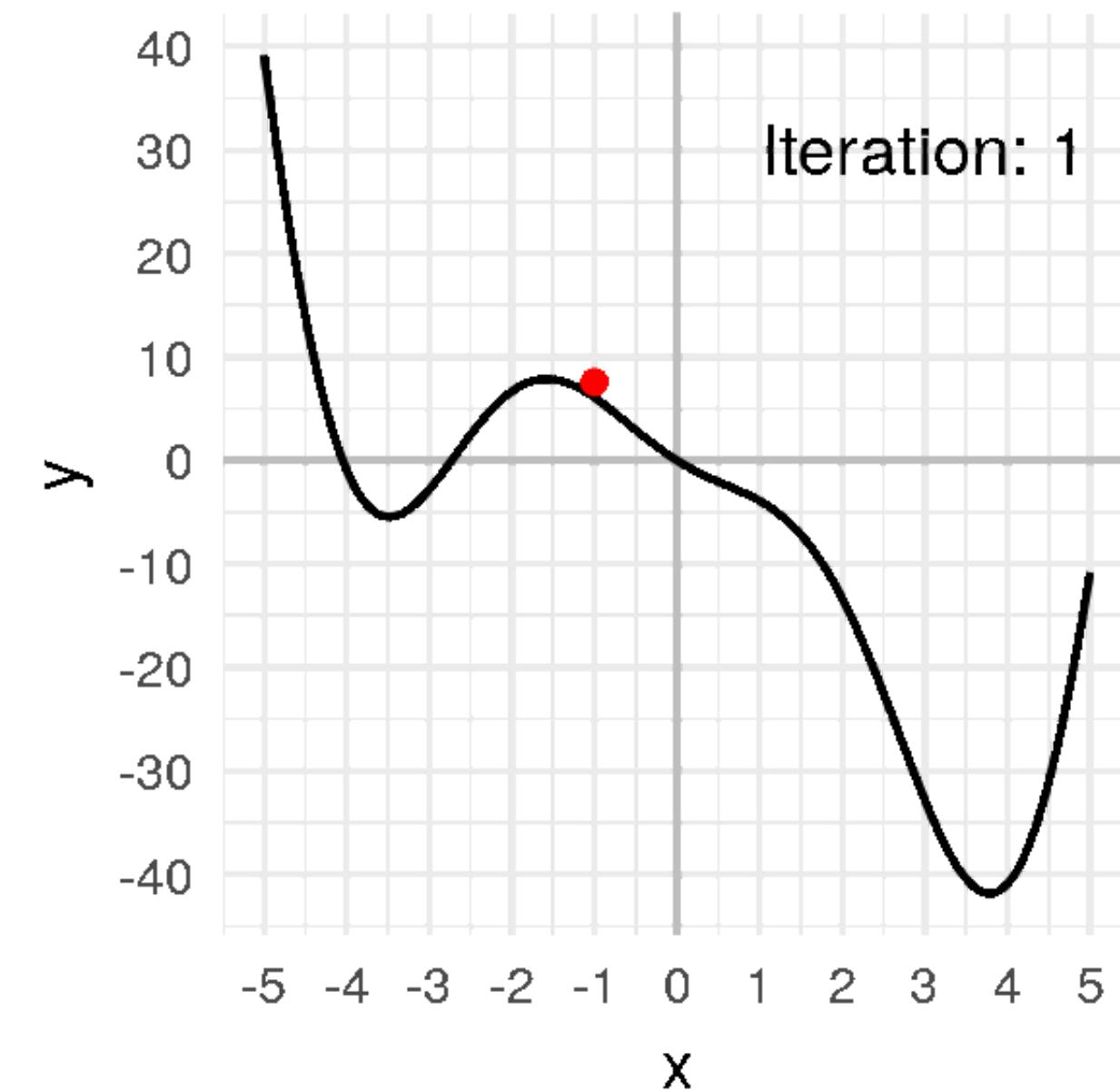
infinitely many
global min

Step size α

$$w^{(t)} = w^{(t-1)} - \underbrace{\alpha}_{\text{Step size}} f'(w^{(t-1)})$$

- Step-size α should be carefully selected as
 - Small step-sizes may lead to slow convergence
 - Large step-sizes may lead to oscillation around the minimum and divergence



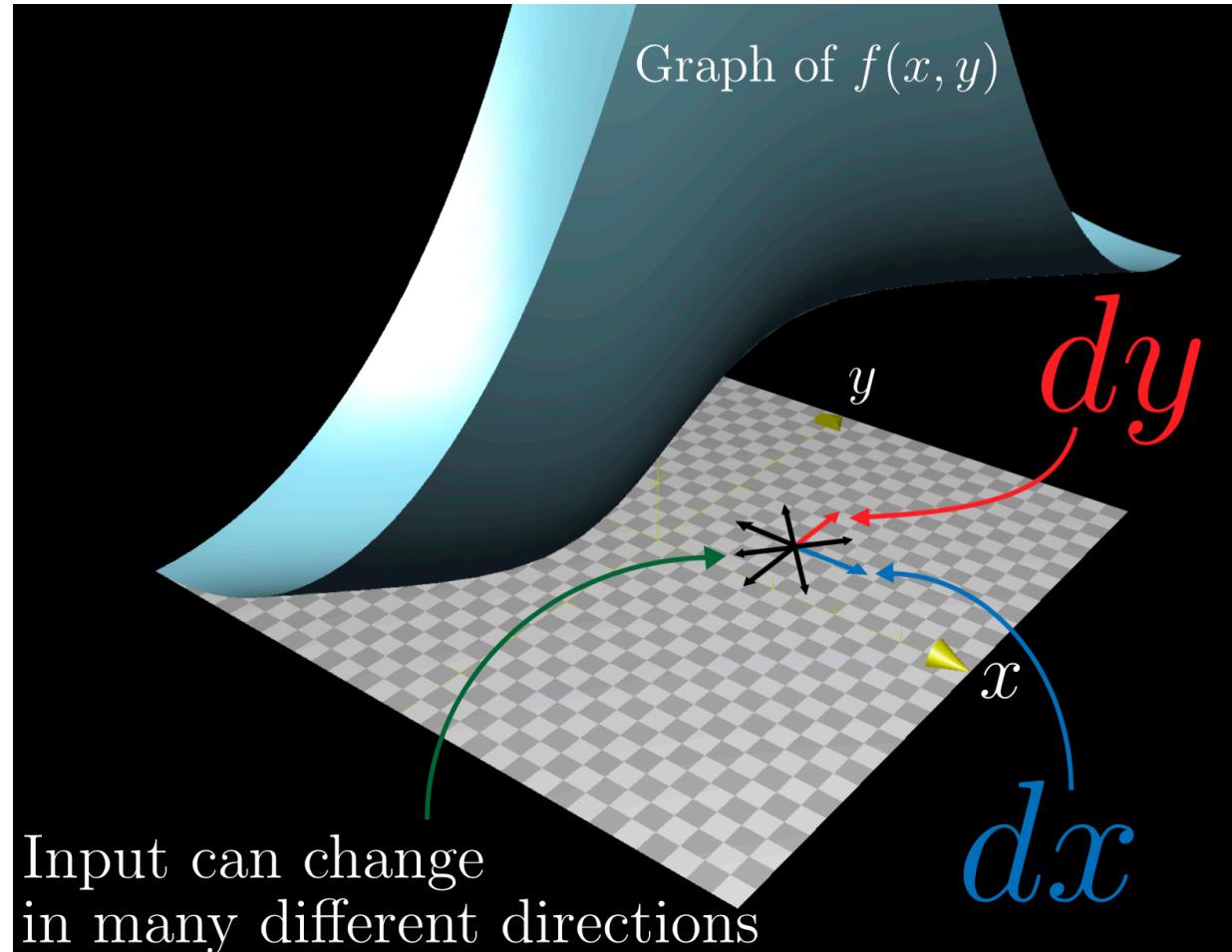


What if you have more than one w

You need to consider which direction $\Phi(w)$ decreases with respect to all w

Partial Derivatives

When you have a function with two inputs $f(x, y)$,

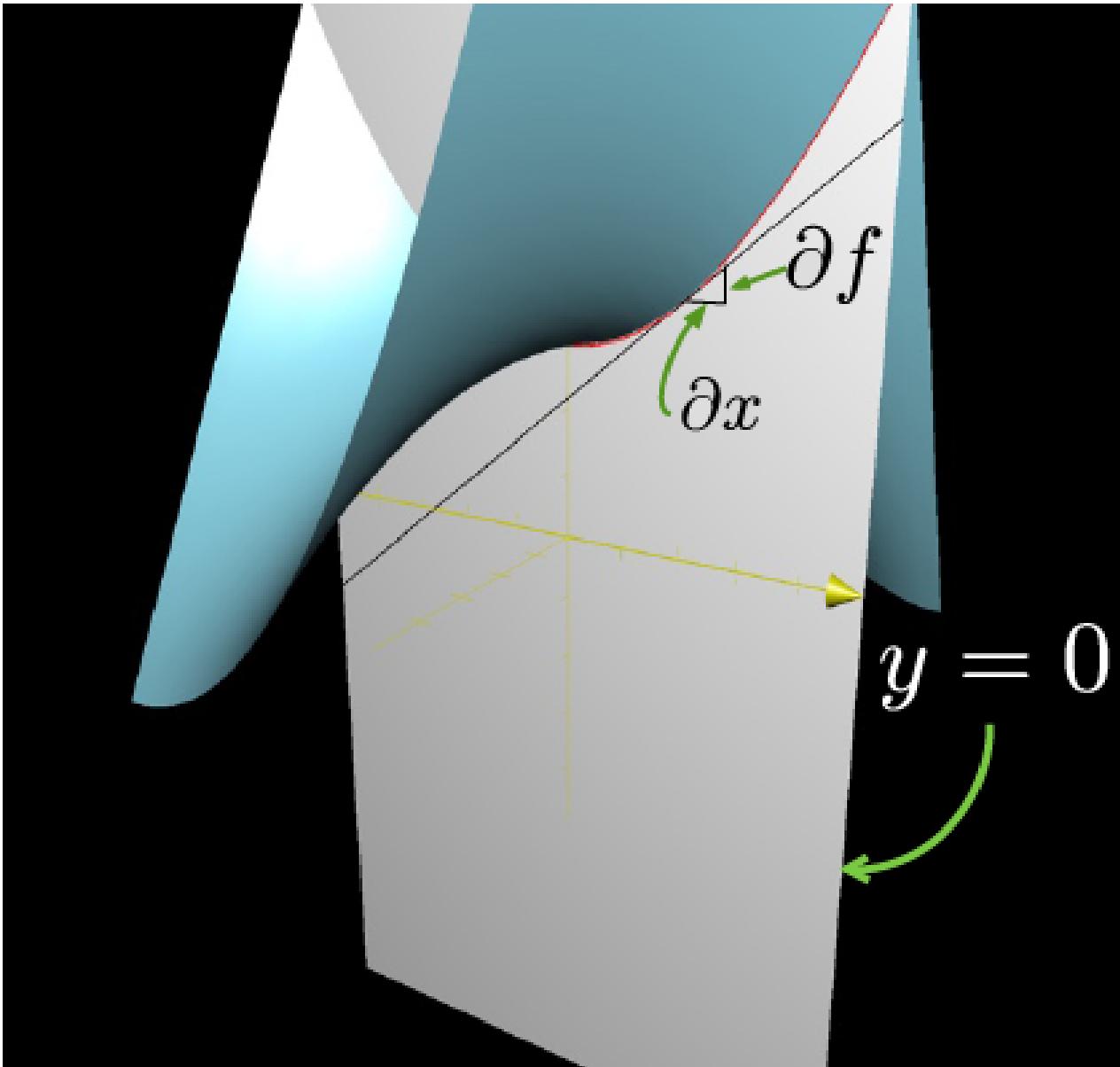


Partial Derivatives

When you have a function with two inputs $f(x, y)$,

- you can look at the tiny change in output with respect to a tiny change in x

$$\frac{\partial f}{\partial x}$$

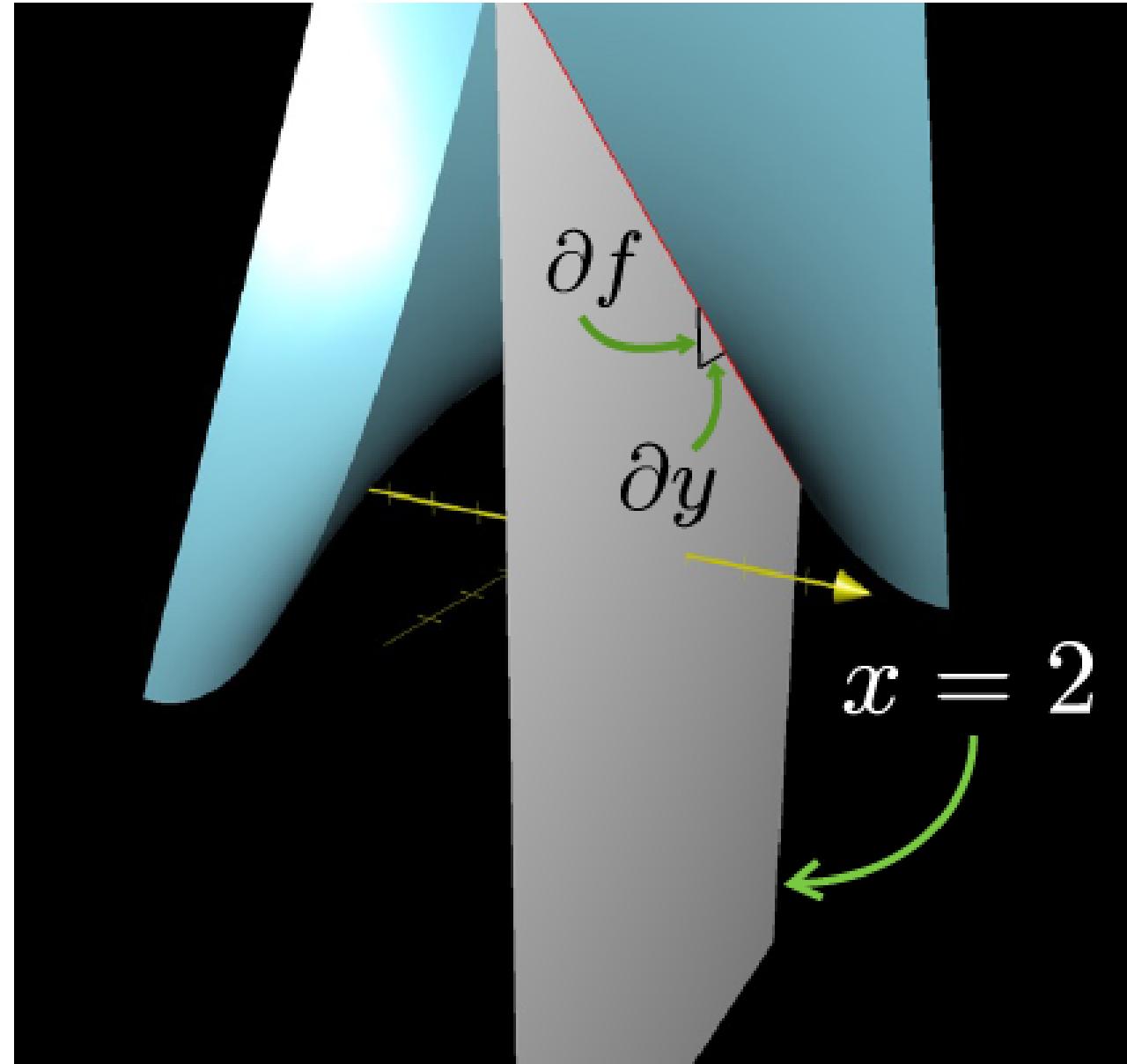


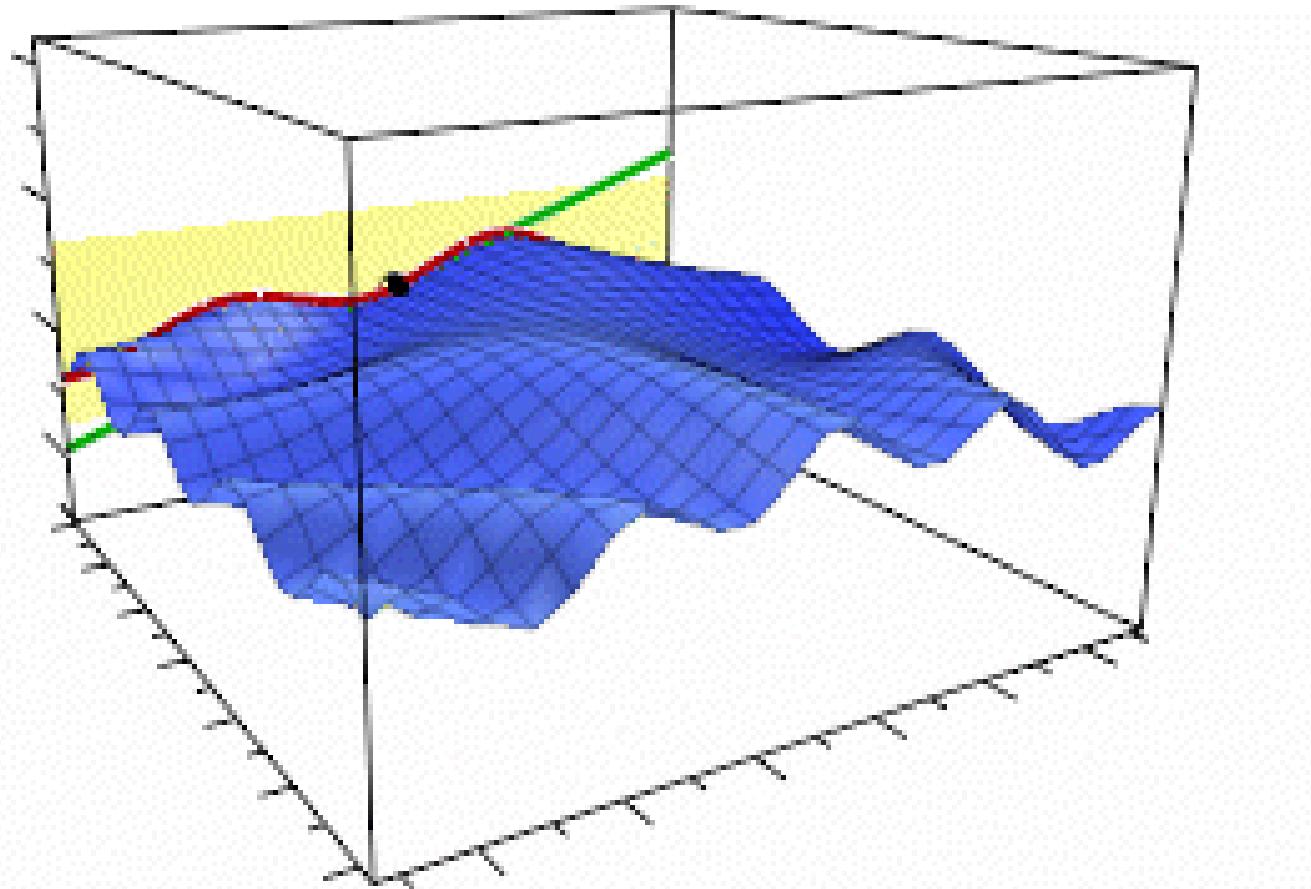
Partial Derivatives

Suppose you have a function with two inputs $f(x, y)$,

- you can look at the tiny change in output with respect to a tiny change in x
- or you can look at the tiny change in output with respect to a tiny change in y

$$\frac{\partial f}{\partial y}$$





Gradient

Suppose you have a function with two inputs $f(x, y)$,

Gradient of f is a vector that contains all partial derivative information

$$\nabla_f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Gradient of Φ w.r.t w

is the vector of its partial derivatives

$$\nabla_w \Phi = \begin{bmatrix} \partial \Phi / \partial w_1 \\ \partial \Phi / \partial w_2 \\ \vdots \\ \partial \Phi / \partial w_d \end{bmatrix}$$

where $w \in \mathbb{R}^d$, $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}$

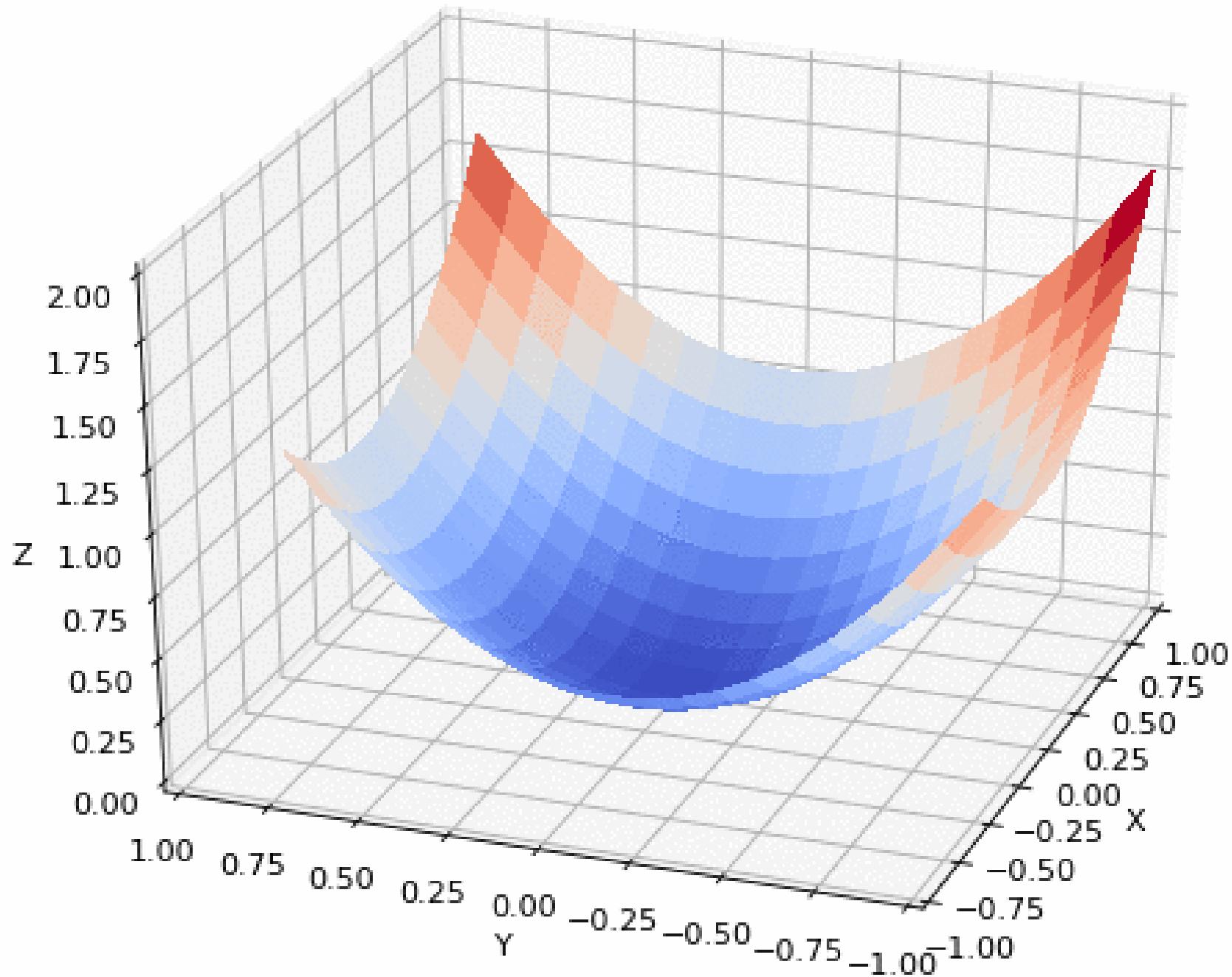
Gradient

Suppose you have a function with two inputs $f(x, y)$,

Gradient of f is a vector that contains all partial derivative information

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Gradient points in the direction of greatest increase in f



Gradient Descent of Multiple Dimensions

Start from an initial point $w^{(0)} \in \mathbb{R}^d, t = 0$

while

- $t = t + 1$
- $w^{(t)} = w^{(t-1)} - \alpha \nabla_w f(w^{(t-1)})$

until $|f(w^{(t)}) - f(w^{(t-1)})| < \epsilon$

return $w^{(t)}$

Logistic Regression Example

$$\Phi(w) = \frac{1}{n} \sum_{i=1}^n - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right) + \frac{\lambda}{2} \|w\|^2$$

Logistic Regression Example

$$\Phi(w) = \frac{1}{n} \sum_{i=1}^n - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right) + \frac{\lambda}{2} \|w\|^2$$

$$g^{(i)} = S(w^T x^{(i)}) = \frac{1}{1 + e^{- (w^T x^{(i)})}}$$

Logistic Regression Example

$$\Phi(w) = \frac{1}{n} \sum_{i=1}^n - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right) + \frac{\lambda}{2} \|w\|^2$$

$$g^{(i)} = S(w^T x^{(i)}) = \frac{1}{1 + e^{-(w^T x^{(i)})}}$$

$$\nabla_w \Phi(w) = \frac{1}{n} \sum_{i=1}^n (g^{(i)} - y^{(i)}) x^{(i)} + \lambda w$$

Logistic Regression Example

Start from an initial point $w^{(0)} \in \mathbb{R}^d, t = 0$

while

- $t = t + 1$
- $w^{(t)} = w^{(t-1)} - \alpha \frac{1}{n} \sum_{i=1}^n (S(w^{(t-1)T} x^{(i)}) - y^{(i)}) x^{(i)} + \lambda w^{(t-1)}$

until $|f(w^{(t)}) - f(w^{(t-1)})| < \epsilon$

return $w^{(t)}$

Batch Gradient Descent

- The gradient descent algorithm we saw is called **Batch Gradient Descent (BGD)**
- In *batch gradient descent*, we sum it for all data and we take a large step in the direction of the gradient

Stochastic Gradient Descent

- In **Stochastic Gradient Descent (SGD)**, we randomly select one data point and take small step
 - Like perceptron
 - Small steps average out the direction of the gradient

Stochastic gradient descent

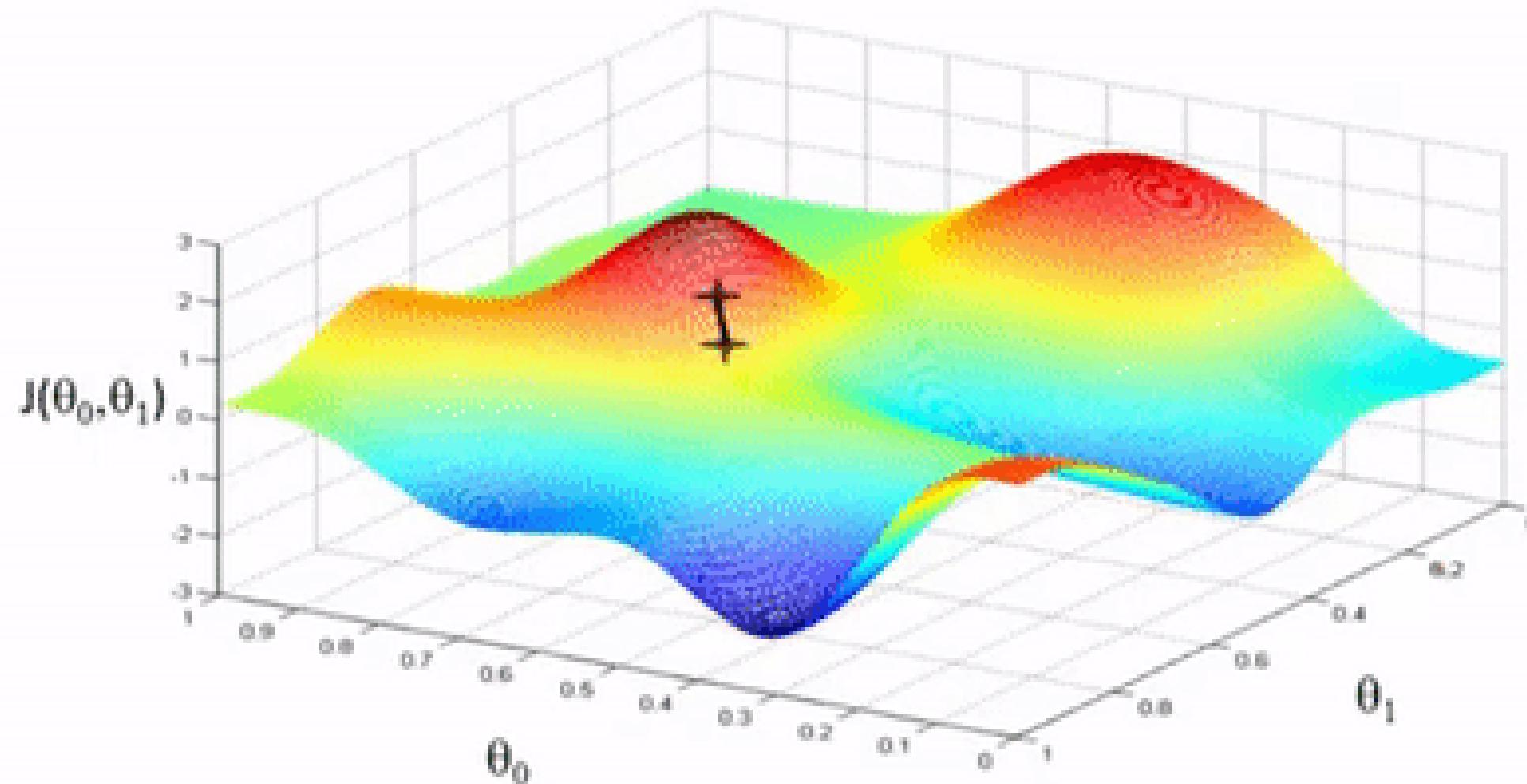
for $t = 1$ to T

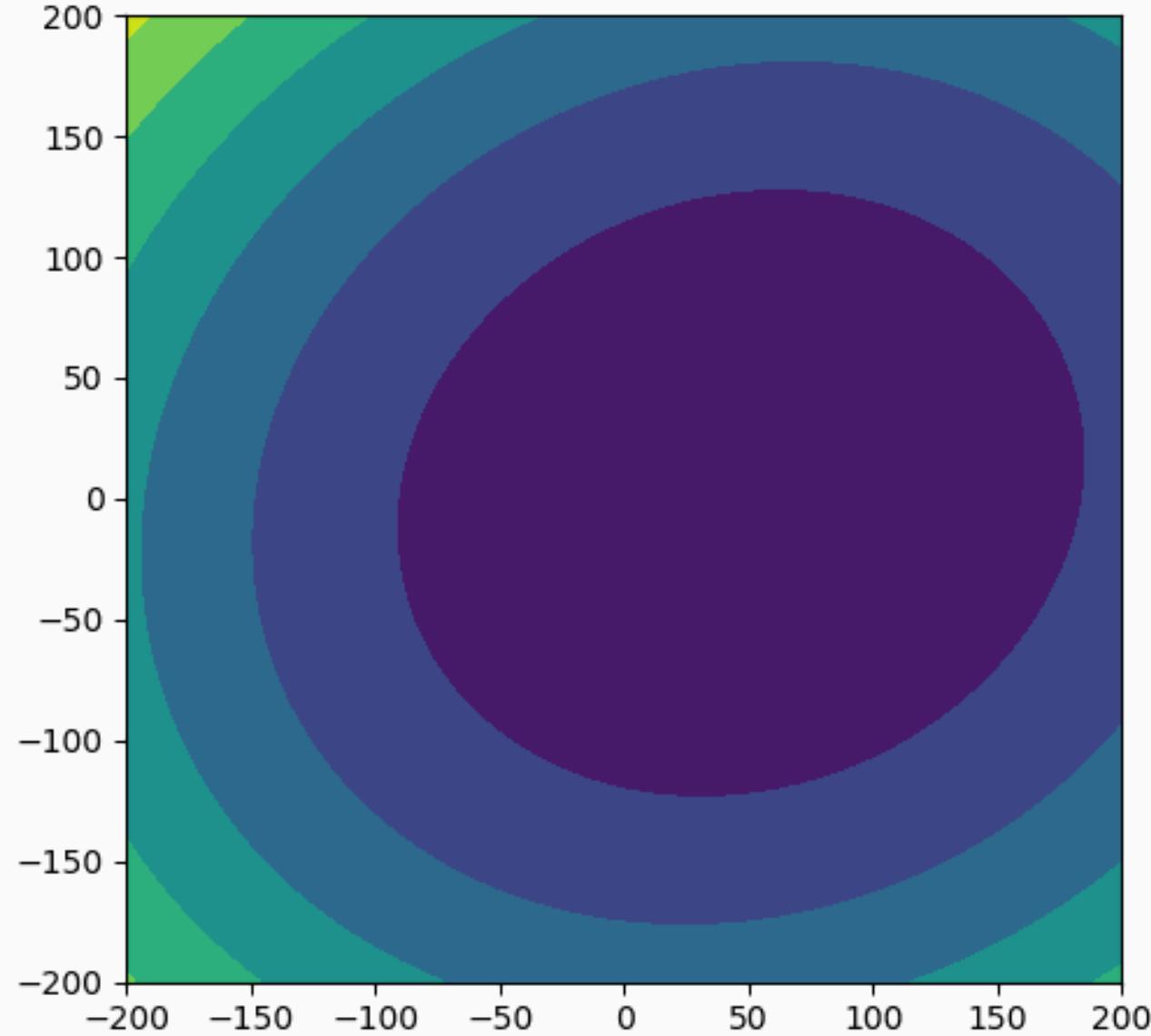
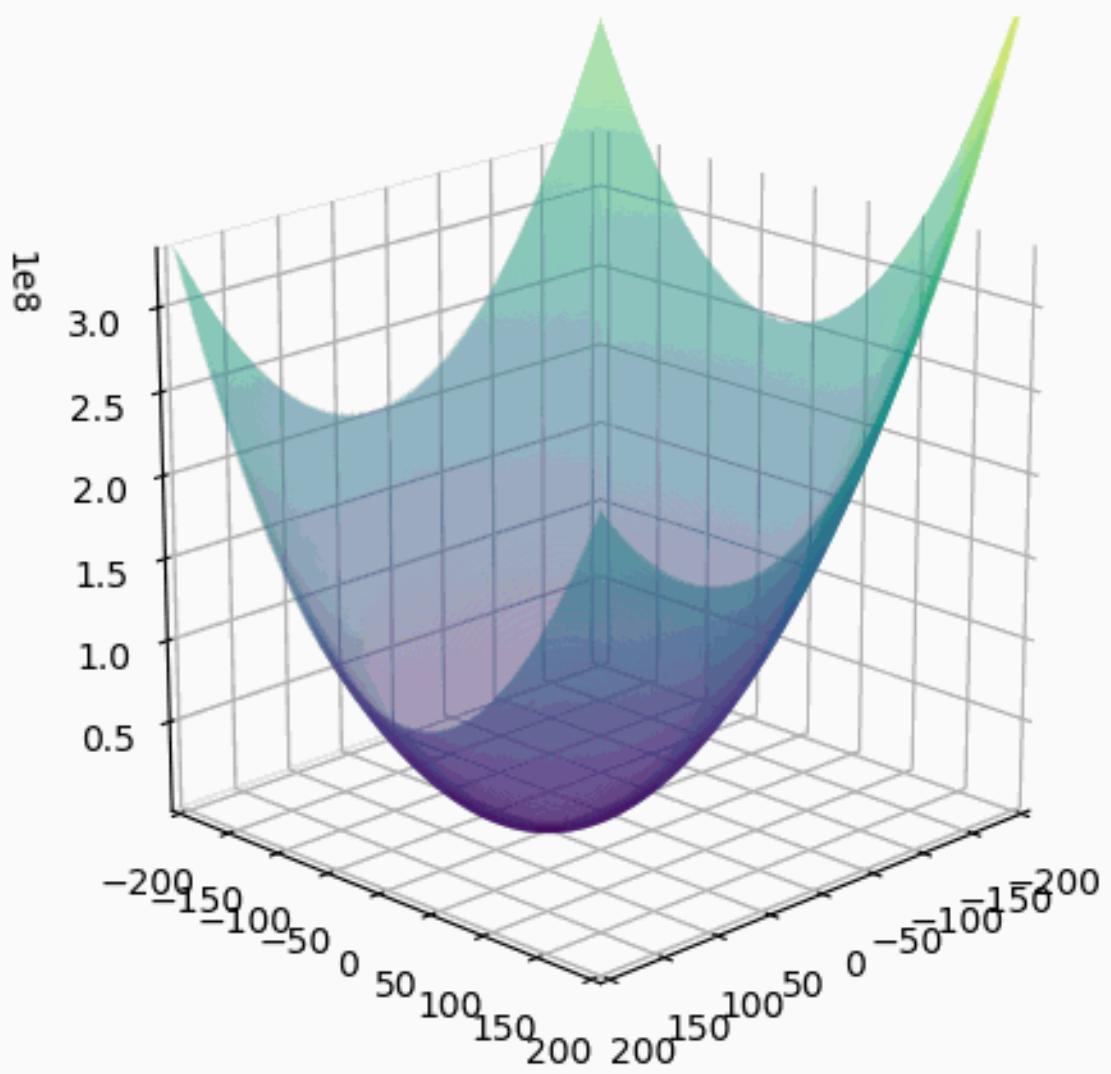
- randomly select $i \in 1, 2, \dots, n$
 - $w^{(t)} = w^{(t-1)} - \alpha(t) \nabla_w f_i(w^{(t-1)})$
- return** $w^{(t)}$

Stochastic gradient descent

- Note that the step size $\alpha(t)$ is a function of t
- For stochastic gradient descent to converge, the step size has to decrease as a function of t

- e.g. $\alpha(t) = \frac{1}{t}$





Advantages of SGD vs BGD

- If the function is non-convex local optima may trap BGD. Taking samples from the gradient may bounce around the landscape and out of the local optima
- Optimizing with BGS may overfit, SGD may avoid overfitting
- BGD requires calculations for every data point. Time and memory requirements for this can be infeasible in large datasets.

COGS514 - Cognition and Machine Learning

Regression

Supervised Learning - Regression

- Predicted values (Y) are continuous

property	room	bathroom	area m^2	location	price (Y)
1	2	1	100	Çayyolu	500000
2	3	1	140	Yıldız	650000
3	3	2	180	Oran	700000
...

Typical Machine Learning Objective Function

$$\Phi(\mathbf{w}) = \underbrace{\frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)})}_{\text{Empirical risk of classifier } h} + \underbrace{\lambda R(\mathbf{w})}_{\substack{\text{Constant} \\ \text{Regularizer}}}$$

Notation Reminder

$$\Phi(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) + \lambda R(\mathbf{w})$$

$h(\mathbf{x}^{(i)}; \mathbf{w})$ guess

$y^{(i)}$ actual

Loss function for linear regression

$$\ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)})$$

Loss function for linear regression

Squared Error Loss Function

$$\text{loss}(\text{guess}, \text{actual}) = ((\text{guess}) - (\text{actual}))^2$$

Squared Error Loss Function

$$\text{loss}(\text{guess}, \text{actual}) = ((\text{guess}) - (\text{actual}))^2$$

Properties:

- Penalizes the positive errors and negative errors the same amount, and the positive and negative errors don't cancel each other out when you add them.
- The higher errors are penalized more
 - as it is a squared number

Regression

- Dataset

$$\mathcal{D} \in \left\{ (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}) \right\}$$

Regression

- Dataset

$$\mathcal{D} \in \left\{ (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}) \right\}$$

- In regression problem, y-values are **real-valued**

$$y^{(i)} \in \mathbb{R}$$

Linear Regression

- Dataset

$$\mathcal{D} \in \left\{ (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}) \right\}$$

- In regression problem, y-values are **real-valued**

$$y^{(i)} \in \mathbb{R}$$

- Hypothesis class (predictions)

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

Linear Regression

- Dataset

$$\mathcal{D} \in \left\{ (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}) \right\}$$

- In regression problem, y-values are **real-valued**

$$y^{(i)} \in \mathbb{R}$$

- Hypothesis class (predictions)

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d$$

Linear Regression

- Dataset

$$\mathcal{D} \in \left\{ (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}) \right\}$$

- In regression problem, y-values are **real-valued**

$$y^{(i)} \in \mathbb{R}$$

- Hypothesis class (predictions)

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d$$

- Loss function is *squared error*

$$\ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

Linear Regression

- In regression problem, y-values are **real-valued**

$$y^{(i)} \in \mathbb{R}$$

- Hypothesis class (predictions)

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

- Loss function is *squared error*

$$\ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

- Objective function is **mean squared error**

$$\Phi(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

Optimizing Loss Function

There is an analytical solution!

Analytical Solution

There is a closed-form formula for optimal solution if we use a *linear hypothesis* that minimize mean squared error. We can find this by:

- Find $\partial\Phi/\partial w_k$ for $k \in \{1, \dots, d\}$
- Construct a set of d equations of the form $\partial\Phi/\partial w_k = 0$
- Solve the system for values of w_k

Analytical Solution

There is a closed-form formula for optimal solution if we use a *linear hypothesis* that minimize mean squared error. We can find this by:

1. Find $\partial\Phi/\partial w_k$ for $k \in \{1, \dots, d\}$ (that is $\nabla_{\mathbf{w}}\Phi$)
2. Construct a set of d equations of the form $\partial\Phi/\partial w_k = 0$ (that is $\nabla_{\mathbf{w}}\Phi = 0$)
3. Solve the system $\nabla_{\mathbf{w}}\Phi = 0$ for values of w_k

We can use Matrix operations to do this more concisely

Data

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Data

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad \mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$$

-Every data example $\mathbf{x}^{(i)}$ is a row in \mathbf{X}

Analytical solution - Step 1 Gradient $\nabla_{\mathbf{w}} \Phi$

$$\Phi(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

Analytical solution - Step 1 Gradient $\nabla_{\mathbf{w}} \Phi$

Objective function with matrix operations

$$\Phi(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

$$\Phi(\mathbf{w}) = \frac{1}{n} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Analytical solution - Step 1 Gradient $\nabla_{\mathbf{w}} \Phi$

$$\Phi(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

$$\Phi(\mathbf{w}) = \frac{1}{n} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Gradient with respect to \mathbf{w} is

$$\nabla_{\mathbf{w}} \Phi = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Recall: Gradient of Φ w.r.t \mathbf{w} ($\nabla_{\mathbf{w}}\Phi$)

is the vector of its partial derivatives

$$\nabla_{\mathbf{w}}\Phi = \begin{bmatrix} \partial\Phi/\partial w_1 \\ \partial\Phi/\partial w_2 \\ \vdots \\ \partial\Phi/\partial w_d \end{bmatrix}$$

where $\mathbf{w} \in \mathbb{R}^d$, $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}$

In this case

$$\nabla_{\mathbf{w}}\Phi = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Analytical solution - Step 2 Make $\nabla_{\mathbf{w}} \Phi = 0$

$$\nabla_{\mathbf{w}} \Phi = 0$$

Analytical solution - Step 2 Make $\nabla_{\mathbf{w}}\Phi = 0$

$$\nabla_{\mathbf{w}}\Phi = 0$$

$$\nabla_{\mathbf{w}}\Phi = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

Analytical solution - Step 2 Make $\nabla_{\mathbf{w}}\Phi = 0$

$$\nabla_{\mathbf{w}}\Phi = 0$$

$$\nabla_{\mathbf{w}}\Phi = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

Analytical solution - Step 3 solve $\nabla_{\mathbf{w}} \Phi = 0$ for values of \mathbf{w}

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

Analytical solution - Step 3 solve $\nabla_{\mathbf{w}} \Phi = 0$ for values of \mathbf{w}

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{X}^T \mathbf{y} = 0$$

Analytical solution - Step 3 solve $\nabla_{\mathbf{w}} \Phi = 0$ for values of \mathbf{w}

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{X}^T \mathbf{y} = 0$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

Analytical solution - Step 3 solve $\nabla_{\mathbf{w}} \Phi = 0$ for values of \mathbf{w}

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{X}^T \mathbf{y} = 0$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Analytical solution - Step 3 solve $\nabla_{\mathbf{w}} \Phi = 0$ for values of \mathbf{w}

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{X}^T \mathbf{y} = 0$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Note: dimensions $\mathbf{X} \in \mathbb{R}^{n \times d}$, $\mathbf{y} \in \mathbb{R}^{n \times 1}$ $\mathbf{w} \in \mathbb{R}^{d \times 1}$

Questions

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

1. What if $(\mathbf{X}^T \mathbf{X})$ is not invertible?
2. We only optimized for empirical risk? Doesn't that overfit?

1. What if $(X^T X)$ is not invertible?

- When $X^T X$ is not invertible equation $\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$ will not have a unique solution (It will have infinite number of solutions)
- This happens when you have fewer training samples than your features than your $n < d$
 - where $X \in \mathbb{R}^{n \times d}$, $\mathbf{y} \in \mathbb{R}^{n \times 1}$ $\mathbf{w} \in \mathbb{R}^{d \times 1}$
- In this case, we take **pseudo-inverse**
 - If the matrix is invertible its pseudo inverse is the same as its inverse

Questions

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

1. What if $(\mathbf{X}^T \mathbf{X})$ is not invertible?
2. We only optimized for empirical risk? Doesn't that overfit?

2. We only optimized for empirical risk? Doesn't that overfit?

Answer

Add ℓ_2 regularization to the objective function (*ridge regression*)

Ridge Regression

Ordinary Least Squares with ℓ_2 regularizer

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

Ridge Regression

Orginary Least Squares with ℓ_2 regularizer

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

Higher λ values forces \mathbf{w} to be near 0

Analytical Solution (Ridge) - Step 1

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

Analytical Solution (Ridge) - Step 1

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \|\mathbf{w}\|^2$$

Gradient w.r.t \mathbf{w} is

$$\nabla_{\mathbf{w}} \Phi_{ridge} = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda \mathbf{w}$$

Analytical solution (Ridge) - Step 2 Make $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\nabla_{\mathbf{w}} \Phi_{ridge} = 0$$

Analytical solution (Ridge) - Step 2 Make $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\nabla_{\mathbf{w}} \Phi_{ridge} = 0$$

$$\nabla_{\mathbf{w}} \Phi_{ridge} = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

Analytical solution (Ridge) - Step 2 Make $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\nabla_{\mathbf{w}} \Phi_{ridge} = 0$$

$$\nabla_{\mathbf{w}} \Phi_{ridge} = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

Analytical solution (Ridge) - Step 3 solve $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

Note: dimensions $\mathbf{X} \in \mathbb{R}^{n \times d}$, $\mathbf{y} \in \mathbb{R}^{n \times 1}$ $\mathbf{w} \in \mathbb{R}^{d \times 1}$

Analytical solution (Ridge) - Step 3 solve $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} - \frac{1}{n} \mathbf{X}^T \mathbf{y} + \lambda\mathbf{w} = 0$$

Analytical solution (Ridge) - Step 3 solve $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} - \frac{1}{n} \mathbf{X}^T \mathbf{y} + \lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \frac{1}{n} \mathbf{X}^T \mathbf{y}$$

Analytical solution (Ridge) - Step 3 solve $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} - \frac{1}{n} \mathbf{X}^T \mathbf{y} + \lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \frac{1}{n} \mathbf{X}^T \mathbf{y}$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} + n\lambda\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

Analytical solution (Ridge) - Step 3 solve $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} - \frac{1}{n} \mathbf{X}^T \mathbf{y} + \lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \frac{1}{n} \mathbf{X}^T \mathbf{y}$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} + n\lambda\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$(\mathbf{X}^T \mathbf{X} + n\lambda I)\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

Analytical solution (Ridge) - Step 3 solve $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} - \frac{1}{n} \mathbf{X}^T \mathbf{y} + \lambda\mathbf{w} = 0$$

$$\frac{1}{n} \mathbf{X}^T \mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \frac{1}{n} \mathbf{X}^T \mathbf{y}$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} + n\lambda\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$(\mathbf{X}^T \mathbf{X} + n\lambda I)\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + n\lambda I)^{-1} \mathbf{X}^T \mathbf{y}$$

Analytical solution (Ridge) - Step 3 solve $\nabla_{\mathbf{w}} \Phi_{ridge} = 0$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + n\lambda I)^{-1} \mathbf{X}^T \mathbf{y}$$

$(\mathbf{X}^T \mathbf{X} + n\lambda I)$ is invertible when $\lambda > 0$

Regularization and offset

- When we showed the analytical solution for ridge regression, we included the offset term w_0 in \mathbf{w} for simplicity
- However, you **don't typically regularize w_0** as this finds the right level of surface for your data.
- The objective function for ridge regression can be shown as follows

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

Optimizing using Gradient Descent

Optimizing using Gradient Descent

- Analytical solution is impractical for large d (large number of features)
- *Inverting* $d \times d$ matrix $\mathbf{X}^T \mathbf{X}$ takes $O(d^3)$ time
- Gradient descent can be used in high dimensions
 - batch or stochastic gradient descent

Gradient Descent of Multiple Dimensions

Start from an initial point $\mathbf{w}^{(0)} \in \mathbb{R}^d, t = 0$

while

- $t = t + 1$
- $\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \alpha \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-1)})$

until $|f(\mathbf{w}^{(t)}) - f(\mathbf{w}^{(t-1)})| < \epsilon$

return $\mathbf{w}^{(t)}$

Gradient Descent of Multiple Dimensions

Start from an initial point $\mathbf{w}^{(0)} \in \mathbb{R}^d, t = 0$

while

- $t = t + 1$
- $\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \alpha \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-1)})$

until $|f(\mathbf{w}^{(t)}) - f(\mathbf{w}^{(t-1)})| < \epsilon$

return $\mathbf{w}^{(t)}$

- We need $\nabla_{\mathbf{w}} \Phi_{ridge}$ for gradient descent and that is

$$\nabla_{\mathbf{w}} \Phi_{ridge} = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda \mathbf{w}$$

Gradient Descent

- If we show the offset separately (to not to include it in the regularizer)
The objective function is

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

Gradient Descent

- If we show the offset separately (to not to include it in the regularizer)
The objective function is

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

- Gradient w.r.t \mathbf{w} is

$$\nabla_{\mathbf{w}} \Phi_{ridge} = \frac{2}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)}) \mathbf{x}^{(i)} + 2\lambda \mathbf{w}$$

Gradient Descent

- If we show the offset separately (to not to include it in the regularizer)
The objective function is

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

- Gradient w.r.t \mathbf{w} is

$$\nabla_{\mathbf{w}} \Phi_{ridge} = \frac{2}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)}) \mathbf{x}^{(i)} + 2\lambda \mathbf{w}$$

- Partial derivative w.r.t. w_0 is

$$\frac{\partial \Phi}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)})$$

Gradient Descent

- If we show the offset separately (to not to include it in the regularizer)
The objective function is

$$\Phi_{ridge}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)})^2 + \lambda \|\mathbf{w}\|^2$$

- Gradient w.r.t \mathbf{w} is

$$\nabla_{\mathbf{w}} \Phi_{ridge} = \frac{2}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)}) \mathbf{x}^{(i)} + 2\lambda \mathbf{w}$$

- Partial derivative w.r.t. w_0 is

$$\frac{\partial \Phi}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} + w_0 - y^{(i)})$$

Use this gradient and partial derivative in gradient descent

Gradient Descent for Ridge Regression

- The objective function for *ridge regression* and *ordinary least squares* is **convex**
- Gradient descent is **guaranteed** to find a **global optimal solution** with a suitable α parameter

Bias - Variance Trade-off

Structural vs Estimation Error

Bias (structural error) vs Variance (estimation error) trade-off

Model's generalization error can be expressed in terms of

1. **Structural error (bias):** This happens when no hypothesis $h \in \mathcal{H}$ will perform well on the data.
 - e.g. the data is generated by a sin wave but you try to fit a line

Bias (structural error) vs Variance (estimation error) trade-off

Model's generalization error can be expressed in terms of

1. **Structural error (bias):** This happens when no hypothesis $h \in \mathcal{H}$ will perform well on the data.
 - e.g. the data is generated by a sin wave but you try to fit a line
2. **Estimation error (variance):** This error arises because data is not good enough to enable us to select a good $h \in \mathcal{H}$

Training vs. Test Error

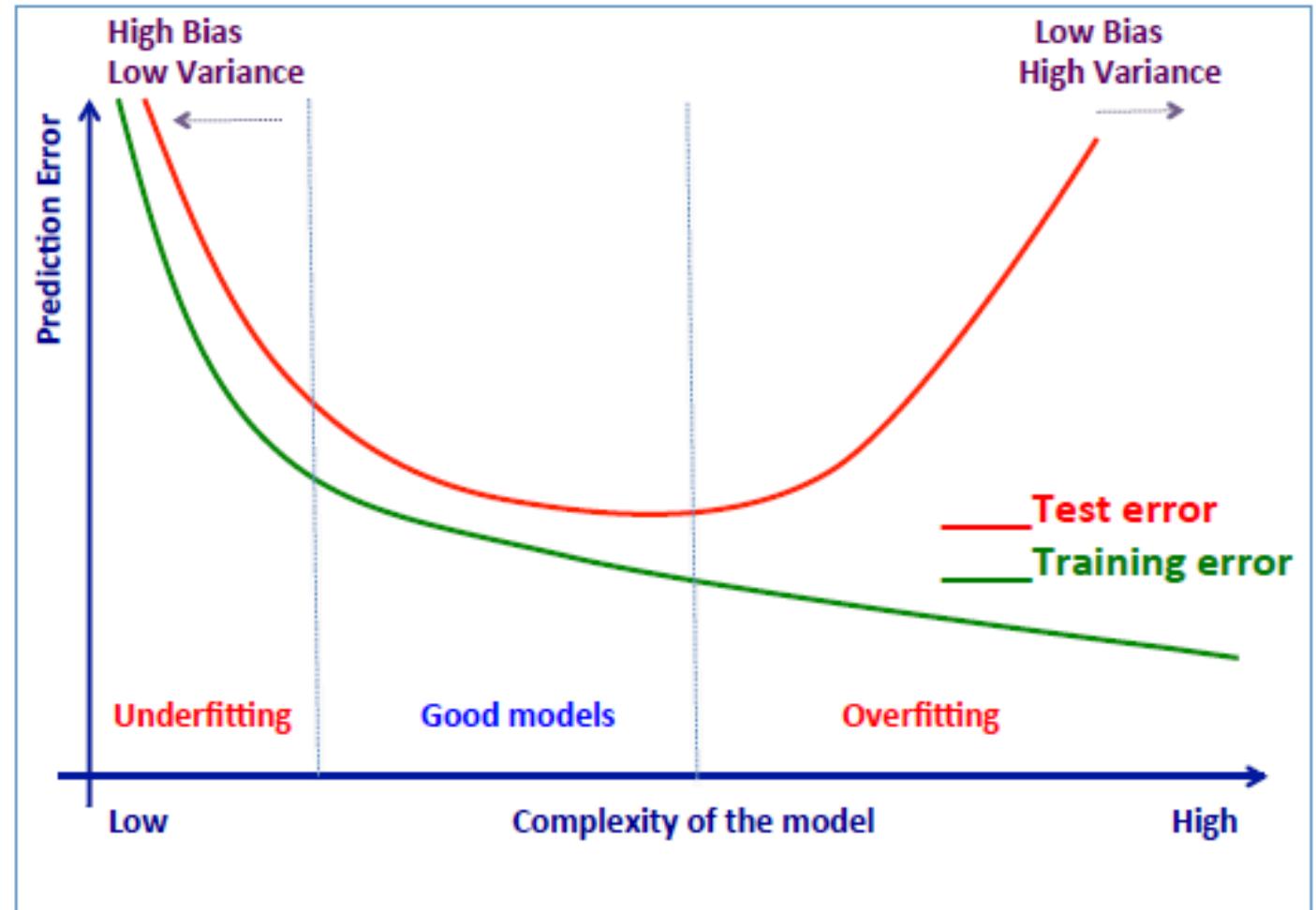
To understand the generalization properties (bias & variance) of a machine learning algorithm

- We separate the data into training, validation and test
- We only use the training dataset to learn the parameters (to optimize the objective function)
- We use the validation dataset to select hyperparameters (e.g. λ) and understand generalization properties (bias & variance)
- We use the test dataset to make a final evaluation, we don't change the model after this.



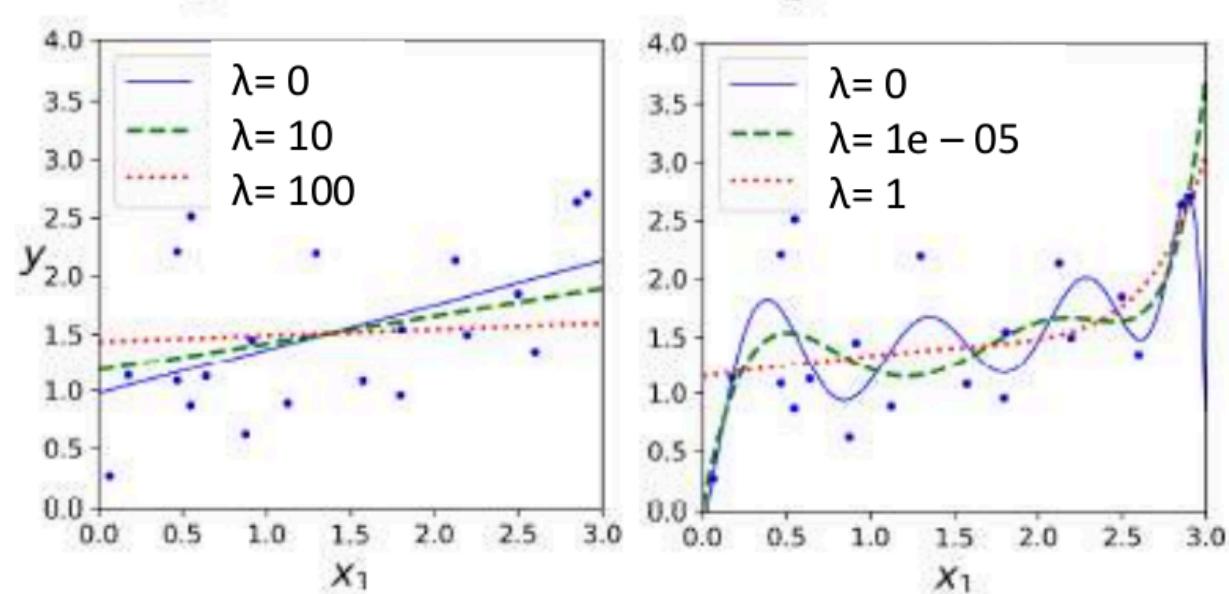
Bias (structural error) vs Variance (estimation error) trade-off

1. Structural error (bias):
2. Estimation error (variance):



Regularization - Ridge Regression

- Ridge regression trained with different λ values
- Increasing λ leads to flatter (less extreme) predictions



Important - Scaling

Important - Scaling

Ridge regression (and other regularized models) are sensitive to the scale of the features

Don't forget to scale (standardize) the data

$$\frac{x - \mu}{\sigma}$$

Regularization - Lasso Regression

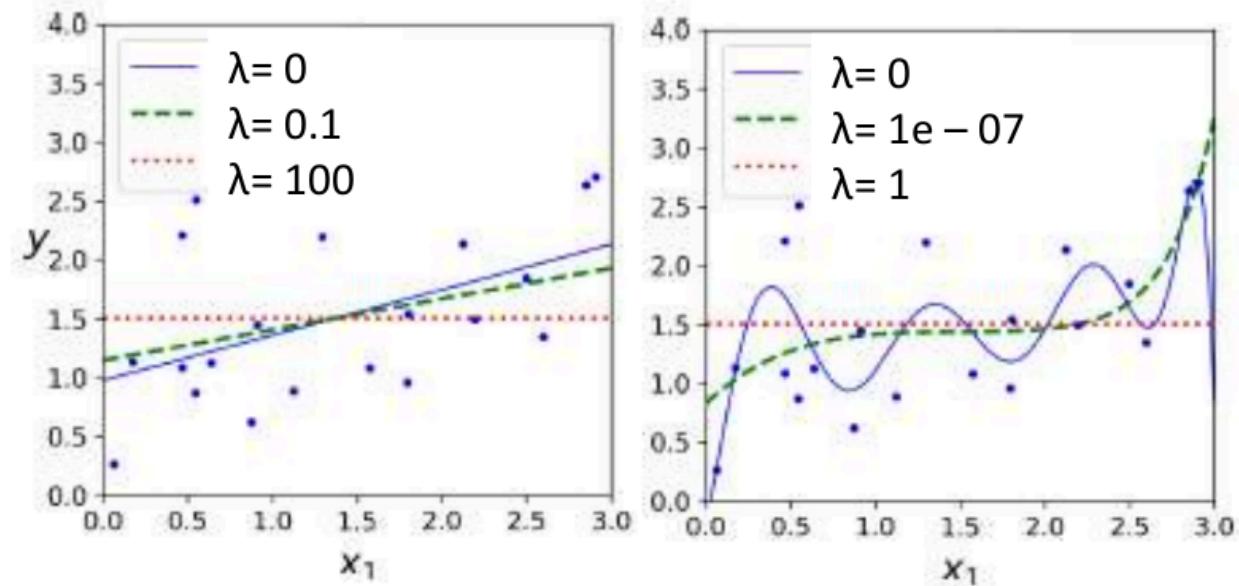
$$\Phi_{lasso}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \lambda \|\mathbf{w}\|_1$$

- Ordinary least squares regression with $\ell 1$ regularization

Regularization - Lasso Regression

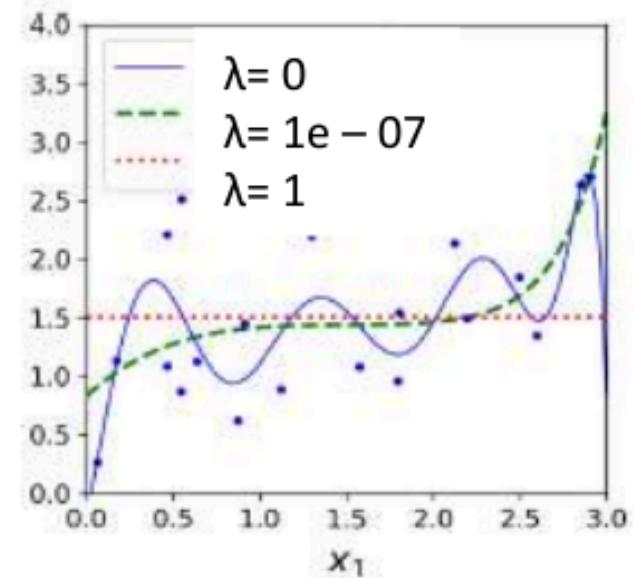
$$\Phi_{lasso}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \lambda \|\mathbf{w}\|_1$$

- Lasso tends to eliminate the weights of least important features
 - It automatically selects features

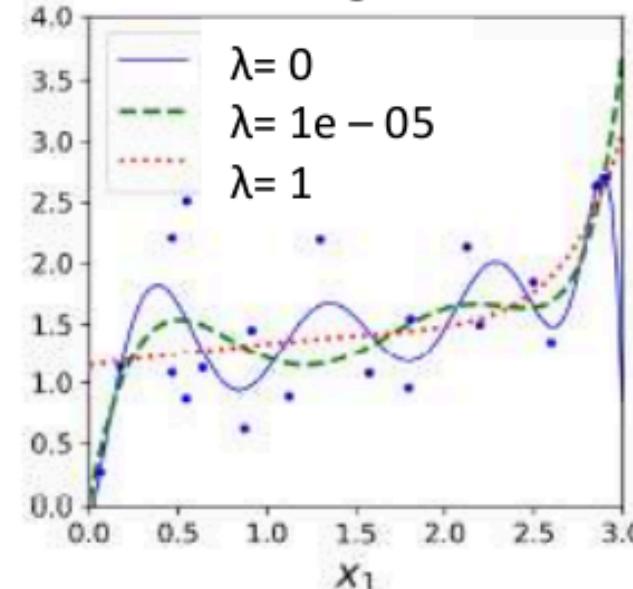


Lasso vs. Ridge

Lasso

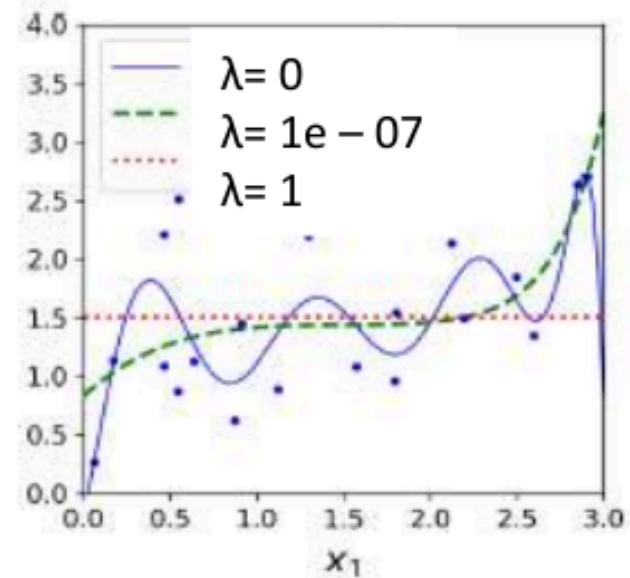


Ridge

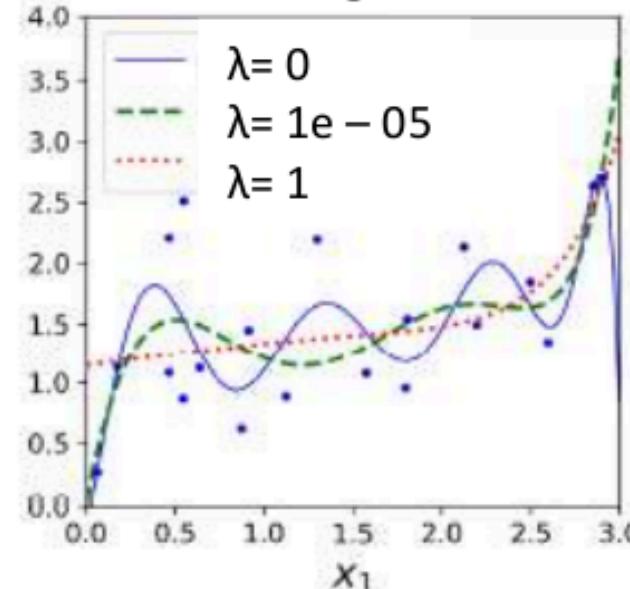


Lasso vs. Ridge

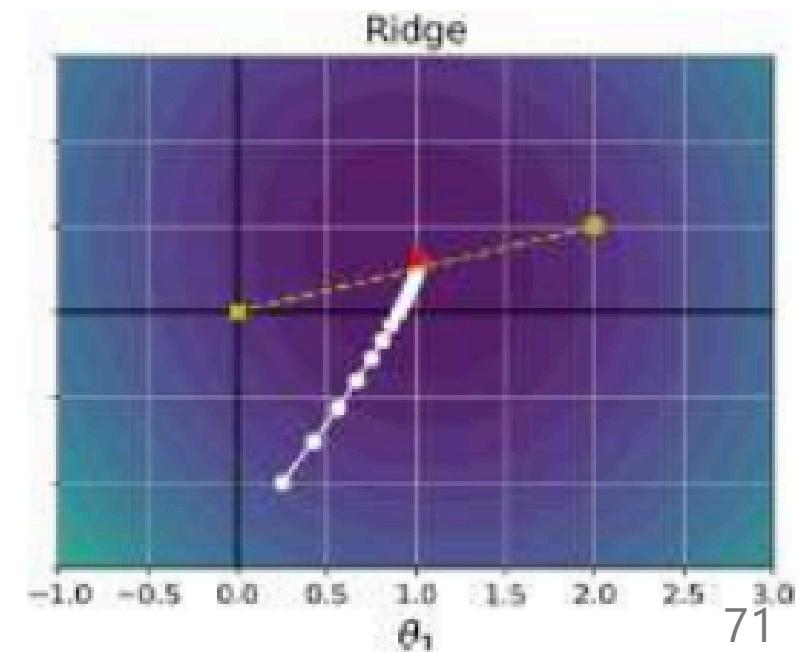
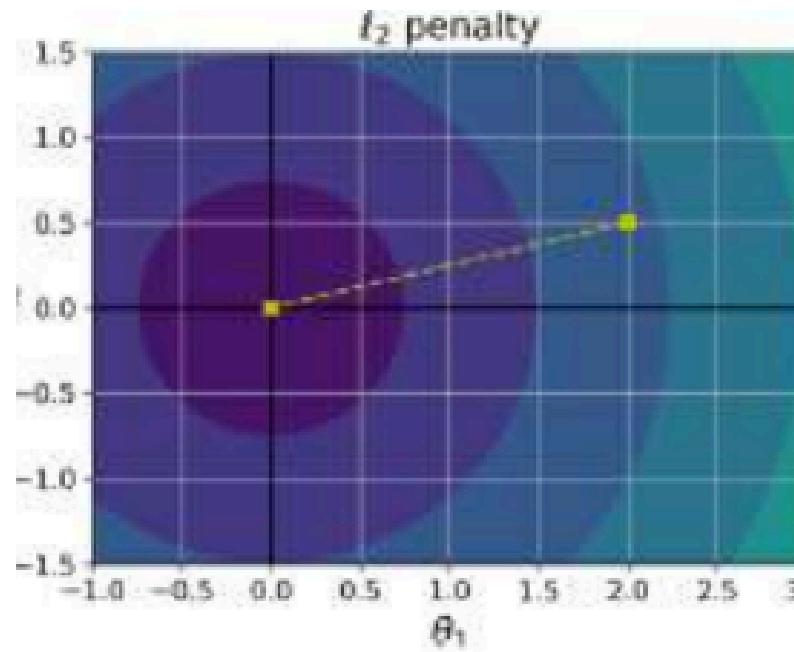
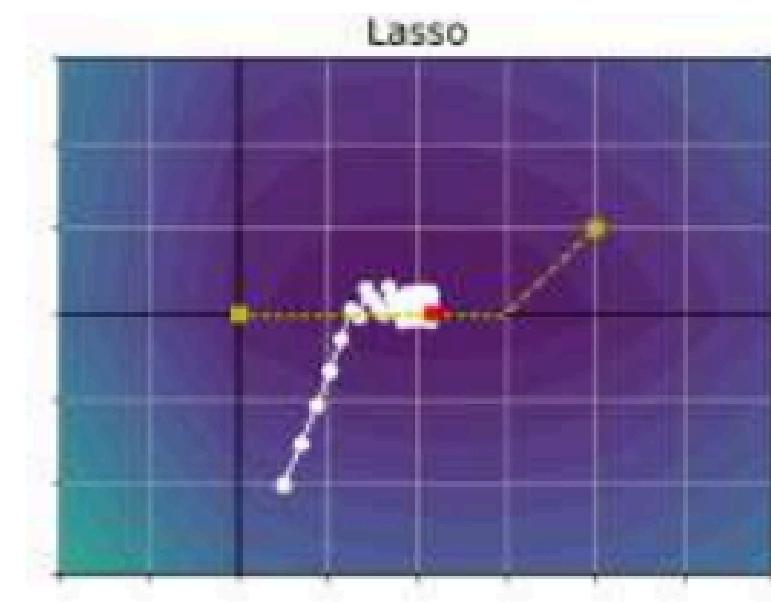
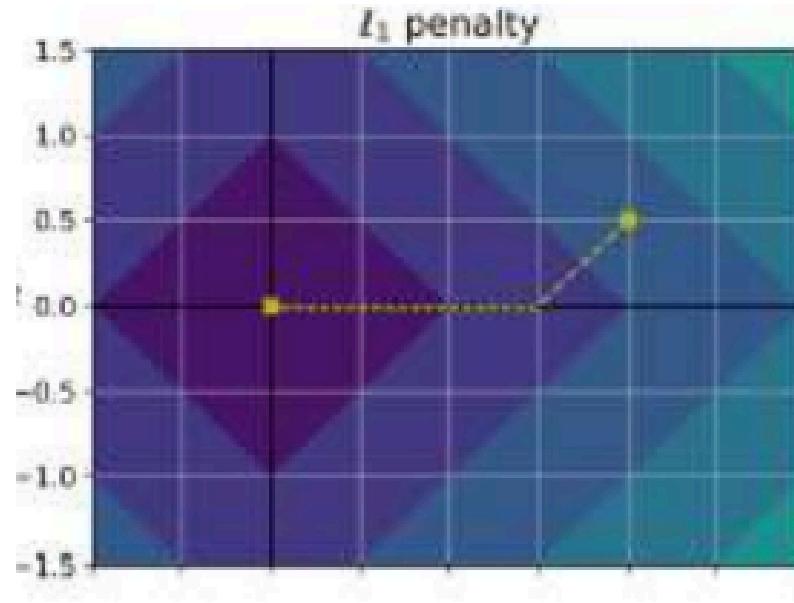
Lasso



Ridge



Lasso vs. Ridge



Elastic Net

A combination of lasso and ridge regression

Elastic Net

$$\Phi_{elastic}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 + r\lambda \|\mathbf{w}\|_1 + (1-r)\lambda \|\mathbf{w}\|_2^2$$

Elastic Net

$$\Phi_{elastic}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 + r\lambda \|\mathbf{w}\|_1 + (1-r)\lambda \|\mathbf{w}\|_2^2$$

$r \in [0, 1]$ is the control mix ratio. It assigns how much weight to give lasso and ridge regularization

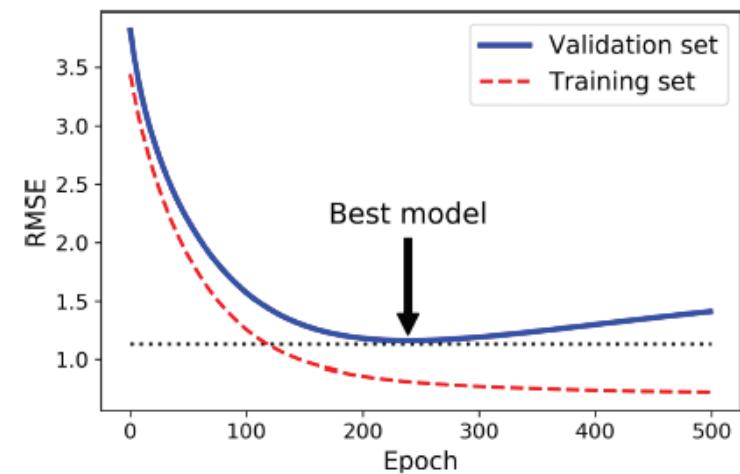
- $r = 1$ is equivalent to lasso regression
- $r = 0$ is equivalent to ridge regression

Suggestions

- Ridge is a good default
- Use lasso if you think only a few features are relevant
- Lasso may behave erratic when you have more features than data.

Early Stopping - A different way to regularize

- Monitor both training and validation error
- Stop training as soon as validation error reaches a minimum
- It can be used for regularization for iterative learning algorithms like stochastic gradient descent



COGS514 - Cognition and Machine Learning

Error Metrics

Error Metrics in Machine Learning

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Types of Errors - Binary Classification

True Positive Rate (TPR)

$$TPR = P(h(x) = +1 \mid +1)$$

- Probability of predicting $+1$ when the true value is $+1$
- Also known as *sensitivity*, *power*, *probability of detection* or *recall*

Types of Errors - Binary Classification

False Negative Rate (FNR)

$$FNR = 1 - TPR = P(h(x) = -1 \mid +1)$$

- Probability of predicting -1 when the true value is $+1$
- Also known as **type II error**

Types of Errors - Binary Classification

False Positive Rate (FPR)

$$FPR = P(h(x) = +1 \mid -1)$$

- Probability of predicting $+1$ when the true value is -1
- Also known as **type I error** or *false alarm rate*

Types of Errors - Binary Classification

False Positive Rate (FPR)

$$TNR = 1 - FPR = P(h(x) = -1 \mid -1)$$

- Probability of predicting -1 when the true value is -1
- Also known as **specificity**

Types of Errors - Binary Classification

- True Positive Rate (TPR)

$$TPR = P(h(x) = +1 \mid +1)$$

- False Negative Rate (FNR)

$$FNR = P(h(x) = -1 \mid +1)$$

- False Positive Rate (FPR)

$$FPR = P(h(x) = +1 \mid -1)$$

- True Negative Rate (TNR)

$$TNR = P(h(x) = -1 \mid -1)$$

Other Error Metrics in Machine Learning

Precision

$$P(y = +1 \mid h(x) = +1)$$

- Probability that the true value is $+1$ when the prediction is $+1$

$$\text{precision} = \frac{p_{+1}TPR}{p_{-1}FPR + p_{+1}TPR}$$

Other Error Metrics in Machine Learning

False Discovery Rate

$$FDR = 1 - \text{precision} = P(y = -1 \mid h(x) = +1)$$

Other Error Metrics in Machine Learning

F-1 score

Harmonic mean of precision and recall

$$F_1 = \frac{2 \ TPR}{1 + TPR + \frac{p_{-1}}{p_{+1}} FPR}$$

Optimize TPR and FPR

- TPR and FPR are computing objectives
 - maximize TPR
 - minimize FPR
- Minimizing risk is to optimize a balance between TPR and FPR

$$R[h] = \mathbb{E}[\ell(h(x), y)] = \alpha FPR - \beta TPR + \gamma$$

- α and β are positive values and γ is some constant
- We can always achieve $(TPR = 0, FPR = 0)$ and $(TPR = 1, FPR = 1)$
- What happens to the values in between?

Neyman - Pearson Lemma

- Suppose we want to maximize TPR by setting an upper bound on FPR

maximize TPR

subject to $FPR \leq \alpha$

Neyman - Pearson Lemma

- Suppose we want to maximize TPR by setting an upper bound on FPR

maximize TPR

subject to $FPR \leq \alpha$

- Lets optimize over probabilistic predictor that returns $+1$ with probability $p(x)$ and -1 with probability $1 - p(x)$

Neyman - Pearson Lemma

- Suppose we want to maximize TPR by setting an upper bound on FPR

$$\begin{aligned} & \text{maximize} && TPR \\ & \text{subject to} && FPR \leq \alpha \end{aligned}$$

- Let's optimize over probabilistic predictor that returns $+1$ with probability $p(x)$ and -1 with probability $1 - p(x)$
- The optimal probabilistic decision rule for this problem is a deterministic likelihood test

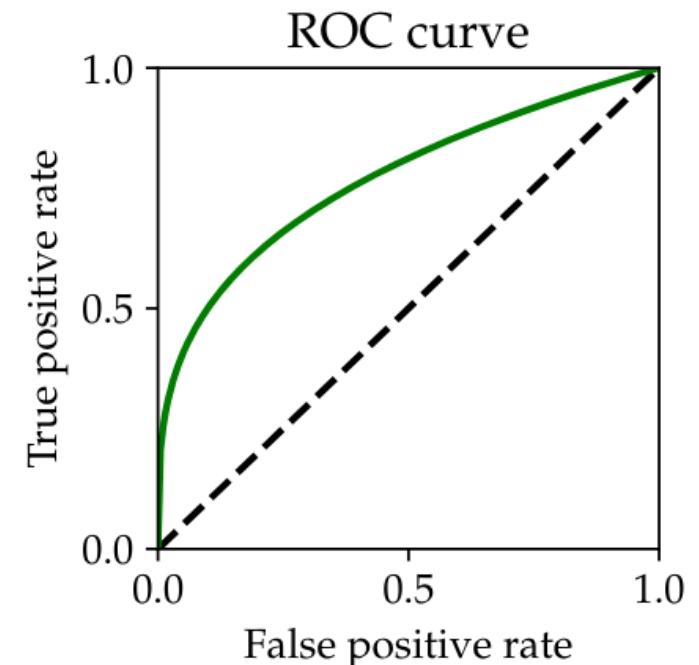
$$h(\mathbf{x}) = \begin{cases} +1 & \mathcal{L}(x) \geq \eta \\ -1 & \text{otherwise} \end{cases}$$

Receiver Operating Characteristic (ROC) Curve

- Curve of all (TPR, FPR) pairs for a decision rule (classifier)

$$R[h] = \mathbb{E}[\ell(h(x), y)] = \alpha FPR - \beta TPR + \gamma$$

- Properties
 - (0,0) and (1,1) are always on the ROC curve
 - $TPR \geq FPR$
 - ROC curve is concave



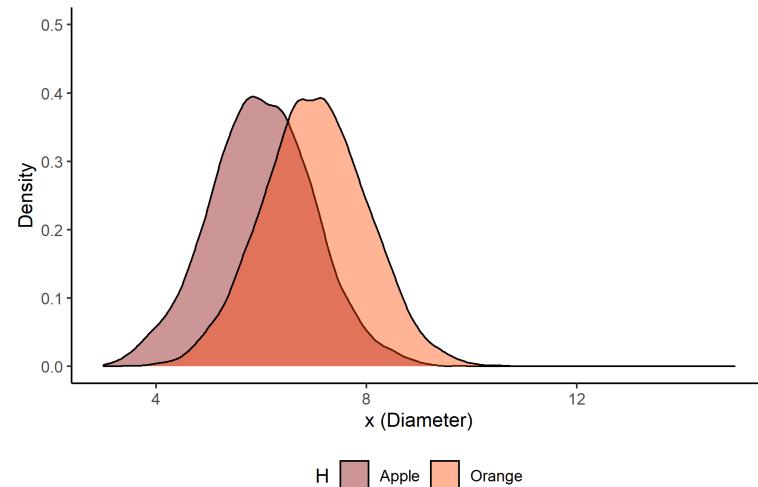
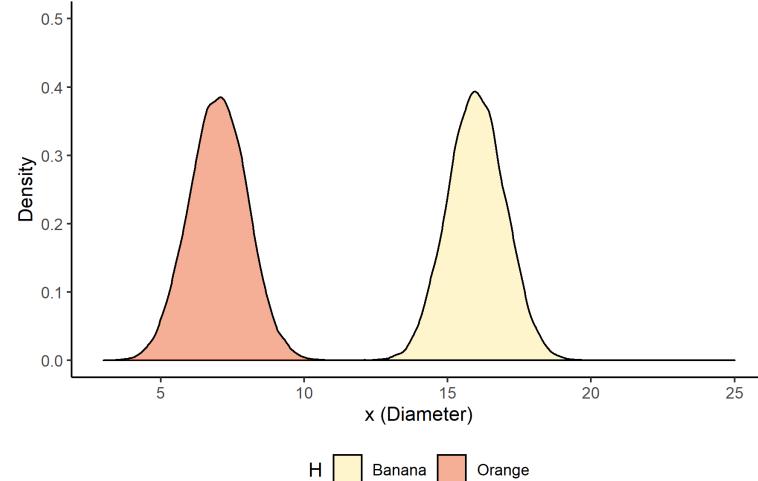
Example: Needle in the haystack

$$p(x \mid y = -1) = \text{Gaussian}(\mu, \sigma)$$

$$p(x \mid y = +1) = \text{Gaussian}(\mu + s, \sigma)$$

- Signal to noise ratio

$$SNR = \frac{s}{\sigma}$$



Example: Needle in the haystack

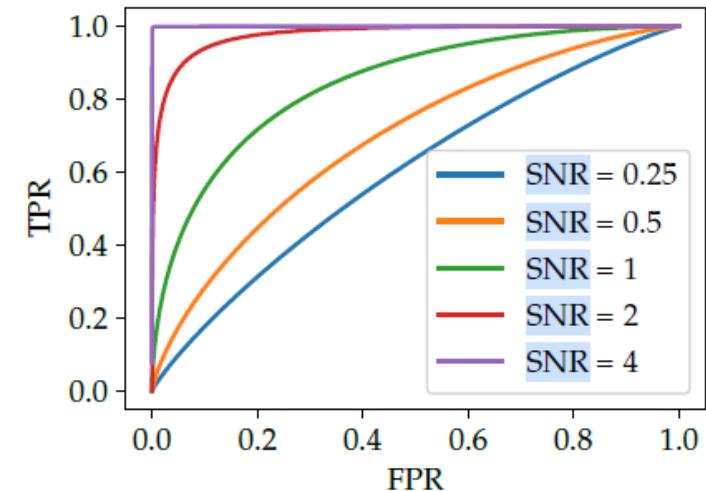
$$p(x \mid y = -1) = \text{Gaussian}(\mu, \sigma)$$

$$p(x \mid y = +1) = \text{Gaussian}(\mu + s, \sigma)$$

- Signal to noise ratio

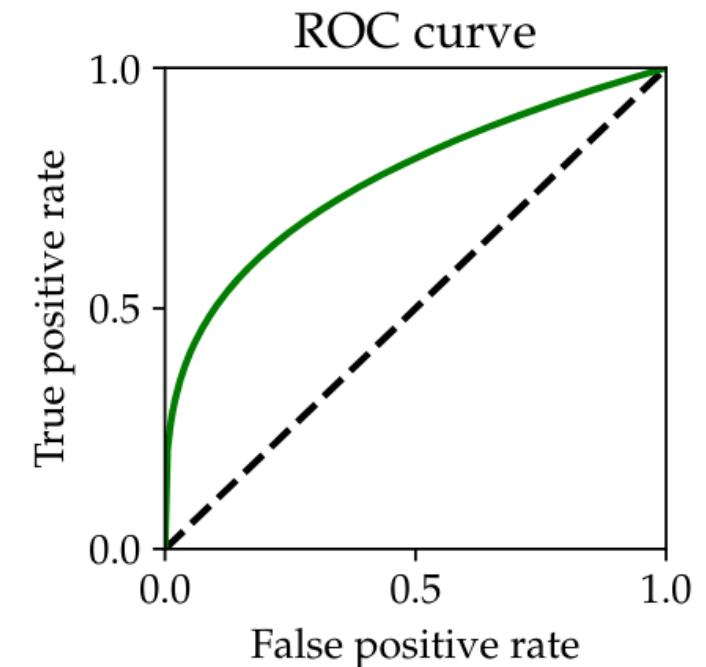
$$SNR = \frac{s}{\sigma}$$

- For small SNR ROC curve is close to $FPR = TPR$ line
- For large SNR , TPR approaches 1 for all values of FPR



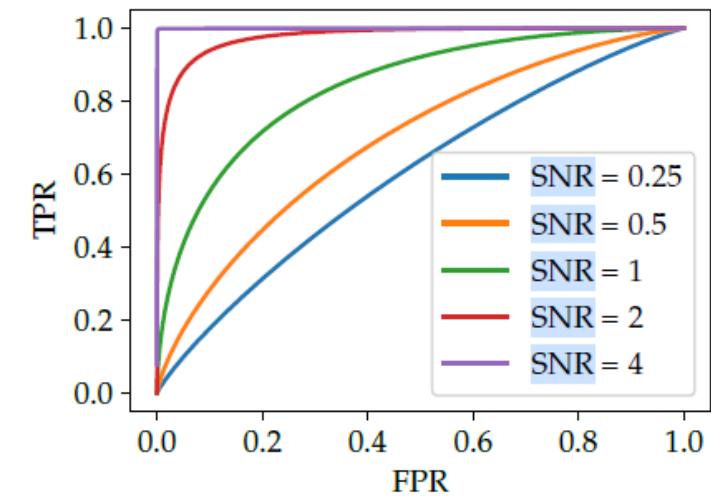
Area under the ROC curve (AUC)

- Summary measure for evaluating the quality of a decision function (classifier)



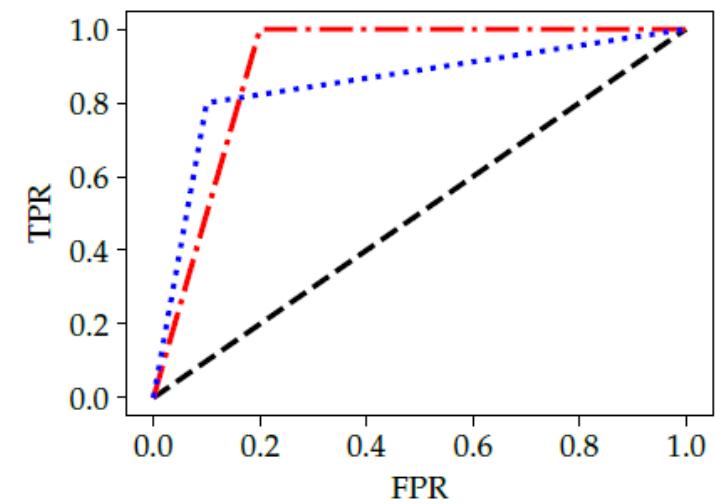
Area under the ROC curve (AUC)

- Summary measure for evaluating the quality of a decision function (classifier)
- As signal to noise ratio increase, AUC increase



Area under the ROC curve (AUC)

- Summary measure for evaluating the quality of a decision function (classifier)
- As signal to noise ratio increase, AUC increase
- Can be misleading, two ROC curves with the same AUCs can have different



Sample vs Population

$$R[h] = \mathbb{E}[\ell(h(x), y)]$$

$$h(\mathbf{x}) = \begin{cases} +1 & \frac{P(\mathbf{x}|y = +1)}{P(\mathbf{x}|y = -1)} \geq \frac{p_{-1}(\ell(1, -1) - \ell(-1, -1))}{p_{+1}(\ell(-1, 1) - \ell(1, 1))} \\ -1 & \text{otherwise} \end{cases}$$

$$p_{-1} = P(y = -1)$$

$$p_{+1} = P(y = +1)$$

Sample vs Population

$$R[h] = \mathbb{E}[\ell(h(x), y)]$$

$$h(\mathbf{x}) = \begin{cases} +1 & \frac{P(\mathbf{x}|y = +1)}{P(\mathbf{x}|y = -1)} \geq \frac{p_{-1}(\ell(1, -1) - \ell(-1, -1))}{p_{+1}(\ell(-1, 1) - \ell(1, 1))} \\ -1 & \text{otherwise} \end{cases}$$

$$p_{-1} = P(y = -1)$$

$$p_{+1} = P(y = +1)$$

- We often don't know these probability distribution. We don't have access to the entire population of instances

Sample vs Population

$$R[h] = \mathbb{E}[\ell(h(x), y)]$$

$$h(\mathbf{x}) = \begin{cases} +1 & \frac{P(\mathbf{x}|y = +1)}{P(\mathbf{x}|y = -1)} \geq \frac{p_{-1}(\ell(1, -1) - \ell(-1, -1))}{p_{+1}(\ell(-1, 1) - \ell(1, 1))} \\ -1 & \text{otherwise} \end{cases}$$

$$p_{-1} = P(y = -1)$$

$$p_{+1} = P(y = +1)$$

- We have some examples in a dataset. We assume these examples are from the same distribution and they are drawn independently

$$\mathcal{D} \in \left\{ (\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}) \right\}$$

Empirical Risk Minimization - Supervised Learning

- Empirical Risk

$$R_{\mathcal{D}}[h] = \mathbb{E}_{\mathcal{D}}[\ell(h(x), y)]$$

$$= \frac{1}{n} \sum_{i=1}^n \ell(h(x^{(i)}), y^{(i)})$$

Empirical Risk Minimization - Supervised Learning

- Empirical Risk

$$R_{\mathcal{D}}[h] = \mathbb{E}_{\mathcal{D}}[\ell(h(x), y)]$$

$$= \frac{1}{n} \sum_{i=1}^n \ell(h(x^{(i)}), y^{(i)})$$

- In supervised learning we aim to find a good classifier from data points by optimizng empirical risk

$$\min_{h \in \mathcal{H}} R_{\mathcal{D}}[h]$$

Empirical Risk Minimization - Supervised Learning

- Empirical Risk

$$R_{\mathcal{D}}[h] = \mathbb{E}_{\mathcal{D}}[\ell(h(x), y)]$$

$$= \frac{1}{n} \sum_{i=1}^n \ell(h(x^{(i)}), y^{(i)})$$

- In supervised learning we aim to find a good classifier from data points by optimizing empirical risk

$$\min_{h \in \mathcal{H}} R_{\mathcal{D}}[h]$$

- Empirical risk $R_{\mathcal{D}}[h]$ serves as a proxy for risk $R[h]$

Generalization

- How well the performance of our classifier transfers from seen examples to unseen examples drawn from the same population?

Generalization

- How well the performance of our classifier transfers from seen examples to unseen examples drawn from the same population?

Generalization Gap

$$R[h] = R_{\mathcal{D}}[h] + (R[h] - R_{\mathcal{D}}[h])$$

Generalization

- How well the performance of our classifier transfers from seen examples to unseen examples drawn from the same population?

Generalization Gap

$$R[h] = R_{\mathcal{D}}[h] + (R[h] - R_{\mathcal{D}}[h])$$

Generalization Gap: How much empirical risk underestimates the risk?

Three Important Questions for Machine Learning

Representation

- What class of \mathcal{H} functions (classifiers) we should choose?

Optimization

- How can we efficiently solve the optimization problem?

Generalization

- Will the performance of a classifier generalize from training data to unseen data?

COGS514 - Cognition and Machine Learning

Neural Networks

History

- 1940s - model of neurons from McCulloch and Pitts and learning ideas of Hebb.
 - There was no way to learn multiple layers (only perceptron algorithm)
- 1980s - became popular again due to *back-propagation* algorithm
 - Training was slow, and gets stuck on *local-optima*
- 2010s - improvement in computation power, new approaches to avoid local optima

Neural Networks

Neural networks can be viewed as:

1. Application of stochastic gradient descent with a rich hypothesis class for regression and classification

Neural Networks

Neural networks can be viewed as:

1. Application of stochastic gradient descent with a rich hypothesis class for regression and classification
2. A brain-inspired network of neuron-like computing elements that learn distributed representations

Neural Networks

Neural networks can be viewed as:

1. Application of stochastic gradient descent with a rich hypothesis class for regression and classification
2. A brain-inspired network of neuron-like computing elements that learn distributed representations
3. Method for making predictions based on large amounts of data.

Variations of Neural Networks

- Feed-forward neural networks
- Convolutional neural networks
- Recurrent neural networks
- ...

Motivation: Linear Models vs Neural Networks

Linear Models

$$\hat{y} = \begin{cases} +1 & \text{if } \mathbf{w}^T \phi(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

In short we can write it as

$$\hat{y} = \text{sign}(\mathbf{w}^T \phi(\mathbf{x}))$$

Motivation: Linear Models vs Neural Networks

Linear Models

$$\hat{y} = \text{sign}(w^T \phi(x))$$

- We can map x to $\phi(x)$ by polynomial expansion to solve non-linear classification problems.
- However $\phi(x)$ mapping is fixed. It is **not optimized** for the task we are trying to solve.

Motivation: Linear Models vs Neural Networks

Linear Models

$$\hat{y} = \text{sign}(w^T \phi(x))$$

- We can map x to $\phi(x)$ by polynomial expansion to solve non-linear classification problems.
- However $\phi(x)$ mapping is fixed. It is **not optimized** for the task we are trying to solve.

Neural Networks

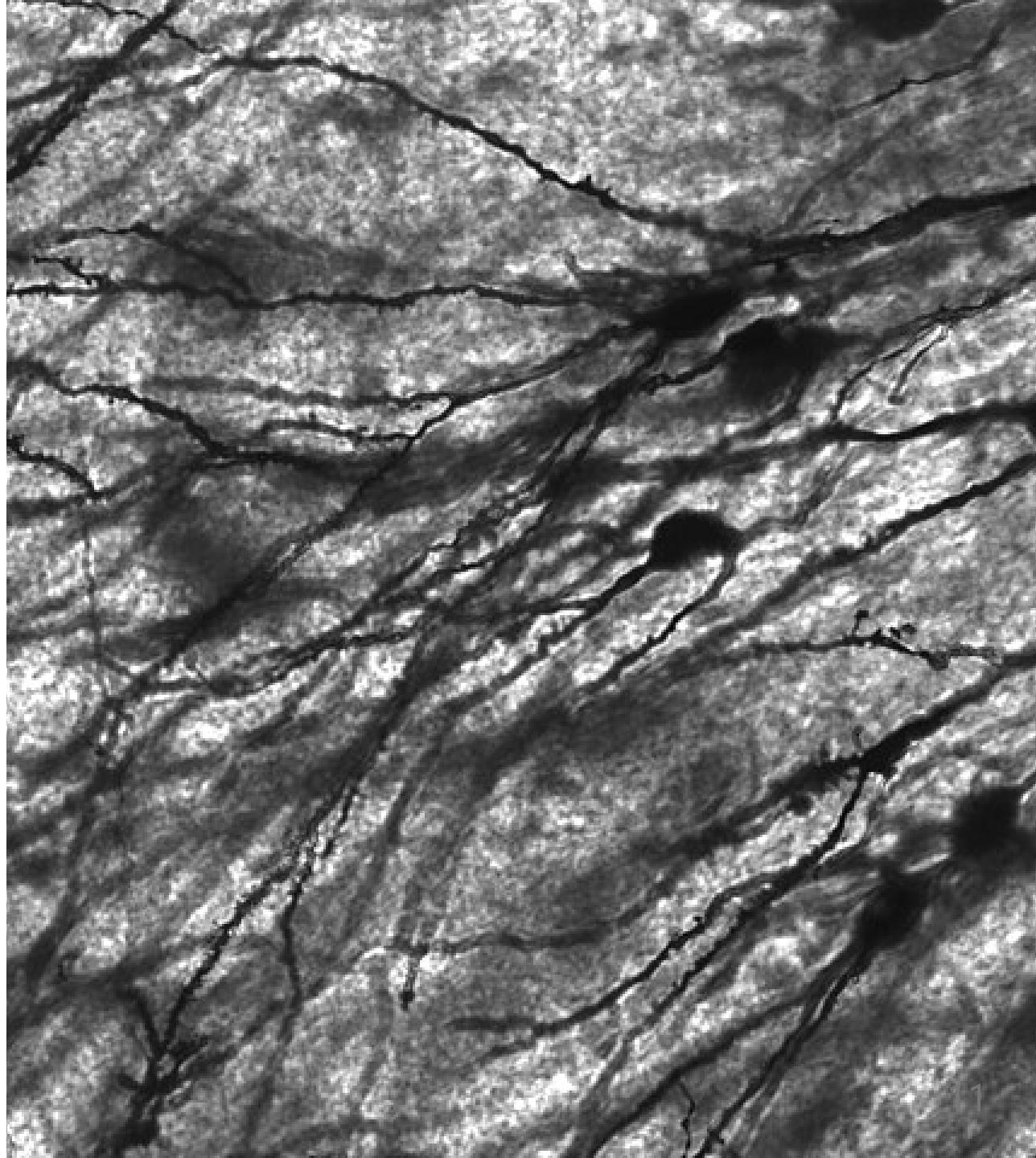
- Neural networks tries to optimize the feature representation (representation learning)

Connectionism

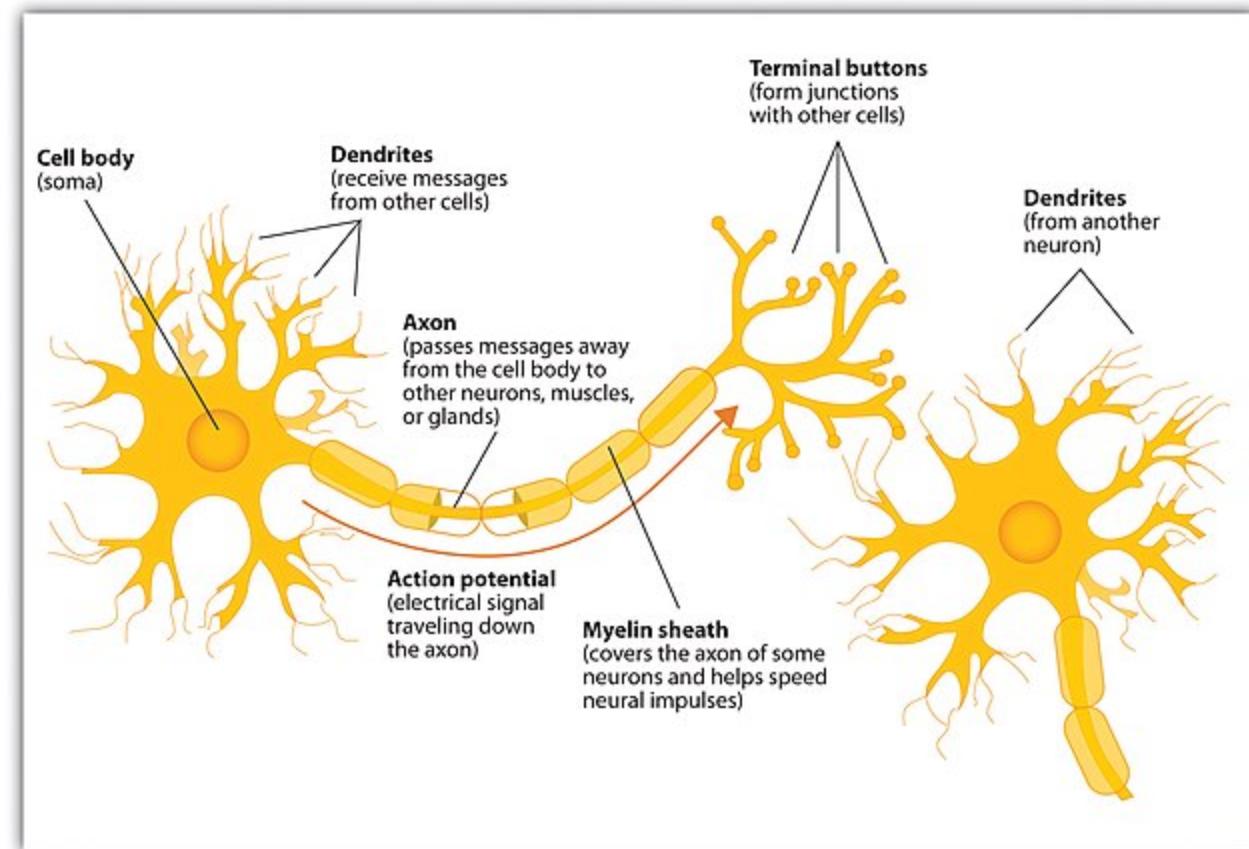
- Study of human cognition using neural networks

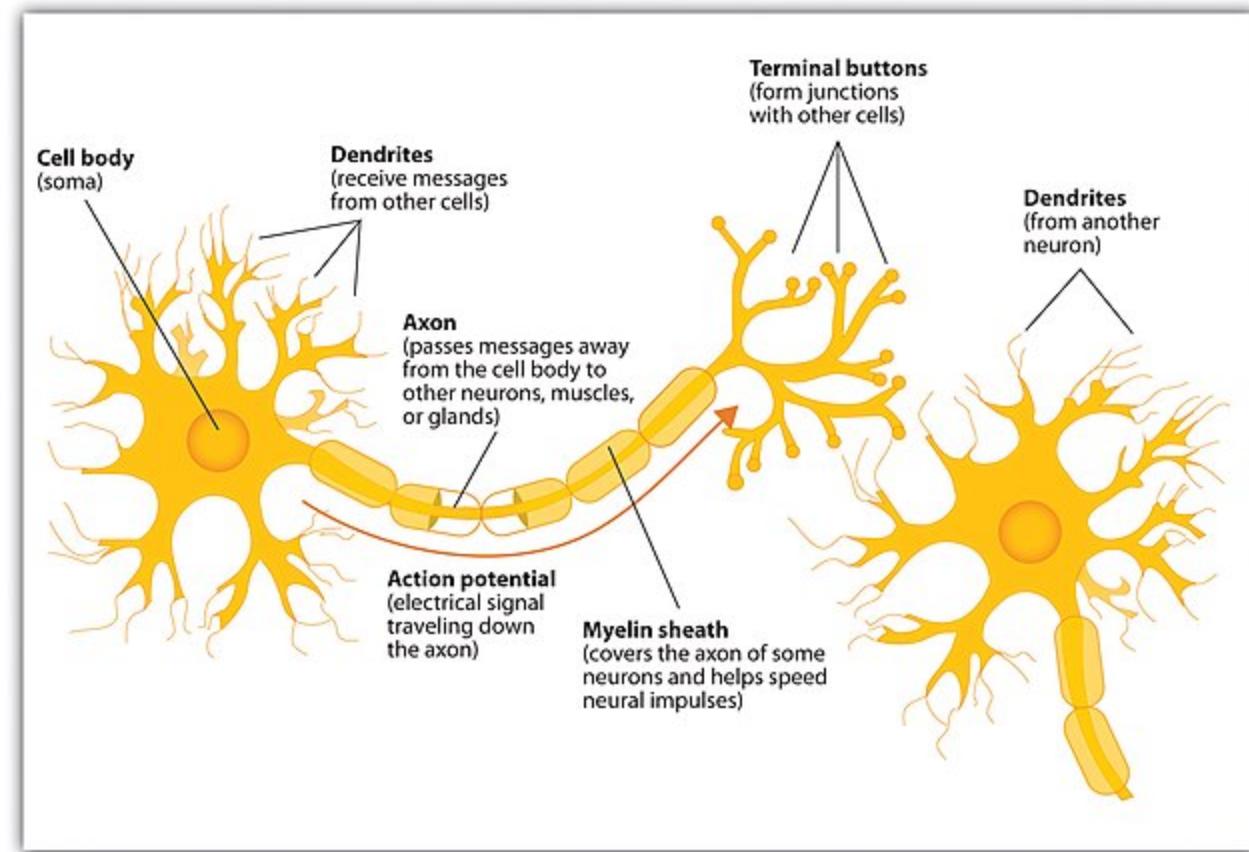
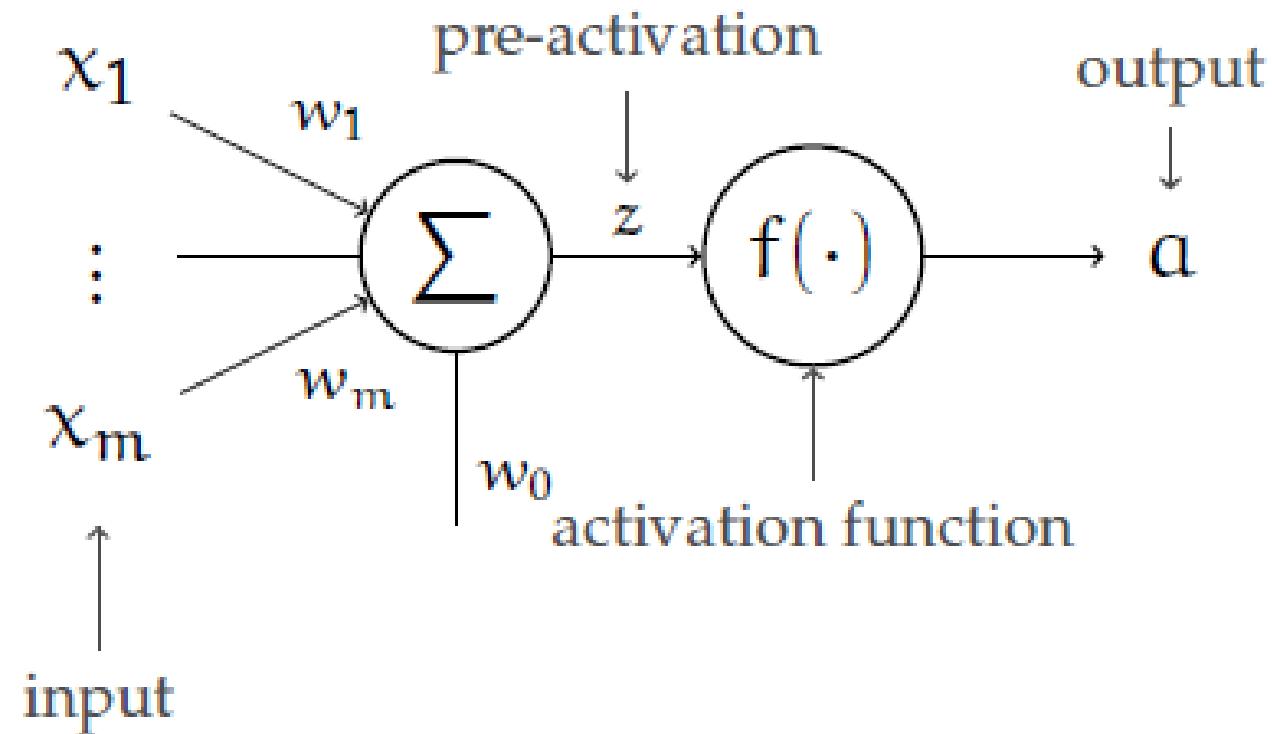
Biological Neural Networks

Image: Neurons in Human Hippocampal Tissue. Source: [Wikipedia](#)



Biological Neurons



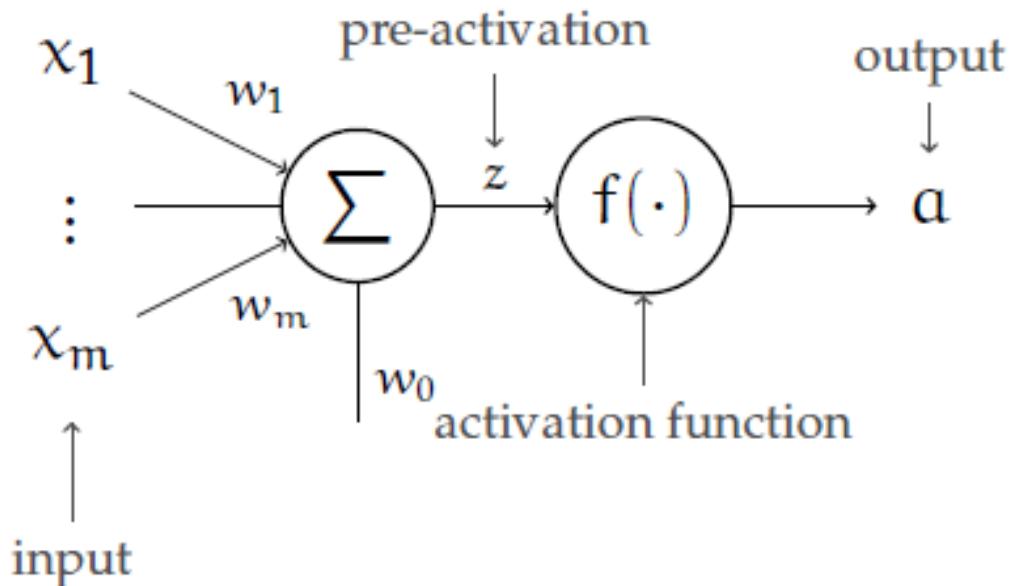


Basic Element

Neuron - Unit - Node

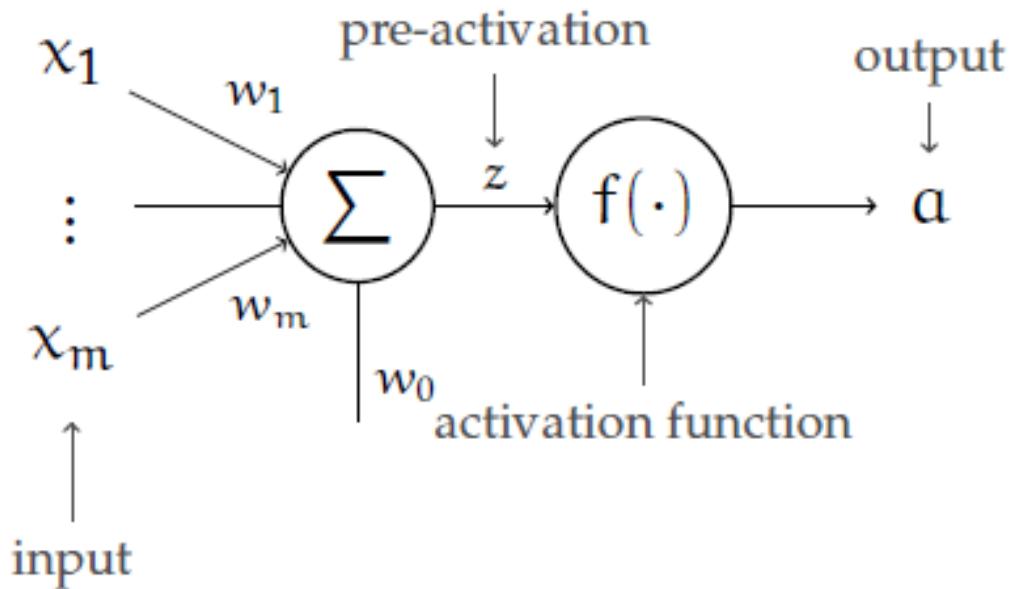
A non-linear function of

- an input vector $x \in \mathbb{R}^m$
 - to a single output $a \in \mathbb{R}$.
- parameterised by
- weights $[w_1, \dots, w_m]^T \in \mathbb{R}^m$,
 - and an *offset* or *threshold* $w_0 \in \mathbb{R}$.
 - bias term



Basic Element

- an input vector $x \in \mathbb{R}^m$
- to a single output $a \in \mathbb{R}$. parameterised by
- weights $[w_1, \dots, w_m]^T \in \mathbb{R}^m$,
- and an *offset* or *threshold* $w_0 \in \mathbb{R}$.
- to introduce non-linear behavior, an *activation function* $f : \mathbb{R} \rightarrow \mathbb{R}$.
 - If these neurons are all linear, neural networks does not lead to an interesting behavior (it works in a similar way to linear regression etc.).

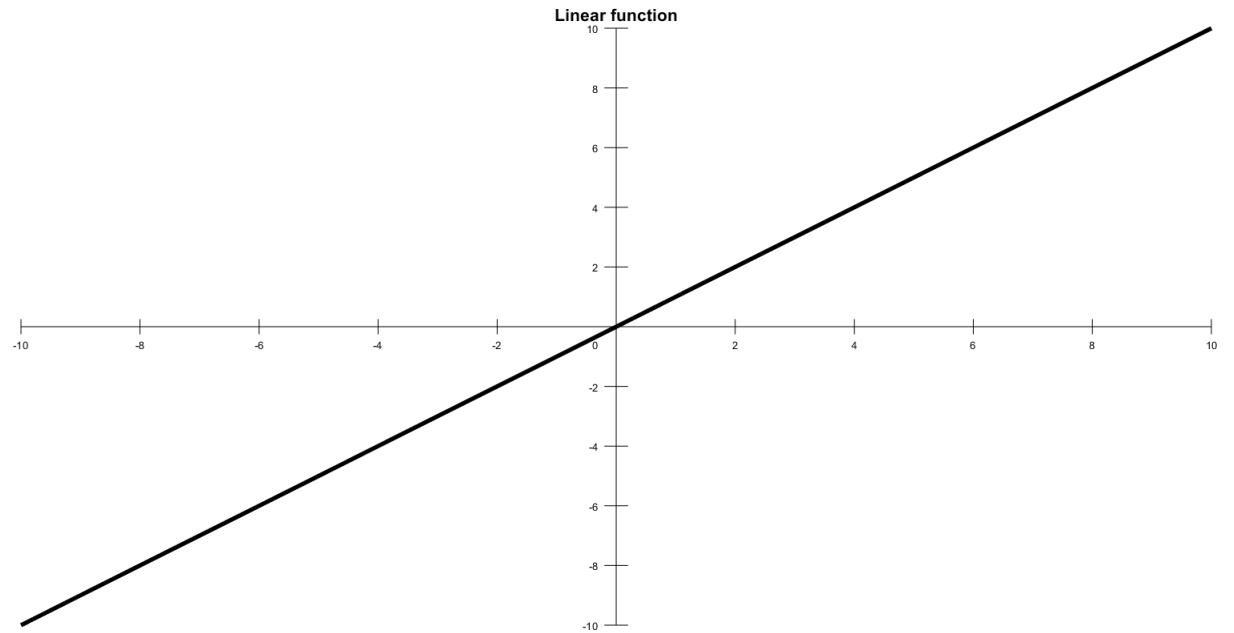


What could be the non-linear activation function $f(\cdot)$?

Activation Functions

Linear (Identity)

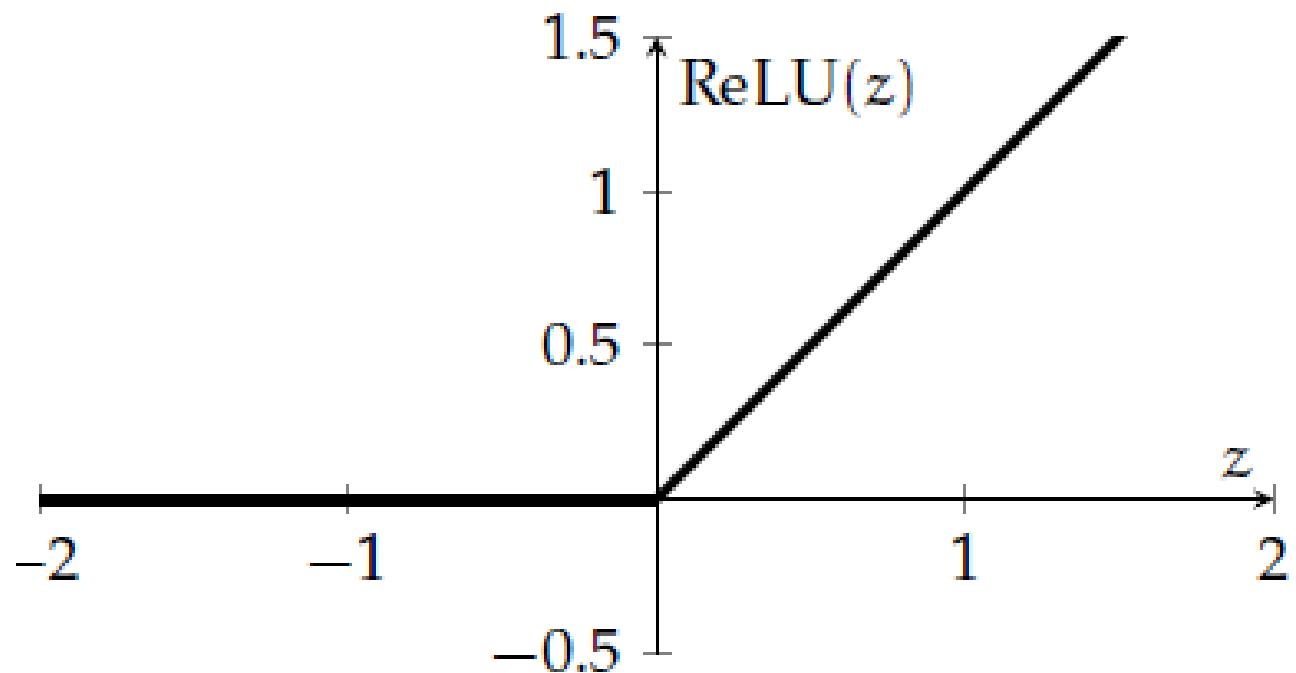
$$f(z) = z$$



Non-Linear Activation Functions

Rectified linear unit

$$\text{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} = \max(0, z)$$

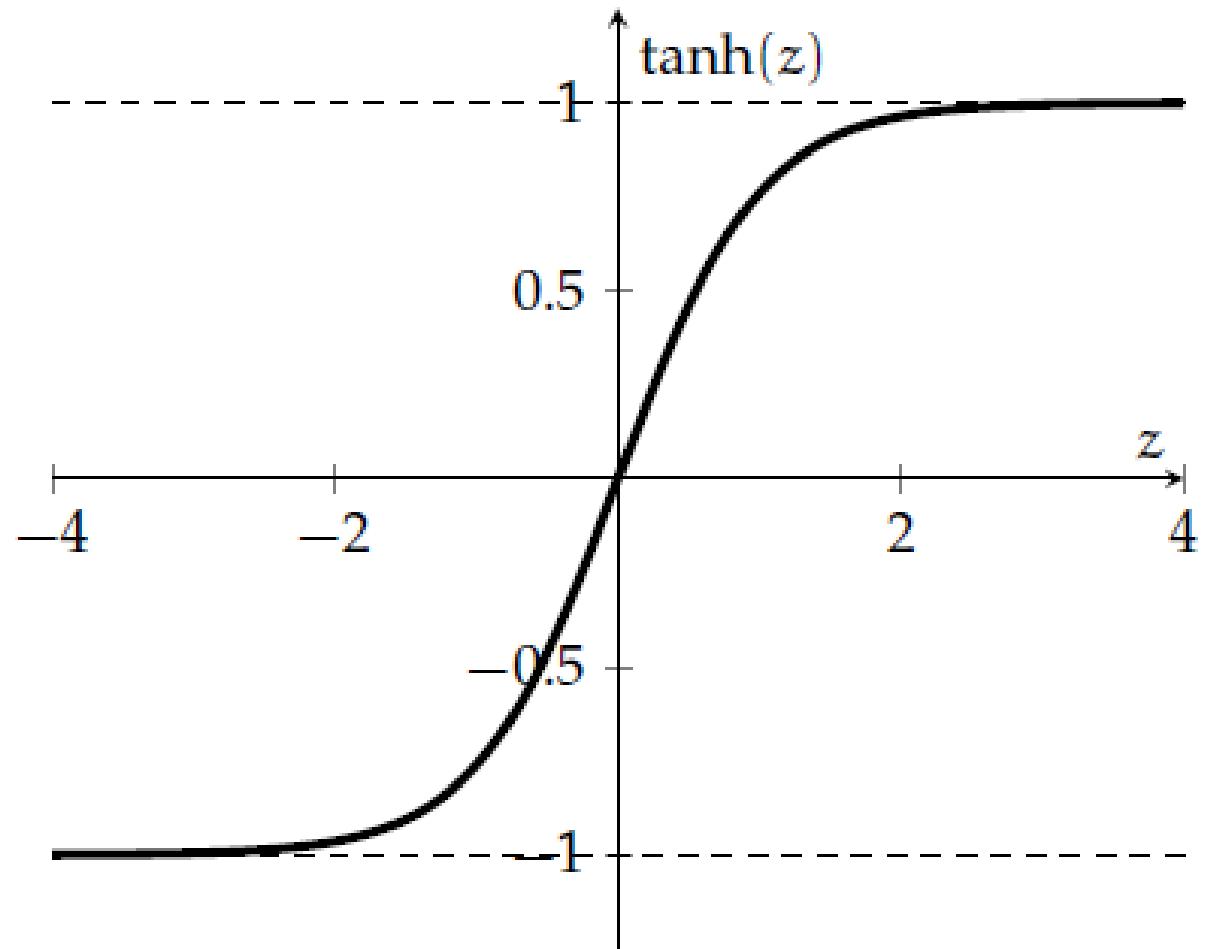


Non-Linear Activation Functions

Hyperbolic Tangent

always in the range $[-1, 1]$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

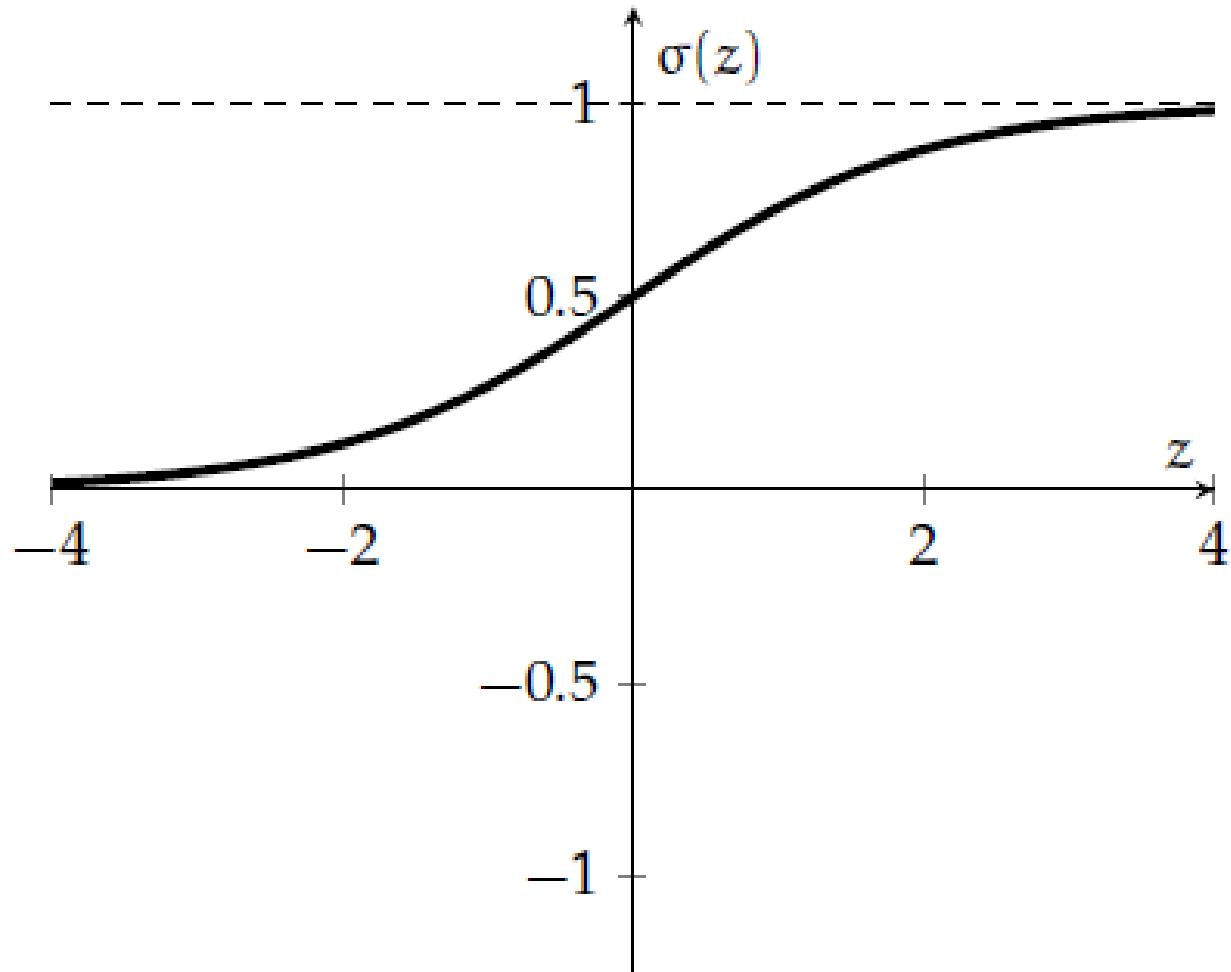


Non-Linear Activation Functions

Sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

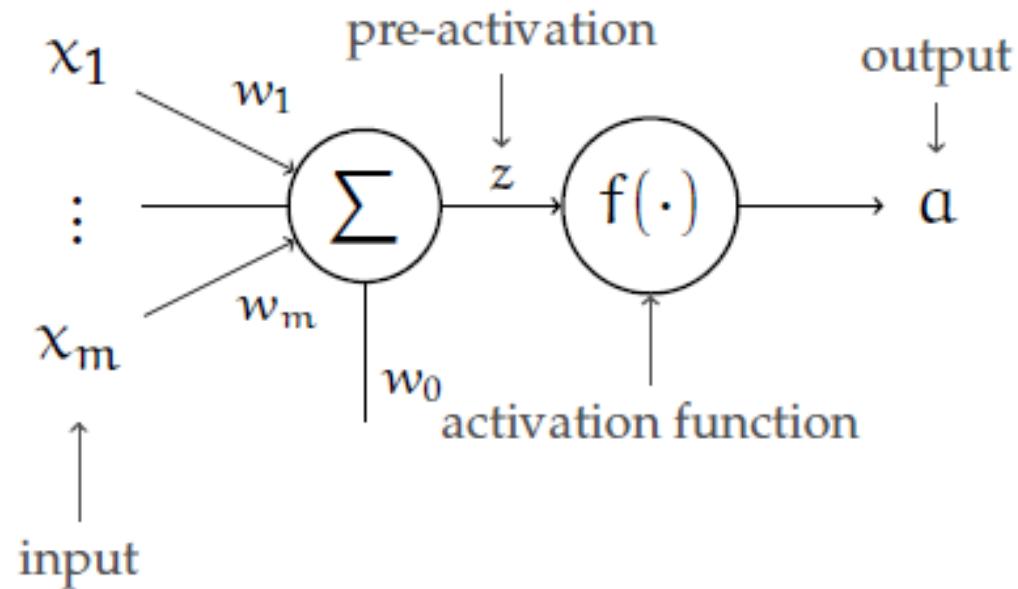
also known as *logistic* function. This can be interpreted as a probability because the output is in $[0, 1]$



Basic Element

The function represented by the neuron is as follows

$$a = f(z) = f(w^T x + w_0) = f \left(\sum_{j=1}^m w_j x_j + w_0 \right)$$



Learning a Single Neuron

Learning a Single Neuron

Given a loss function $loss(guess, actual)$ and a dataset

$$\mathcal{D} = \left\{ (x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \right\}$$

we can do (stochastic) gradient descent, adjusting weights w and w_0 to minimize:

$$\Phi(w, w_0) = \sum_i loss \left(NN(x^{(i)}; w, w_0), y^{(i)} \right)$$

Learning a Single Neuron

Given a loss function $loss(guess, actual)$ and a dataset

$$\mathcal{D} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$$

we can do (stochastic) gradient descent, adjusting weights w and w_0 to minimize:

$$\Phi(w, w_0) = \sum_i loss \left(NN(x^{(i)}; w, w_0) y^{(i)} \right)$$

NN is the output of the neural network for a given input

- $loss$ can be *quadratic loss* for *regression* or *negative log likelihood* for *classification*
- Activation function is $f(x) = x$ for regression and $f(x) = \sigma(x)$ for classification

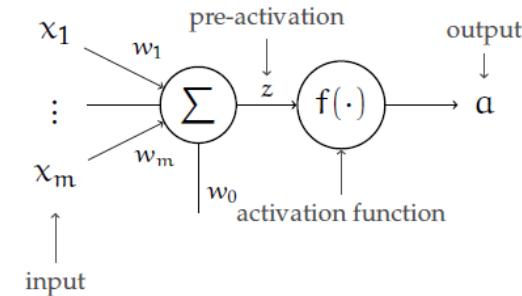
Loss and Activation Functions

$$z = w^T x + w_0$$

Regression

- Activation Function (Identity) $f(z) = z$
- $a = f(z) = w^T x + w_0$
- Loss function can be

$$\text{loss}(\text{guess}, \text{actual}) = (\text{actual} - \text{guess})^2$$



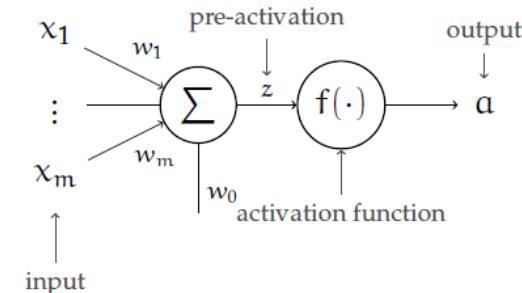
Loss and Activation Functions

$$z = w^T x + w_0$$

Classification

- Activation Function (Sigmoid) $f(z) = \frac{1}{1+e^{-z}}$
- $a = f(z) = \sigma(w^T x + w_0) = \frac{1}{1+e^{-(w^T x + w_0)}}$
- Loss function can be

$$\text{loss}_{\text{nll}}(\text{guess}, \text{actual}) = -(\text{actual} \cdot \log(\text{guess}) + (1 - \text{actual}) \cdot \log(1 - \text{guess}))$$

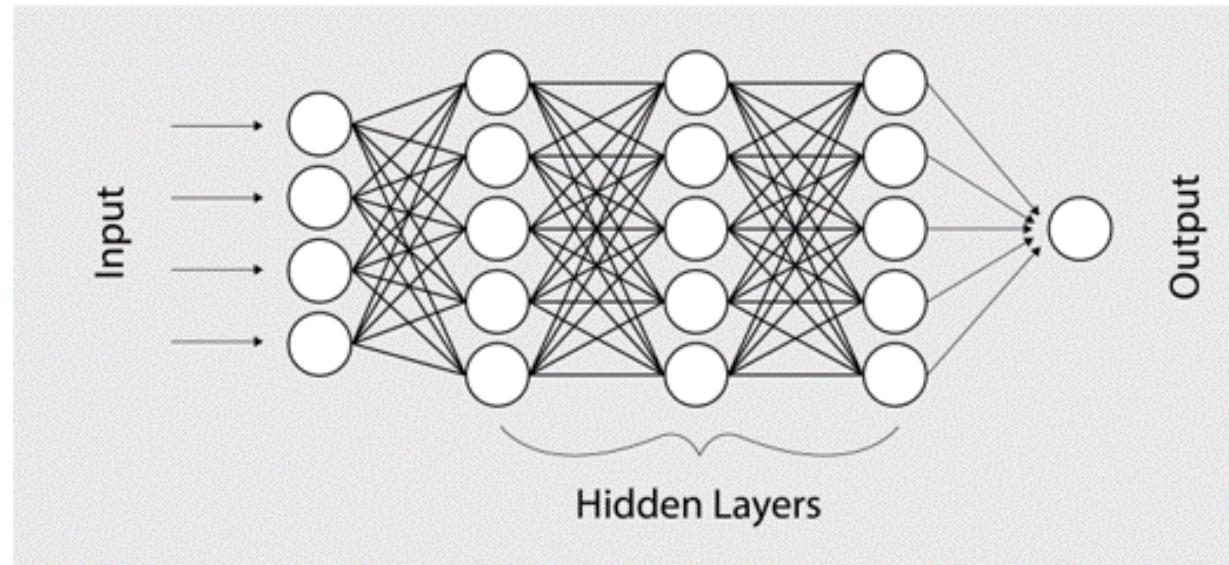


Neural Networks

- A *neural network* is composed of multiple neurons.
- Inputs of a neuron can be x or outputs of other neurons a

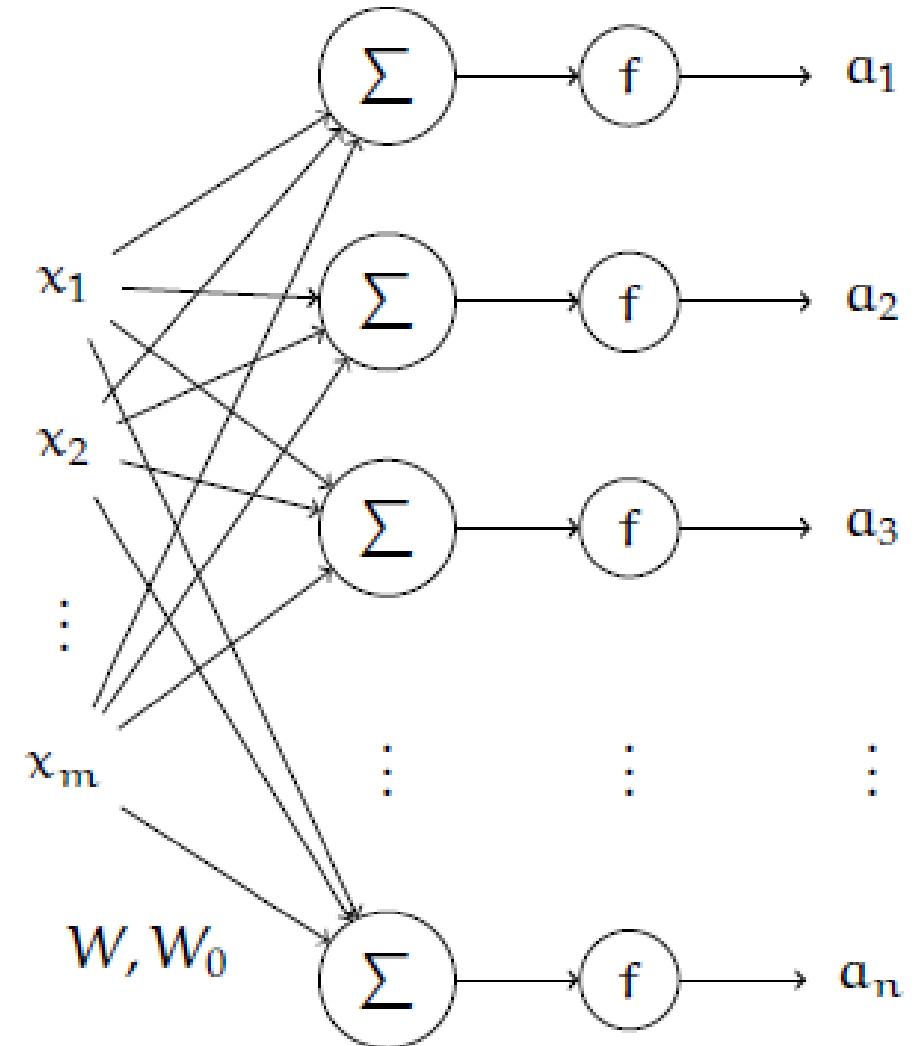
Feed-forward Neural Networks

- Data flows one-way: *Inputs* → *Outputs*
- Inputs of a neuron in a *feed-forward neural networks* cannot depend on its or its descendant's outputs.
- The structure of feed-forward NNs are usually built in **layers**.



A Layer

- **Layer:** A group of aligned neurons :
 - They are not connected to each other
 - Their *inputs* are the outputs of neurons from the *previous layer*
 - Their *outputs* are inputs to the neurons in the *next layer*.
- A layer is *fully connected* if the inputs to each unit in the layer are the same.



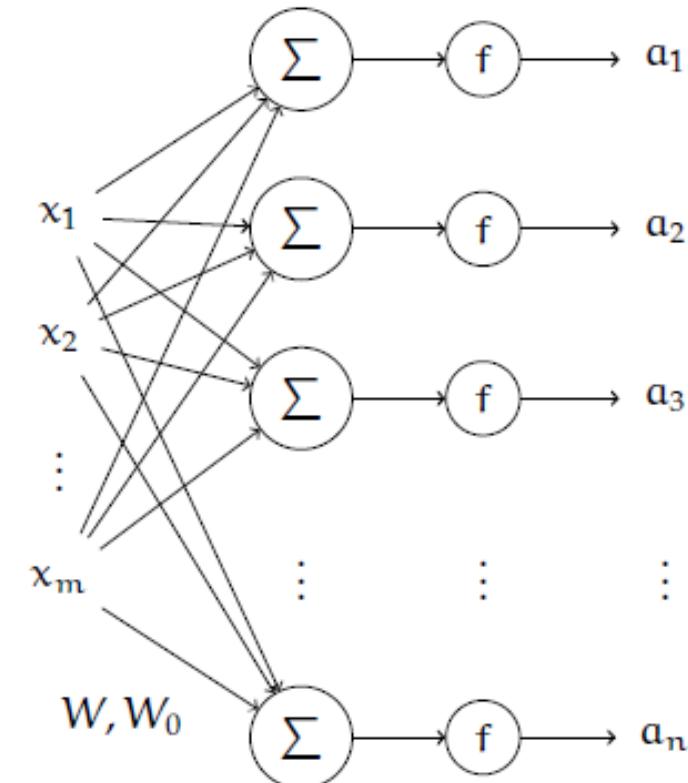
A Layer

A layer has

- input $x \in \mathbb{R}^m$
- an output (activation) $a \in \mathbb{R}^n$.

Since each unit is has a vector of weights and a single offset, we can think of

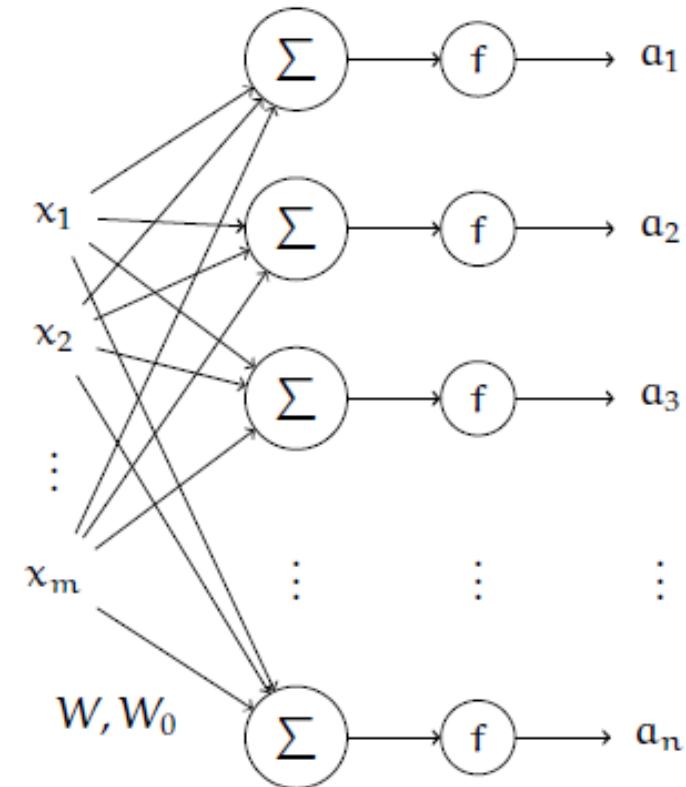
- the weights of the whole layer as a matrix W , and
- the collections of all offsets as a vector w_0 .



A Layer

If we have m inputs and n units and n outputs, then

- W is an $m \times n$ matrix



A Layer

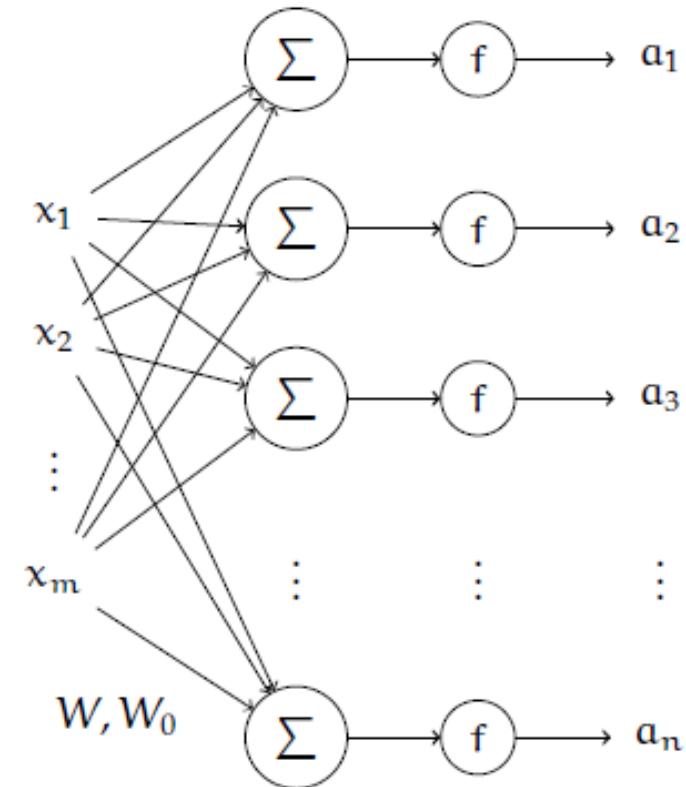
If we have m inputs and n units and n outputs, then

- W is an $m \times n$ matrix
- x is an $m \times 1$ vector

A Layer

If we have m inputs and n units and n outputs, then

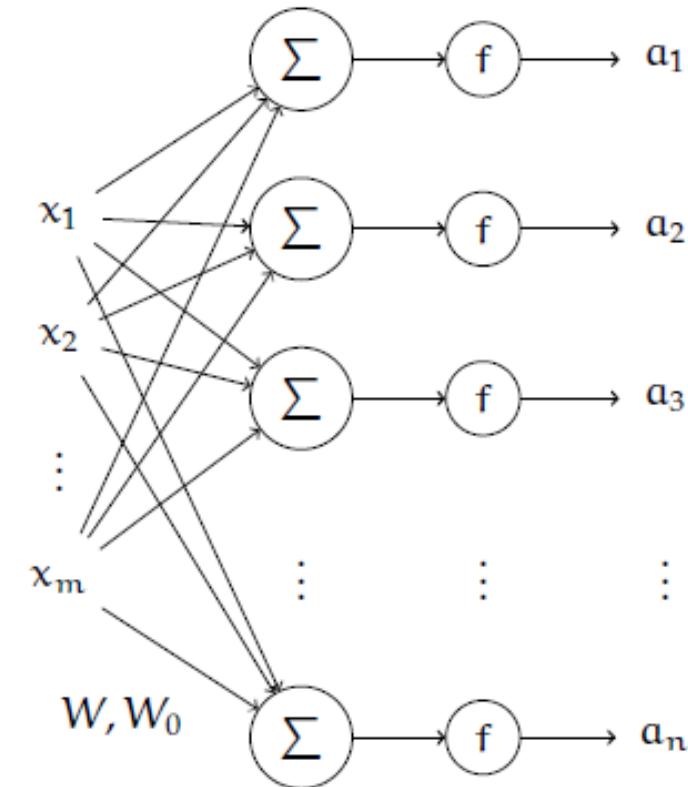
- W is an $m \times n$ matrix
- x is an $m \times 1$ vector
- w_0 is a $n \times 1$ vector



A Layer

If we have m inputs and n units and n outputs, then

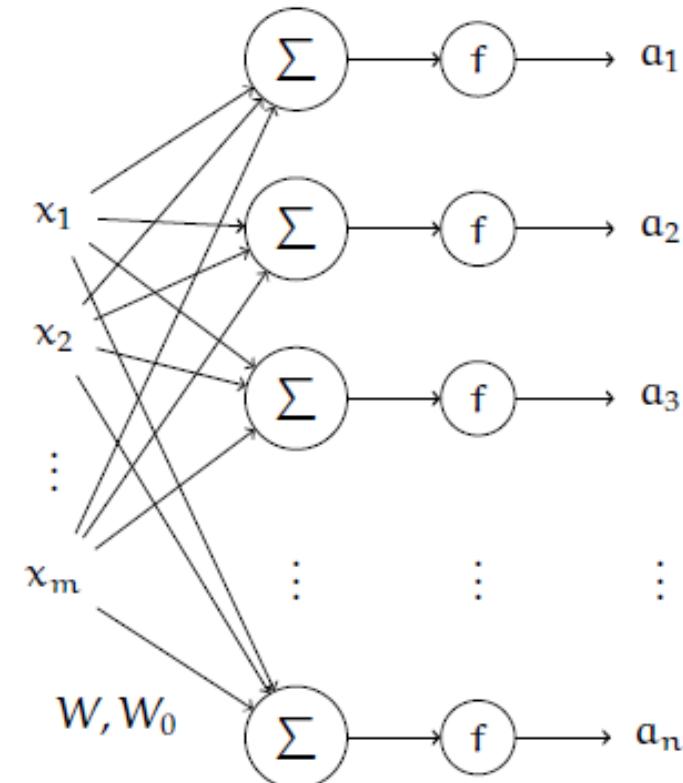
- W is an $m \times n$ matrix
- x is an $m \times 1$ vector
- w_0 is a $n \times 1$ vector
- $z = W^t x + w_0$ is an $n \times 1$ vector (called *pre-activation*)



A Layer

If we have m inputs and n units and n outputs, then

- W is an $m \times n$ matrix
- x is an $m \times 1$ vector
- w_0 is a $n \times 1$ vector
- $z = W^T x + w_0$ is an $n \times 1$ vector (called *pre-activation*)
- a is an $n \times 1$ vector (called *activation*)



A Layer

The output vector is

$$a = f(z) = f(W^T x + w_0)$$

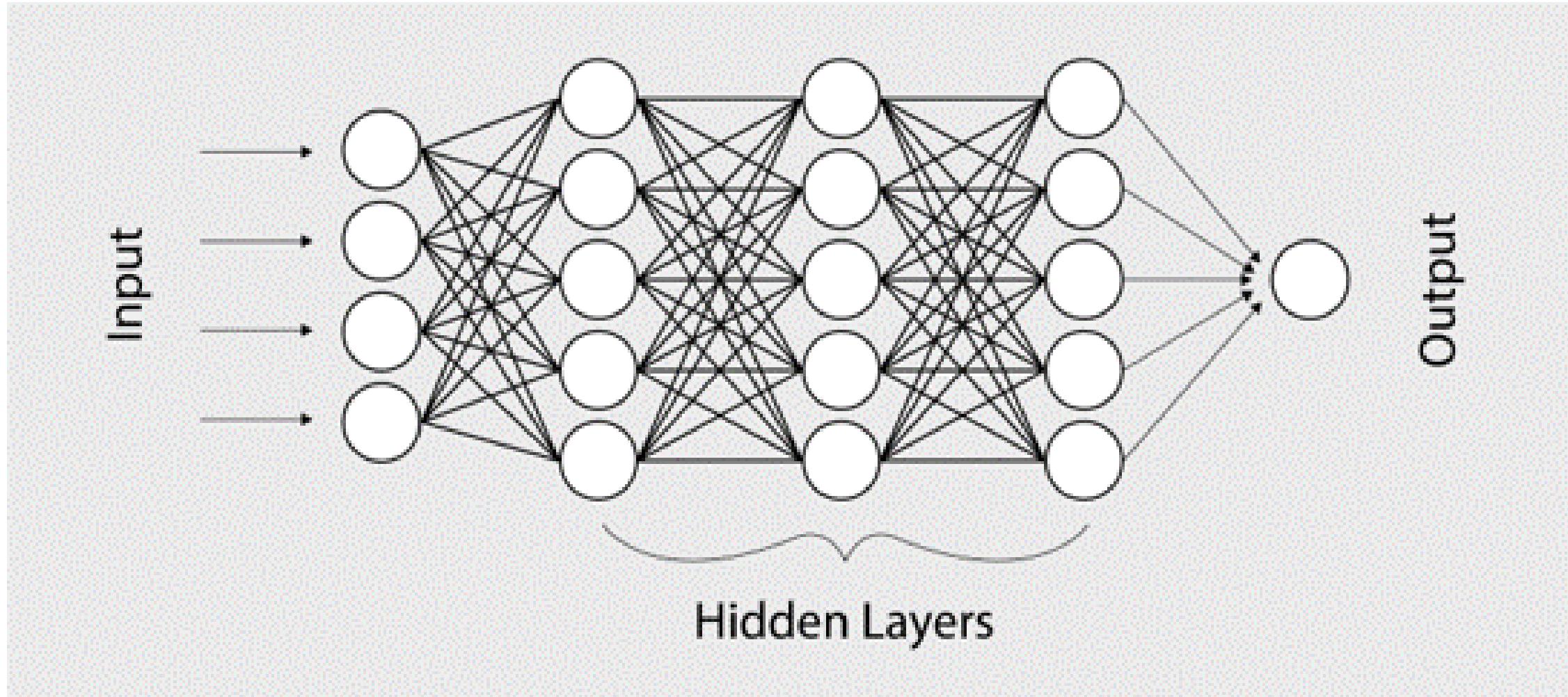
The activation function f is applied element-wise to the pre-activation values z .

What can be done with a single layer?

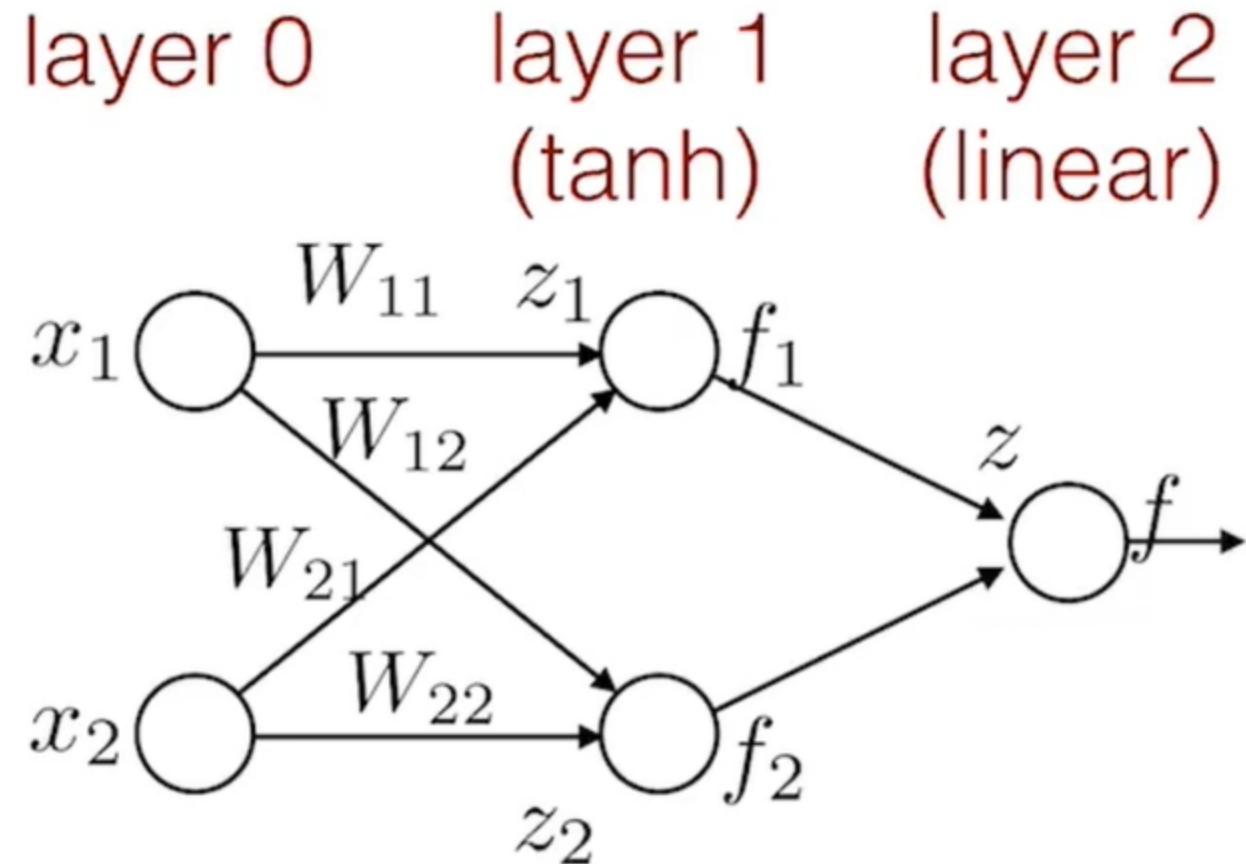
- All we can do with a single layer is a **linear hypothesis**.
 - In linear models, we used a *step* or *sigmoid* but the resulting separator was still linear.
- The whole point of using neural networks is to make *non-linear* hypotheses. We need to consider *multiple layers* for this.

Deep Neural Networks

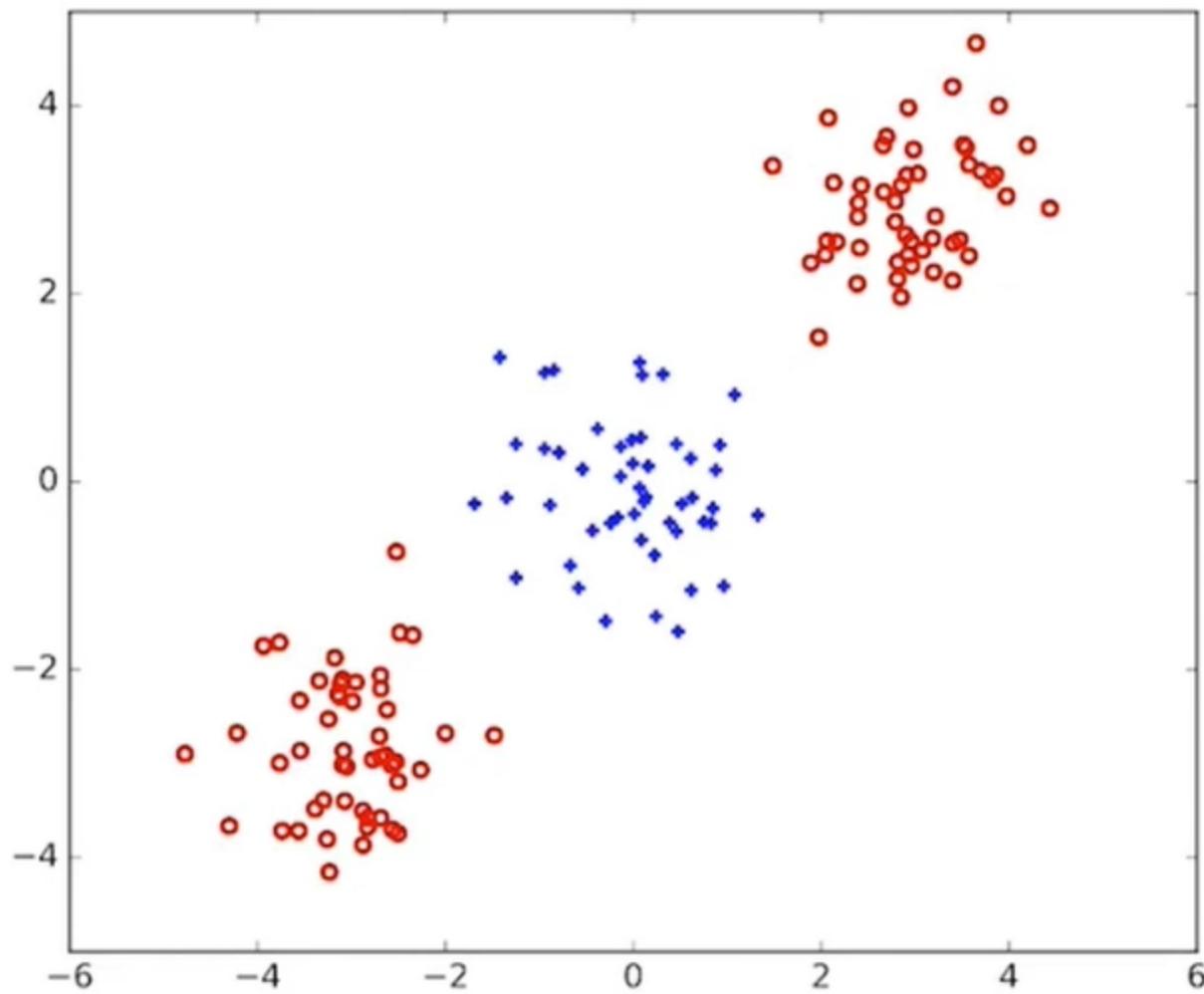
Neural Networks with Multiple Layers



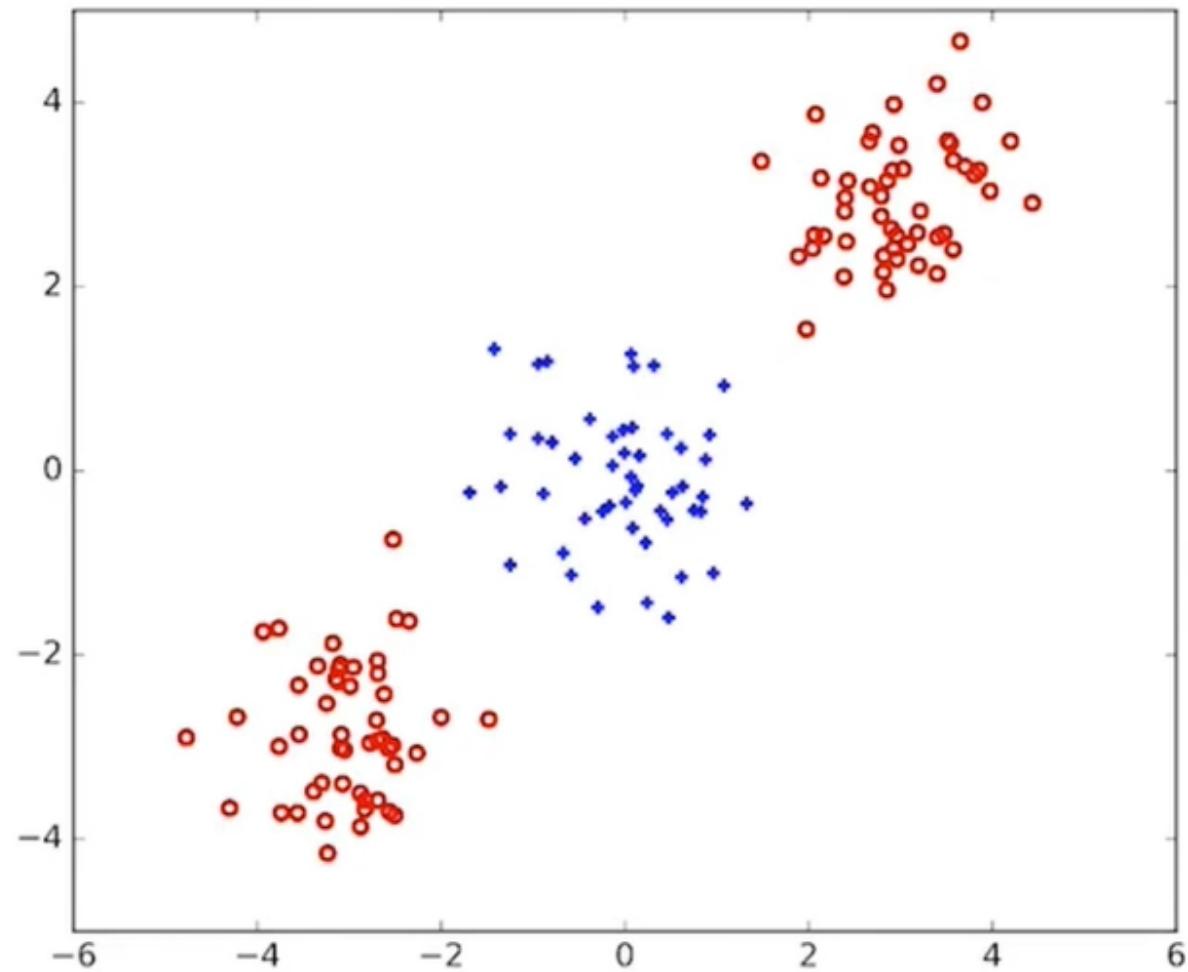
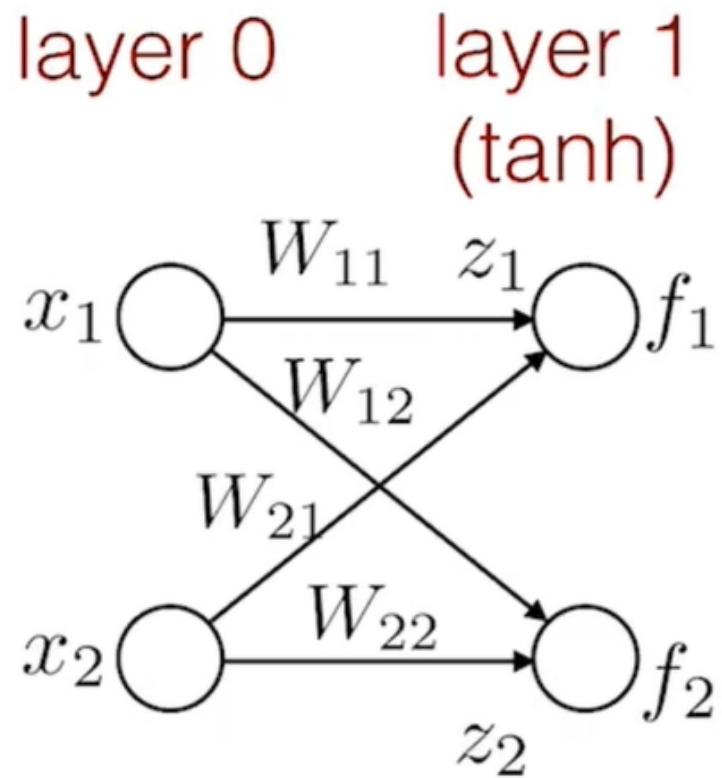
Hidden Layers - Example



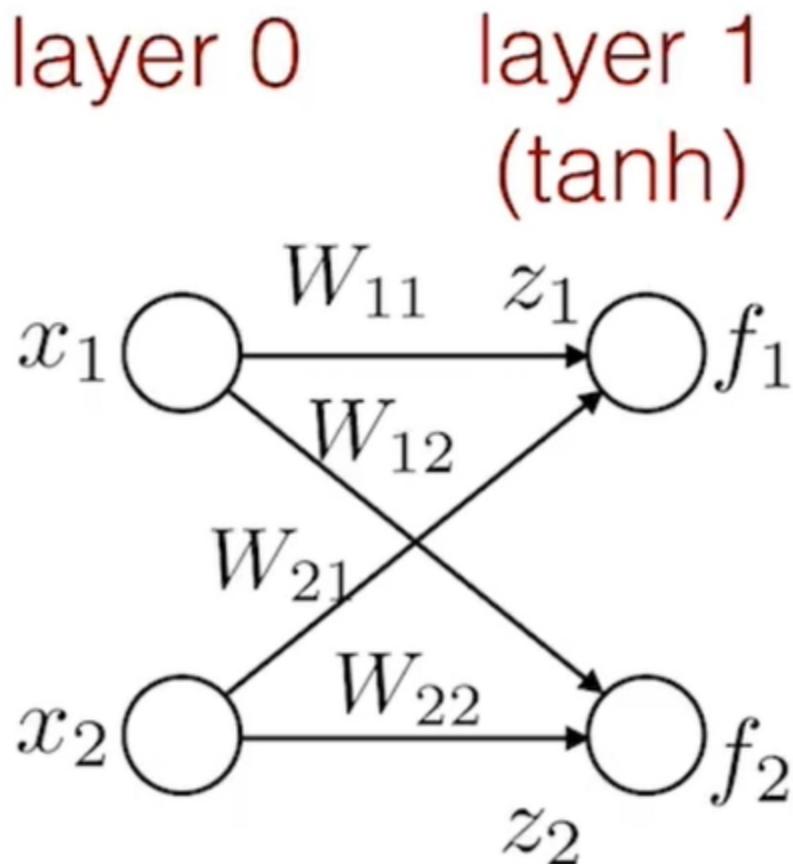
Example Problem



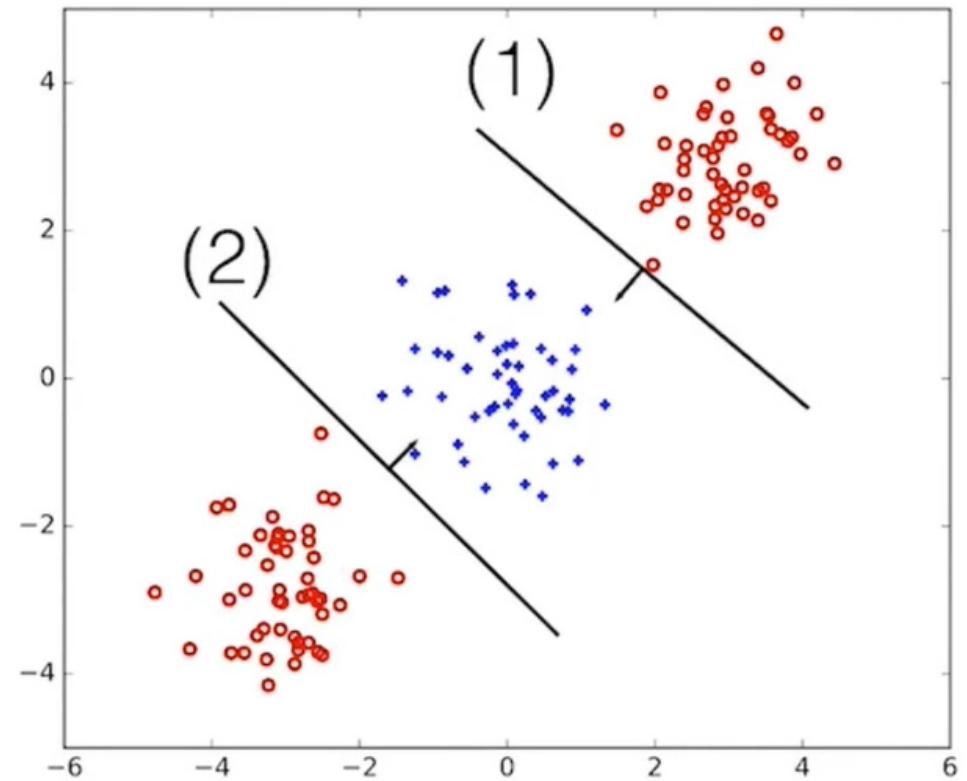
Example Problem



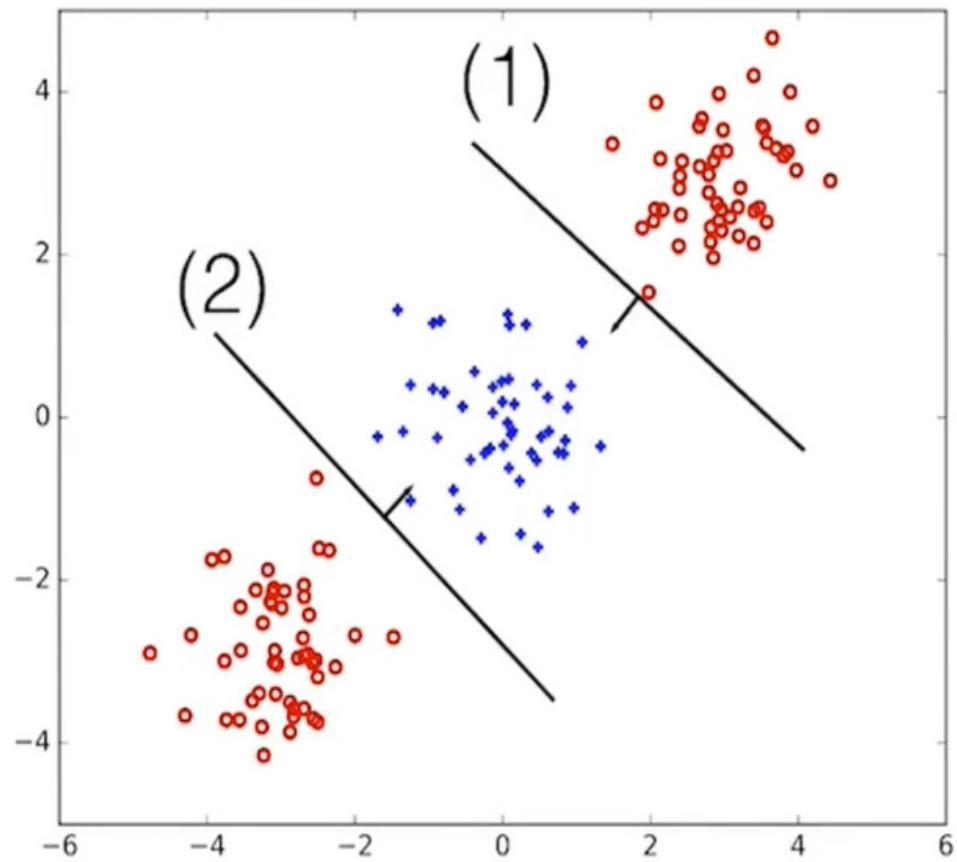
Example Problem



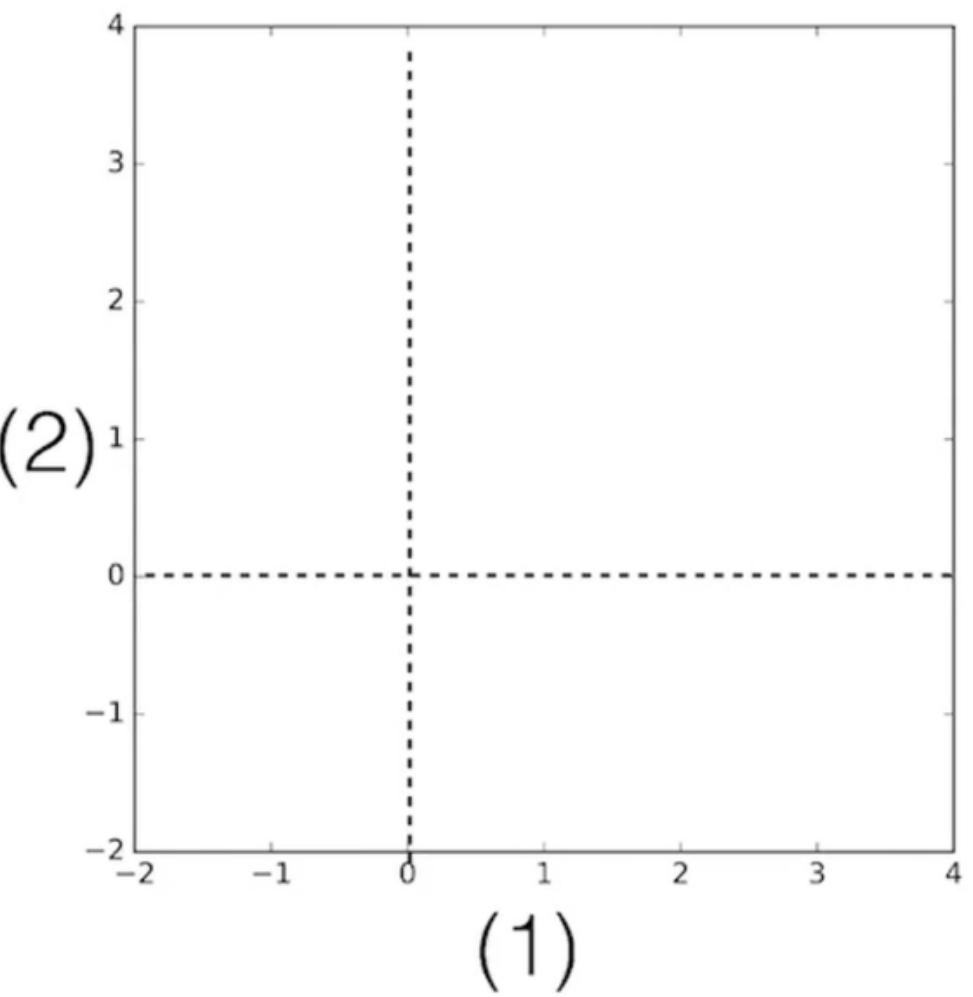
Hidden layer units



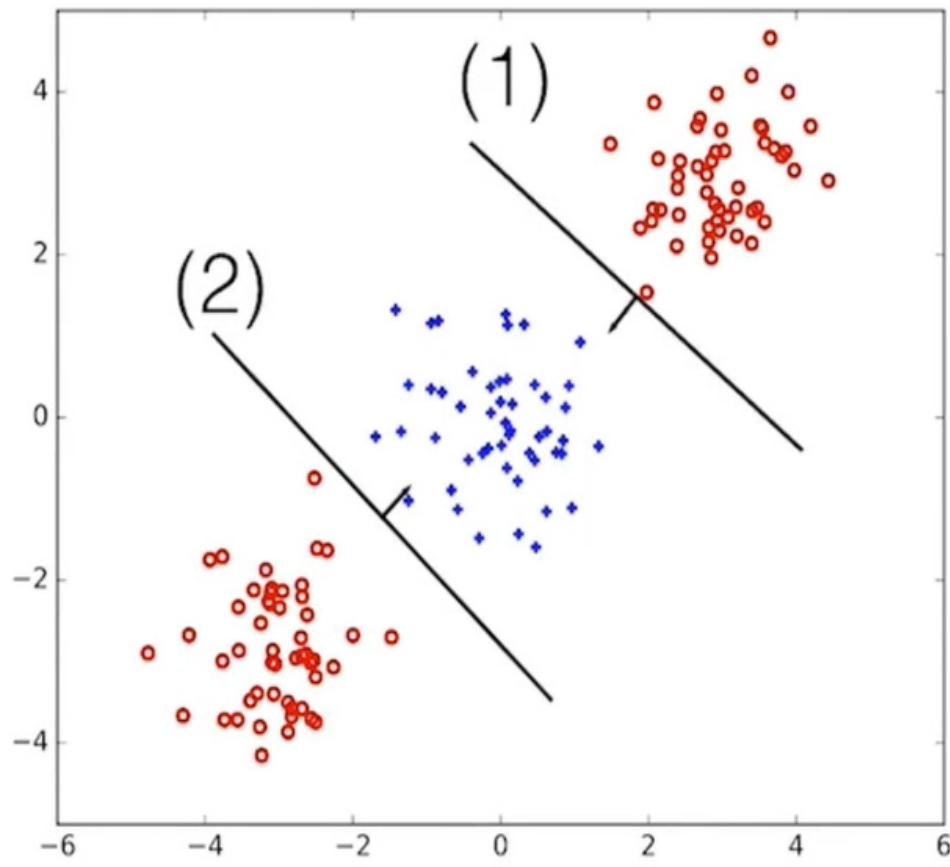
Hidden layer units



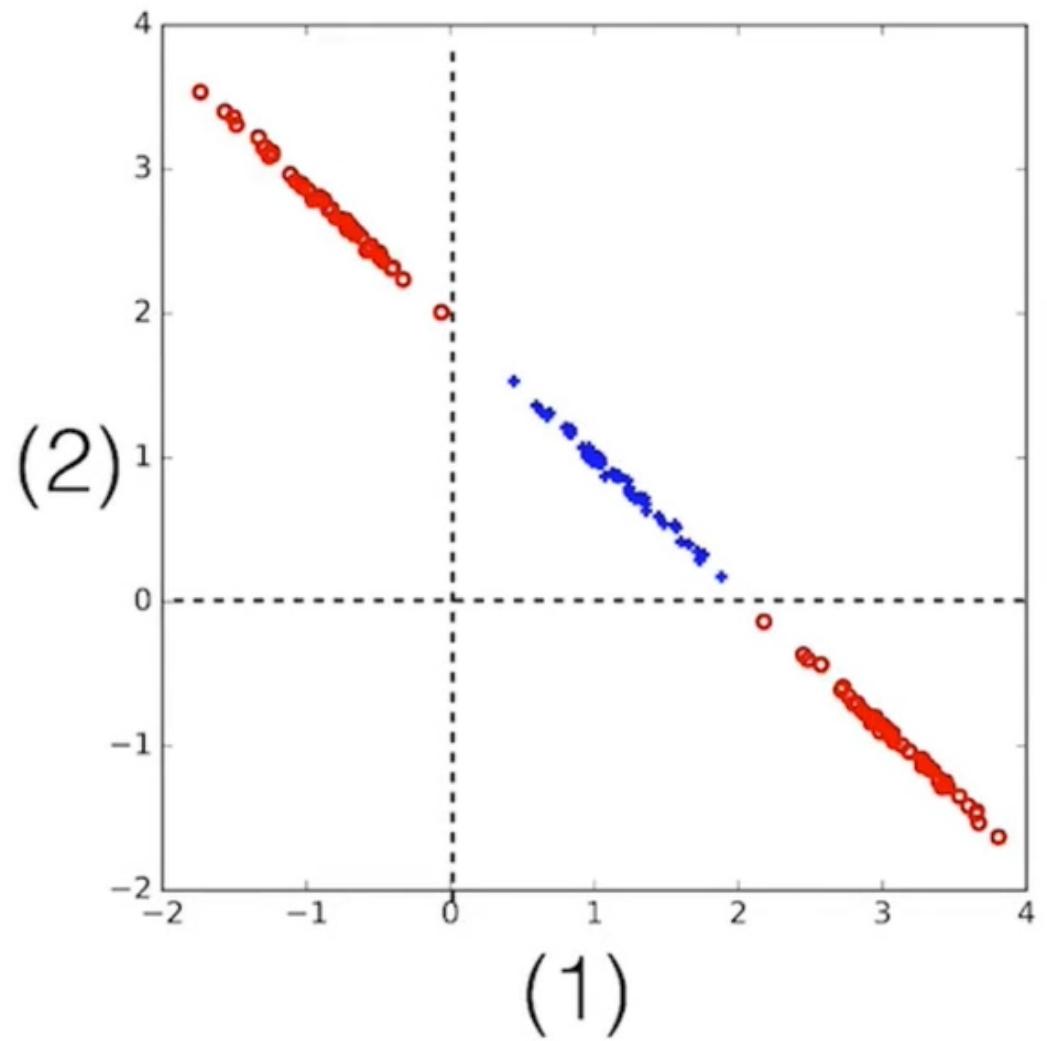
Linear activation



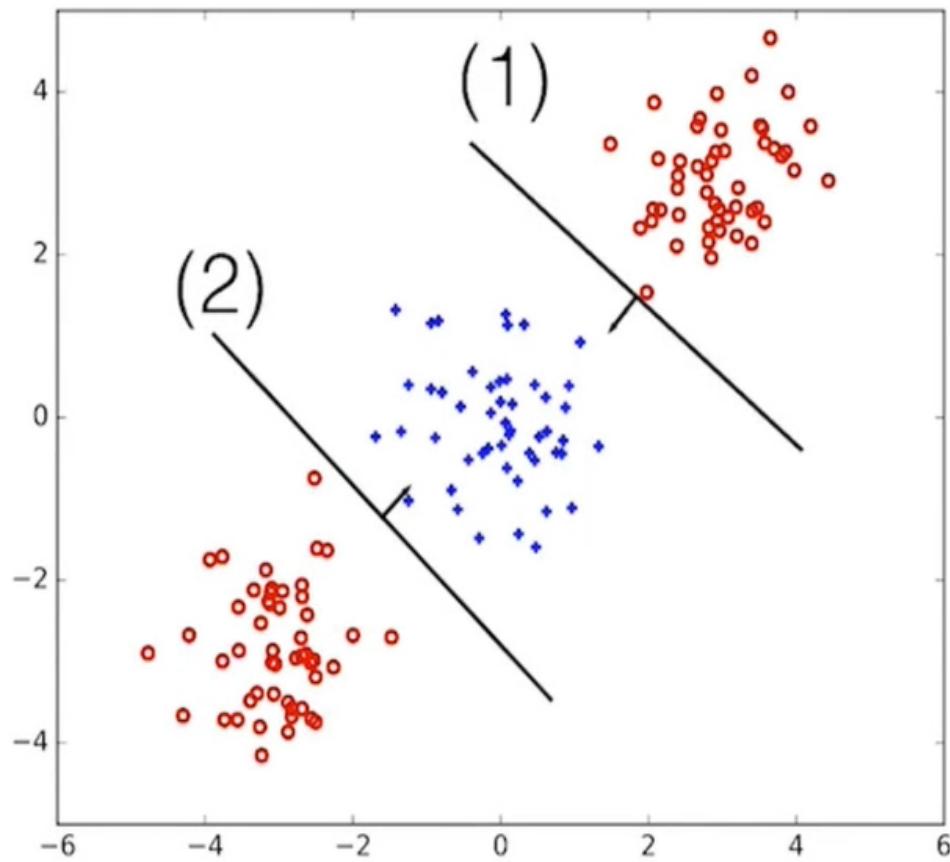
Hidden layer units



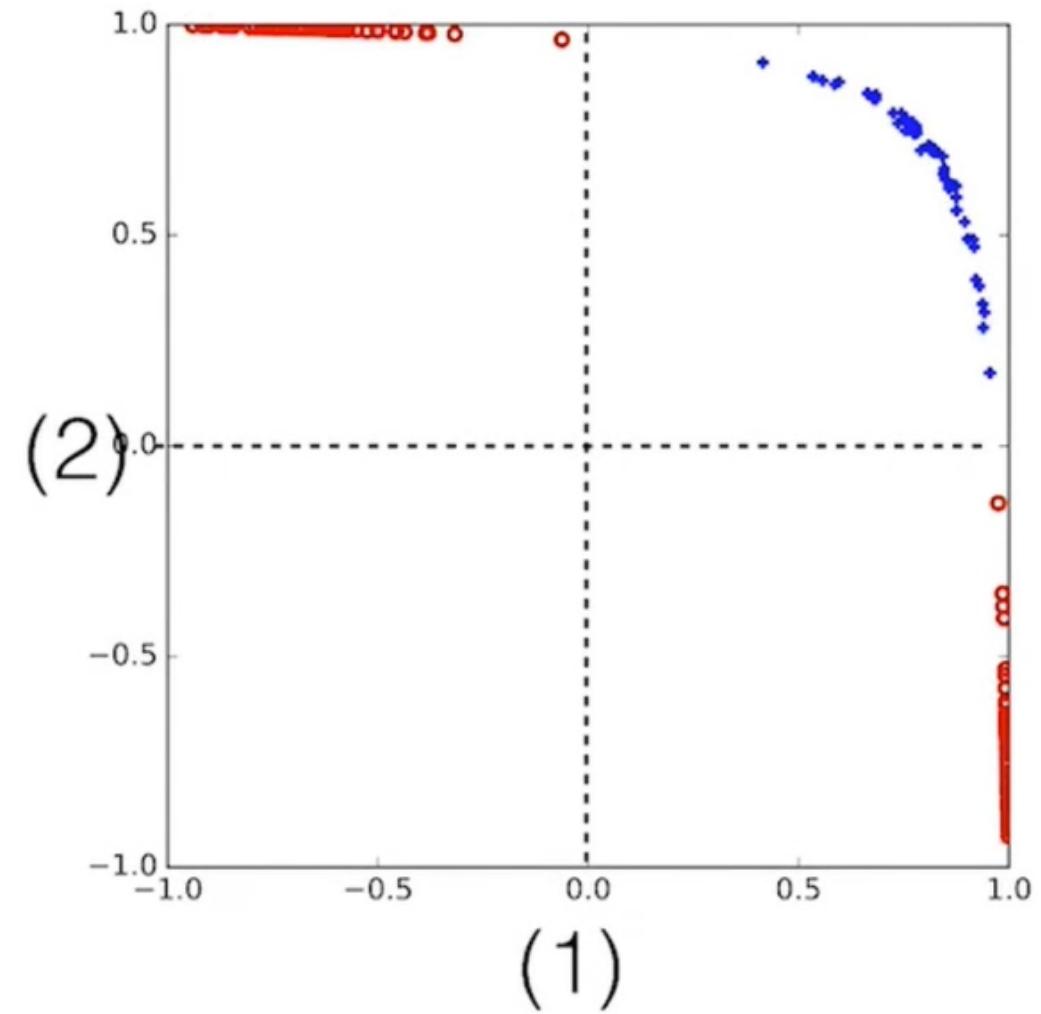
Linear activation



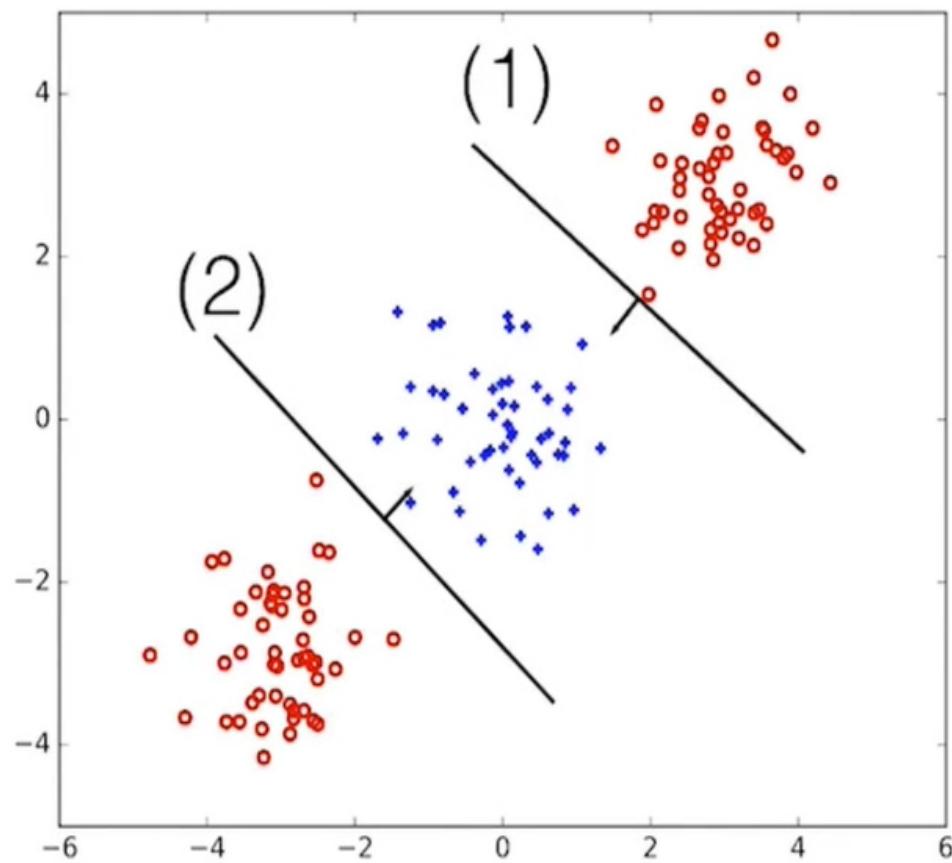
Hidden layer units



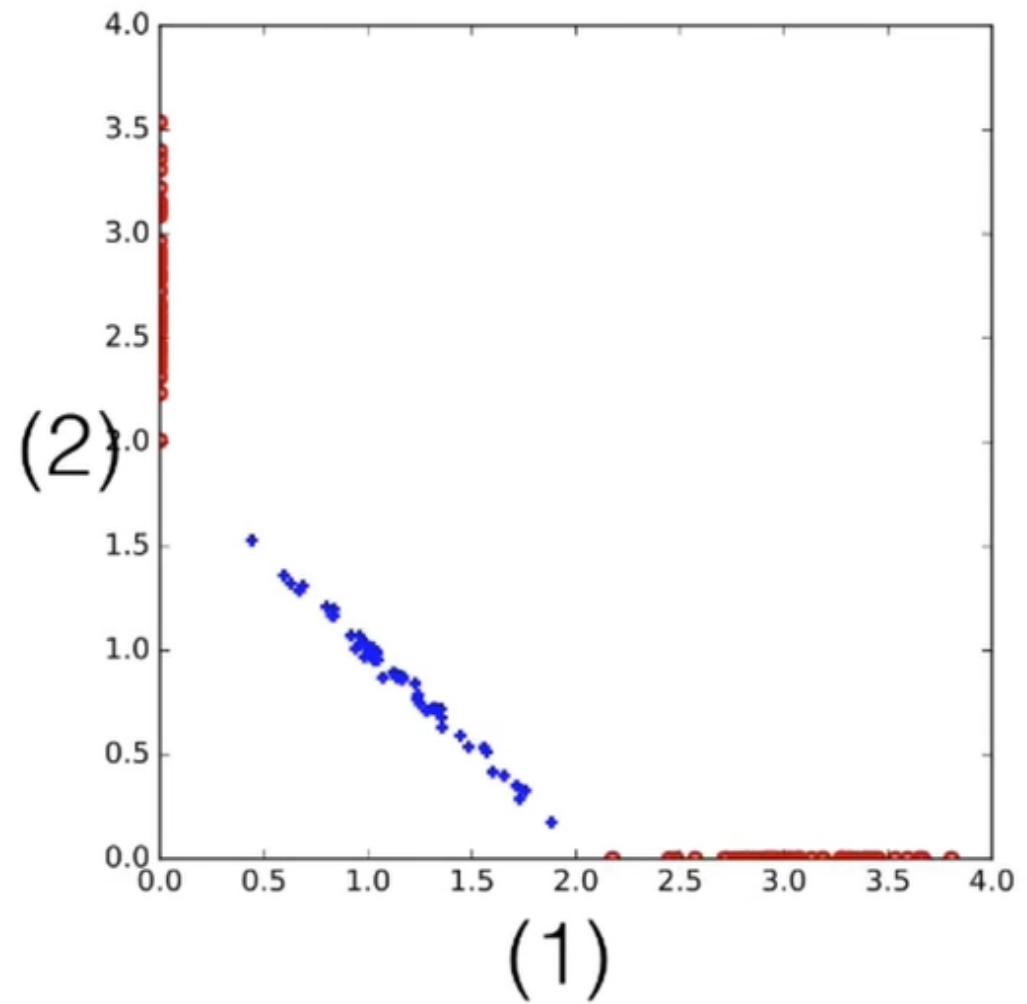
tanh activation



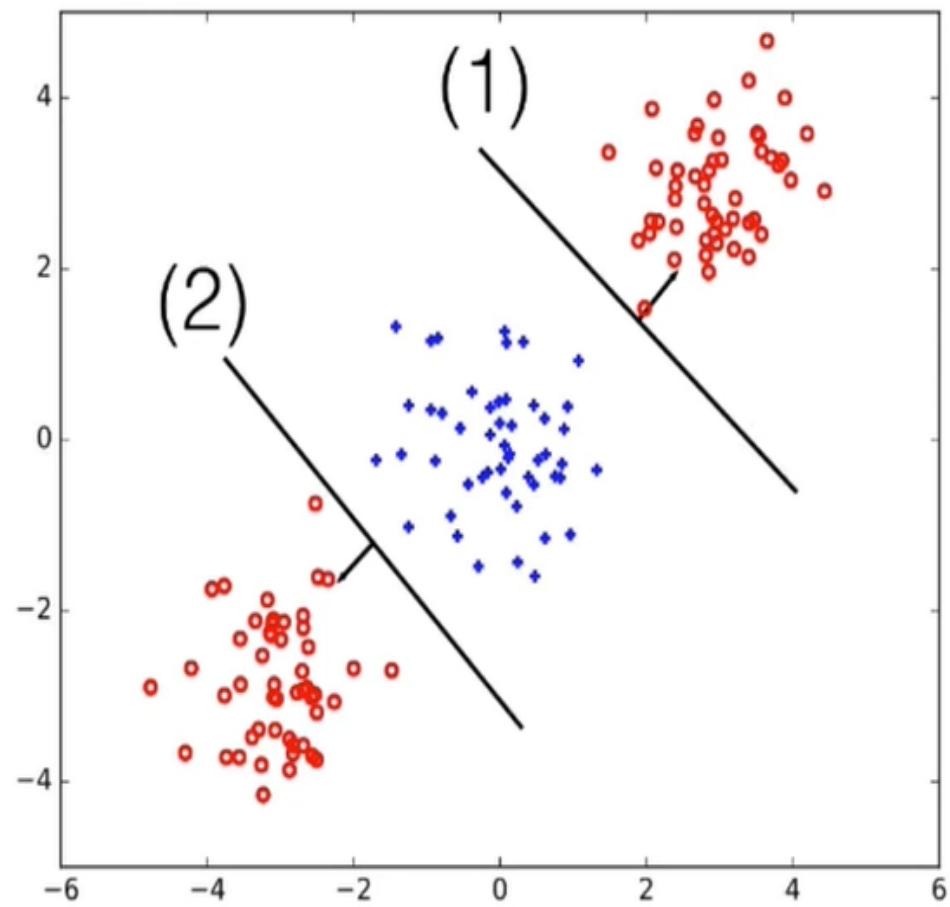
Hidden layer units



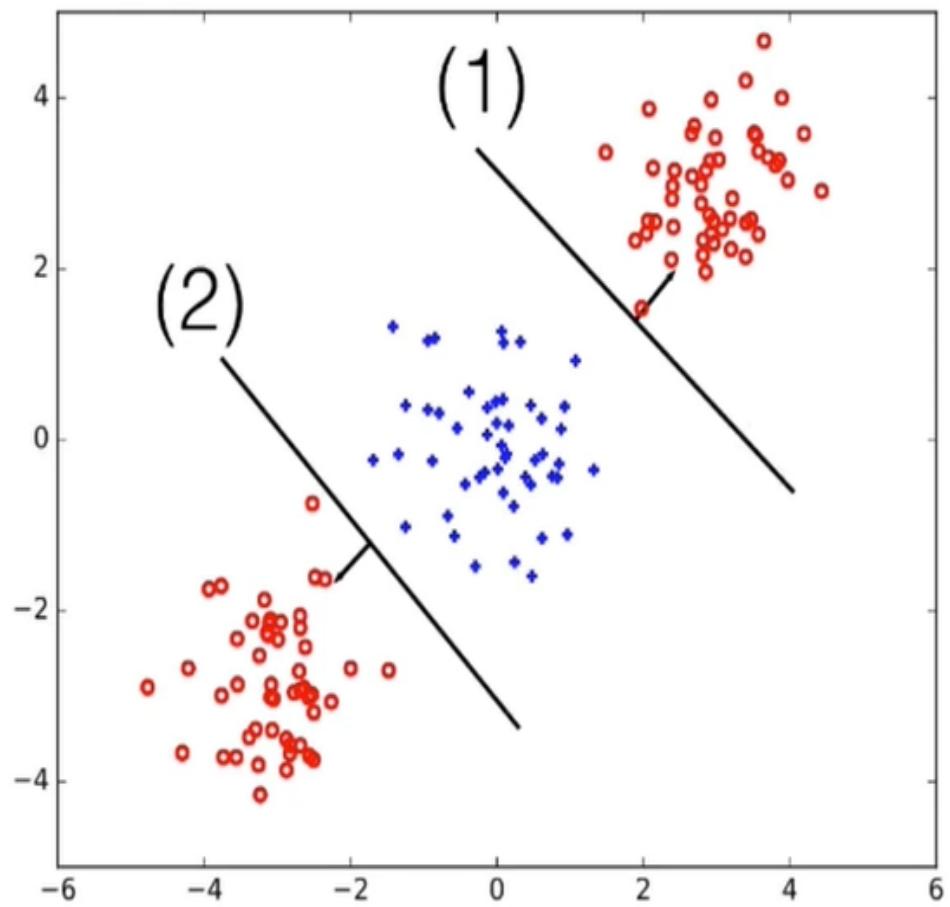
ReLU activation



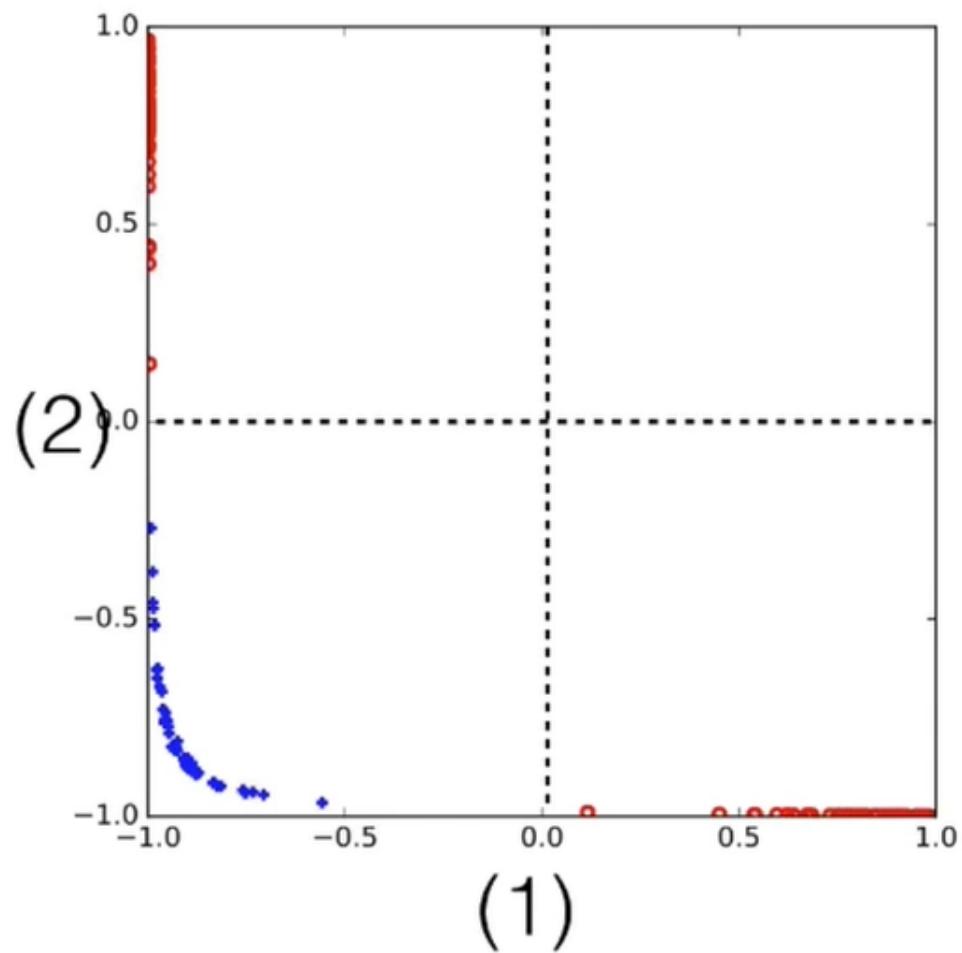
Hidden layer units



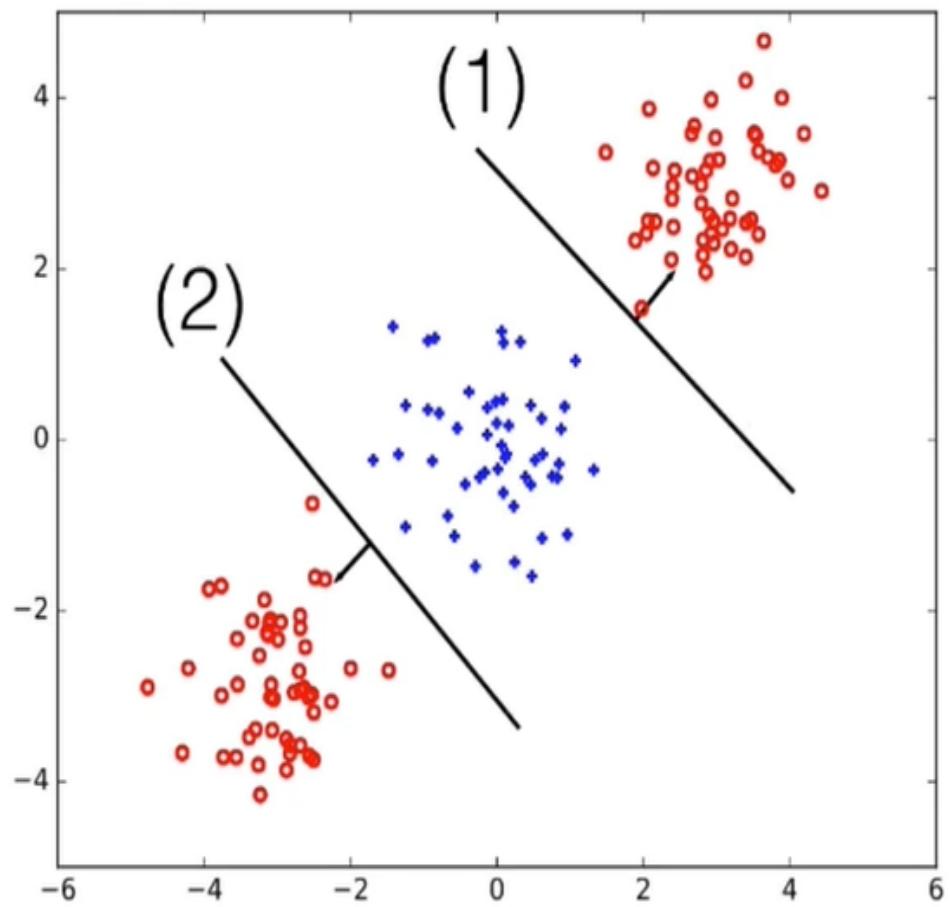
Hidden layer units



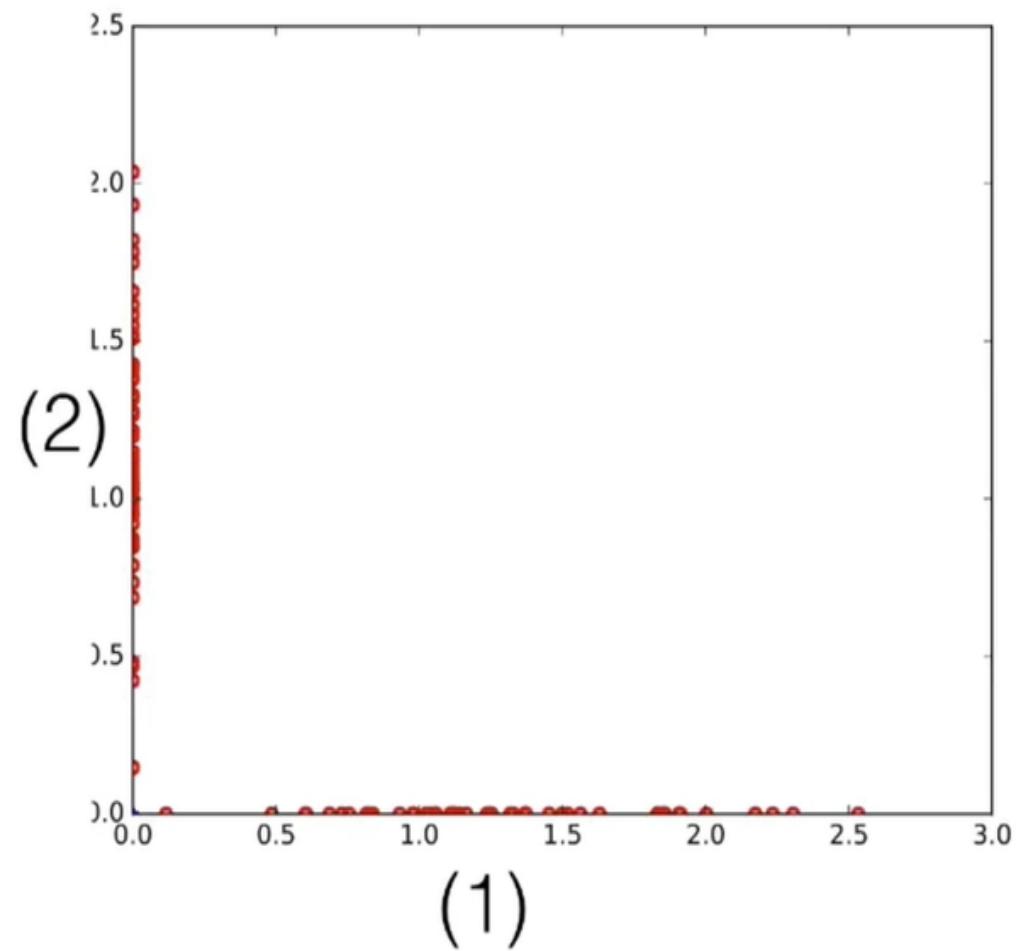
tanh activation



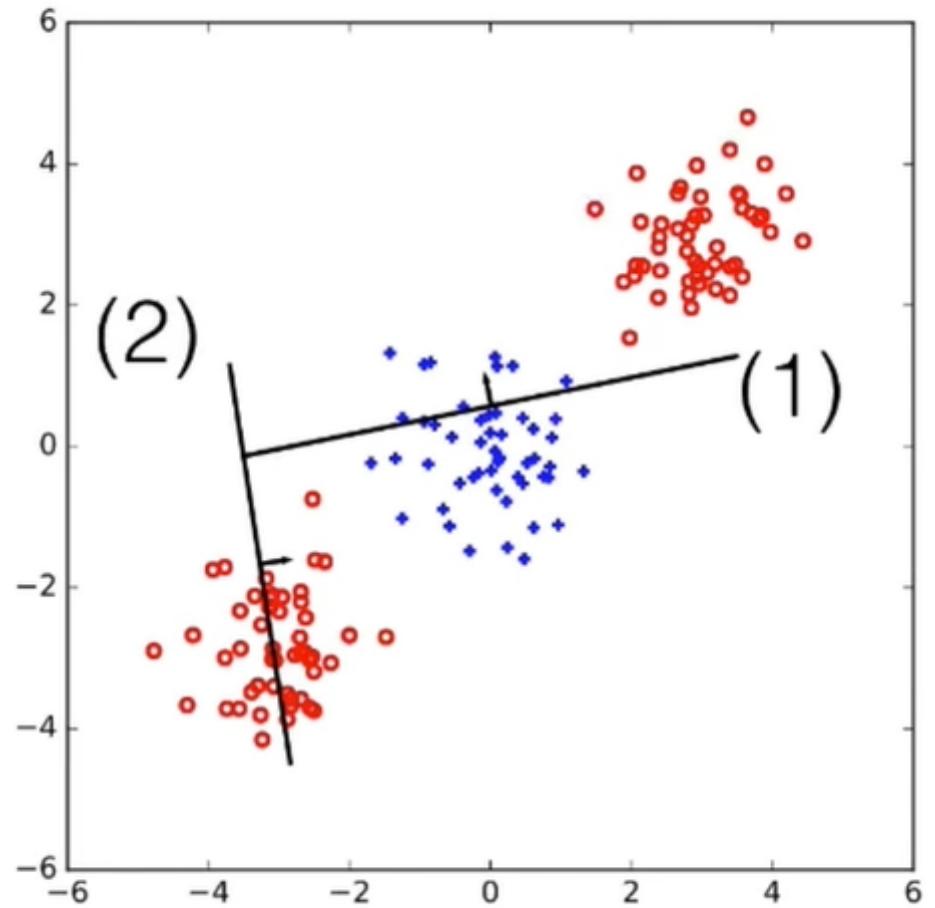
Hidden layer units



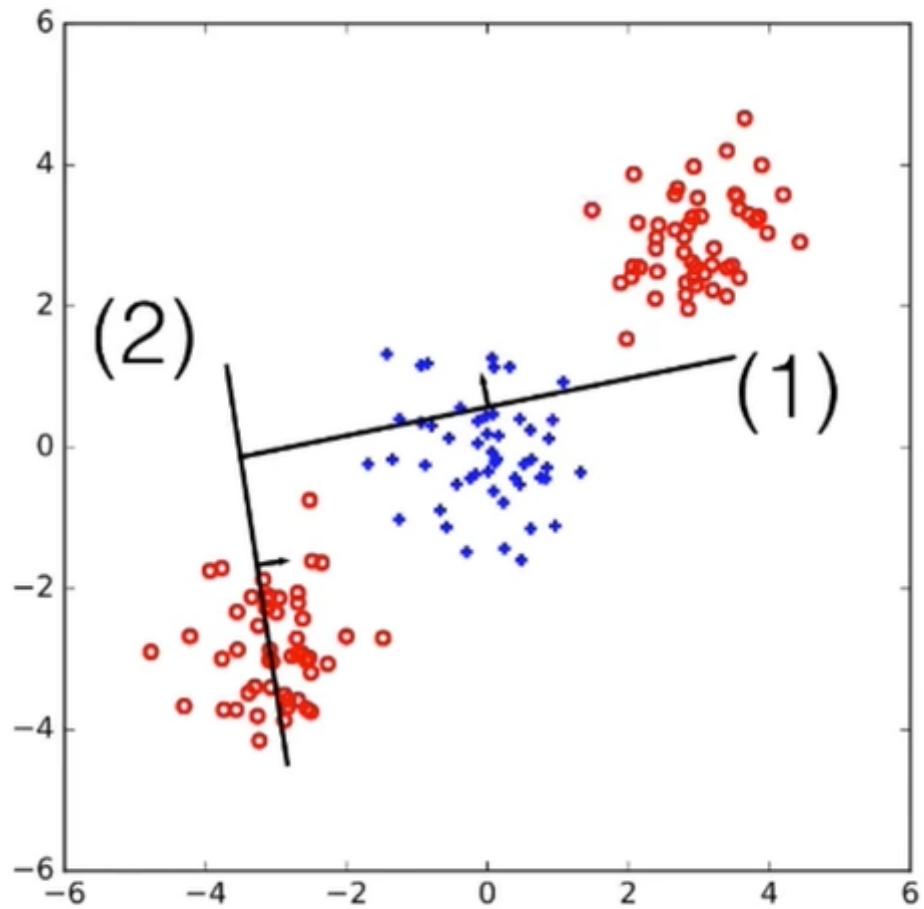
ReLU activation



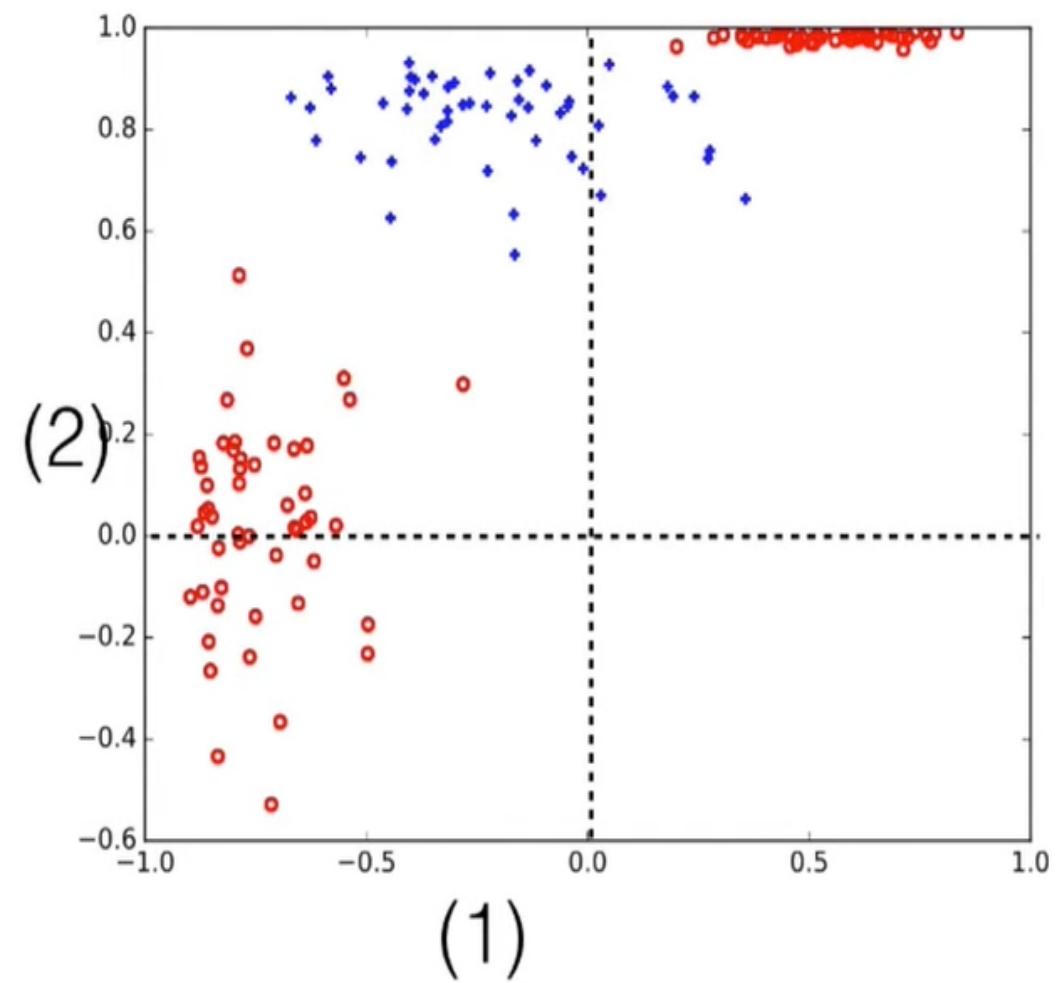
Hidden layer units



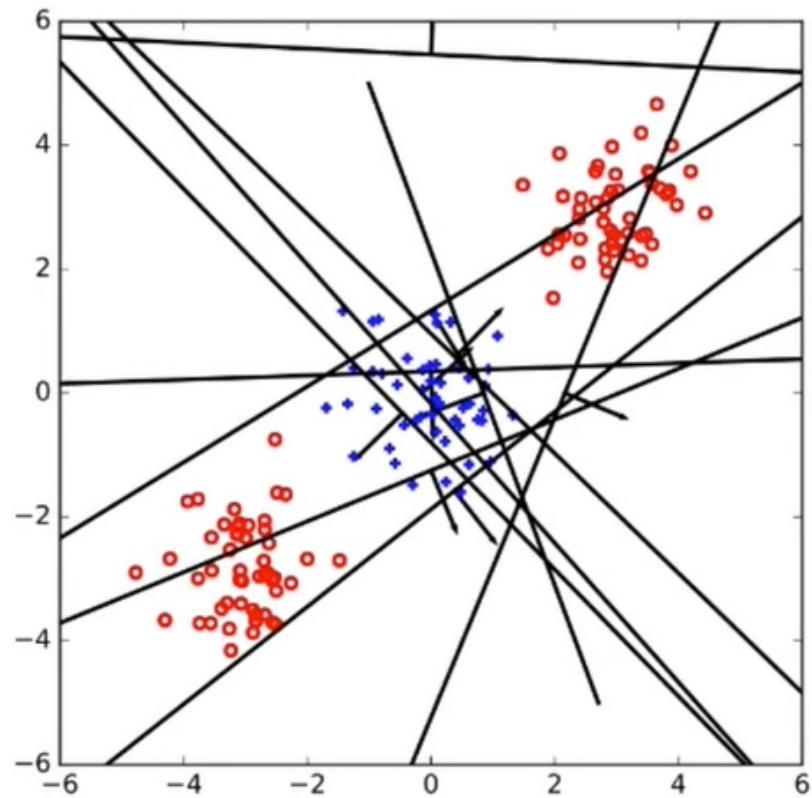
Hidden layer units



tanh activation

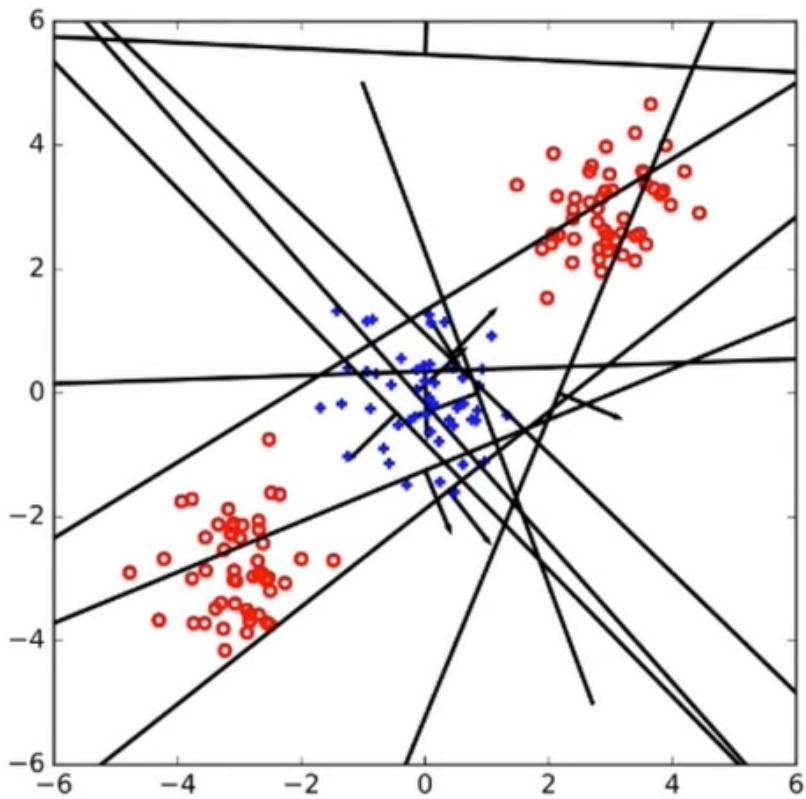


Hidden layer units

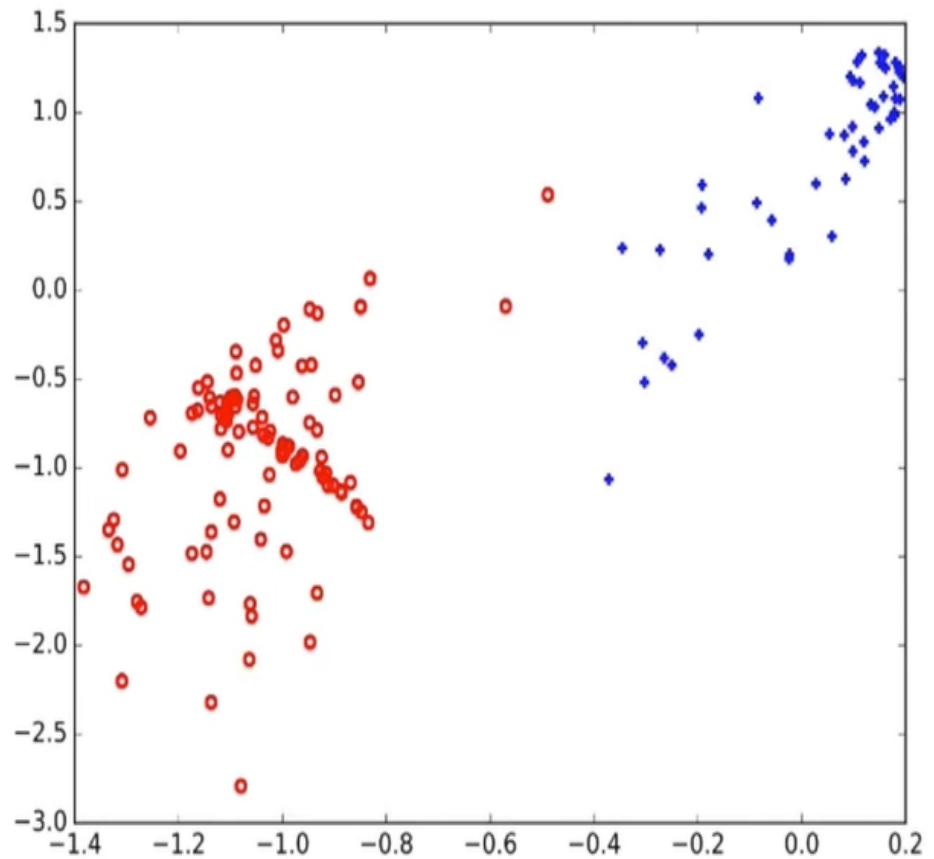


(10 randomly chosen units)

Hidden layer units



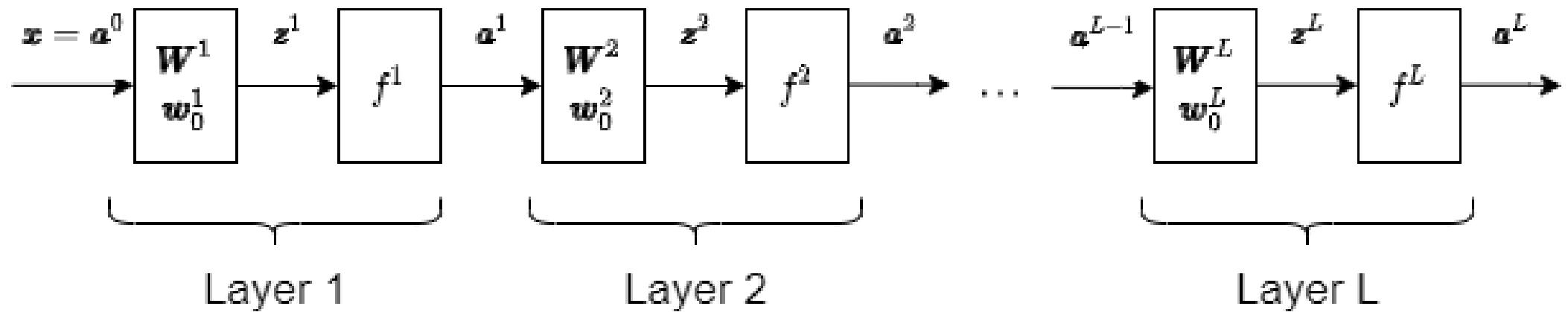
(10 randomly chosen units)



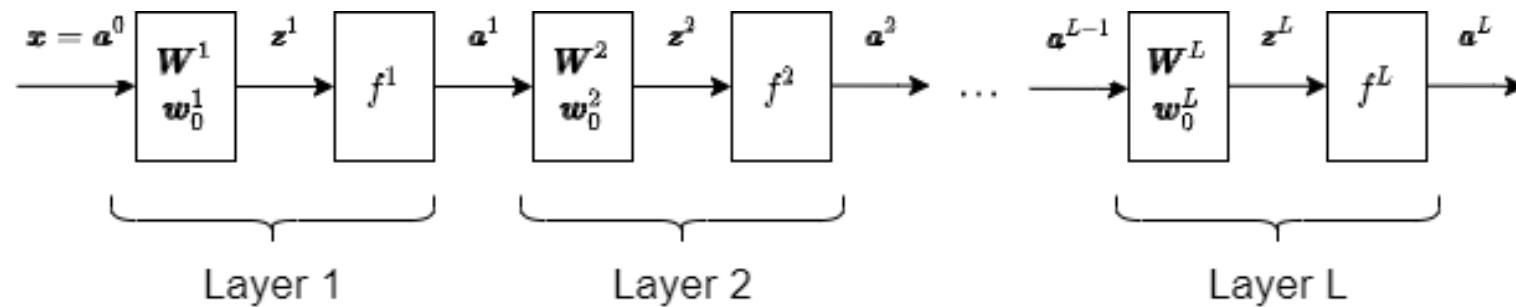
what are the coordinates??

Multiple Layers

Neural Networks with Multiple Layers



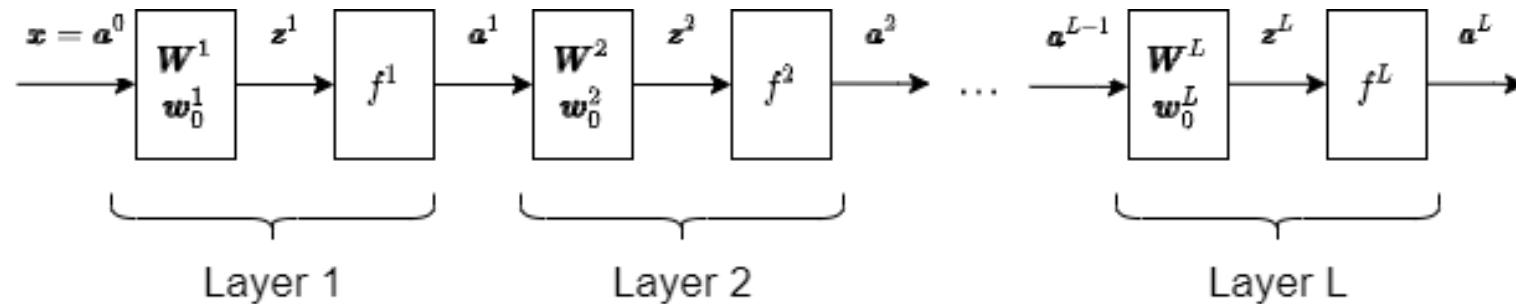
Multiple Layers



Let l be a layer, m^l be the number of inputs to the layer, and n^l be the number of outputs from a layer.

- W^l and w_0^l have $m^l \times n^l$ and $n^l \times 1$ shapes respectively
- f^l be the activation function of layer l

Multiple Layers

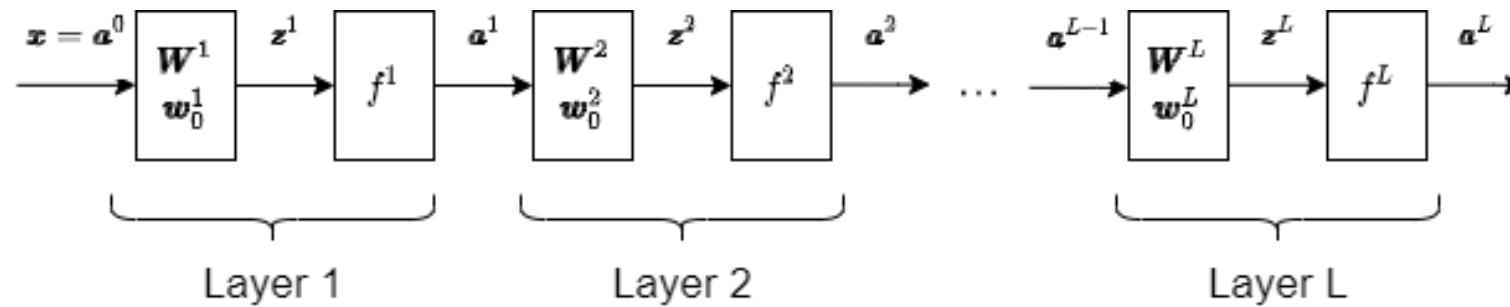


Let l be a layer, m^l be the number of inputs to the layer, and n^l be the number of outputs from a layer.

- W^l and w_0^l have $m^l \times n^l$ and $n^l \times 1$ shapes respectively
- f^l be the activation function of layer l
- z^l pre-activation are $n^l \times 1$ vector.

$$z^l = W^{l^T} a^{l-1} + w_0^l$$

Multiple Layers



Let l be a layer, m^l be the number of inputs to the layer, and n^l be the number of outputs from a layer.

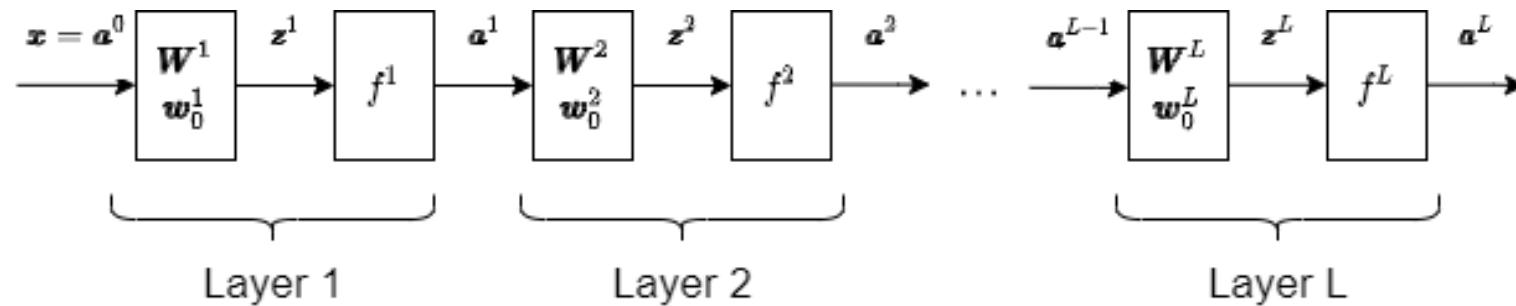
- W^l and w_0^l have $m^l \times n^l$ and $n^l \times 1$ shapes respectively
- f^l be the activation function of layer l
- z^l pre-activation are $n^l \times 1$ vector.

$$z^l = W^{l^T} a^{l-1} + w_0^l$$

- a^l activation outputs are $n^l \times 1$ vectors.

$$a^l = f^l(z^l)$$

Activation Functions

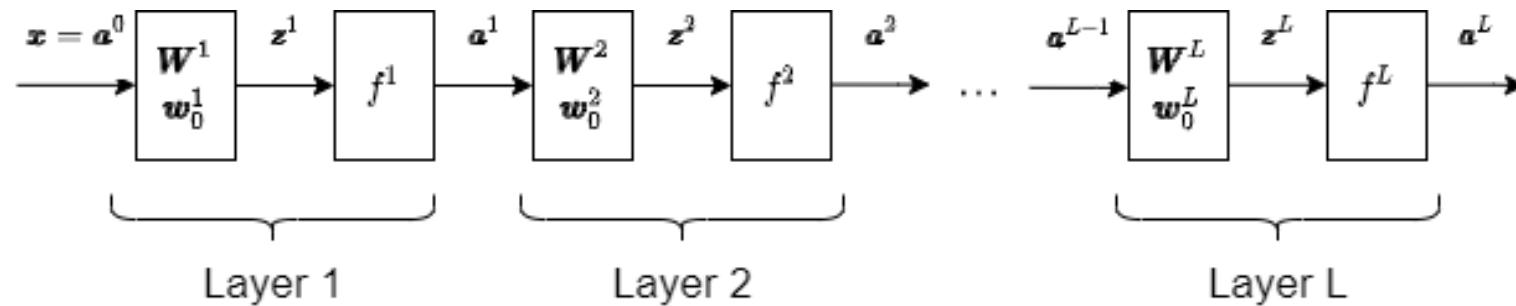


Consider the L layers in a NN and support f is an **identity function**.

$$a^L = W^{L^T} a^{L-1}$$

Note identity function $f(x)$ is $f(x) = x$

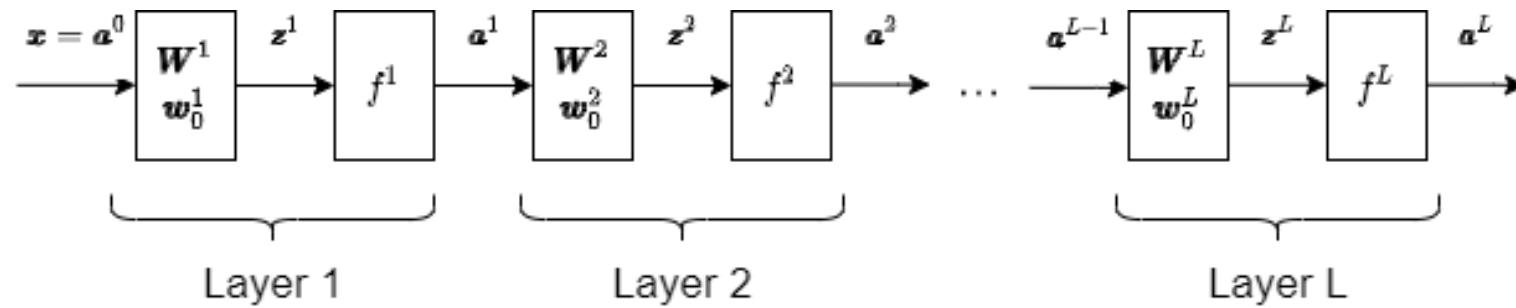
Activation Functions



Consider the L layers in a NN and support f is an **identity function**.

$$\begin{aligned} a^L &= W^{L^T} a^{L-1} \\ &= W^{L^T} (W^{L-1^T} a^{L-2}) \end{aligned}$$

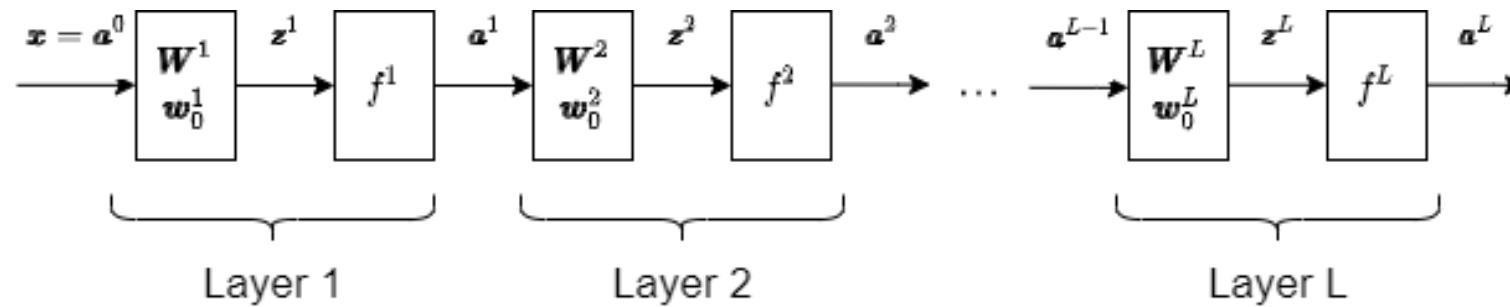
Activation Functions



Consider the L layers in a NN and support f is an **identity function**.

$$\begin{aligned} a^L &= W^{L^T} a^{L-1} \\ &= W^{L^T} (W^{L-1^T} a^{L-2}) \\ &= W^{L^T} (W^{L-1^T} (W^{L-2^T} a^{L-3})) \end{aligned}$$

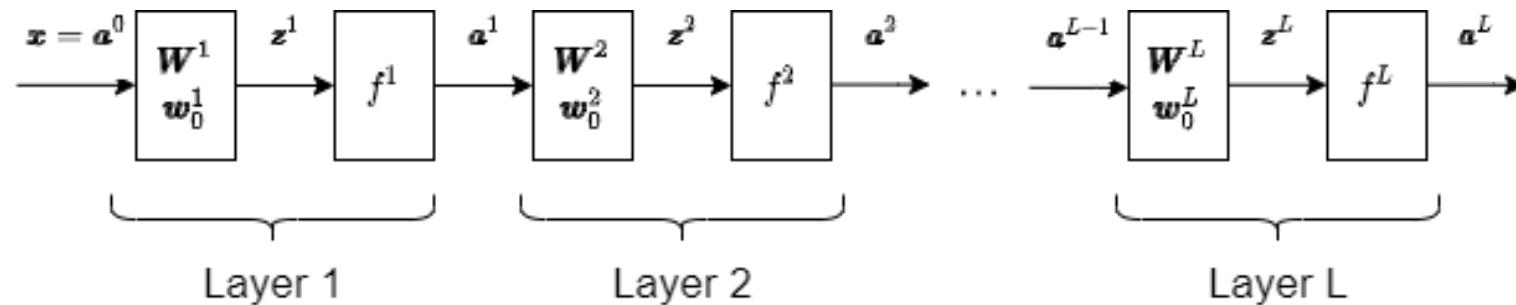
Activation Functions



Consider the L layers in a NN and support f is an **identity function**.

$$\begin{aligned} a^L &= W^{L^T} a^{L-1} \\ &= W^{L^T} (W^{L-1^T} a^{L-2}) \\ &= W^{L^T} (W^{L-1^T} (W^{L-2^T} a^{L-3})) \\ &= W^{L^T} (W^{L-1^T} (W^{L-2^T} \dots (W^{1^T} x))) \end{aligned}$$

Activation Functions



Consider the L layers in a NN and suppose f is an **identity function**.

$$a^L = W^{L^T} a^{L-1} = W^{L^T} (W^{L-1^T} a^{L-2}) = W^{L^T} (W^{L-1^T} \dots (W^{1^T} x))$$

So multiplying out the weight matrices we find that

$$a^L = W^{\text{total}} x$$

Activation Functions

$$a^L = W^{\text{total}} x$$

this is a linear function of x . Having those layers did not change the representational capacity of the network. The **non-linearity** of the activation function is crucial.

Activation Functions

$$a^L = W^{\text{total}} x$$

this is a linear function of x . Having those layers did not change the representational capacity of the network. The **non-linearity** of the activation function is crucial.

- Any function representable by any number of linear layers can be represented by a single layer.

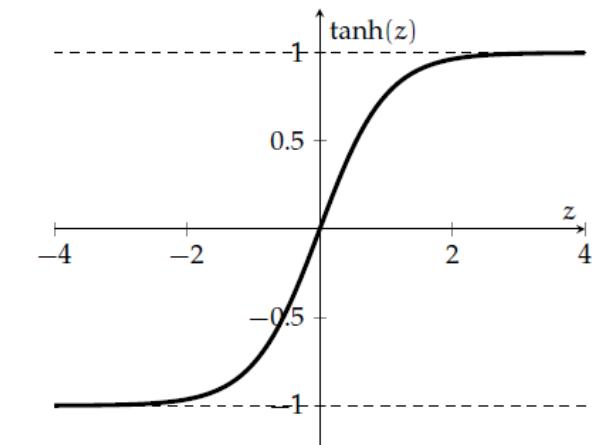
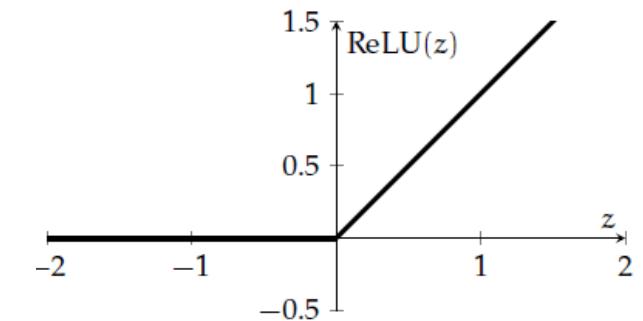
Activation Functions

$$a^L = W^{\text{total}} x$$

this is a linear function of x . Having those layers did not change the representational capacity of the network. The **non-linearity** of the activation function is crucial.

- Any function representable by any number of linear layers can be represented by a single layer.

We need *non-linear* activation functions (ReLU, tanh).



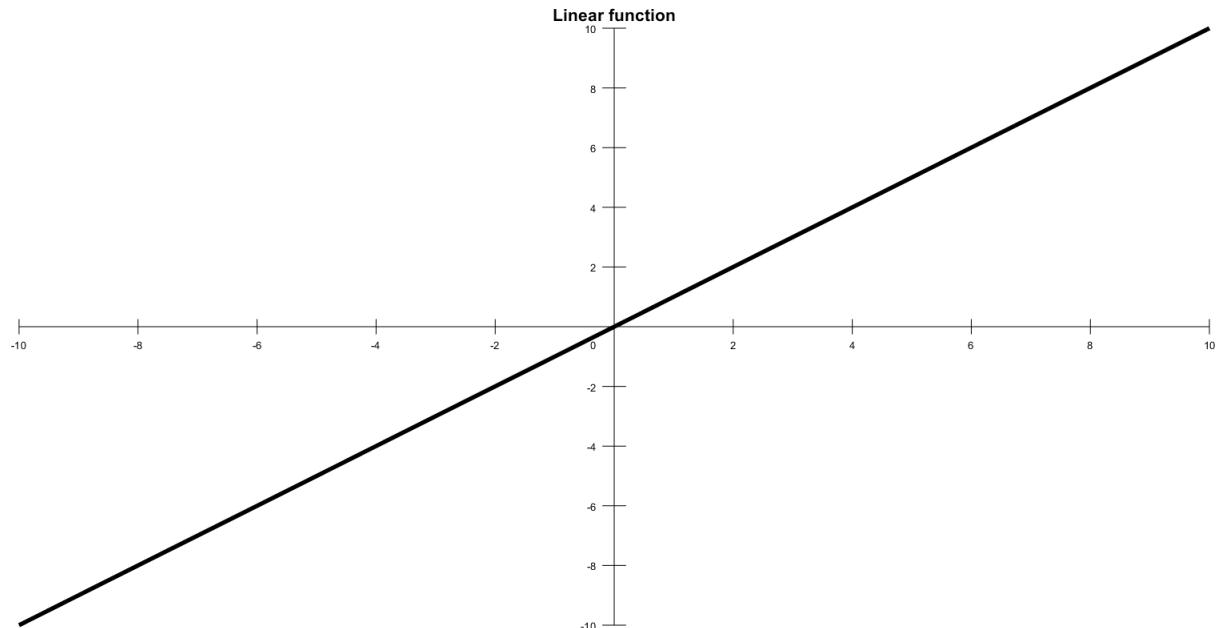
Activation Functions for the Final Layer

Activation Functions for the Final Layer

Linear (Identity)

$$f(z) = z$$

This is used in the final layer when doing regression with NNs.

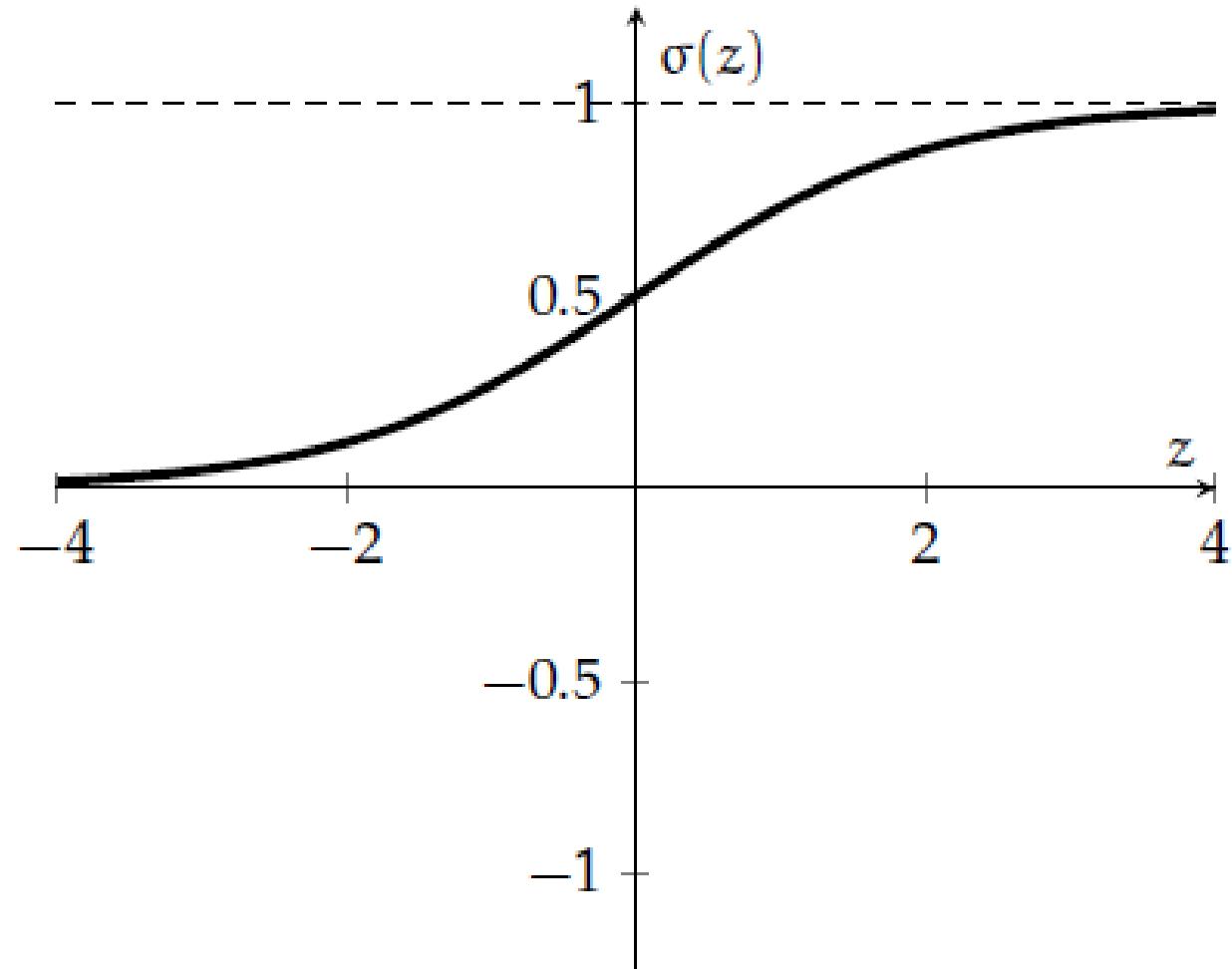


Activation Functions for the Final Layer

Sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

also known as *logistic* function. This can be interpreted as a probability because the output is in $[0, 1]$. Used in the final layer in binary classification problems.



Softmax

Softmax takes a vector z and generates a output vector $a \in [0, 1]^n$ with the property that $\sum_i a_i = 1$. We can interpret it as a probability distribution over n items. Used in the final layer for multi-class classification problems.

$$\text{softmax}(z) = \begin{bmatrix} e^{z_1} / \sum_i e^{z_i} \\ \vdots \\ e^{z_n} / \sum_i e^{z_i} \end{bmatrix}$$

Softmax example

$$z = \begin{bmatrix} 6 \\ -3 \\ 2.5 \\ 0.4 \end{bmatrix}$$

$$\text{softmax}(z) =$$

$$\begin{bmatrix} \frac{e^6}{e^6 + e^{-3} + e^{2.5} + e^{0.4}} \\ \frac{e^{-3}}{e^6 + e^{-3} + e^{2.5} + e^{0.4}} \\ \frac{e^{2.5}}{e^6 + e^{-3} + e^{2.5} + e^{0.4}} \\ \frac{e^{0.4}}{e^6 + e^{-3} + e^{2.5} + e^{0.4}} \end{bmatrix} = \begin{bmatrix} 0.9671 \\ 0.0001 \\ 0.0292 \\ 0.0036 \end{bmatrix}$$

Activation Functions

- **ReLU** is commonly used in *hidden layers*,
- **identity** activations are used for the output of *regression* tasks.
- **sigmoid** activations are common for the output of *binary* classification.
- **softmax** is used for output of *multi-class* classification.

COGS514 - Cognition and Machine Learning

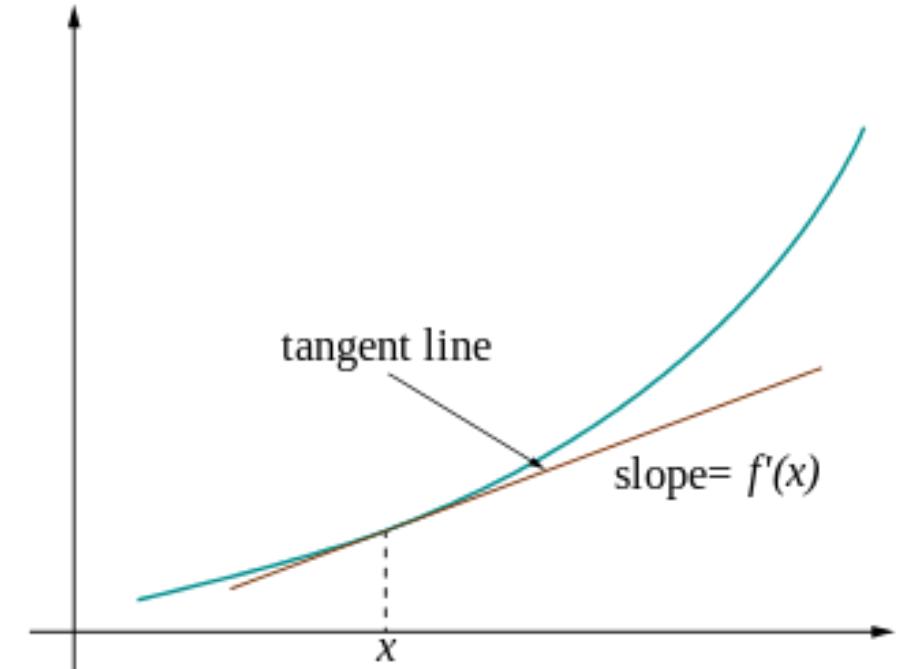
Learning Neural Networks

Calculus Reminder

Derivative

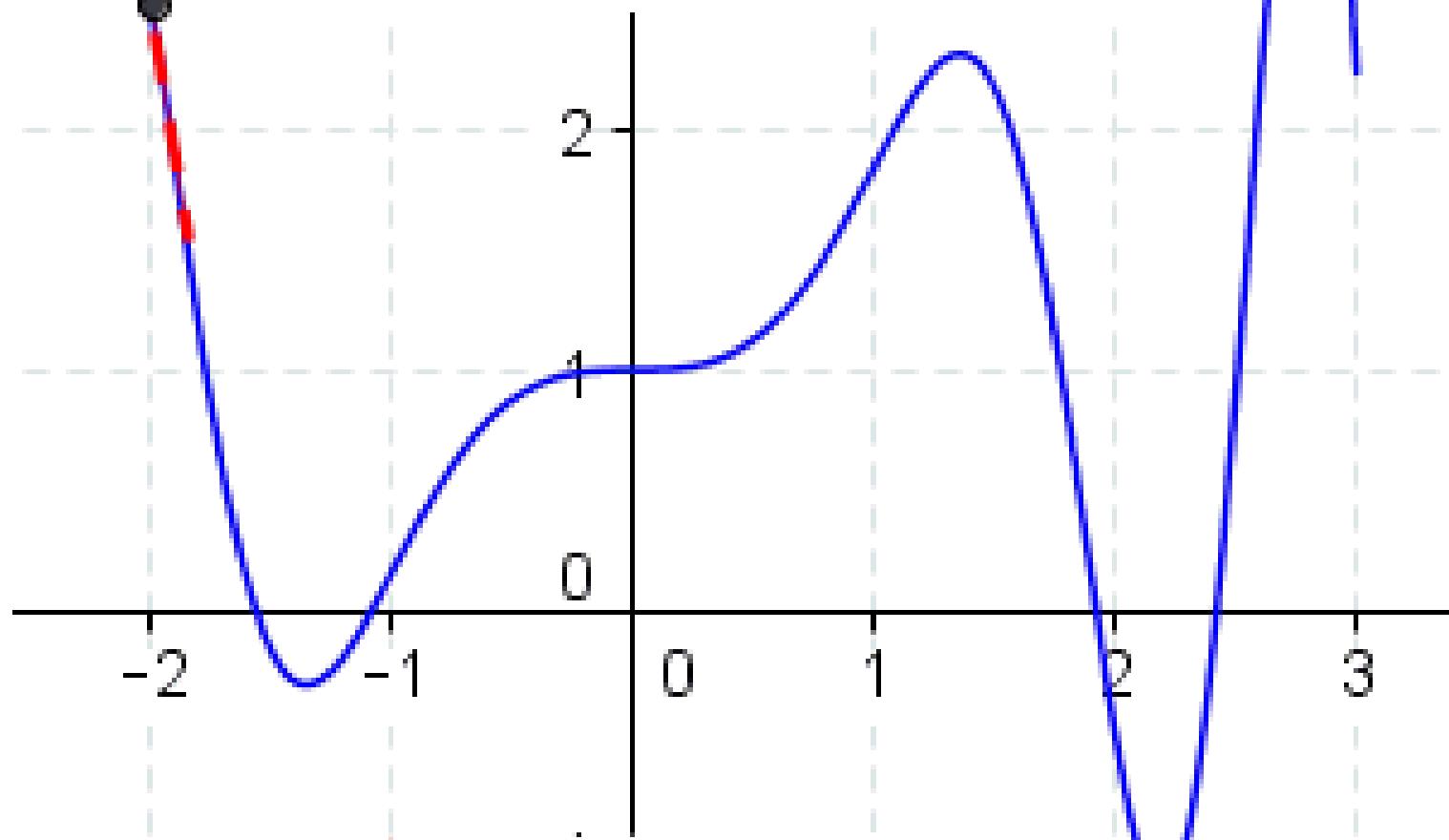
$$f'(x) = \frac{df}{dx}$$

- Derivative of a function is its sensitivity to change w.r.t to its inputs
 - df i.e. tiny change in $f(x)$ with respect dx i.e. to tiny change in x
- $f'(x)$ can be visualised as the slope of the function $f(x)$ at x



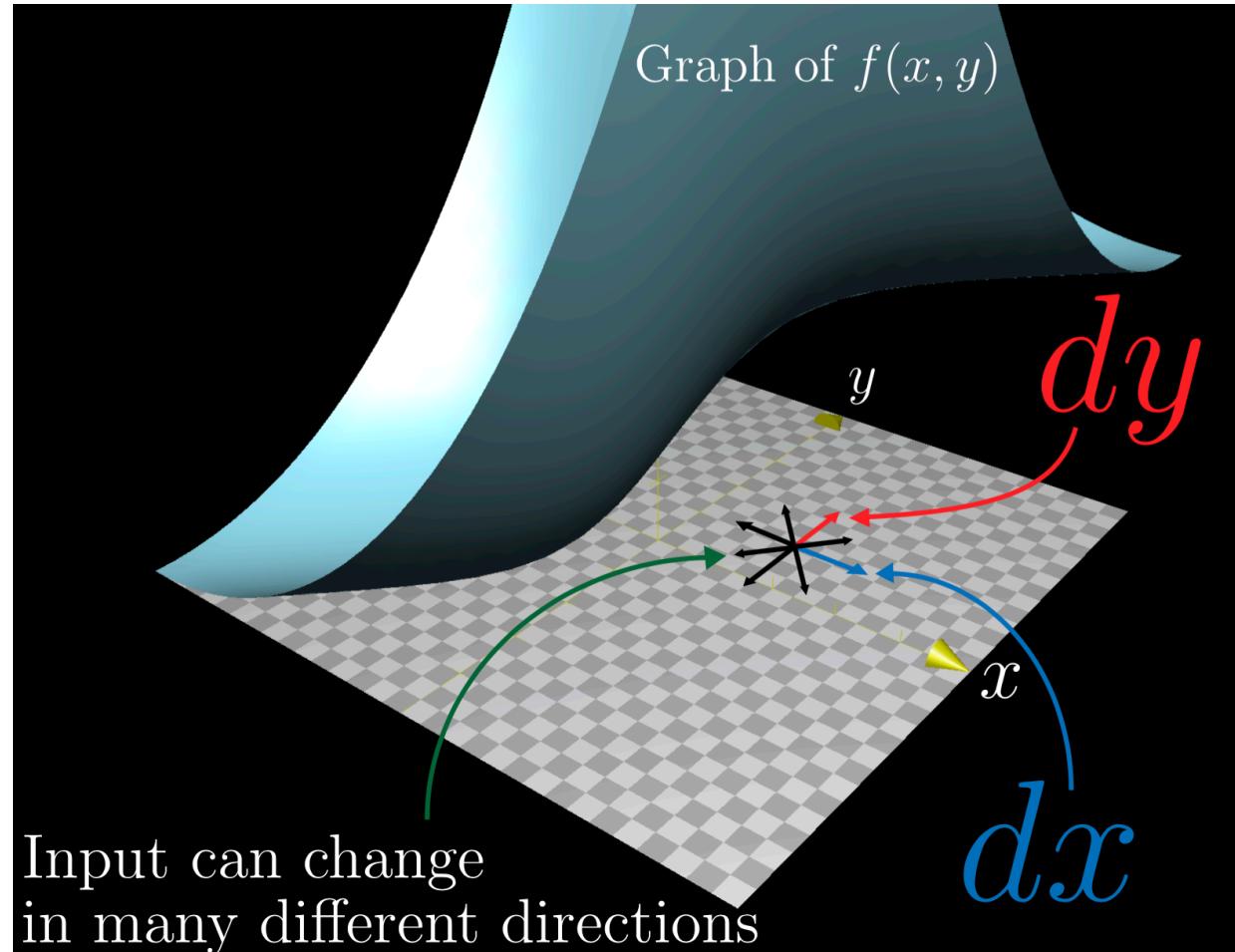
$$f(x) = x \sin(x^2) + 1$$

$$A = (-2, 2.51)$$



Partial Derivatives

When you have a function with two inputs $f(x, y)$,

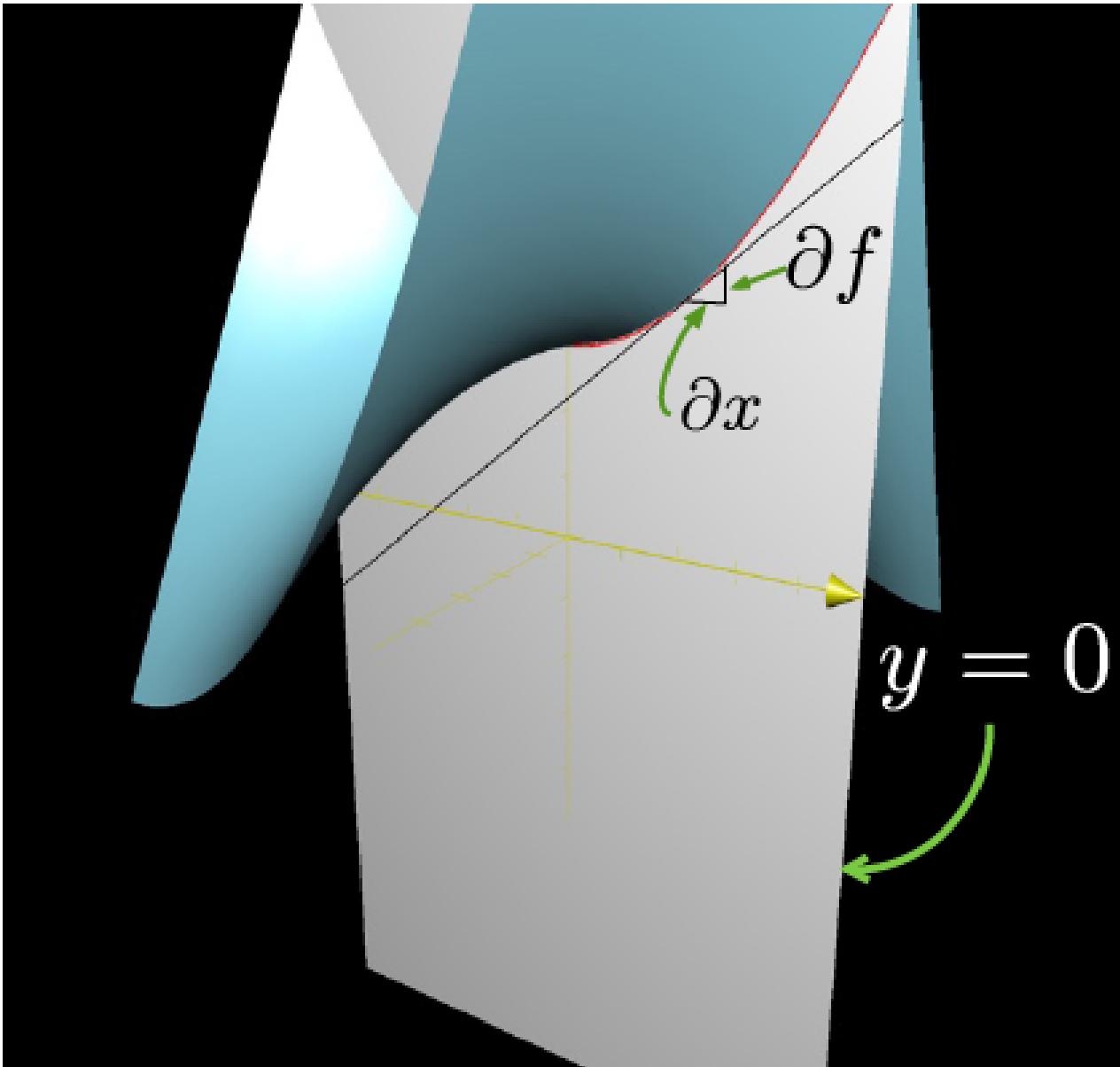


Partial Derivatives

When you have a function with two inputs $f(x, y)$,

- you can look at the tiny change in output with respect to a tiny change in x

$$\frac{\partial f}{\partial x}$$

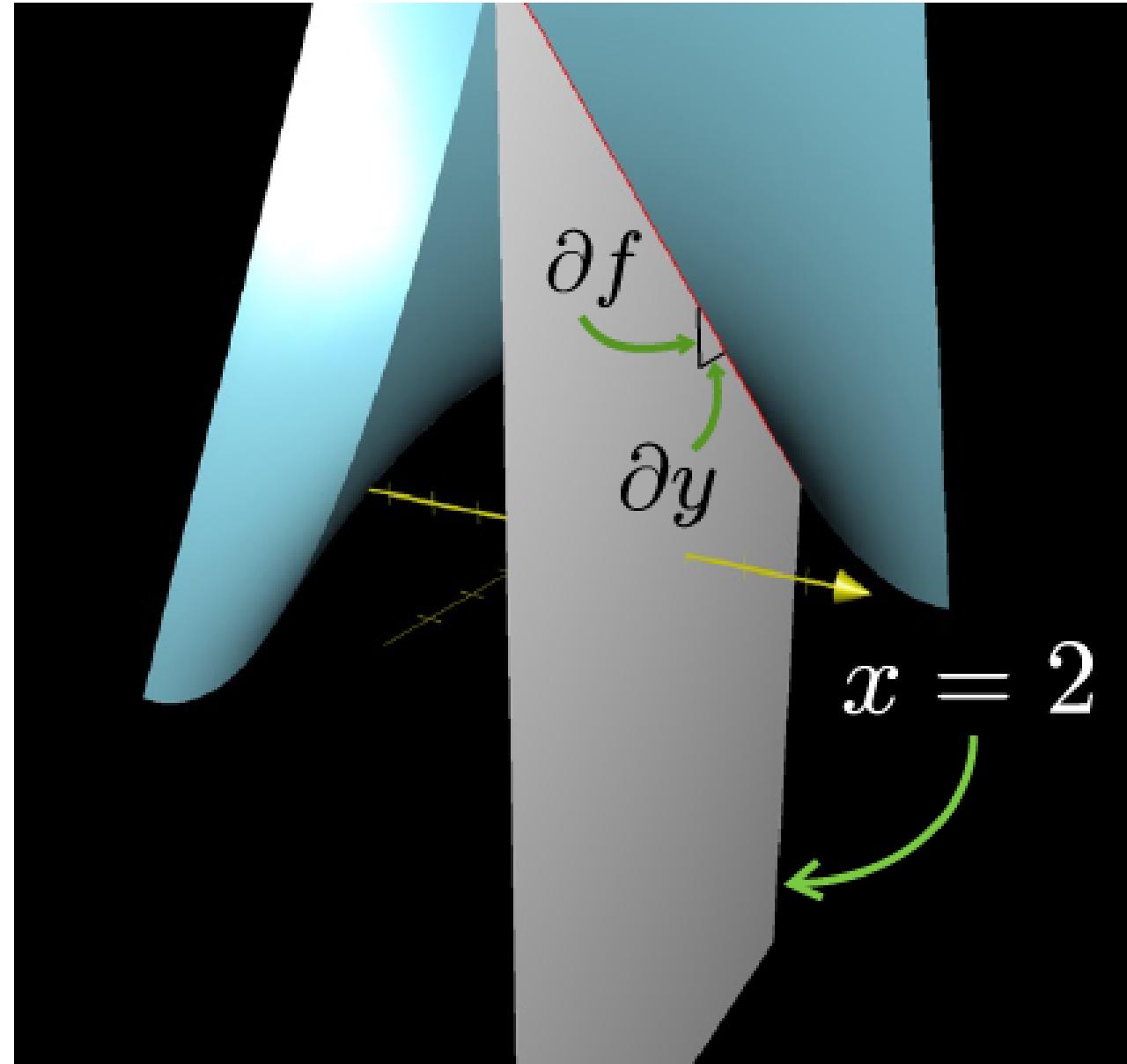


Partial Derivatives

Suppose you have a function with two inputs $f(x, y)$,

- you can look at the tiny change in output with respect to a tiny change in x
- or you can look at the tiny change in output with respect to a tiny change in y

$$\frac{\partial f}{\partial y}$$



Gradient of Φ w.r.t w

is the vector of its partial derivatives

$$\nabla_w \Phi = \begin{bmatrix} \partial \Phi / \partial w_1 \\ \partial \Phi / \partial w_2 \\ \vdots \\ \partial \Phi / \partial w_d \end{bmatrix}$$

where $w \in \mathbb{R}^d$, $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}$

Gradient

Suppose you have a function with two inputs $f(x, y)$,

Gradient of f is a vector that contains all partial derivative information

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Gradient points in the direction of greatest increase in f

Chain Rule

For differentiating composite functions such as $f(g(c))$

If $a = f(b)$ and $b = g(c)$, so that $a = f(g(c))$

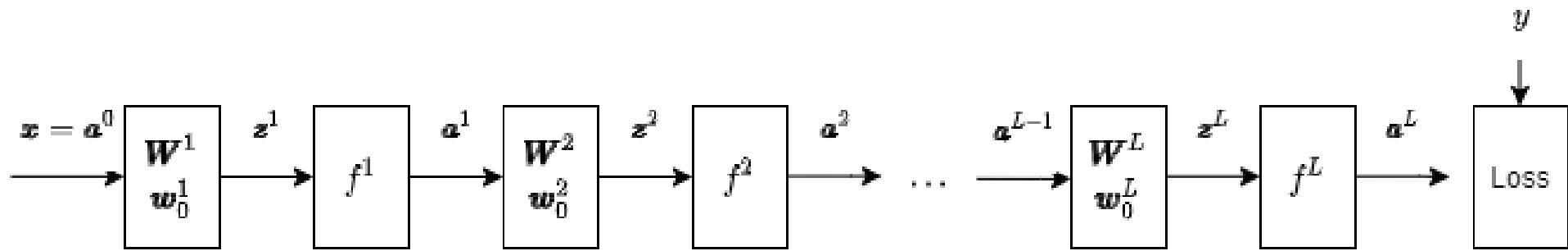
Chain Rule

For differentiating composite functions such as $f(g(c))$

If $a = f(b)$ and $b = g(c)$, so that $a = f(g(c))$ then

$$\frac{da}{dc} = \frac{da}{db} \cdot \frac{db}{dc} = f'(b)g'(c) = f'(g(c))g'(c)$$

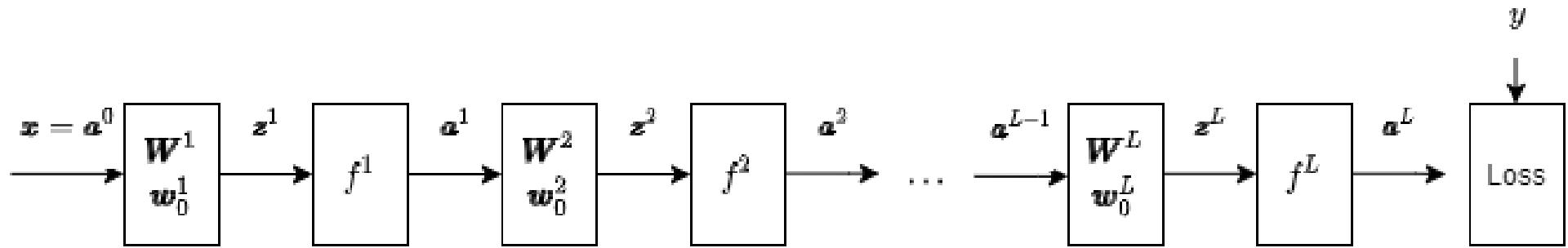
Neural Nets - Composite functions



First layer

$$f^1 \underbrace{\left(W^1 x + w_0^1 \right)}_{z_1} \underbrace{a_1}_{a_1}$$

Neural Nets - Composite functions

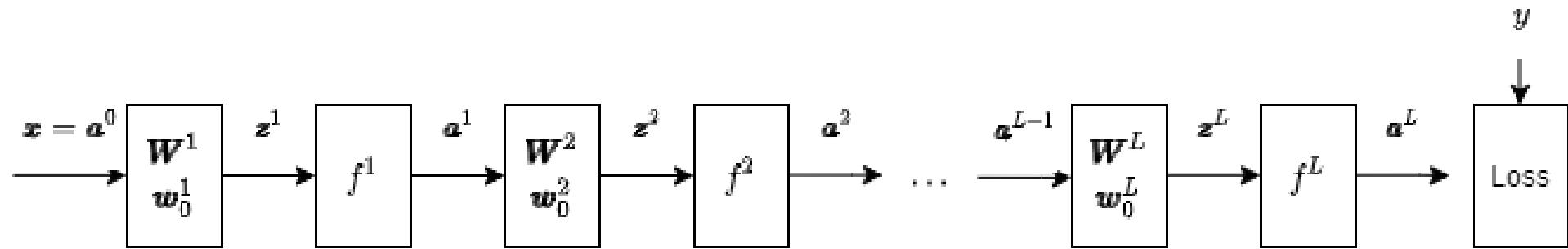


Second layer

$$f^2(W^2 a_1 + w_0^2) = f^2(\underbrace{W^2 f^1(W^1 x + w_0^1) + w_0^2}_{z_2})$$

$\underbrace{\hspace{10em}}_{a_2}$

Neural Nets - Composite functions



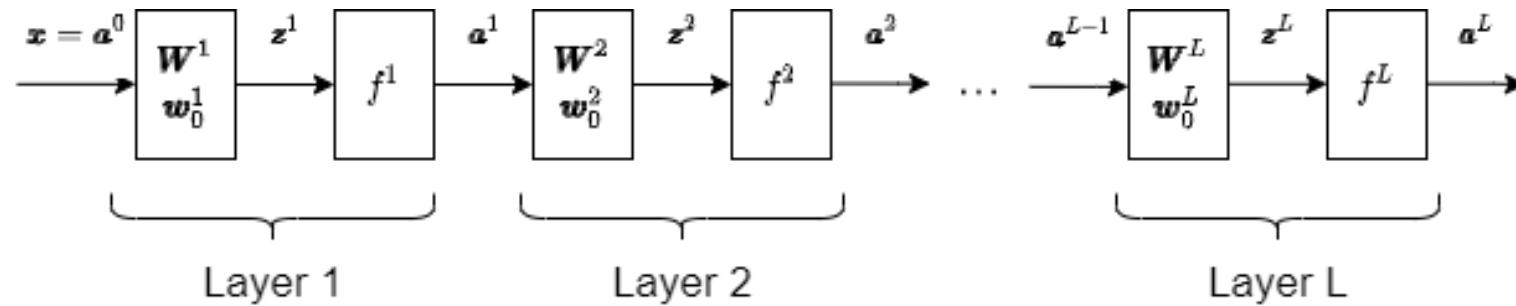
Third layer

$$f^3(W^3 a_2 + w_0^3) = f^3(\underbrace{W^3 f^2(W^2 f^1(W^1 x + w_0^1) + w_0^2) + w_0^3}_{\underbrace{z_3}_{a_3}})$$

and so on

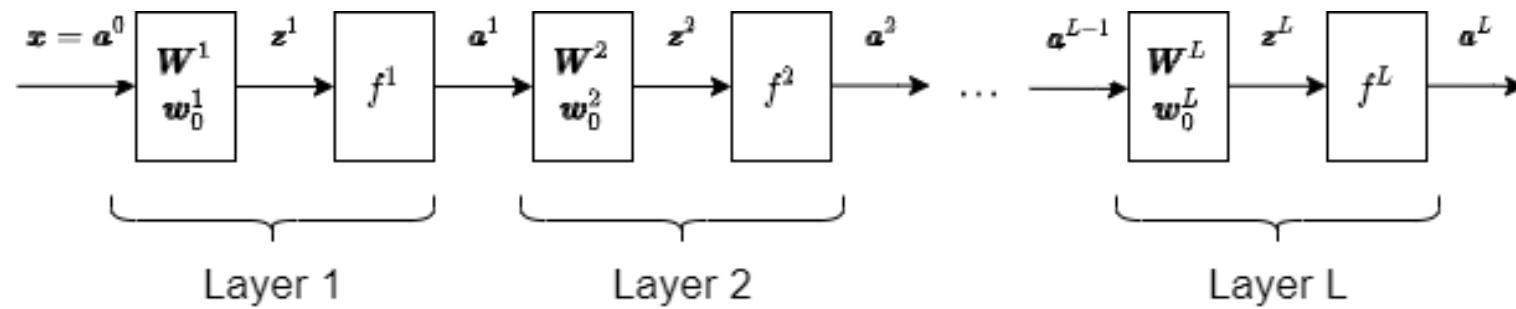
Learning Deep Neural Networks

We learn deep neural networks with *stochastic gradient descent*.

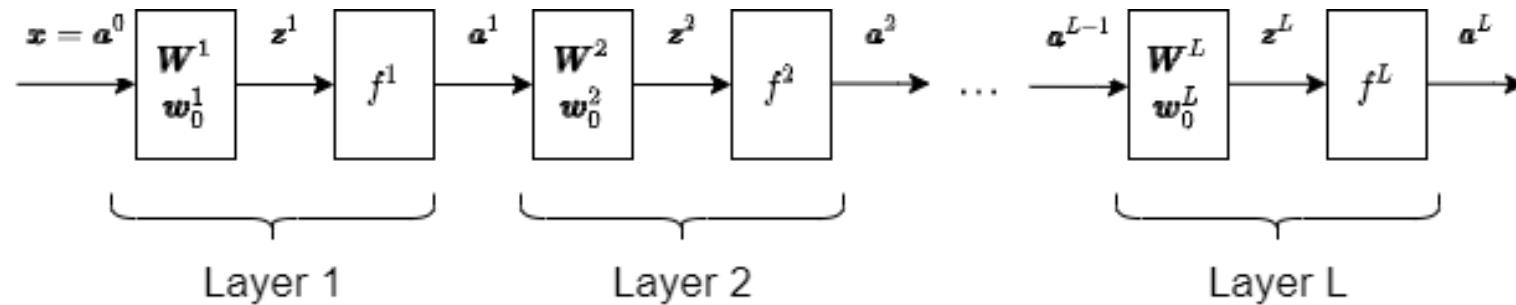


1. Predict \hat{y} from x (Forward Pass 

- $\hat{y} = NN(x; W)$



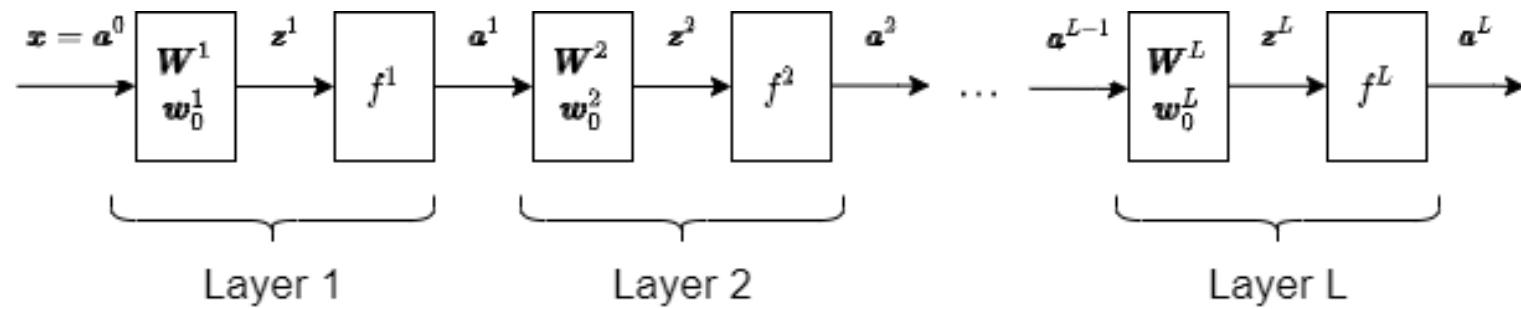
- $\hat{y} = NN(x; W)$
2. Compare y and \hat{y} and calculate *loss*
- $loss(\hat{y}, y)$



- $\hat{y} = NN(x; W)$
- $loss(NN(x; W), y)$

3. Calculate the gradient of weights w.r.t loss $\nabla_W loss$ and update each weight in the opposite direction of gradient

- $W = W - \alpha \nabla_W loss(NN(x, W), y)$



- $\hat{y} = NN(x; W)$
- $loss(NN(x; W), y)$
- $W = W - \alpha \nabla_W loss(NN(x, W), y)$

Note: W represents all weights W^l, w_0^l in all layers $l = (1, \dots, L)$

Stochastic Gradient Descent

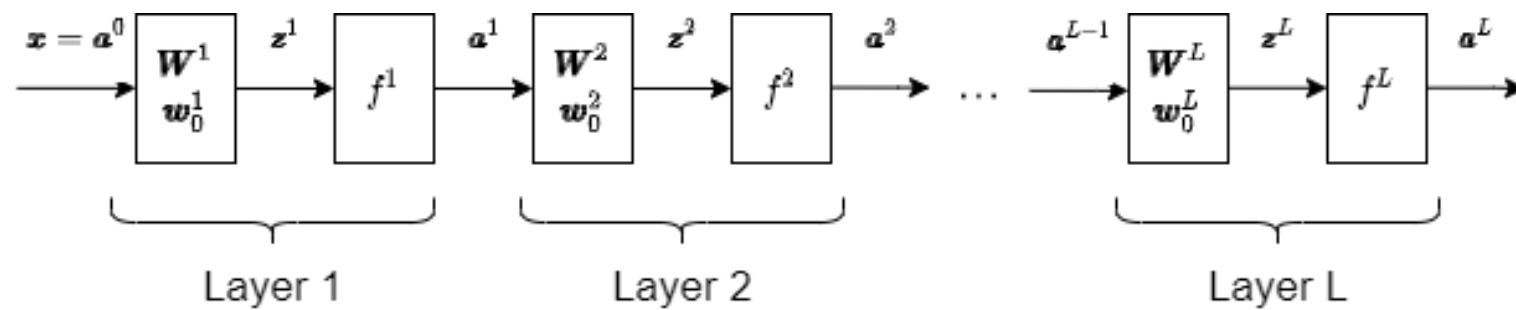
Start from an initial point $W^{(0)}$

for $t = 1$ to T

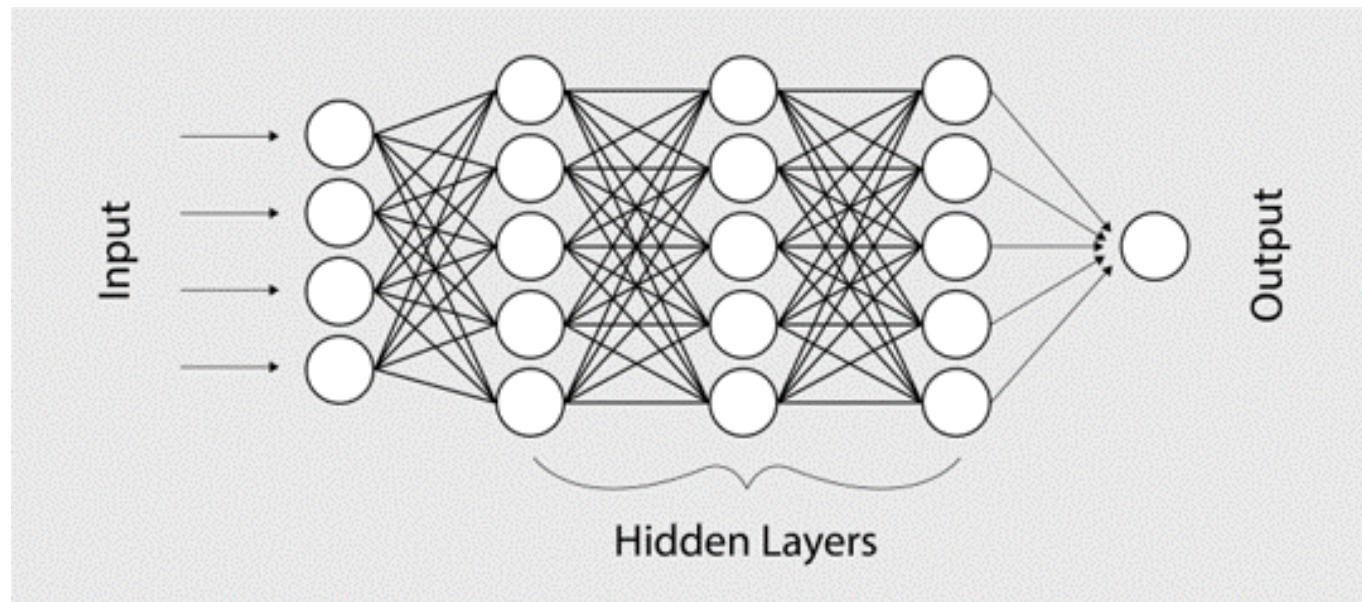
- randomly select $i \in 1, 2, \dots, n$
- $W^{(t)} = W^{(t-1)} - \alpha(t) \nabla_W \text{loss}(\text{NN}(x^{(i)}, W^{(t-1)}), y^{(i)})$

return $W^{(t)}$

How to calculate the gradient $\nabla_W loss(NN(x, W), y)$?

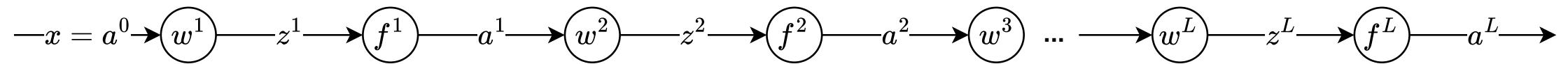


How to calculate the gradient $\nabla_W loss(NN(x, W), y)$?

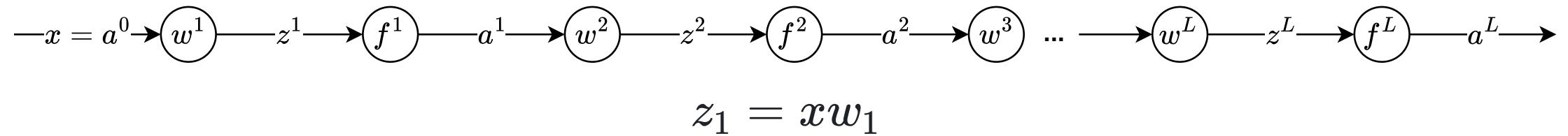


Backpropagation

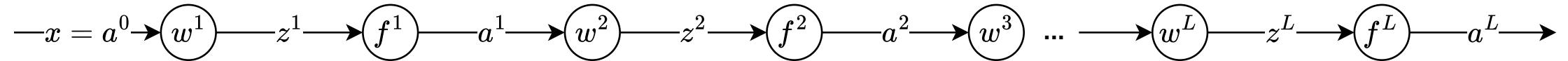
Example - Deep Neural Network with just one unit at every layer



Example - Deep Neural Network with just one unit at every layer



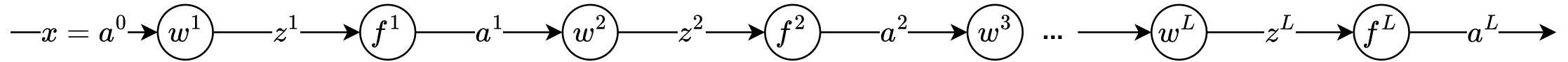
Example - Deep Neural Network with just one unit at every layer



$$z_1 = xw_1$$

$$a_1 = f_1(xw_1)$$

Example - Deep Neural Network with just one unit at every layer

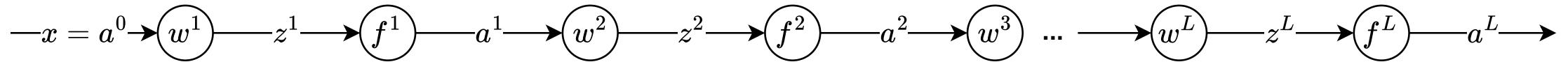


$$z_1 = xw_1$$

$$a_1 = f_1(xw_1)$$

$$z_2 = a_1 w_2$$

Example - Deep Neural Network with just one unit at every layer



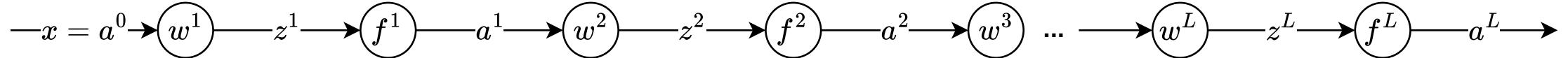
$$z_1 = xw_1$$

$$a_1 = f_1(xw_1)$$

$$z_2 = a_1 w_2$$

$$a_2 = f_2(a_1 w_2)$$

Example - Deep Neural Network with just one unit at every layer



$$z_1 = xw_1$$

$$a_1 = f_1(xw_1)$$

$$z_2 = a_1 w_2$$

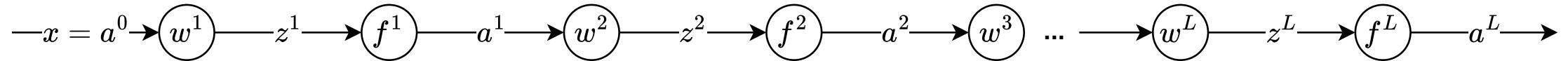
$$a_2 = f_2(a_1 w_2)$$

⋮
⋮

$$z_L = a_{L-1} w_L$$

$$a_L = f_L(a_{L-1} W_L)$$

Example - Deep Neural Network with just one unit at every layer



$$z_1 = xw_1$$

$$loss(y, a_L) = \frac{1}{2}(y - a_L)^2$$

$$a_1 = f_1(xw_1)$$

$$z_2 = a_1 w_2$$

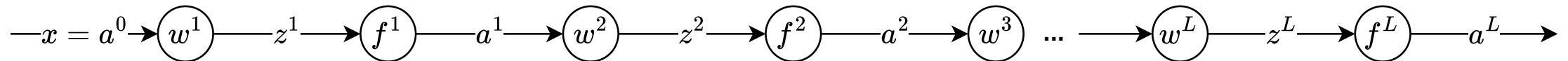
$$a_2 = f_2(a_1 w_2)$$

⋮
⋮

$$z_L = a_{L-1} w_L$$

$$a_L = f_L(a_{L-1} W_L)$$

Example - Deep Neural Network with just one unit at every layer



$$z_1 = xw_1$$

$$loss(y, a_L) = \frac{1}{2}(y - a_L)^2$$

$$a_1 = f_1(xw_1)$$

$$z_2 = a_1 w_2$$

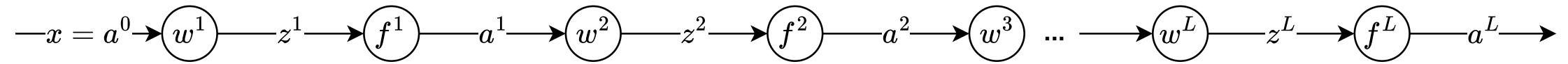
$$a_2 = f_2(a_1 w_2)$$

⋮

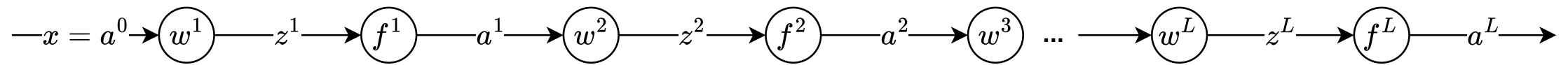
$$z_L = a_{L-1} w_L$$

$$a_L = f_L(a_{L-1} w_L)$$

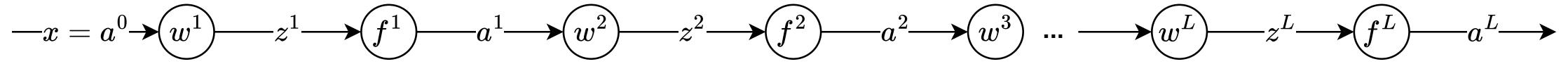
Forward Pass 



To update w_1 we need $\frac{\partial loss}{\partial w_1}$

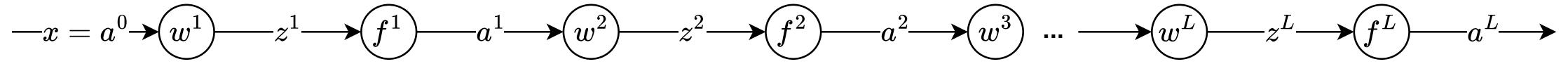


$$\frac{\partial \text{loss}}{\partial w_1} = \frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1} \frac{\partial \text{loss}}{\partial a_1}$$



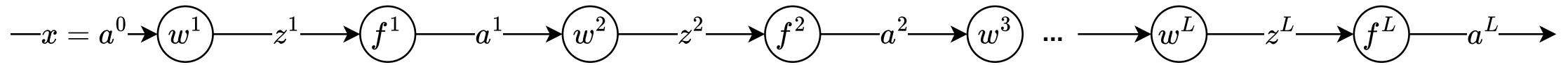
Suppose all activation functions in hidden layers are \tanh

$$\frac{\partial \text{loss}}{\partial w_1} = \frac{\partial z_1}{\partial w_1} \underbrace{\frac{\partial a_1}{\partial z_1}}_{1-a_1^2} \frac{\partial \text{loss}}{\partial a_1}$$



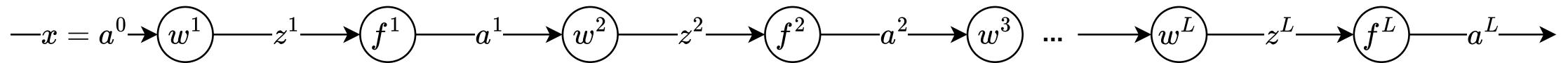
Suppose all activation functions in hidden layers are \tanh

$$\frac{\partial \text{loss}}{\partial w_1} = \underbrace{\frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1}}_{x(1-a_1^2)} \frac{\partial \text{loss}}{\partial a_1}$$



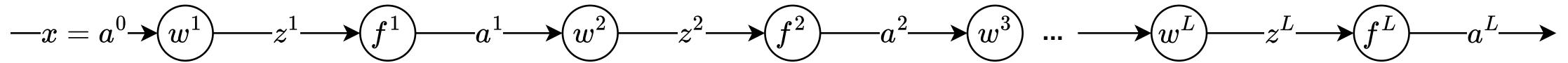
$$\frac{\partial \text{loss}}{\partial w_1} = \underbrace{\frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1}}_{x(1-a_1^2)} \frac{\partial \text{loss}}{\partial a_1}$$

$$\frac{\partial \text{loss}}{\partial a_1} = ?$$



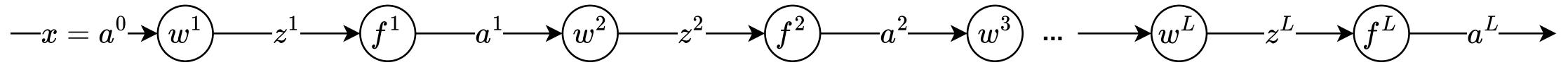
$$\frac{\partial \text{loss}}{\partial w_1} = \underbrace{\frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1}}_{x(1-a_1^2)} \frac{\partial \text{loss}}{\partial a_1}$$

$$\frac{\partial \text{loss}}{\partial a_1} = \frac{\partial z_2}{\partial a_1} \frac{\partial a_2}{\partial z_2} \frac{\partial \text{loss}}{\partial a_2}$$



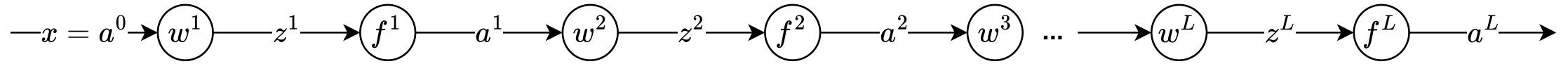
$$\frac{\partial \text{loss}}{\partial w_1} = \underbrace{\frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1}}_{x(1-a_1^2)} \frac{\partial \text{loss}}{\partial a_1}$$

$$\frac{\partial \text{loss}}{\partial a_1} = \underbrace{\frac{\partial z_2}{\partial a_1} \frac{\partial a_2}{\partial z_2}}_{w_2(1-a_2^2)} \frac{\partial \text{loss}}{\partial a_2}$$



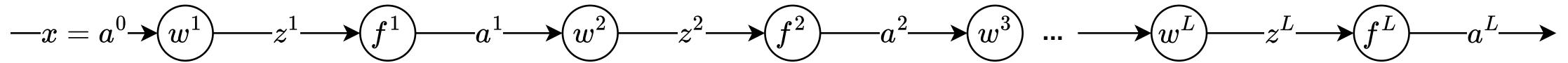
$$\frac{\partial \text{loss}}{\partial w_1} = x(1 - a_1^2)w_2(1 - a_2^2) \frac{\partial \text{loss}}{\partial a_2}$$

$$\frac{\partial \text{loss}}{\partial a_2} =$$



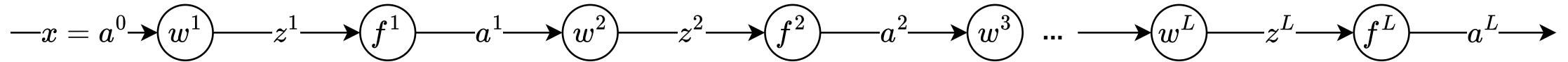
$$\frac{\partial \text{loss}}{\partial w_1} = x(1 - a_1^2)w_2(1 - a_2^2) \frac{\partial \text{loss}}{\partial a_2}$$

$$\frac{\partial \text{loss}}{\partial a_2} = \frac{\partial z_3}{\partial a_2} \frac{\partial a_3}{\partial z_3} \frac{\partial \text{loss}}{\partial a_3}$$



$$\frac{\partial \text{loss}}{\partial w_1} = x(1 - a_1^2)w_2(1 - a_2^2) \frac{\partial \text{loss}}{\partial a_2}$$

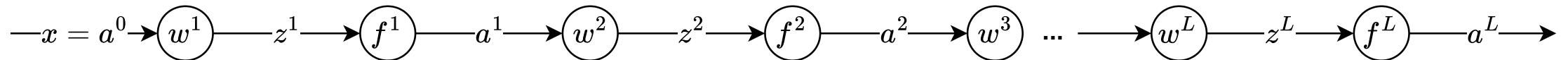
$$\frac{\partial \text{loss}}{\partial a_2} = \underbrace{\frac{\partial z_3}{\partial a_2} \frac{\partial a_3}{\partial z_3}}_{w_3(1 - a_3^2)} \frac{\partial \text{loss}}{\partial a_3}$$



$$\frac{\partial loss}{\partial w_1} = x(1 - a_1^2)w_2(1 - a_2^2)w_3(1 - a_3^2) \frac{\partial loss}{\partial a_3}$$

$$\frac{\partial loss}{\partial a_3} =$$

⋮



$$\frac{\partial \text{loss}}{\partial w_1} = \frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1} \frac{\partial \text{loss}}{\partial a_2}$$

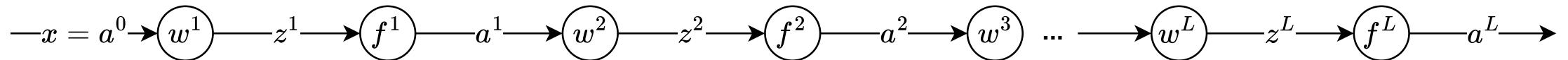
$$\frac{\partial \text{loss}}{\partial a_2} = \frac{\partial z_3}{\partial a_2} \frac{\partial a_3}{\partial z_3} \frac{\partial \text{loss}}{\partial a_3}$$

$$\frac{\partial \text{loss}}{\partial a_3} = \frac{\partial z_4}{\partial a_3} \frac{\partial a_4}{\partial z_4} \frac{\partial \text{loss}}{\partial a_4}$$

⋮

$$\frac{\partial \text{loss}}{\partial a_L} =$$

Suppose loss function is squared loss: $1/2(y - a_L)^2$



$$\frac{\partial \text{loss}}{\partial w_1} = \frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1} \frac{\partial \text{loss}}{\partial a_2}$$

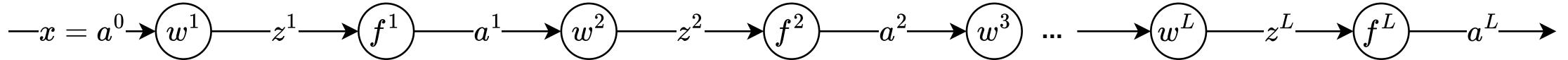
$$\frac{\partial \text{loss}}{\partial a_2} = \frac{\partial z_3}{\partial a_2} \frac{\partial a_3}{\partial z_3} \frac{\partial \text{loss}}{\partial a_3}$$

$$\frac{\partial \text{loss}}{\partial a_3} = \frac{\partial z_4}{\partial a_3} \frac{\partial a_4}{\partial z_4} \frac{\partial \text{loss}}{\partial a_4}$$

⋮

$$\frac{\partial \text{loss}}{\partial a_L} = -(y - a_L)$$

Suppose loss function is squared loss: $1/2(y - a_L)^2$



$$\frac{\partial \text{loss}}{\partial w_1} = \frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1} \frac{\partial \text{loss}}{\partial a_2}$$

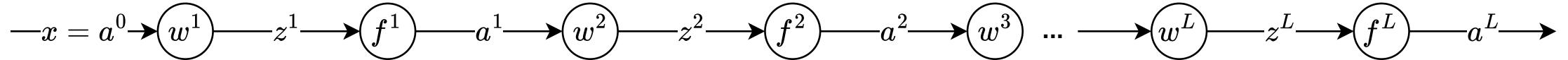
$$\frac{\partial \text{loss}}{\partial a_2} = \frac{\partial z_3}{\partial a_2} \frac{\partial a_3}{\partial z_3} \frac{\partial \text{loss}}{\partial a_3}$$

⋮

$$\frac{\partial \text{loss}}{\partial a_{L-1}} = \frac{\partial z_L}{\partial a_{L-1}} \frac{\partial a_L}{\partial z_L} \frac{\partial \text{loss}}{\partial a_L}$$

$$\frac{\partial \text{loss}}{\partial a_L} = -(y - a_L)$$

Suppose loss function is squared loss: $1/2(y - a_L)^2$



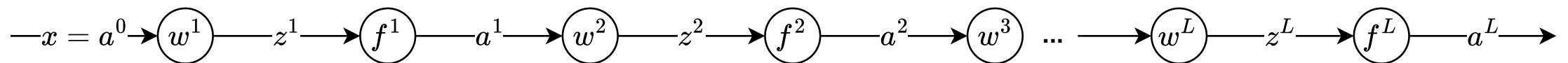
$$\frac{\partial \text{loss}}{\partial w_1} = \frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1} \frac{\partial \text{loss}}{\partial a_2}$$

$$\frac{\partial \text{loss}}{\partial a_2} = \frac{\partial z_3}{\partial a_2} \frac{\partial a_3}{\partial z_3} \frac{\partial \text{loss}}{\partial a_3}$$

$$\vdots$$

$$\frac{\partial \text{loss}}{\partial a_{L-1}} = \frac{\partial z_L}{\partial a_{L-1}} \frac{\partial a_L}{\partial z_L} \frac{\partial \text{loss}}{\partial a_L} = w_L(1 - a_L^2) - (y - a_L)$$

$$\frac{\partial \text{loss}}{\partial a_L} = -(y - a_L)$$



$$\frac{\partial \text{loss}}{\partial w_1} = \frac{\partial z_1}{\partial w_1} \frac{\partial a_1}{\partial z_1} \frac{\partial \text{loss}}{\partial a_2}$$

$$\frac{\partial \text{loss}}{\partial a_2} = \frac{\partial z_3}{\partial a_2} \frac{\partial a_3}{\partial z_3} \frac{\partial \text{loss}}{\partial a_3}$$

⋮
⋮

$$\frac{\partial \text{loss}}{\partial a_{L-1}} = \frac{\partial z_L}{\partial a_{L-1}} \frac{\partial a_L}{\partial z_L} \frac{\partial \text{loss}}{\partial a_L} = w_L(1 - a_L^2) - (y - a_L)$$

$$\frac{\partial \text{loss}}{\partial a_L} = -(y - a_L)$$

Backpropagation Algorithm

General case

Backpropagation algorithm

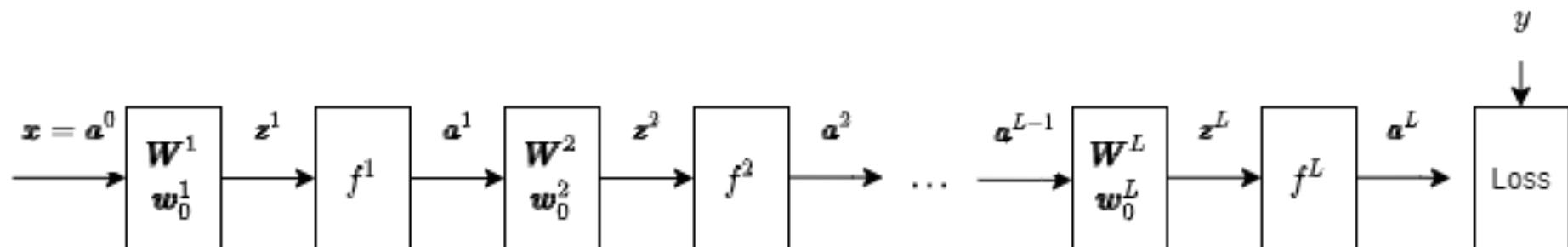
At each iteration backpropgation does a

- *Forward pass* to compute the prediction NN loss value for that iteration.
- *Backward pass* to compute the gradients of the loss with respect to the weights in each layer

Forward Pass

Suppose we have a feed forward neural network such as the one below. To do SGD for a training example (x, y) first we need to calculate the loss function value $loss(NN(x, W), y)$ for that example by using the current weights W .

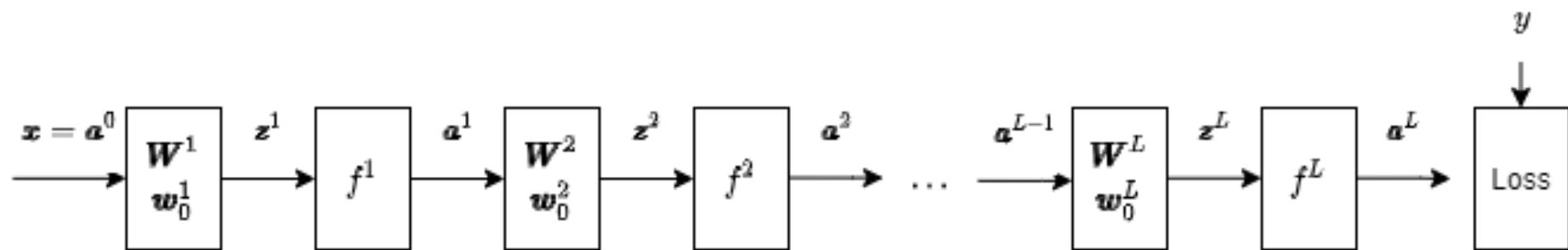
This is called *forward pass*



Forward Pass

Suppose we have a feed forward neural network such as the one below. To do SGD for a training example (x, y) first we need to calculate the loss function value $loss(NN(x, W), y)$ for that example by using the current weights W .

This is called *forward pass*



Note that NN represents the prediction made with x and W , so it's equivalent to a^L .

$$loss(NN(x, W), y) = loss(a^L, y)$$

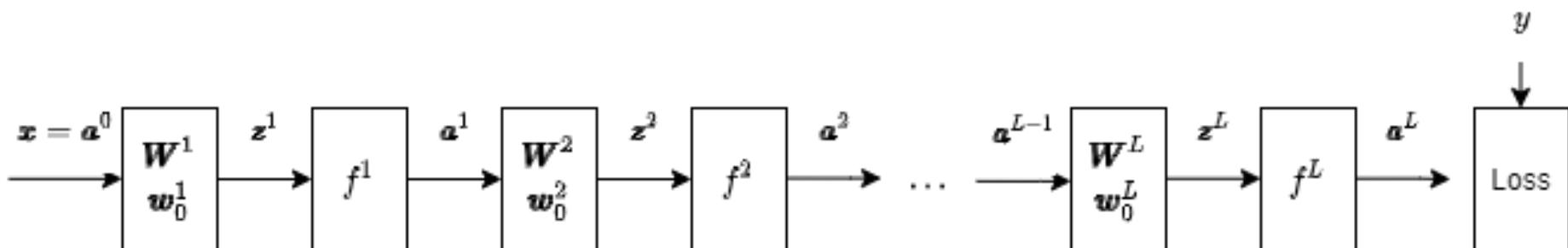
Gradients for the last layer

Lets see how the loss depends on the weights in the final layer W^L .

$$a^L = f^L(z^L)$$

$$z^L = W^{L^T} \cdot a^{L-1}$$

To update the weights at W^L we need the gradient $\frac{\partial \text{loss}}{\partial W^L}$



Gradients for the last layer

From chain rule we can write

$$\frac{\partial \text{loss}}{\partial W^L} = \frac{\partial \text{loss}}{\partial z^L} \frac{\partial z^L}{\partial W^L}$$

And since $z^L = W^{L^T} \cdot a^{L-1}$

$$\frac{\partial z^L}{\partial W^L} = a^{L-1}$$

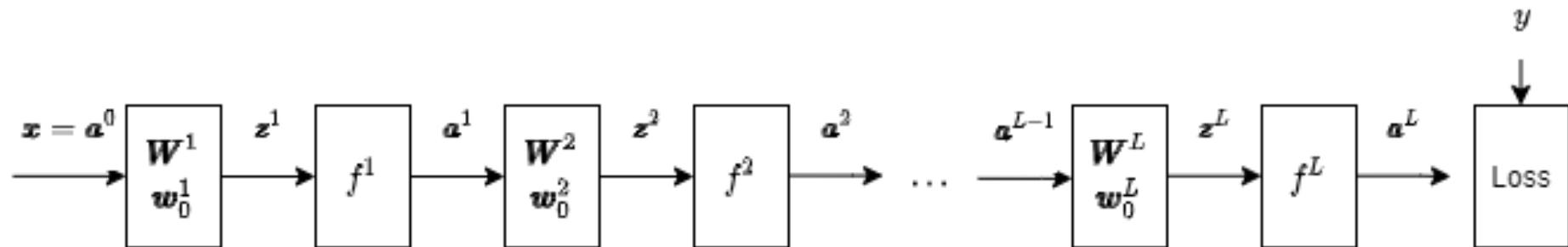
So

$$\frac{\partial \text{loss}}{\partial W^L} = \frac{\partial \text{loss}}{\partial z^L} a^{L-1}$$

Gradients for the second last layer

Lets calculate the gradient for $\frac{\partial \text{loss}}{\partial W^{L-1}}$

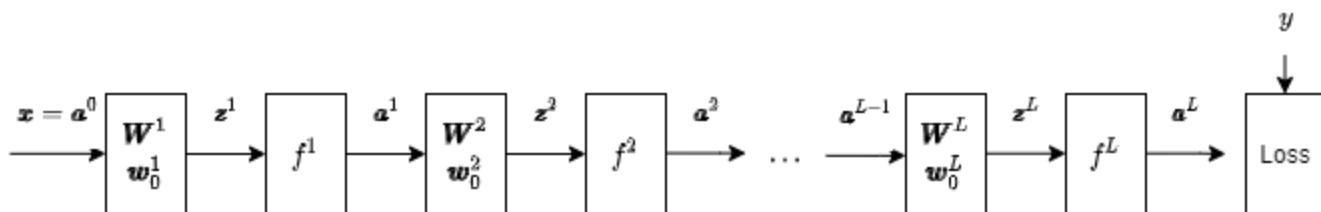
$$\frac{\partial \text{loss}}{\partial W^{L-1}} = \frac{\partial \text{loss}}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial a^{L-1}} \frac{\partial a^{L-1}}{\partial z^{L-1}} \frac{\partial z^{L-1}}{\partial W^{L-1}}$$



Gradients for the second last layer

Lets calculate the gradient for $\frac{\partial \text{loss}}{\partial W^{L-1}}$

$$\begin{aligned}\frac{\partial \text{loss}}{\partial W^{L-1}} &= \frac{\partial \text{loss}}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial a^{L-1}} \frac{\partial a^{L-1}}{\partial z^{L-1}} \frac{\partial z^{L-1}}{\partial W^{L-1}} \\ &= \frac{\partial \text{loss}}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial a^{L-1}} \frac{\partial a^{L-1}}{\partial z^{L-1}} a^{L-2}\end{aligned}$$



Gradients for any layer

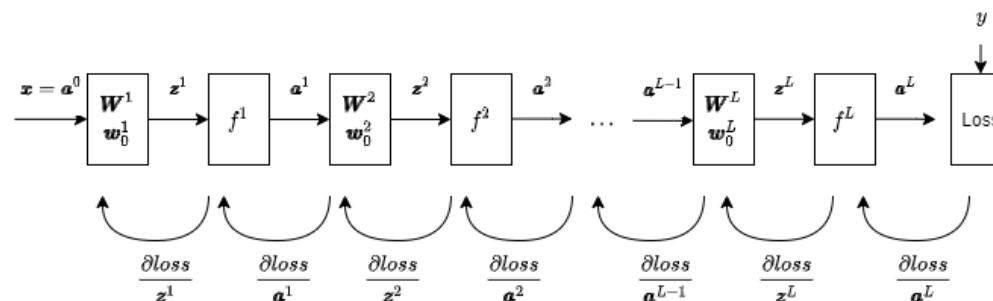
$$\frac{\partial \text{loss}}{\partial W^l} = \frac{\partial \text{loss}}{\partial z^l} a^{l-1}$$

Gradients for the first layer

$$\frac{\partial \text{loss}}{\partial W^1} = \frac{\partial \text{loss}}{\partial z^1} a^0 = \frac{\partial \text{loss}}{\partial z^1} x$$

And the gradient of loss with respect to the pre-activation in the first layer is obtained by repeatedly applying the *chain rule*.

$$\frac{\partial \text{loss}}{\partial z^1} = \frac{\partial \text{loss}}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial a^{L-2}} \cdot \dots \cdot \frac{\partial a^2}{\partial z^2} \cdot \frac{\partial z^2}{\partial a^1} \cdot \frac{\partial a^1}{\partial z^1}$$



Gradients for the first layer

$$\frac{\partial \text{loss}}{\partial W^1} = \frac{\partial \text{loss}}{\partial z^1} a^0 = \frac{\partial \text{loss}}{\partial z^1} x$$

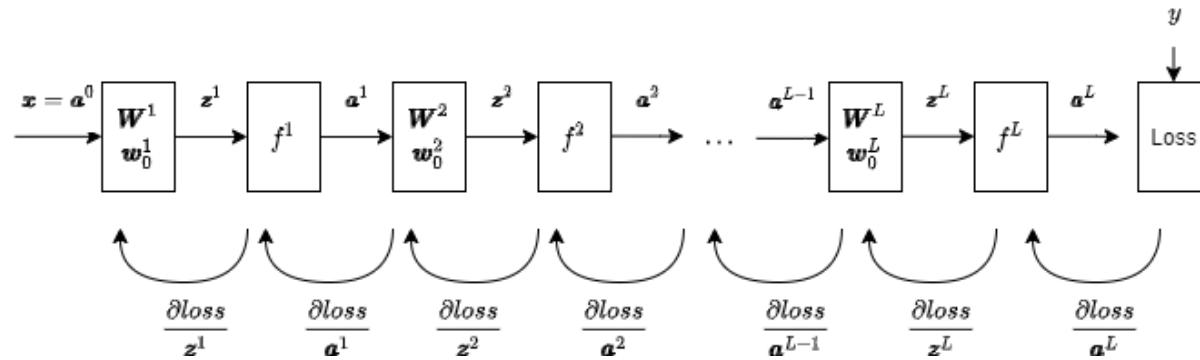
And the gradient of loss with respect to the pre-activation in the first layer is obtained by repeatedly applying the *chain rule*.

$$\frac{\partial \text{loss}}{\partial z^1} = \underbrace{\frac{\partial \text{loss}}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial a^{L-2}} \cdots \frac{\partial a^2}{\partial z^2} \cdot \frac{\partial z^2}{\partial a^1} \cdot \frac{\partial a^1}{\partial z^1}}_{\frac{\partial \text{loss}}{\partial z^2}} \underbrace{\frac{\partial \text{loss}}{\partial a^1}}_{\frac{\partial \text{loss}}{\partial a^1}}$$

Backpropagation

- **Forward pass:** compute a and z values in all layers, and the loss value for the example.
- **Backward pass:** compute the gradient of the loss with respect to the weights in each layer, starting at layer L and going back to layer 1.

$$\frac{\partial \text{loss}}{\partial z^1} = \frac{\partial \text{loss}}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial a^{L-2}} \cdot \dots \cdot \frac{\partial a^2}{\partial z^2} \cdot \frac{\partial z^2}{\partial a^1} \cdot \frac{\partial a^1}{\partial z^1}$$



How can Learning Process go Wrong?

Vanishing Gradients

- If derivatives are small/zero, as you keep multiplying them, they will become zero

Exploding Gradients

- If derivatives are large, as you keep multiplying them, they will become really large numbers

Getting the dimensions match

$$\frac{\partial \text{loss}}{\partial z^1} = \frac{\partial \text{loss}}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial a^{L-2}} \cdot \cdots \cdot \frac{\partial a^2}{\partial z^2} \cdot \frac{\partial z^2}{\partial a^1} \cdot \frac{\partial a^1}{\partial z^1}$$

In order to work out the dimensions, we need to write it backwards

Getting the dimensions match

- $\frac{\partial \text{loss}}{\partial a^L}$ is $n^L \times 1$ and depends on the particular loss function you are using.
- $\frac{\partial z^l}{\partial a^{l-1}}$ is $m^l \times n^l$ and is W^l
 - $z^l = W^l \cdot a^{l-1}$

$$\frac{\partial \text{loss}}{\partial z^l} = \frac{\partial a^l}{\partial z^l} \cdot W^{l+1} \cdot \frac{\partial a^{l+1}}{\partial z^{l+1}} \cdot \dots \cdot W^{L-1} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot W^L \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L}$$

Stochastic Gradient Descent (SGD) Algorithm

- for $l = 1$ to L
 - Initialize weights
- for $t = 1$ to T :
 - $i =$ random sample from $\{1, \dots, n\}$
 - $a^0 = x^{(i)}$
// forward pass to compute the output A^L
 - for $l = 1$ to L
 - $z^l = W^{lT} z^{l-1} + w_0^l$
 - $a^l = f^l(z^l)$
 - $Loss = loss(a^L, y^{(i)})$
 - for $l = L$ to 1
 - Compute the gradients and update the weights

SGD - Compute the gradients and update the weights

for $l = L$ to 1

- $\partial \text{loss} / \partial a^l = \text{if } l < L \text{ then } \partial \text{loss} / \partial z^{l+1} \cdot \partial z^{l+1} / \partial a^l \text{ else } \partial \text{loss} / \partial a^L$
- $\partial \text{loss} / \partial z^l = \text{if } l < L \text{ then } \partial \text{loss} / \partial a^l \cdot \partial a^l / \partial z^l \text{ else } \partial \text{loss} / \partial a^L$
- $\partial \text{loss} / \partial W^l = \partial \text{loss} / \partial z^l \cdot \partial z^l / \partial W^l$
- $\partial \text{loss} / \partial w_0^l = \partial \text{loss} / \partial z^l \cdot \partial z^l / \partial w_0^l$
// stochastic gradient update
- $W^l = W^l - \alpha(t) \cdot \partial \text{loss} / \partial W^l$
- $w_0^l = w_0^l - \alpha(t) \cdot \partial \text{loss} / \partial w_0^l$

SGD - Initalizing Weights

- *Weight initialization* is important
- If they start at same values (e.g. 0 weights), this will prevent them moving in useful directions
- We want keep the initial weights small that gradients are non-zero, and gradient descent will have useful signal about which way to go.
- A good strategy is to start from a normal distribution with mean 0 and standard deviation $1/m$ where m is the number of inputs to the unit.

for $l = 1$ to L

- $W_{ij}^l \sim \text{Gaussian}(0, 1/m^l)$
- $w_{0j}^l \sim \text{Gaussian}(0, 1)$

Stochastic Gradient Descent (SGD) Algorithm

- for $l = 1$ to L
 - Initialize weights
- for $t = 1$ to T :
 - $i =$ random sample from $\{1, \dots, n\}$
 - $a^0 = x^{(i)}$
// forward pass to compute the output A^L
 - for $l = 1$ to L
 - $z^l = W^{lT} z^{l-1} + w_0^l$
 - $a^l = f^l(z^l)$
 - $Loss = loss(a^L, y^{(i)})$
 - for $l = L$ to 1
 - Compute the gradients and update the weights

Loss functions and activation functions at the last layer

Loss functions and activation functions at the last layer

Activation functions in the last layer and the associated loss functions

$loss$	f^L
squared (quadratic)	linear
hinge	linear
negative log likelihood (NLL)	sigmoid
negative log likelihood multiclass (NLLM)	softmax

Binary classification and log-likelihood loss function

- For classification, the step function seems like a natural choice but it's not convenient for gradient-based learning as its derivative is discontinuous.
- Negative log likelihood (NLL) is a better option here. To work with NLL we label y values as $y \in \{0, 1\}$.

$$\text{loss}_{\text{nll}}(\text{guess}, \text{actual}) = -(\text{actual} \cdot \log(\text{guess}) + (1 - \text{actual}) \cdot \log(1 - \text{guess}))$$

Binary classification and hinge loss function

- Hinge loss also provides us a way to deal with binary classification. The hinge loss is defined as
- In hinge loss we label actual values as $y \in \{+1, -1\}$.

$$\text{loss}_h(\text{guess}, \text{actual}) = \max(1 - (\text{guess} \cdot \text{actual}), 0)$$

Multi-class classification

We can extend NLL to multi-classes with K classes. The training examples are represented with one-hot vector $y = [y_1, \dots, y_K]^T$.

Multi-class classification

We can extend NLL to multi-classes with K classes. The training examples are represented with one-hot vector $y = [y_1, \dots, y_K]^T$.

Assume that our network uses softmax in the last layer so the output is $a = [a_1, \dots, a_K]^T$, which represents a probability distribution over K possible classes. The probability that the network predicts the correct class is $\prod_{k=1}^K a_k^{y_k}$, and the log of that probability that it is correct is $\sum_{k=1}^K y_k \log a_k$

$$\text{loss}_{nllm}(\text{guess}, \text{actual}) = - \sum_{k=1}^K \text{actual}_k \cdot \log(\text{guess}_k)$$

We call this NLLM for negative log likelihood multiclass.

COGS514 - Cognition and Machine Learning

Neural Networks - Making Them Work

Neural Networks - Making Them Work

- Modular structure of the Neural Networks makes it easy to compute the gradient each layer.
- *Stochastic Gradient* descent can be applied as the loss function is a sum over terms.
- Training deep NN with many layers can be challenging
 - Overfitting, vanishing gradients, exploding gradients
- We will examine strategies to handle the *step-size* α , and training.

Some Problems

1. Hard to determine α in Stochastic Gradient Descent
2. Vanishing & Exploding Gradients
- 3.

Batches

Suppose we have the objective function

$$\Phi(w) = \sum_{i=1}^n loss(h(x^{(i)}, w), y^{(i)})$$

Batch Gradient Descent

- Start from an initial point $w^{(0)} \in \mathbb{R}^d$
- **for** $t = 1$ to T
 - $w^{(t)} = w^{(t-1)} - \alpha \sum_{i=1}^n \nabla_w \text{loss}(h(x^{(i)}, w), y^{(i)})$
- **return** $w^{(t)}$

We sum up the gradient loss at each training example, and take a big step in the negative direction of the gradient.

Stochastic Gradient Descent

- Start from an initial point $w^{(0)} \in \mathbb{R}^d$
- **for** $t = 1$ to T
 - randomly select $i \in 1, 2, \dots, n$
 - $w^{(t)} = w^{(t-1)} - \alpha \nabla_w \text{loss}(h(x^{(i)}, w), y^{(i)})$
- **return** $w^{(t)}$

We randomly select one training example, take a small step in the negative direction of the gradient loss at that example. On average, these small steps move in the same direction as the gradient.

Batches

Batch Gradient Descent

- Takes a step in the exact gradient direction.
- Large computational requirements especially if the dataset is large.

Stochastic Gradient Descent

- Efficient
- Can make good progress even with using only a part of the data.
- If there is high variability in the data, require small α to effectively average over the individual steps moving in useful directions

Mini-batch Gradient Descent

- Combination of batch and stochastic gradient descent.
- Mini-batch size k
- Select k distinct data points randomly from the dataset, and update based on their contributions to the gradient

$$\sum_{i=1}^k \nabla_w \text{loss}(h(x^{(i)}, w), y^{(i)})$$

- k random data points can be selected by randomly shuffling the (*indices of the*) dataset

Mini-batch Gradient Descent

Let n be the number of examples in the training data and k be the mini-batch size.

- **for** $t = 1$ to T
 - randomly shuffle data
 - **for** $j = 1$ to n/k
 - $w = w - \alpha \sum_{i=(j-1)k}^{jk} \nabla_w \text{loss}(h(x^{(i)}, w), y^{(i)})$
- **return** w

Vanishing / Exploding Gradients

$$\frac{\partial \text{loss}}{\partial W^l} = \text{loss} \partial z^l a^{l-1}$$

Vanishing / Exploding Gradients

$$\frac{\partial \text{loss}}{\partial W^l} = \frac{\partial \text{loss}}{\partial z^l} a^{l-1}$$

$$\frac{\partial \text{loss}}{\partial W^L} = \frac{\partial \text{loss}}{\partial z^L} a^{L-1} = \left(\frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{L-1}$$

Vanishing / Exploding Gradients

$$\frac{\partial \text{loss}}{\partial W^l} = \frac{\partial \text{loss}}{\partial z^l} a^{l-1}$$

$$\frac{\partial \text{loss}}{\partial W^L} = \frac{\partial \text{loss}}{\partial z^L} a^{L-1} = \left(\frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{L-1}$$

$$\frac{\partial \text{loss}}{\partial W^{L-1}} = \frac{\partial \text{loss}}{\partial z^{L-1}} a^{L-2} = \left(\frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot W^L \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{L-2}$$

Vanishing / Exploding Gradients

$$\frac{\partial \text{loss}}{\partial W^l} = \frac{\partial \text{loss}}{\partial z^l} a^{l-1}$$

$$\frac{\partial \text{loss}}{\partial W^L} = \frac{\partial \text{loss}}{\partial z^L} a^{L-1} = \left(\frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{L-1}$$

$$\frac{\partial \text{loss}}{\partial W^{L-1}} = \frac{\partial \text{loss}}{\partial z^{L-1}} a^{L-2} = \left(\frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot W^L \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{L-2}$$

$$\frac{\partial \text{loss}}{\partial W^l} = \frac{\partial \text{loss}}{\partial z^l} a^{l-1} = \left(\frac{\partial a^l}{\partial z^l} \cdot W^{l+1} \cdot \frac{\partial a^{l+1}}{\partial z^{l+1}} \cdot \dots \cdot W^{L-1} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot W^L \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{l-1}$$

Vanishing / Exploding Gradients

$$\frac{\partial \text{loss}}{\partial W^l} = \frac{\partial \text{loss}}{\partial z^l} a^{l-1}$$

$$\frac{\partial \text{loss}}{\partial W^L} = \frac{\partial \text{loss}}{\partial z^L} a^{L-1} = \left(\frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{L-1}$$

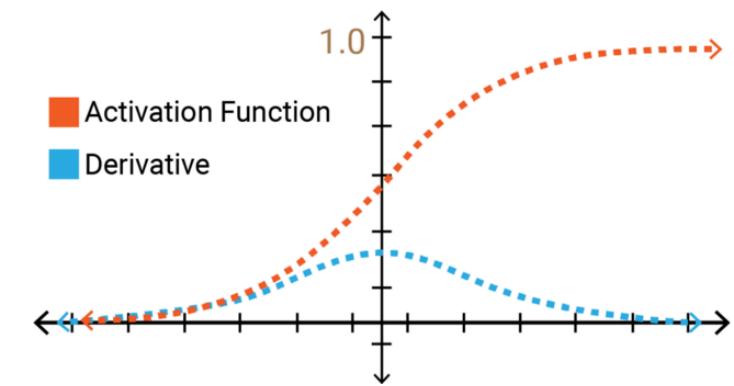
$$\frac{\partial \text{loss}}{\partial W^{L-1}} = \frac{\partial \text{loss}}{\partial z^{L-1}} a^{L-2} = \left(\frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot W^L \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{L-2}$$

$$\frac{\partial \text{loss}}{\partial W^l} = \frac{\partial \text{loss}}{\partial z^l} a^{l-1} = \left(\frac{\partial a^l}{\partial z^l} \cdot W^{l+1} \cdot \frac{\partial a^{l+1}}{\partial z^{l+1}} \cdot \dots \cdot W^{L-1} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot W^L \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial \text{loss}}{\partial a^L} \right) a^{l-1}$$

- To compute $\partial \text{loss} / \partial W_l$ gradient is multiplied by derivative of activation layers $\frac{\partial a}{\partial z}$ and weight matrices W from L to l
- If these terms are large, then the gradient will increase exponentially and *explode*
- If these terms are small, than the gradient will decrease exponentially and *vanish*

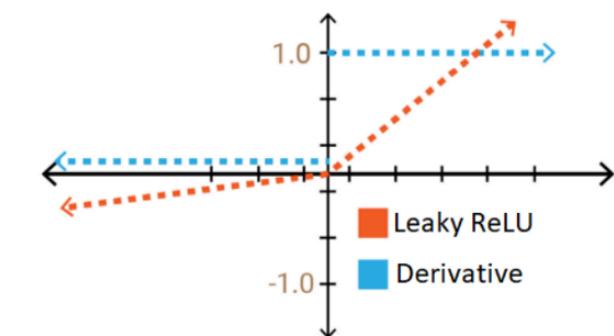
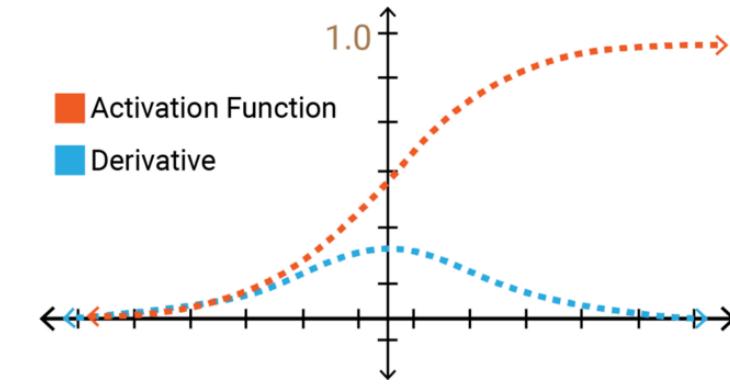
Vanishing / Exploding Gradients

- Sigmoid function used to be a popular activation function a for hidden layers. However, derivative of sigmoid $\frac{\partial a}{\partial z}$ is small when z is very large or very small. This leads to *vanishing gradients*.
 - Gradients of earlier units become 0.



Vanishing / Exploding Gradients

- Sigmoid function used to be a popular activation function a for hidden layers. However, derivative of sigmoid $\frac{\partial a}{\partial z}$ is small when z is very large or very small. This leads to *vanishing gradients*.
 - Gradients of earlier units become 0.
- ReLU overcomes this problem as the derivative of ReLU is either 0 or 1.



ReLU and Leaky ReLU

$$\text{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} = \max(0, z)$$

- ReLU does not saturate for positive values, and is fast to compute.
- When ReLU activation function is used, some neurons outputs only zero. This is known as the *dying neurons* problem.

ReLU and Leaky ReLU

$$\text{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} = \max(0, z)$$

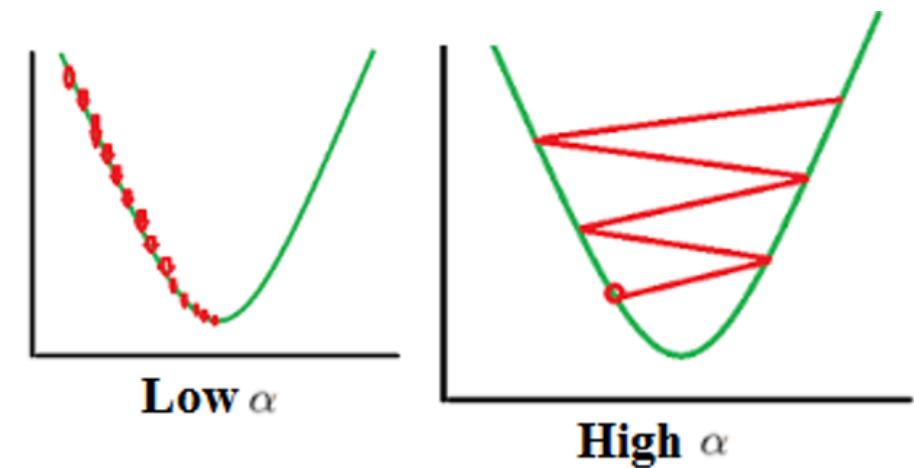
- ReLU does not saturate for positive values, and is fast to compute.
- When ReLU activation function is used, some neurons outputs only zero. This is known as the *dying neurons* problem.
- *Leaky ReLU* can be used to avoid this problem

$$\text{Leaky ReLU}(z) = \begin{cases} cz & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} = \max(cz, z)$$

where c is a small number typically set to 0.01.

Optimization - Step-size α

- Picking a value for α is difficult.
- Small α values slows down convergence
- Large α values leads to oscillation and divergence.
- Further problems in *stochastic* or *mini-batch* gradient descent as we need to decrease the step size for convergence.



Adaptive Step Size

- We may want to have different α values in different layers of the neural network.
- We will have a independent step-size parameter for *each weight*, and update them based on how the gradient updates are going.
- We can use methods like running averages as strategies for computing α .

Running averages

Running average is used for estimating a weighted average of a data sequence.

Let our data sequence be a_1, a_2, \dots . A sequence of running average values A_0, A_1, A_2, \dots can be defined as follows

$$A_0 = 0$$

$$A_t = \gamma_t A_{t-1} + (1 - \gamma_t) a_t$$

where $\gamma_t \in (0, 1)$. If γ_t is a constant, this is called a *moving average*.

Running Averages

$$\begin{aligned} A_T &= \gamma A_{T-1} + (1 - \gamma) a_T \\ &= \gamma(\gamma A_{T-2} + (1 - \gamma) a_{T-1}) + (1 - \gamma) a_T \\ &= \gamma(\gamma(\gamma A_{T-3} + (1 - \gamma) a_{T-2}) + (1 - \gamma) a_{T-1}) + (1 - \gamma) a_T \\ &= \sum_{t=0}^T \gamma^T - t(1 - \gamma) a_t \end{aligned}$$

The inputs a_t , closer to the end of the sequence have higher effect on A_t than early inputs.

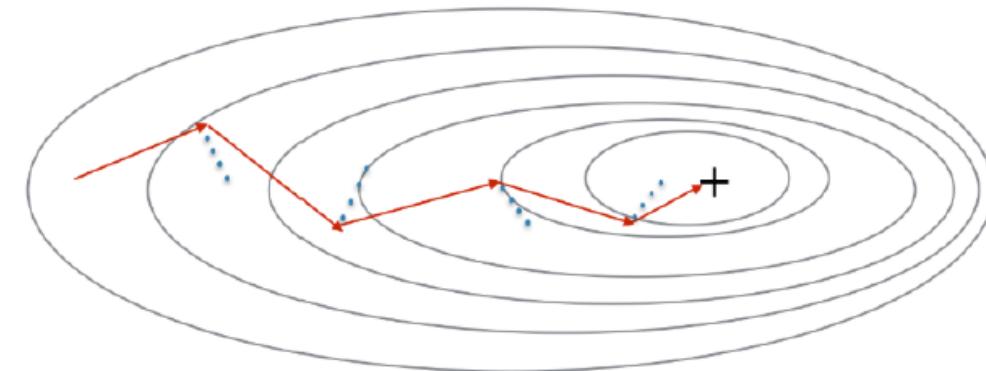
Momentum

- The simplest method is *momentum*
- We average recent gradient updates. If they have been bouncing back and forth in some direction, averaging takes out that component of the motion.
 - Blue arrow shows the direction of the gradient, red arrows show the update after gradient descent with momentum.

$$V_0 = 0$$

$$V_t = \gamma V_{t-1} + \alpha \nabla_W \Phi(W_{t-1})$$

$$W_t = W_{t-1} - V_t$$



Momentum

Suppose $\alpha = \alpha'(1 - \gamma)$. Then:

$$M_0 = 0$$

$$M_t = \gamma M_{t-1} + (1 - \gamma) \nabla_W \Phi(W_{t-1})$$

$$W_t = W_{t-1} - \alpha' M_t$$

This is like doing an update with step size α' on a moving average of the gradients with parameter γ .

Momentum

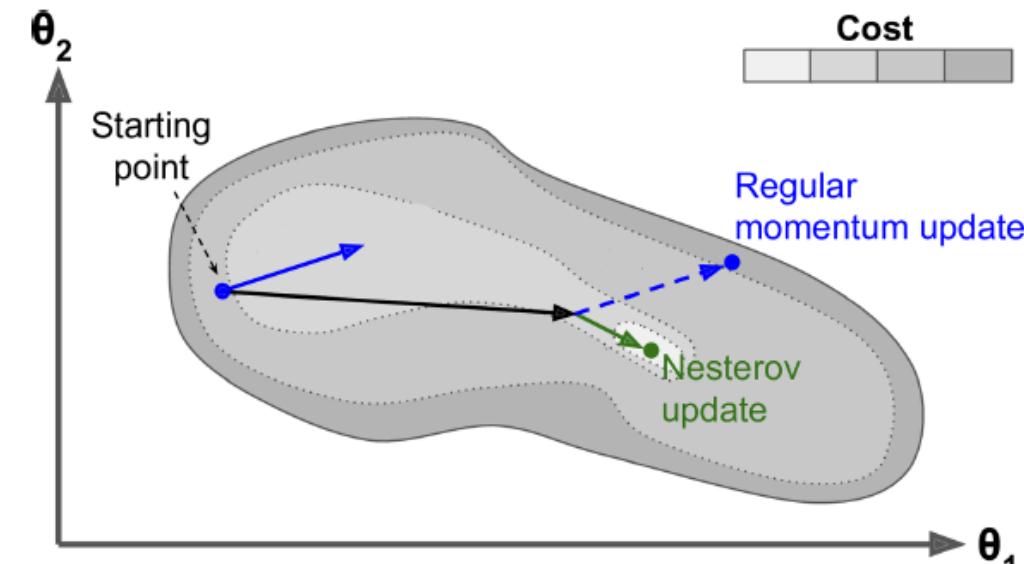
- V_t will be larger in dimensions that consistently has the same sign for ∇_W
- We need to set α and γ .
- γ is often set to ~ 0.9

Nesterov Accelerated Gradient

- A small variant to momentum that speeds up the optimization
- Calculates the position of the gradient not at W but at slightly ahead in the direction of the momentum at $W + \gamma V$

$$V_t = \gamma V_{t-1} + \alpha \nabla_W \Phi(W_{t-1} + \gamma V_{t-1})$$

$$W_t = W_{t-1} - V_t$$



Adadelta

Take larger steps where $\Phi(W)$ is flat, and small steps when it is steep. If we apply this independently to each weight, this is called *adadelta* (a variant of *adagrad*).

Adadelta

Take larger steps where $\Phi(W)$ is flat, and small steps when it is steep. If we apply this independently to each weight, this is called *adadelta* (a variant of *adagrad*).

Let w_j be any weight in the network.

$$g_{t,j} = \nabla_W \Phi(W_{t-1})_j$$

$$G_{t,j} = \gamma G_{t-1,j} + (1 - \gamma) g_{t,j}^2$$

$$W_{t,j} = W_{t-1,j} - \frac{\alpha}{\sqrt{G_{t,j} + \epsilon}} g_{t,j}$$

The sequence $G_{t,j}$ is a moving average of the square of j^{th} component of the gradient. We square it in order to be insensitive to the sign. We divide the step by $\sqrt{G_{t,j} + \epsilon}$ which is larger when the surface is steeper in direction j at point W_{t-1}

Adam

Adaptive Moment Estimation (Adam) has become the default method of managing step sizes in neural networks. It combines the ideas *momentum* and *adadelta*. We have moving averages of *gradient* and of *squared gradient*, which reflect estimates of the mean and variance of the gradient for j

$$g_{t,j} = \nabla_W \Phi(W_{t-1})_j$$

$$m_{t,j} = B_1 m_{t-1,j} + (1 - B_1) g_{t,j}$$

$$v_{t,j} = B_2 v_{t-1,j} + (1 - B_2) g_{t,j}^2$$

$$W_{t,j} = W_{t-1,j} - \frac{\alpha}{\sqrt{\hat{v}_{t,j} + \epsilon}} \hat{m}_{t,j}$$

Adam

If we initialize $m_0 = v_0 = 0$, they will always be biased (slightly too small). We will correct for that bias

$$\hat{m}_{t,j} = \frac{m_{t,j}}{1 - B_1^t}$$

$$\hat{v}_{t,j} = \frac{v_{t,j}}{1 - B_2^t}$$

$$W_{t,j} = W_{t-1,j} - \frac{\alpha}{\sqrt{\hat{v}_{t,j} + \epsilon}} \hat{m}_{t,j}$$

Note that B_1^t and B_2^t are respectively B_1 and B_2 raised to the power t .

Adam

If we were to expand $m_{t,j}$ in terms of $m_{0,j}$ and $g_{0,j}, g_{1,j}, \dots, g_{t,j}$, the coefficients would sum to 1.

However coefficient of m_{0j} is B_1^t and since $m_{0ij} = 0$ the sum of coefficient of non-zero terms is $1 - B_1^t$. Therefore the $\frac{1}{1-B_1^t}$ correction is done.

Adam

- Adam has a step size that takes the steepness into account (like *adadelta*)
- It also tends to move in the same direction (like *momentum*)
- Kingma and Ba suggests $B_1 = 0.9, B_2 = 0.999, \alpha = 10^{-8}$.

Animations and Further Resources

- [SGD Animations](#)
- [Alec Radford's Animations](#)
- [Ruder's Blog Post for Gradient Descent Variants](#)

Regularization

Regularization

$$\Phi(W) = \sum_{i=1}^n loss(h(x^{(i)}, W), y^{(i)})$$

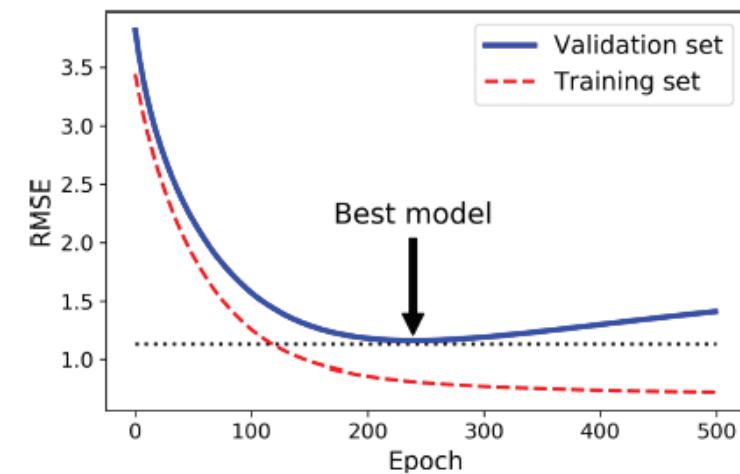
- Up to now, we have only optimized loss on training data. This may lead to overfitting.
- In deep neural networks trained with large amount of data, overfitting is not a big problem.
- This conflict with the current theoretical understanding, and is a area of research.

Epoch

- When training deep neural networks, we run stochastic gradient descent (variants) on all data in multiple cycles
- An epoch is one cycle of running SGD in all of the data
- Loss on training data will get lower at each epoch

Regularization - Early Stopping

- At every *epoch*, evaluate the loss of the current W on a *validation dataset*.
- Loss on the training set goes down with each iteration
- Loss on the validation set will initially decrease, but then it will start to increase.
- Once you see the validation loss is systematically increasing, stop training and return to W that had the lowest loss on the validation set.



Regularization - Weight Decay

- Weight decay is to penalize the norm of weights (like ridge regression)

$$\Phi(W) = \sum_{i=1}^n loss(NN(x, W), y^{(i)}) + \lambda \|W\|^2$$

This results in an update of the form

$$\begin{aligned} W_t &= W_{t-1} - \alpha \left((\nabla_W loss(NN(x, W_{t-1}), y^{(i)})) + \lambda W_{t-1} \right) \\ &= W_{t-1}(1 - \lambda\alpha) - \alpha(\nabla_W loss(NN(x, W_{t-1}), y^{(i)})) \end{aligned}$$

Decays W_{t-1} by a factor of $(1 - \lambda\alpha)$ and takes a gradient of the step.

Regularization - Adding Noise to the Training Data

- We can add a small amount of normally distributed noise values to your gradient data before computing each gradient
- This makes it difficult for network to overfit a particular training data, if they are changed slightly on each training step

Regularization

Early stopping, weight decay and adding noise to the training data have been shown to have similar effects. [Bishop, 1995](#)

Dropout

- *Dropout* is a regularization technique that was designed to work with deep neural networks.
- Rather than perturbing the data, we perturb the network every time we train.
- At each iteration, we select a random set of units in each layer, and remove them from the network.
- This enables network to not to rely on any small subset of weight to do the computation. All units need to participate.
- Makes the network robust to data perturbations

Dropout

- For each unit, randomly with probability p set $a_j^l := 0$.
 - This unit will have no contribution to the output and no gradient update.
- When the training is complete, multiple all weights by p to achieve the same average activation levels.

Dropout - Implementation

- In order to implement dropout

$$a^l = f(z^l) * d^l$$

where $*$ is the element wise multiplication, d^l is a vector of 0s and 1s randomly drawn with probability p .

- p is commonly set to 0.5, but you can experiment different values.

Standardization (Normalization)

- Standardization scales the data centered on 0 with unit variance.

$$\frac{x - \mu}{\sigma}$$

- We standardize (normalize) the initial W values from a Normal distribution with zero mean and unit variance.
- During training we update parameters to different extents, and this normalization is lost.
 - This slows down training and amplifies changes as you need to change your weights to both to compensate the changes in the inputs and to improve prediction.

Batch Normalization

- *Batch normalization* standardize the input values for every mini-batch, and those changes are back-propagated as well.
 - The scale of the inputs in each layer stays the same, no matter how the weights in previous layers change.
 - Weight update computation needs to take these standardizations into account.
- When normalization is a part of the model architecture, we can use higher learning rates, and pay less attention to initial parameters.

Batch Normalization - Regularization

- Batch normalization has a *regularizing effect* as each mini-batch of data is mildly perturbed. This prevents the network from exploiting particular data points.
- *Batch normalization* is used as an alternative to dropout and tends to achieve better performance.

COGS514 - Cognition and Machine Learning

Convolutional Neural Networks

Convolutional Neural Networks

- *Fully connected* feed-forward neural networks are useful if we don't know anything about the mapping from inputs to outputs.
- What if we know something about the problem?
 - If we can reflect our knowledge in the network structure, we can decrease the time and data requirements

Signal Processing

- is an important application area for neural networks. Signals can be
 - i. **spatial**
Two dimensional camera images, three dimensional computerized tomography scans
 - ii. **temporal**
Sound, speech, music
- If we are working on a signal processing problem, we can make use of the *invariant properties* of the problem.

Image Classification

- Suppose you are trying to classify images.

Input: image(pixels) → **Output:** classification (cat or dog)

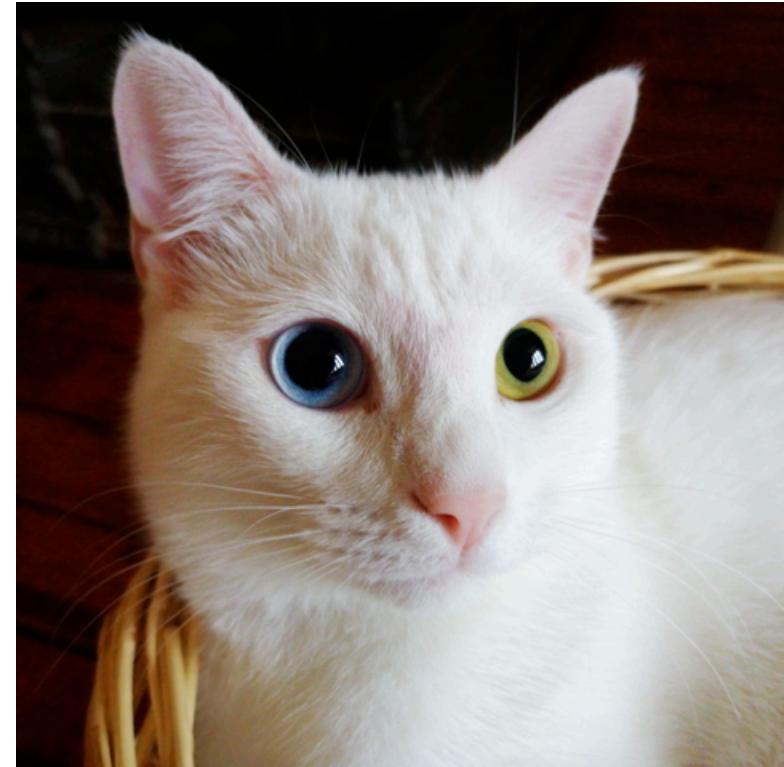
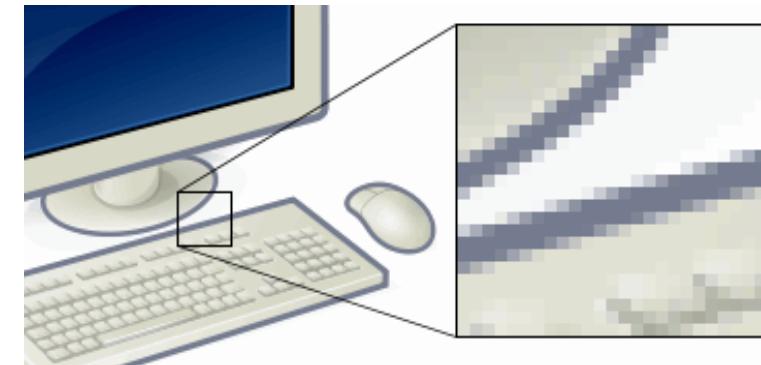


Image Classification

- Suppose you are trying to classify images.

Input: image(pixels) → **Output:** classification (cat or dog)

- Image is described as two dimensional arrays of *pixels*
- In color images, each pixel is represented by three numbers, representing intensity in red, green and blue color channels.



Gray Scale Images



34	34	37	35	38	40	34	
29	30	48	38	42	50	43	
42	43	28	31	62	128	104	
46	36	56	48	104	167	165	
40	46	71	100	130	173	165	
60	42	42	72	124	181	163	
65	37	40	26	91	171	164	

Gray Scale Images

- In grayscale images, each pixel takes a value between 0 and P.
 - In this example, between 0 and 255

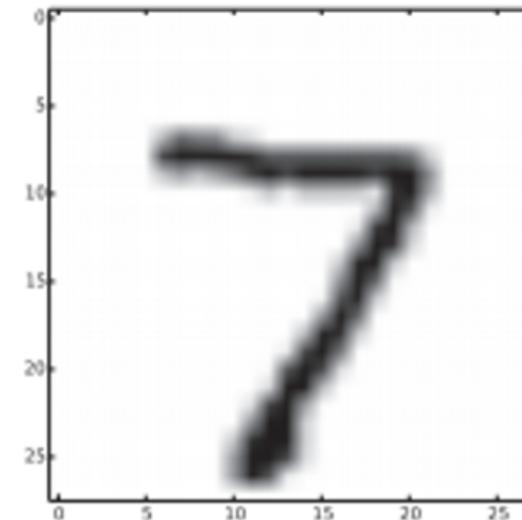


0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	16	122	215	251	238	255	49
3	217	243	255	158	33	226	52	2	0	10	13	232	255	255	36
6	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	111	252	255	248	144	6	0
3	113	255	255	245	255	182	181	248	252	242	220	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	111	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	2	62	255	250	125	3	0
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	140	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	235	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	148	252	252	242	208	36	0
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	245	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Gray Scale Images

- In grayscale images, each pixel takes a value between 0 and P.
- In this example, between 0 and 255.
- How do we input an image to neural network?

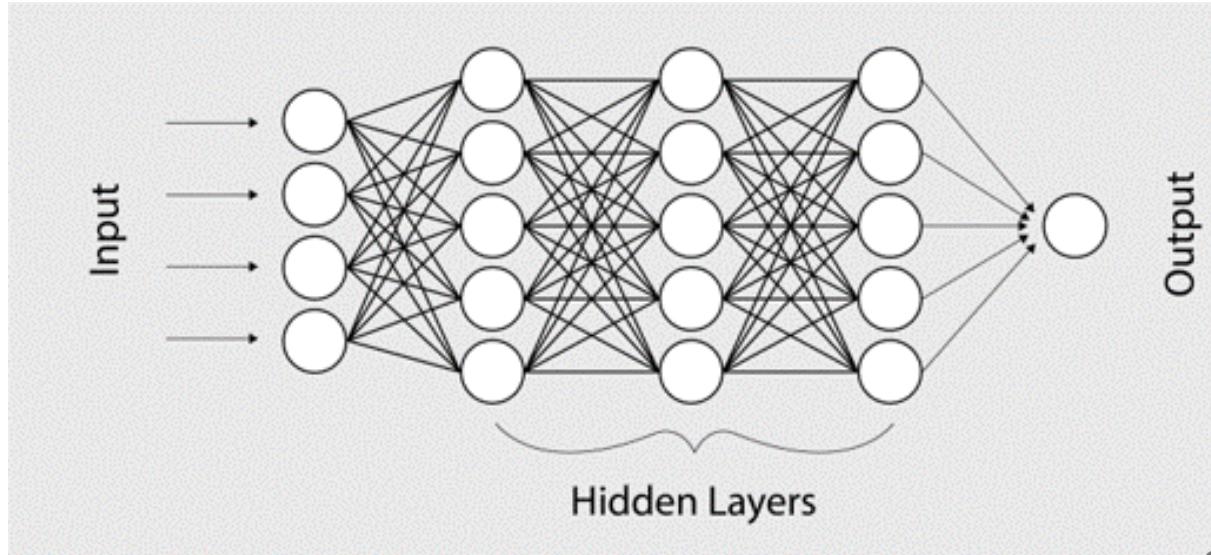


	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	185	159	151	60	36	0	0	0	0	0	0	0	0	0
8	254	254	254	254	241	198	198	198	198	198	198	198	198	170
9	114	72	114	163	227	254	225	254	254	254	254	250	229	254
10	0	0	0	0	17	66	14	67	67	67	59	21	236	254
11	0	0	0	0	0	0	0	0	0	0	0	83	253	209
12	0	0	0	0	0	0	0	0	0	0	22	233	255	83
13	0	0	0	0	0	0	0	0	0	0	129	254	238	44
14	0	0	0	0	0	0	0	0	0	59	249	254	62	0
15	0	0	0	0	0	0	0	0	0	133	254	187	5	0
16	0	0	0	0	0	0	0	0	9	205	248	58	0	0
17	0	0	0	0	0	0	0	0	126	254	182	0	0	0
18	0	0	0	0	0	0	0	75	251	240	57	0	0	0
19	0	0	0	0	0	0	19	221	254	166	0	0	0	0
20	0	0	0	0	0	3	203	254	219	35	0	0	0	0
21	0	0	0	0	0	38	254	254	77	0	0	0	0	0
22	0	0	0	0	31	224	254	115	1	0	0	0	0	0
23	0	0	0	0	133	254	254	52	0	0	0	0	0	0
24	0	0	0	61	242	254	254	52	0	0	0	0	0	0
25	0	0	0	121	254	254	219	40	0	0	0	0	0	0
26	0	0	0	121	254	207	18	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Why does Feed Forward Neural Networks not work well for images?

- Pixels are features
- Treats every pixel separately
- The same handwritten 7 placed a fix pixels above is a completely different image for FFN

	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	185	159	151	60	36	0	0	0	0	0	0	0	0	0
8	254	254	254	254	241	198	198	198	198	198	198	198	198	170
9	114	72	114	163	227	254	225	254	254	254	250	229	254	254
10	0	0	0	0	17	66	14	67	67	67	59	21	236	254
11	0	0	0	0	0	0	0	0	0	0	0	83	253	209
12	0	0	0	0	0	0	0	0	0	0	22	233	255	83
13	0	0	0	0	0	0	0	0	0	0	129	254	238	44
14	0	0	0	0	0	0	0	0	0	59	249	254	62	0
15	0	0	0	0	0	0	0	0	0	133	254	187	5	0
16	0	0	0	0	0	0	0	0	9	205	248	58	0	0
17	0	0	0	0	0	0	0	0	126	254	182	0	0	0
18	0	0	0	0	0	0	0	75	251	240	57	0	0	0
19	0	0	0	0	0	0	19	221	254	166	0	0	0	0
20	0	0	0	0	0	3	203	254	219	35	0	0	0	0
21	0	0	0	0	0	38	254	254	77	0	0	0	0	0
22	0	0	0	0	31	224	254	115	1	0	0	0	0	0
23	0	0	0	0	133	254	254	52	0	0	0	0	0	0
24	0	0	0	61	242	254	254	52	0	0	0	0	0	0
25	0	0	0	121	254	254	219	40	0	0	0	0	0	0
26	0	0	0	121	254	207	18	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Structural Knowledge about Images

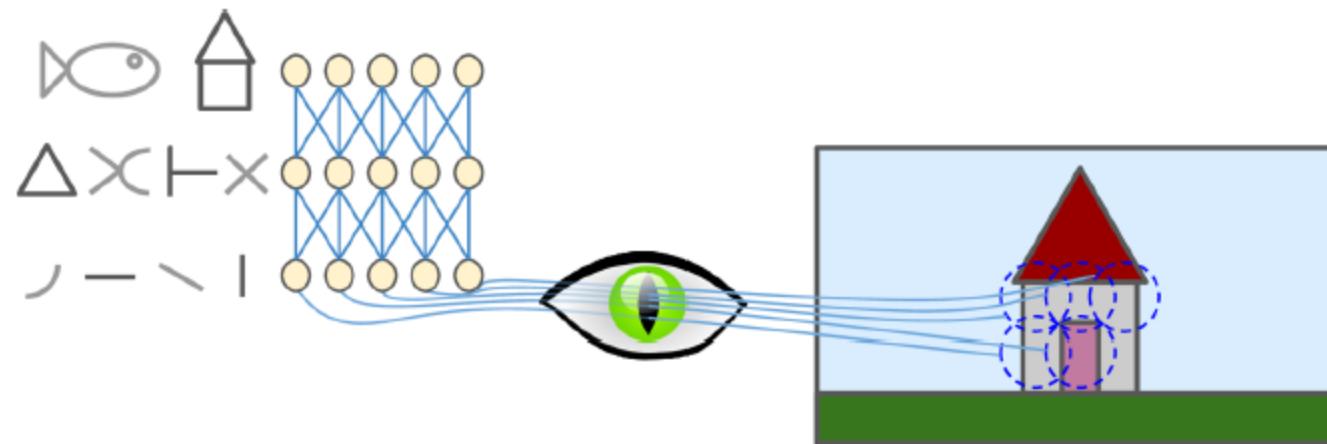
There are two important prior structural knowledge that you can use for your neural network structure

- 1. Spatial locality:** The set of pixels regarding a cat will be near to each other in the image
- 2. Translation invariance:** The pattern of pixels that characterizes a cat is the same regardless where the cat is in the image



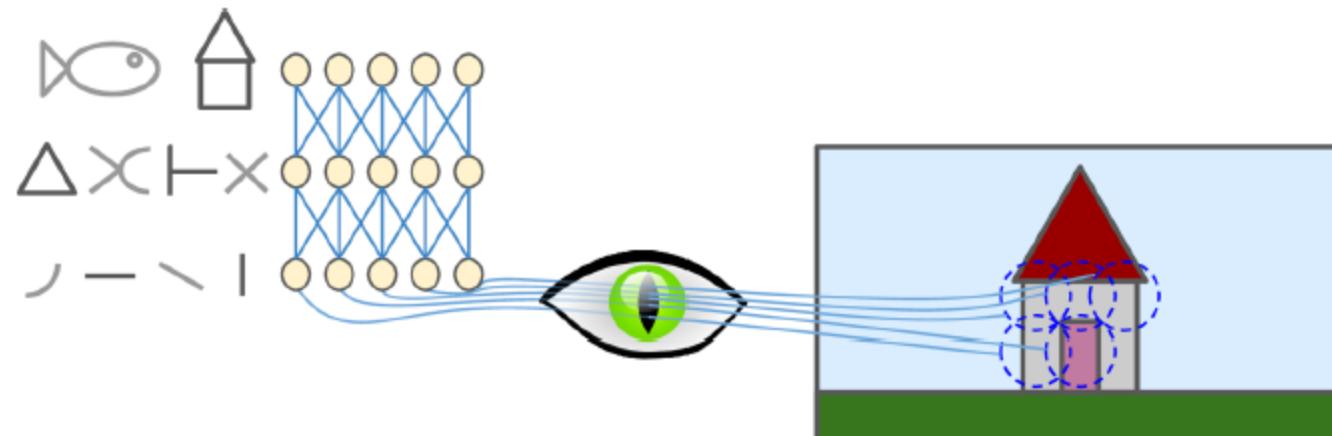
Architecture of Visual Cortex

- Hubel and Wiesel showed that many neurons in the visual cortex reacts only to visual stimuli located in a limited visual field. (Nobel Prize 1981)
- Receptive fields of different neurons may overlap



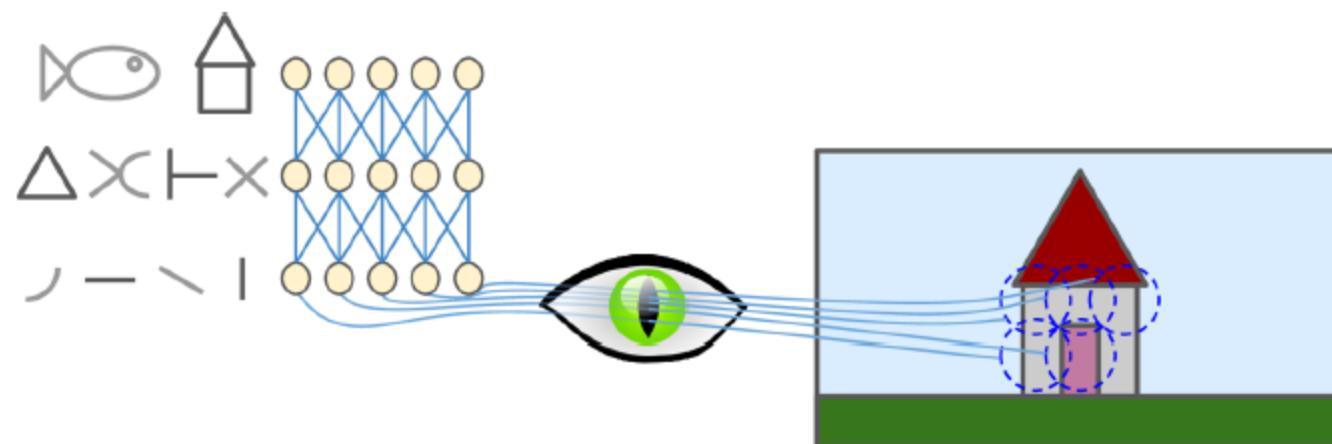
Architecture of Visual Cortex

- Hubel and Wiesel showed that many neurons in the visual cortex reacts only to visual stimuli located in a limited visual field. (Nobel Prize 1981)
- Receptive fields of different neurons may overlap
- Some neurons react to horizontal lines, other neurons react to lines with different orientations.
- Some neurons react to more complex patterns (combinations of lower level patterns)



Architecture of Visual Cortex

- Hubel and Wiesel showed that many neurons in the visual cortex reacts only to visual stimuli located in a limited visual field. (Nobel Prize 1981)
- Receptive fields of different neurons may overlap
- Some neurons react to horizontal lines, other neurons react to lines with different orientations.
- Some neurons react to more complex patterns (combinations of lower level patterns)
- Visual cortex studies inspired *convolutional neural networks*.



filters

Filters

A *filter* is a function that takes in a local spatial neighborhood of pixel values, and detect some pattern in that data.

Filters

- A *filter* is a function that takes in a local spatial neighborhood of pixel values, and detect some pattern in that data.
- Suppose we have a one dimensional image X of size d , and a filter F of size two.

e.g.

$$X = [0, 0, 1, 1, 1, 0, 1, 0, 0, 0]$$

$$F = [-1, +1]$$

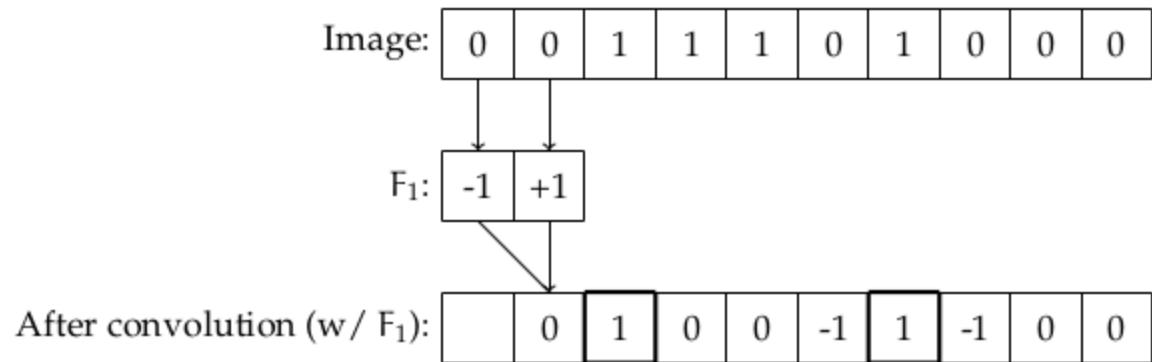
Filters

- A *filter* is a function that takes in a local spatial neighborhood of pixel values, and detect some pattern in that data.
- Suppose we have a one dimensional image X of size d , and a filter F of size two.
- The filter F is a vector of two numbers which we will move along the image, taking dot product between the filter values and the image value at each step. The results will be aggregated to produce a new image.
- Each pixel i of the output image will be

$$Y_i = F \cdot [X_{i-1}, X_i]$$

Filters

$$Y_i = F \cdot [X_{i-1}, X_i]$$



Applying the filter to an image is called **convolution**.

Filter Example

A 1D image:



A filter:



After
convolution*:



Filter Example

A 1D image:



A filter:



$$0 * -1 + 0 * 1 + 1 * -1 = -1$$

After convolution*:



Filter Example

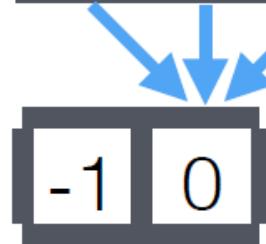
A 1D image:



A filter:

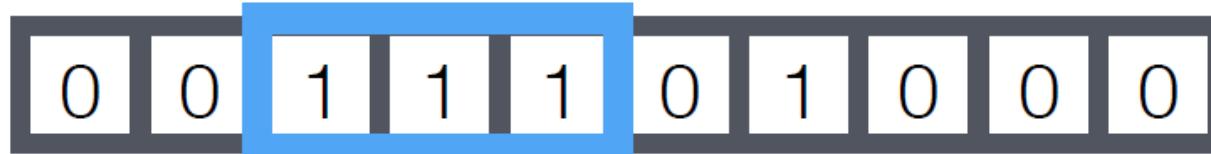


After
convolution*:



Filter Example

A 1D image:



A filter:

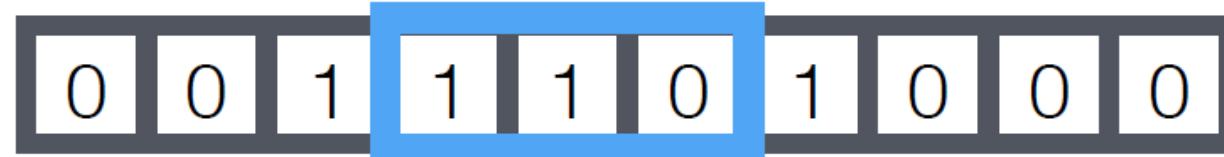


After
convolution*:



Filter Example

A 1D image:



A filter:



After
convolution*:



Filter Example

A 1D image:



A filter:



After
convolution*:



Filter Example

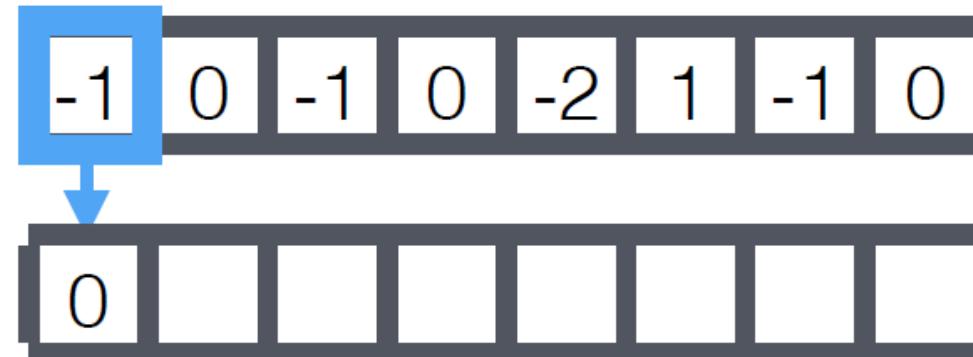
A 1D image:



A filter:



After convolution*:



After ReLU:

Filter Example

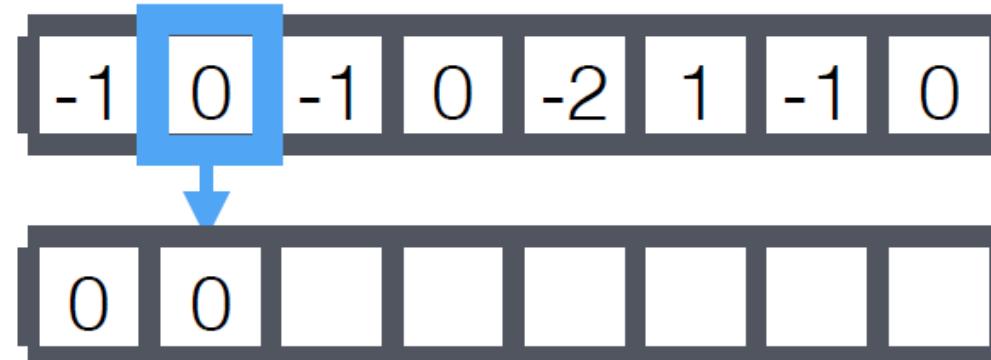
A 1D image:



A filter:



After convolution*:



After ReLU:

Filter Example

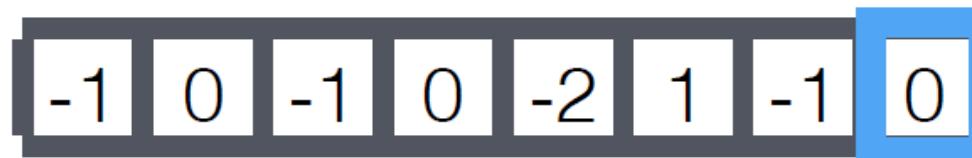
A 1D image:



A filter:



After convolution*:



After ReLU:



This filter is a detector for isolated pixels in the binary image.

A 1D image:



A filter:



After convolution*:



After ReLU:



The resulting image after applying the filter is smaller than the original image. How to have an output image of the same size?

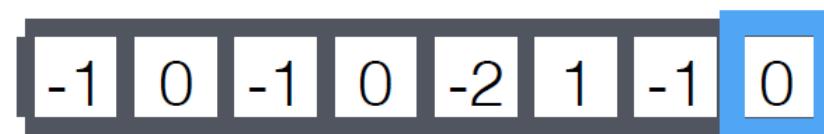
A 1D image:



A filter:



After convolution*:



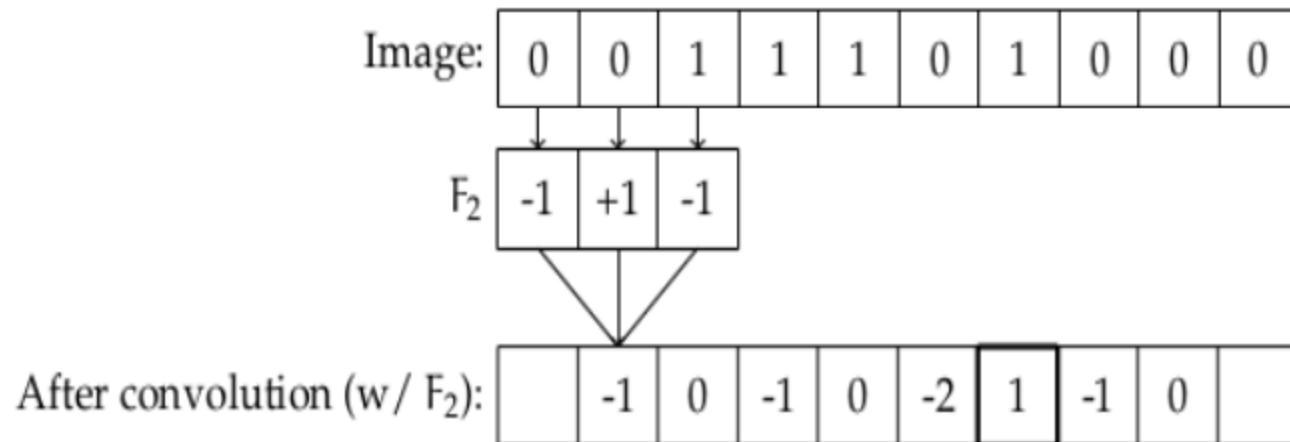
After ReLU:



Padding

In order to have an output image with the same dimensions, we will use **padding**.

When we apply filter, if the pixel we need to multiply is beyond the bounds, we will add 0 values there.



Filter Example - Padding

A 1D image:



A filter:



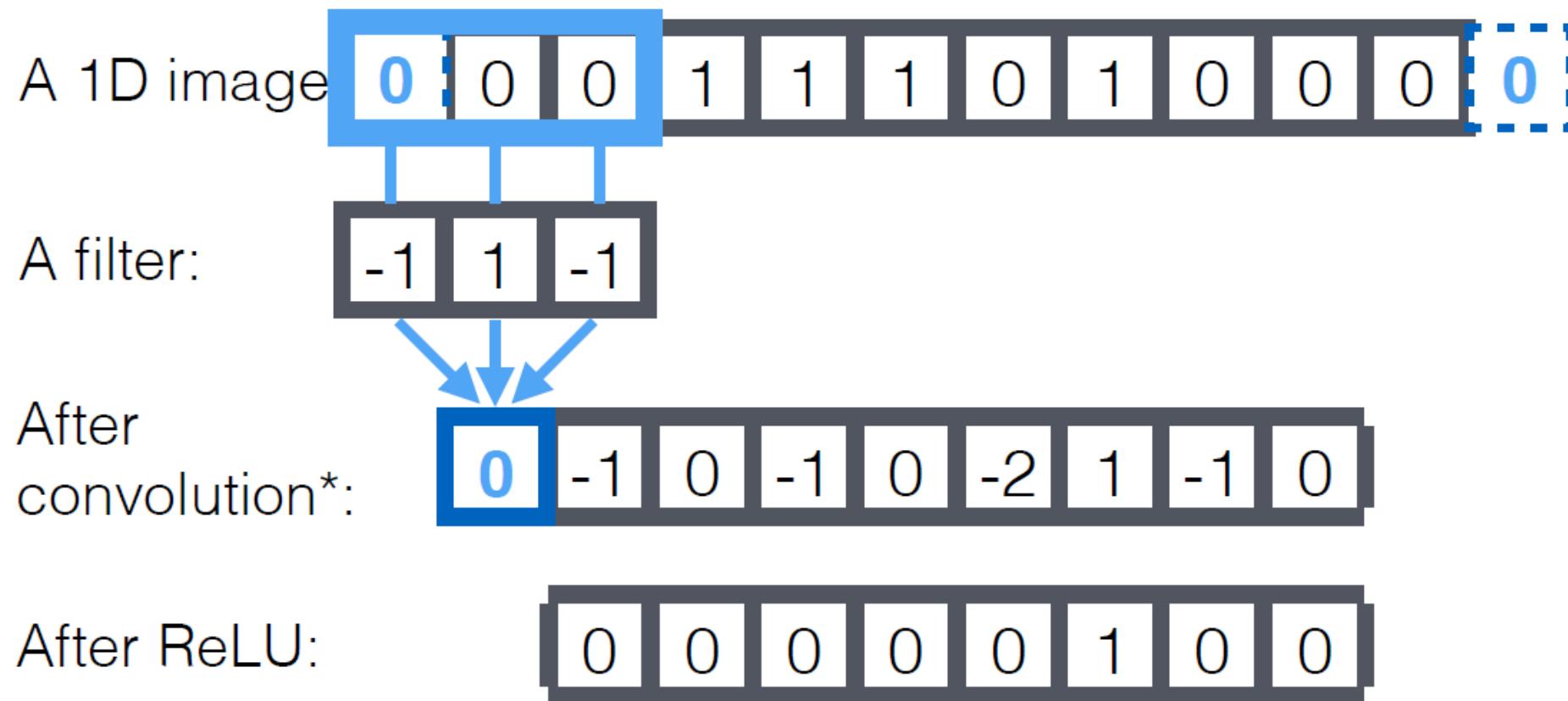
After convolution*:



After ReLU:



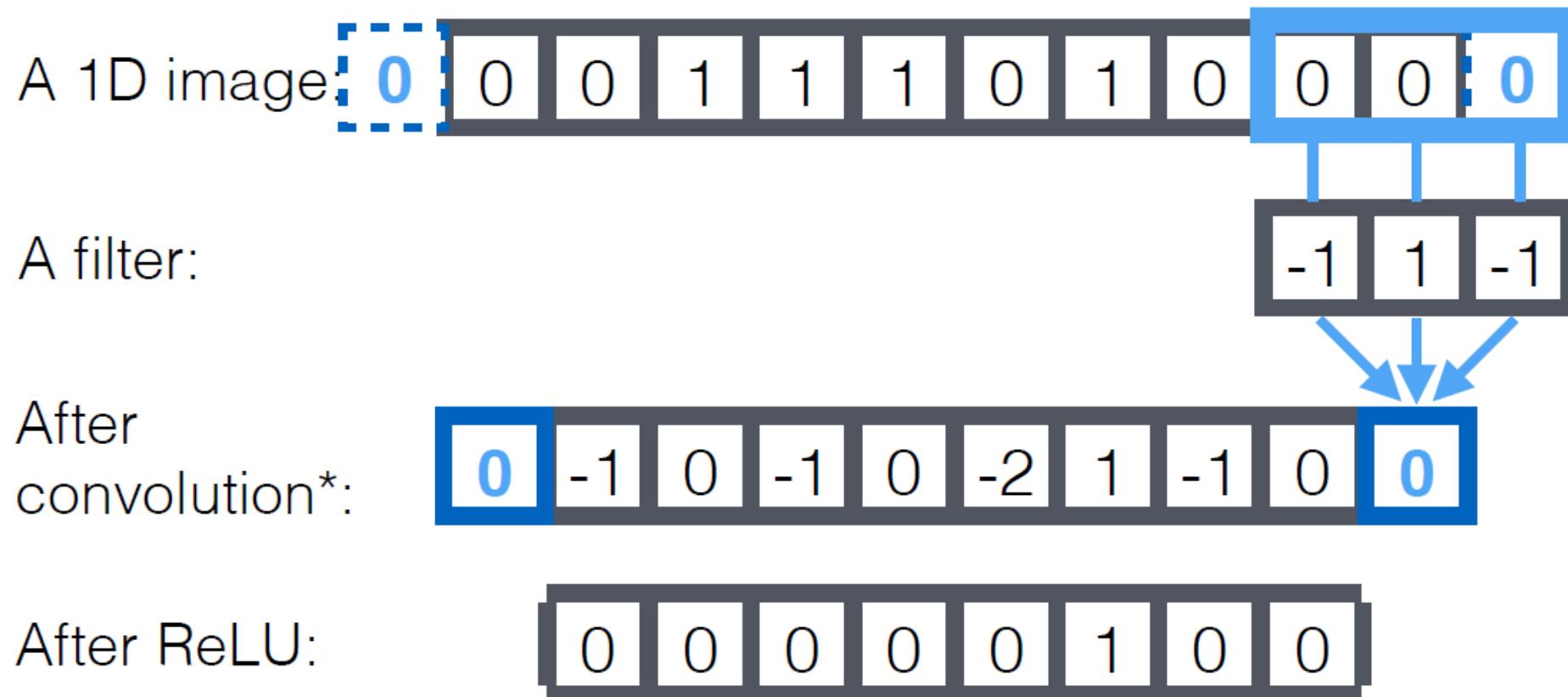
Filter Example - Padding



After ReLU:



Filter Example - Padding



Filter Example - Padding

A 1D image:



A filter:



After convolution*:



After ReLU:



2D Filter Example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

$$\begin{aligned}-1 + 0 + -1 \\ + -1 + 0 + -1 \\ + -1 + -1 + -1 \\ = -7\end{aligned}$$

After convolution:

-7

2D Filter Example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

7	-2
---	----

After convolution:

2D Filter Example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

-7	2	-4
----	---	----

After convolution:

2D Filter Example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

-7	-2	-4
-5		

2D Filter Example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

-7	-2	-4
5	-2	

2D Filter Example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

-7	-2	-4
-5	-2	-5
-7	2	-5

2D Filter Example - Padding

A 2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

A filter:

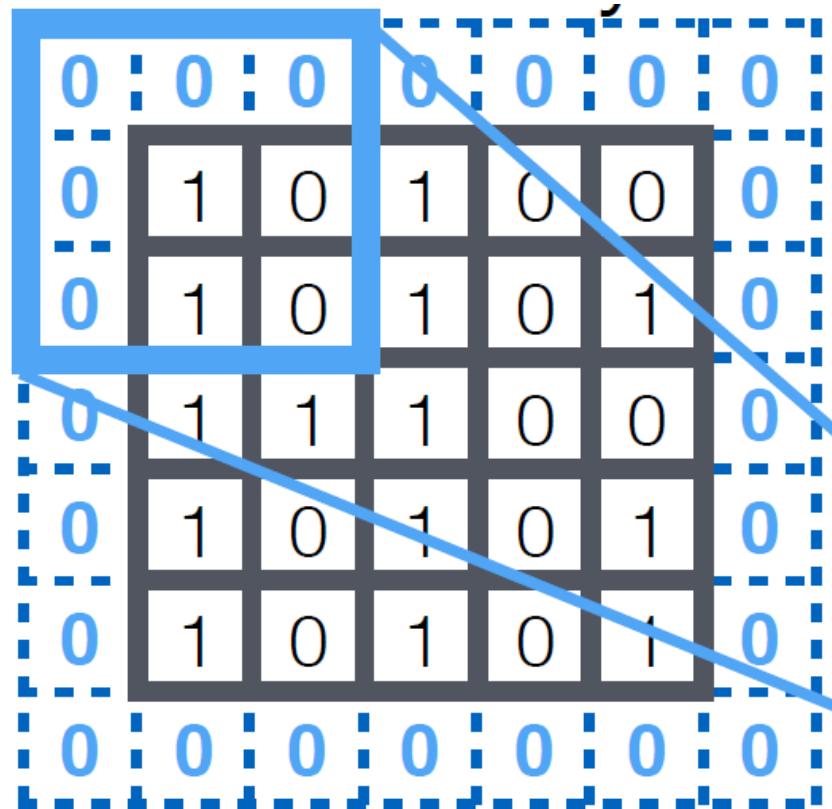
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

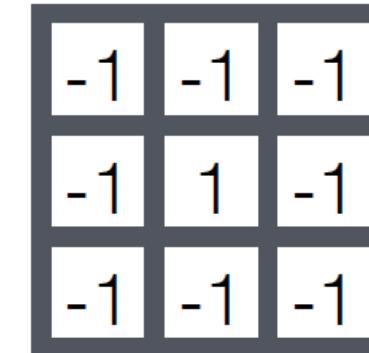
-7	-2	-4
-5	-2	-5
-7	-2	-5

2D Filter Example - Padding

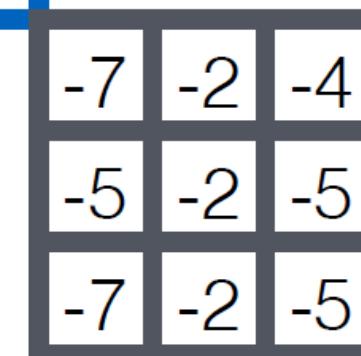
A 2D
image:



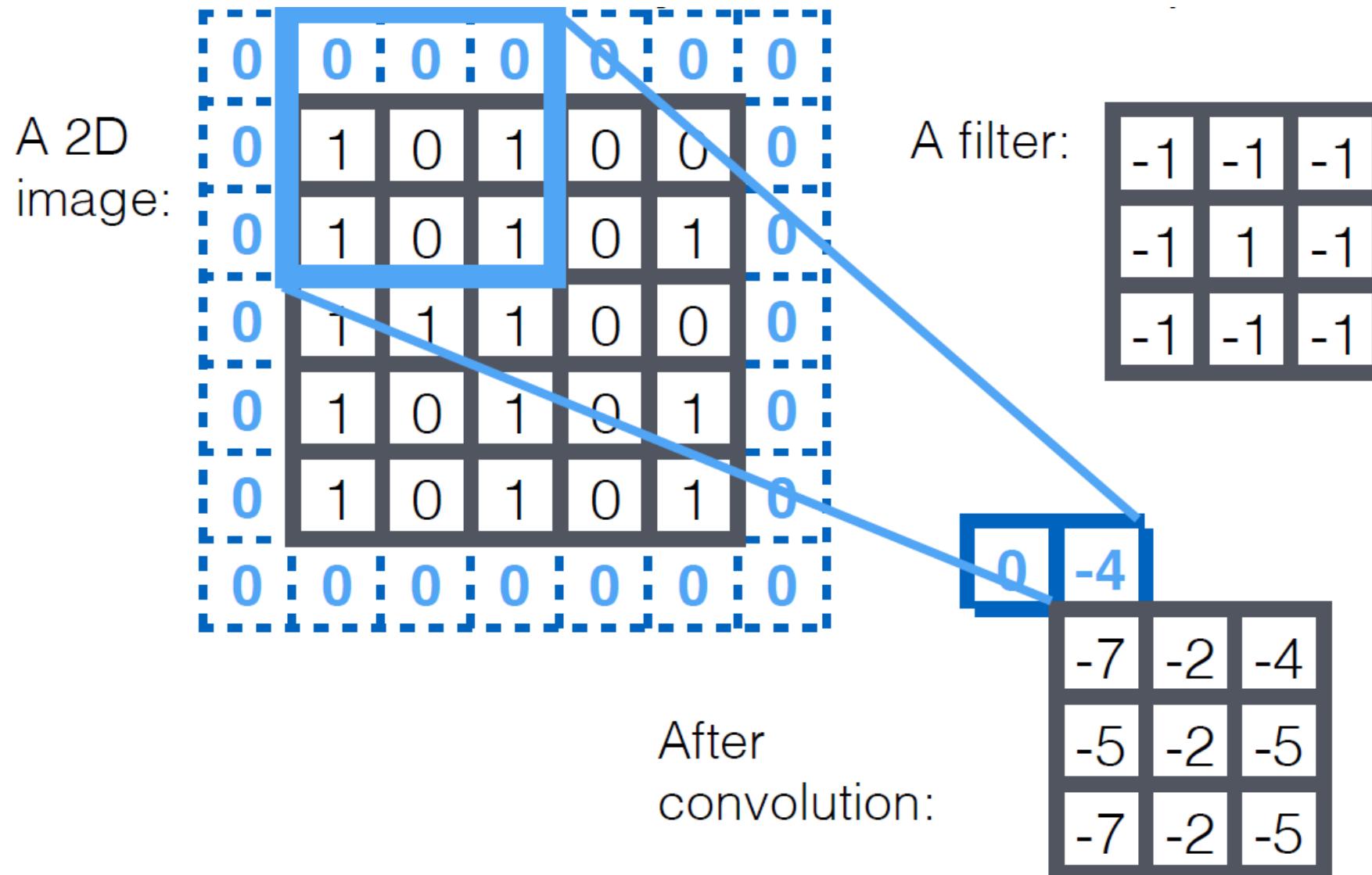
A filter:



After
convolution:



2D Filter Example - Padding



2D Filter Example - Padding

A 2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

2D Filter Example - ReLU

A 2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution & ReLU:

0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

2D Filter Example - ReLU

A 2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution & ReLU:

0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

2D Filter Example - ReLU

A 2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

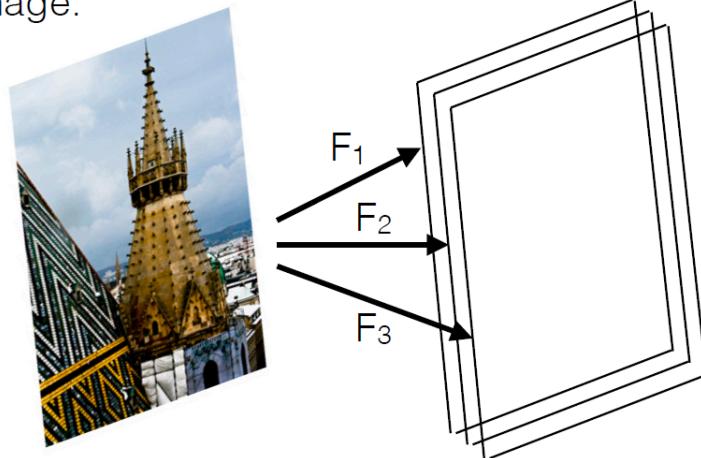
After
convolution
& ReLU:

0	0	0	0	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Filter Banks

- A *filter bank* is a set of filters arranged as follows

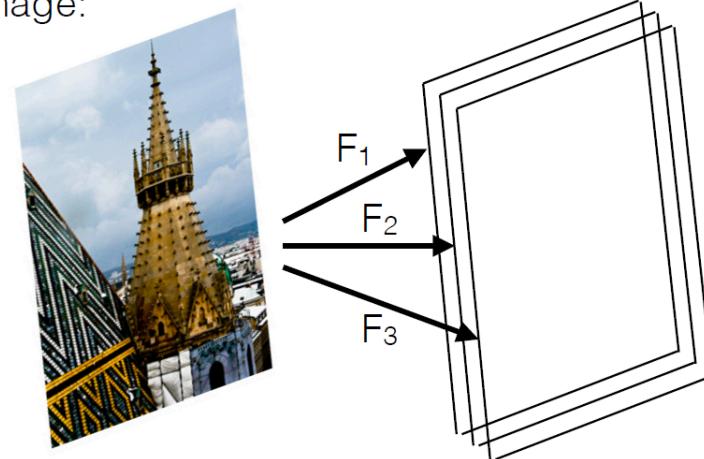
An
image:



Filter Banks

- A *filter bank* is a set of filters arranged as follows
- All filters in the first group are applied to the original image.
 - When there are k filters, the result is k new images.
 - Each of these k new images is called a channel.

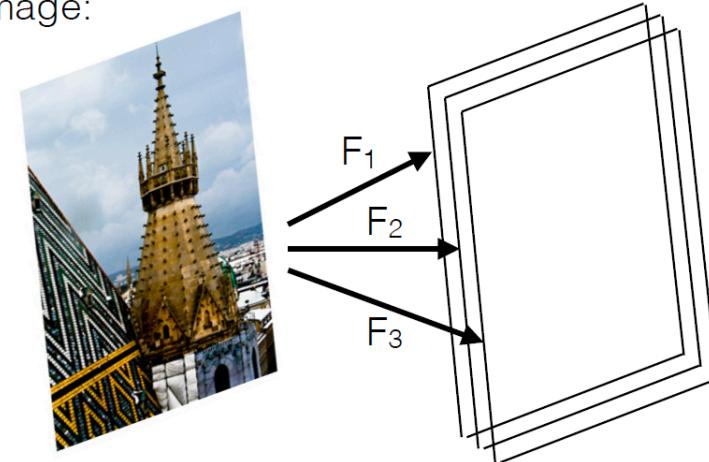
An image:



Filter Banks

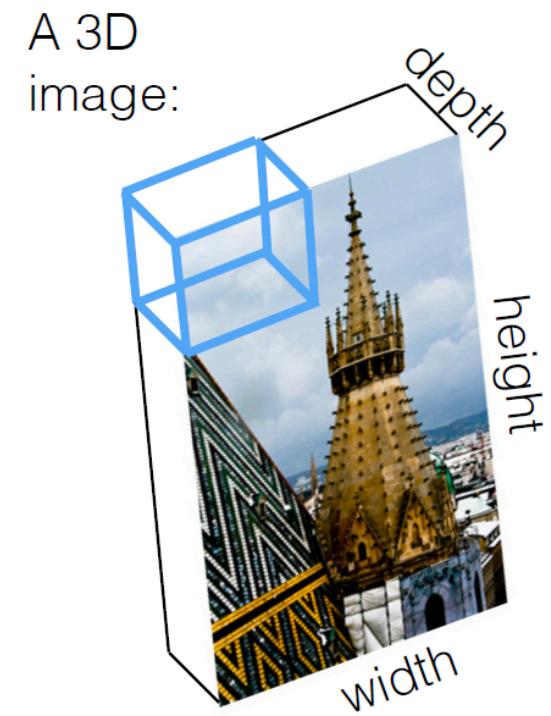
- A *filter bank* is a set of filters arranged as follows
- All filters in the first group are applied to the original image.
 - When there are k filters, the result is k new images.
 - Each of these k new images is called a channel.
- Stacking these new images will result in a 3-dimensional *cube* of data
- The next set of filters in the filter bank will be *three-dimensional*.

An image:

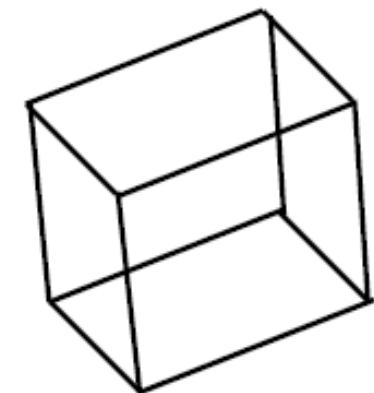


Tensors

- These three-dimensional data structures are called *tensors*.
- Tensors are higher dimensional arrays.
 - PyTorch makes it easy to work with tensors



A filter:



Convolutional Layers in PyTorch

- In PyTorch `nn` module, a convolutional layer of 32 filters, each 3×3 , with stride of 1, and zero padding can be defined as follows.

```
conv_layer = torch.nn.Conv2d(1, 32, (3,3), 1)
```

If the filters are square (as in 3×3) you can concisely define as follows

```
conv_layer = torch.nn.Conv2d(1, 32, 3, 1)
```

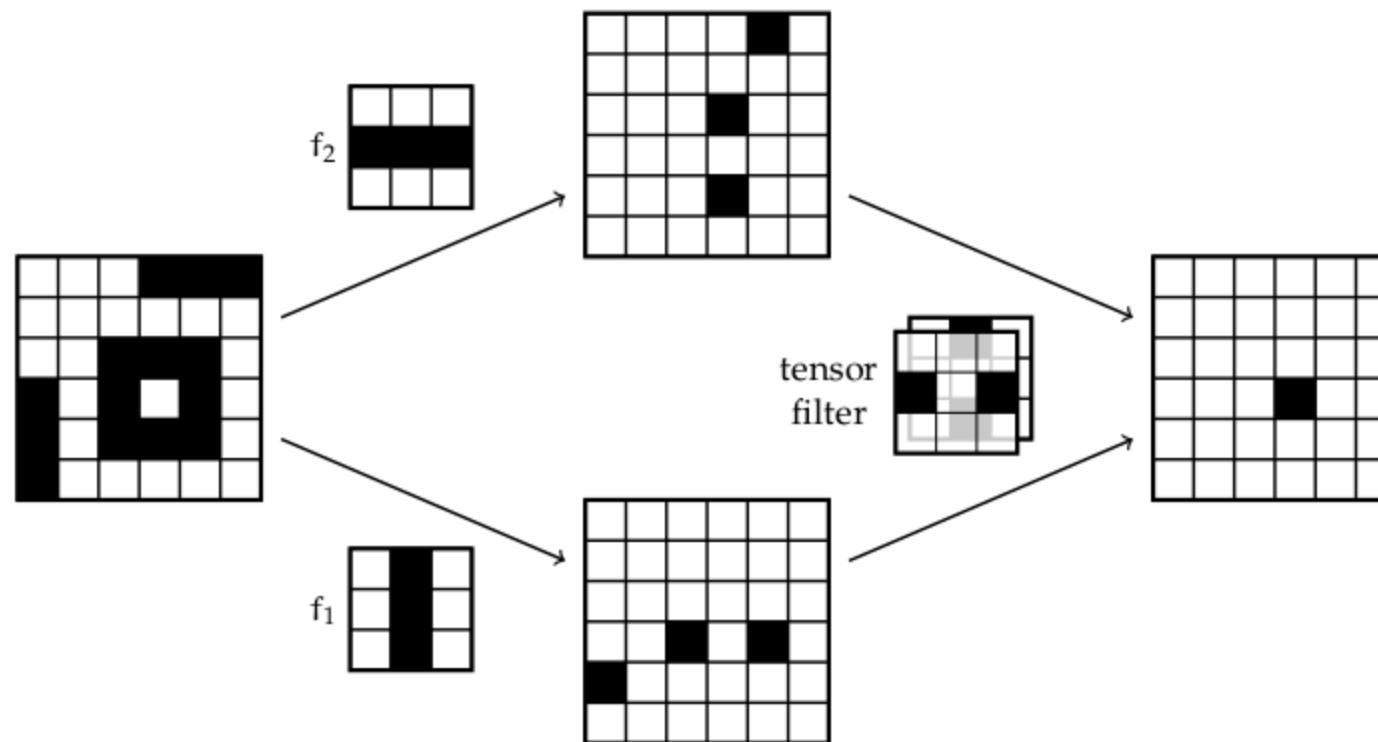
You need to stack these layers with ReLU activation and other layers in the `forward` method of your model (see tutorial).

Convolutional Layers in PyTorch

If you have multiple convolutional layers (filter banks), the dimensions need to match

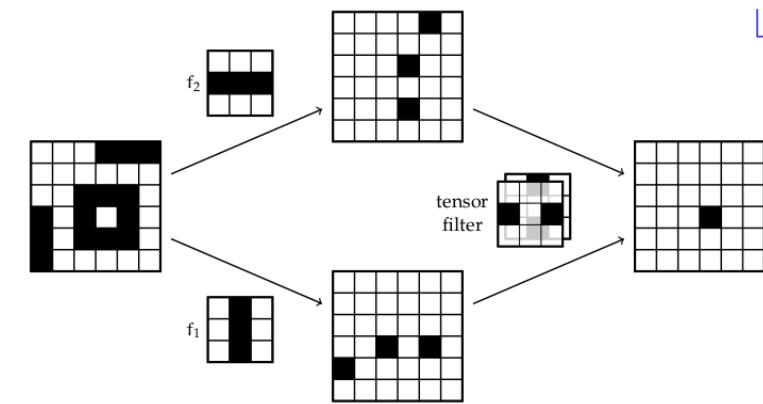
```
conv_layer1 = torch.nn.Conv2d(1, 32, (3,3), 1)
conv_layer2 = torch.nn.Conv2d(32, 32, (3,3), 1)
```

Example



Example

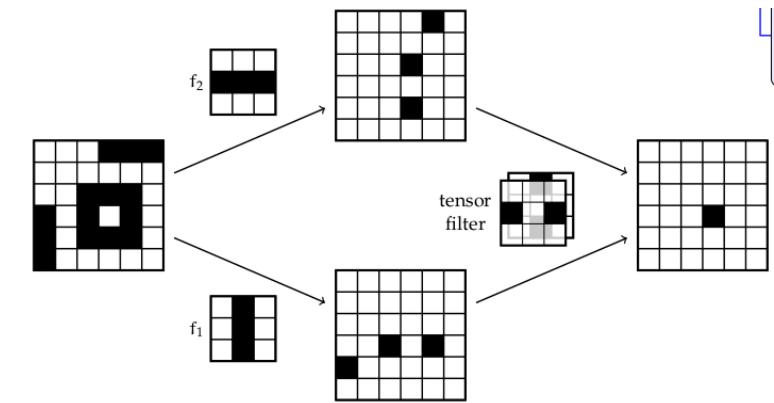
- We have original $n \times n$ image and two 3×3 filters f_1 and f_2 . These look for three pixels in a row horizontally and vertically.
- The result is $n \times n \times 2$ tensor.
- Then we apply a $3 \times 3 \times 2$ filter that looks for a combination of two horizontal and two vertical parts.
- The final result is $n \times n$ structure.



Convolutional Neural Networks

Convolutional Neural Networks

- We will design networks with such structure.
- Each bank of the filter bank will correspond to a neural network layer.
- The numbers in the individual filters will be the weights of the network.
- We will train the network using gradient descent.



Convolutional Neural Networks

- Note that, the same filters will be used many times in each layer.
- This enables that we can transform a large image with a relatively few parameters.

Filter layer l

- A filter layer l will be defined as
 - number of filters m^l
 - size of one filter $k^l \times k^l \times m^{l-1}$ plus 1 bias value
 - input tensor size $n^{l-1} \times n^{l-1} \times m^{l-1}$

Filter layer l

- stride s^l
 - stride is the spacing which we apply the filter to the image. Up to now we have used a stride of 1. If we were to skip and apply the filter only at odd-numbered indices, it would have a stride of two (this would lead to a resulting image of half the size)

Filter layer l

- padding p^l how many extra pixels we add around the edges of the input. For an input size $n^{l-1} \times n^{l-1} \times m^{l-1}$ the new effective input with padding is $(n^{l-1} + 2p^l) \times (n^{l-1} + 2p^l) \times m^{l-1}$

Filter layer

- A filter layer l will be defined as
 - number of filters m^l
 - size of one filter $k^l \times k^l \times m^{l-1}$ plus 1 bias value
 - input tensor size $n^{l-1} \times n^{l-1} \times m^{l-1}$
 - stride s^l
 - stride is the spacing which we apply the filter to the image. Up to now we have used a stride of 1. If we were to skip and apply the filter only at odd-numbered indices, it would have a stride of two (this would lead to a resulting image of half the size)
 - padding p^l how many extra pixels we add around the edges of the input. For an input size $n^{l-1} \times n^{l-1} \times m^{l-1}$ the new effective input with padding is $(n^{l-1} + 2p^l) \times (n^{l-1} + 2p^l) \times m^{l-1}$

Filter layer

This produces

$$n^l \times n^l \times m^l = \lceil (n^{l-1} + 2p^l - (k^l - 1)/s^l) \rceil$$

Max pooling

Max Pooling

- Typically filter banks are structured like a pyramid. Image size gets smaller in successive layers.
- We look for local patterns, and then look for patterns of those patterns. So we are looking for patterns in larger pieces of the image as we apply successive filters.

Max Pooling

- Typically filter banks are structured like a pyramid. Image size gets smaller in successive layers.
- We look for local patterns, and then look for patterns of those patterns. So we are looking for patterns in larger pieces of the image as we apply successive filters.
- Having a stride makes image smaller but does not necessarily aggregate information over the spatial range
- *Max pooling* layers can be used for this aggregation

Max Pooling

- *Max pooling* is like a filter with no weights. We can consider it as a pure functional layer (like ReLU layer in fully connected NNs).
- It has a filter size, it simply returns the maximum value in its field.
- It highlights the location of features.

Max Pooling

- *Max pooling* is like a filter with no weights. We can consider it as a pure functional layer (like ReLU layer in fully connected NNs).
- It has a filter size, it simply returns the maximum value in its field.
- *Max pooling* is applied with the following traits
 - $\text{stride} > 1$ (so that the resulting image is smaller)
 - $k \geq \text{stride}$ (so that the whole image is covered) (k is the size of max pooling)

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:



Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:



Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

0	0
---	---

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

0	0	1
---	---	---

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

0	0	1	1
---	---	---	---

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

0	0	1	1
1	1	1	1
1	1	0	0
1	1	0	0

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

0	0	1	1
1	1	1	1
1	1	0	0
1	1	0	0

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 1

After max pooling:

0	0	1	1
1	1	1	1
1	1	0	0
1	1	0	0

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:



Max Pooling - Example 1

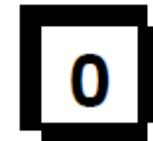
Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:



Max Pooling - Example 1

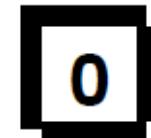
Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:



Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling.

0	1
---	---

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
---	---

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
---	---

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
---	---

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	

Max Pooling - Example 1

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	0

Max Pooling - Example

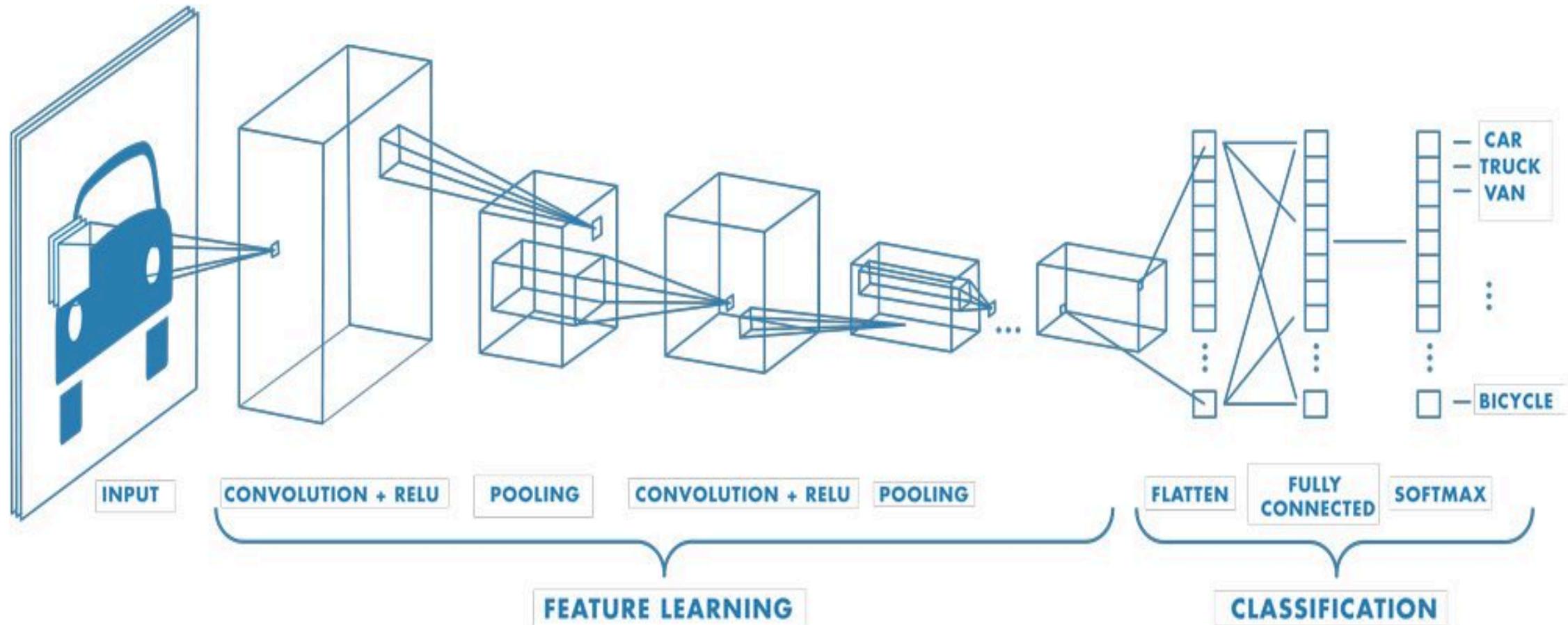
- Max pooling with stride = 3 and $k = 3$ would map a $6 \times 6 \times 1$ image to a $2 \times 2 \times 1$ image.
- Max pooling with stride = 2 and $k = 2$ would map a $64 \times 64 \times 3$ image to a $32 \times 32 \times 3$ image.

Max Pooling with PyTorch

Max pooling layer of $k = 2$ can be applied as follows in PyTorch

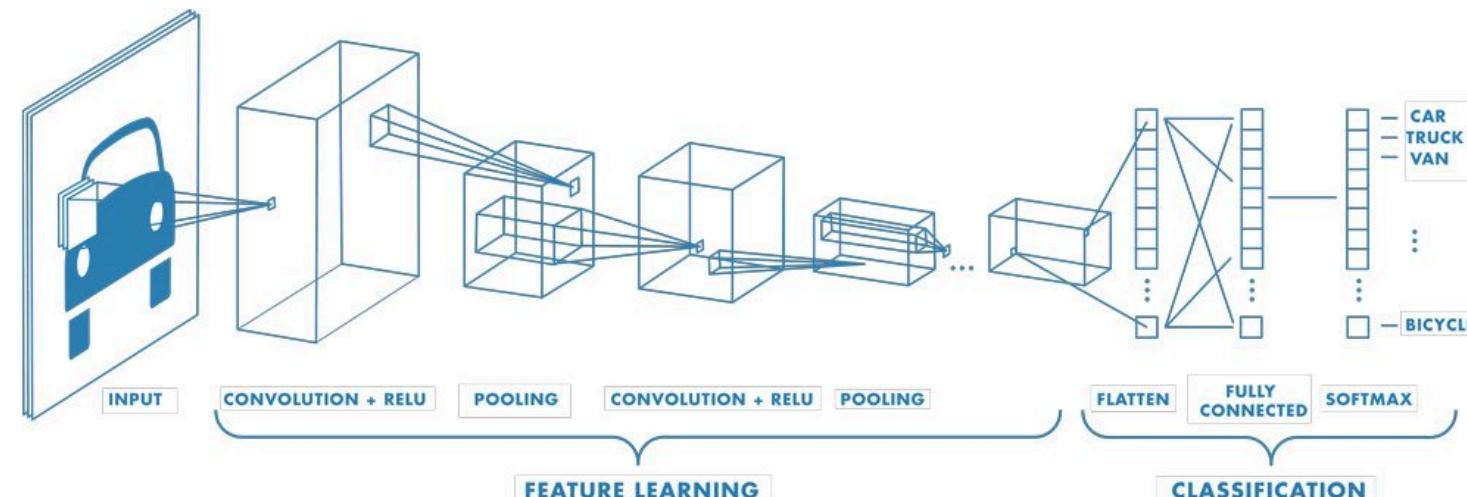
```
pool = torch.nn.MaxPool2d(3, stride=3)
```

Typical Architecture



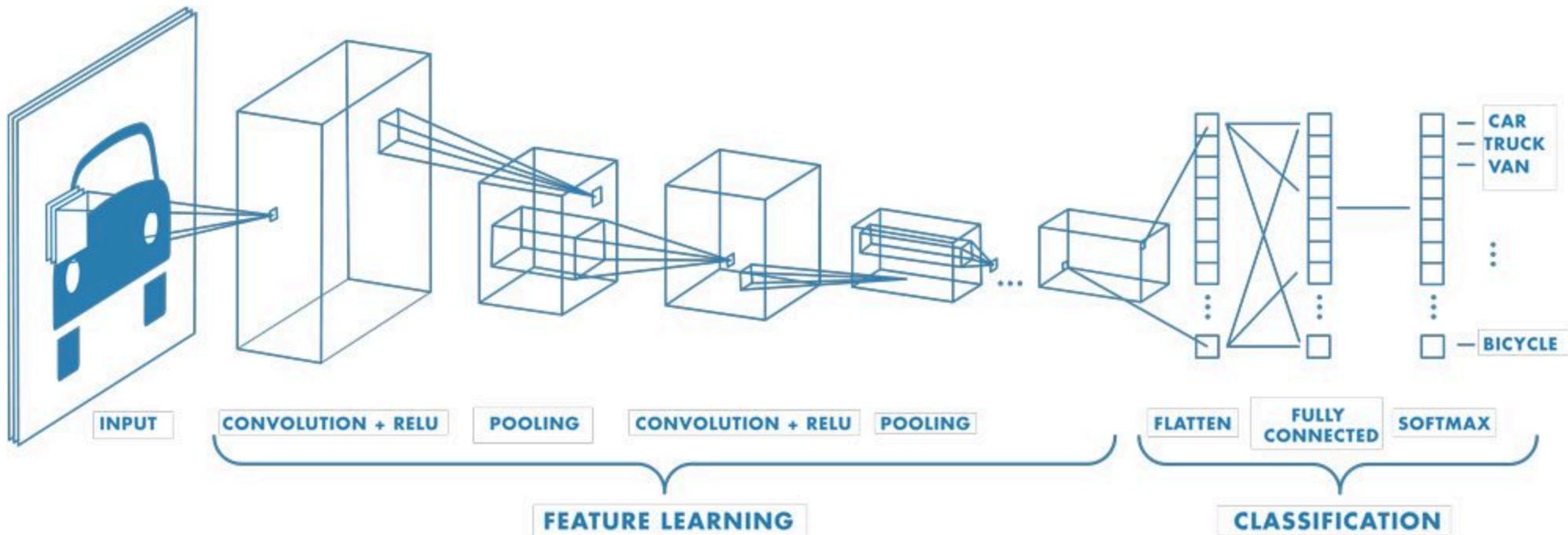
Typical Architecture

- After each filter layer, there is generally a ReLU layer
- After multiple filter/ReLU layers there is a max pooling layer
- There may be some more filter/ReLU layers followed by max pooling layers
- A last fully connected layer
- And a softmax activation function making the classification.
- Design is more an *art* right now.



Typical Architecture

- This is just a large neural network, takes an input (image) and computes and output (classification)
- Mapping is a differentiable function of the weights, we can use *gradient descent* and we can calculate the gradients using *back-propagation*.



Example CNN in PyTorch

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(32 * 32 * 3 * 3, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

Appendix: Back-propagation with CNNs

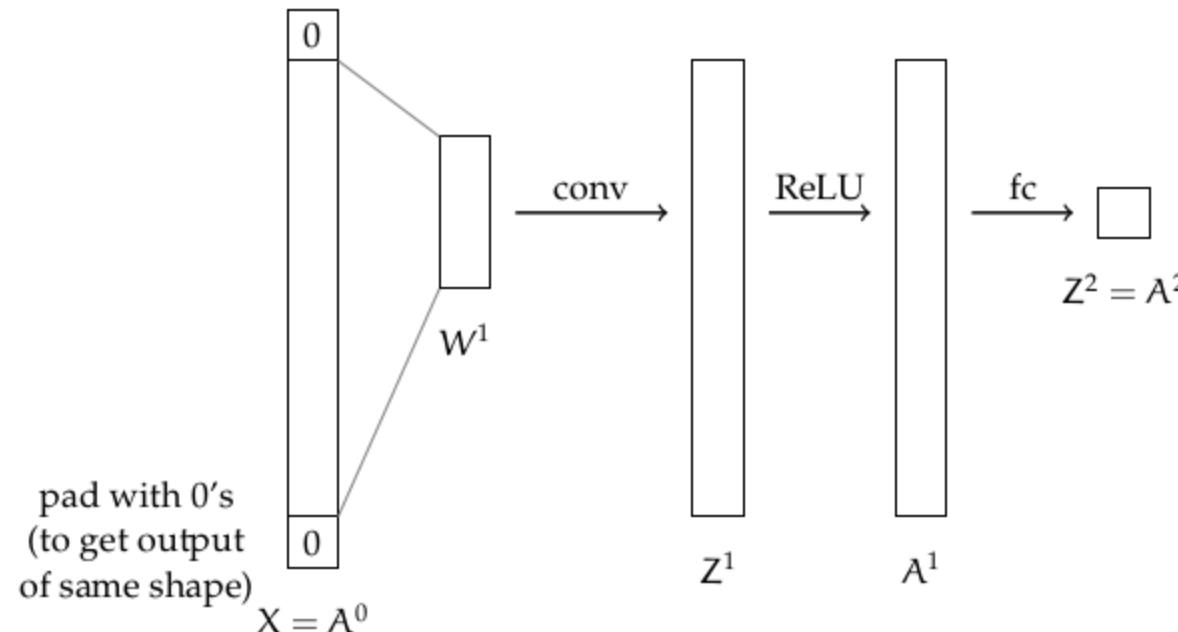
Simple Example of Back-propagation

Suppose we have one dimensional of single-channel image $n \times 1 \times 1$

A single $k \times 1 \times 1$ filter. (lets assume k is odd)

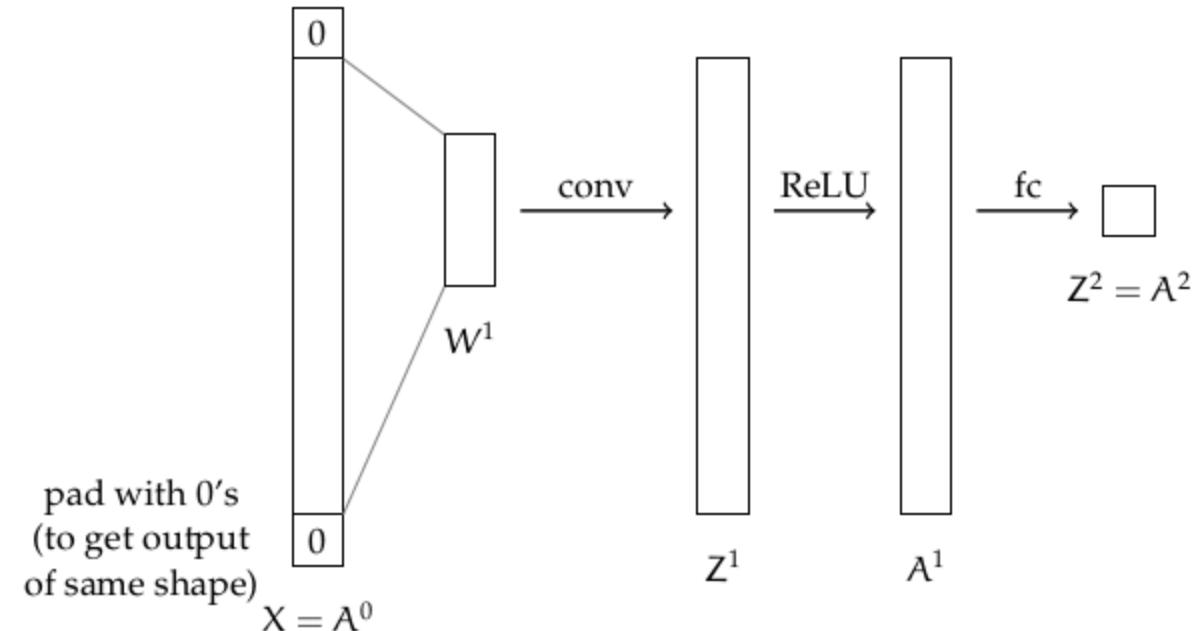
We pass it through a ReLU layer and a fully connected layer with no activation function on the output (regression)

Input image is $X = A^0$



Simple Example - Forward Pass

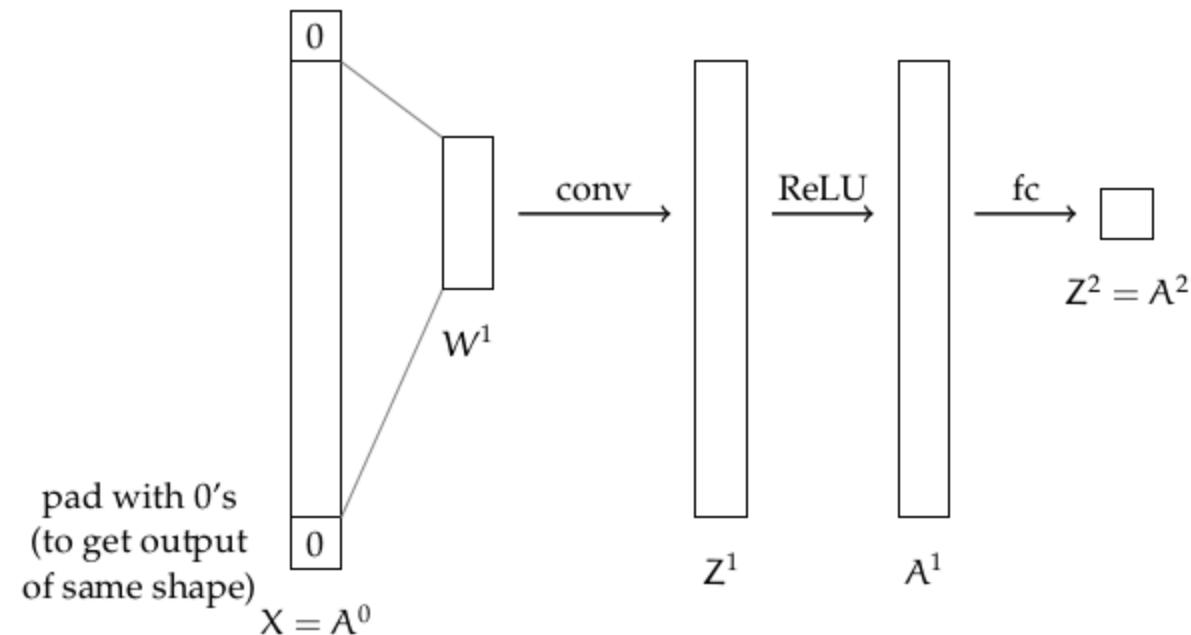
$$Z_i^1 = W^{1^T} \cdot A_{[i-\lfloor k/2 \rfloor : i+\lfloor k/2 \rfloor]}^0$$



Simple Example - Forward Pass

$$Z_i^1 = W^{1^T} \cdot A_{[i-\lfloor k/2 \rfloor : i+\lfloor k/2 \rfloor]}^0$$

$$A^1 = \text{ReLU}(Z^1)$$

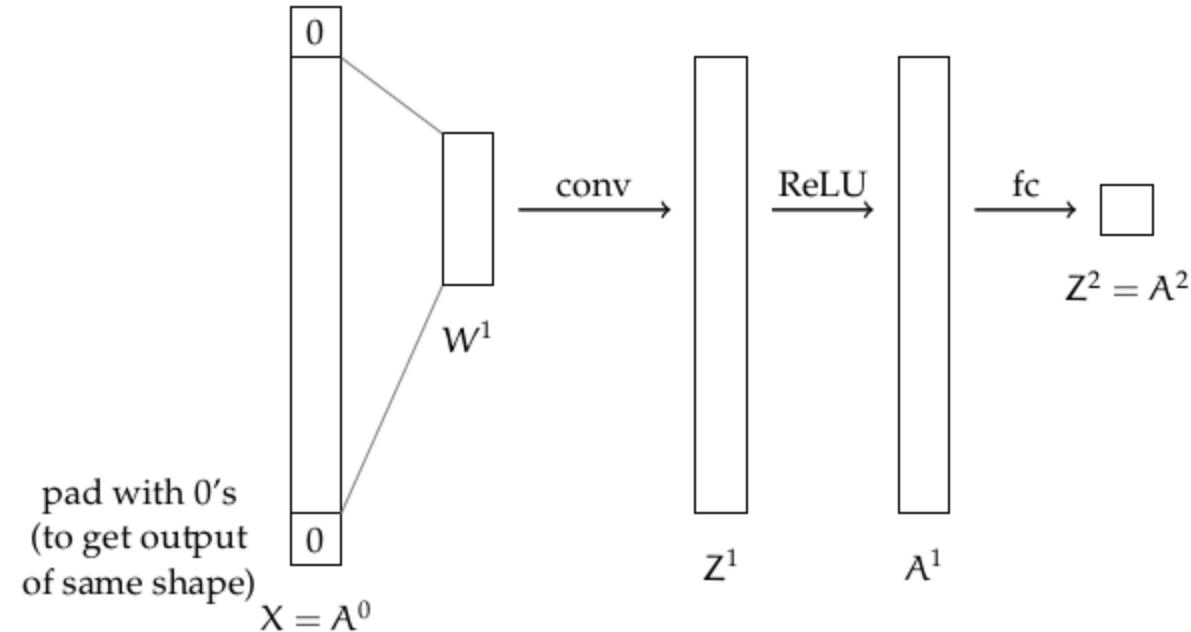


Simple Example - Forward Pass

$$Z_i^1 = W^{1^T} \cdot A_{[i-\lfloor k/2 \rfloor : i+\lfloor k/2 \rfloor]}^0$$

$$A^1 = \text{ReLU}(Z^1)$$

$$A^2 = W^{2^T} \cdot A^1$$



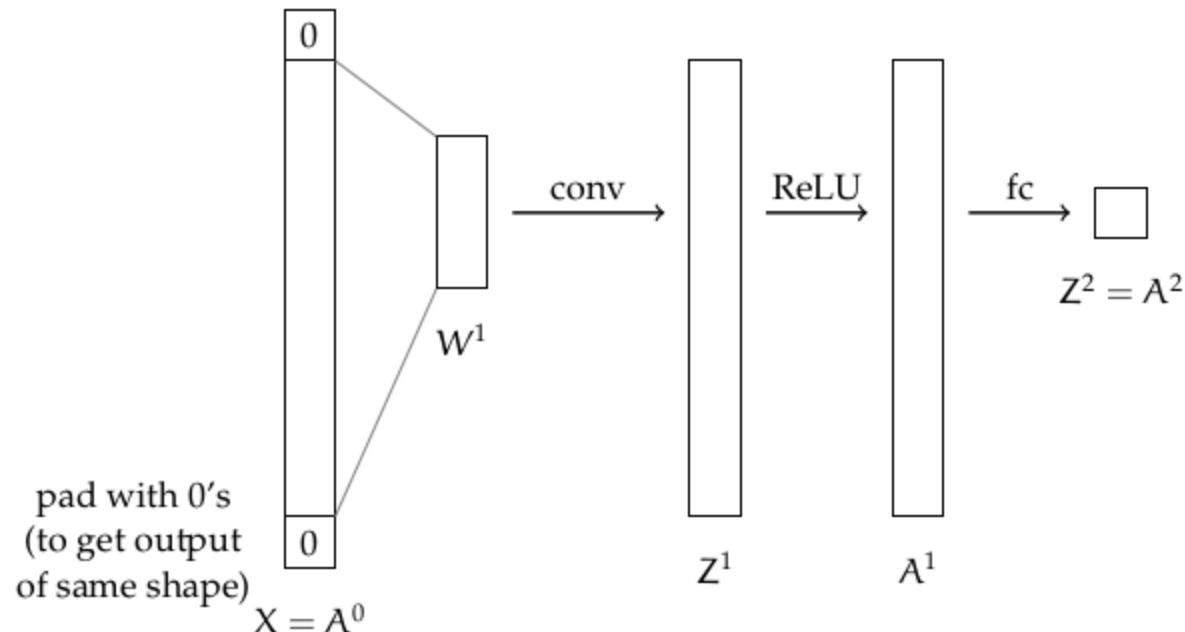
Simple Example - Forward Pass

$$Z_i^1 = W^{1^T} \cdot A_{[i-\lfloor k/2 \rfloor : i+\lfloor k/2 \rfloor]}^0$$

$$A^1 = \text{ReLU}(Z^1)$$

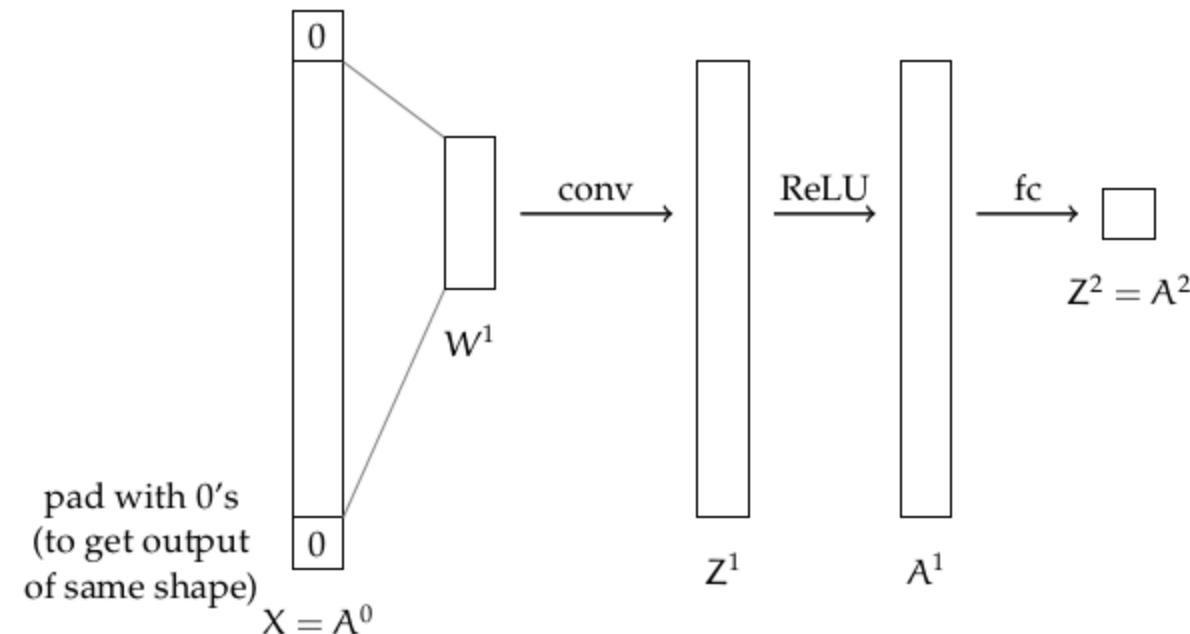
$$A^2 = W^{2^T} \cdot A^1$$

$$\text{Loss}(A^2, y) = (A^2 - y)^2$$



Simple Example - Backward Pass

$$\frac{\partial \text{loss}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1} \cdot \frac{\partial A^1}{\partial Z^1} \cdot \frac{\partial \text{loss}}{\partial A^1}$$



Simple Example - Backward Pass

$$\frac{\partial \text{loss}}{\partial W^1} = \underbrace{\frac{\partial Z^1}{\partial W^1}}_{\cdot} \cdot \frac{\partial A^1}{\partial Z^1} \cdot \frac{\partial \text{loss}}{\partial A^1}$$

- $\frac{\partial Z^1}{\partial W^1}$ is the $k \times n$ matrix such that $\frac{\partial Z_i^1}{\partial W_i^1} = X_{i-\lfloor k/2 \rfloor + j - 1}$.
 - $i = 10$ corresponds to the column 10 in this matrix, which illustrates the dependence of pixel 10 of the output image on the weights. if $k = 5$ elements on column 10 will be $X_8, X_9, X_{10}, X_{11}, X_{12}$
 - Each input is multiplied by several different weights.

Backward Pass

$$\frac{\partial \text{loss}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1} \cdot \underbrace{\frac{\partial A^1}{\partial Z^1}}_{\cdot} \cdot \frac{\partial \text{loss}}{\partial A^1}$$

- $\frac{\partial A^1}{\partial Z^1}$ is the $n \times n$ diagonal matrix such that

$$\partial A_i^1 / \partial Z_i^1 = \begin{cases} 1 & \text{if } Z_i^l > 0 \\ 0 & \text{otherwise} \end{cases}$$

Backward Pass

$$\frac{\partial \text{loss}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1} \cdot \frac{\partial A^1}{\partial Z^1} \cdot \underbrace{\frac{\partial \text{loss}}{\partial A^1}}$$

- $\partial \text{loss} / \partial A^1 = \partial \text{loss} / \partial A^2 \cdot \partial A^2 / \partial A^1 = 2(A^2 - y)W^2$ is a $n \times 1$ vector.

Backward Pass

$$\frac{\partial \text{loss}}{\partial W^1} = \underbrace{\frac{\partial Z^1}{\partial W^1}}_{k \times n} \cdot \underbrace{\frac{\partial A^1}{\partial Z^1}}_{n \times n} \cdot \underbrace{\frac{\partial \text{loss}}{\partial A^1}}_{n \times 1}$$

The output is a $k \times 1$ vector of the gradients.

COGS514 - Cognition and Machine Learning

Recurrent Neural Networks - 1

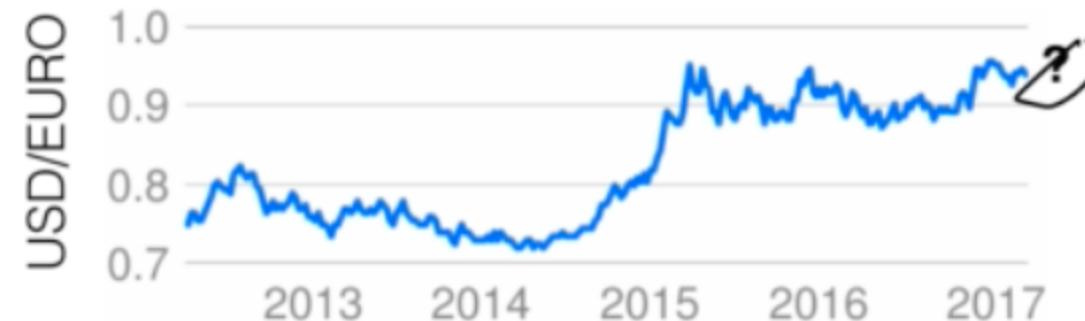
Sequential Models

- Until now, our hypotheses have been *pure* functions which map a *single input x* to y .
 - Outputs y depended only on x

Sequential Models

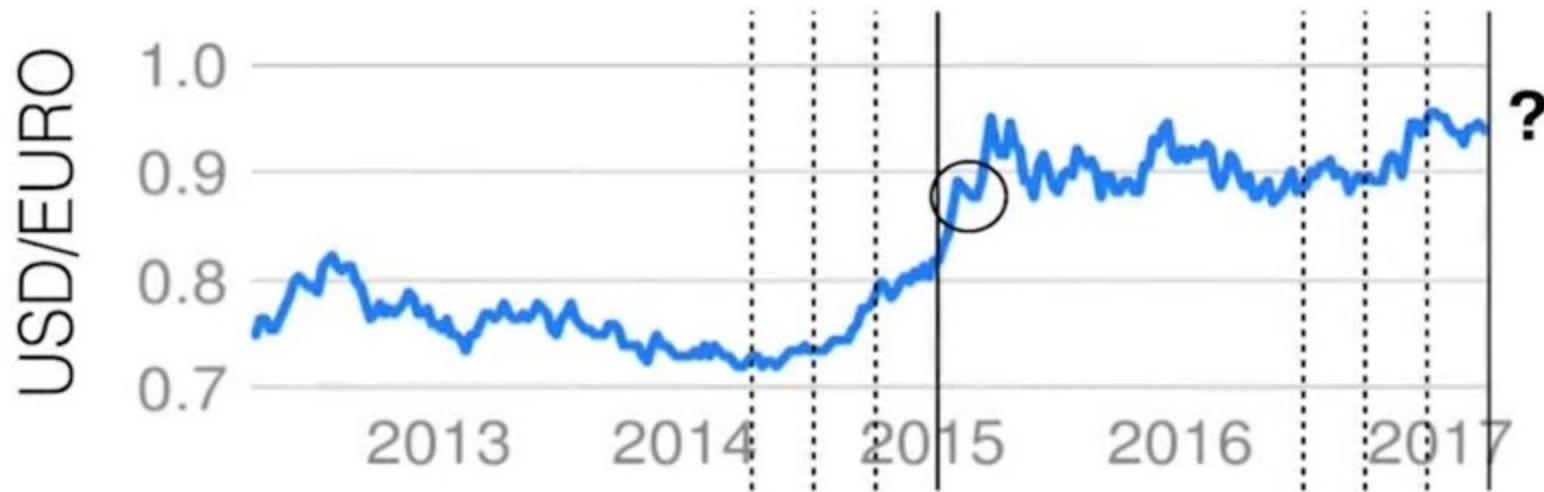
- Until now, our hypotheses have been *pure* functions which map a *single input x* to y .
 - Outputs y depended only on x
- We will now focus on learning from a *sequence of inputs*.
 - For example, *recurrent neural networks, reinforcement learning*
- Firstly, we will examine the sequential models underlying these systems.

Example - Sequential Data



How to formulate as a supervised learning problem?

Example - Sequential Data



- Data can be broken down into feature vectors and target values (sliding window)

$$\begin{bmatrix} 0.82 \\ 0.79 \\ 0.74 \\ 0.71 \end{bmatrix} \quad 0.89$$
$$\phi(t) \qquad y^{(t)}$$

Example - Language modeling

This course has been a tremendous | ...

tremendous

$$\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

?

a

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}$$

$$\phi(t) \quad y^{(t)}$$

Example - Language modeling

This course has been a | tremendous ...

$$a \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}$$

tremendous

$$\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

been

$$\phi(t) \quad y^{(t)}$$

What is missing?

- We can turn sequential prediction problems into a format that can be learned with feed-forward neural networks
- We have to engineer how "history" is mapped to a vector. This vector is fed into a neural network.
 - How many steps back should be considered?
 - How to retain important items far back?

What is missing?

- We can turn sequential prediction problems into a format that can be learned with feed-forward neural networks
- We have to engineer how "history" is mapped to a vector. This vector is fed into a neural network.
 - How many steps back should be considered?
 - How to retain important items far back?
- Instead, we would like to learn how to encode the "history" into a vector.

Learning to encode/decode

- Language modeling

This course has been a



success (?)

- Sentiment classification

I have seen better lectures



-1

- Machine translation

I have seen better lectures



Olen nähnyt parempia
luentoja

encoding

decoding

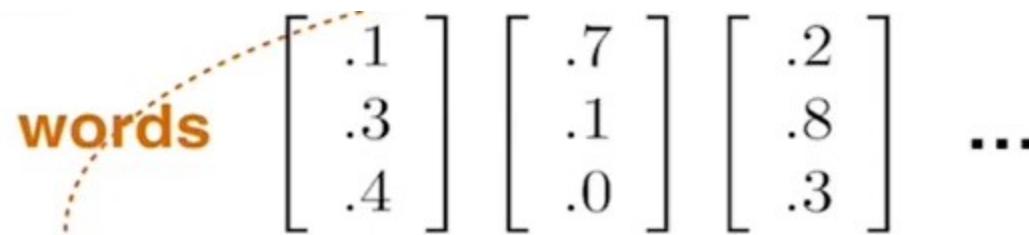
Encoding

- e.g. mapping a sequence to a vector

Decoding

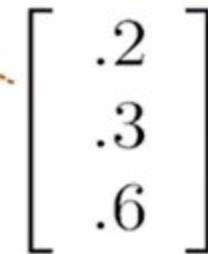
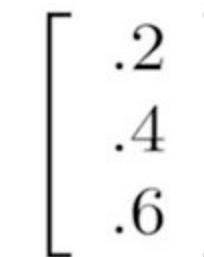
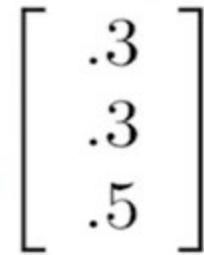
- e.g. mapping a vector to a sequence

Encoding everything



“Efforts and courage are not enough without purpose and direction” — JFK

sentences



State Machines

State Machines

- A *state machine* is a description of a process in terms of its potential sequences of states.
 - State machine is a *transducer*, y depends not only on x but the whole history xs

State Machines

- A *state machine* is a description of a process in terms of its potential sequences of states.
 - State machine is a *transducer*, y depends not only on x but the whole history xs
- A *state* of a system is all you would need to know about the system to predict its future trajectories as well as possible.
 - e.g. locations of your pieces on a game board, velocity or a object.

State Machines

A state machine is defined as $\langle \mathcal{S}, \mathcal{X}, \mathcal{Y}, s_0, f, g \rangle$ where

State Machines

A state machine is defined as $\langle \mathcal{S}, \mathcal{X}, \mathcal{Y}, s_0, f, g \rangle$ where

- \mathcal{S} is the set of possible states

State Machines

A state machine is defined as $\langle \mathcal{S}, \mathcal{X}, \mathcal{Y}, s_0, f, g \rangle$ where

- \mathcal{S} is the set of possible states
- \mathcal{X} is the set of possible inputs
- \mathcal{Y} is the set of possible outputs

State Machines

A state machine is defined as $\langle \mathcal{S}, \mathcal{X}, \mathcal{Y}, s_0, f, g \rangle$ where

- \mathcal{S} is the set of possible states
- \mathcal{X} is the set of possible inputs
- \mathcal{Y} is the set of possible outputs
- $s_0 \in \mathcal{S}$ is the initial state
- $f : \mathcal{S} \times \mathcal{X} \rightarrow \mathcal{S}$ is a transition function which takes an input and a previous state and produces a next state

State Machines

A state machine is defined as $\langle \mathcal{S}, \mathcal{X}, \mathcal{Y}, s_0, f, g \rangle$ where

- \mathcal{S} is the set of possible states
- \mathcal{X} is the set of possible inputs
- \mathcal{Y} is the set of possible outputs
- $s_0 \in \mathcal{S}$ is the initial state
- $f : \mathcal{S} \times \mathcal{X} \rightarrow \mathcal{S}$ is a transition function which takes an input and a previous state and produces a next state
- $g : \mathcal{S} \rightarrow \mathcal{Y}$ is an output function, which takes a state and produces an output.

State Machines

A state machine is defined as $\langle \mathcal{S}, \mathcal{X}, \mathcal{Y}, s_0, f, g \rangle$ where

- \mathcal{S} is the set of possible states
- \mathcal{X} is the set of possible inputs
- \mathcal{Y} is the set of possible outputs
- $s_0 \in \mathcal{S}$ is the initial state
- $f : \mathcal{S} \times \mathcal{X} \rightarrow \mathcal{S}$ is a transition function which takes an input and a previous state and produces a next state
- $g : \mathcal{S} \rightarrow \mathcal{Y}$ is an output function, which takes a state and produces an output.

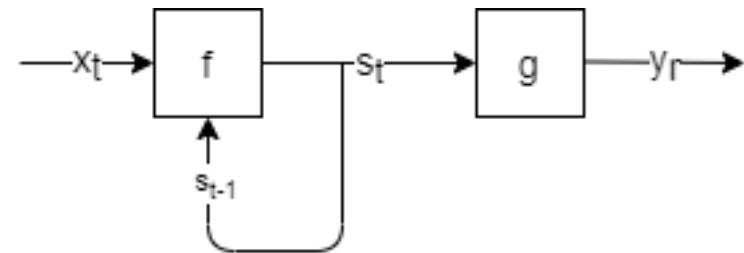
Variations are available, e.g. y may depend on both s and x

State Machines

State machine starts with s_0 and iteratively computes for $t = 1, 2, \dots$

$$s_t = f(s_{t-1}, x_t)$$

$$y_t = g(s_t)$$



State Machines

In a state machine, y depends not only on x but the whole history xs . *How?*

- First input is x_1 and first output is

$$y_1 = g(f(s_0, x1))$$

i.e. $g(s_1)$

State Machines

- First input is x_1 and first output is

$$\begin{aligned} y_1 &= g(\underbrace{f(s_0, x_1)}_{s_1}) \\ &= g(s_1) \end{aligned}$$

State Machines

- Second input is x_2 and second output is

$$y_2 = g(f(s_1, x_2))$$

State Machines

- Second input is x_2 and second output is

$$\begin{aligned} y_2 &= g(\underbrace{f(s_1, x_2)}_{s_2}) \\ &= g(s_2) \end{aligned}$$

State Machines

- Second input is x_2 and second output is

$$\begin{aligned} y_2 &= g(f(s_1, x_2)) \\ &= g(f(\underbrace{f(s_0, x_1)}_{s_1}, x_2)) \end{aligned}$$

State Machines

- Second input is x_2 and second output is

$$\begin{aligned}y_2 &= g(s_2) \\&= g(f(s_1, x_2)) \\&= g(f(f(s_0, x_1), x_2))\end{aligned}$$

State Machines

- Third input is x_3 and third output is

$$\begin{aligned} y_3 &= g(\underbrace{f(s_2, x_3)}_{s_3}) \\ &= g(s_3) \end{aligned}$$

State Machines

- Third input is x_3 and third output is

$$\begin{aligned} y_3 &= g(\underbrace{f(s_2, x_3)}_{s_3}) \\ &= g(f(\underbrace{f(\underbrace{s_1, x_2}_{s_2}, x_3)}_{s_1})) \end{aligned}$$

State Machines

- Third input is x_3 and third output is

$$\begin{aligned}y_3 &= g(s_3) \\&= g(f(s_2, x_3)) \\&= g(f(f(s_1, x_2), x_3)) \\&= g(f(\underbrace{f(f(s_0, x_1), x_2)}_{s_1}, x_3)) \\&\quad \underbrace{\quad}_{s_2} \\&\quad \underbrace{\quad}_{s_3}\end{aligned}$$

State Machines

- Third input is x_3 and third output is

$$\begin{aligned}y_3 &= g(s_3) \\&= g(f(s_2, x_3)) \\&= g(f(f(s_1, x_2), x_3)) \\&= g(f(f(f(s_0, x_1), x_2), x_3))\end{aligned}$$

y depends not only on x but the whole history xs

State Machines

Given a sequence of inputs x_1, x_2, \dots the machine generates a sequence of outputs:

$$\underbrace{g(f(s_0, x_1))}_{y_1}, \underbrace{g(f(f(s_0, x_1), x_2))}_{y_2}, \dots$$

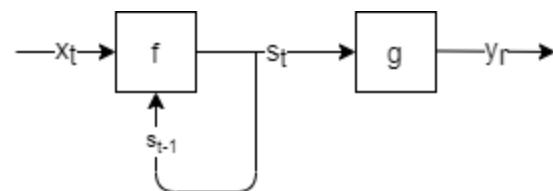
The output at t can depend on inputs from steps 1 to t .

State Machine

State machine starts with s_0 and iteratively computes for $t = 1, 2, \dots$

$$s_t = f(s_{t-1}, x_t)$$

$$y_t = g(s_t)$$



Given a sequence of inputs x_1, x_2, \dots the machine generates a sequence of outputs:

$$y_1 = g(f(s_0, x_1))$$

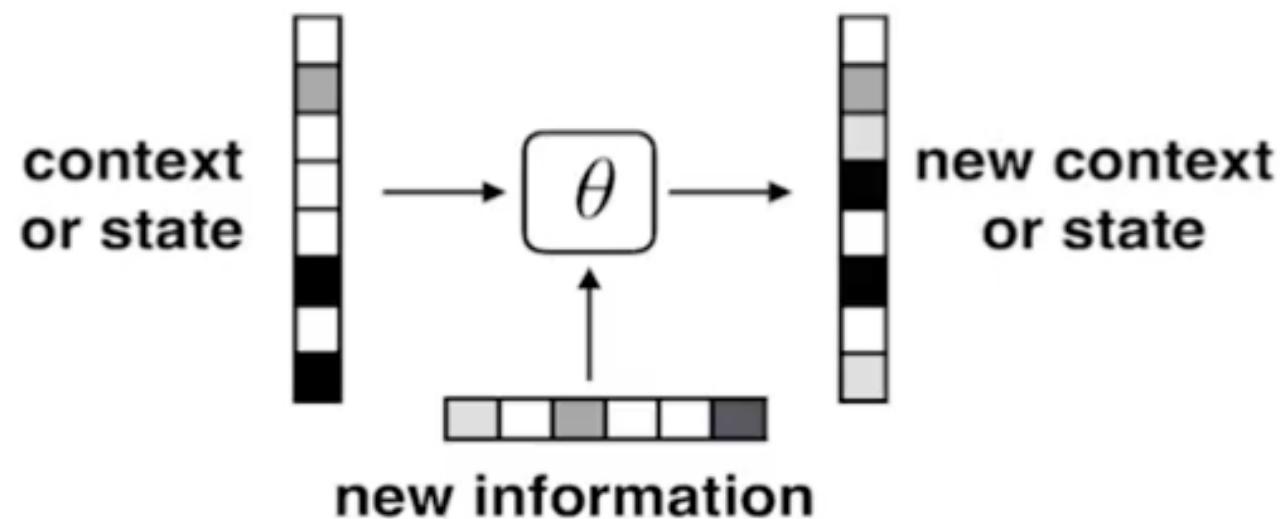
$$y_2 = g(f(s_1, x_2)) = g(f(f(s_0, x_1), x_2))$$

$$y_3 = g(f(s_2, x_3)) = g(f(f(s_1, x_2), x_3)) = g(f(f(f(s_0, x_1), x_2), x_3))$$

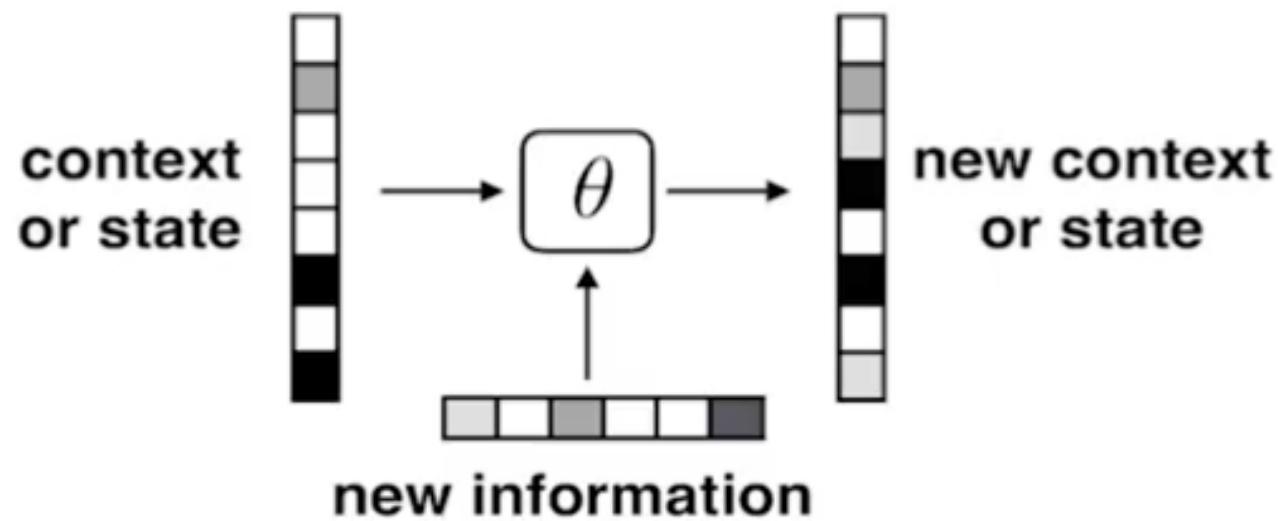
⋮

Recurrent Neural Networks

Recurrent Neural Networks (encoding)

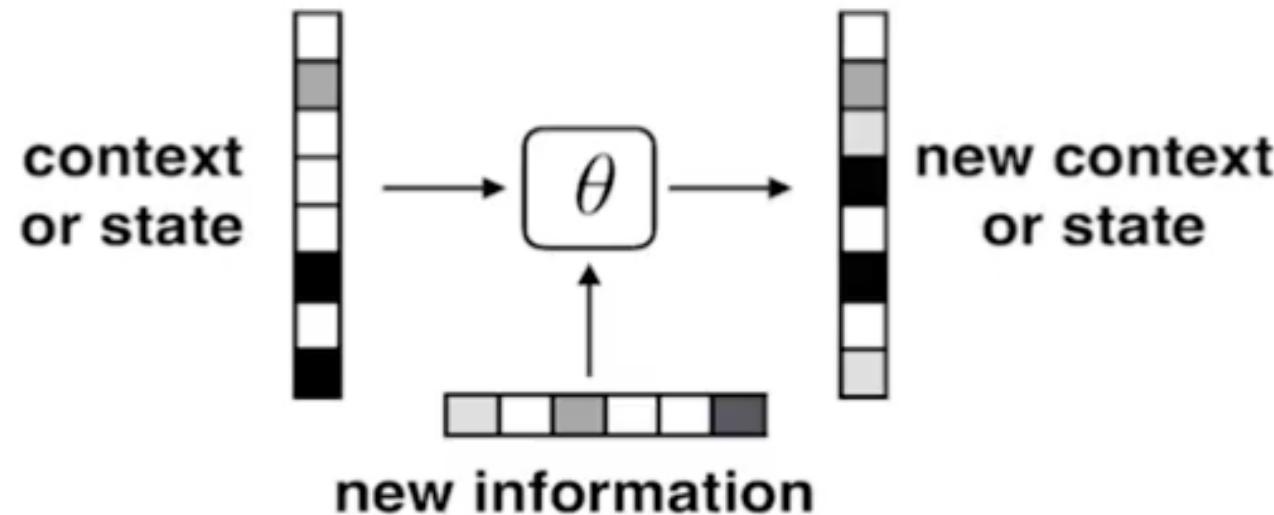


Recurrent Neural Networks (encoding)



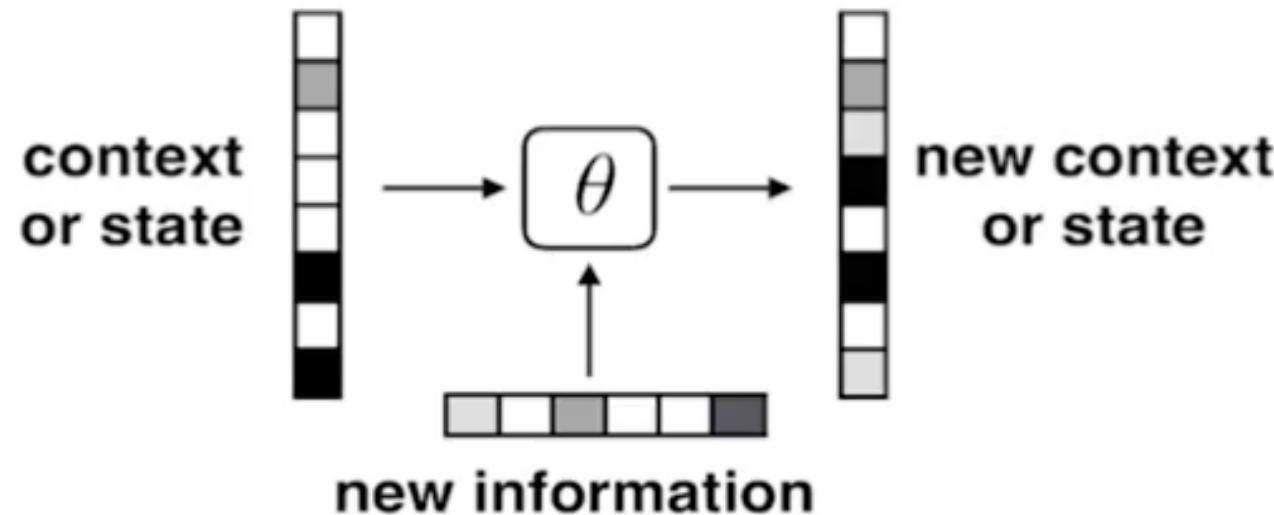
$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$

Recurrent Neural Networks (encoding)



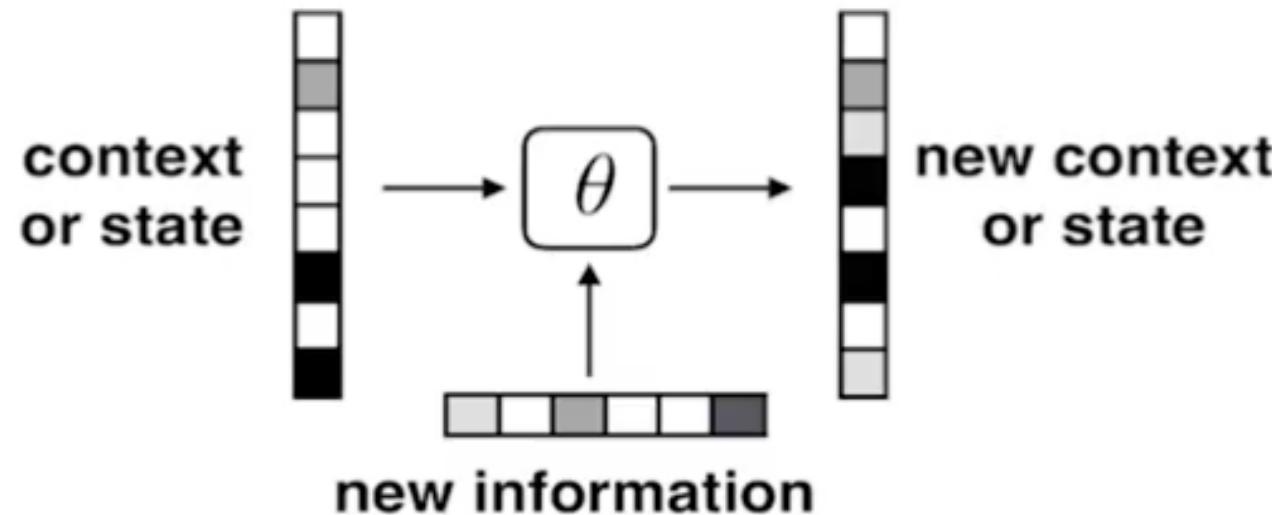
$$\underbrace{s_t}_{m \times 1} = \tanh(W^{s,s} \underbrace{s_{t-1}}_{m \times 1} + W^{s,x} x_t)$$

Recurrent Neural Networks (encoding)



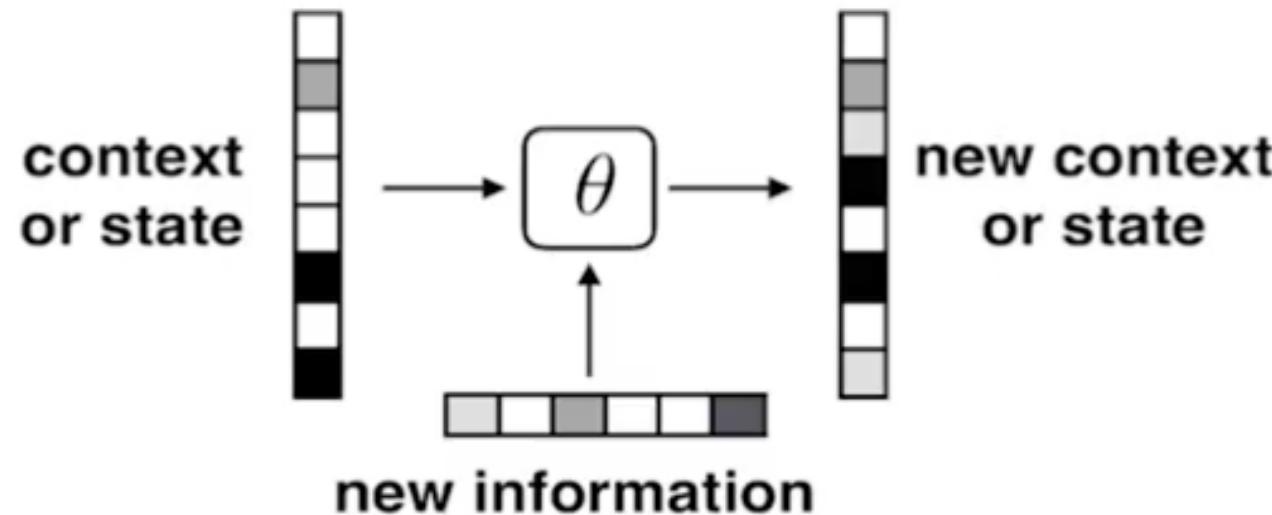
$$\underbrace{s_t}_{m \times 1} = \tanh(\underbrace{W^{s,s}}_{m \times m} \underbrace{s_{t-1}}_{m \times 1} + \underbrace{W^{s,x}}_{m \times 1} x_t)$$

Recurrent Neural Networks (encoding)



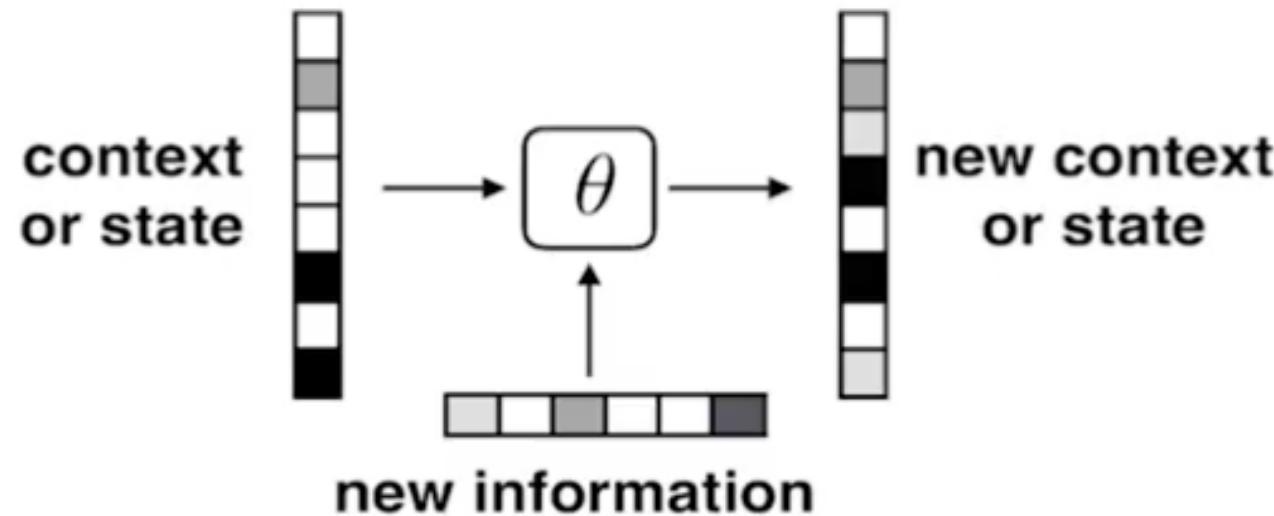
$$\underbrace{s_t}_{m \times 1} = \tanh(\underbrace{W^{s,s}}_{m \times m} \underbrace{s_{t-1}}_{m \times 1} + \underbrace{W^{s,x}}_{m \times d} \underbrace{x_t}_{d \times 1})$$

Recurrent Neural Networks (encoding)



$$\underbrace{s_t}_{m \times 1} = \tanh(\underbrace{W^{s,s}}_{m \times m} \underbrace{s_{t-1}}_{m \times 1} + \underbrace{W^{s,x}}_{m \times d} \underbrace{x_t}_{d \times 1})$$

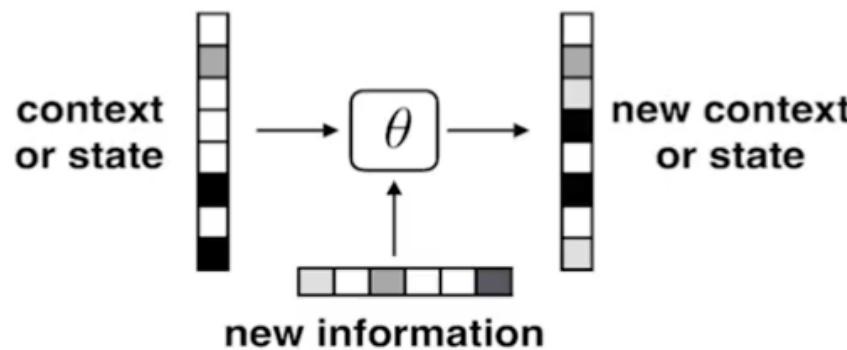
Recurrent Neural Networks (encoding)



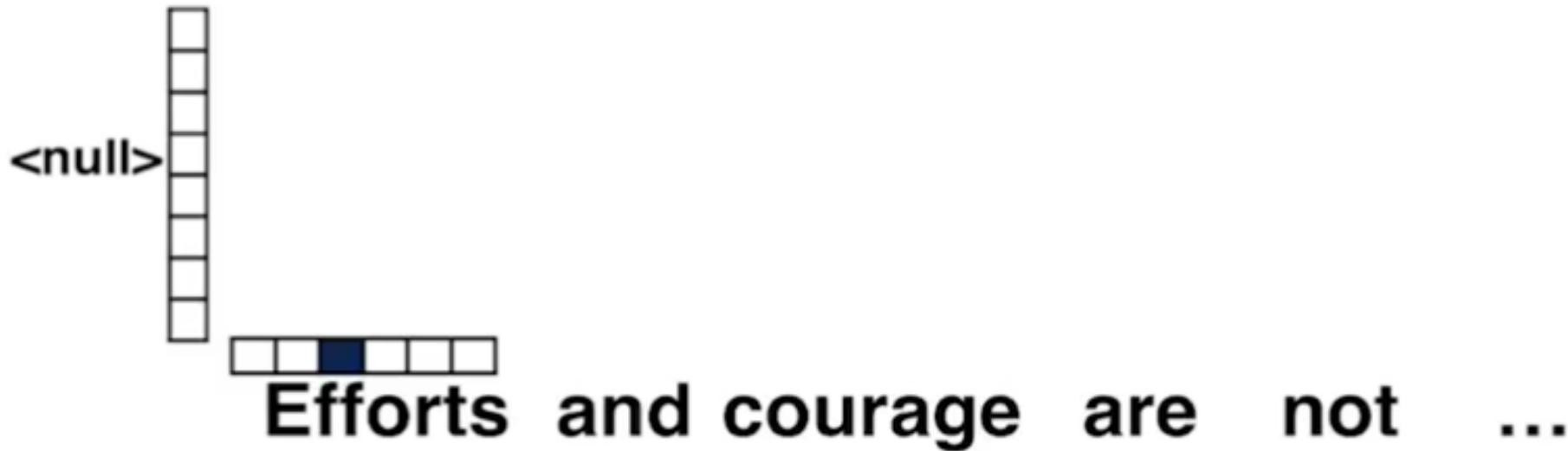
with offset

$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t + W_0^{s,s})$$

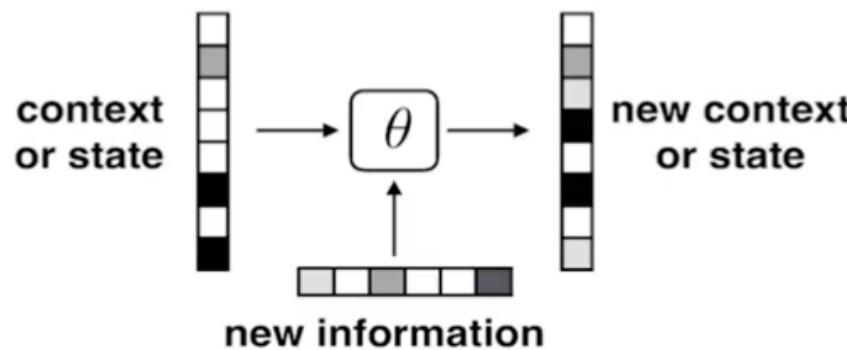
Recurrent Neural Networks (encoding)



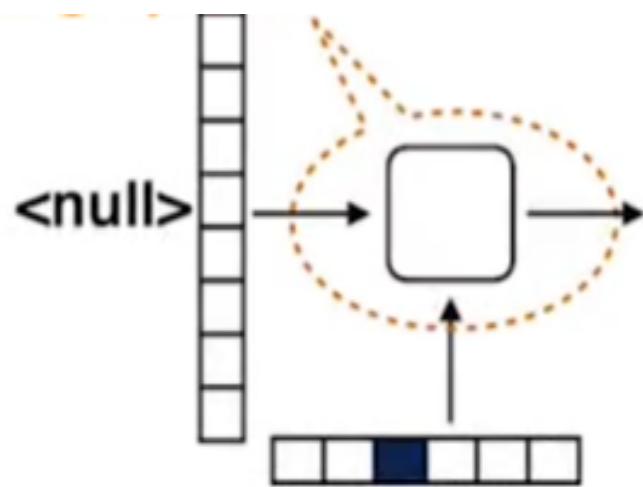
$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$



Recurrent Neural Networks (encoding)

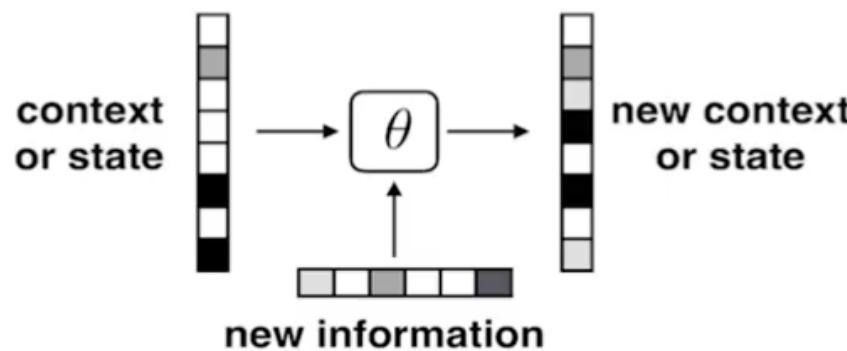


$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$

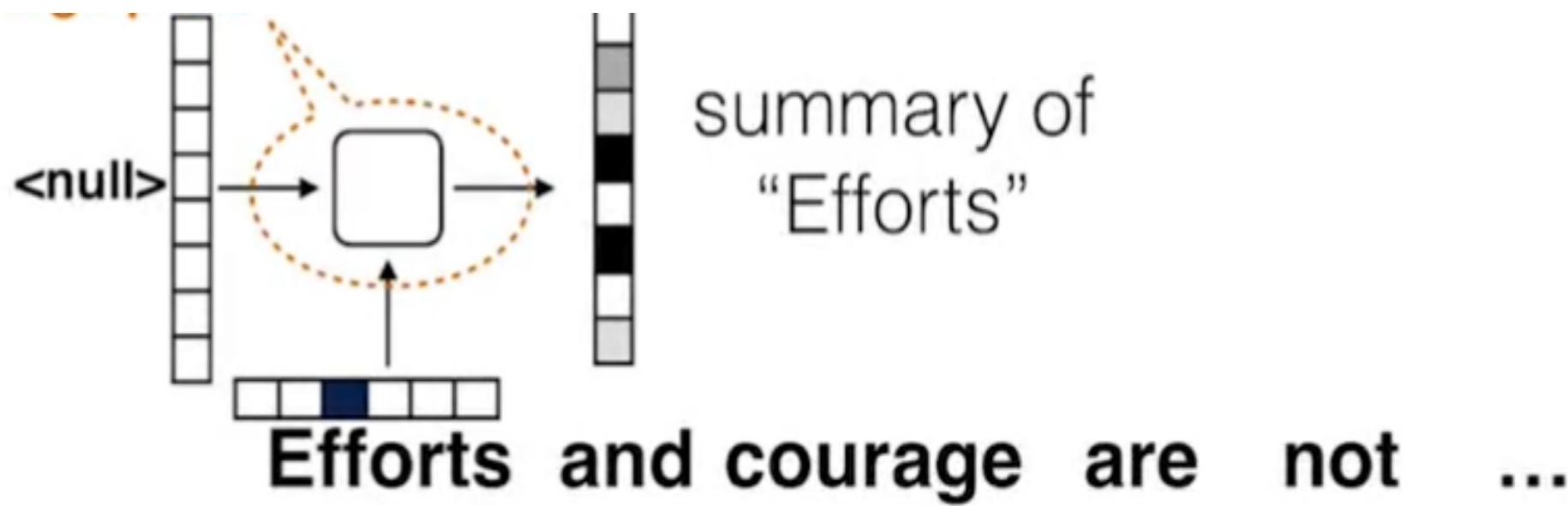


Efforts and courage are not ...

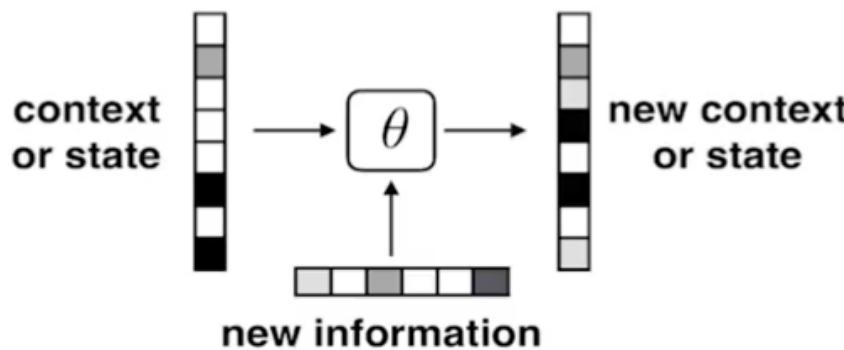
Recurrent Neural Networks (encoding)



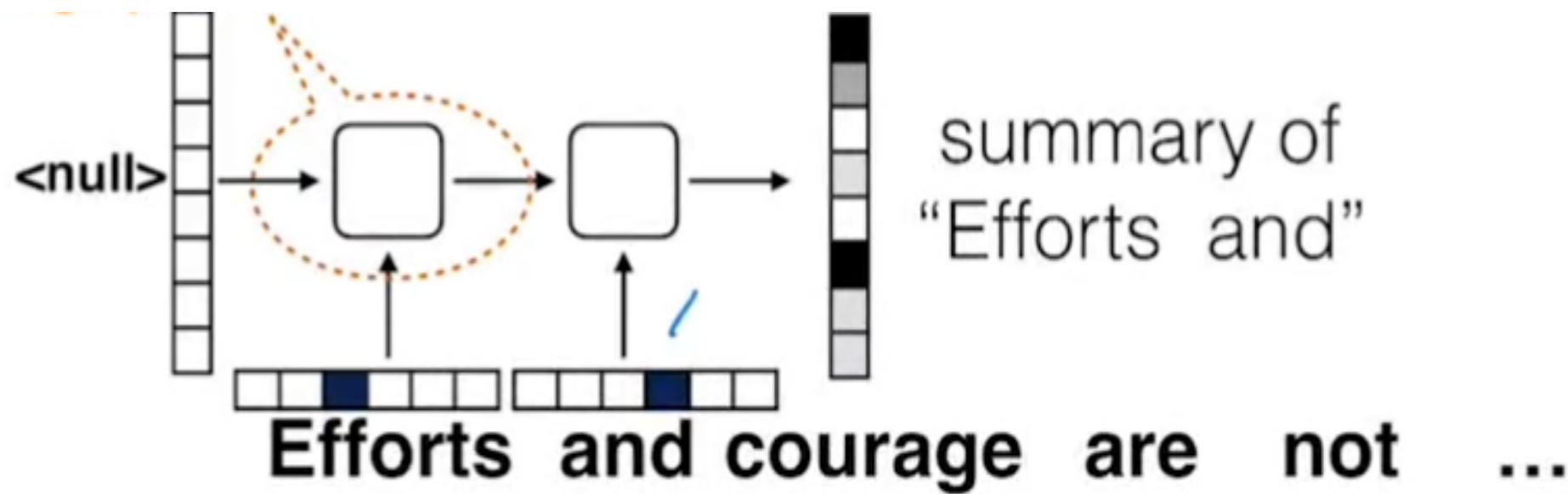
$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$



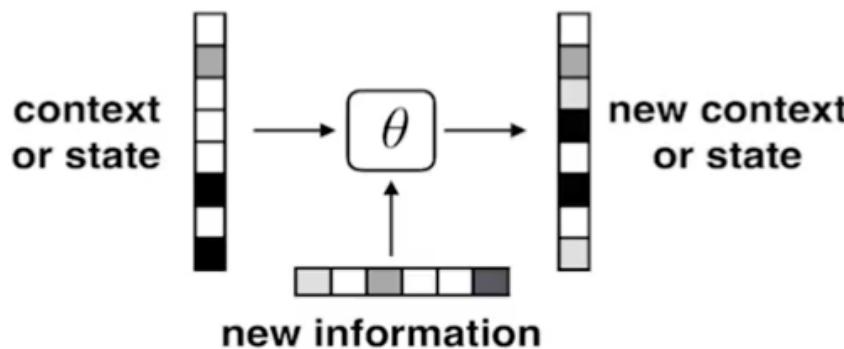
Recurrent Neural Networks (encoding)



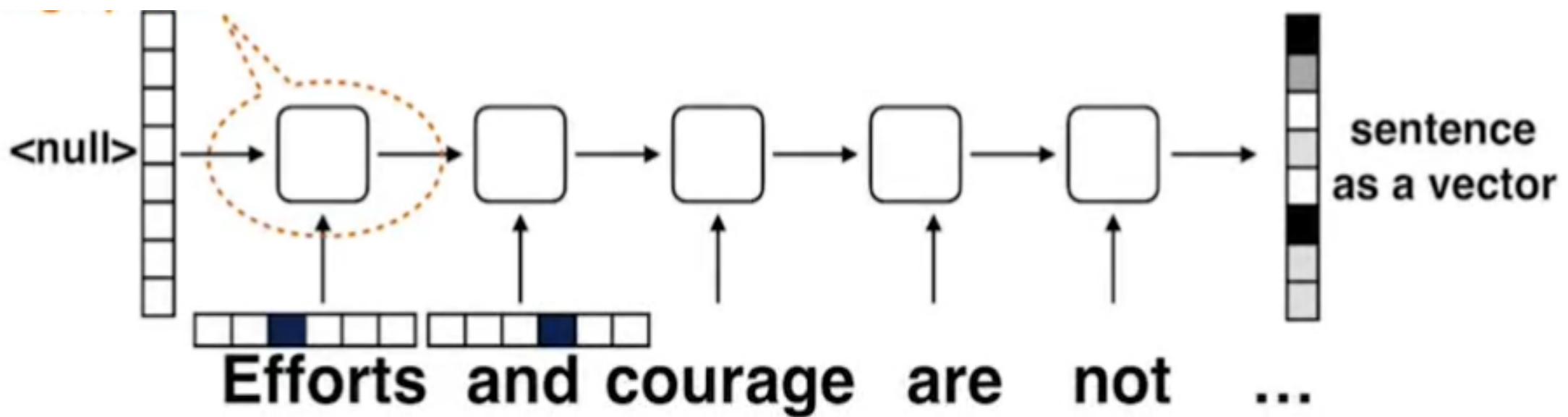
$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$



Recurrent Neural Networks (encoding)

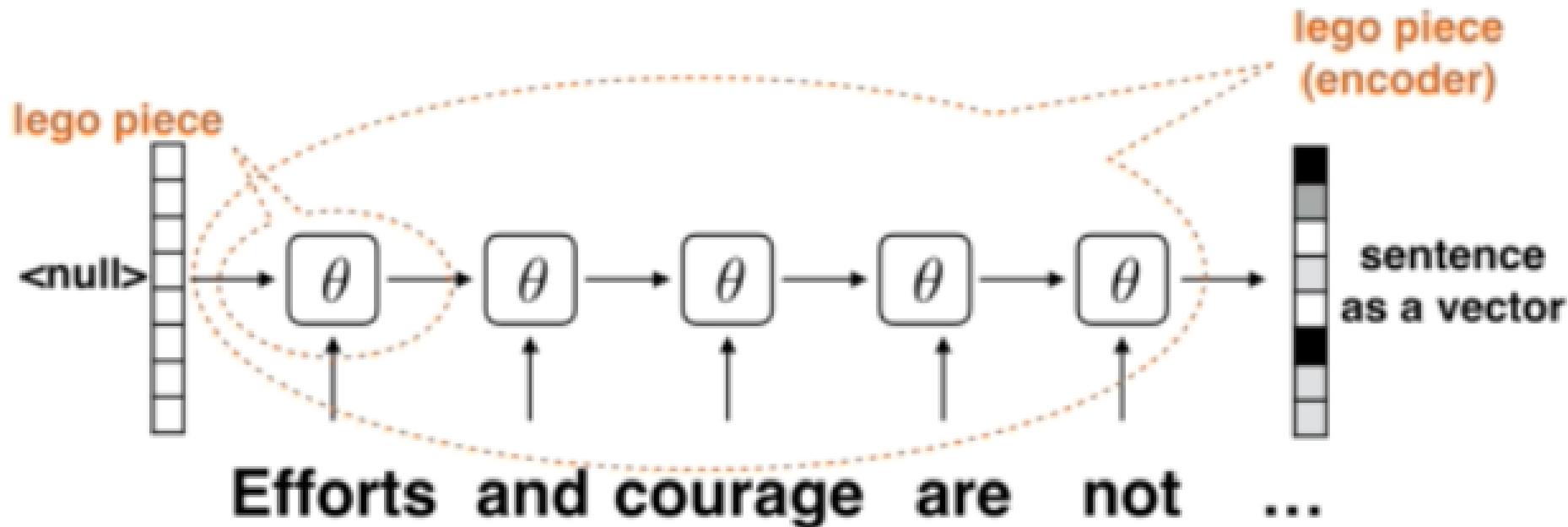


$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$



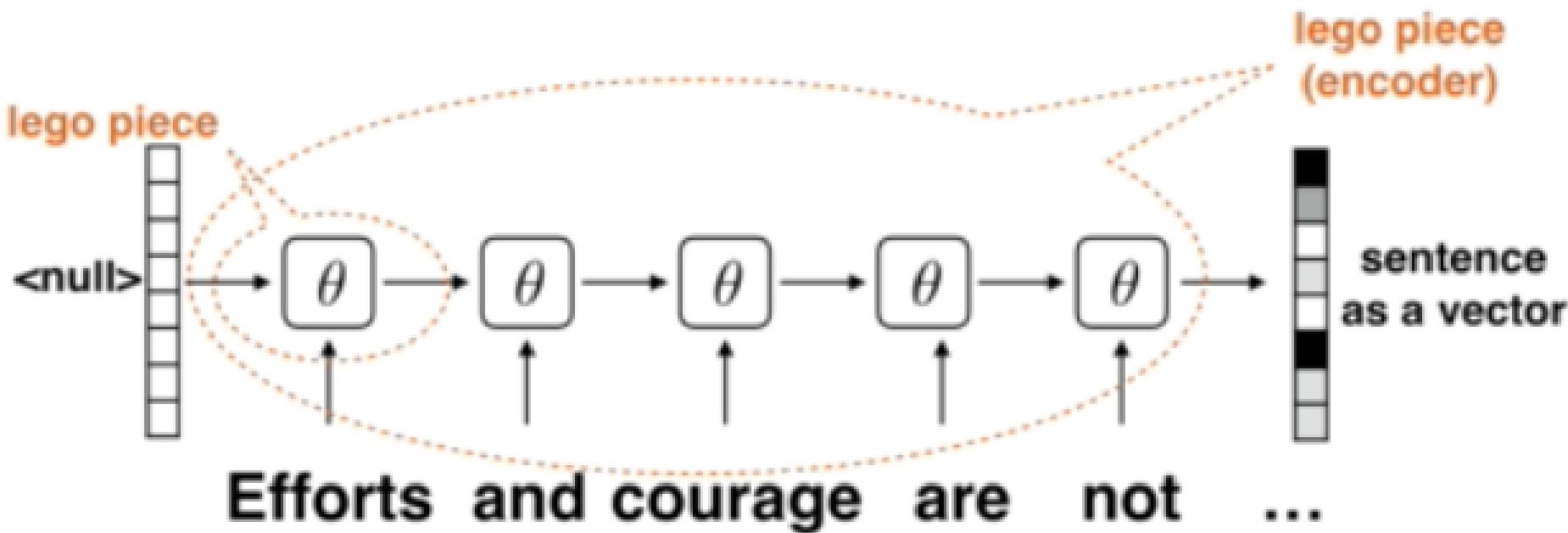
Differences between encoder and standard feed forward NNs

- input is received at each layer, not just at the beginning
- number of layers varies and depends on the lenght of the sentence
- parameters of each layer are shared (same RNN at each step)



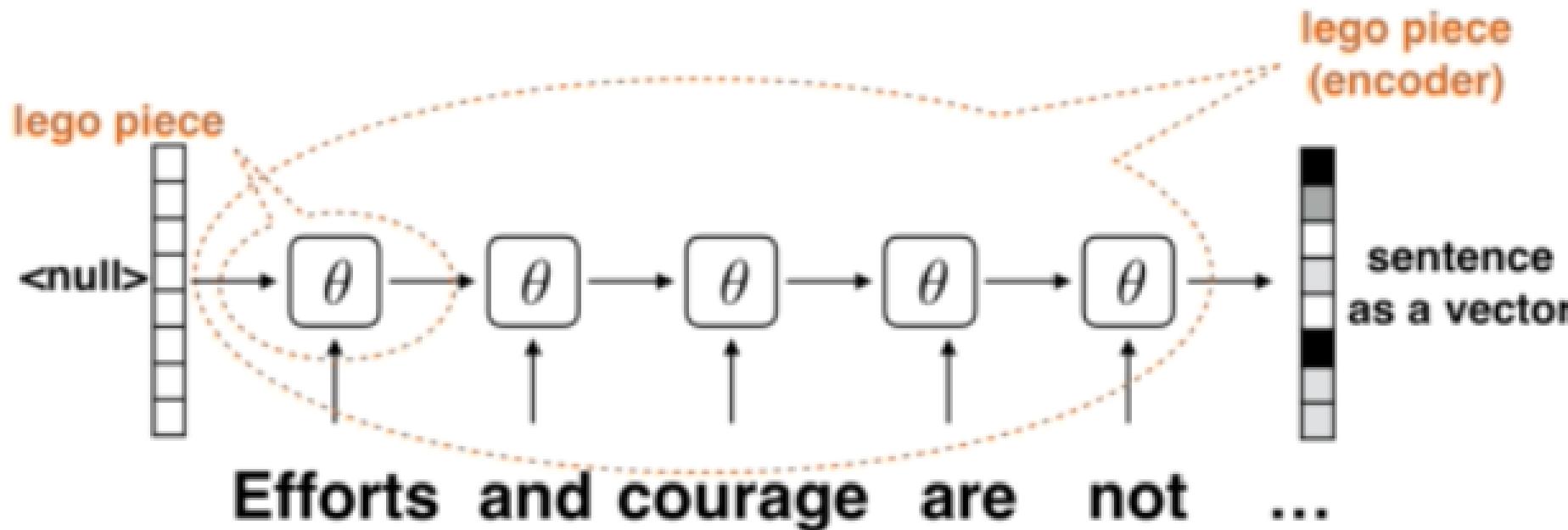
Learning RNNs

- RNNs are learned with backpropagation
 - Turn sentence into a vector
 - Make a prediction (compute loss)
 - Backpropagate error signal
- *Parameters are shared*



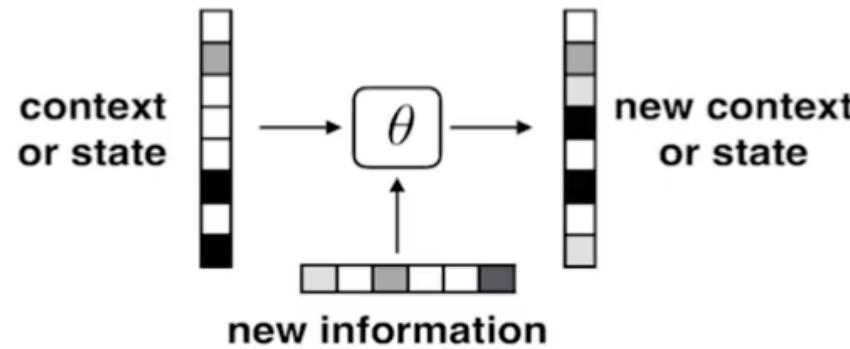
Problems with Learning RNNs - Vanishing & Exploding Gradients

- Consider a case where only the output at the *end of the sequence* is incorrency, and it depends on the input at time 1.
- We backpropagate the layer through the layer-wise transformation
- Becomes a very deep network for large sequences
- Gradients can vanish or explode (see Week 9)



Extensions to RNN

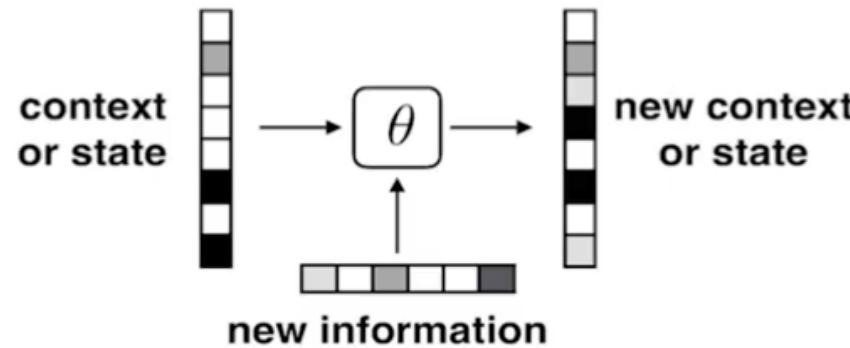
Recurrent Neural Networks (encoding)



$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$

State is completely overridden with new information

Gating (Simple Gated Neural Network)

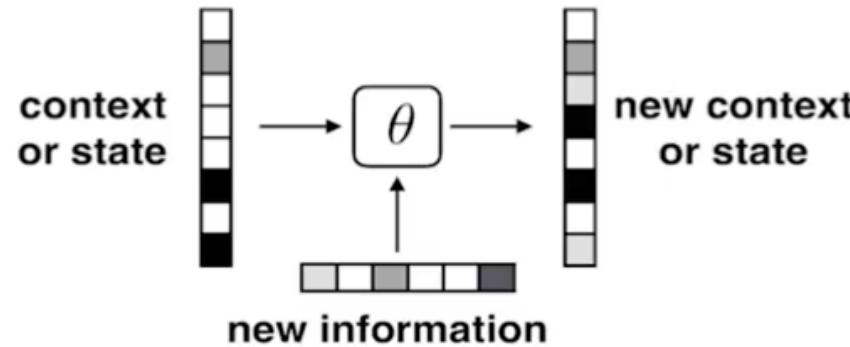


$$g_t = \text{sigmoid}(W^{g,s} s_{t-1} + W^{g,x} x_t)$$

$$s_t = (1 - g_t) \odot s_{t-1} + g_t \odot \tanh(W^{s,s} s_{t-1} + W^{s,x} x_t)$$

Gating decides how much of the state should be updated. This enables network to store some information in some dimension, and change it under only certain conditions on the input.

Gating (Simple Gated Neural Network)

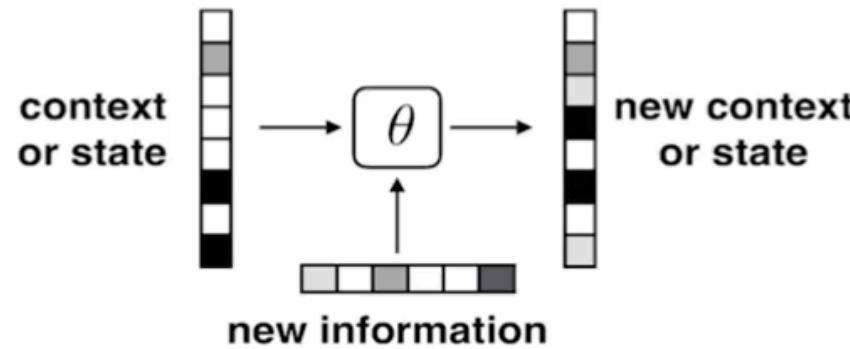


$$g_t = \text{sigmoid}(W^{g,s} s_{t-1} + W^{g,x} x_t)$$

$$g_t \in [0, 1]$$

$$s_t = (1 - g_t) \odot s_{t-1} + g_t \odot \tanh(W^{s,s} s_{t-1} + W^{s,x} x_t)$$

Gating (Simple Gated Neural Network)



$$g_t = \text{sigmoid}(W^{g,s} s_{t-1} + W^{g,x} x_t)$$

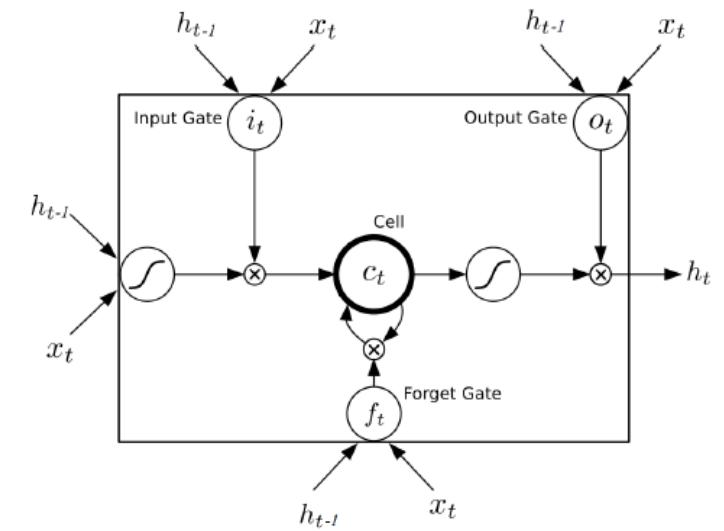
$$g_t \in [0, 1]$$

$$s_t = \underbrace{(1 - g_t)}_{\begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}} \odot s_{t-1} + \underbrace{g_t}_{\begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix}} \odot \tanh(W^{s,s} s_{t-1} + W^{s,x} x_t)$$

We can make RNNs even more sophisticated.

Long short-term memory (LSTM) Networks

- It has three gating networks
- *Input gate* selects which dimensions of the state will be updated with new values
- *Forget gate* decides which dimensions of the states will have old values toward 0
- *Output gate* decides, which dimensions will be used to compute the output value.



Long short-term memory (LSTM) Networks

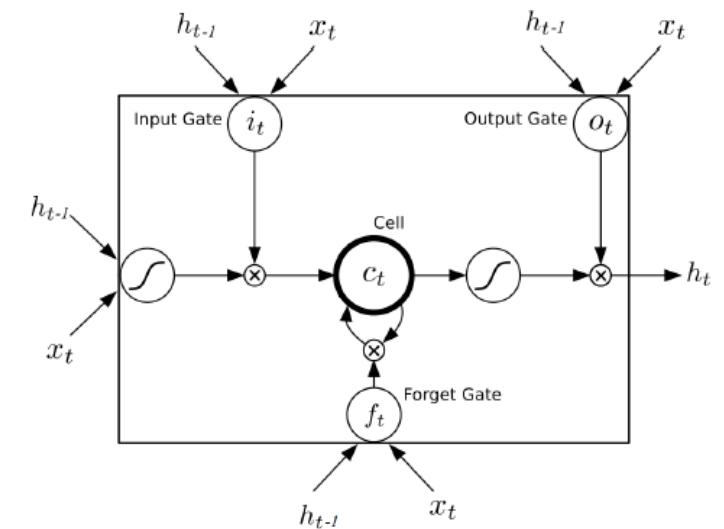
$$f_t = \text{sigmoid}(W^{f,h}h_{t-1} + W^{f,x}x_t) \text{ forget gate}$$

$$i_t = \text{sigmoid}(W^{i,h}h_{t-1} + W^{i,x}x_t) \text{ input gate}$$

$$o_t = \text{sigmoid}(W^{o,h}h_{t-1} + W^{o,x}x_t) \text{ output gate}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W^{c,h}h_{t-1} + W^{c,x}x_t) \text{ memory cell}$$

$$h_t = o_t \odot \tanh(c_t) \text{ visible state}$$



COGS514 - Cognition and Machine Learning

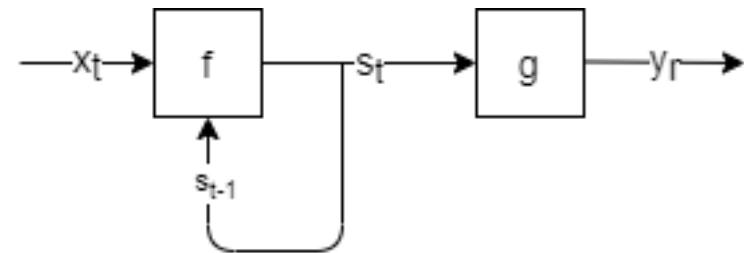
Recurrent Neural Networks - 2

State Machines

State machine starts with s_0 and iteratively computes for $t = 1, 2, \dots$

$$s_t = f(s_{t-1}, x_t)$$

$$y_t = g(s_t)$$



Learning to encode/decode

- Language modeling

This course has been a



success (?)

- Sentiment classification

I have seen better lectures



-1

- Machine translation

I have seen better lectures



Olen nähnyt parempia
luentoja

encoding

decoding

Modeling Sequences: Language Models

- Markov Models
- Neural Networks
- Hidden State, Recurrent Neural Networks (RNN)

Markov Models

- Next word in a sentence depends on previous words (one, two or more words)

The lecture leaves me befuddled

Markov Language Models

- Let $w \in V$ denote the set of possible words that includes
 - UNK symbol for any unknown word (out of vocabulary)
 - symbol for specifying start of sentence
 - symbol for specifying end of sentence
- $\text{<beg> The lecture leaves me UNK <end>}$

$w_0 \quad w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5 \quad w_6$

Markov Language Models

- Let $w \in V$ denote the set of possible words that includes
 - UNK symbol for any unknown word (out of vocabulary)
 - symbol for specifying start of sentence
 - symbol for specifying end of sentence
- $\text{<beg> The lecture leaves me UNK <end>}$
- $w_0 \quad w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5 \quad w_6$

In a bigram model (first order Markov model), the next word only depends on the previous one

$$P(w_1, w_2, \dots, w_6) = P(w_1 \mid w_0)P(w_2 \mid w_1) \dots P(w_6 \mid w_5)$$

First order Markov model

- Each symbol is predicted using the same conditional probability table until an `is` is seen.

w_i

		ML	course	is	UNK	<end>
		0.7	0.1	0.1	0.1	0.0
		0.1	0.5	0.2	0.1	0.1
w_{i-1}	<beg>	0.0	0.0	0.7	0.1	0.2
	ML	0.1	0.3	0.0	0.6	0.0
course	is	0.1	0.2	0.2	0.3	0.2
	UNK					

First order Markov model

- Each symbol is predicted using the same conditional probability table until an is seen.

		w_i				
		ML	course	is	UNK	<end>
w_{i-1}	<beg>	0.7	0.1	0.1	0.1	0.0
	ML	0.1	0.5	0.2	0.1	0.1
	course	0.0	0.0	0.7	0.1	0.2
	is	0.1	0.3	0.0	0.6	0.0
	UNK	0.1	0.2	0.2	0.3	0.2

course is great

First order Markov model

- Each symbol is predicted using the same conditional probability table until an `is` is seen.

		w_i				
		ML	course	is	UNK	<end>
w_{i-1}	<beg>	0.7	0.1	0.1	0.1	0.0
	ML	0.1	0.5	0.2	0.1	0.1
	course	0.0	0.0	0.7	0.1	0.2
	is	0.1	0.3	0.0	0.6	0.0
	UNK	0.1	0.2	0.2	0.3	0.2

course is UNK

First order Markov model

- Each symbol is predicted using the same conditional probability table until an `is` is seen.

$$w_i$$

		ML	course	is	UNK	<end>
		0.7	0.1	0.1	0.1	0.0
		0.1	0.5	0.2	0.1	0.1
w_{i-1}	course	0.0	0.0	0.7	0.1	0.2
	is	0.1	0.3	0.0	0.6	0.0
		0.1	0.2	0.2	0.3	0.2

`<beg> course is UNK <end>`

First order Markov model

- Each symbol is predicted using the same conditional probability table until an is seen.

		w_i				
		ML	course	is	UNK	<end>
w_{i-1}	<beg>	0.7	0.1	0.1	0.1	0.0
	ML	0.1	0.5	0.2	0.1	0.1
	course	0.0	0.0	0.7	0.1	0.2
	is	0.1	0.3	0.0	0.6	0.0
	UNK	0.1	0.2	0.2	0.3	0.2

<beg> course is UNK <end>

$$P(\text{course} \mid \text{<beg>})P(\text{is} \mid \text{course})P(\text{UNK} \mid \text{course})P(\text{end} \mid \text{UNK})$$

First order Markov model

- Each symbol is predicted using the same conditional probability table until an is seen.

		w_i				
		ML	course	is	UNK	<end>
w_{i-1}	<beg>	0.7	0.1	0.1	0.1	0.0
	ML	0.1	0.5	0.2	0.1	0.1
	course	0.0	0.0	0.7	0.1	0.2
	is	0.1	0.3	0.0	0.6	0.0
	UNK	0.1	0.2	0.2	0.3	0.2

<beg> course is UNK <end>

$$0.1 \times 0.7 \times 0.6 \times 0.2 = 0.0084$$

Sampling from a Markov model

		w_i				
		ML	course	is	UNK	<end>
w_{i-1}	<beg>	0.7	0.1	0.1	0.1	0.0
	ML	0.1	0.5	0.2	0.1	0.1
	course	0.0	0.0	0.7	0.1	0.2
	is	0.1	0.3	0.0	0.6	0.0
	UNK	0.1	0.2	0.2	0.3	0.2

Learning the probabilities in Markov Model (Maximum Likelihood Estimation)

- Maximize the probability that model can generate all observed sentences in corpus S
- ML estimate is obtained as normalized counts of successive word occurrences

$$\hat{P}(w' \mid w) = \frac{\text{count}(w, w')}{\sum_{\tilde{w}} \text{count}(w, \tilde{w})}$$

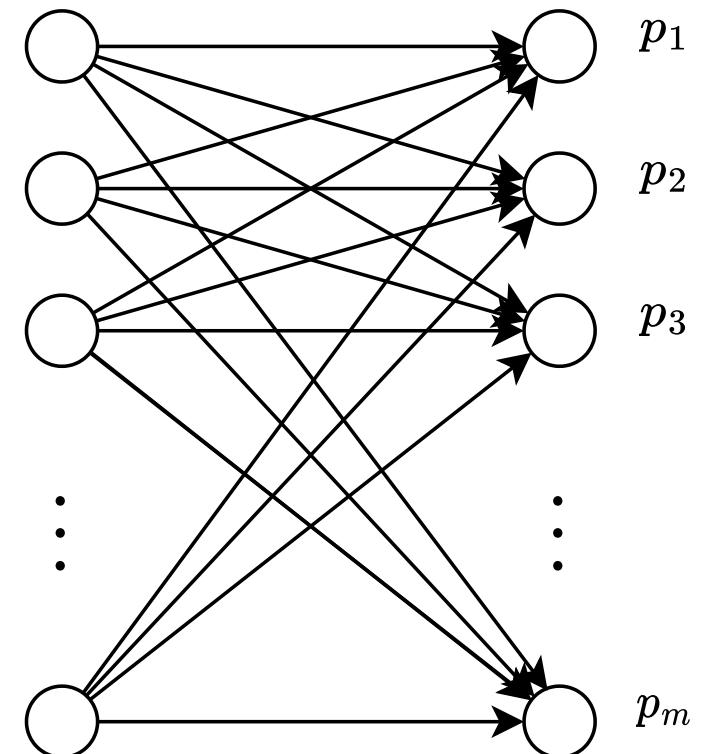
Modeling Markov model as a feed forward neural network

Modeling Markov model as a feed forward neural network

$$z_k = Wx_j + w_o$$

$$p_k = \text{softmax}(z_k)$$

$$x = \phi(w_{i-1}) \quad W \quad p_k = p(w_i = k | w_{i-1})$$



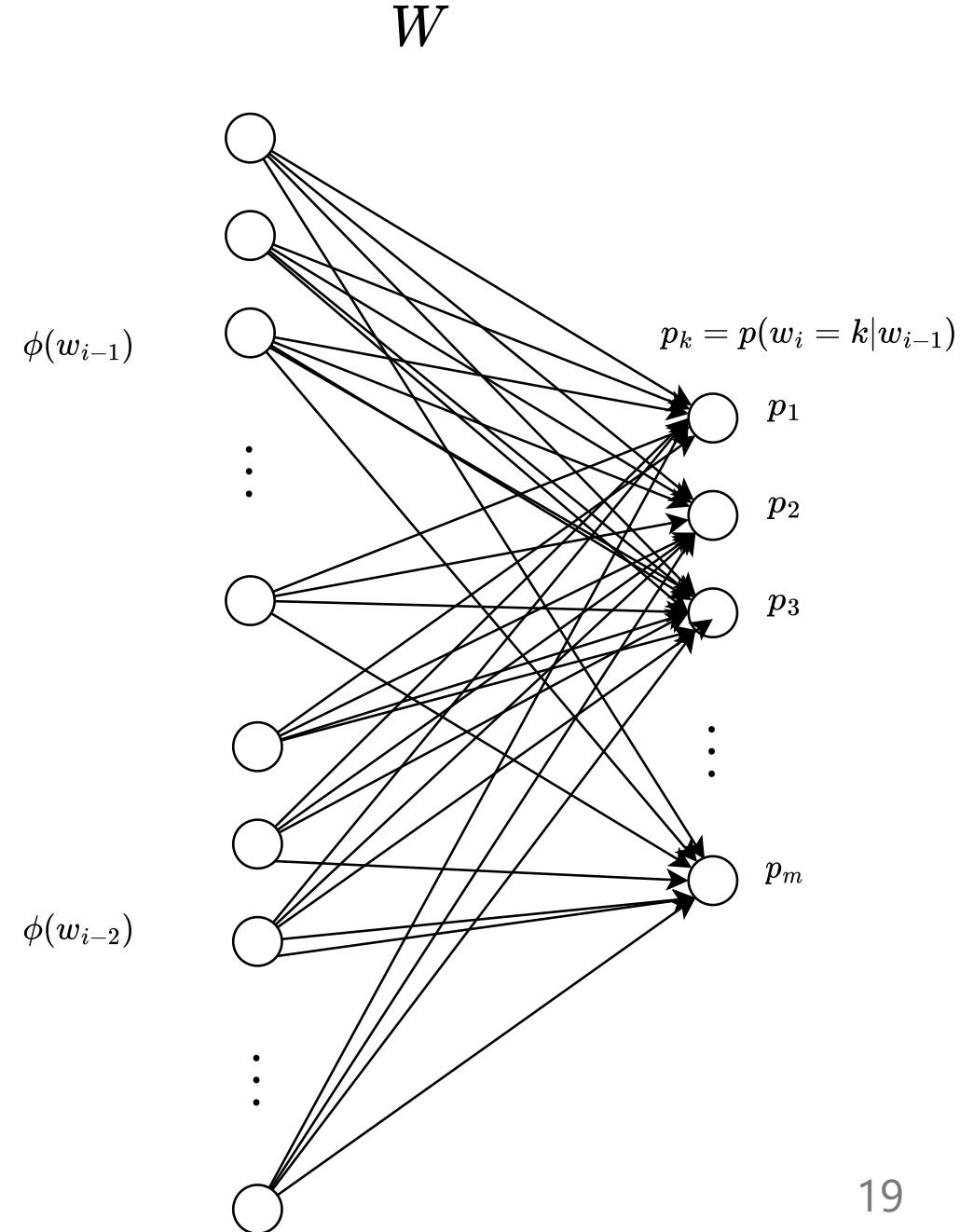
Softmax - reminder

Softmax takes a vector z and generates a output vector $a \in [0, 1]^n$ with the property that $\sum_i a_i = 1$. We can interpret it as a probability distribution over n items. Used in the final layer for multi-class classification problems.

$$\text{softmax}(z) = \begin{bmatrix} e^{z_1} / \sum_i e^{z_i} \\ \vdots \\ e^{z_n} / \sum_i e^{z_i} \end{bmatrix} \quad z = \begin{bmatrix} 6 \\ -3 \\ 2.5 \\ 0.4 \end{bmatrix}$$

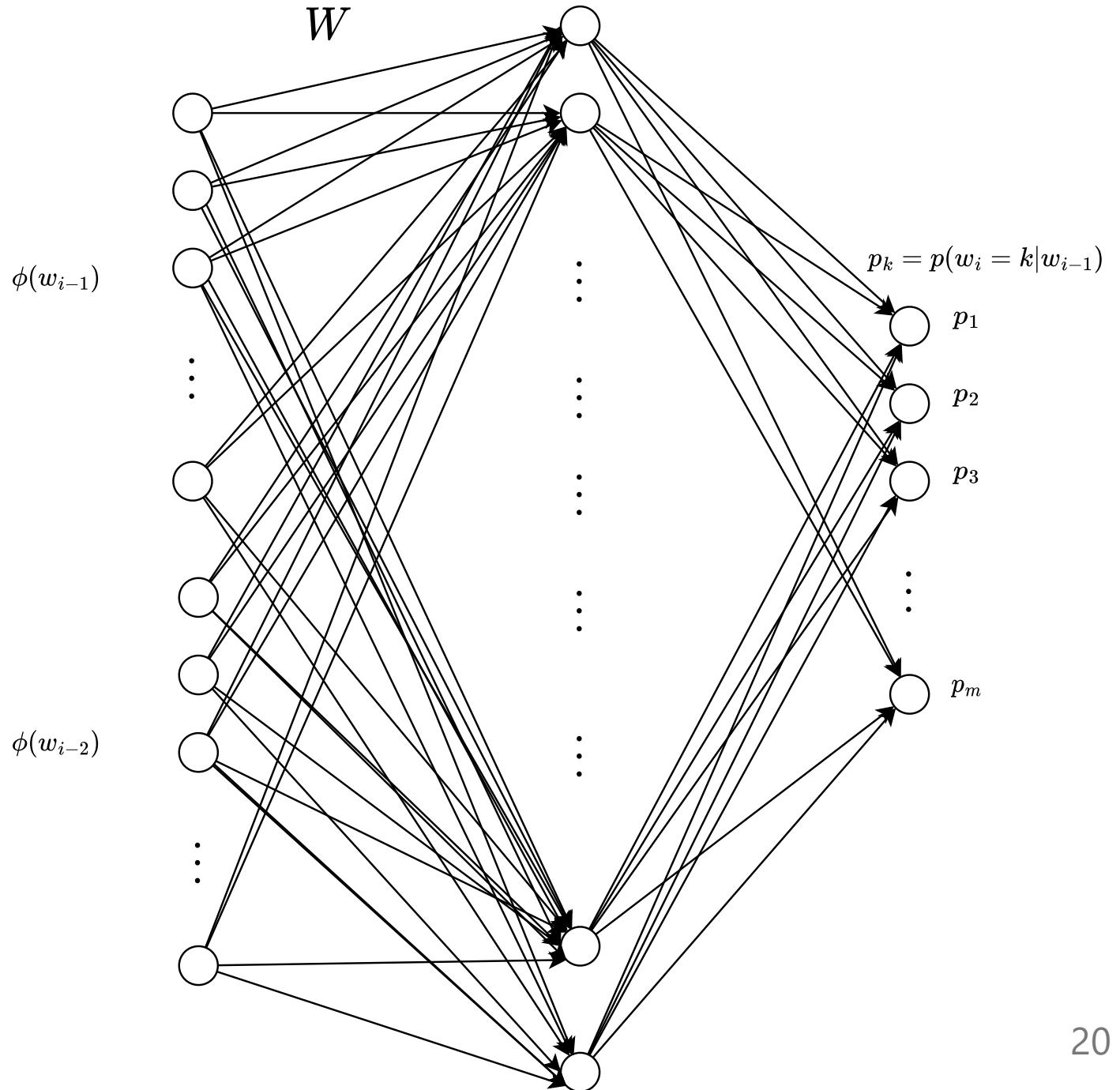
$$\text{softmax}(z) = \begin{bmatrix} \frac{e^6}{e^6 + e^{-3} + e^{2.5} + e^{0.4}} \\ \frac{e^{-3}}{e^6 + e^{-3} + e^{2.5} + e^{0.4}} \\ \frac{e^{2.5}}{e^6 + e^{-3} + e^{2.5} + e^{0.4}} \\ \frac{e^{0.4}}{e^6 + e^{-3} + e^{2.5} + e^{0.4}} \end{bmatrix} = \begin{bmatrix} 0.9671 \\ 0.0001 \\ 0.0292 \\ 0.0036 \end{bmatrix}$$

Modelling trigram model (Second Order Markov Model) as feed forward neural network



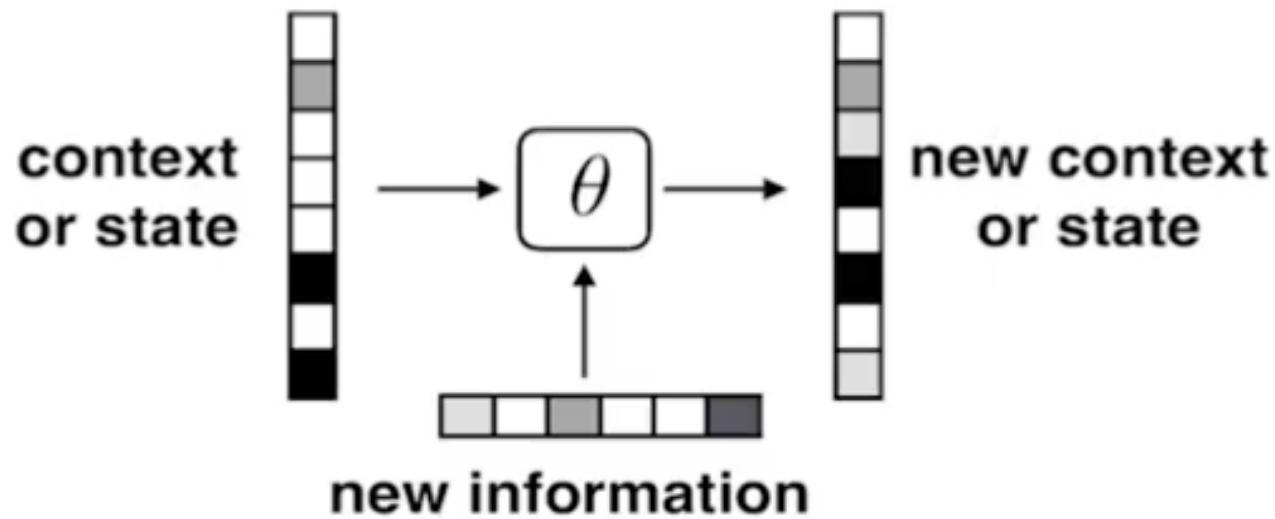
Adding hidden layers

Easily extendable



RNN Language Models

RNN Language Models



$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$

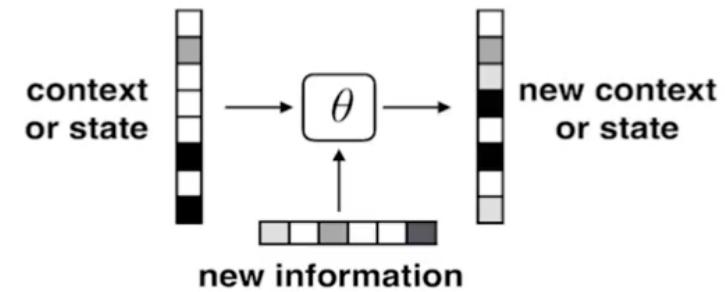
This course has been a tremendous | ...

$$x_t = \text{tremendous} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

This course has been a tremendous | ...

$$x_t = \text{tremendous} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$

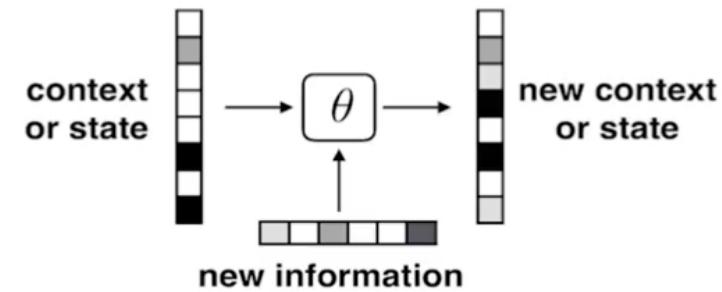


This course has been a tremendous | ...

$$x_t = \text{tremendous} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$

$$p_t = \text{softmax}(W^o s_t)$$

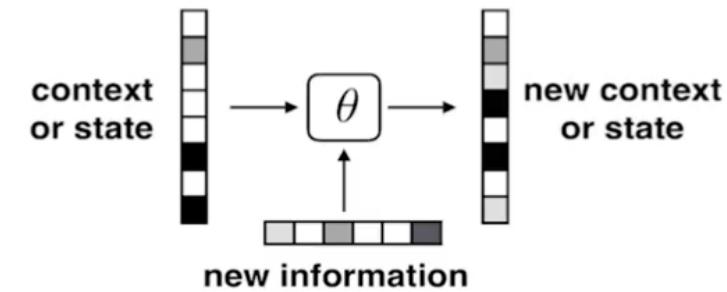


This course has been a tremendous | ...

$$x_t = \text{tremendous} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

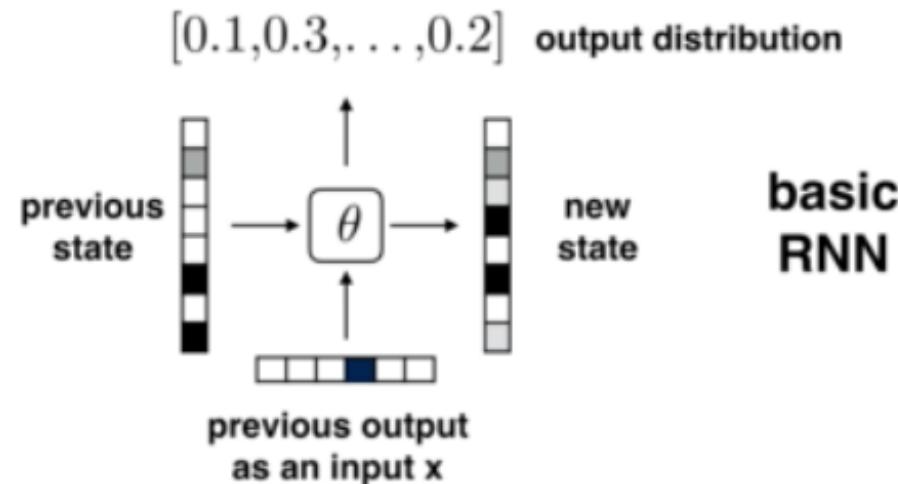
$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t) \quad \text{state}$$

$$p_t = \text{softmax}(W^o s_t) \quad \text{output distribution}$$



Decoding, RNNs

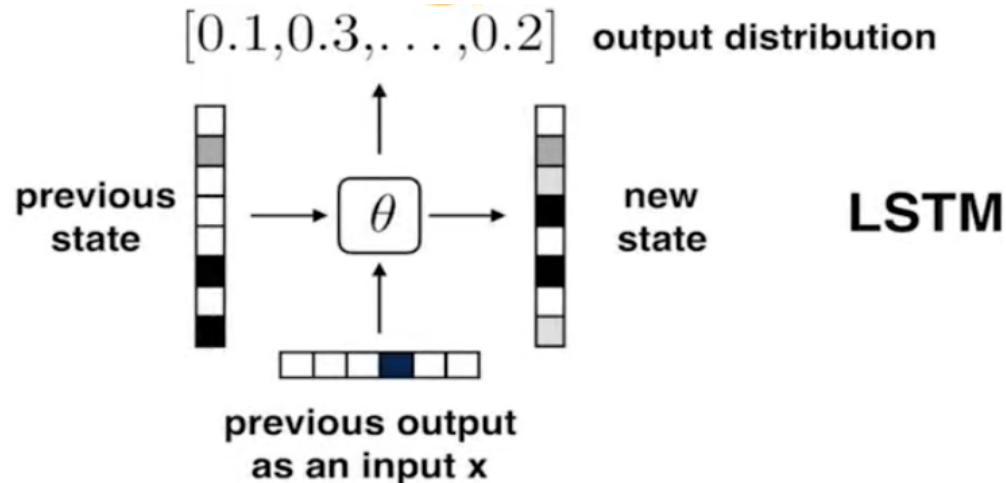
RNN produces both output and state



$$s_t = \tanh(W^{s,s} s_{t-1} + W^{s,x} x_t) \quad \text{state}$$

$$p_t = \text{softmax}(W^o s_t) \quad \text{output distribution}$$

Decoding, LSTMs



$$f_t = \text{sigmoid}(W^{f,h}h_{t-1} + W^{f,x}x_t) \text{ forget gate}$$

$$i_t = \text{sigmoid}(W^{i,h}h_{t-1} + W^{i,x}x_t) \text{ input gate}$$

$$o_t = \text{sigmoid}(W^{o,h}h_{t-1} + W^{o,x}x_t) \text{ output gate}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W^{c,h}h_{t-1} + W^{c,x}x_t) \text{ memory cell}$$

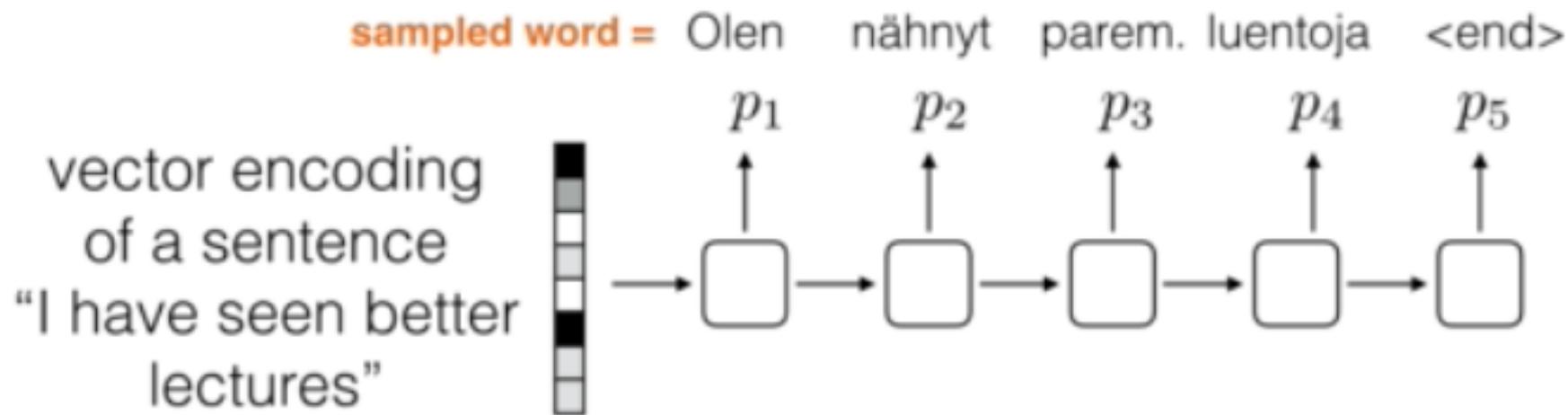
$$h_t = o_t \odot \tanh(c_t) \text{ visible state}$$

$$p_t = \text{softmax}(W^o h_t) \text{ output distribution}$$

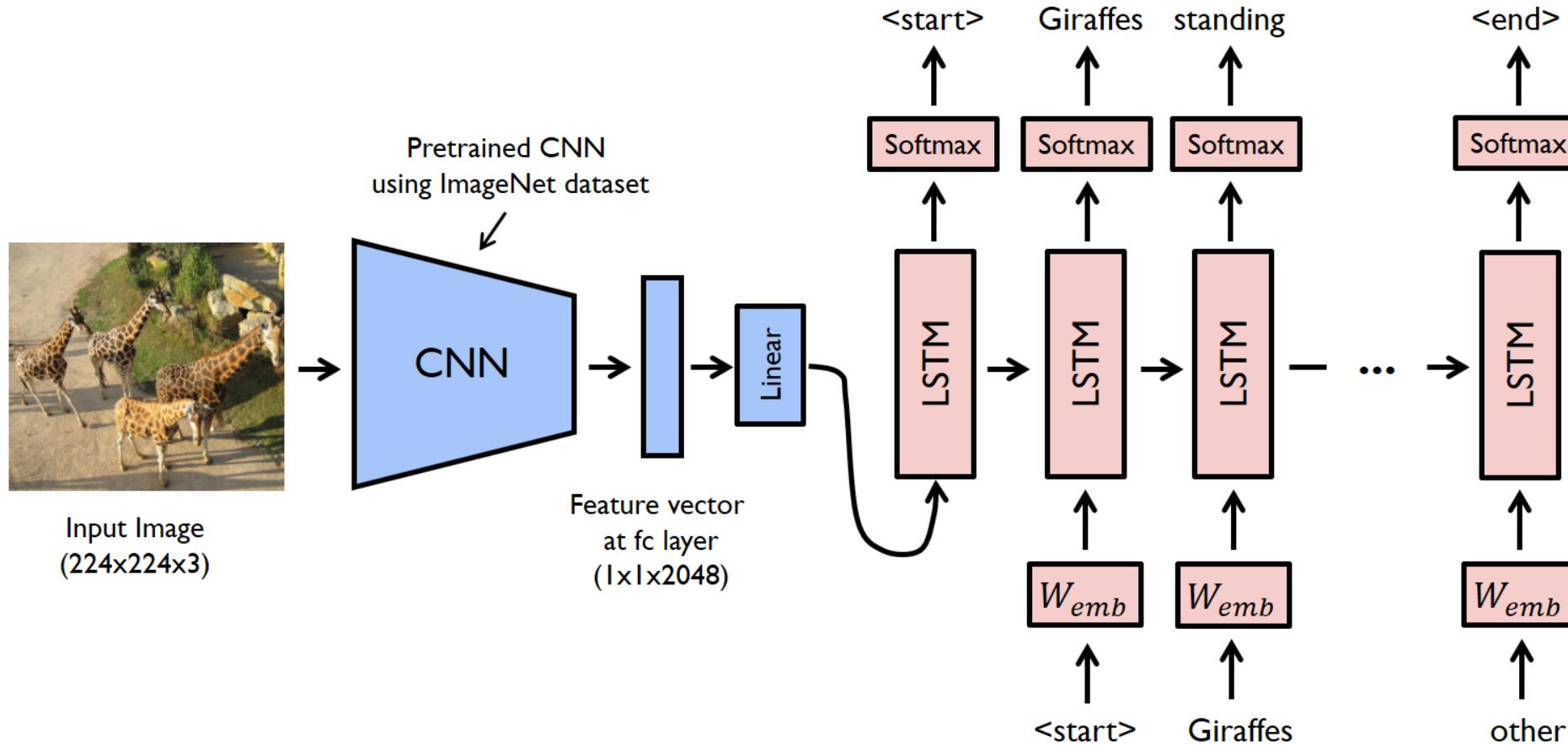
Sequential Models

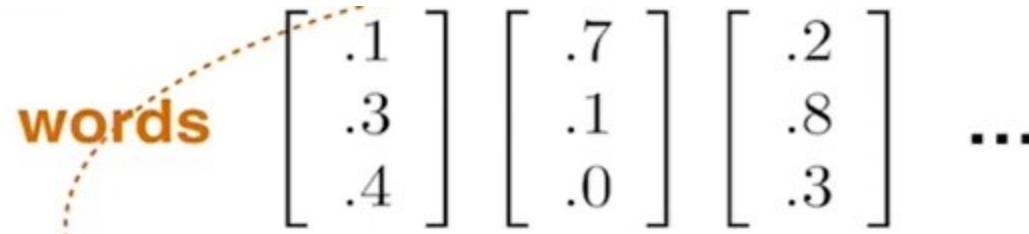
- Until now, our hypotheses have been *pure* functions which map a *single input x* to y .
 - Outputs y depended only on x

Decoding into a sentence



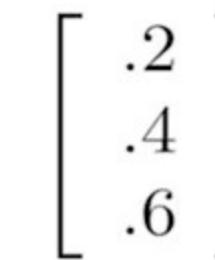
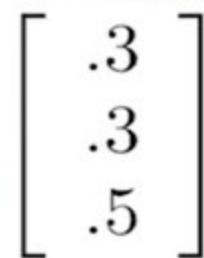
Decoding an image into sentence





“Efforts and courage are not enough without purpose and direction” — JFK

sentences



COGS514 - Cognition and Machine Learning

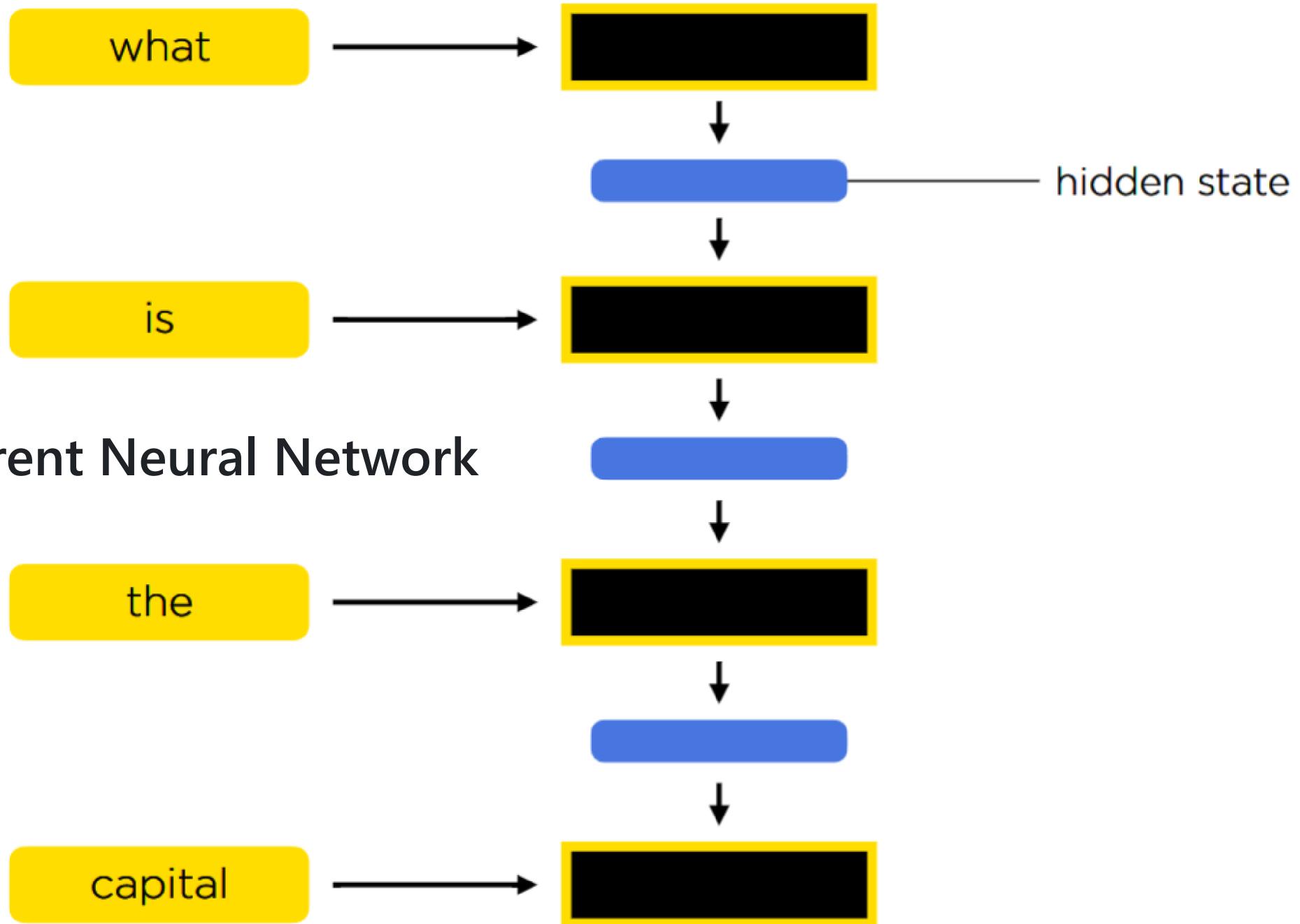
Transformers

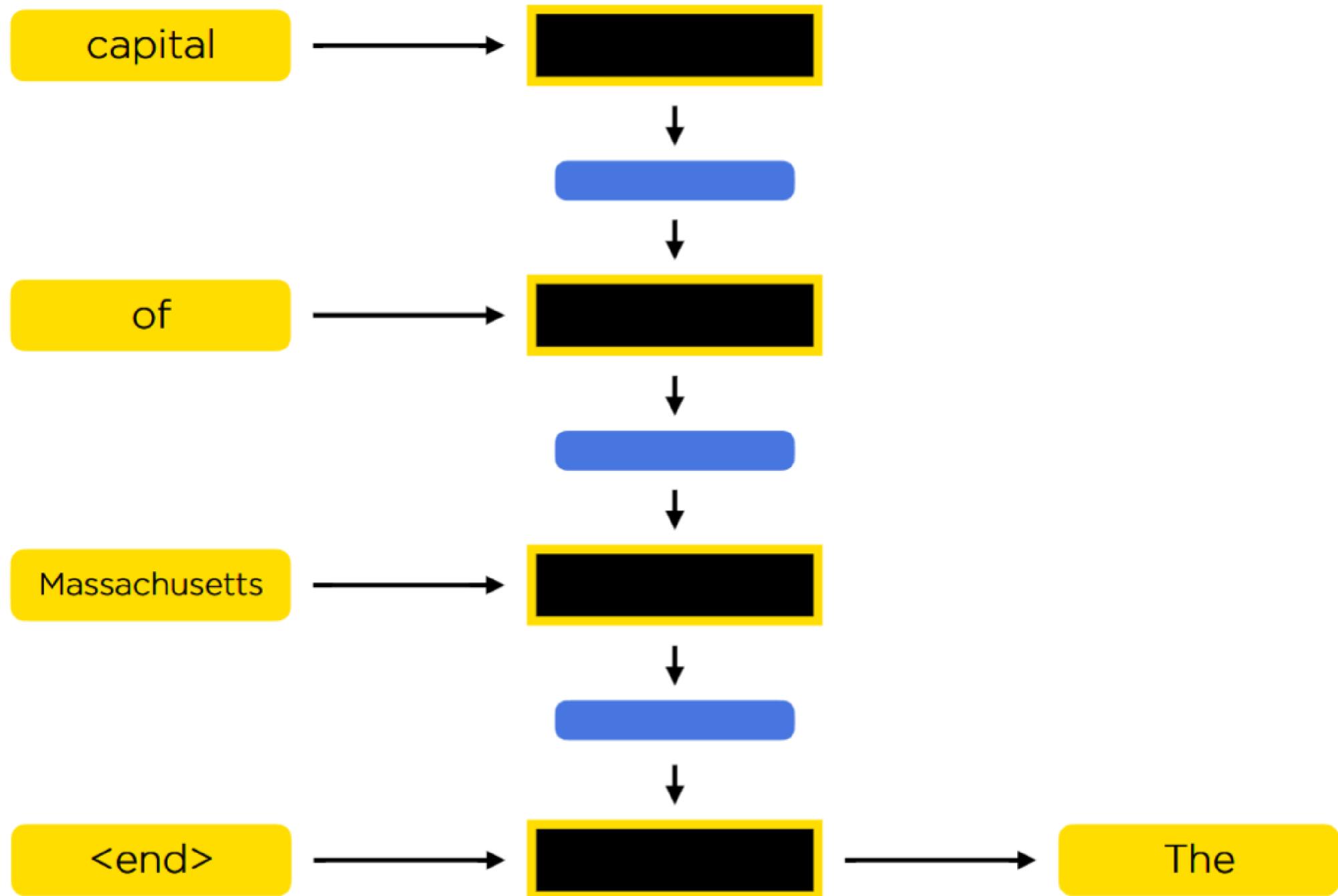
Example - Question Answering

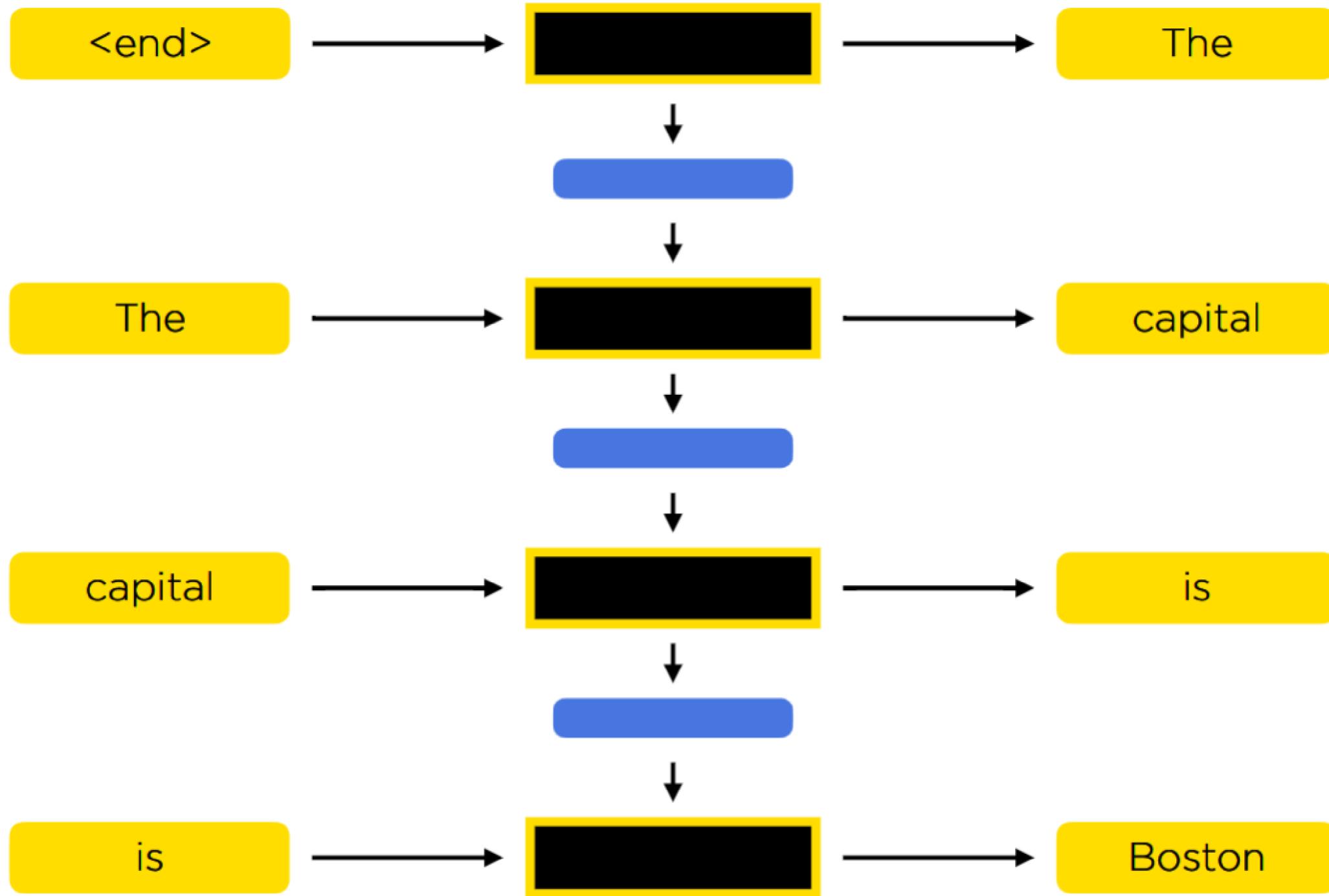
What is the
capital of
Massachusetts?

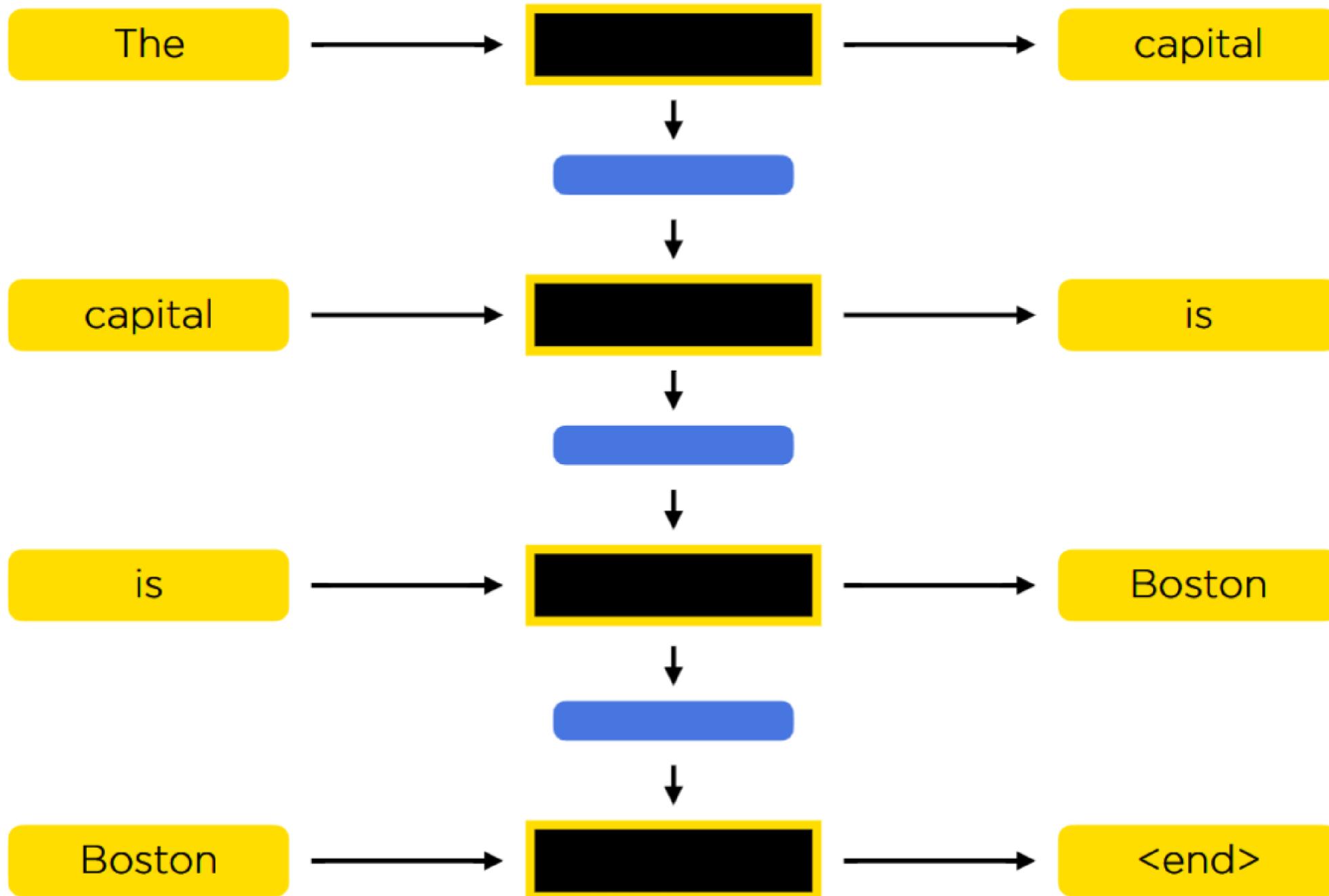


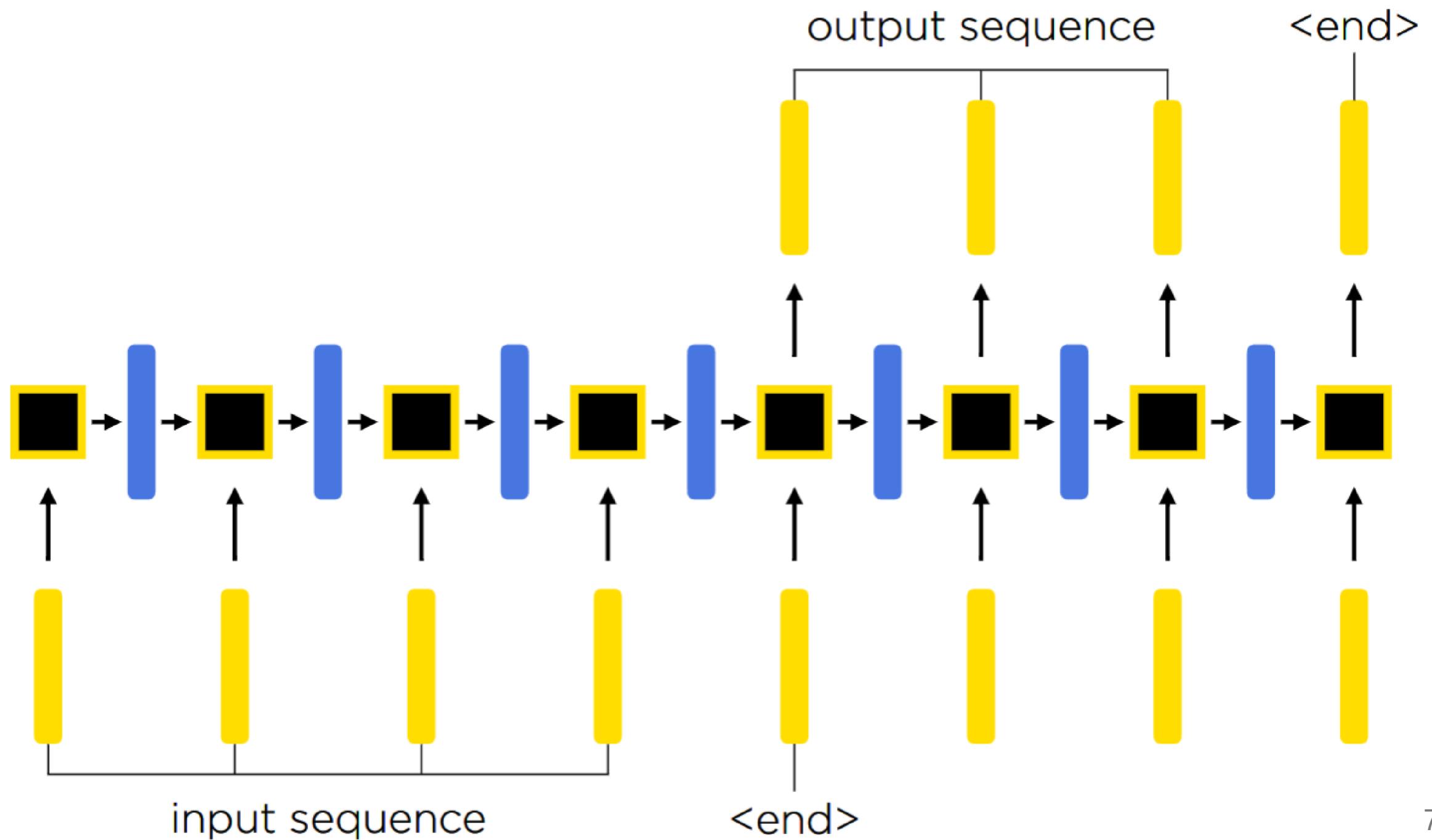
The capital
is Boston.

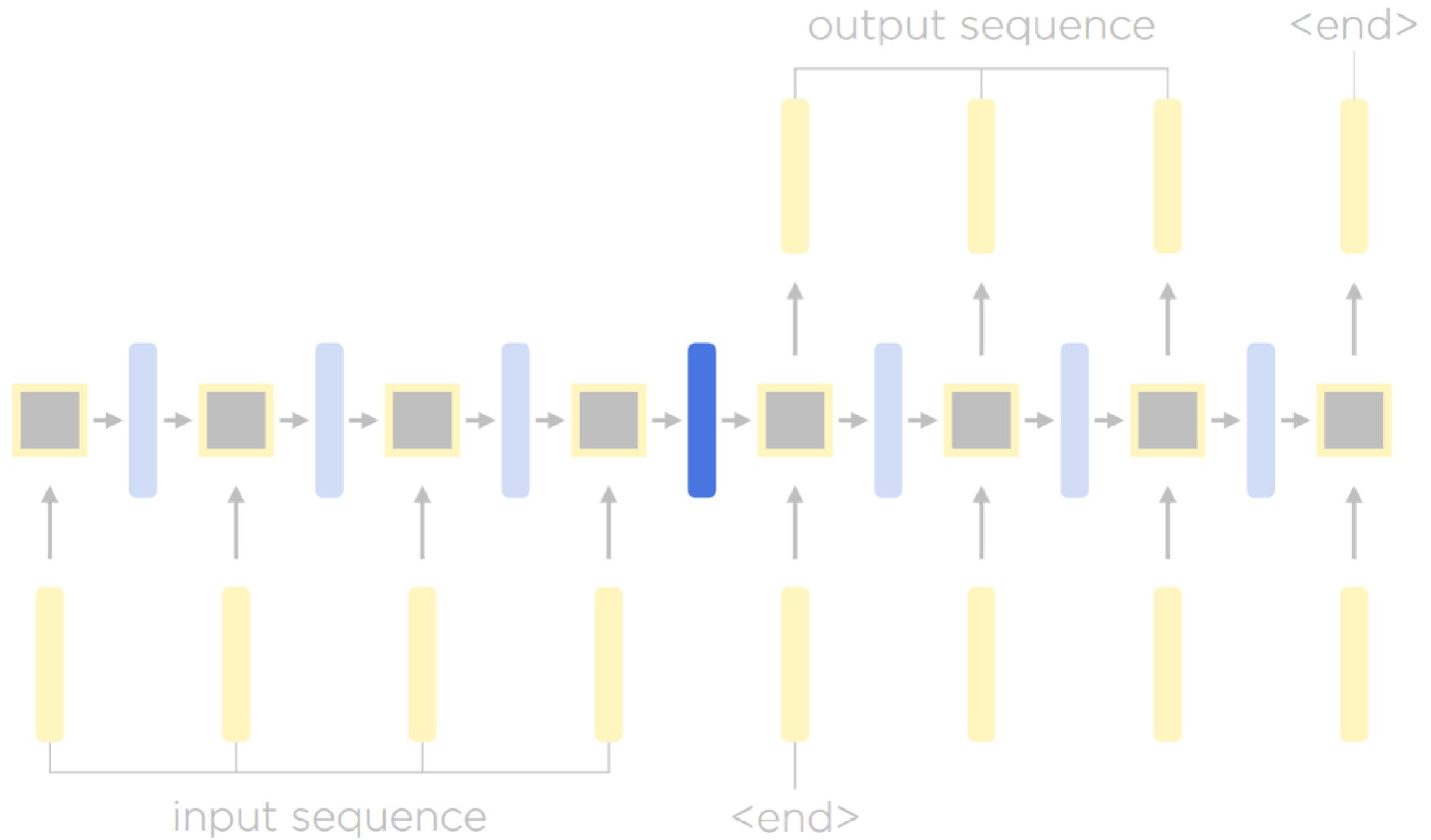


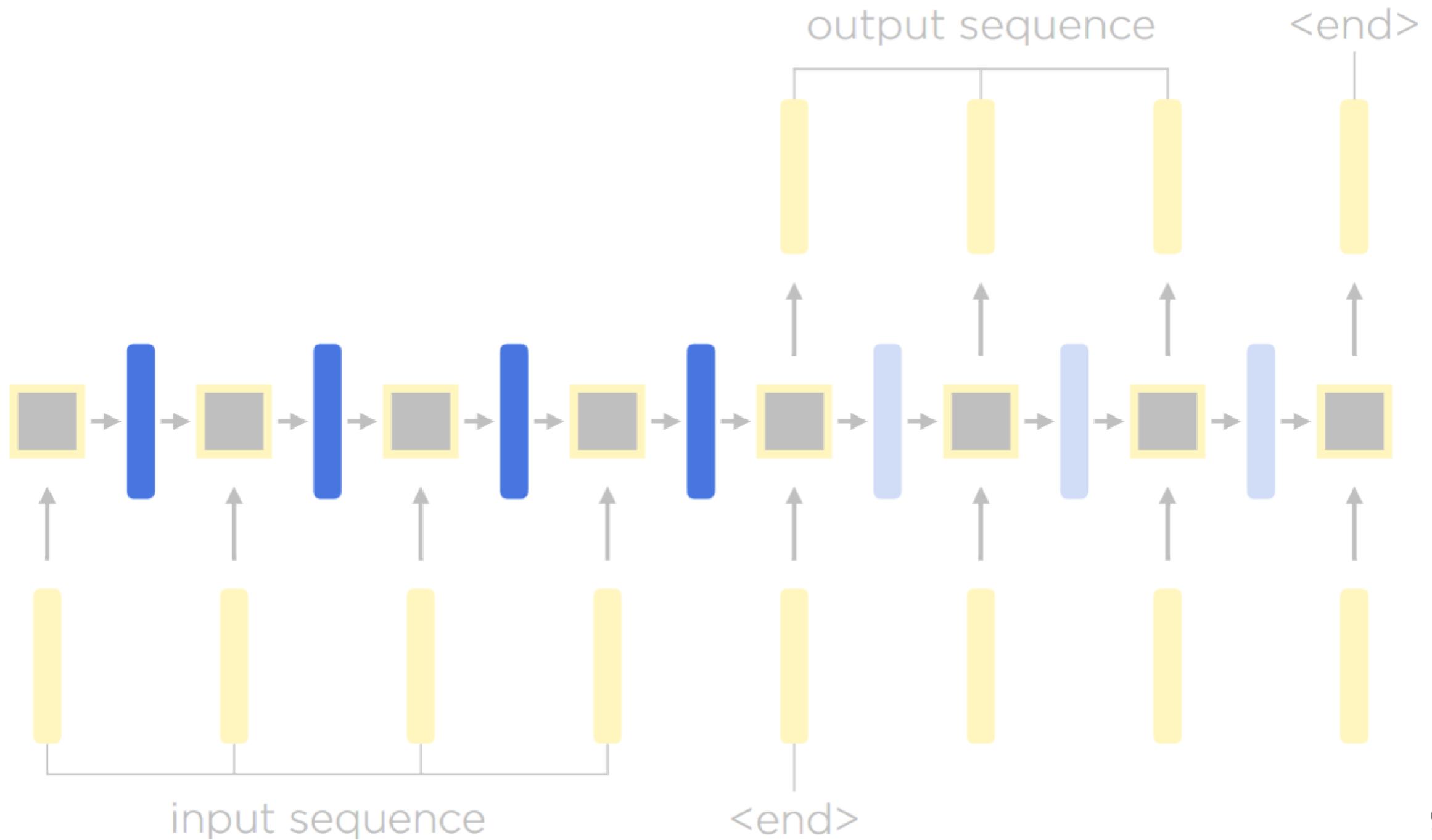












Attention

the

capital

is

what

is

the

capital

of

Massachusetts

the

capital

is

what

is

the

capital

of

Massachusetts

the

capital

is

The diagram illustrates the calculation of a weighted sum of word vectors. It shows a series of blue vertical bars representing vectors, each multiplied by a scalar weight (0.04, 0.02, 0.01, 0.28, 0.03, 0.62) and then summed together to produce a final vector (represented by a blue bar on the right). The words corresponding to these vectors are: what, is, the, capital, of, and Massachusetts. The word 'capital' is highlighted in red, indicating it is the target word being predicted.

$$0.04 \times \text{what} + 0.02 \times \text{is} + 0.01 \times \text{the} + 0.28 \times \text{capital} + 0.03 \times \text{of} + 0.62 \times \text{Massachusetts} = \text{Final Vector}$$

what

is

the

capital

of

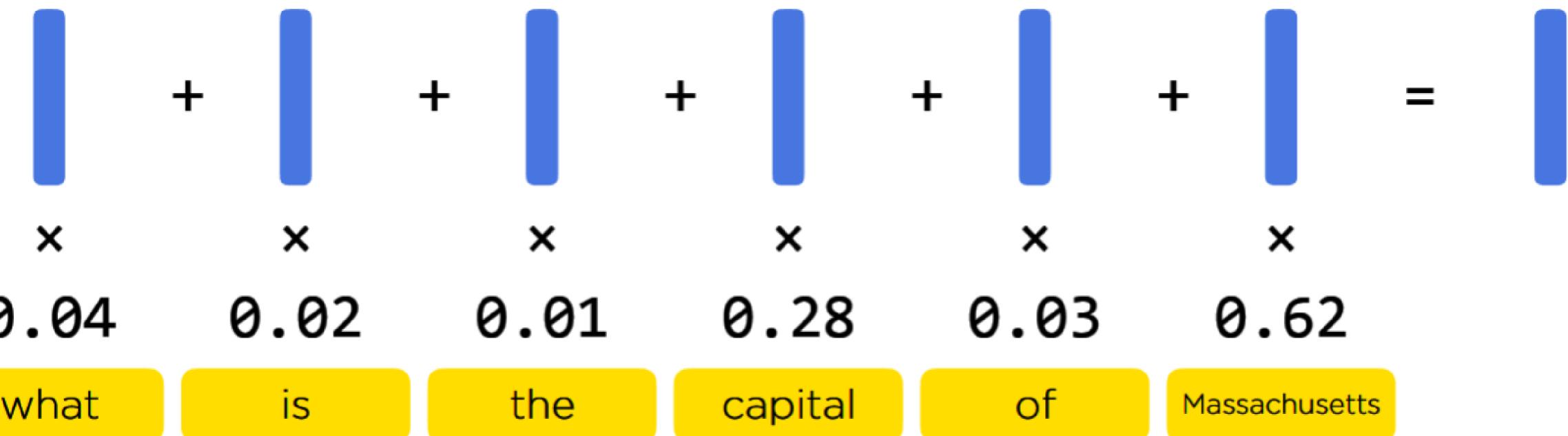
Massachusetts

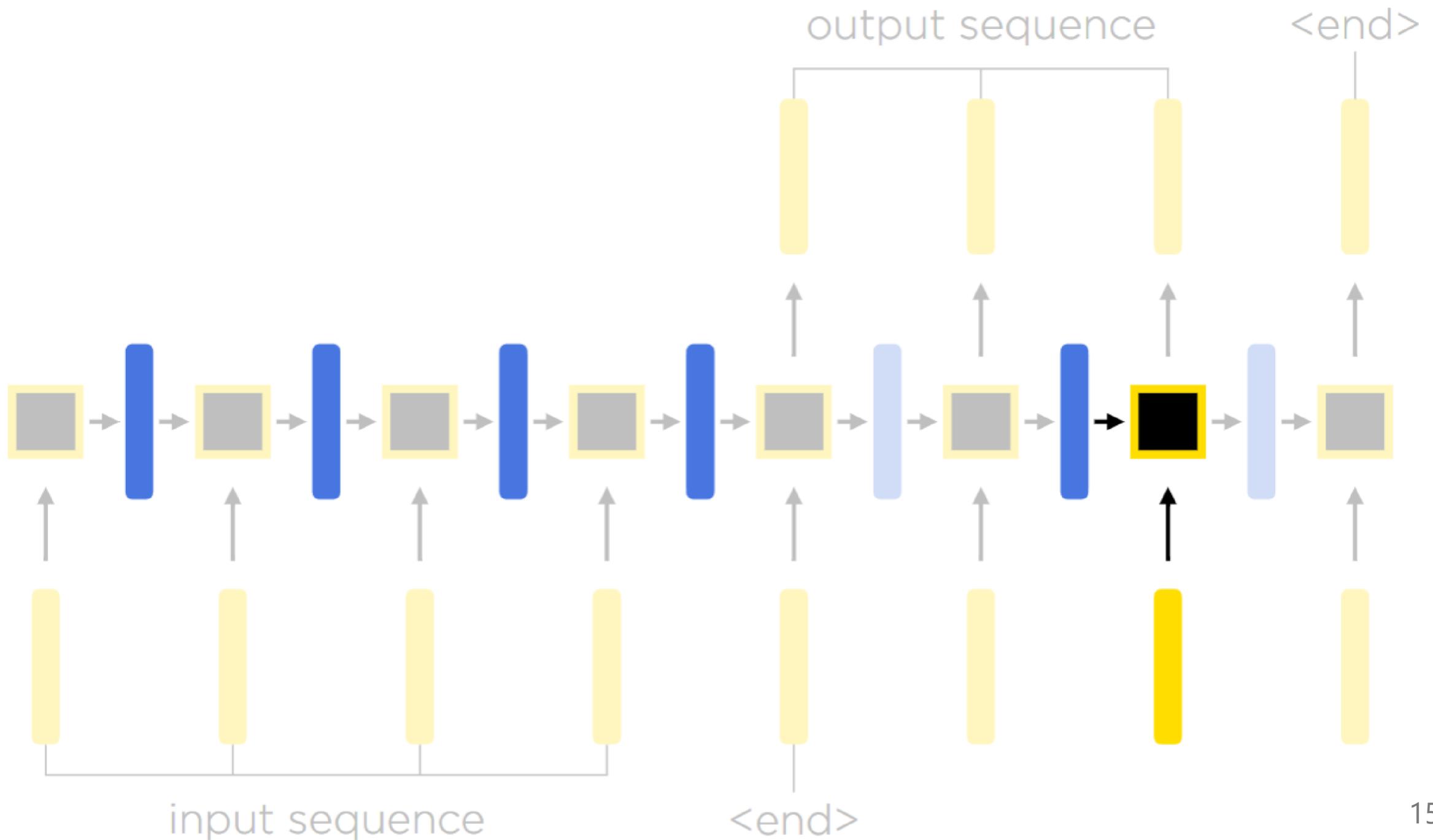
the

capital

is

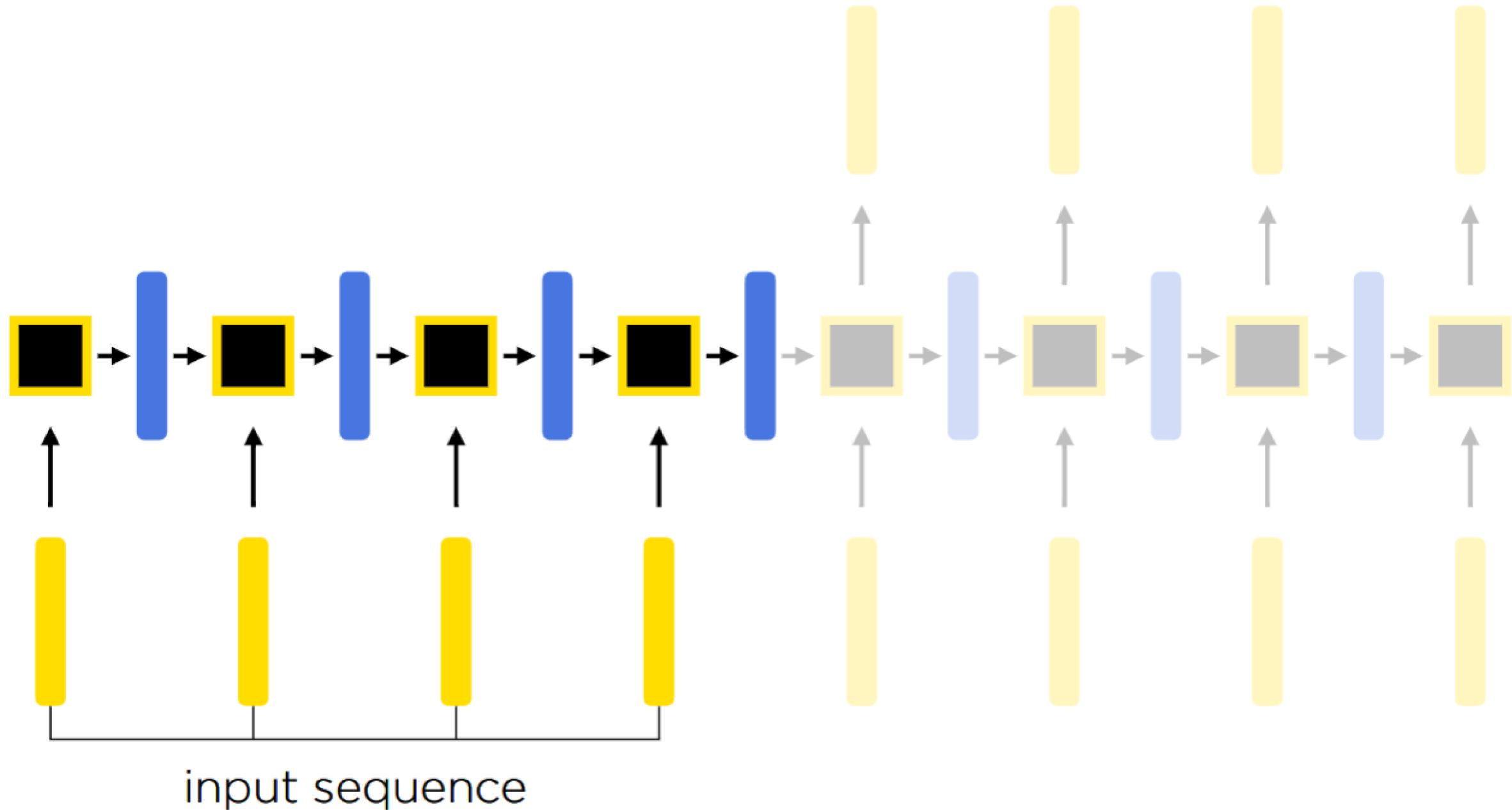
Boston

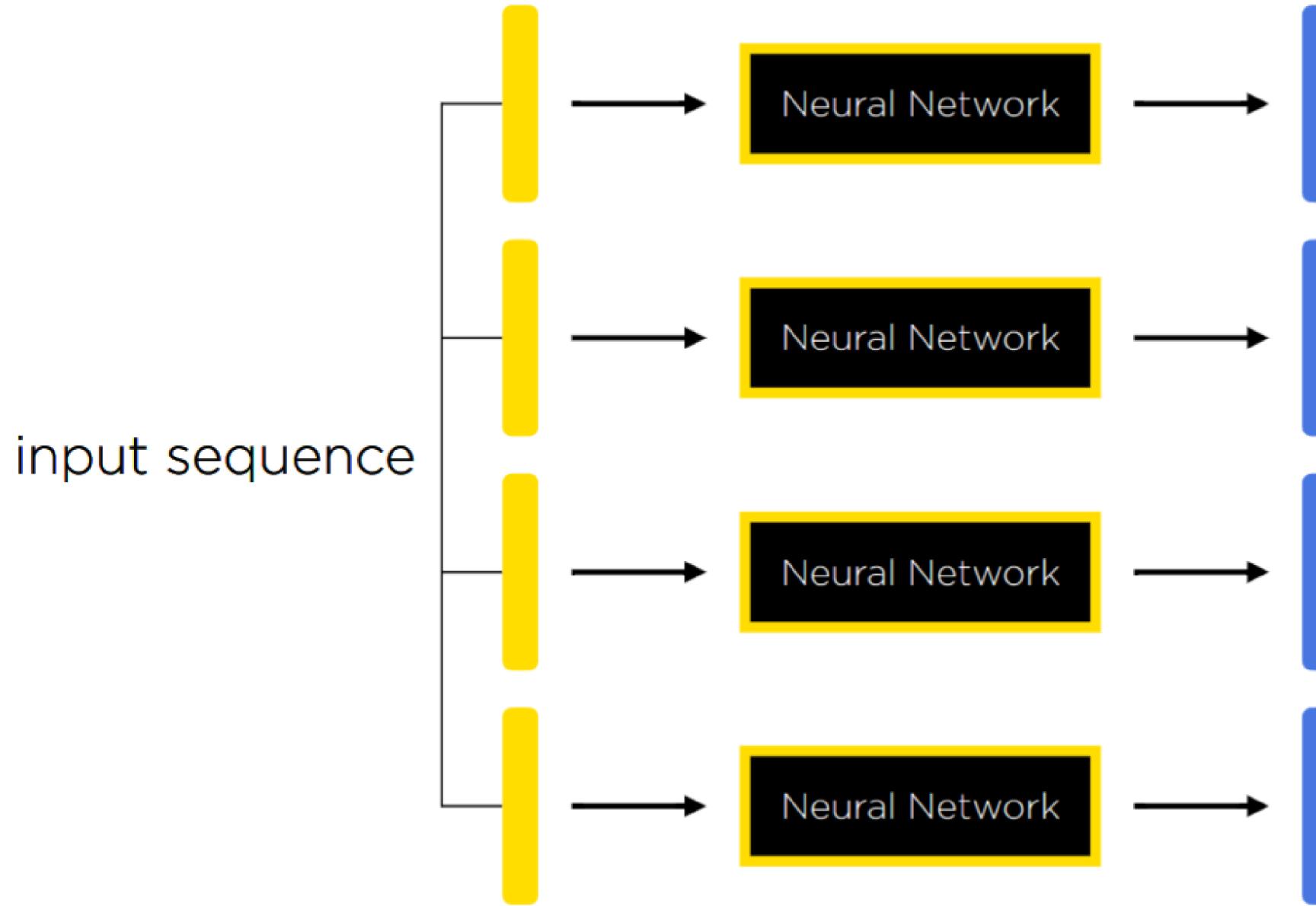




Transformers

1. Encoding



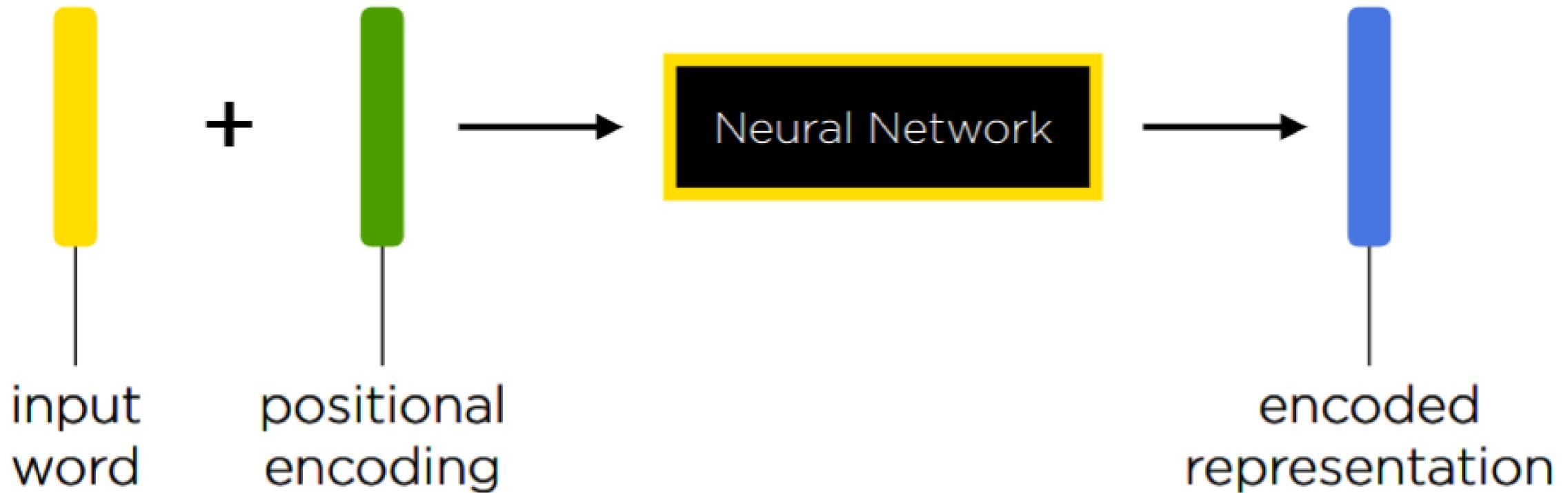


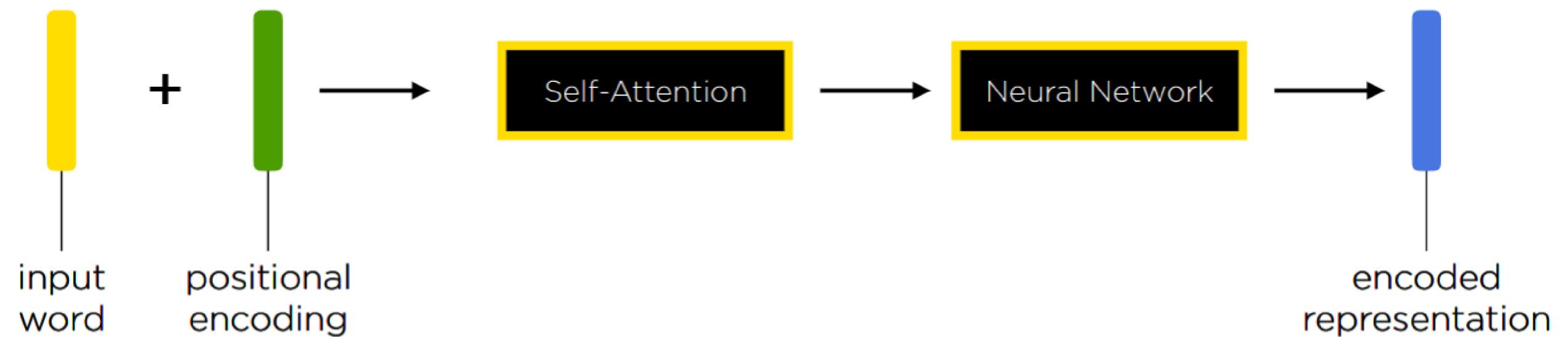


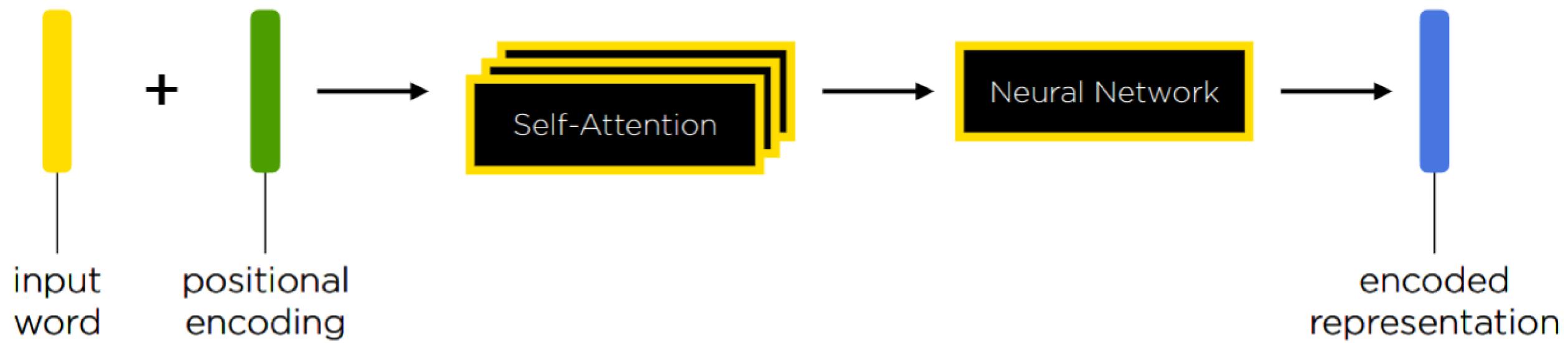
input
word

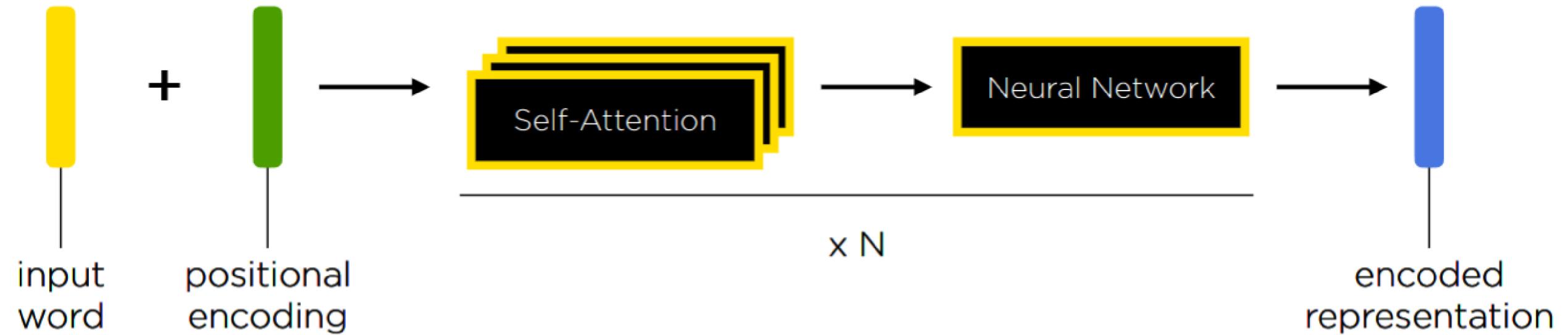


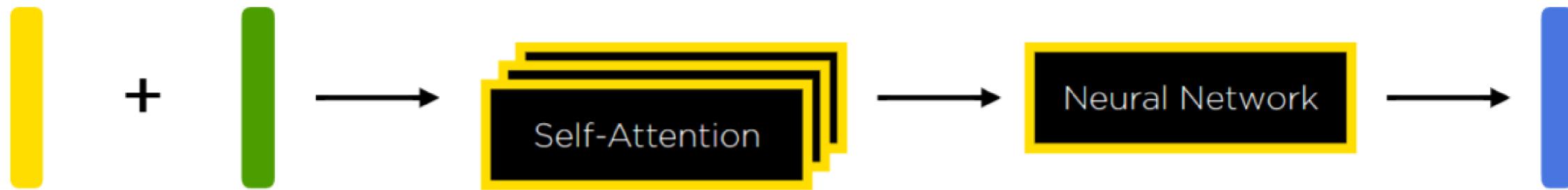
encoded
representation

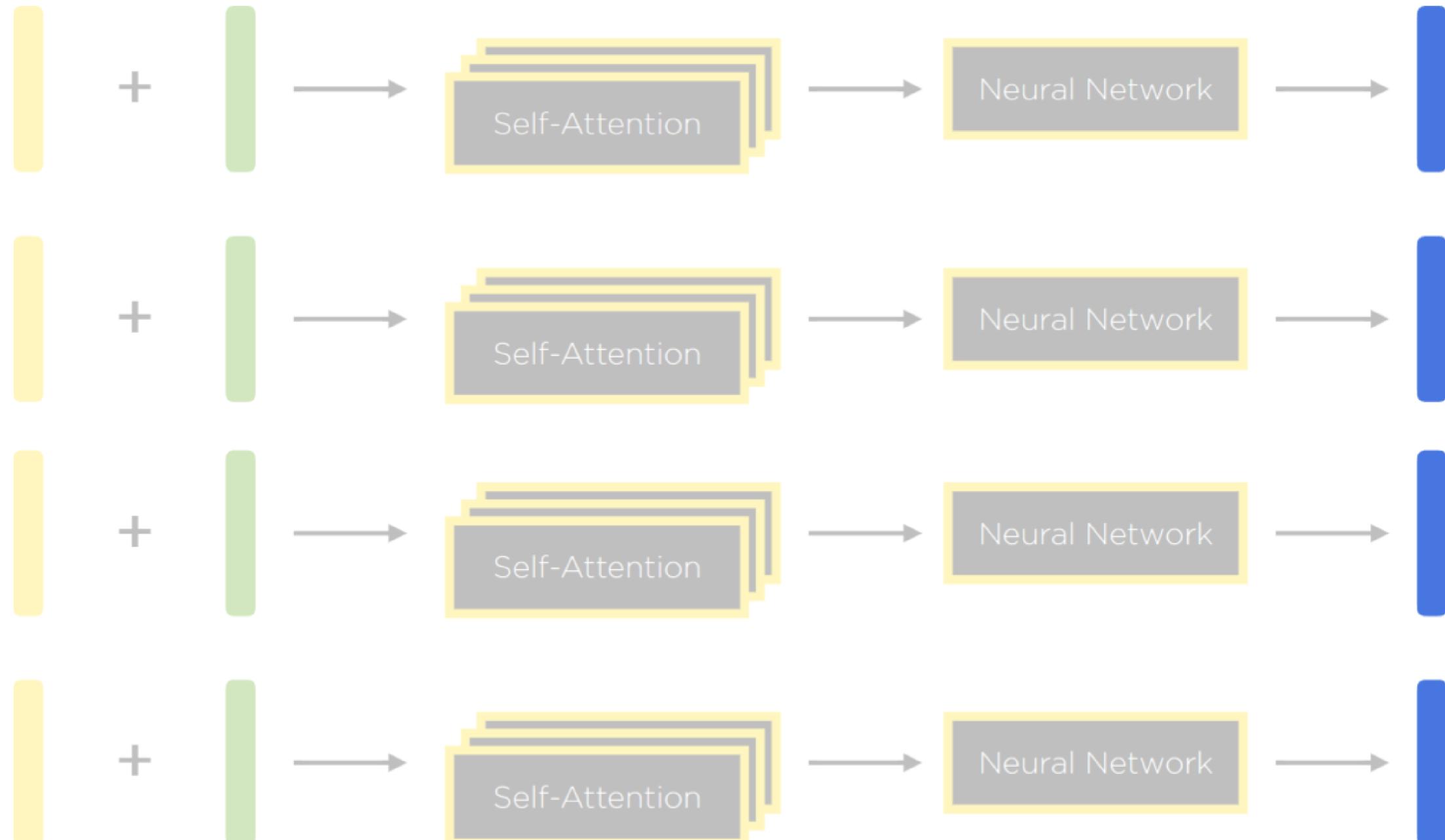






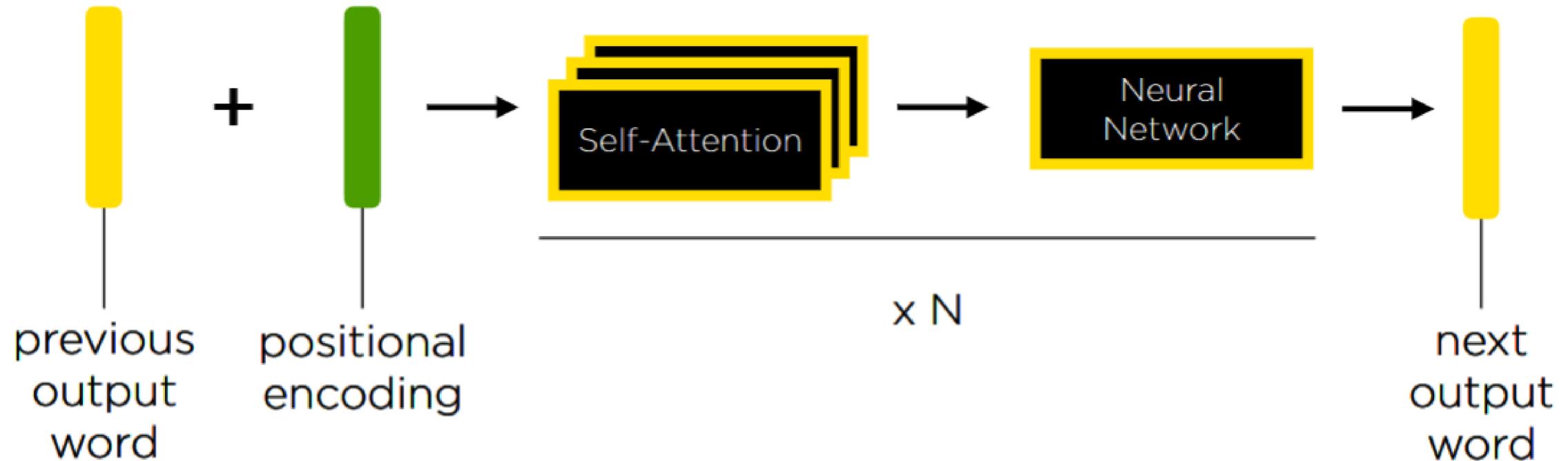


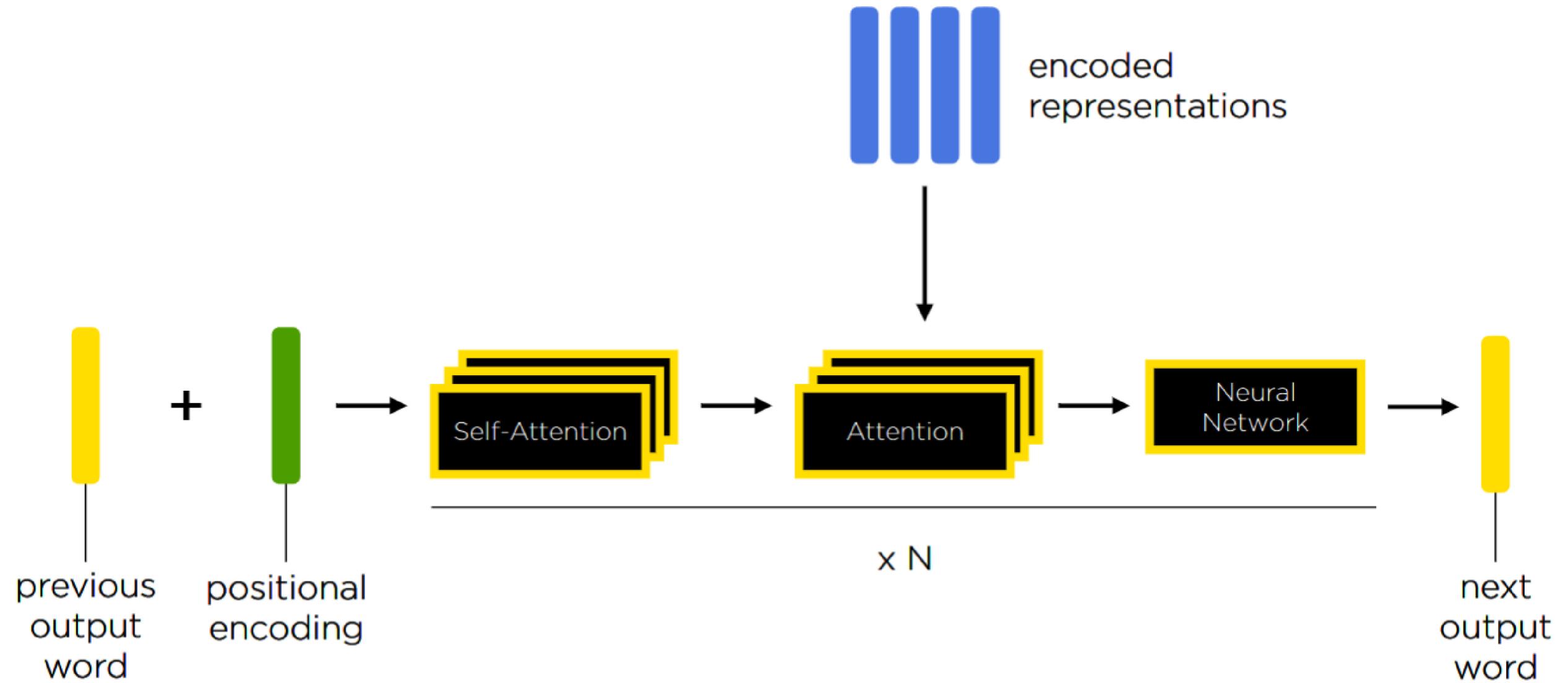


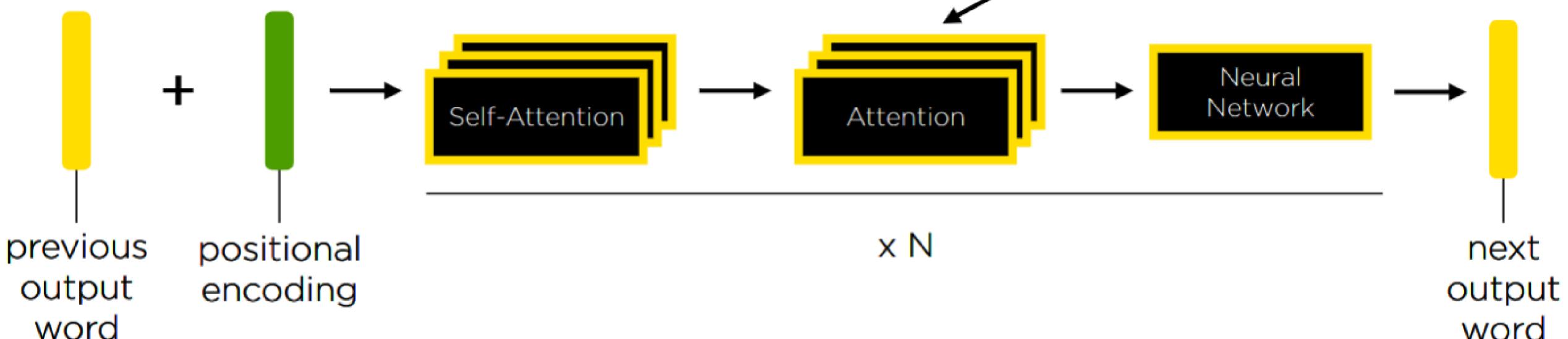
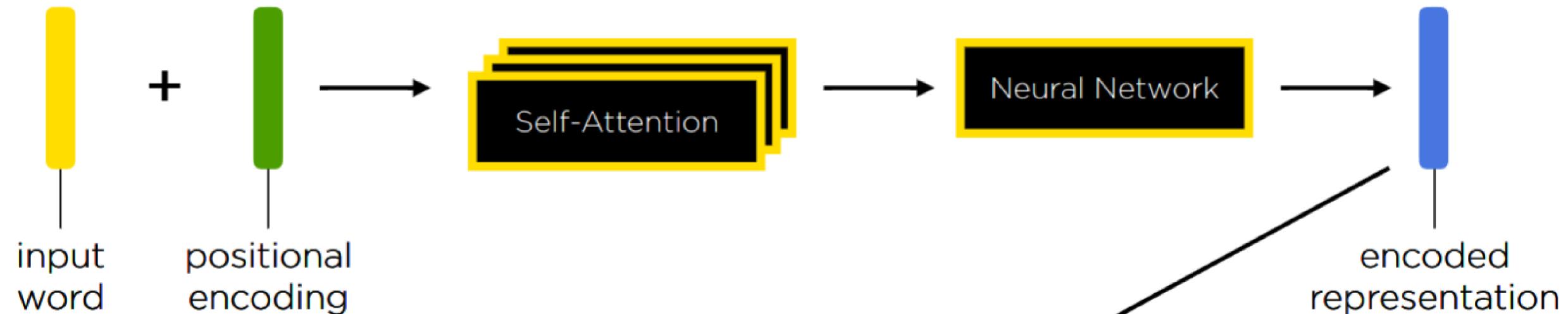


encoded representations

Decoding







COGS514 - Cognition and Machine Learning

Decision Trees and Ensemble Methods

Non-parametric methods

- They have parameters but the number of parameters is not defined in advance
- Parameterization adapts to learning problem

1. **Tree models:** partition the input space and use simple predictors on different regions
2. **Ensemble methods:** train several different classifiers on the whole space and average the answer to decrease the estimation error

Outline

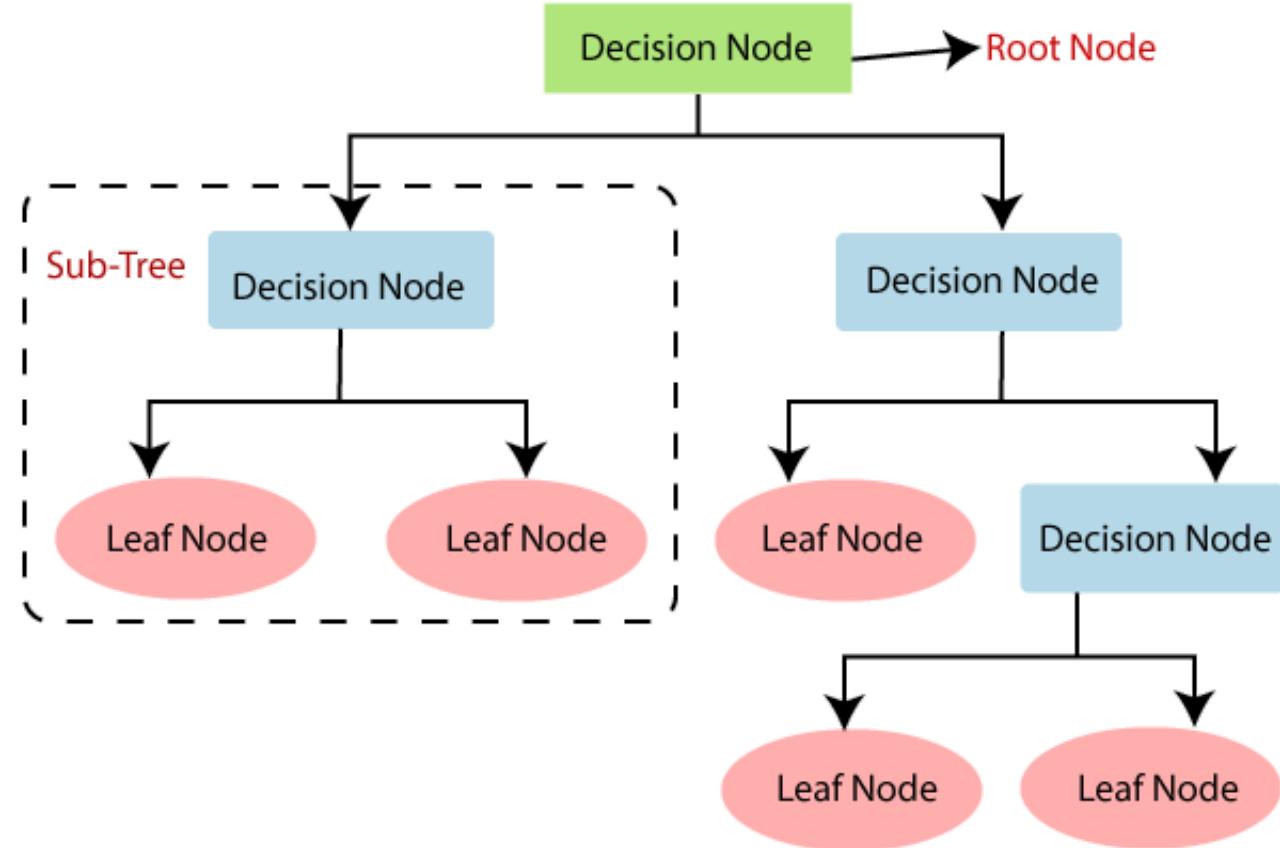
- Decision Trees
- Ensemble Methods
 - Bagging
 - Random Forests
 - Boosting
 - Voting

Decision Trees

Decision Trees

They lost popularity due to recent success of neural networks but they are still useful

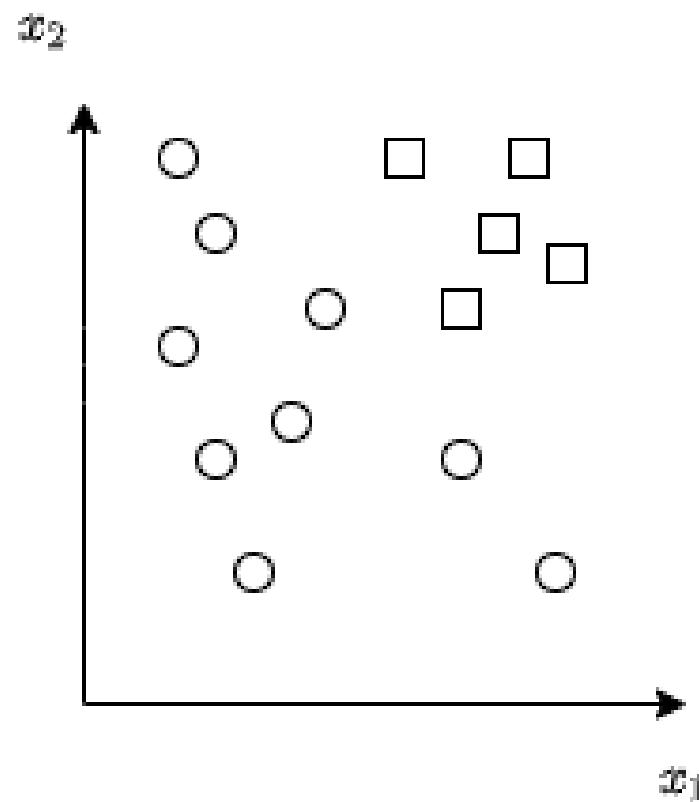
- They usually work well
- They are fast to implement (no hyper-parameters to tune)
- Decision trees can be interpreted by humans



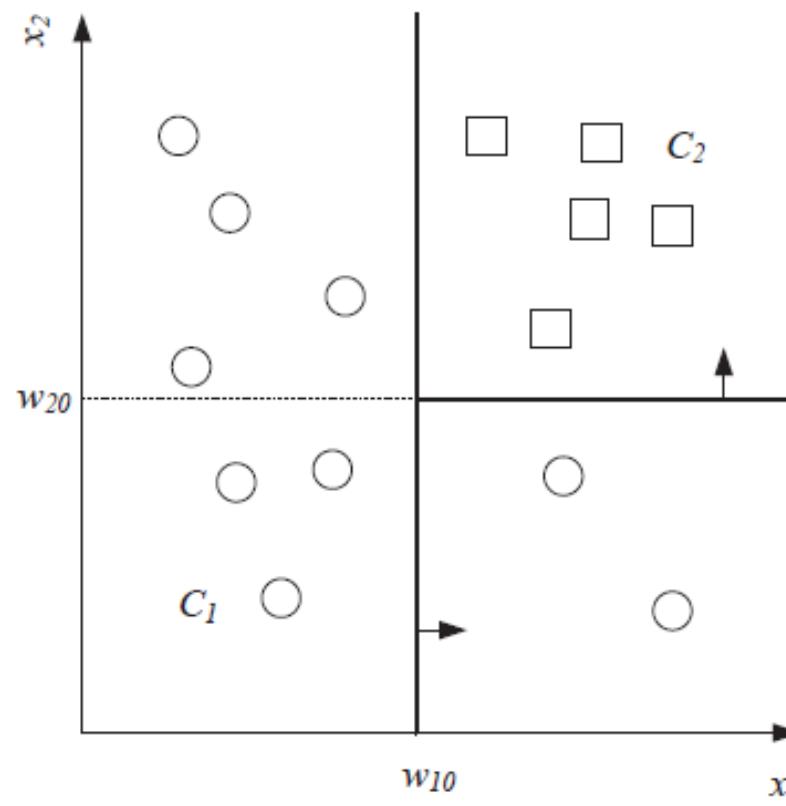
Decision Trees: Basics

- Recursively partition the input space into multiple pieces
- Simple predictors for each partition
- Partition is described by a *decision tree*

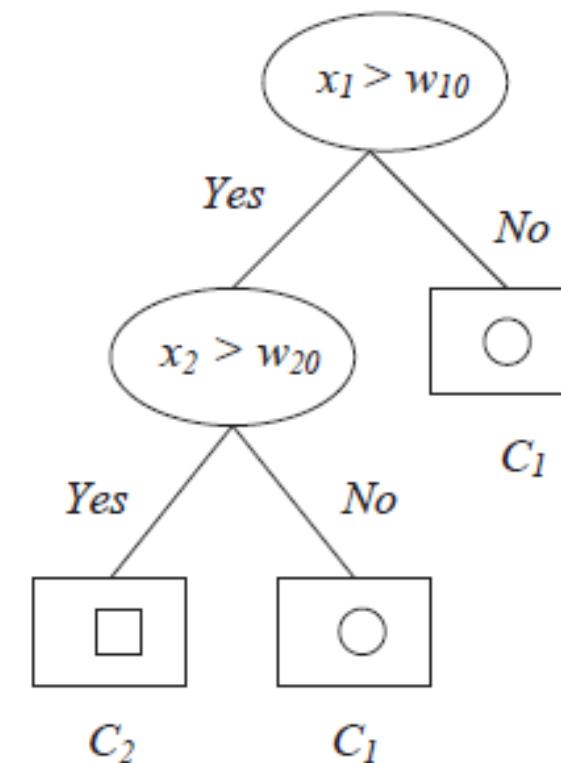
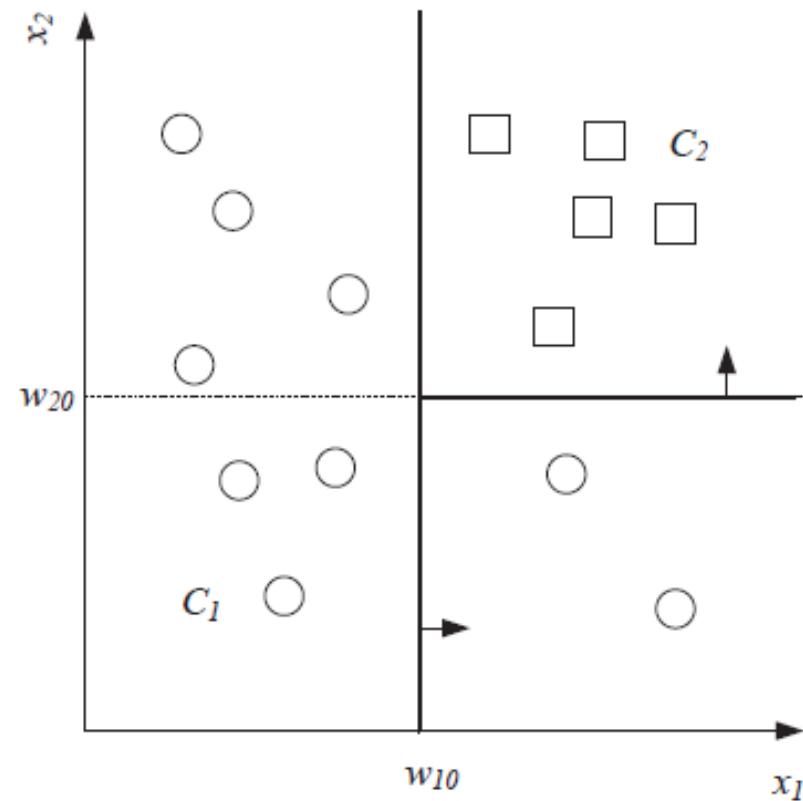
Decision Tree Example Classification



Decision Tree Example Classification



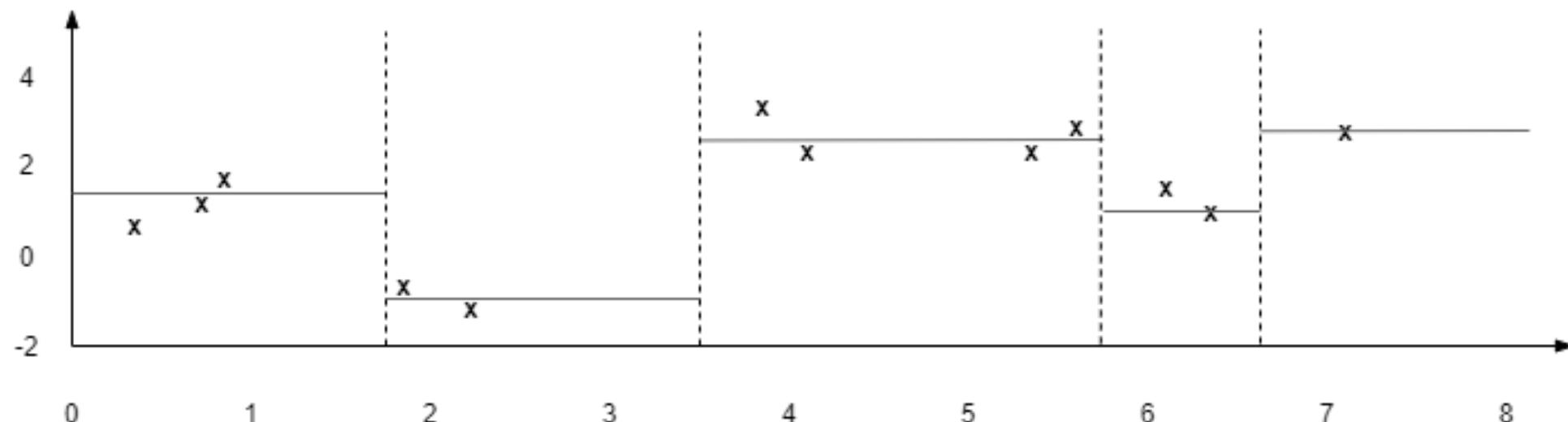
Decision Tree Example Classification

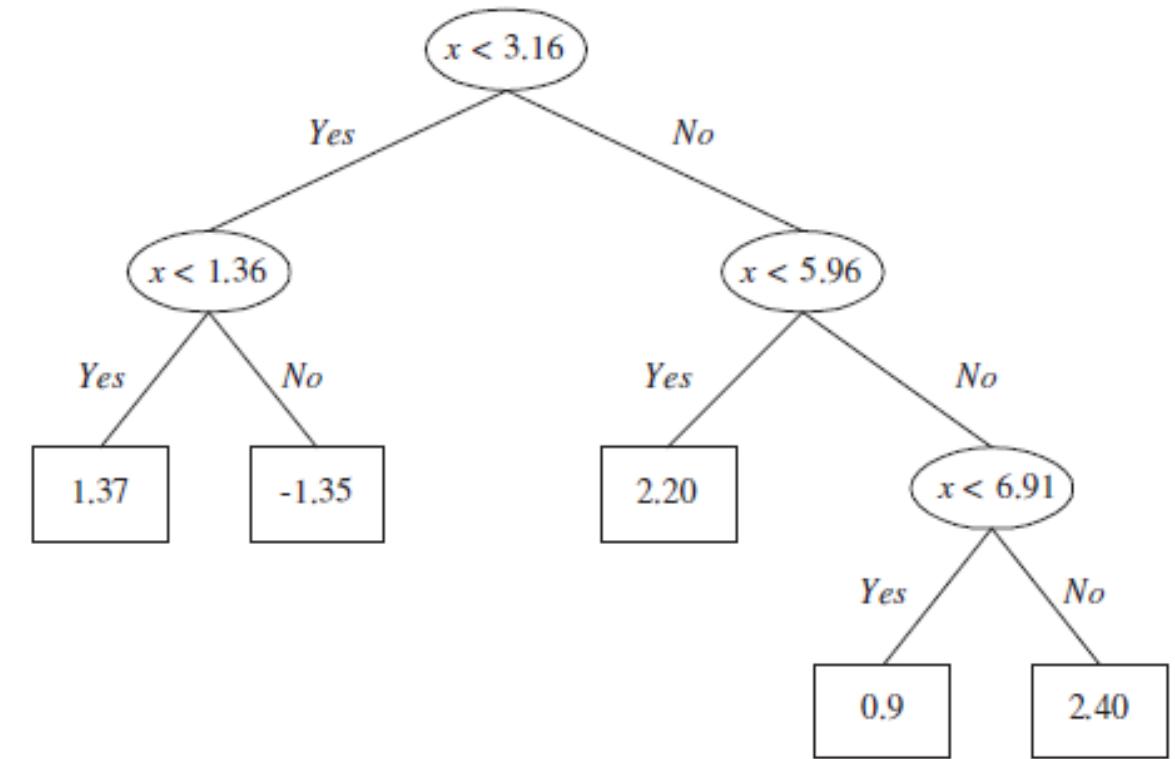
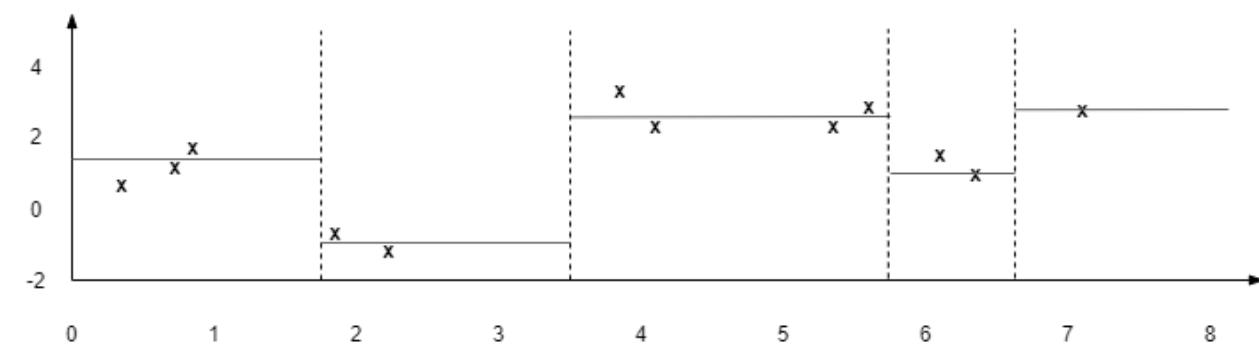


Decision Tree Example Regression



Decision Tree Example Regression





Decision tree methods differ by:

- How they split the space at each node
 - Usually, linear splits are aligned with the axis of the space
 - More general classifiers can be used.
- How they generate predictions
 - Usually, the predictors are simply constants
 - More general regression or classification models can be used
- How we control the complexity of the hypothesis
- Algorithm used for making the partitions and fitting the models.

Decision Trees are appropriate when:

- Input space is not high-dimensional
- Input features have some useful information individually or in small groups
- For example, a set of measurements of a patient in the hospital.

Decision Trees are appropriate when:

- Input space is not high-dimensional
- Input features have some useful information individually or in small groups
- For example, a set of measurements of a patient in the hospital.

Not appropriate for:

- Image input

CART / ID3 Decision Tree Algorithms

We focus on CART/ID3 algorithms which were independently developed by AI and Statistics communities.

- Greedily partitions
- Splits are aligned axes
- Fits a constant model in the leaves
- Similar regression and classification versions

Decision Tree: Regression

The predictor is composed of:

- a partition function mapping elements of the input space into exactly one of M regions R_1, \dots, R_M
- a collection of M predicted output values, one output O_m for each region.

Decision Tree: Regression

The predictor is composed of:

- a partition function mapping elements of the input space into exactly one of M regions R_1, \dots, R_M
- a collection of M predicted output values, one output O_m for each region.
- where, O_m is the constant output for region R_m to be the average of the training examples in that region, i.e.:

$$O_m = \text{average}_{\{i|x^{(i)} \in R_m\}} y^{(i)}$$

Regression error

Error in the region is

$$E_m = \sum_{\{i|x^{(i)} \in R_m\}} (y^{(i)} - O_m)^2$$

Regression error

Error in the region is

$$E_m = \sum_{\{i|x^{(i)} \in R_m\}} (y^{(i)} - O_m)^2$$

We want to select the partition to minimize

$$\sum_{m=1}^M E_m + \lambda M$$

where λ is some regularization constant.

Building a tree

- Greedy approach
- Use a criterion for the best single split of that data
- Apply it recursively to partition the space.
- Select the partition that minimizes **the sum of the mean squared errors** of each partition.

Building a tree: notation

Given a dataset D ,

- $R_{j,s}^+(D) = \{x \in D | x_j \geq s\}$ be the set of examples in dataset D whose value in dimension j is greater than or equal to s

Building a tree: notation

Given a dataset D ,

- $R_{j,s}^+(D) = \{x \in D | x_j \geq s\}$ be the set of examples in dataset D whose value in dimension j is greater than or equal to s
- $R_{j,s}^-(D) = \{x \in D | x_j < s\}$ be the set of examples in dataset D whose value in dimension j is less than s

Building a tree: notation

Given a dataset D ,

- $R_{j,s}^+(D) = \{x \in D | x_j \geq s\}$ be the set of examples in dataset D whose value in dimension j is greater than or equal to s
- $R_{j,s}^-(D) = \{x \in D | x_j < s\}$ be the set of examples in dataset D whose value in dimension j is less than s
- $\hat{y}_{j,s}^+ = \text{average}_{\{i | x^{(i)} \in R_{j,s}^+\}} y^{(i)}$ be the average y value of the data points in set $R_{j,s}^+(D)$

Building a tree: notation

Given a dataset D ,

- $R_{j,s}^+(D) = \{x \in D | x_j \geq s\}$ be the set of examples in dataset D whose value in dimension j is greater than or equal to s
- $R_{j,s}^-(D) = \{x \in D | x_j < s\}$ be the set of examples in dataset D whose value in dimension j is less than s
- $\hat{y}_{j,s}^+ = \text{average}_{\{i | x^{(i)} \in R_{j,s}^+\}} y^{(i)}$ be the average y value of the data points in set $R_{j,s}^+(D)$
- $\hat{y}_{j,s}^- = \text{average}_{\{i | x^{(i)} \in R_{j,s}^-\}} y^{(i)}$ be the average y value of the data points in set $R_{j,s}^-(D)$

Building a tree: algorithm

BuildTree(D, k):

- if $|D| \leq k$,
 - return Leaf(D)
- find the dimension j and split point s that minimizes

$$E_{R_{j,s}^-(D)} + E_{R_{j,s}^+(D)}$$

- return Node($j, s, \text{BuildTree}(R_{j,s}^-(D, k)), \text{BuildTree}(R_{j,s}^+(D, k))$)

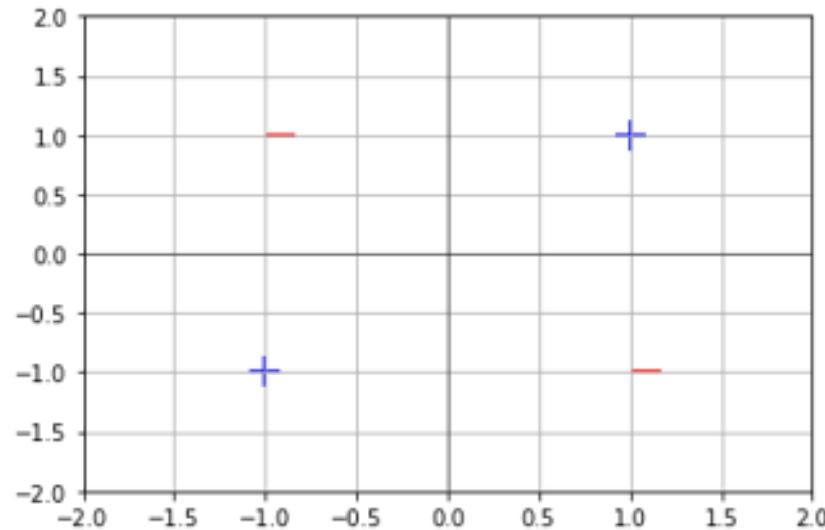
Pruning Decision Trees

Pruning

- You should **not** use a **short-sighted stopping rule** such as:
 - using a large value of k or stopping when splitting does not decrease the error
- Sometimes, there may not be improvement in a single split, but a split after several steps can provide a high error decrease.
- Build a large tree and then prune it back.

Pruning

- Example: XOR in two dimensions.
 - Training error does not change with single split.



Pruning

- The cost complexity of a tree T where m ranges over its leaves is

$$C_\alpha(T) = \sum_{m=1}^{|T|} E_m(T) + \alpha|T|$$

- For a fixed α we can find a T that minimizes $C_\alpha(T)$ by the "weakest-link" pruning.

Pruning

- Cost of complexity of a tree $C_\alpha(T) = \sum_{m=1}^{|T|} E_m(T) + \alpha|T|$
- For a fixed α we can find a T that minimizes $C_\alpha(T)$ by the "weakest-link" pruning.
 - Create a sequence of trees by successively removing the bottom-level split that minimizes the increase in overall error, until the root is reached.
 - Return the T in the sequence that minimizes the criterion.

Classification - prediction

- Classification trees are built in a similar way to regression trees.
- In a region R_m corresponding to the leaf of the tree, we would pick the output class y to be the value that exists the most frequently in the data points whose x values are in that region.

$$O_m = \text{majority}_{\{i|x^{(i)} \in R_m\}} y^{(i)}$$

Classification - error

- Define the error in a region as the number of data points that do not have the value O_m .

$$E_m = \left| \{i | x^{(i)} \in R_m \text{ and } y^{(i)} \neq O_m\} \right|$$

Classification - empirical probabilities

- Empirical probability of an item from class k occurring in region m :

$$\hat{P}_{mk} = \hat{P}(R_m)(k) = \frac{|\{i|x^{(i)} \in R_m \text{ and } y^{(i)} = k\}|}{N_m}$$

where N_m is the number of training points in region m .

Empirical probabilities of split values

- Empirical probability of split value v in dimension j occurring in region m :

$$\hat{P}_{mjv} = \hat{P}(R_{mj})(v) = \frac{\left| \{i | x^{(i)} \in R_m \text{ and } x_j^{(i)} \geq v\} \right|}{N_m}$$

where N_m is the number of training points in region m .

Impurity Measures

To decide which split to make next. There are several criteria to measure *impurity* in child nodes to decide which split to make next

Impurity Measures

To decide which split to make next. There are several criteria to measure *impurity* in child nodes to decide which split to make next

- Entropy

$$Q_m(T) = H(R_m) = - \sum_k \hat{P}_{mk} \log_2 \hat{P}_{mk}$$

Impurity Measures

To decide which split to make next. There are several criteria to measure *impurity* in child nodes to decide which split to make next

- Entropy

$$Q_m(T) = H(R_m) = - \sum_k \hat{P}_{mk} \log_2 \hat{P}_{mk}$$

- Gini index

$$Q_m(T) = \sum_k \hat{P}_{mk} (1 - \hat{P}_{mk})$$

Impurity Measures

To decide which split to make next. There are several criteria to measure *impurity* in child nodes to decide which split to make next

- Entropy

$$Q_m(T) = H(R_m) = - \sum_k \hat{P}_{mk} \log_2 \hat{P}_{mk}$$

- Gini index

$$Q_m(T) = \sum_k \hat{P}_{mk} (1 - \hat{P}_{mk})$$

- Misclassification error

$$Q_m(T) = \frac{E_m}{N_m} = 1 - \hat{P}_{mO_m}$$

Impurity Measures

- Entropy

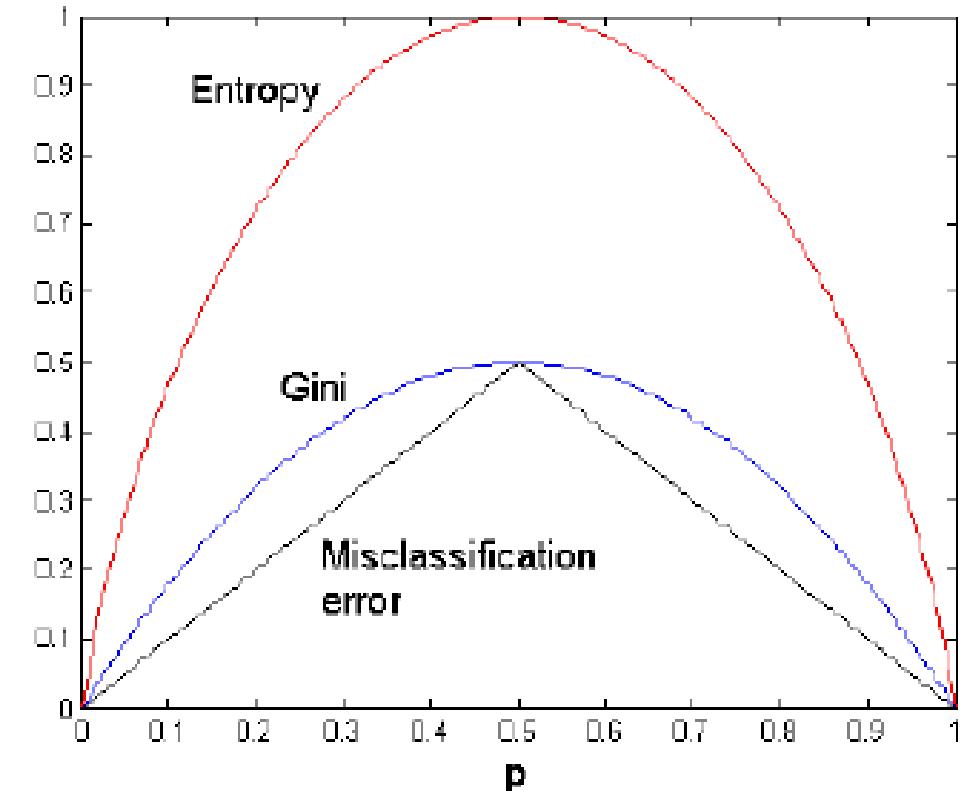
$$Q_m(T) = - \sum_k \hat{P}_{mk} \log_2 \hat{P}_{mk}$$

- Gini index

$$Q_m(T) = \sum_k \hat{P}_{mk} (1 - \hat{P}_{mk})$$

- Misclassification error

$$Q_m(T) = \frac{E_m}{N_m} = 1 - \hat{P}_{mO_m}$$



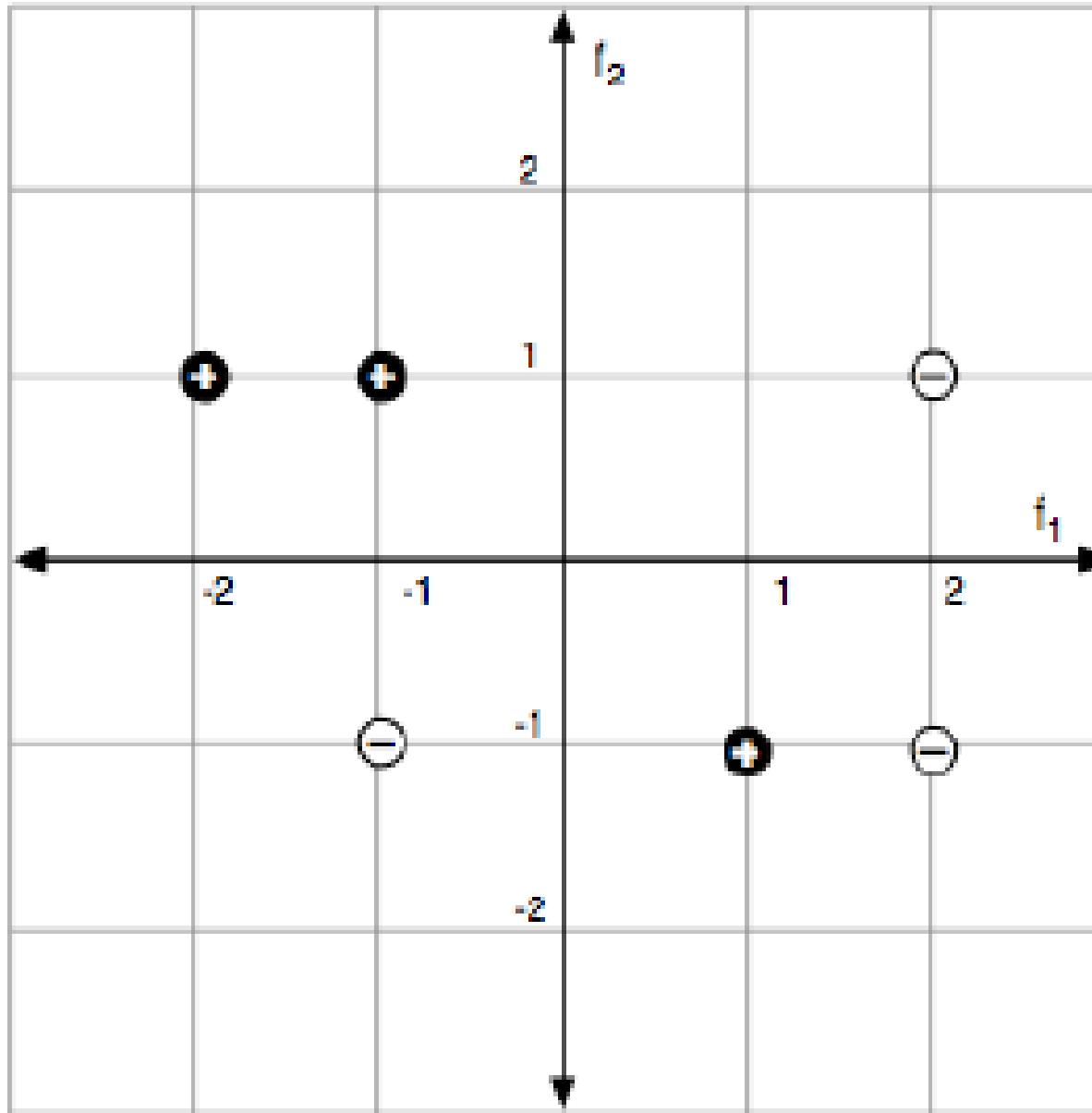
Using Entropy as an impurity measure

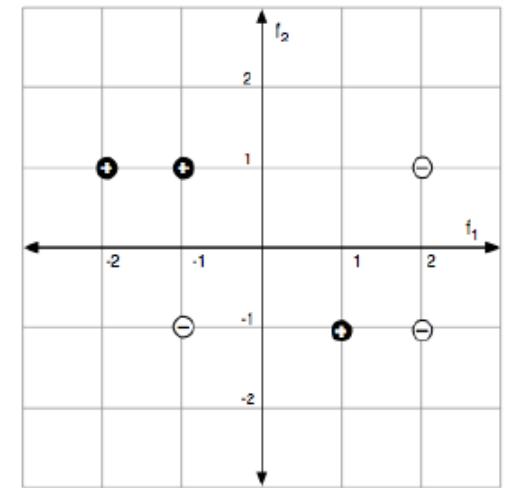
- Choosing the split v that minimizes the entropy is equivalent to maximize the information gain of the test $X_j = v$

$$\text{infoGain}(X_j = v, R_m) = H(R_m) - \left(\hat{P}_{mjv} H(R_{j,v}^+) + (1 - \hat{P}_{mjv}) H(R_{j,v}^-) \right)$$

- Entropy is zero when $\begin{cases} 0 & \text{when } \hat{P}_{m0} = 0 \\ 0 & \text{when } \hat{P}_{m0} = 1 \end{cases}$

Example





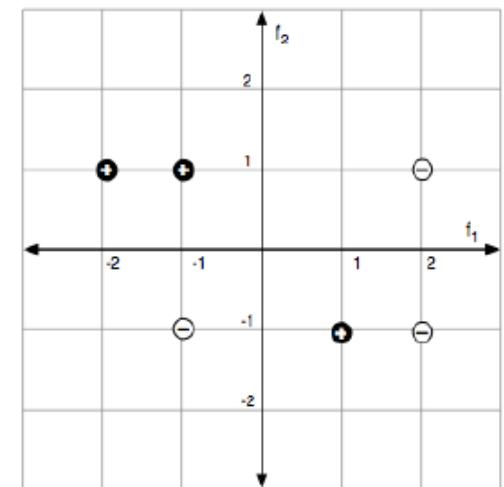
Example

Lets split at 1.5

Entropy without split is

$$H(R_1) = -(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

Example



Lets split at 1.5

Entropy without split is

$$H(R_1) = -(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

Entropy with split is on the r.h.s is

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

Example

Lets split at 1.5

Entropy without split is

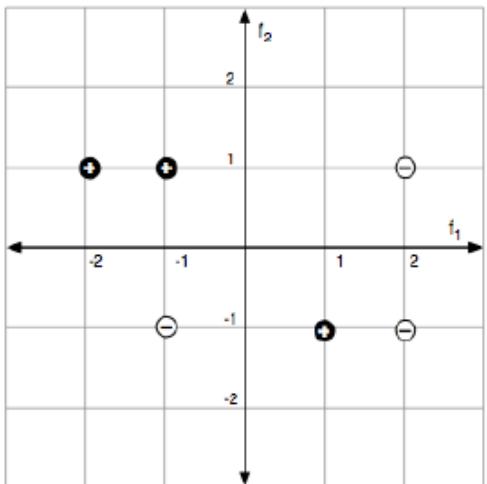
$$H(R_1) = -(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

Entropy with split is on the r.h.s is

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

on the l.h.s is

$$-(0.75 \log_2(0.75) + 0.25 \log_2(0.25)) = 0.81$$



Example

Lets split at 1.5

Entropy without split is

$$H(R_1) = -(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

Entropy with split is on the r.h.s is

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

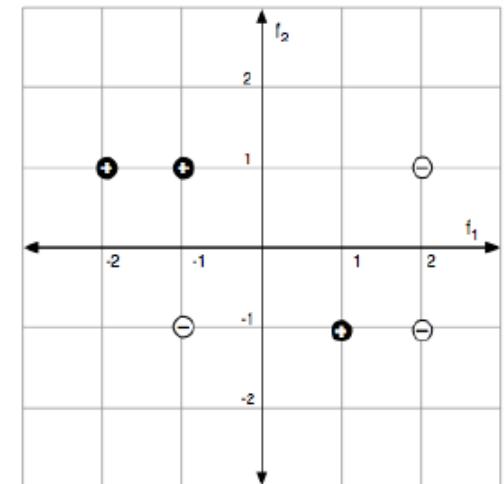
on the l.h.s is

$$-(0.75 \log_2(0.75) + 0.25 \log_2(0.25)) = 0.81$$

Entropy after split is

$$\frac{2}{3} \times 0.81 + \frac{1}{3} \times 0 = 0.54$$

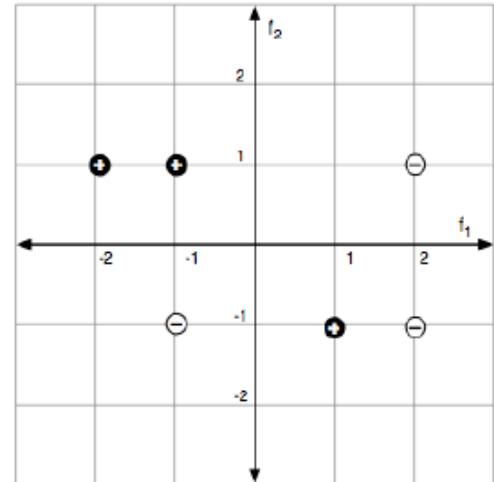
Information gain is $1 - 0.54 = 0.46$



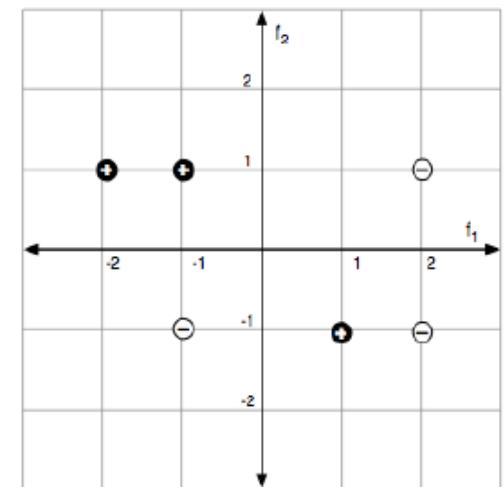
Example

Now lets focus on the region on the left after splitting $f_2 > 1.5$

If we split at $f_1 > 0$



Example



Now lets focus on the region on the left

If we split at $f_1 > 0$

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

$$-(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

$$\text{Information gain is } 0.81 - (0.5 \times 0 + 0.5 \times 1) = 0.31$$

Example

Now lets focus on the region on the left

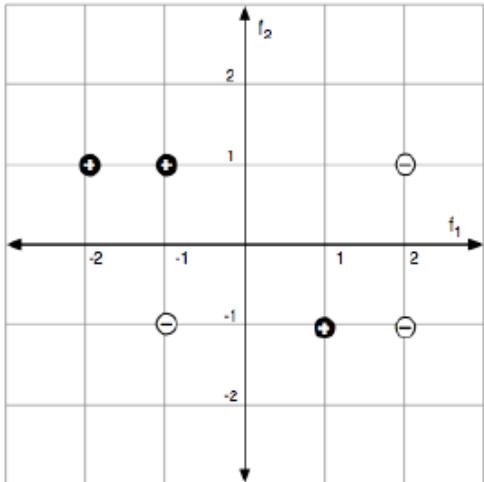
If we split at $f_1 > 0$

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

$$-(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

Information gain is $0.81 - (0.5 \times 0 + 0.5 \times 1) = 0.31$

Alternatively we could split at $f_2 > 0$



Example

Now lets focus on the region on the left

If we split at $f_1 > 0$

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

$$-(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

Information gain is $0.81 - (0.5 \times 0 + 0.5 \times 1) = 0.31$

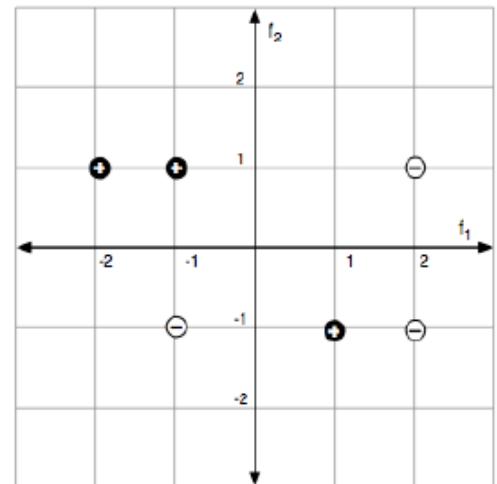
Alternatively we could split at $f_2 > 0$

In this case

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

$$-(\frac{2}{3} \log_2(\frac{2}{3}) + \frac{1}{3} \log_2(\frac{1}{3})) = 0.92$$

Information gain is $0.81 - (0.25 \times 0 + 0.75 \times 0.92) = 0.12$



Example

Now lets focus on the region on the left

If we split at $f_1 > 0$

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

$$-(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

Information gain is $0.81 - (0.5 \times 0 + 0.5 \times 1) = 0.31$

Alternatively we could split at $f_2 > 0$

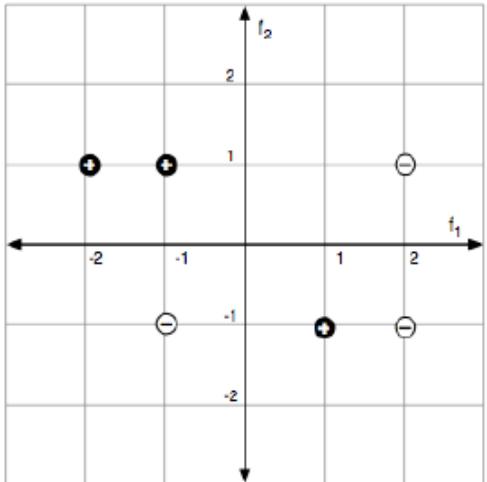
In this case

$$-(0 \log_2(0) + 1 \log_2(1)) = 0$$

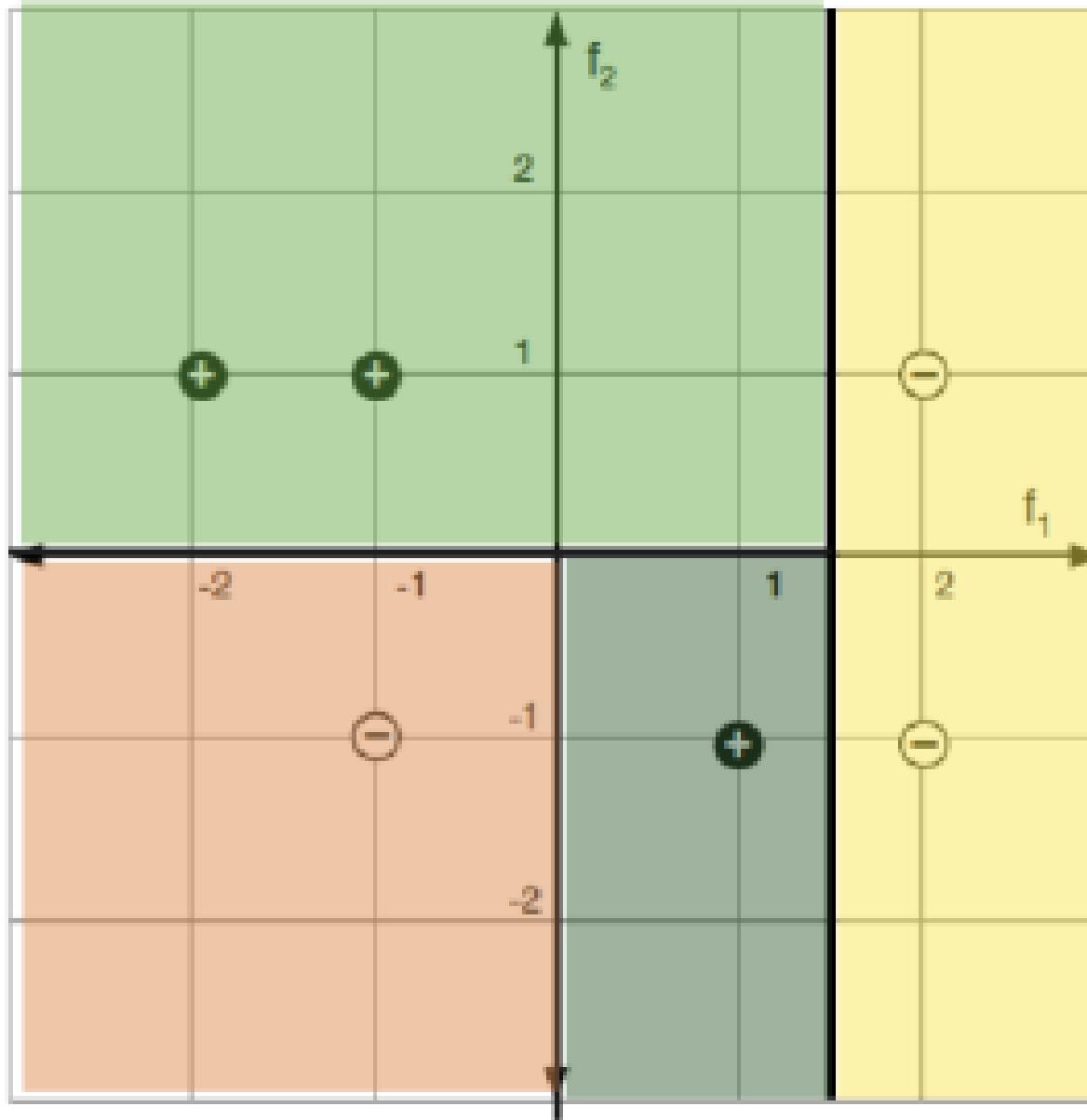
$$-(\frac{2}{3} \log_2(\frac{2}{3}) + \frac{1}{3} \log_2(\frac{1}{3})) = 0.92$$

Information gain is $0.81 - (0.25 \times 0 + 0.75 \times 0.92) = 0.12$

Since information gain is higher we split at $f_1 > 0$



Example



Which impurity measure to use?

- Entropy? Gini index? Misclassification error?
- Similar measures, not obvious which one is better
- It is conventional to use
 - Entropy to select which node to split to grow the tree
 - Misclassification error to prune the tree.

Decision Trees Advantages

- Fast to train
- Easy to interpret
- Easy to handle multi-class classification
- Easy to handle different loss functions (just change the predictor in the leaves)

Decision Trees Disadvantages

- High estimation error
 - Small changes in the data can lead to large changes in the hypothesis
- Usually not the best predictor

Ensemble Methods

Bagging (Bootstrap aggregation)

Bootstrap aggregation (or bagging) is a technique for reducing the estimation error of a non-linear predictor, or one that is adaptive to the data.



Bagging - regression

1. Construct B new data sets of size n by sampling with replacement from D .
2. Train a predictor for each one \hat{f}^b .
3. Use bagged predictor: $\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$

Bagging - Classification

3. Use majority bagged predictor: Let $\hat{f}^b(x)$ be a "one-hot" vector with a single 1 and $K - 1$ zeros, so that $\hat{y}^b(x) = \arg \max_k \hat{f}^b(x)_k$. Then

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

The predicted output is

$$\hat{y}_{bag}(x) = \arg \max_k \hat{f}_{bag}(x)_k$$

Random Forests

- Ensemble of decision trees, trained using the bagging method
- Adds an extra element of randomness by searching the best feature among a random subset of features (rather than all features)
- Often have high classification performance

Random Forests

For $b = 1 \dots B$:

- Draw a bootstrap sample D_b of size n from D
- Grow a tree on data D_b by recursively repeating these steps:
 - Select m variables at random from the d variables.
 - Pick the best variable and split among them
 - Split the node
- Return tree T_b

Random Forests

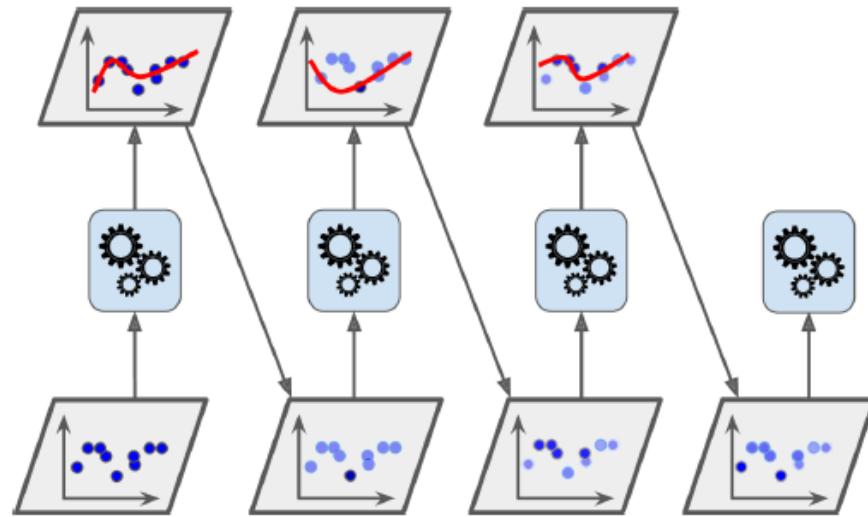
- Random forests returns an ensemble of B trees.
- Predictions are generated by voting

Boosting

- Combines several weak learners into a strong learner.
- Trains predictors sequentially, each tries to correct its predecessor
- e.g. AdaBoost and Gradient Boosting

AdaBoost

- Uses weighted samples, each example has a weight $w^{(i)}$
- Trains a classifier and uses it to make predictions on the training set
- Increases the **weight of misclassified training examples**
- Trains a second classifier on the updated weights, increase the weight of misclassified examples, and so on...



AdaBoost

Initialize weights $w^{(i)} = \frac{1}{n}$ for $i = 1, \dots, n$

For $j = 1 \dots J$:

- Train a classifier with weight $\alpha_j = \eta \log \frac{1-r_j}{r_j}$
- Update the weight of examples as follows

$$w^{(i)} := \begin{cases} w^{(i)} & \text{if } y_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{if } y_j^{(i)} \neq y^{(i)} \end{cases}$$

- Normalize weights $w^{(i)} := \frac{w^{(i)}}{\sum_{i=1}^n w^{(i)}}$

AdaBoost

Classifier weight is

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

where the error rate r_j of predictor j is $r_j = \frac{\sum_{\{i | \hat{y}_j^{(i)} \neq y^{(i)}\}} w^{(i)}}{\sum_{i=1}^n w^{(i)}}$ and η is the learning rate

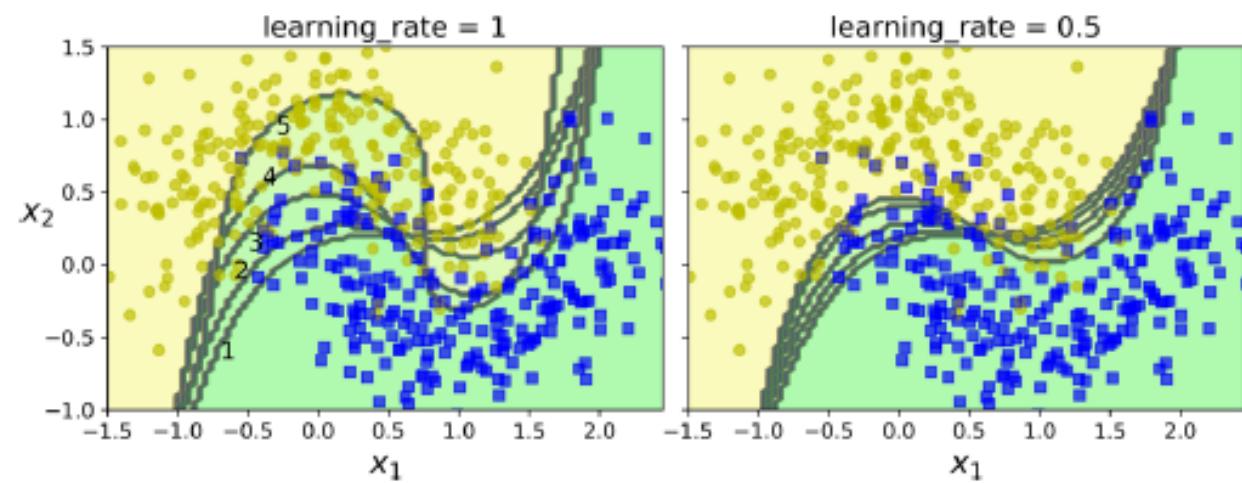
AdaBoost - Predictions

The predicted class is the class that receives the majority of the weighted votes

$$\hat{y}(x) = \arg \max_k \sum_{\{j | \hat{y}_j(x)^{(i)} = k\}} \alpha_j$$

AdaBoost Example

- Five predictors with boosting
- Different learning rates
- Lowering the learning rate η boosts misclassified examples less

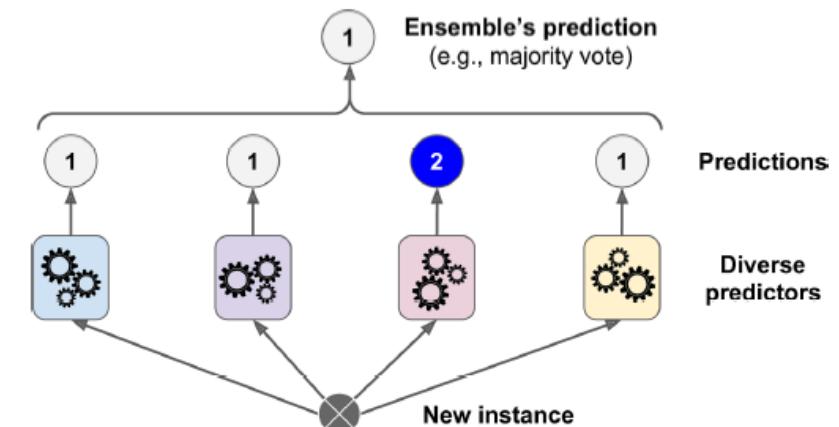
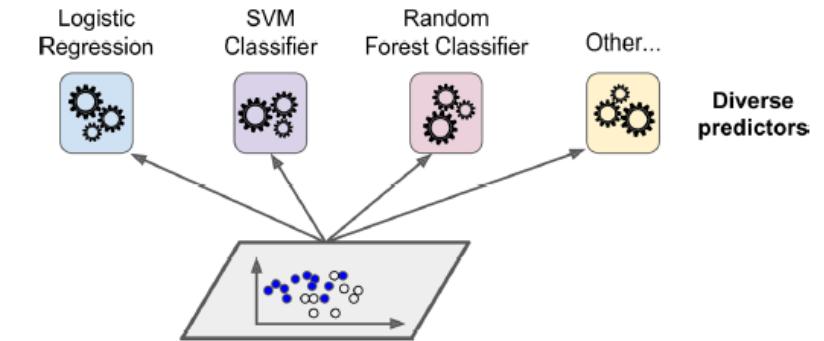


Gradient Boosting

- Another popular boosting algorithm is *Gradient Boosting*.
- Gradient Boosting sequentially adds predictors, but rather than tweaking the weights, it fits a new predictor on the *residual errors*.

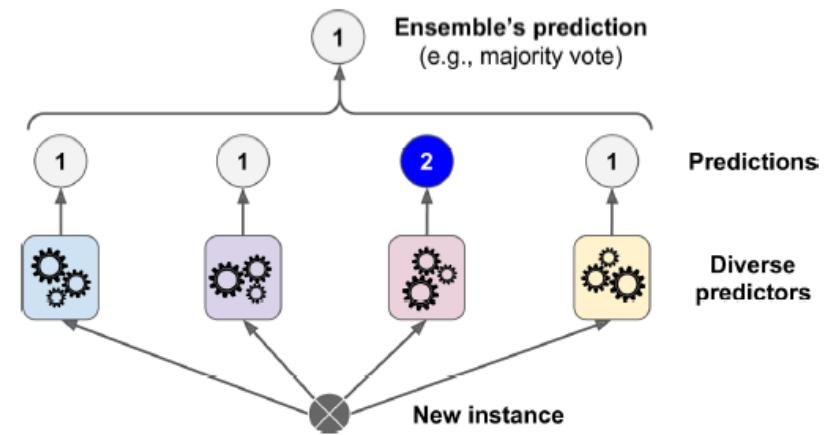
Voting Classifiers

- Suppose you have learned several classifiers each achieving about 80% accuracy (e.g. Logistic regression, SVM, k-nearest neighbor).
- You can aggregate the predictions of each classifier, and predict the class that gets most votes. (called **hard voting classifier**)



Ensemble Methods

- If classifiers are **independent**, the voting classifier can achieve a higher accuracy than the best classifier in the system.
- This is only true if they are *independent*
 - Otherwise, they tend to make the same kind of errors and the majority votes will be for the wrong class.



COGS514 - Cognition and Machine Learning

Recommender Systems

Top Picks for Joshua



Trending Now



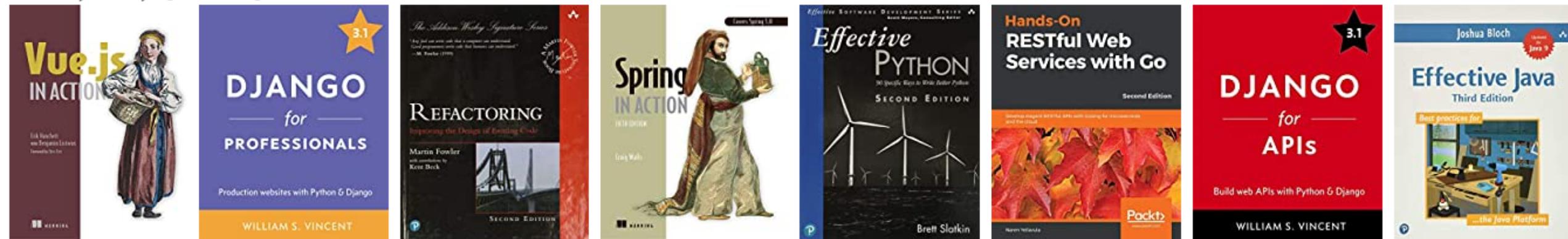
Because you watched Narcos



New Releases



Sizin için seçtiğimiz diğer ürünler



Alışveriş tercihlerinize göre belirlenen ürünler



Recommender Systems

1. Content-based recommendations
2. Collaborative filtering

Content-based Recommendations

Content-based Recommendations

- Learns from the movies that **you** have rated
- Predicts the rating for a movie that you would give based on the **features of the movie**
- We need to design input and output representation

Output representation

- Ratings ★★★★☆
- How can we encode this?

Output representation

- Ratings ★★★★☆
- How can we encode this?
 - One-hot representation ✗

Output representation

- Ratings ★★★★☆
- How can we encode this?
 - One-hot representation ✗
 - Numeric encoding $0 \leq y \leq 5$ ✓

Content-based Recommendation Inputs

Possible features include (but not limited to)

- Movie genre
- Length
- Actors
- Director
- Ratings by critics, IMDB, etc.

Content-based Recommendation Inputs

Possible features include (but not limited to)

- Movie genre, length, actors, director, ratings by critics, IMDB, etc.

We need to transform these features into a vector

$$\psi : \text{movie} \rightarrow \text{vector}$$

Content-based Recommendation Inputs

Possible features include (but not limited to)

- Movie genre, length, actors, director, ratings by critics, IMDB, etc.

We need to transform these features into a vector

$$\psi : \text{movie} \rightarrow \text{vector}$$

- $\psi(x^{(i)})$ represents the feature transformation for the i^{th} movie

Content-based recommendation

$$\mathcal{D} = \left\{ \left(\psi(x^{(1)}), (y^{(1)}) \right), \left(\psi(x^{(2)}), (y^{(2)}) \right), \dots, \left(\psi(x^{(n)}), (y^{(n)}) \right) \right\}$$

Content-based recommendation

$$\mathcal{D} = \left\{ \left(\psi(x^{(1)}), (y^{(1)}) \right), \left(\psi(x^{(2)}), (y^{(2)}) \right), \dots, \left(\psi(x^{(n)}), (y^{(n)}) \right) \right\}$$

- Looks like regression
 - x is a vector of features
 - y is real-valued

Content-based recommendation

$$\mathcal{D} = \left\{ \left(\psi(x^{(1)}), (y^{(1)}) \right), \left(\psi(x^{(2)}), (y^{(2)}) \right), \dots, \left(\psi(x^{(n)}), (y^{(n)}) \right) \right\}$$

- Looks like regression
 - x is a vector of features
 - y is real-valued
- For example, we can use *ridge regression* and optimize

$$\Phi_{ridge}(w) = \frac{1}{n} \sum_{i=1}^n (w^T \psi(x^{(i)}) + w_0 - y^{(i)})^2 + \lambda \|w\|^2$$

Problems with content-based recommendation

1. It's hard to find a good feature set
 - **Movies:** movie genre, length, actors, director, ratings by critics, IMDB, etc.
 - **Amazon:** what features to use for Amazon products considering the range of items they sell?

Problems with content-based recommendation

1. It's hard to find a good feature set

- **Movies:** movie genre, length, actors, director, ratings by critics, IMDB, etc.
- **Amazon:** what features to use for Amazon products considering the range of items they sell?

2. We only use a specific person's movie ratings, and we often don't have enough ratings for a user

- Few ratings from a specific user
- This approach does not use the ratings from other users

Collaborative filtering

Collaborative filtering

- Find other people who like the kind of movie I like
- Predict the movies I will like based on the movies they liked

Collaborative Filtering - Data

y_{ai} represents the rating for movie i from user a

Collaborative Filtering - Data

y_{ai} represents the rating for movie i from user a

- **Sparse matrix:** many missing values
- **Netflix challenge:** 480,000 users, 17,700 movies, only 1% of the matrix is filled

Collaborative Filtering - Data

y_{ai} represents the rating for movie i from user a

- **Sparse matrix:** many missing values
 - **Netflix challenge:** 480,000 users, 17,700 movies, only 1% of the matrix is filled
 - **Note:** We never actually build this matrix.
 - We represent it in terms of tuples $\mathcal{D} = \{(a, i, r)\}$ where a is the user index, i is the movie index, r is the rating for movie i from user a

Collaborative Filtering - Objective Function Option

- Let X be our predicted matrix of ratings
 - x_{ai} is the rating predicted for user a and movie i

Collaborative Filtering - Objective Function Option

- Let X be our predicted matrix of ratings
 - x_{ai} is the rating predicted for user a and movie i
- Can this be a good objective function for this method?

$$\Phi(X) = \sum_{(a,i) \in \mathcal{D}} (x_{ai} - y_{ai})^2 + \sum_{\forall (a,i)} x_{ai}^2$$

Collaborative Filtering - Objective Function Option

- Let X be our predicted matrix of ratings
 - x_{ai} is the rating predicted for user a and movie i
- Can this be a good objective function for this method?

$$\Phi(X) = \sum_{(a,i) \in \mathcal{D}} (x_{ai} - y_{ai})^2 + \sum_{\forall (a,i)} x_{ai}^2$$

Bad idea 

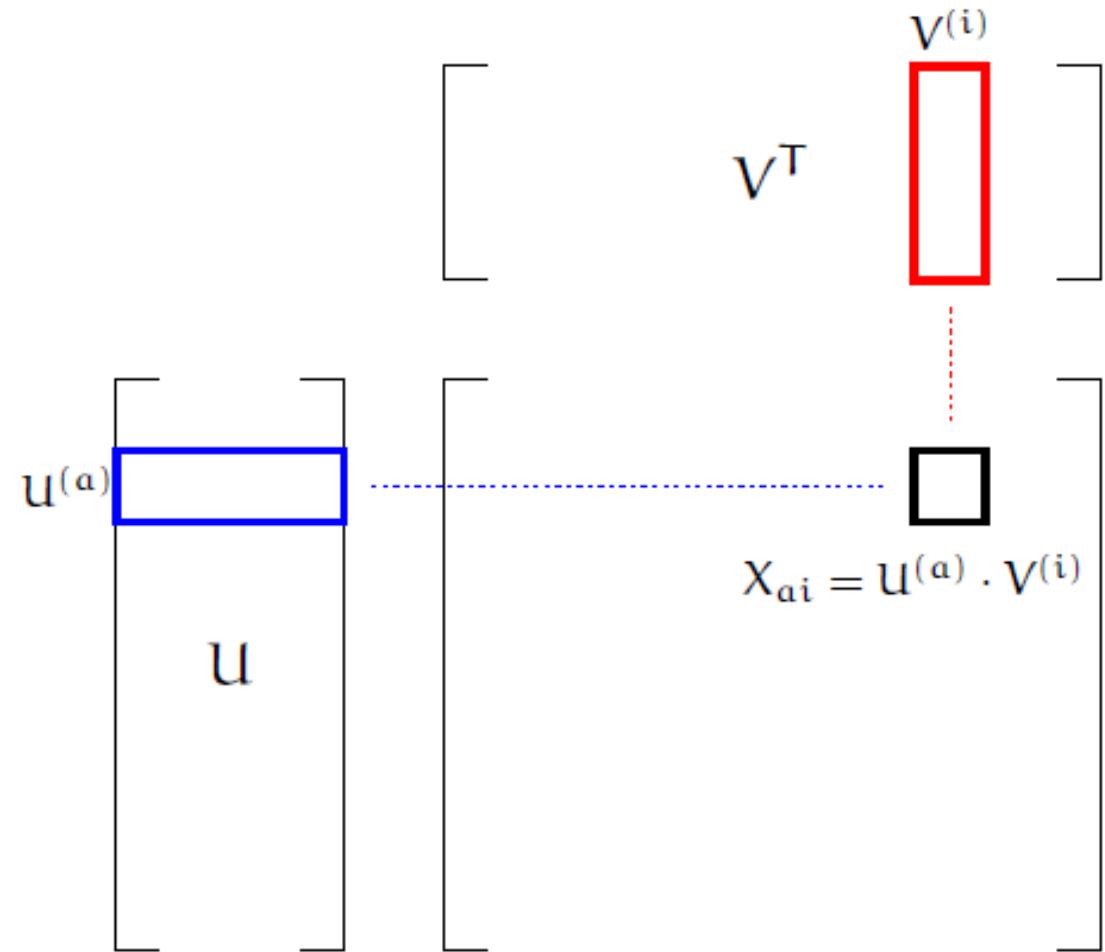
- It will set $x_{ai} = 0$ for all $(a, i) \notin \mathcal{D}$
- In other words, if the user did not rate a movie, this will predict it to be 0
- Why?

Collaborative Filtering - A Better Idea

- We have m users and n movies. X and Y are both $m \times n$ matrices.

$$X = U \cdot V^T$$

- $U \in \mathbb{R}^m$, i.e. $m \times 1$ vector
- $V \in \mathbb{R}^n$, i.e. $n \times 1$ vector



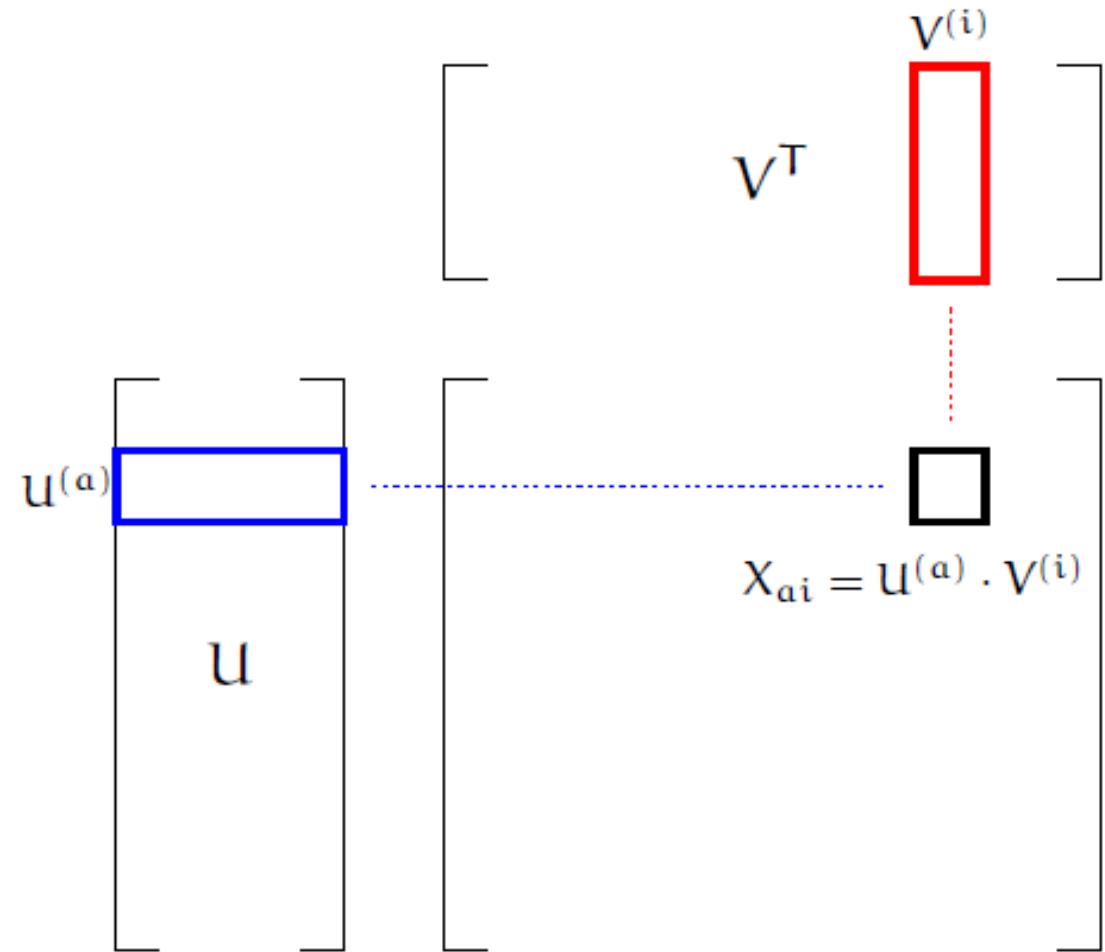
Collaborative Filtering

- Find a $m \times 1$ vector U and $n \times 1$ vector V

$$X = U \cdot V^T$$

- Prediction for user a and movie i is

$$x_{ai} = U^{(a)} V^{(i)}$$



Collaborative Filtering

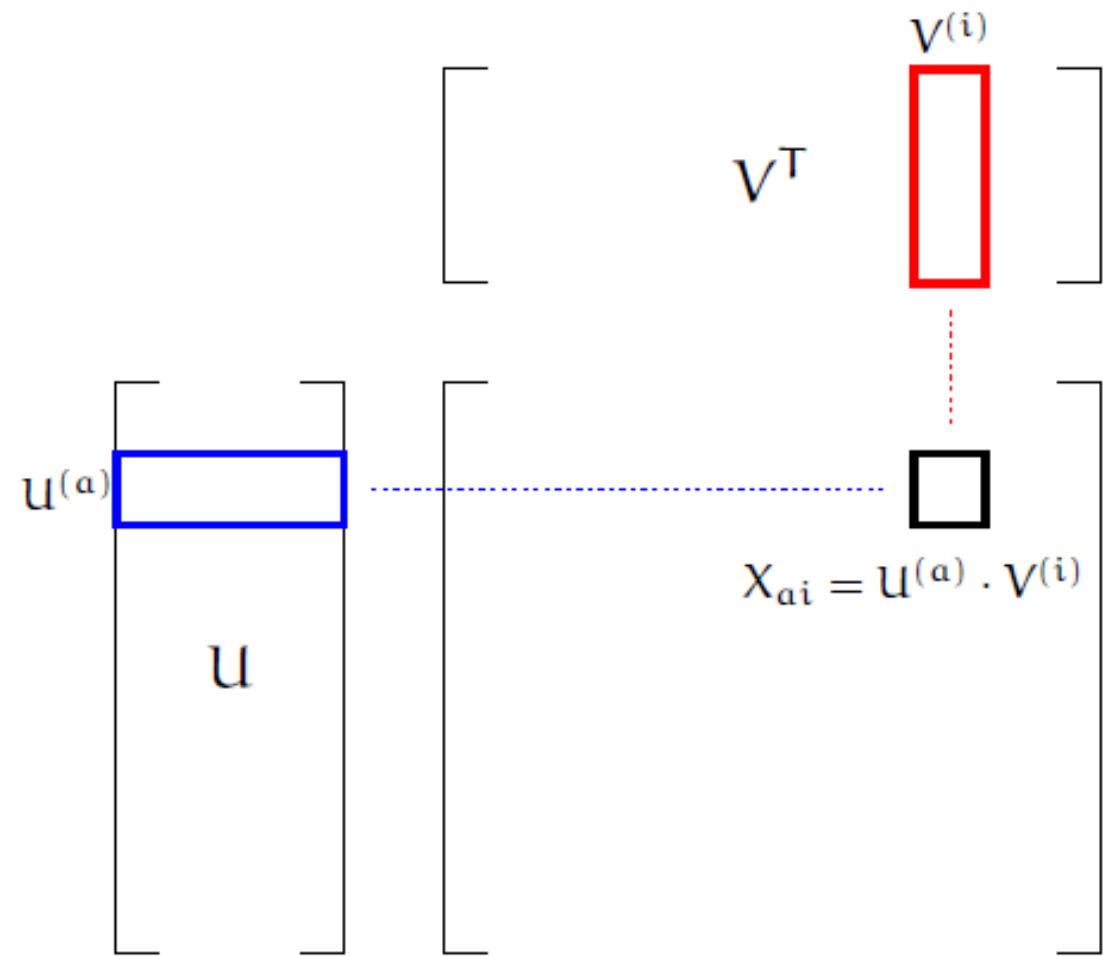
- Find a $m \times 1$ vector U and $n \times 1$ vector V

$$X = U \cdot V^T$$

- Prediction for user a and movie i is

$$x_{ai} = U^{(a)} V^{(i)}$$

- We represent every user (and every movie) with one parameter.
 - This is not expressive enough, only represents overall enthusiasm of users and general popularity of movies



Collaborative Filtering with k parameters

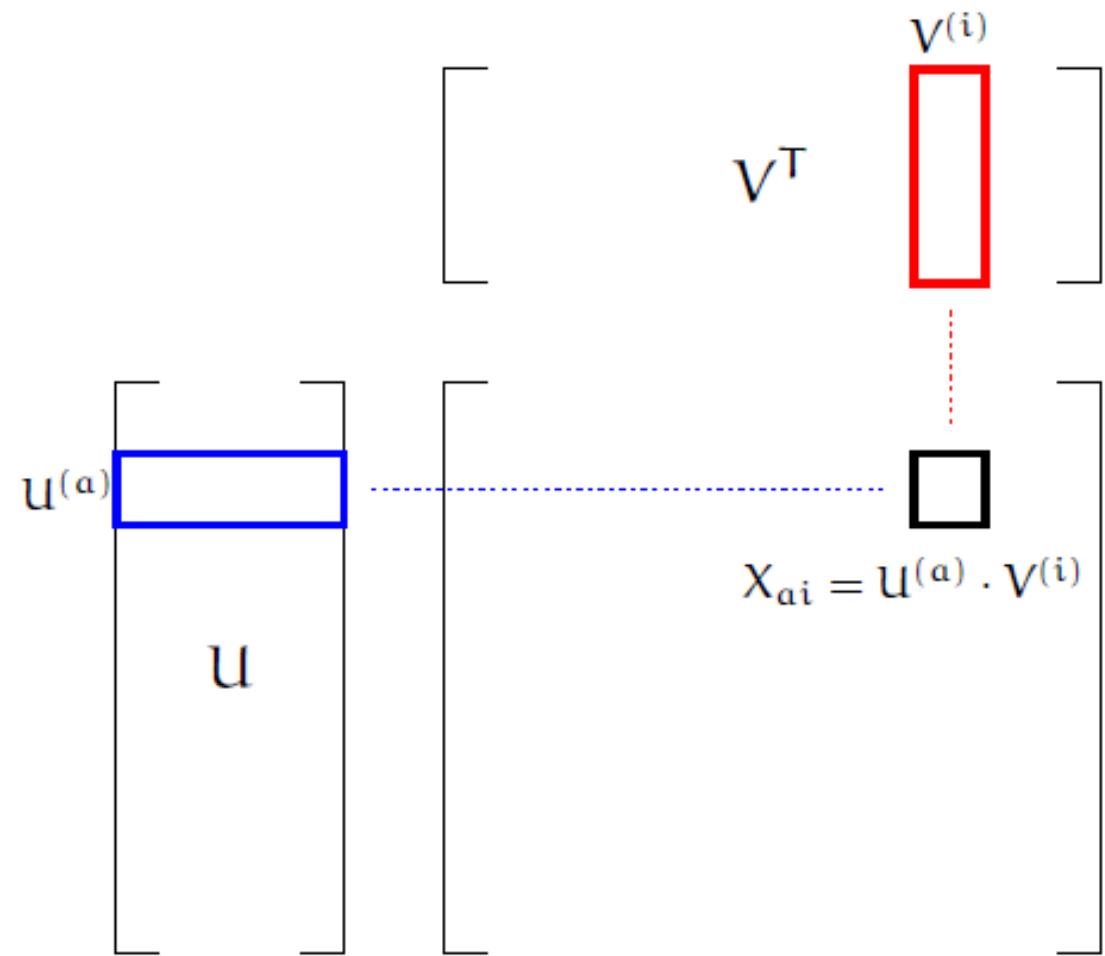
- Find a $m \times k$ matrix U and $n \times k$ matrix V

$$X = U \cdot V^T$$

- Prediction for user a and movie i is

$$x_{ai} = U^{(a)} V^{(i)}$$

- Now we have k parameters for each movie and user.



Collaborative Filtering - Optimization

Objective Function

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(\alpha)} V^{(i)} - Y_{ai} \right)^2 + \text{Regularizers for } U \text{ and } V$$

Collaborative Filtering - Optimization

Objective Function with offset and regularizers

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|U^{(a)}\|^2 + \frac{\lambda}{2} \|V^{(i)}\|^2$$

Offsets $b_U \in \mathbb{R}^{m \times 1}$ and $b_V \in \mathbb{R}^{n \times 1}$ vectors

Recall - Stochastic Gradient Descent (SGD)

for $t = 1$ to T

- randomly select $i \in 1, 2, \dots, n$
 - $w^{(t)} = w^{(t-1)} - \alpha(t) \nabla_w f_i(w^{(t-1)})$
- return $w^{(t)}$

Collaborative Filtering - Optimization

Objective Function

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} - Y_{ai} \right)^2 + \text{Regularizers for } U \text{ and } V$$

Collaborative Filtering - Optimization with SGD

- Gradients for the loss function (without the regularizer)

$$\frac{\partial \Phi(U, V)}{\partial U^{(a)}} = \sum_{i|(a,i) \in \mathcal{D}} (U^{(a)} V^{(i)} - y_{a,i}) V^{(i)}$$

$$\frac{\partial \Phi(U, V)}{\partial V^{(i)}} = \sum_{a|(a,i) \in \mathcal{D}} (U^{(a)} V^{(i)} - y_{a,i}) U^{(a)}$$

Collaborative Filtering - Optimization

Objective Function with offset and regularizers

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|U^{(a)}\|^2 + \frac{\lambda}{2} \|V^{(i)}\|^2$$

Collaborative Filtering - Optimization with SGD

- Gradients with offset and regularizer

$$\frac{\partial \Phi(U, V)}{\partial U^{(a)}} = \sum_{i|(a,i) \in \mathcal{D}} (U^{(\alpha)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai}) V^{(i)} + \lambda_U^{(a)} U^{(a)}$$

$$\frac{\partial \Phi(U, V)}{\partial b_U^{(a)}} = \sum_{i|(a,i) \in \mathcal{D}} (U^{(\alpha)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai})$$

$$\lambda_U^{(a)} = \frac{\lambda}{\text{number of ratings for user } a} = \frac{\lambda}{\sum_{i|(a,i) \in \mathcal{D}} 1}$$

Collaborative Filtering - Optimization with SGD

- Gradients with offset and regularizer (contd.)

$$\frac{\partial \Phi(U, V)}{\partial V^{(i)}} = \sum_{a|(a,i) \in \mathcal{D}} (U^{(\alpha)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai}) U^{(a)} + \lambda_V^{(i)} V^{(i)}$$

$$\frac{\partial \Phi(U, V)}{\partial b_V^{(i)}} = \sum_{a|(a,i) \in \mathcal{D}} (U^{(\alpha)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai})$$

$$\lambda_V^{(i)} = \frac{\lambda}{\text{number of ratings for movie } i} = \frac{\lambda}{\sum_{a|(a,i) \in \mathcal{D}} 1}$$

SGD for Collaborative Filtering

for $t = 1$ to T :

- randomly select $a \in 1, 2, \dots, m$ and $i \in 1, 2, \dots, n$
- $U = U - \alpha(t) \nabla_{U^{(a)}} \Phi(U, V)$
- $b_U^{(a)} = b_U^{(a)} - \alpha(t) \nabla_{b_U^{(a)}} \Phi(U, V)$
- $V = V - \alpha(t) \nabla_{V^{(i)}} \Phi(U, V)$
- $b_V^{(i)} = b_V^{(i)} - \alpha(t) \nabla_{b_V^{(i)}} \Phi(U, V)$

return U, V

Another Approach for Optimizing Alternating Least Squares

Alternating Least Squares

- Alternate between
 1. Fix U and b_U , and minimize V and b_V with linear regression (least squares)
 2. Fix V and b_V , and minimize U and b_U with linear regression (least squares)

Alternating Least Squares

1. Initialize V and b_V at random

Alternating Least Squares

1. Initialize V and b_V at random

2. For a in $1, 2, \dots, n$:

- Construct a linear regression problem to find $U^{(a)}$ and $b_U^{(a)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|U^{(a)}\|^2$$

Alternating Least Squares

1. Initialize V and b_V at random

2. For a in $1, 2, \dots, n$:

- Construct a linear regression problem to find $U^{(a)}$ and $b_U^{(a)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|U^{(a)}\|^2$$

3. For i in $1, 2, \dots, m$:

- Construct a linear regression problem to find $V^{(i)}$ and $b_V^{(i)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|V^{(i)}\|^2$$

Alternating Least Squares

1. Initialize V and b_V at random

2. For a in $1, 2, \dots, n$:

- Construct a linear regression problem to find $U^{(a)}$ and $b_U^{(a)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|U^{(a)}\|^2$$

3. For i in $1, 2, \dots, m$:

- Construct a linear regression problem to find $V^{(i)}$ and $b_V^{(i)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|V^{(i)}\|^2$$

4. Alternate between steps 2 and 3 (optimize U and V). Stop after a fixed number of iterations or when the difference between successive parameters is small

Alternating Least Squares

1. Initialize V and b_V at random

2. For a in $1, 2, \dots, n$:

- Construct a linear regression problem to find $U^{(a)}$ and $b_U^{(a)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_u^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|U^{(a)}\|^2$$

3. For i in $1, 2, \dots, m$:

- Construct a linear regression problem to find $V^{(i)}$ and $b_V^{(i)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_u^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|V^{(i)}\|^2$$

4. Alternate between steps 2 and 3 (optimize U and V). Stop after a fixed number of iterations or when the difference between successive parameters is small

- We know how to minimize

$$\Phi(w) = \frac{1}{n} (Xw - y)^T (Xw - y)$$

Alternating Least Squares

1. Initialize V and b_V at random

2. For a in $1, 2, \dots, n$:

- Construct a linear regression problem to find $U^{(a)}$ and $b_U^{(a)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_u^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|U^{(a)}\|^2$$

3. For i in $1, 2, \dots, m$:

- Construct a linear regression problem to find $V^{(i)}$ and $b_V^{(i)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_u^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|V^{(i)}\|^2$$

4. Alternate between steps 2 and 3 (optimize U and V). Stop after a fixed number of iterations or when the difference between successive parameters is small

- We know how to minimize

$$\Phi(w) = \frac{1}{n} (Xw - y)^T (Xw - y)$$

- In step 2 consider $w = U^{(a)}$, y is the target value for the movies that has been rated. X is the matrix whose rows are $V^{(i)}$ where user a has rated movie i

Alternating Least Squares

1. Initialize V and b_V at random

2. For a in $1, 2, \dots, n$:

- Construct a linear regression problem to find $U^{(a)}$ and $b_U^{(a)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_u^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|U^{(a)}\|^2$$

3. For i in $1, 2, \dots, m$:

- Construct a linear regression problem to find $V^{(i)}$ and $b_V^{(i)}$ that minimizes

$$\Phi(U, V) = \frac{1}{2} \sum_{(a,i) \in \mathcal{D}} \left(U^{(a)} V^{(i)} + b_u^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \|V^{(i)}\|^2$$

4. Alternate between steps 2 and 3 (optimize U and V). Stop after a fixed number of iterations or when the difference between successive parameters is small

- We know how to minimize

$$\Phi(w) = \frac{1}{n} (Xw - y)^T (Xw - y)$$

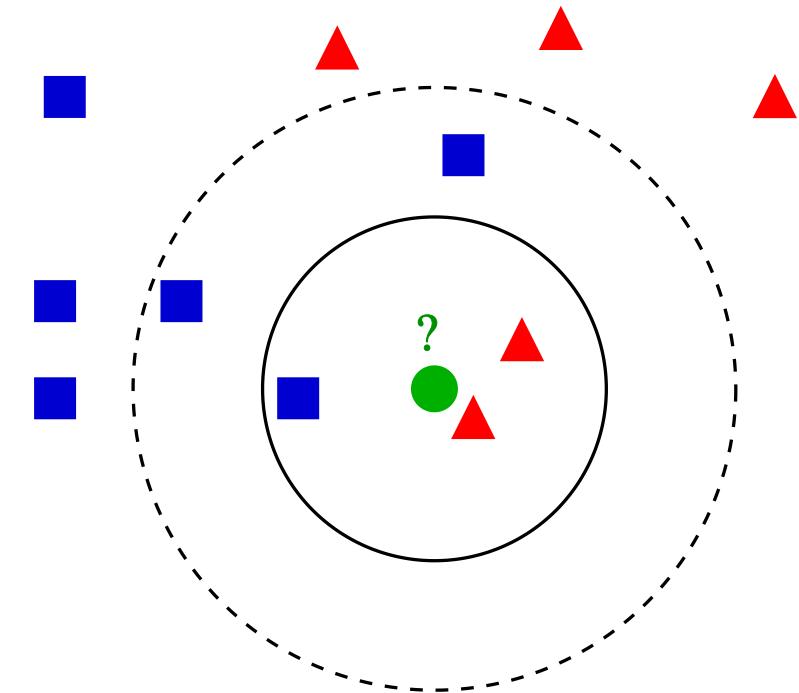
- In step 3 consider $w = V^{(i)}$, y is the target value for the movies that has been rated. X is the matrix whose rows are $U^{(a)}$ where user a has rated movie i

Another approach for recommendation

k -nearest neighbours

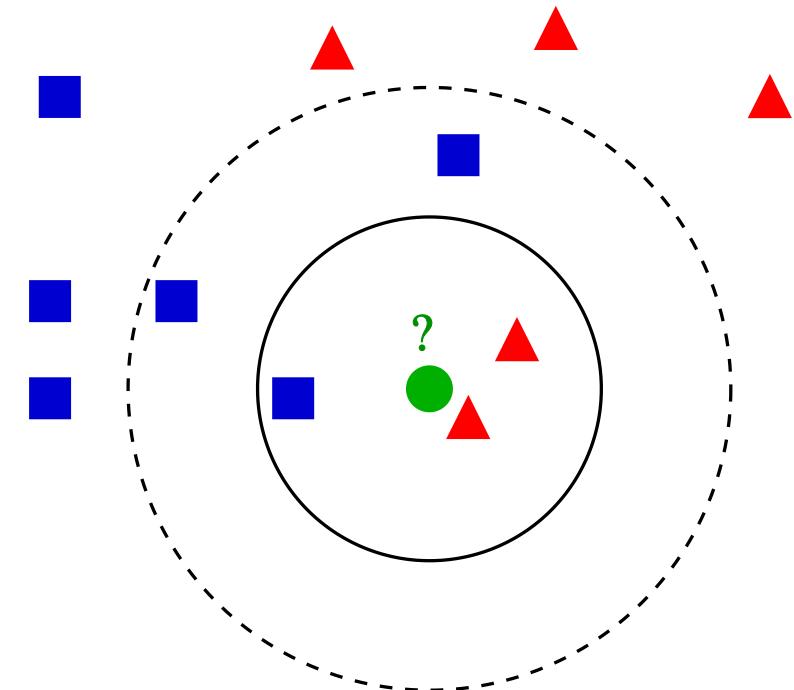
k-nearest neighbours

- Find the closest users to me
- Recommend the movies they like



k -nearest neighbours

- Find the closest users to me
- Recommend the movies they like
- We need a **distance metric**



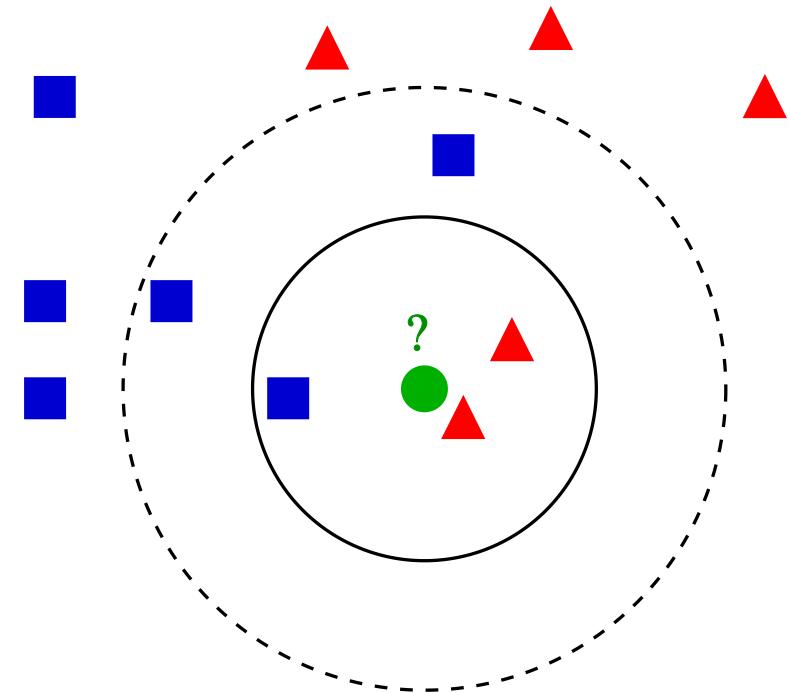
Distance metric

A distance function $d : X \times X \rightarrow \mathbb{R}$ has the following properties:

$$d(x, x) = 0$$

$$d(x, x') = d(x', x)$$

$$d(x, x'') \leq d(x, x') + d(x', x'')$$

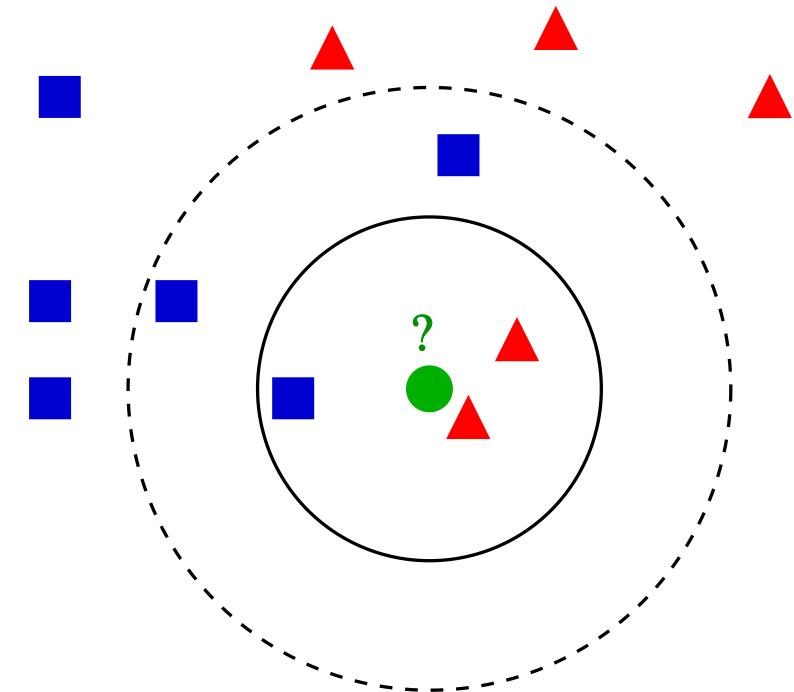


k-nearest neighbour

Prediction is the example with the smallest distance

$$h(x) = y^{(i)}$$

$$i = \arg \min_i d(x, x^{(i)})$$



k-nearest neighbour - some similarity measures

Euclidian distance

$$d(x, x') = \|x - x'\|$$

Cosine similarity

$$d(x, x') = \frac{x \cdot x'}{\|x\| \|x'\|}$$

COGS514 - Cognition and Machine Learning

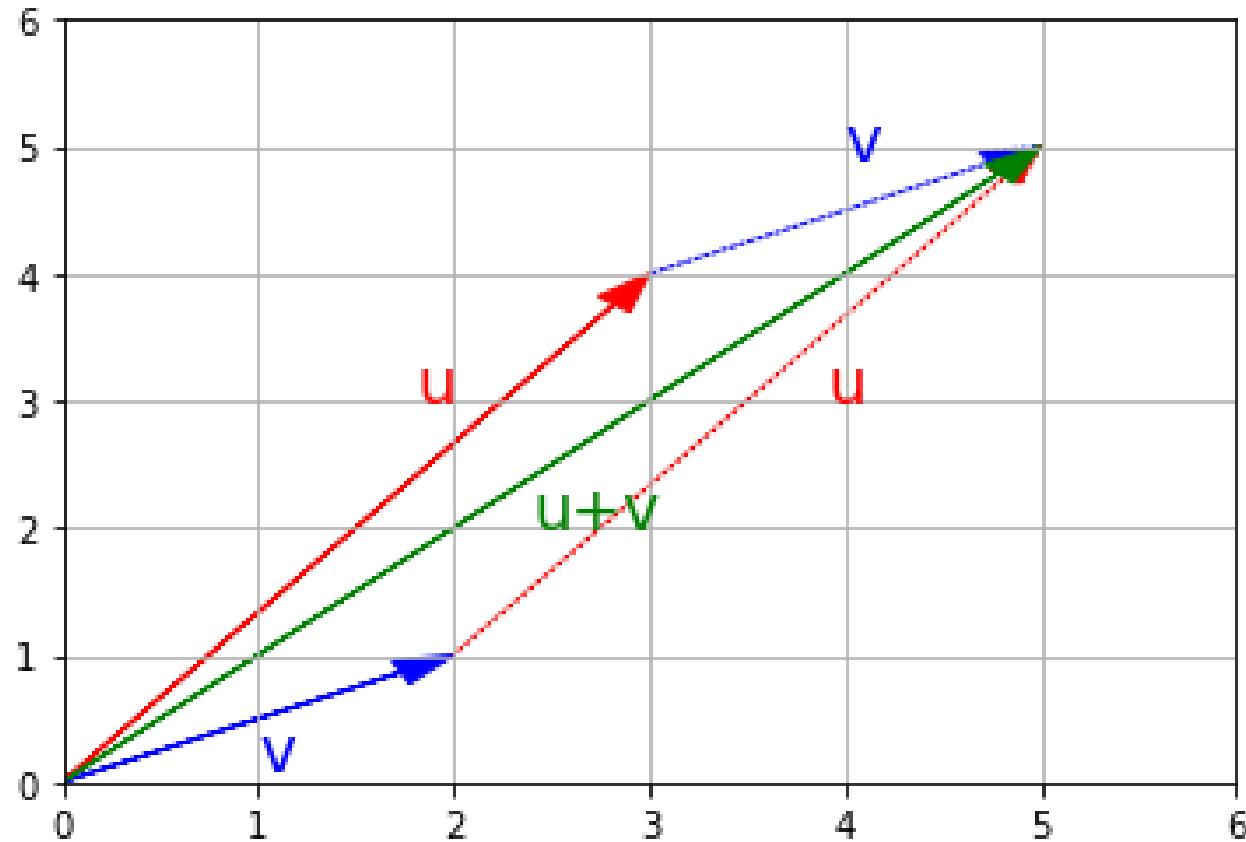
Review

Linear Algebra

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \in \mathbb{R}^2$$

$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$$

$$u + v = z = \begin{bmatrix} 3 + 2 \\ 4 + 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \in \mathbb{R}^2$$



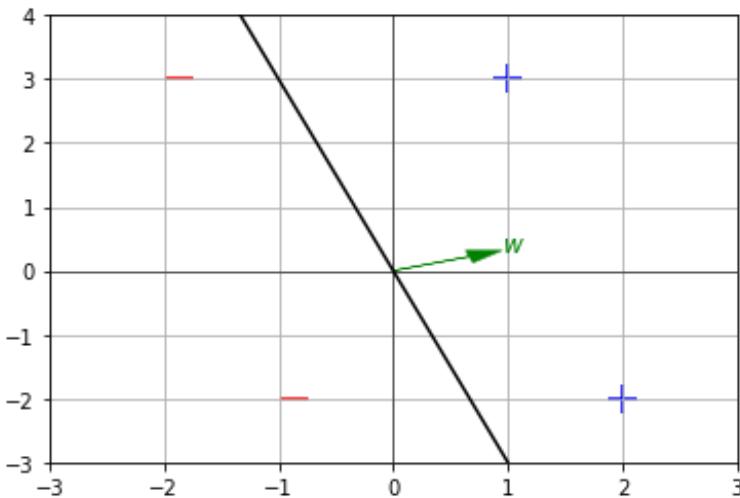
Linear Classifiers

$$\mathbf{X} = \begin{bmatrix} [1] & [2] & [-2] & [-1] \\ [3] & [-2] & [3] & [-2] \end{bmatrix}$$

$$\mathbf{Y} = [[+1] \quad [+1] \quad [-1] \quad [-1]]$$

$$\mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$h(\mathbf{x}, \mathbf{w}) = \begin{cases} +1 & \text{if } 3x_1 + 1x_2 > 0 \\ -1 & \text{otherwise} \end{cases}$$



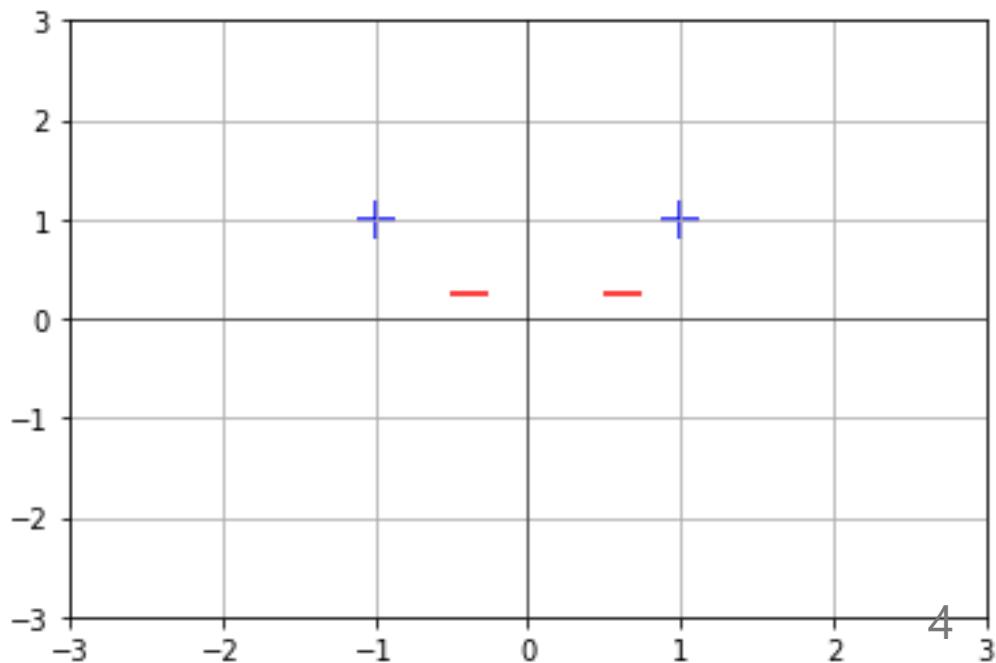
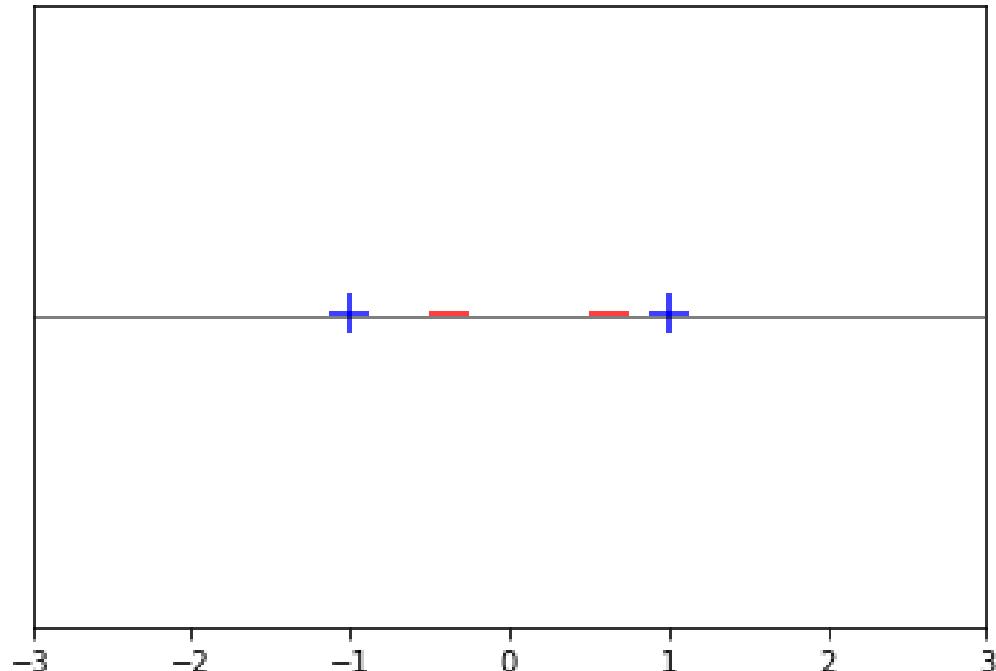
Non-linear Classifiers

$$\mathbf{X} = [[-1] \quad [-0.5] \quad [0.5] \quad [1]]$$

$$\mathbf{Y} = [[+1] \quad [-1] \quad [-1] \quad [+1]]$$

$$\phi(\mathbf{x}) = [x \quad x^2]^T$$

$$\phi(\mathbf{X}) = \begin{bmatrix} [-1] & [-0.5] & [0.5] & [1] \\ [1] & [0.25] & [0.25] & [1] \end{bmatrix}$$



Encoding

- Blood type $\{A+, A-, B+, B-, 0+, 0-, AB+, AB-\}$
- One hot encoding, Blood Type is $B+:$

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

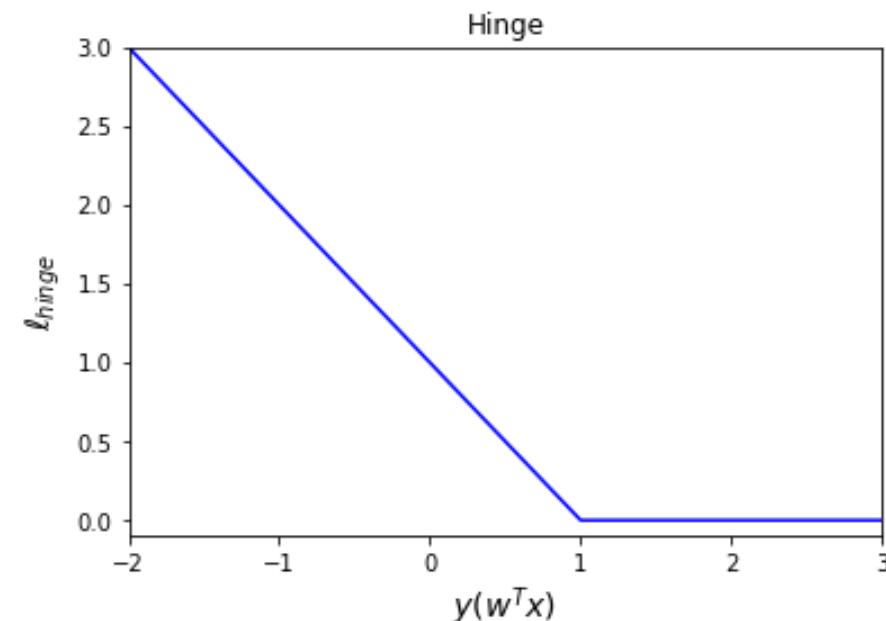
Machine Learning is Optimization

$$\Phi(\mathbf{w}) = \underbrace{\frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)})}_{\text{Empirical risk of classifier } h} + \overbrace{\lambda}^{\text{Constant}} \underbrace{R(\mathbf{w})}_{\text{Regularizer}}$$

SVM, Logistic Regression

Hinge Loss (SVM)

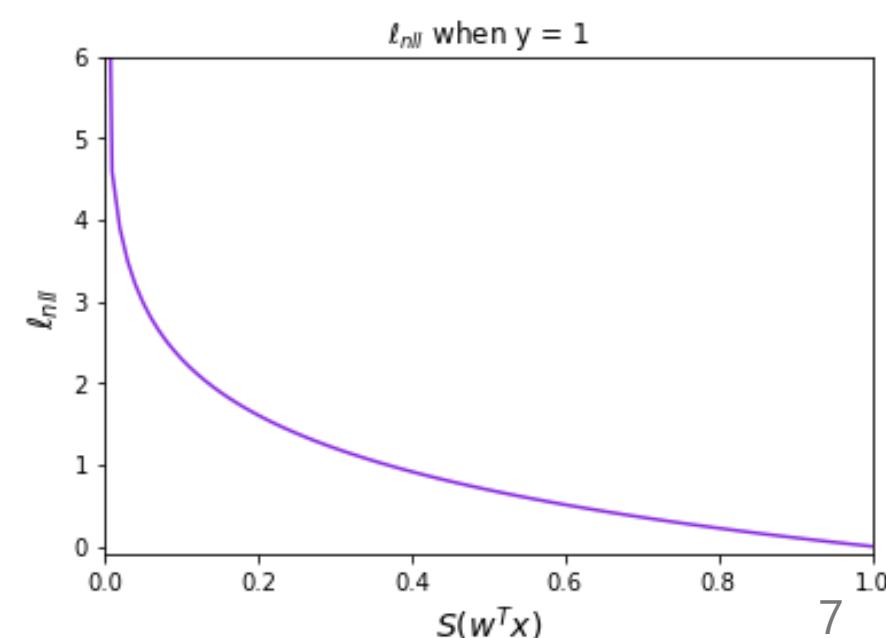
$$\ell_h(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = \begin{cases} 0 & y^{(i)}(w^T x^{(i)}) > 1 \\ 1 - y^{(i)}(w^T x^{(i)}) & \text{otherwise} \end{cases}$$
$$= \max(1 - y^{(i)}(w^T x^{(i)}), 0)$$



Negative Log-Likelihood Loss Function ℓ_{nll} (Logistic Regression)

$$\ell_{nll}(g^{(i)}, y^{(i)}) = - \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right)$$

$$\ell_{nll}(\text{guess, actual}) = - (\text{actual} \cdot \log(\text{guess}) + (1 - \text{actual}) \cdot \log(1 - \text{guess}))$$



Linear Regression

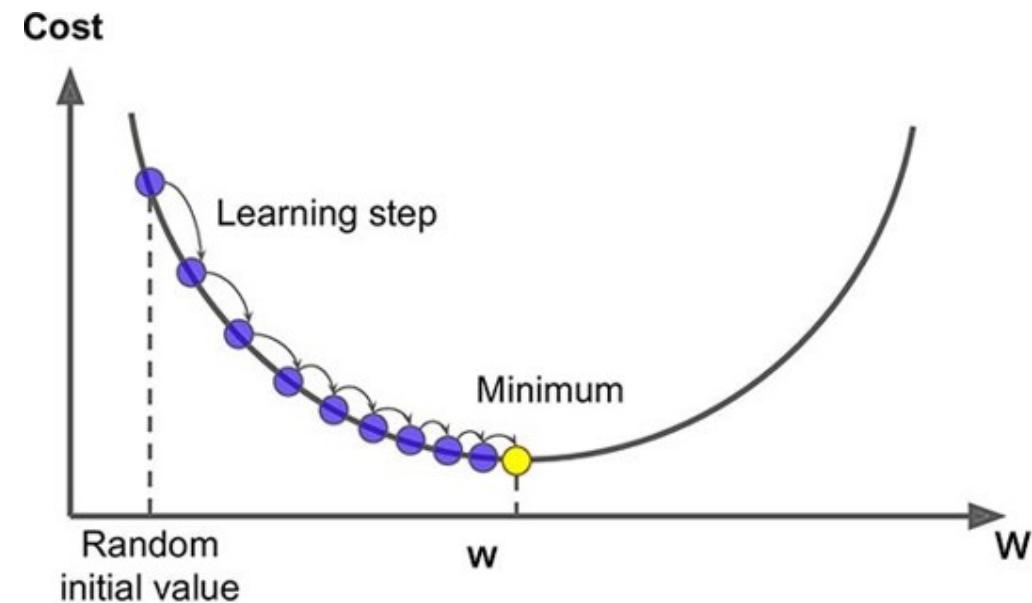
$$\ell_{\text{squared}}(h(\mathbf{x}^{(i)}; \mathbf{w}), y^{(i)}) = (w^T x^{(i)} - y^{(i)})^2$$

(Stochastic) Gradient Descent

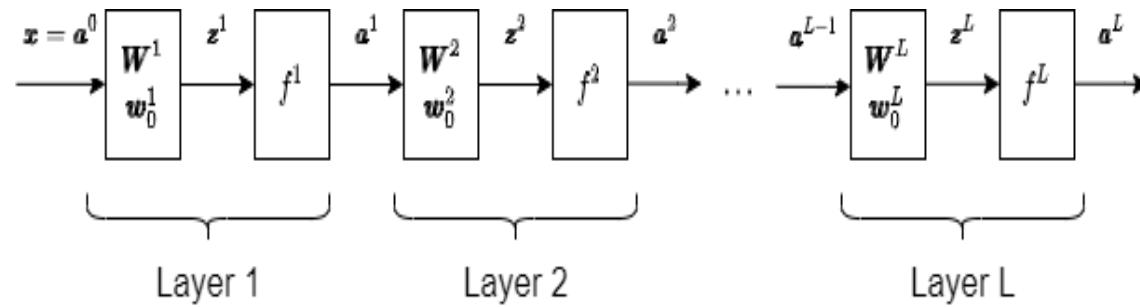
for $t = 1$ to T

- randomly select $i \in 1, 2, \dots, n$
 - $w^{(t)} = w^{(t-1)} - \alpha(t) \nabla_w \Phi(w^{(t-1)})$
- return $w^{(t)}$
-

$$\nabla_w \Phi = \begin{bmatrix} \partial \Phi / \partial w_1 \\ \partial \Phi / \partial w_2 \\ \vdots \\ \partial \Phi / \partial w_d \end{bmatrix}$$



Neural Networks

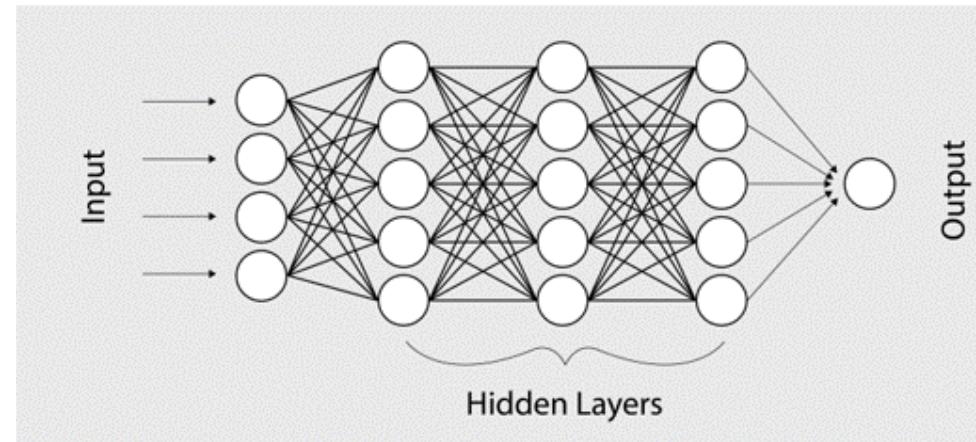
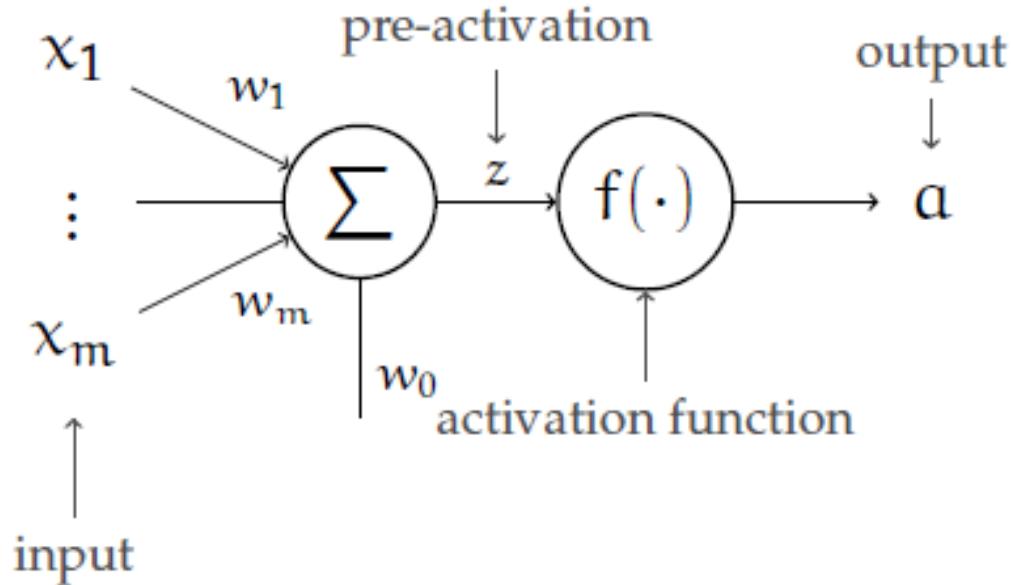


- W^l and w_0^l have $m^l \times n^l$ and $n^l \times 1$ shapes respectively
- f^l be the activation function of layer l
- z^l pre-activation are $n^l \times 1$ vector.

$$z^l = W^{l^T} a^{l-1} + w_0^l$$

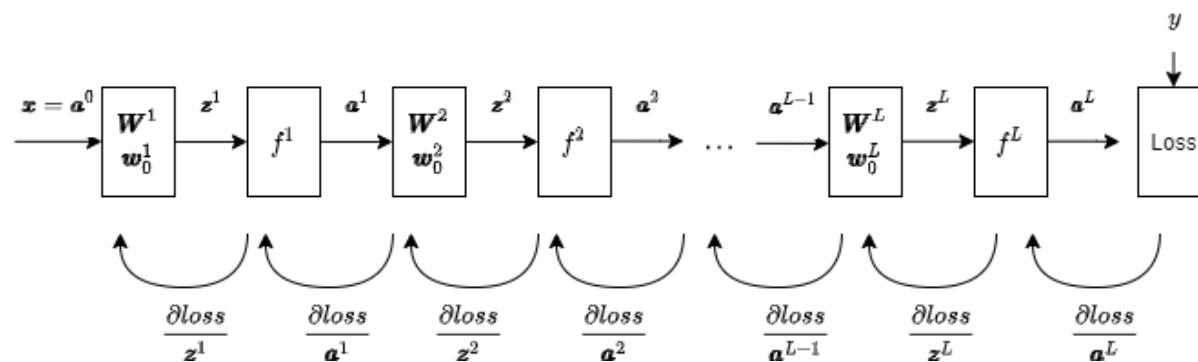
- a^l activation outputs are $n^l \times 1$ vectors.

$$a^l = f^l(z^l)$$



Backpropagation

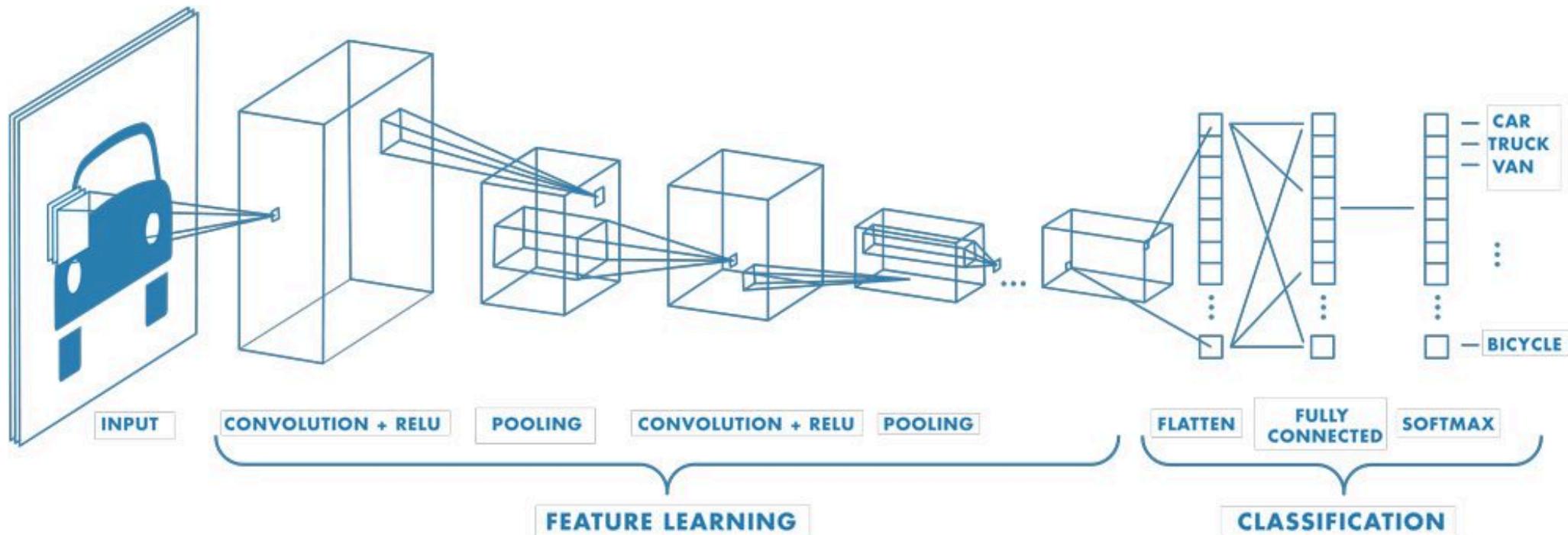
$$\frac{\partial \text{loss}}{\partial z^1} = \frac{\partial \text{loss}}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial a^{L-2}} \cdot \dots \cdot \frac{\partial a^2}{\partial z^2} \cdot \frac{\partial z^2}{\partial a^1} \cdot \frac{\partial a^1}{\partial z^1}$$



Making Deep Neural Networks Work

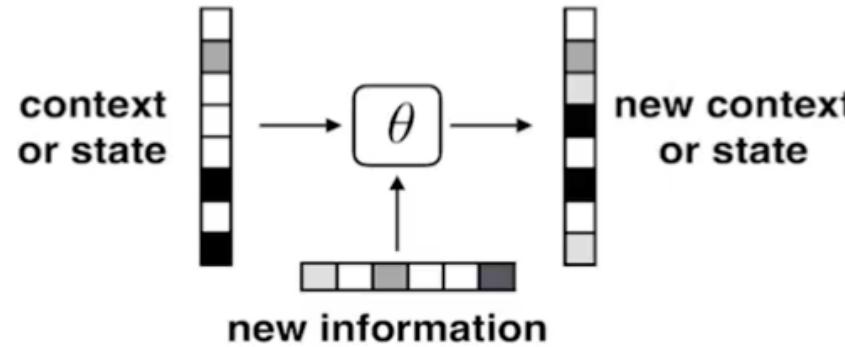
- Batch gradient descent
- ReLU and Leaky ReLU for preventing vanishing gradients
- Improved optimization (momentum, ADAM)
- Regularization (Dropout, Batch Normalization)

Convolutional Neural Networks

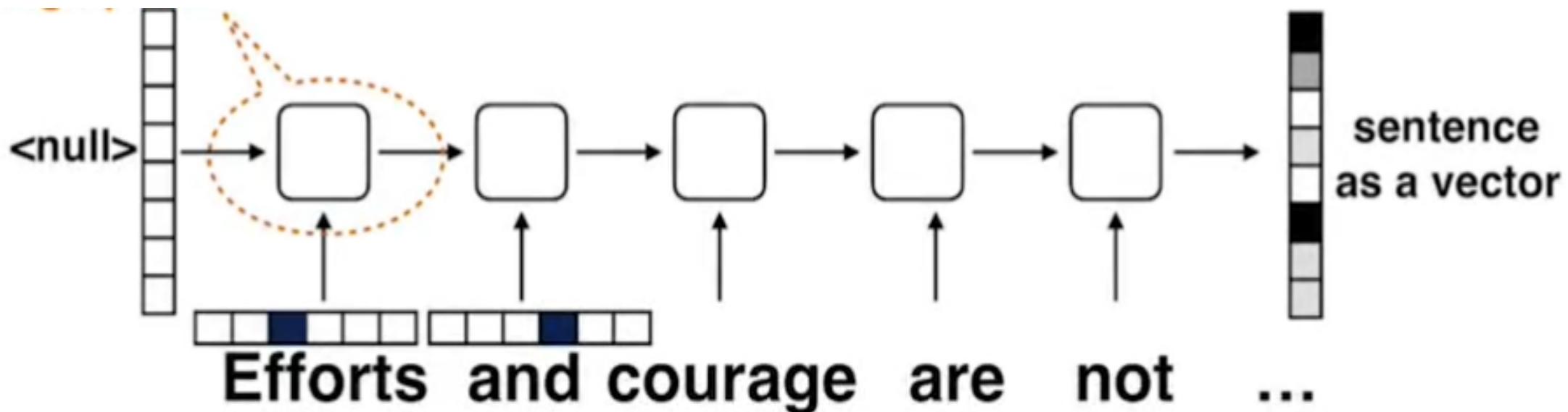


Recurrent Neural Networks

Encoding

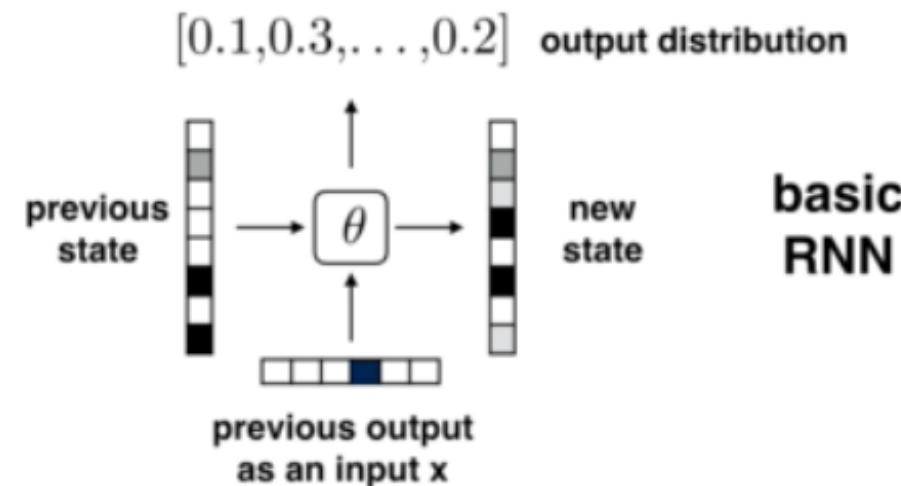


$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t)$$



Recurrent Neural Networks

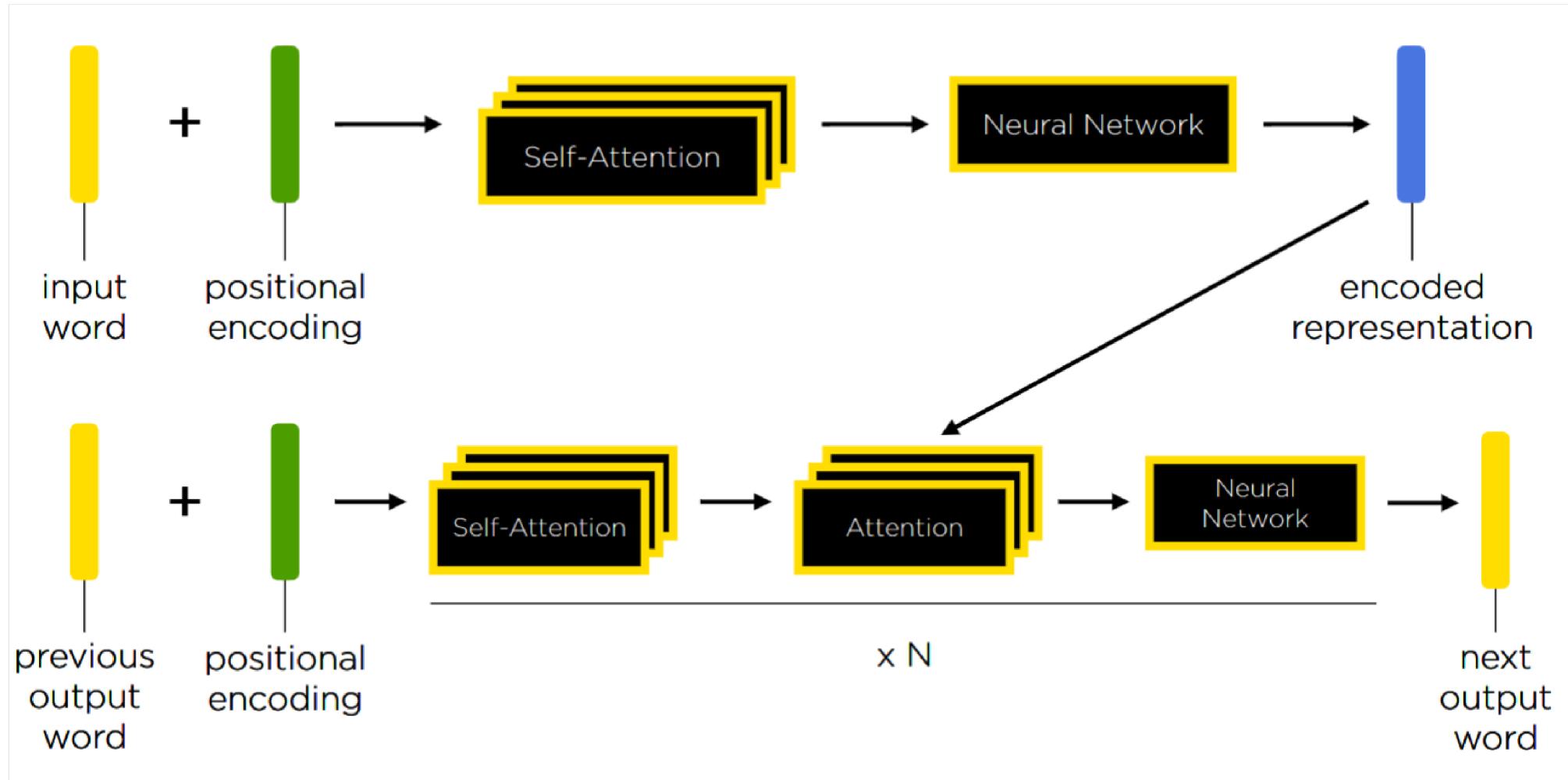
Decoding



$$s_t = \tanh(W^{s,s}s_{t-1} + W^{s,x}x_t) \quad \text{state}$$

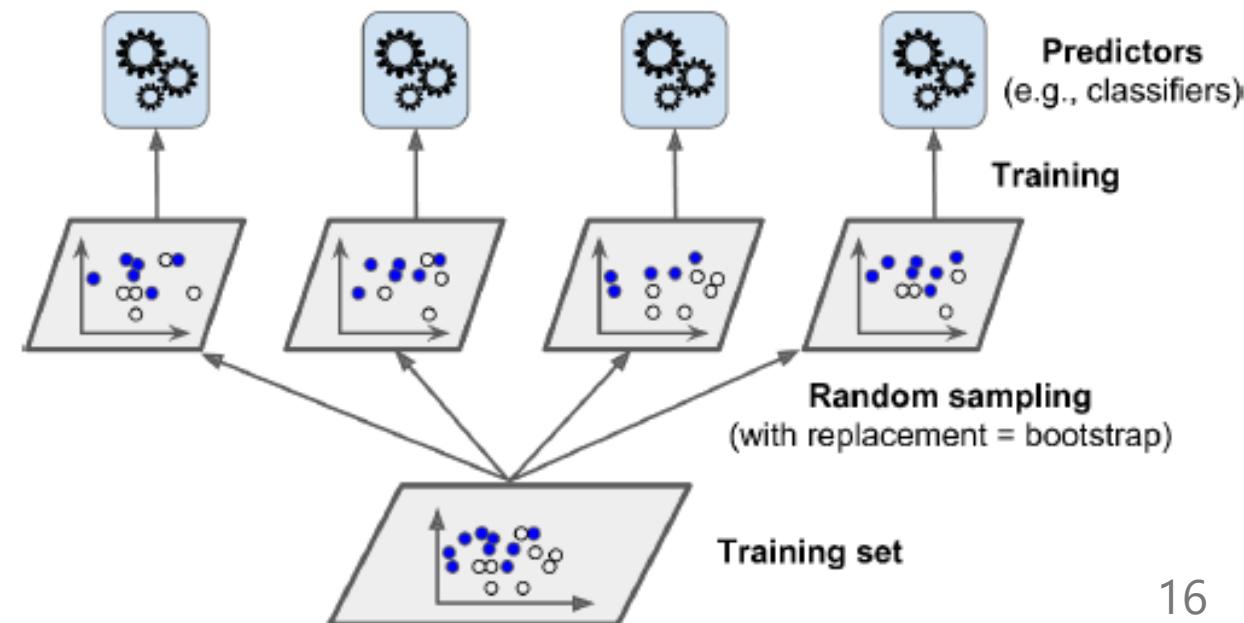
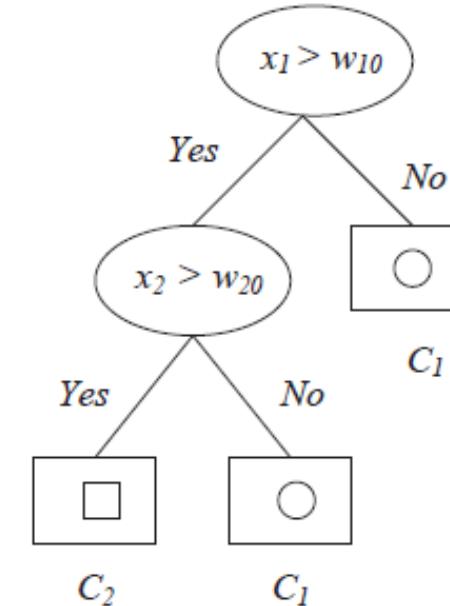
$$p_t = \text{softmax}(W^o s_t) \quad \text{output distribution}$$

Transformers



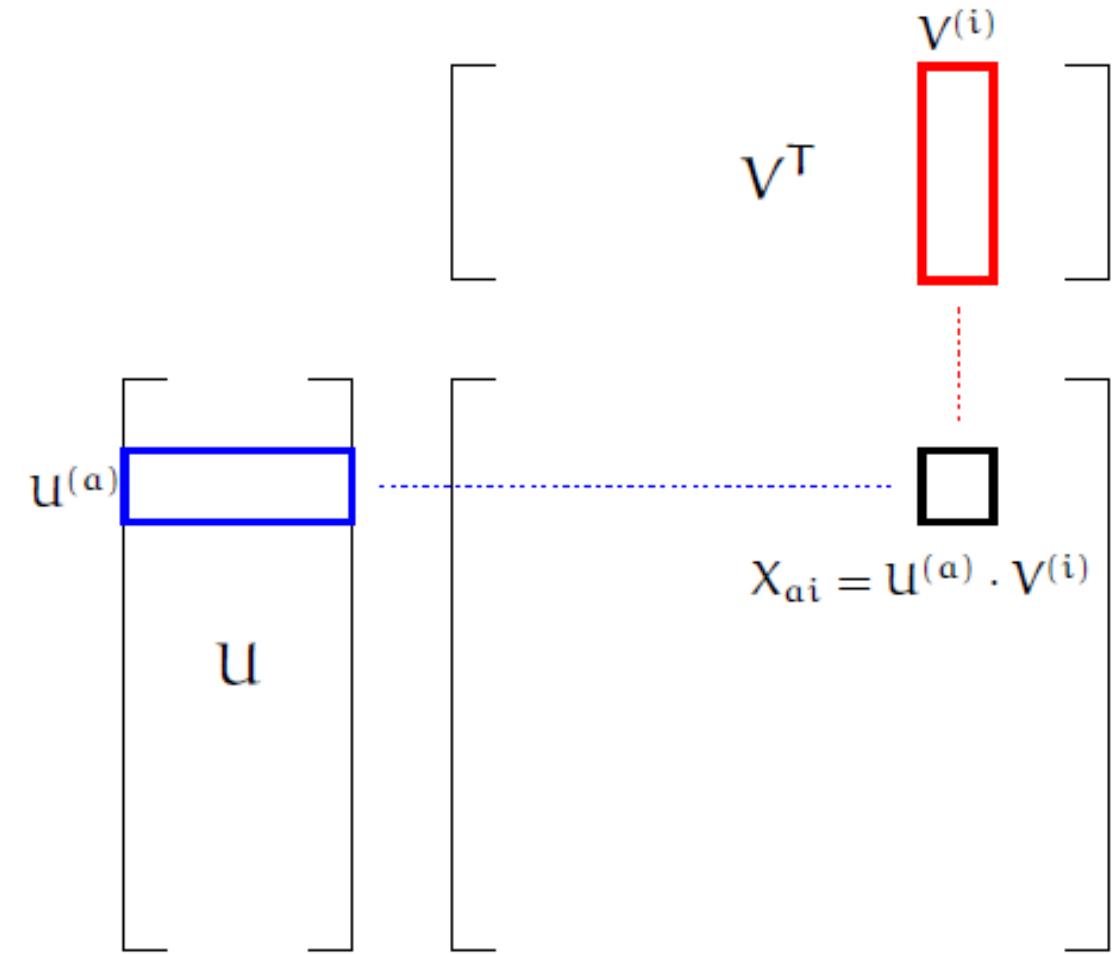
Semi-parametric Methods

- Decision Trees
- Bagging
- Boosting
- Random Forests
- Ensemble Methods



Recommender Systems

- Collaborative Filtering
 - Matrix Factorization



What's next?

- Deep Learning
 - DI 504 / MMI 727 Foundations of Deep Learning
 - More Advanced:
 - DI 725 Transformers and Attention-Based Deep Networks
 - MMI 711 Sequence Models in Multimedia
 - MMI 714 Generative Models for Multimedia
 - MMI 711 Sequence Models in Multimedia
- Probabilistic Machine Learning
 - COGS 516 Introduction to Probabilistic Programming
- Connectionism
 - COGS 555 Connectionism and Human Behavior