

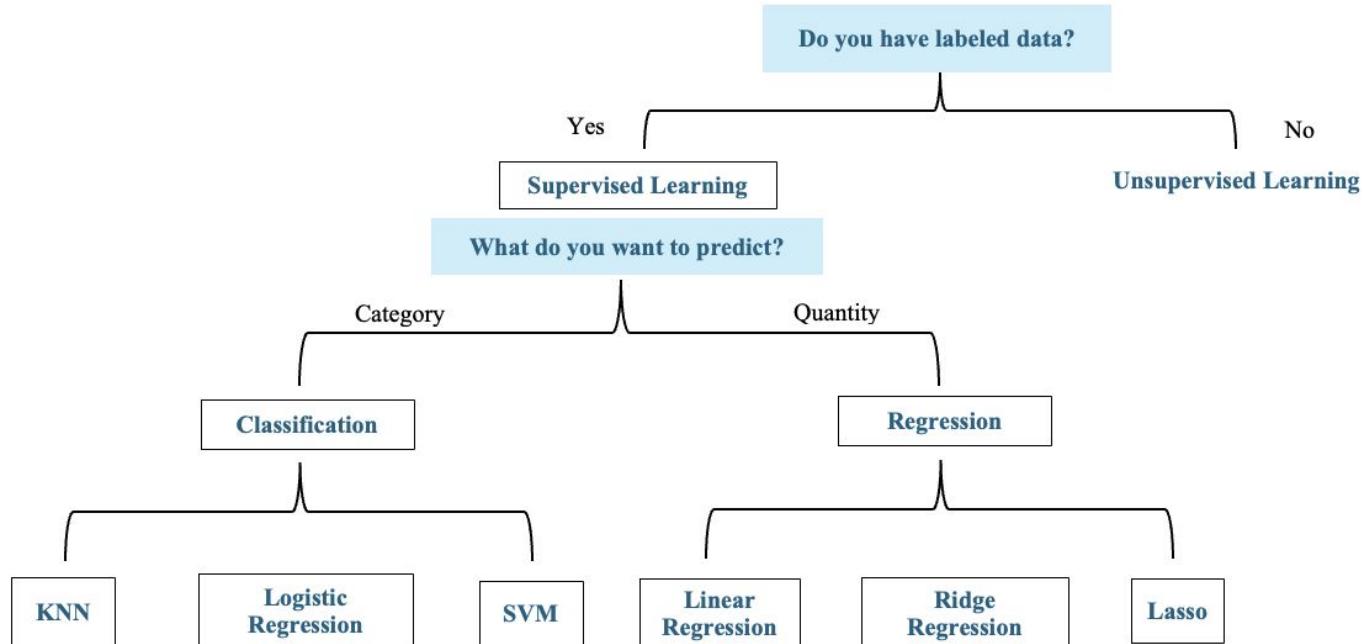
Week 7

SVM & Deep Learning

Content

- Support Vector Machines (SVM)
- Example: Algorithms in python
- Deep Learning Introduction
- Neural networks
- Python: deep learning in Keras, Pytorch, Tensorflow

Recap: Machine Learning Methods



Support Vector Machines (SVM)

Introduction to SVM

- Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks.
- Support vector machine (SVM) is a supervised method for binary classification (two class). It is a generalization of 2 classifiers below.
 - **Maximal margin classifier:** only applicable to linearly separable data.
 - **Support vector classifier:** can be applied to data that is not linearly separable. Decision boundary still linear.
- **Support vector machine:** non-linear decision boundary.
- SVM aims to find the optimal hyperplane that separates data points of different classes with the maximum margin.

Support Vector Machines (SVM)

Hyperplanes

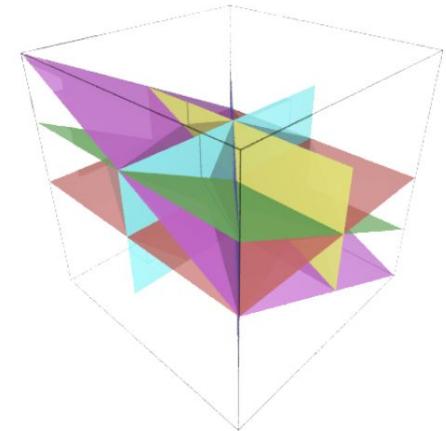
- In p-dimensional space, a hyperplane is a (p-1)-dimensional affine subspace.
 - In 2D, a hyperplane is a flat 1D subspace, aka a line.
 - In 3D, a hyperplane is a flat 2D subspace, aka a plane.
- Mathematical Definition - 2D hyperplane
 - A 2D hyperplane is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

- By “define”, we mean that any for which the above equation holds is a point on the hyperplane.
- Note that the above is the equation of a line, aka a hyperplane in 2D.
- Mathematical Definition - pD hyperplane
 - In p dimensions, a hyperplane is defined by the equation

$$\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p = 0$$

- Similarly, any for which the above equation holds is a point on the hyperplane.



Support Vector Machines (SVM)

Separating Hyperplane

- Instead of a point on the hyperplane, consider \mathbf{X} for which

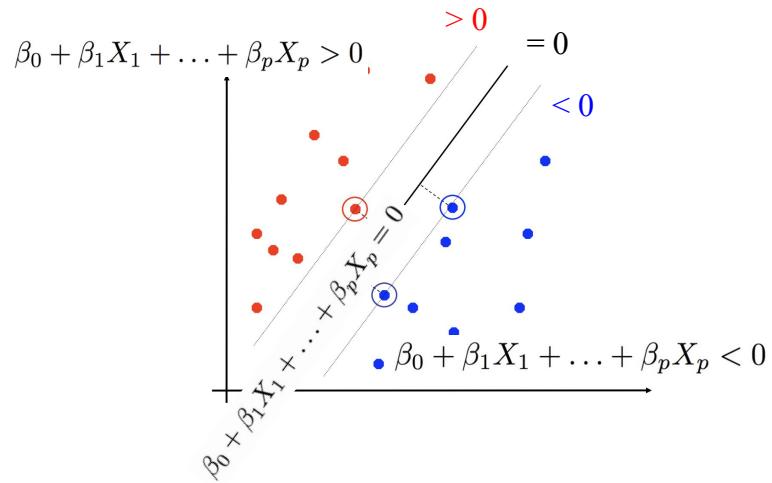
$$\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p > 0$$

- This point lies on one side of the hyperplane. An \mathbf{X} for which

$$\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p < 0$$

lies on the other side of the hyperplane.

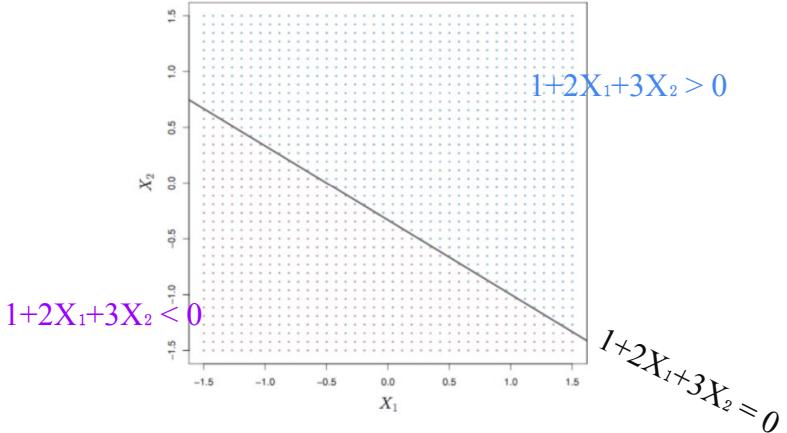
- We can think of the hyperplane as dividing the p -dimensional space into two halves.



Support Vector Machines (SVM)

Separating Hyperplane: Example

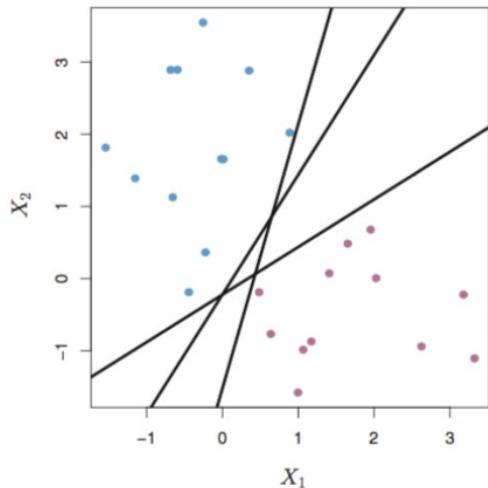
- This hyperplane in 2 dimensions is the line $1+2X_1+3X_2 = 0$.
- The blue region is the set of points for which $1+2X_1+3X_2 > 0$.
- The purple region is the set of points for which $1+2X_1+3X_2 < 0$.



Support Vector Machines (SVM)

Separating Hyperplane Classifier

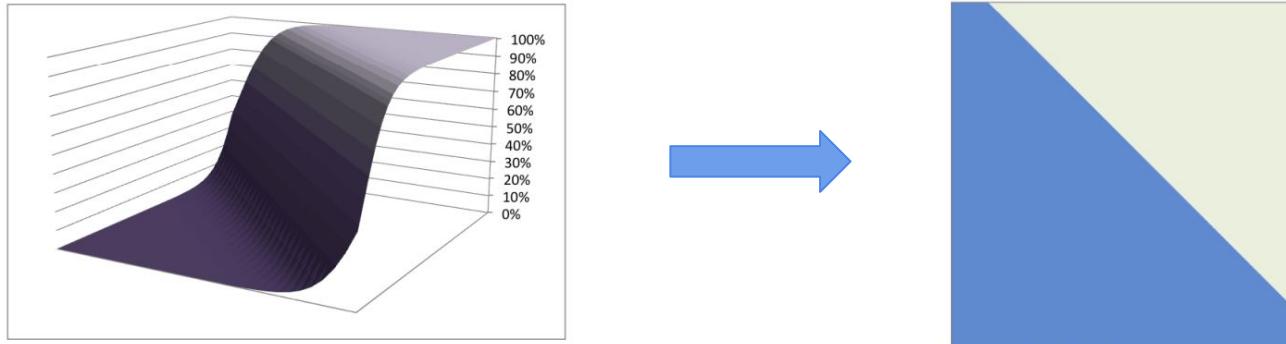
- **Idea:** Use a separating hyperplane for binary classification.
- **Key assumption:** Classes can be separated by a linear decision boundary



Support Vector Machines (SVM)

Separating Hyperplane Classifier

- **Aside:** Logistic regression effectively finds a separating hyperplane.



Maximal margin classifiers
and SVMs do this differently.

Support Vector Machines (SVM)

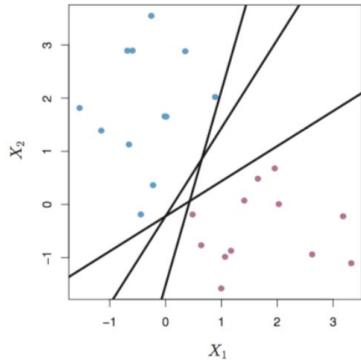
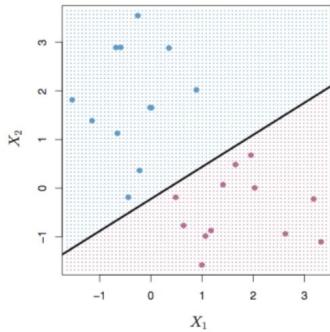
Separating Hyperplane Classifier

- To classify new data points: Assign class by location of new data point with respect to the hyperplane.

$$\hat{y} = \text{sign}(\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p)$$

- The farther away a point is from the separating hyperplane, the more confident we are about its class assignment.
- Notice that for a linearly separable dataset, there are **many possible separating hyperplanes** that divide the dataset into two classes (in fact, an infinite number).

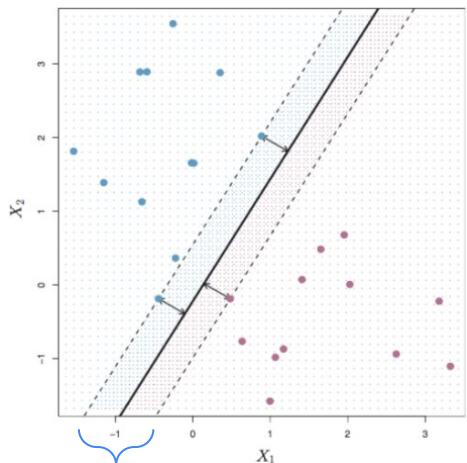
Q: Which decision boundary?



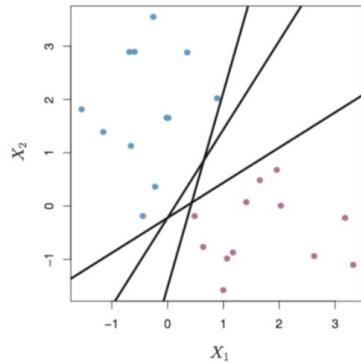
Support Vector Machines (SVM)

Maximal Margin Classifier

- Which of the infinite separating hyperplanes should we choose?
- A natural choice is the **maximal margin hyperplane**, the separating hyperplane that is farthest from the training samples.



Maximal Margin Hyperplane

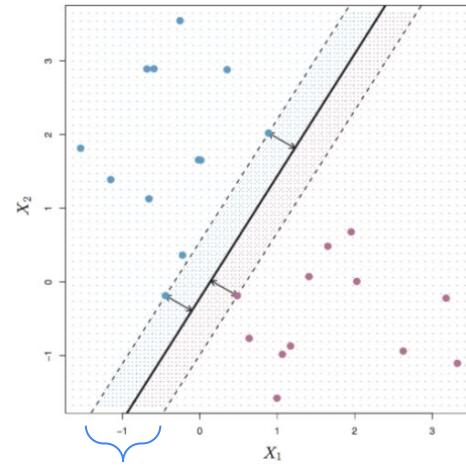


Q: Which decision boundary?

Support Vector Machines (SVM)

Maximal Margin (MM) Classifier

- **Margin:** smallest distance between any training observation and the hyperplane.
- **Support vectors:** training observations whose distance to the hyperplane is equal to the margin
- **Q: Why is it called a support vector?**
 - “Support”: maximal margin hyperplane only depends on these observations.
 - “Vector”: points are vectors in p-dimensional space.
- If support vectors are perturbed, then MM hyperplane will **change**.
- If other training observations perturbed (provided not perturbed within margin distance of hyperplane), then MM hyperplane **not affected**.



Maximal Margin Hyperplane

Support Vector Machines (SVM)

Finding Maximal Margin Classifier

- To find the maximal margin hyperplane on data $(\vec{x}^{(i)}, y^{(i)})$, $y^{(i)} \in \{-1, 1\}$ solve:

$$\max_{\beta_0, \dots, \beta_p} M$$

maximize the margin, M

subject to $\sum_{j=0}^p \beta_j^2 = 1$

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M, \quad \forall i$$

Support Vector Machines (SVM)

Finding Maximal Margin Classifier

- To find the maximal margin hyperplane on data $(\vec{x}^{(i)}, y^{(i)})$, $y^{(i)} \in \{-1, 1\}$ solve:

$$\max_{\beta_0, \dots, \beta_p} M$$

maximize the margin, M

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1$$

constraint necessary for
well-defined optimization problem

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M, \quad \forall i$$

Support Vector Machines (SVM)

Finding Maximal Margin Classifier

- To find the maximal margin hyperplane on data $(\vec{x}^{(i)}, y^{(i)})$, $y^{(i)} \in \{-1, 1\}$ solve:

$$\max_{\beta_0, \dots, \beta_p} M$$

maximize the margin, M

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1$$

constraint necessary for
well-defined optimization problem

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M, \quad \forall i$$

all training points must be at least distance M from hyperplane

Support Vector Machines (SVM)

Finding Maximal Margin Classifier

- To find the maximal margin hyperplane on data $(\vec{x}^{(i)}, y^{(i)})$, $y^{(i)} \in \{-1, 1\}$ solve:

Can be written as a convex optimization problem.

We know how to solve convex optimization problems efficiently to find M and β .

$$\max_{\beta_0, \dots, \beta_p} M$$

maximize the margin, M

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1$$

constraint necessary for well-defined optimization problem

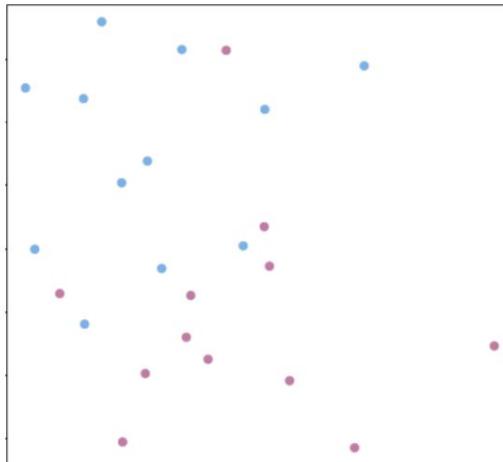
$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M, \quad \forall i$$

all training points must be at least distance M from hyperplane

Support Vector Machines (SVM)

Maximal Margin Classifier

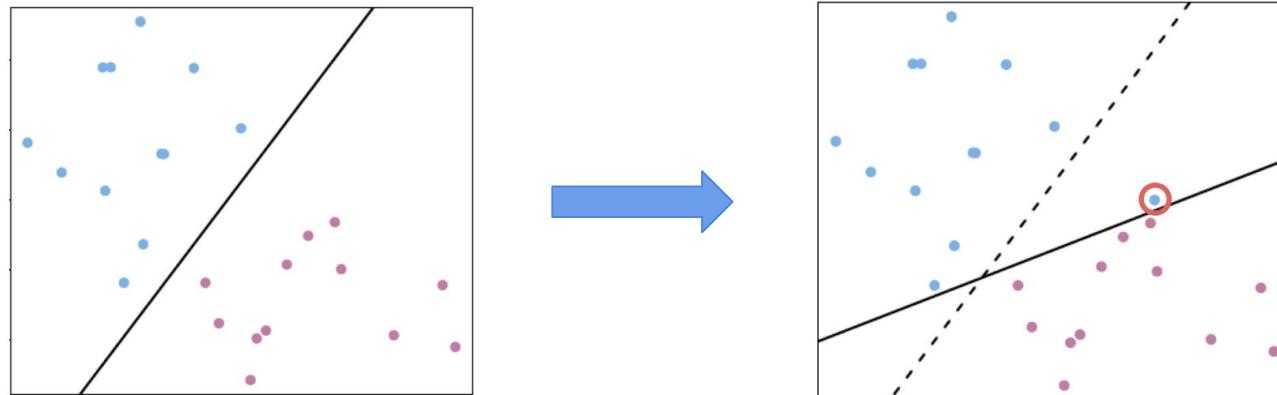
- Recall the assumption: Classes can be separated by a linear decision boundary.
- **Q: What if there is no separating hyperplane?**



Support Vector Machines (SVM)

Maximal Margin Classifier

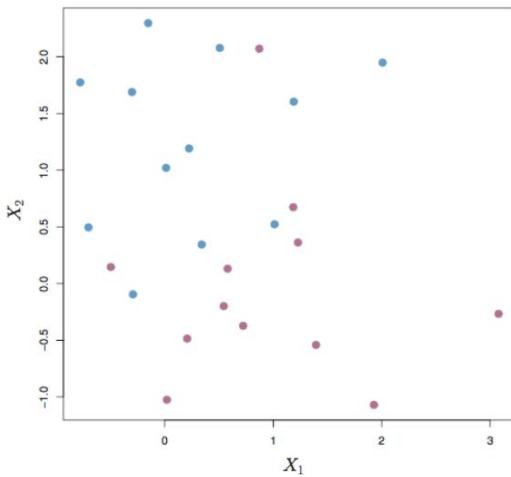
- Furthermore, notice a disadvantage of the maximal margin classifier:
 - Can be sensitive to individual observations
 - May overfit training data



Support Vector Machines (SVM)

Support Vector Classifier

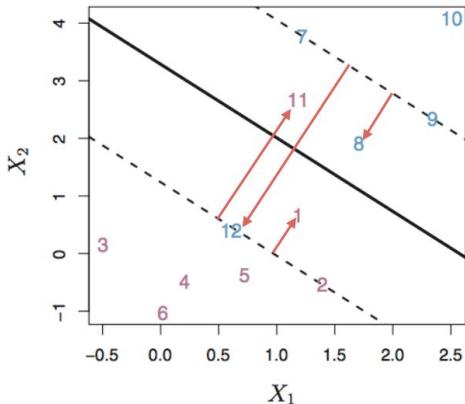
- Like the maximal margin classifier, it looks for a hyperplane to perform classification.



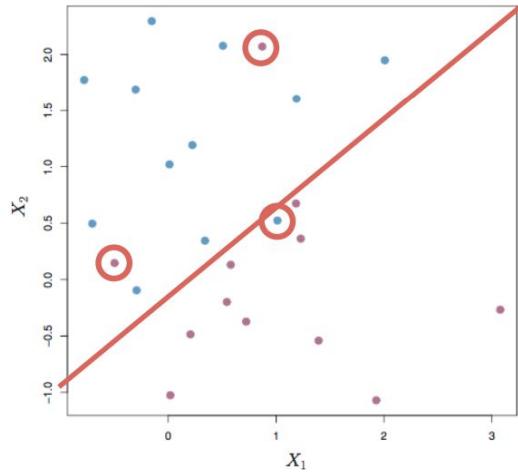
Support Vector Machines (SVM)

Support Vector Classifier

- Like the maximal margin classifier, it looks for a hyperplane to perform classification.
- However, training samples are allowed to be on the “**wrong side**” of the margin or hyperplane.
- This hyperplane almost separates the classes using a “**soft margin**”.



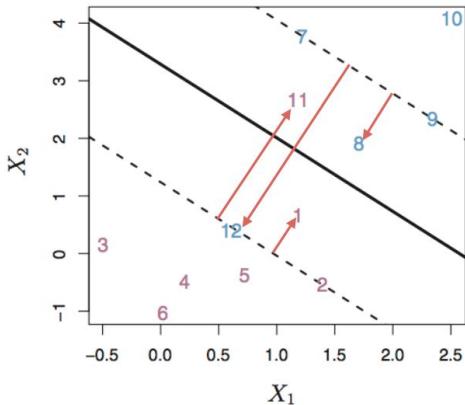
Some points are allowed to violate the margin.



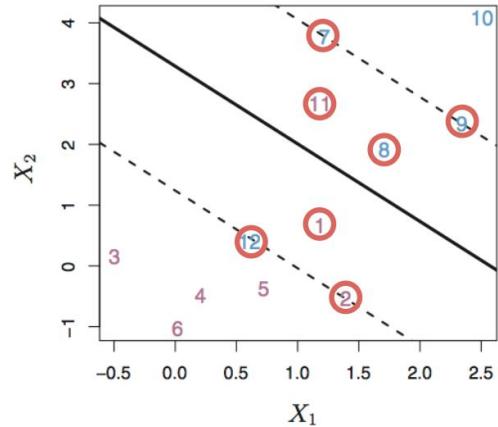
Support Vector Machines (SVM)

Support Vector Classifier

- Support vector classifiers also have support vectors.
- They are points lying directly on the margin, or on the wrong side of the margin for their class.
- These observations affect the hyperplane.



Some points are allowed to violate the margin.



Support Vector Machines (SVM)

Finding Support Vector Classifier

- To find the support vector classifier hyperplane, solve:

$$\max_{\beta_0, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n} M \quad \text{maximize the margin, } M$$

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1$$

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M(1 - \epsilon_i), \quad \forall i$$

$$\sum_{i=1}^n \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \forall i$$

Support Vector Machines (SVM)

Finding Support Vector Classifier

- To find the support vector classifier hyperplane, solve:

$$\max_{\beta_0, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n} M \quad \text{maximize the margin, } M$$

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1 \quad \text{constraint necessary for well-defined optimization problem}$$

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M(1 - \epsilon_i), \quad \forall i$$

$$\sum_{i=1}^n \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \forall i$$

Support Vector Machines (SVM)

Finding Support Vector Classifier

- To find the support vector classifier hyperplane, solve:

$$\max_{\beta_0, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n} M \quad \text{maximize the margin, } M$$

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1 \quad \text{constraint necessary for well-defined optimization problem}$$

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M(1 - \epsilon_i), \quad \forall i$$

$$\sum_{i=1}^n \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \forall i$$

training points must be at least distance M from hyperplane, or pay a penalty ϵ_i

Support Vector Machines (SVM)

Finding Support Vector Classifier

- To find the support vector classifier hyperplane, solve:

$$\max_{\beta_0, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n} M \quad \text{maximize the margin, } M$$

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1 \quad \text{constraint necessary for well-defined optimization problem}$$

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M(1 - \epsilon_i), \quad \forall i$$

training points must be at least distance M from hyperplane, or pay a penalty ϵ_i

$$\sum_{i=1}^n \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \forall i \quad \text{“slack” variable } \epsilon_i$$

Support Vector Machines (SVM)

Finding Support Vector Classifier

- To find the support vector classifier hyperplane, solve:

$$\max_{\beta_0, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n} M \quad \text{maximize the margin, } M$$

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1 \quad \text{constraint necessary for well-defined optimization problem}$$

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M(1 - \epsilon_i), \quad \forall i$$

training points must be at least distance M from hyperplane, or pay a penalty ϵ_i

$$\sum_{i=1}^n \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \forall i \quad \text{“slack” variable } \epsilon_i$$

limit on total penalties. C is a constant.

Support Vector Machines (SVM)

Finding Support Vector Classifier

- To find the support vector classifier hyperplane, solve:

In other words, you can violate the margin, but only by a total amount C on your entire dataset

$$\max_{\beta_0, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n} M \quad \text{maximize the margin, } M$$

$$\text{subject to } \sum_{j=0}^p \beta_j^2 = 1 \quad \text{constraint necessary for well-defined optimization problem}$$

$$y^{(i)}(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}) \geq M(1 - \epsilon_i), \quad \forall i$$

training points must be at least distance M from hyperplane, or pay a penalty ϵ_i

$$\sum_{i=1}^n \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \forall i \quad \text{“slack” variable } \epsilon_i$$

limit on total penalties. C is a constant.

Support Vector Machines (SVM)

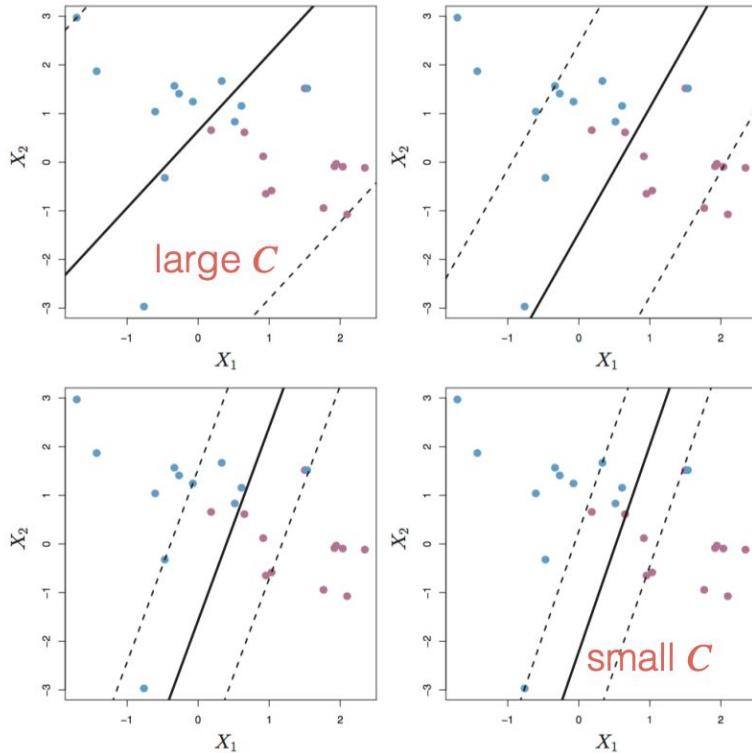
Support Vector Classifier

- Slack variables ϵ_i allow for violations of the margin.
 - $\epsilon_i = 0$: training point is on correct side of margin
 - $\epsilon_i > 0$: training point violates the margin
 - $\epsilon_i > 1$: training point is misclassified (wrong side of hyperplane)
- Penalty parameter C is the total “budget” for violations.
 - Allows at most C misclassifications on training set.
- Q: How do we choose C ?
 - As with many things we don’t know a priori in machine learning, C is a hyperparameter that we tune using cross-validation.
 - Note that it must be non-negative.
 - If $C = 0$, we recover the maximal margin classifier (if one exists).
 - As C goes from small to large, there is a bias-variance tradeoff.

Support Vector Machines (SVM)

Support Vector Classifier

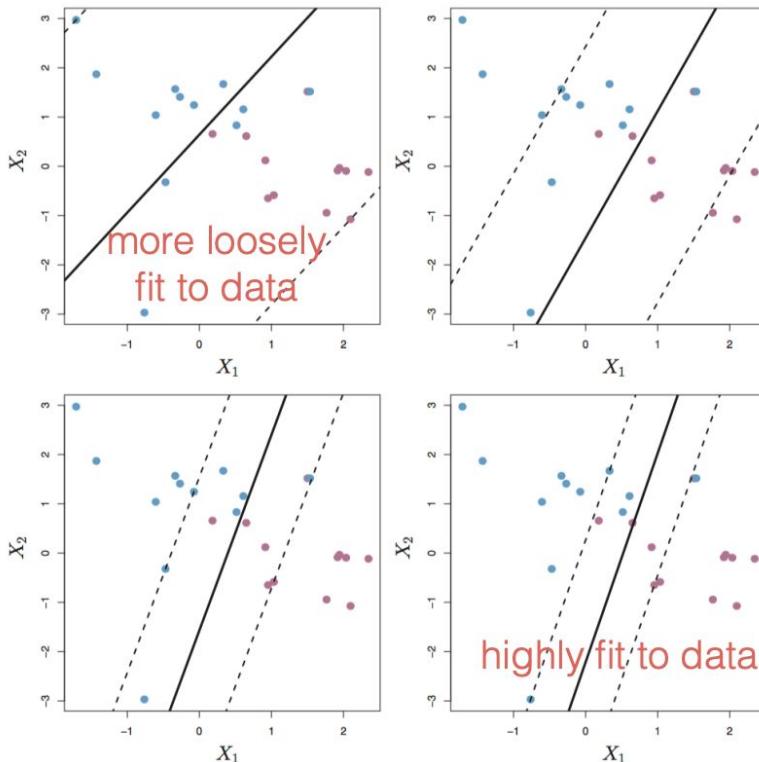
- Bias, Variance and C
 - Large C
 - Large violation budget
 - Large margin
 - Many support vectors
 - Small C
 - Small violation budget
 - Small margin
 - Few support vectors



Support Vector Machines (SVM)

Support Vector Classifier

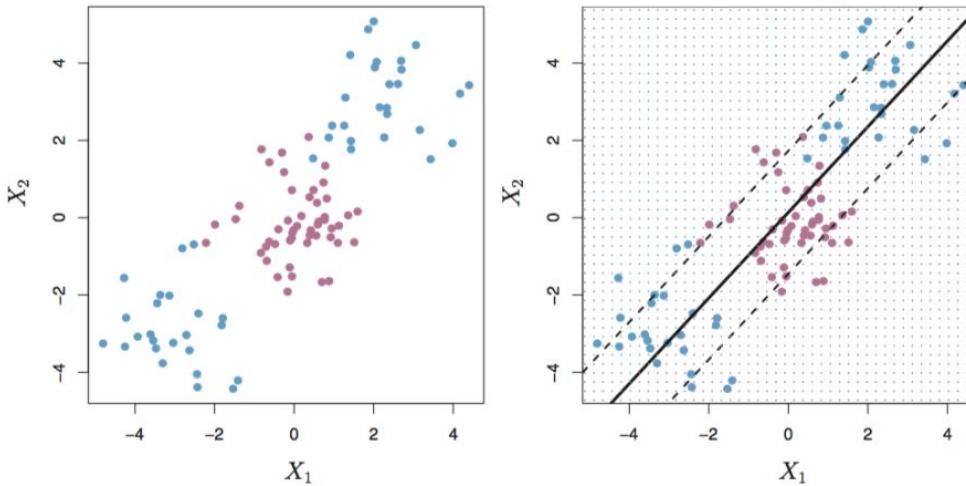
- Bias, Variance and C
 - Large C
 - High bias
 - Low variance
 - Small C
 - Low bias
 - High variance



Support Vector Machines (SVM)

Support Vector Classifier

- We are still using a linear decision boundary.



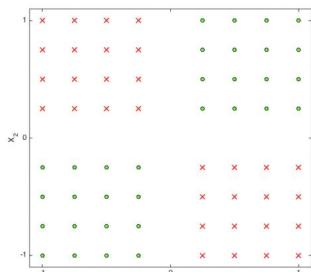
Support Vector Machines (SVM)

Support Vector Classifier

- **Expanding Feature Space**

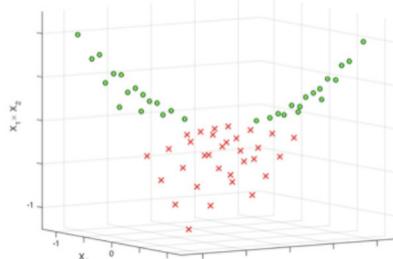
- Some datasets are not linearly separable, but they become linearly separable when transformed into a higher dimensional space.
- (Note: Yes, higher dimension also increases chance of overfitting. But in some cases the tradeoff is worthwhile.)

Original feature space

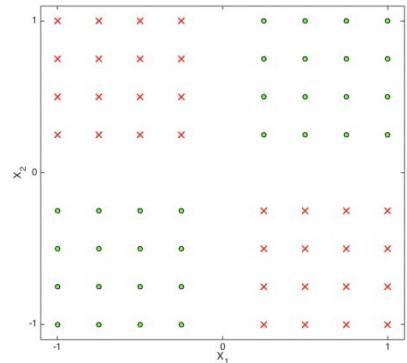


variables x_1, x_2

New feature space



variables x_1, x_2, x_1x_2



Support Vector Machines (SVM)

Support Vector Classifier

- **Expanding Feature Space**

- Recall that in linear regression, we created new features to capture non-linearity of data.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \epsilon$$

- We can apply the same technique to support vector classifiers.
 - Suppose our original data has p features.

$$\vec{X} = (X_1, X_2, \dots, X_p)$$

- We can expand the feature space to include e.g. $2p$ features.

$$\tilde{\vec{X}} = (X_1, X_1^2, X_2, X_2^2, \dots, X_p, X_p^2)$$

$$= (\tilde{X}_1, \tilde{X}_2, \tilde{X}_3, \tilde{X}_4, \dots, \tilde{X}_{2p-1}, \tilde{X}_{2p})$$

Support Vector Machines (SVM)

Support Vector Classifier

- **Expanding Feature Space**

- Support vector classifier will find a hyperplane in $2p$ dimensions.

$$\beta_0 + \beta_1 \tilde{X}_1 + \beta_2 \tilde{X}_2 + \dots + \beta_{2p-1} \tilde{X}_{2p-1} + \beta_{2p} \tilde{X}_{2p} = 0$$

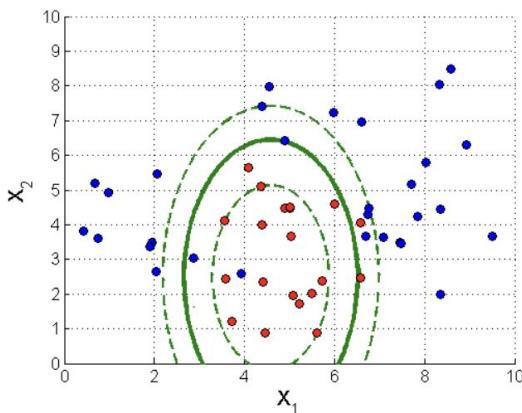
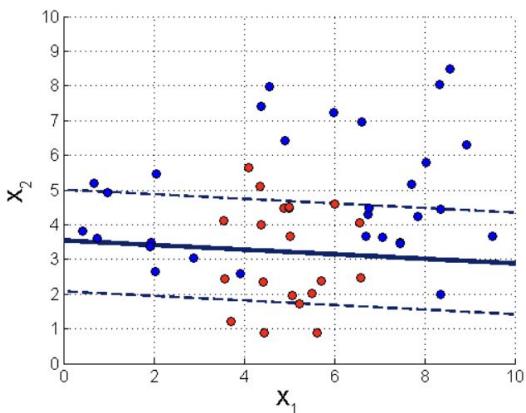
- Hyperplane will be non-linear in original feature space. In this case, it is an ellipse.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \dots + \beta_{2p-1} X_p + \beta_{2p} X_p^2 = 0$$

Support Vector Machines (SVM)

Support Vector Classifier

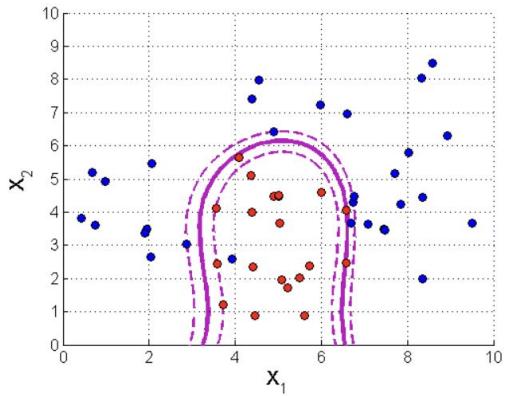
- Non-linear Decision Boundary



Support Vector Machines (SVM)

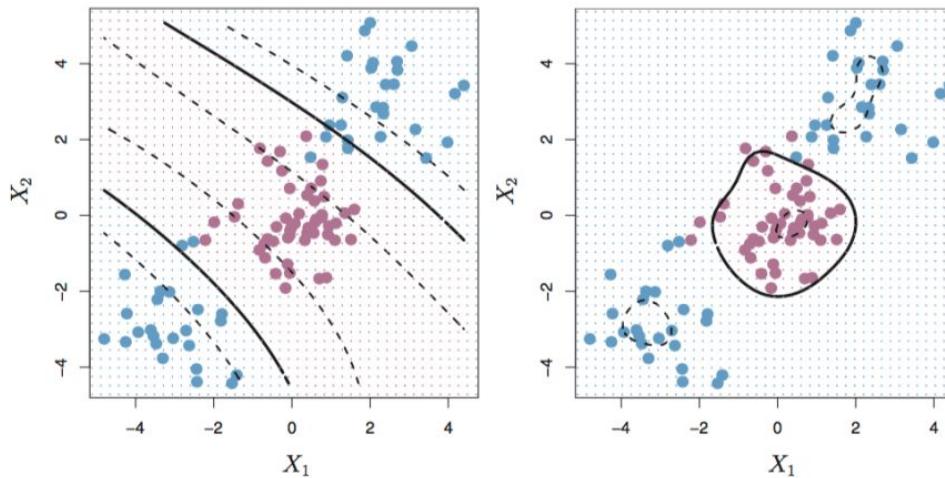
Support Vector Classifier

- **Expanding Feature Space**
 - Support vector classifier will find a decision boundary. Can imagine adding higher order polynomial terms, quotients, and more to expand features set. in 2p dimensions.
 - Large number of features becomes computationally challenging.
 - We need an efficient way to work with large number of features.
 - [Support Vector Machines](#)



Support Vector Machines (SVM)

Extends the support vector classifier by using **kernel functions** to achieve **non-linear decision boundaries**.



Support Vector Machines (SVM)

Kernel function

- **Kernel function:** generalization of inner product. It takes in two arguments and implicitly computes their inner product in some feature space.
- Kernels are an efficient computational approach to create **non-linear decision boundaries**.
- They (implicitly) map data into a higher-dimensional space.
- We then apply a support vector classifier in this high-dimensional space with a linear decision boundary (hyperplane).

Support Vector Machines (SVM)

Linear SVM

- It can be shown that a support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

Support Vector Machines (SVM)

Linear SVM

- It can be shown that a support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle \quad \text{inner product}$$

recall: $f(x) > 0$ is one class
 $f(x) < 0$ is another

S is the set of
support vectors

Where $\langle \vec{u}, \vec{v} \rangle = \sum_{j=1}^p u_j v_j$

Support Vector Machines (SVM)

General SVM

- It can be shown that a support vector classifier can be represented as tor classifier can be represented as

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

- In SVM we replace the inner product with some kernel function

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x^{(i)}, x)$$

Support Vector Machines (SVM)

Properties of Kernels

- Generalization of inner product:

for an explicit feature map $\phi : \mathcal{X} \rightarrow \mathcal{X}^\phi$

$$x \mapsto \phi(x)$$

$$K(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{X}^\phi}$$

- Symmetric: $K(x, x') = K(x', x)$
- Gives a measure of similarity between \mathbf{X} and \mathbf{X}'
 - If \mathbf{X} and \mathbf{X}' are close together, then large
 - If \mathbf{X} and \mathbf{X}' are far apart, then small

For a more formal definition:
<http://mlweb.loria.fr/book/en/constructingkernels.html>

Support Vector Machines (SVM)

Common SVM Kernels

- **Linear kernel**

$$K(x, x') = \langle x, x' \rangle$$

- **Polynomial kernel (degree p)**

$$K(x, x') = (1 + \langle x, x' \rangle)^p$$

- **Radial basis kernel**

$$K(x, x') = \exp(-\gamma ||x - x'||^2)$$

(an infinite-dimensional feature map!)

Support Vector Machines (SVM)

Q: Why use kernels?

- Why use kernels instead of explicitly constructing a larger feature space?
 - Computational advantage:

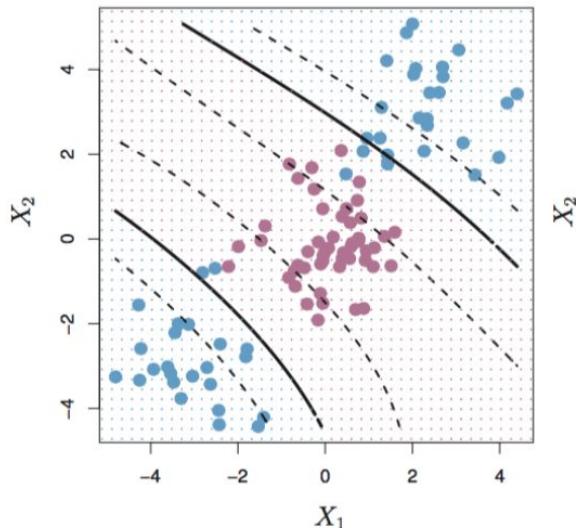
$$\phi : \mathbb{R}^p \rightarrow \mathbb{R}^P, \quad p \ll P$$

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \quad \text{in } O(p)$$

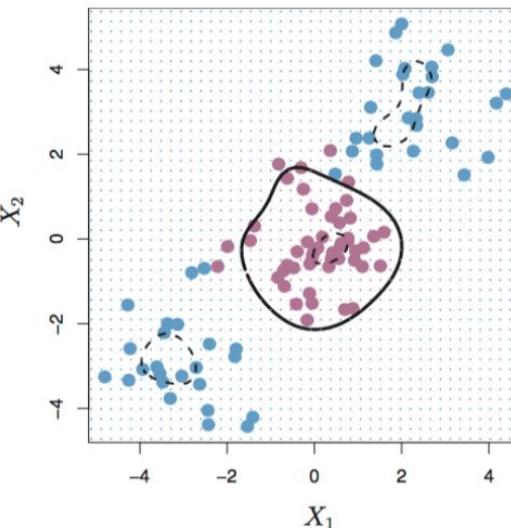
Support Vector Machines (SVM)

SVM with Non-Linear Kernels

Cubic polynomial kernel



Radial kernel



Support Vector Machines (SVM)

SVM Summary

- Pros:
 - Regularization parameter C helps avoid overfitting
 - Use of kernel gives flexibility in shape of decision boundary
 - Optimization problem is convex — unique solution
- Cons:
 - Must tune hyperparameters (e.g. C, kernel function)
 - Must formulate as binary classification
 - Difficult to interpret

Support Vector Machines (SVM)

SVM with 3+ Classes

- SVMs are designed for binary classification, given the nature of a separating hyperplane.
- We can adapt SVMs to perform classification when we have more than 2 classes.
- Popular approaches:
 - One-versus-one
 - One-versus-all

Support Vector Machines (SVM)

One-versus-one Classification

- Construct an SVM for each pair of classes. For k classes, this requires training $k(k-1)/2$ SVMs.
- To classify a new observation, apply all $k(k-1)/2$ SVMs to the observation.
- Take the most frequent class among pairwise results as the predicted class.
- **Con:** computationally expensive for large k .

Support Vector Machines (SVM)

One-versus-all Classification

- Construct an SVM for each class against the k-1 other classes pooled together.
- For k classes, this requires training k SVMs.
- Distance to separating hyperplane is a proxy for confidence of classification. For new observation, choose the “highest confidence” class as the prediction.
- **Con:** may exacerbate class imbalances, distance to hyperplane may not correspond well to confidence.

Example: Algorithms in python

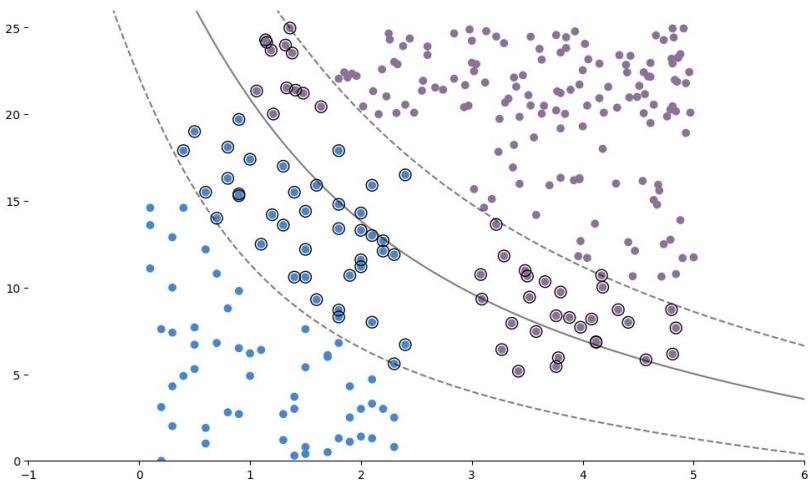
SVM: 2nd degree polynomial Kernel

```
from sklearn import svm
model = svm.SVC(kernel='poly', degree=2)
model.fit(x_train, y_train)
```

✓ 0.4s

SVC

SVC(degree=2, kernel='poly')



Example: Algorithms in python

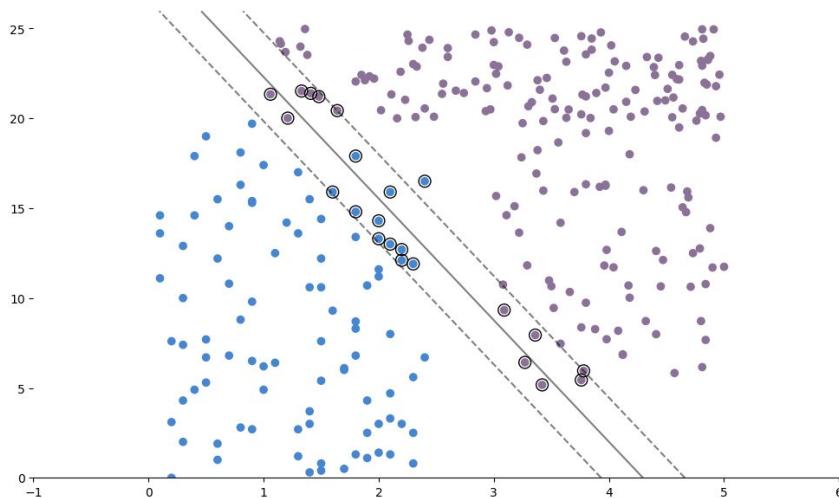
SVM: Linear SVM

```
model = svm.SVC(kernel='linear')
model.fit(x_train, y_train)
```

✓ 0.0s

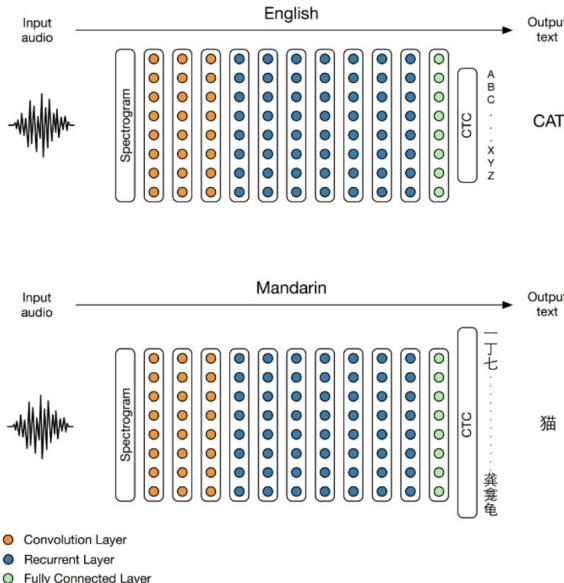
▼ SVC  

```
SVC(kernel='linear')
```



Deep Learning Introduction

DL Today: Speech-to-Text



[Baidu 2014]

Deep Learning Introduction

DL Today: Computer Vision (CV)



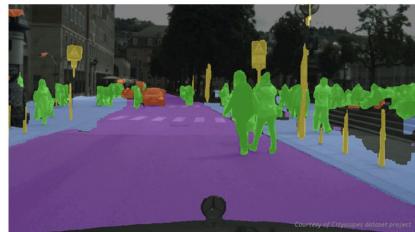
[Krizhevsky 2012]



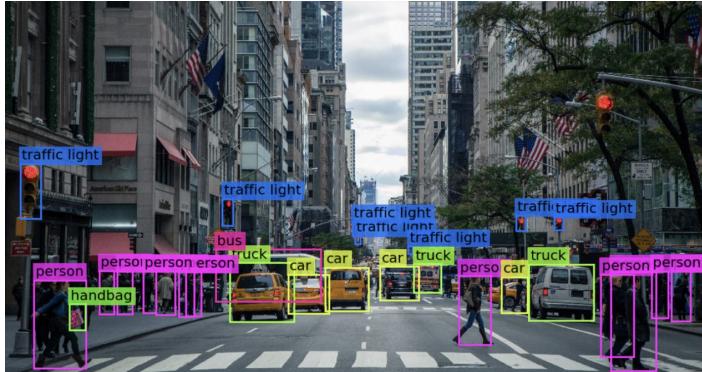
[Ciresan et al. 2013]



[Faster R-CNN - Ren 2015]



[NVIDIA dev blog]



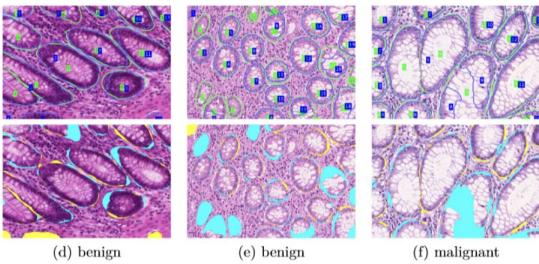
[Grokking Deep Learning for Computer Vision Book]

Deep Learning Introduction

DL Today: Computer Vision (CV)



[Stanford 2017]



[Nvidia Dev Blog 2017]

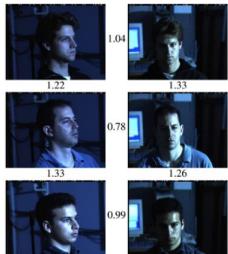
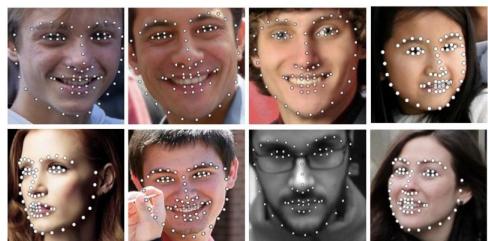
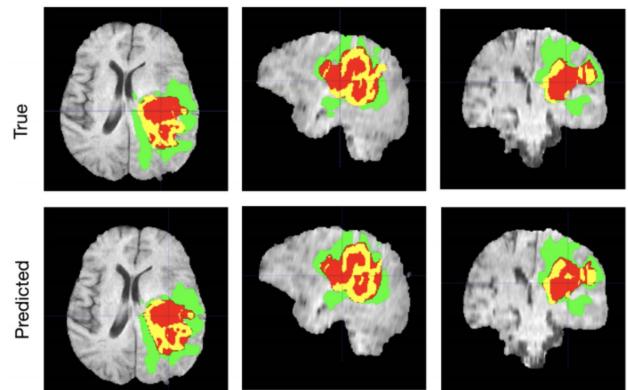


Figure 1. Illumination and Pose invariance.

[FaceNet - Google 2015]



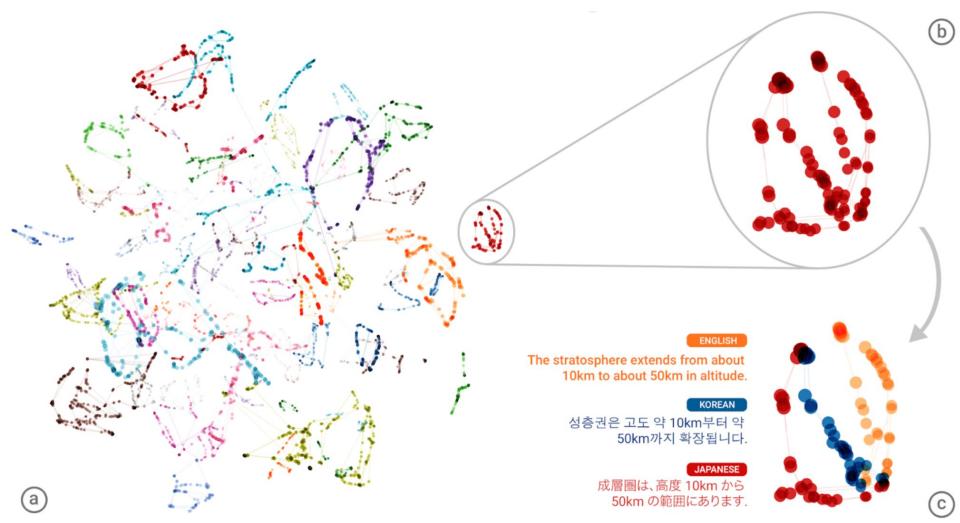
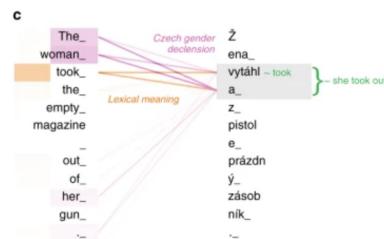
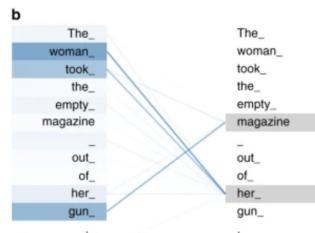
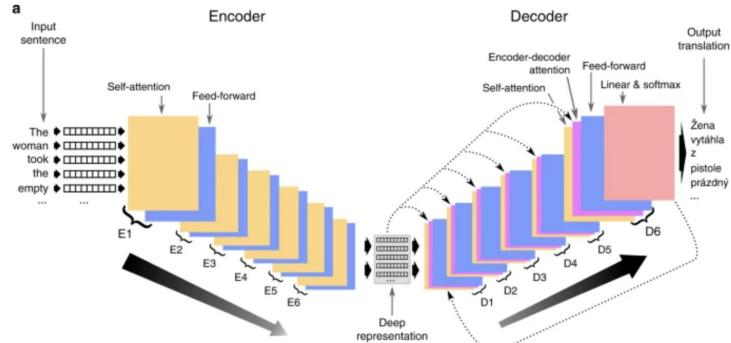
[Facial landmark detection CUHK 2014]



[Brain tumor segmentation with AI. Source: Nvidia. Developer]

Deep Learning Introduction

DL Today: Natural Language Processing (NLP)



Popel, M., Tomkova, M., Tomek, J., et al. (2020). Transforming machine translation: A deep learning system reaches news translation quality comparable to human professionals. *Nature Communications*, 11, 4381. <https://doi.org/10.1038/s41467-020-18073-9>

Deep Learning Introduction

DL Today: Natural Language Processing (NLP)



[Google Inbox Smart Reply]



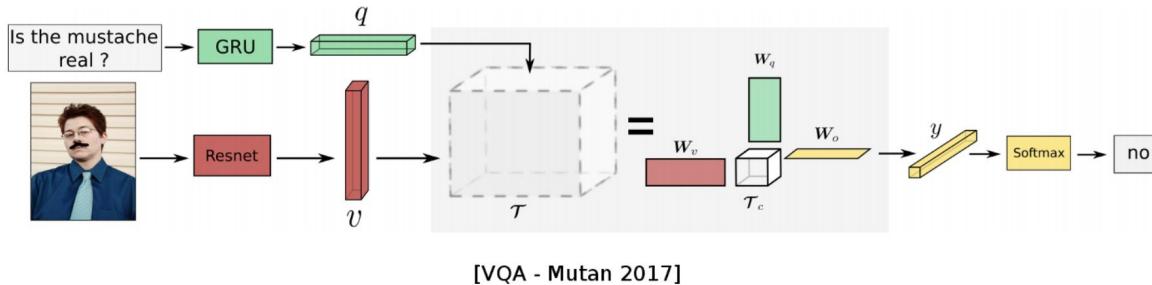
[Amazon Echo / Alexa]



[Blog: Best AI Chatbots in 2024]

Deep Learning Introduction

DL Today: CV + NLP



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."

[Karpathy 2015]

Deep Learning Introduction

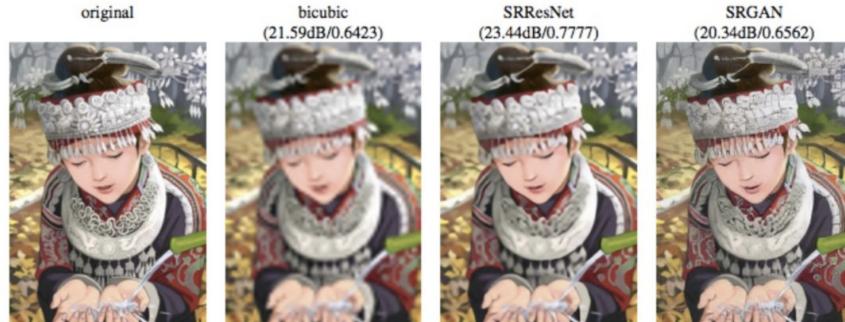
DL Today: Image translation



[DeepDream 2015]



[Gatys 2015]



[Ledig 2016]



Copilot

✓ Analyzing the image: Privacy blur hides faces from Copilot

The image shows a tranquil water canal flowing through a traditional Chinese architectural setting. The buildings feature multi-tiered, curved roofs with upturned eaves, typical of historical Chinese architecture. A gently arched bridge spans the canal, and the reflections of the buildings are visible on the water's surface. The clear sky and soft lighting suggest it might be late afternoon. Red lanterns hanging from the buildings add to the cultural ambiance. There are a few people on the bridge and near the water's edge, bringing a sense of scale and liveliness to the serene environment.

It looks like a beautiful and peaceful place! Is there anything specific you would like to know about this scene?

Deep Learning Introduction

DL Today: Generative models



[Generated by Canva AI]

Deep Learning Introduction

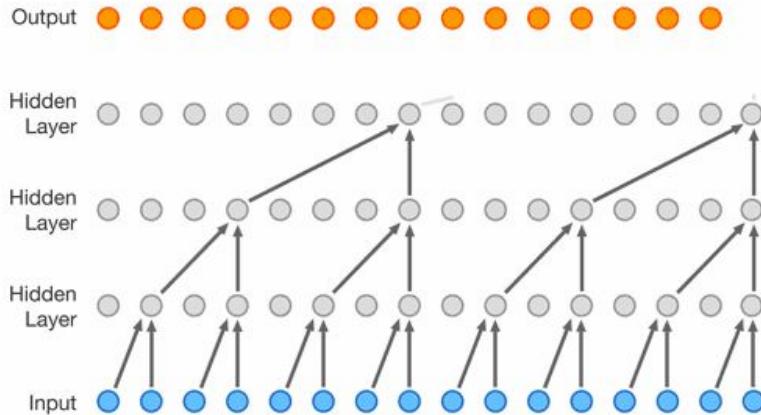
DL Today: Generative models

Text description	This bird is blue with white and has a very short beak	This bird has wings that are brown and has a yellow belly	A white bird with a black crown and yellow beak	This bird is white, black, and brown in color, with a brown beak	The bird has small beak, with reddish brown crown and gray belly	This is a small, black bird with a white breast and white on the wingbars.	This bird is white black and yellow in color, with a short black beak
Stage-I images							
Stage-II images							

StackGAN v2 [Zhang 2017]

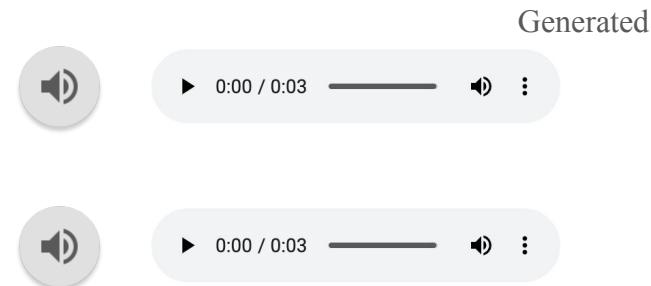
Deep Learning Introduction

DL Today: Generative models



Sound generation with WaveNet [DeepMind 2017]

Guess which one is generated?

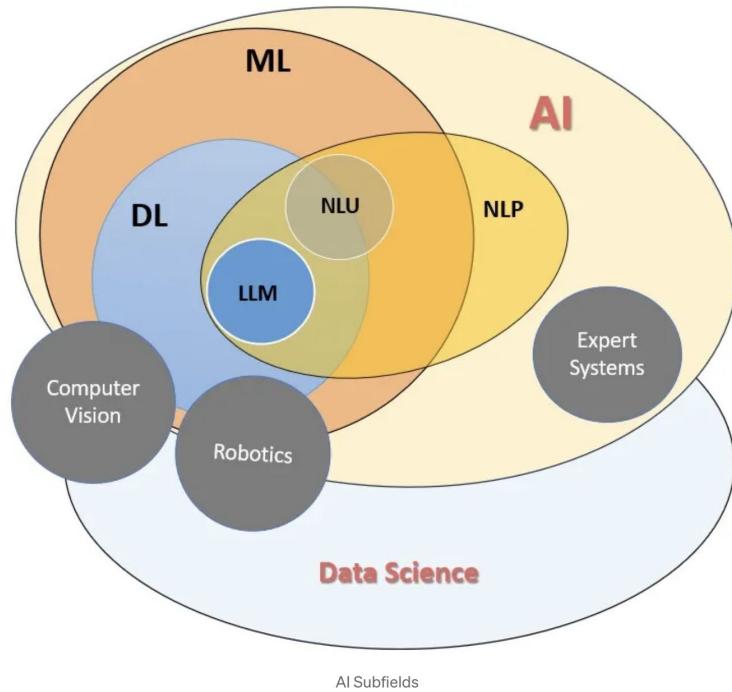


Tacotron 2 Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions, 2017

Deep Learning Introduction

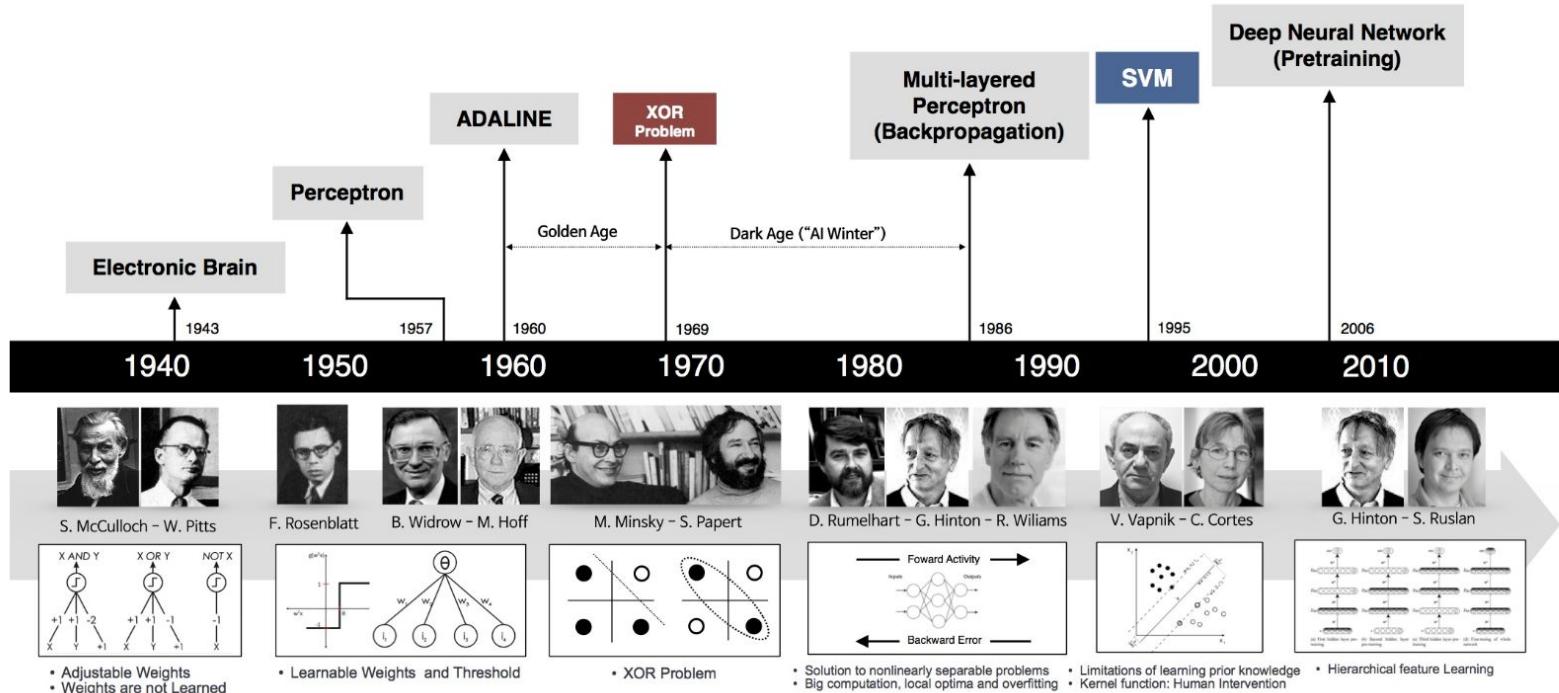
Recap: The Landscape of Artificial Intelligence

- Natural Language Processing (NLP)
- Natural Language Understanding (NLU)
- Large Language Models (LLM)
- Computer Vision
- Robotics
- Expert Systems
- Data Science



Deep Learning Introduction

History and Background



Deep Learning Introduction

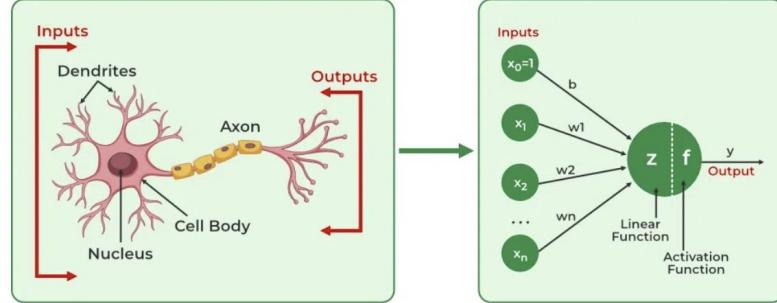
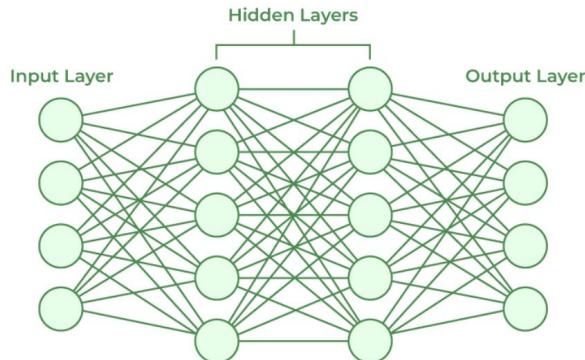
History and Background

- **Early Beginnings:** The concept of neural networks dates back to the 1940s with the work of Warren McCulloch and Walter Pitts, who created a computational model for neural networks.
- **Perceptron and Initial Hurdles:** In the 1950s, Frank Rosenblatt developed the perceptron, an early type of neural network. However, limitations in computational power and the inability to solve non-linear problems led to a decline in interest.
- **Backpropagation and Renewed Interest:** The development of the backpropagation algorithm in the 1980s by Geoffrey Hinton and others reignited interest in neural networks, allowing for more effective training of multi-layer networks.
- **Deep Learning Emergence:** In the 2000s, advancements in computational power, availability of large datasets, and improved algorithms led to the rise of deep learning. Techniques like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) became prominent.
- **Modern Applications:** Today, deep learning is applied in various fields, including image and speech recognition, natural language processing, and autonomous systems, driving significant advancements in artificial intelligence.

Neural networks

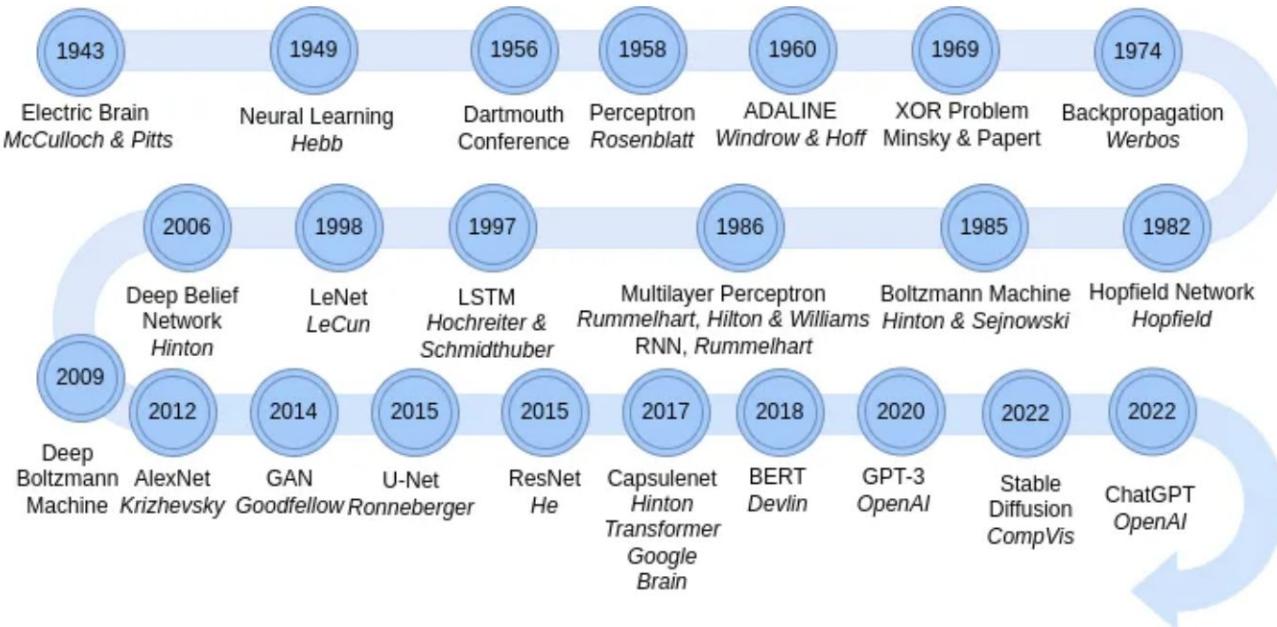
Artificial Neural Networks

- **Artificial Neural Networks (ANN)** are among the oldest and most intensely studied Machine Learning approaches
- Inspired by biological neural networks, they mimic the learning process of animals and humans.
- However, the model is very coarse, and several improvements to the basic approach have even abandoned the biological analogy.
- **Advantage:** Highly flexible, good performance
- **Disadvantage:** Black-box models not easy to interpret.



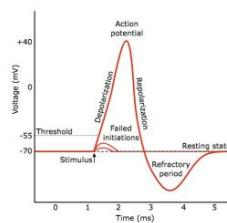
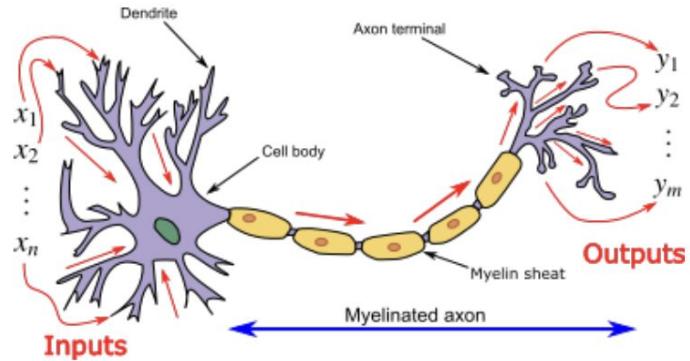
Neural networks

A Brief History of Neural Nets

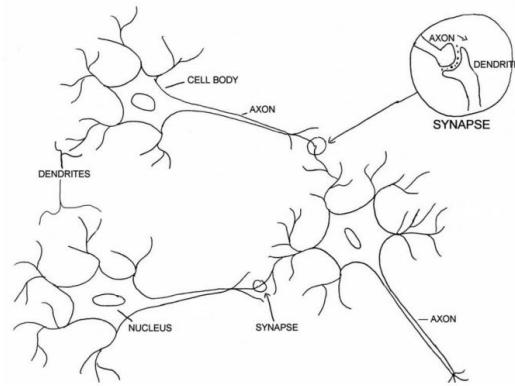


Neural networks

Biological Neuron vs. Perceptron



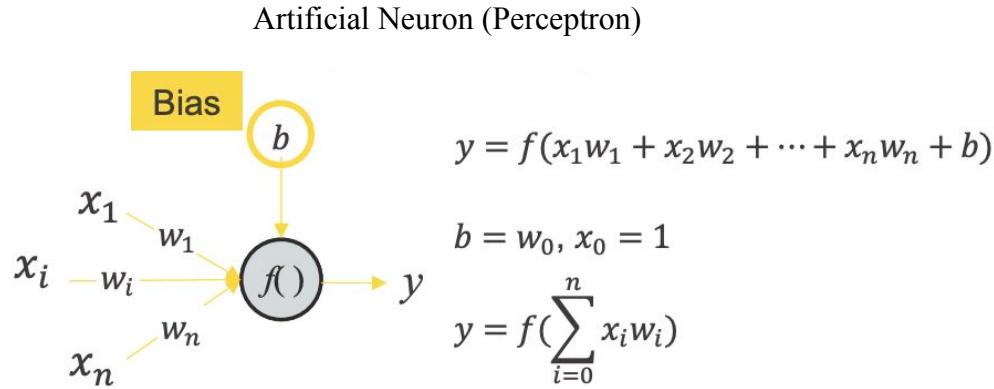
Biological Neuron



Biological Neural Networks

Neural networks

Biological Neuron vs. Perceptron

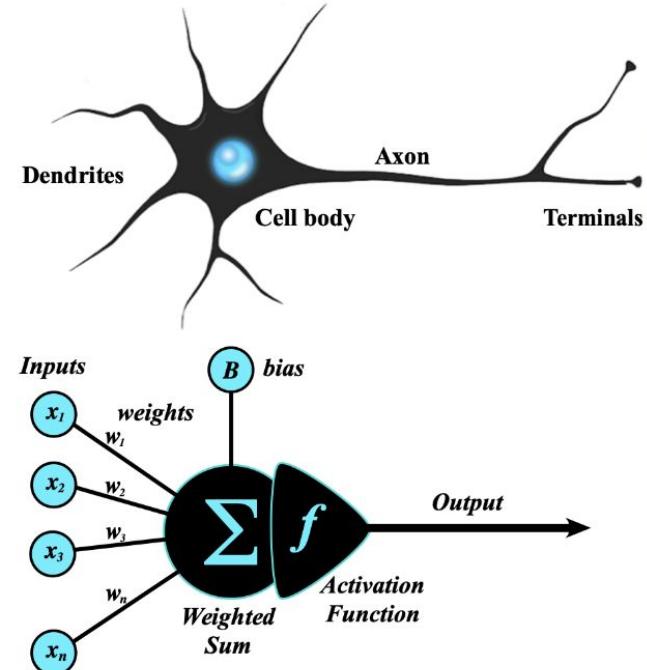


Input:

- numerical attributes

Output:

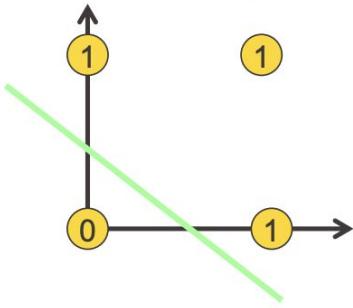
- Probability (0,1)
- Numerical outcome $(-\infty, \infty)$



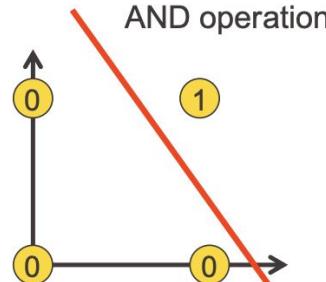
Neural networks

What Can a Single Perceptron Do?

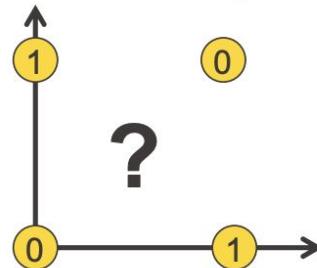
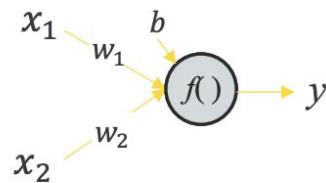
OR operation



AND operation

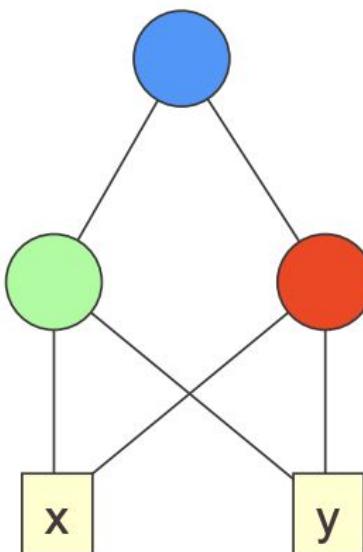
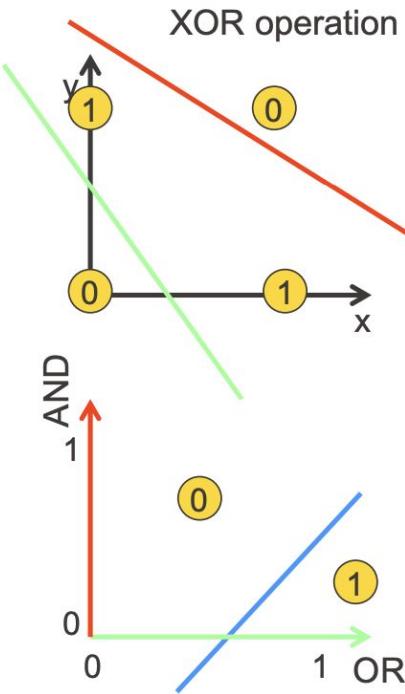


XOR operation



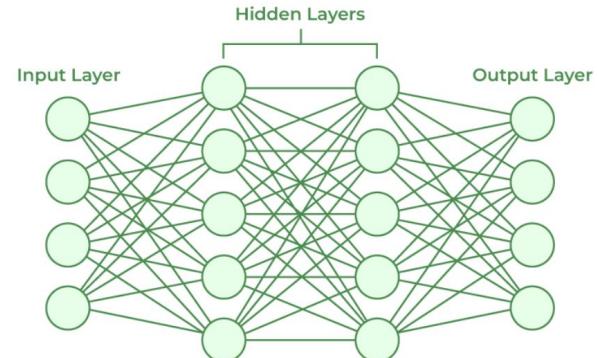
Neural networks

What Can a 3-neuron MLP Do?



Next lecture: Feedforward Neural Networks

- A network of artificial neurons with multiple layers



Python: deep learning in Keras, Pytorch, Tensorflow

Libraries & Frameworks



PYTORCH



Microsoft
CNTK

Caffe2

dmlc
mxnet

gensim

spaCy

theano

Python: deep learning in Keras, Pytorch, Tensorflow

Libraries & Frameworks

- **TensorFlow**
 - Developed by Google Brain Team.
 - Known for its flexibility and comprehensive ecosystem.
 - Supports both high-level APIs (like Keras) and low-level operations.
- **PyTorch**
 - Developed by Facebook's AI Research lab.
 - Known for its dynamic computation graph, which makes it more flexible and easier for debugging.
 - Widely used in research and academia.
- **Keras**
 - Initially developed as an independent library, now integrated with TensorFlow.
 - High-level API that simplifies building and training deep learning models.
 - User-friendly and modular.

Python: deep learning in Keras, Pytorch, Tensorflow

Key features and differences

- **TensorFlow**: Offers both high-level and low-level APIs, making it suitable for both beginners and advanced users. It has a comprehensive ecosystem for deploying models in production.
- **PyTorch**: Preferred for research due to its dynamic computation graph, which allows for more flexibility and ease of debugging.
- **Keras**: Simplifies model building with its high-level API, making it ideal for quick prototyping and beginners.

This lecture is using **Keras**: high level frontend for **TensorFlow** (and MXnet, Theano, CNTK)

One TA lecture will be dedicated to a short Pytorch introduction.

Python: deep learning in Keras, Pytorch, Tensorflow

Key features and differences

- **TensorFlow**: Offers both high-level and low-level APIs, making it suitable for both beginners and advanced users. It has a comprehensive ecosystem for deploying models in production.
- **PyTorch**: Preferred for research due to its dynamic computation graph, which allows for more flexibility and ease of debugging.
- **Keras**: Simplifies model building with its high-level API, making it ideal for quick prototyping and beginners.

This lecture is using **Keras**: high level frontend for **TensorFlow** (and MXnet, Theano, CNTK)

One TA lecture will be dedicated to a short Pytorch introduction.

Python: deep learning in Keras, Pytorch, Tensorflow

Example: TensorFlow

```
model = Sequential()
model.add(Dense(32, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

✓ 0.0s

```
model.fit(X_train, y_train, epochs=50, batch_size=10, validation_split=0.2)
```

✓ 1.8s

```
Epoch 1/50
57/57 ━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5359 - loss: 0.6828 - val_accuracy: 0.7552 - val_loss: 0.5727
Epoch 2/50
57/57 ━━━━━━━━━━ 0s 446us/step - accuracy: 0.7144 - loss: 0.5873 - val_accuracy: 0.7832 - val_loss: 0.4959
Epoch 3/50
57/57 ━━━━━━━━━━ 0s 404us/step - accuracy: 0.8047 - loss: 0.5142 - val_accuracy: 0.8182 - val_loss: 0.4570
Epoch 4/50
57/57 ━━━━━━━━━━ 0s 472us/step - accuracy: 0.8034 - loss: 0.4710 - val_accuracy: 0.8182 - val_loss: 0.4322
Epoch 5/50
57/57 ━━━━━━━━━━ 0s 424us/step - accuracy: 0.7987 - loss: 0.4567 - val_accuracy: 0.8182 - val_loss: 0.4193
Epoch 6/50
57/57 ━━━━━━━━━━ 0s 485us/step - accuracy: 0.8108 - loss: 0.4511 - val_accuracy: 0.8252 - val_loss: 0.4122
Epoch 7/50
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy*100:.2f}%')
```

✓ 0.0s

6/6 ━━━━━━━━━━ 0s 459us/step - accuracy: 0.8096 - loss: 0.4326

Accuracy: 81.01%

Python: deep learning in Keras, Pytorch, Tensorflow

Example: Pytorch

```
class TitanicModel(nn.Module):
    def __init__(self):
        super(TitanicModel, self).__init__()
        self.fc1 = nn.Linear(X_train.shape[1], 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x

model = TitanicModel()
```

✓ 0.0s

```
model.eval()
with torch.no_grad():
    outputs = model(X_test).squeeze()
    predicted = (outputs > 0.5).float()
    accuracy = (predicted == y_test).float().mean()
    print(f'Accuracy: {accuracy:.4f}')
```

✓ 0.0s

Accuracy: 0.8212

```
# Create DataLoader
train_dataset = TensorDataset(X_train, y_train)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs.squeeze(), labels)
        loss.backward()
        optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

] ✓ 0.6s

Epoch [10/100], Loss: 0.6330
Epoch [20/100], Loss: 1.1626
Epoch [30/100], Loss: 0.1475
Epoch [40/100], Loss: 0.6610
Epoch [50/100], Loss: 0.1602
Epoch [60/100], Loss: 0.6602
Epoch [70/100], Loss: 0.4953
Epoch [80/100], Loss: 0.0686
Epoch [90/100], Loss: 0.2583
Epoch [100/100], Loss: 0.6319

Python: deep learning in Keras, Pytorch, Tensorflow

Example: Keras

```
model = Sequential()
model.add(Dense(32, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

✓ 0.0s
```

```
model.fit(X_train, y_train, epochs=50, batch_size=10, validation_split=0.2)
✓ 1.9s
```

```
Epoch 1/50
57/57 0s 1ms/step - accuracy: 0.7250 - loss: 0.6115 - val_accuracy: 0.8112 - val_loss: 0.5010
Epoch 2/50
57/57 0s 435us/step - accuracy: 0.7885 - loss: 0.5029 - val_accuracy: 0.8112 - val_loss: 0.4352
Epoch 3/50
57/57 0s 418us/step - accuracy: 0.8022 - loss: 0.4524 - val_accuracy: 0.8182 - val_loss: 0.4142
Epoch 4/50
57/57 0s 483us/step - accuracy: 0.7908 - loss: 0.4602 - val_accuracy: 0.8322 - val_loss: 0.4029
Epoch 5/50
57/57 0s 875us/step - accuracy: 0.7921 - loss: 0.4676 - val_accuracy: 0.8322 - val_loss: 0.3961
Epoch 6/50
57/57 0s 449us/step - accuracy: 0.8340 - loss: 0.4090 - val_accuracy: 0.8252 - val_loss: 0.3967
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy*100:.2f}%')

✓ 0.0s
```

```
6/6 0s 447us/step - accuracy: 0.8013 - loss: 0.4348
Accuracy: 80.45%
```