Artificial Neural Networks

Esra Suel

CASA0006: Data Science for Spatial Systems

Slides adapted from Ender Konukoglu at ETH Zurich & Huanfa Chen at UCL

Outline

- Basic concepts and some math
- Simple models: linear regression and logistic regression
- Perceptron model and multilayer extension: Artificial Neural Networks (ANNs)
- Deep learning for images: Convolutional Neural Networks (CNNs)
  - CNN basics
  - CNNs for geospatial research
  - Potential Problems with CNNs
- Deep learning for text data

We will focus on the supervised learning

There is a **task**, e.g. house price prediction, segmentation, detection, recognition, …

Examples have both **features** (predictors, images) and **labels** (prices, classes) related to the task

At inference time you only have the features

Example task: segmentation

- Appearance variation across classes
- Ignore variance within instances of the same class

Training examples are extremely important!

Examples to learn from

Test image Prediction

Basic concepts and some math

Basic notation

$$\mathbf{x} = \{x_1, x_2, \ldots, x_d\}$$

Features

- Observed information
- Numerical or categorical

- # bedrooms, location, …

- Hand-crafted explicit features
- Multispectral images

$$\mathbf{y} = \{y_1, y_2, \ldots, y_m\}$$

Labels

- Unobserved information at prediction
- Numerical (regression) or categorical (classification)

- House prices

- Semantic segmentation labels – pixel-wise
- Object categories – image-wise

Example: Regression

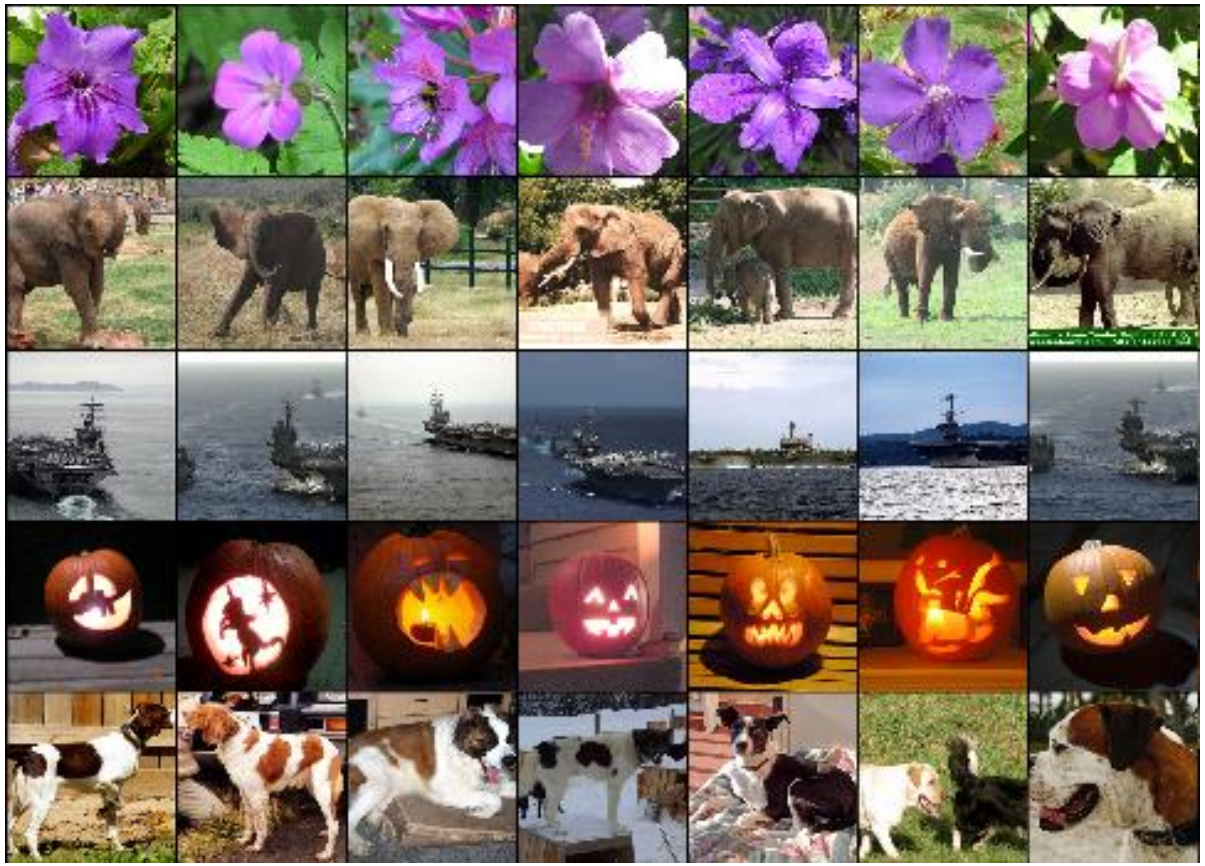
Features:

- # bedrooms
- Size
- Location
- Postcode
- …

Labels: house price

size of house (square feet)	# of bedrooms	price (1000$)
523	1	115
645	1	0.001
708	unknown	210
1034	3	unknown
unknown	4	355
2545	unknown	440

Example: Classification



Flower

Elephant

Ship

Carved Pumpkin

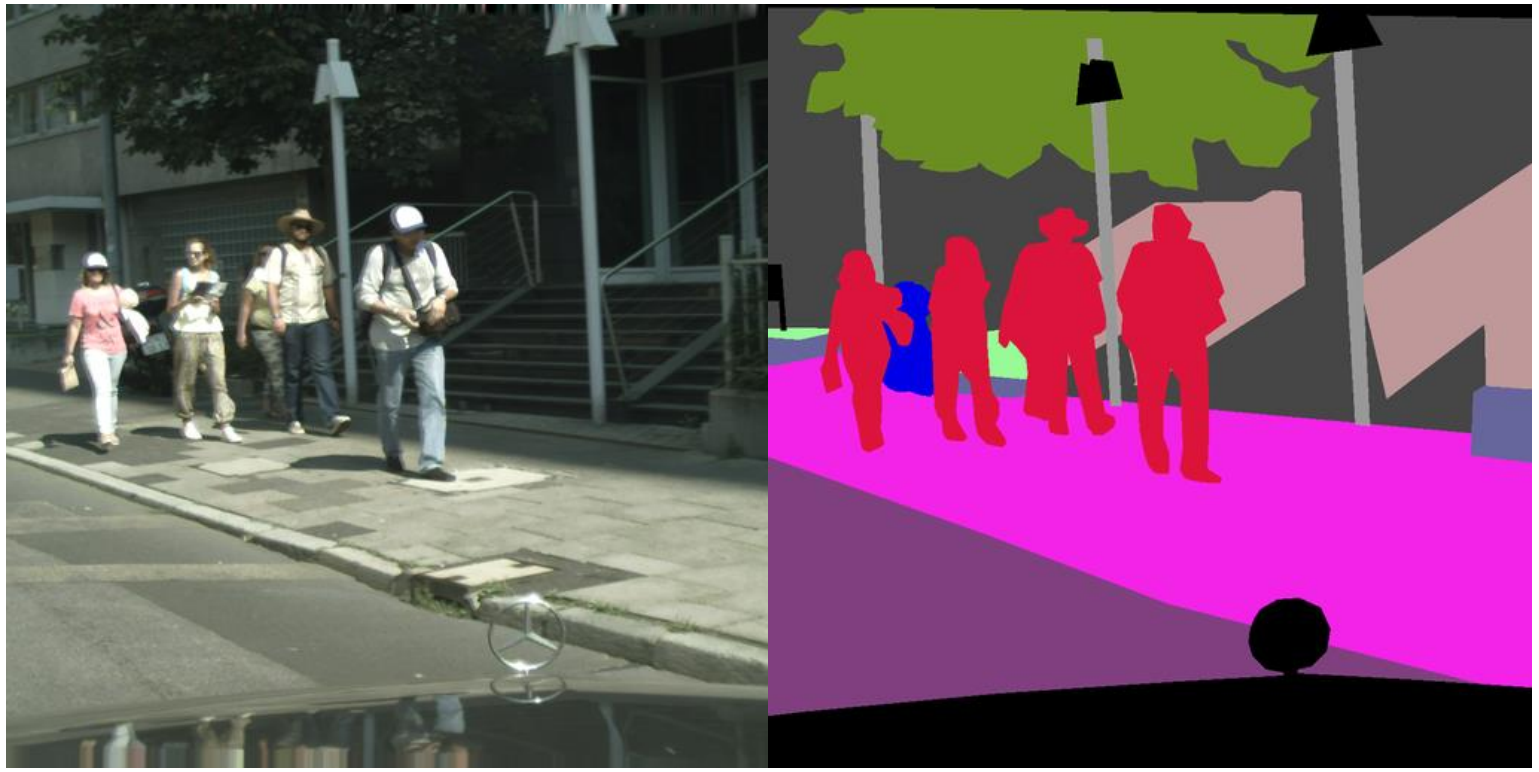
Dog

Features: images, intensity values at each pixel

Labels: object names

[Figure from Krizhevsky, Sutskever and Hinton 2012 – Predictions on ImageNet with CNNs]

Example: Segmentation



Features: Images
intensities at all pixels

Labels: Segmentation maps
object-categories at all pixels

Example: Sentiment prediction

- Sentiment Analysis assigns a sentiment score to each word in a piece of text

I love data science, and our teachers are awesome	+4 (Strongly positive)
Beer is disgusting, why do people even like it?	-1 (Weakly negative)
It's so great that my train is late every single day	+1 (Weakly positive)

- Input: a piece of text
- Output: a piece of text with a sentiment score assigned to each word

Basic concepts - mapping

$$\mathbf{x} = [x_1, x_2, \ldots, x_d] \qquad \mathbf{y} = [y_1, y_2, \ldots, y_m]$$

- Model a mapping between features and labels
- We will focus on parametric mappings with parameters: θ

$$\mathbf{y} = f(\mathbf{x}; \theta)$$

Basic concepts - learning

$$\mathbf{y} = f(\mathbf{x}; \theta)$$

- Determine the ``best'' parameters using the examples
- Labeled examples used for learning are called **“Training set”**
- The examples form a paired dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$
- “Best” parameters are the ones that minimize a cost defined between predictions and “ground-truth” labels

Basic concepts – best parameters

- Cost between ground truth label and prediction

$$\mathcal{L}(y, f(x;\theta))$$

- I can compute this cost for every training pair and sum

$$L(\theta) = \mathcal{L}(y_1, f(x_1;\theta)) + \mathcal{L}(y_2, f(x_2;\theta)) + \mathcal{L}(y_3, f(x_3;\theta)) + \dots$$

- I determine the parameter that achieves the minimum loss

$$\theta^* = \arg_\theta \min L(\theta)$$

- Another notation for this operation is

$$\theta^* = \arg_\theta \min \sum_{n=1}^{N} \mathcal{L}\left(\mathbf{y}_n, f(\mathbf{x}_n;\theta)\right)$$

Basic concepts – cost function

$$\mathcal{L}(y, f(x;\theta)) \qquad \theta^* = \arg_\theta \min \sum_{n=1}^{N} \mathcal{L}\left(\mathbf{y}_n, f(\mathbf{x}_n;\theta)\right)$$

- We model the definition of “best” using the cost function
- Task dependent and ideally defined for your final goal
- Regression: e.g., squared difference

$$\mathcal{L}(y_n, f(x_n;\theta) = (y_n - f(x_n;\theta))^2$$

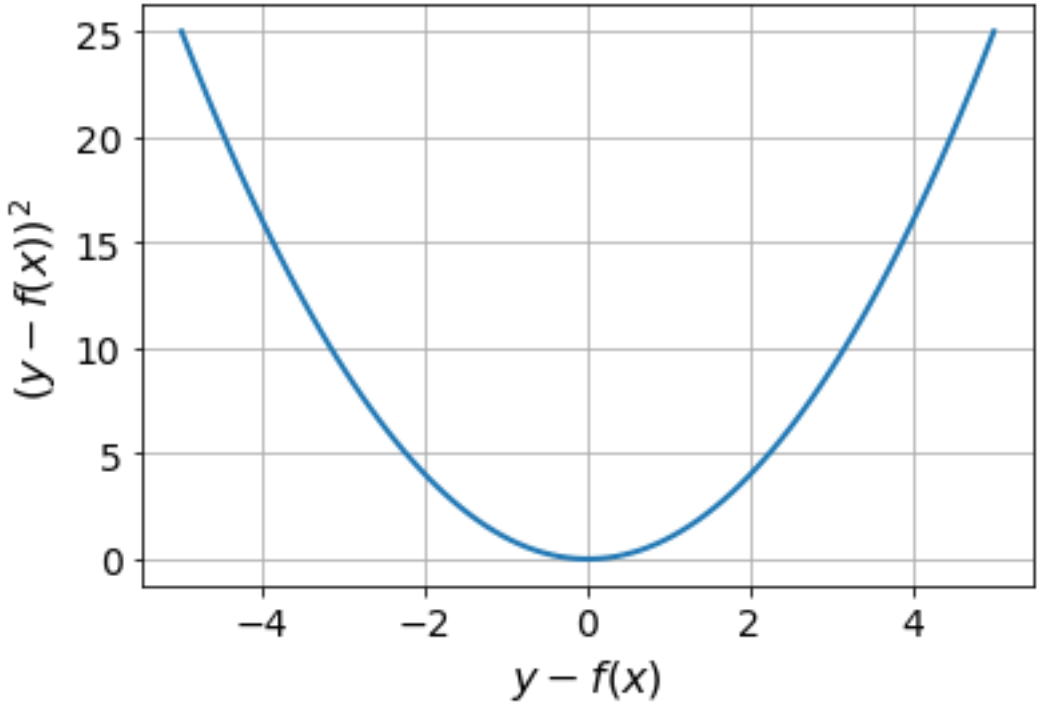
- You can also use different loss functions, e.g.

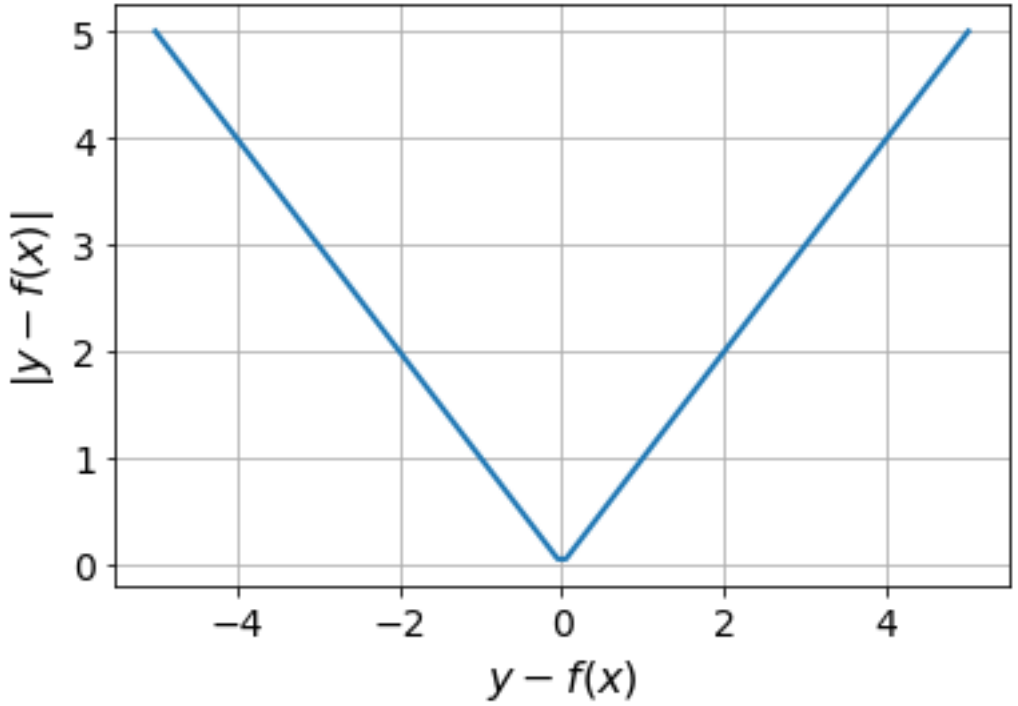
$$\mathcal{L}(y_n, f(x_n;\theta) = |y_n - f(x_n;\theta)|$$

Basic concepts – regression costs

$$\mathcal{L}(y_n, f(x_n;\theta) = (y_n - f(x_n;\theta))^2$$

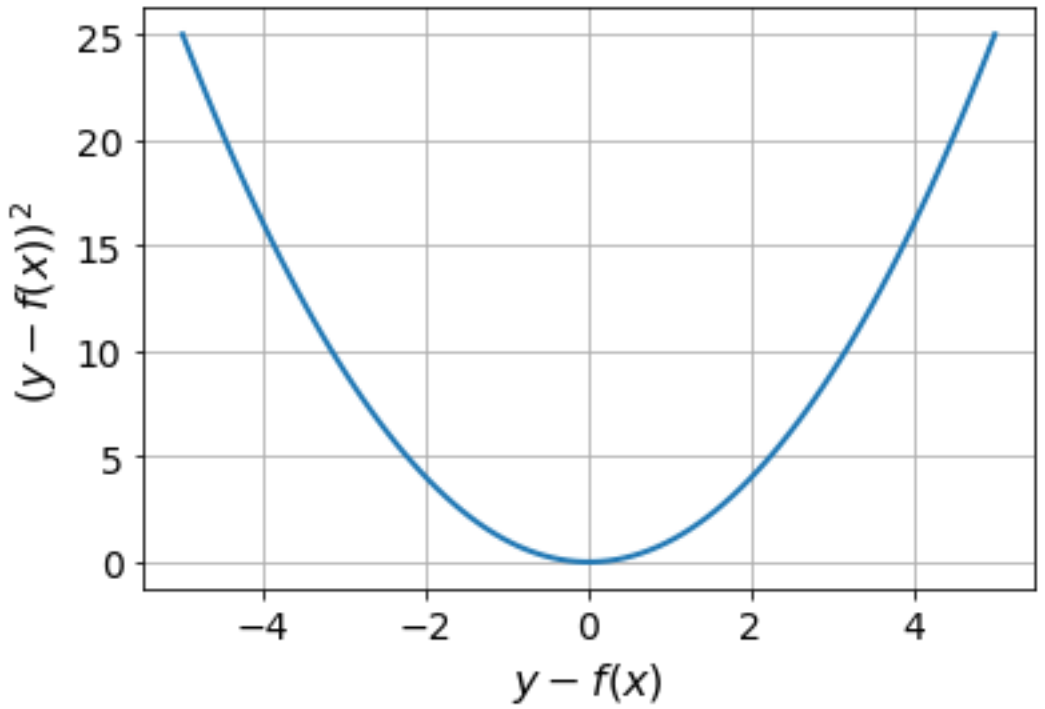
$$\mathcal{L}(y_n, f(x_n;\theta) = |y_n - f(x_n;\theta)|$$





Basic concepts – regression costs

$$\mathcal{L}(y_n, f(x_n;\theta) = (y_n - f(x_n;\theta))^2$$



Ground truth label y_n	Predicted value $f_1(x_n;\theta)$	Cost $L(y_n, f(x_n;\theta)$
3	2.7	0.3^2=0.09
3	1.7	1.3^2=1.69
-3	-1.7	1.3^2=1.69
3	4.3	1.3^2=1.69
4	1.7	2.3^2=5.29

Total cost is the sum of all individual costs

Basic concepts – cost function

$$\mathcal{L}(y, f(x;\theta)) \qquad \theta^* = \arg_\theta \min \sum_{n=1}^{N} \mathcal{L}\left(\mathbf{y}_n, f(\mathbf{x}_n;\theta)\right)$$

Cost function determines the definition of “best”

Task dependent and ideally defined for your final goal

Classification: e.g., cross-entropy for classification

$$y_n \in \mathcal{C}, \mathcal{C}\text{: set of possible classes}$$

$$f(\mathbf{x};\theta) = \left[f_{c_1}(\mathbf{x};\theta), f_{c_2}(\mathbf{x};\theta), \ldots\right], \sum_{k\in\mathcal{C}} f_k(\mathbf{x};\theta) = 1$$

where predictions are considered as class probabilities

Basic concepts – cost function

$$\mathcal{L}(y, f(x;\theta)) \qquad \theta^* = \arg_\theta \min \sum_{n=1}^{N} \mathcal{L}\left(\mathbf{y}_n, f(\mathbf{x}_n;\theta)\right)$$

Cost function determines the definition of “best”

Task dependent and ideally defined for your final goal

Classification: e.g., cross-entropy for classification

$$y_n \in \mathcal{C}, \mathcal{C}\text{: set of possible classes}$$

$$f(\mathbf{x};\theta) = \left[f_{c_1}(\mathbf{x};\theta), f_{c_2}(\mathbf{x};\theta), \ldots\right], \sum_{k\in\mathcal{C}} f_k(\mathbf{x};\theta) = 1$$

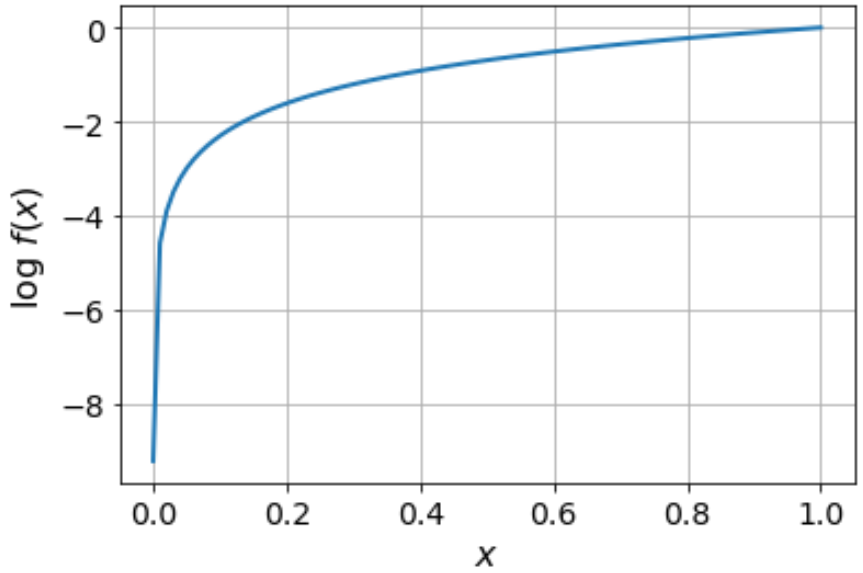
$$\mathcal{L}(y_n, f(\mathbf{x}_n;\theta)) = -\sum_{k\in\mathcal{C}} \log(f_k(\mathbf{x}_n;\theta))\mathbf{1}(y_n = k)$$

Basic concepts – cross entropy

$$\mathcal{L}(y_n, f(\mathbf{x}_n; \theta)) = -\sum_{k \in \mathcal{C}} \log(f_k(\mathbf{x}_n; \theta)) \mathbf{1}(y_n = k)$$

Binary example

$$\mathcal{L}(y_n, f(x_n; \theta)) = -\log(f_0(x_n; \theta)) \mathbf{1}(y_n = 0) - \log(f_1(x_n; \theta) \mathbf{1}(y_n = 1)$$

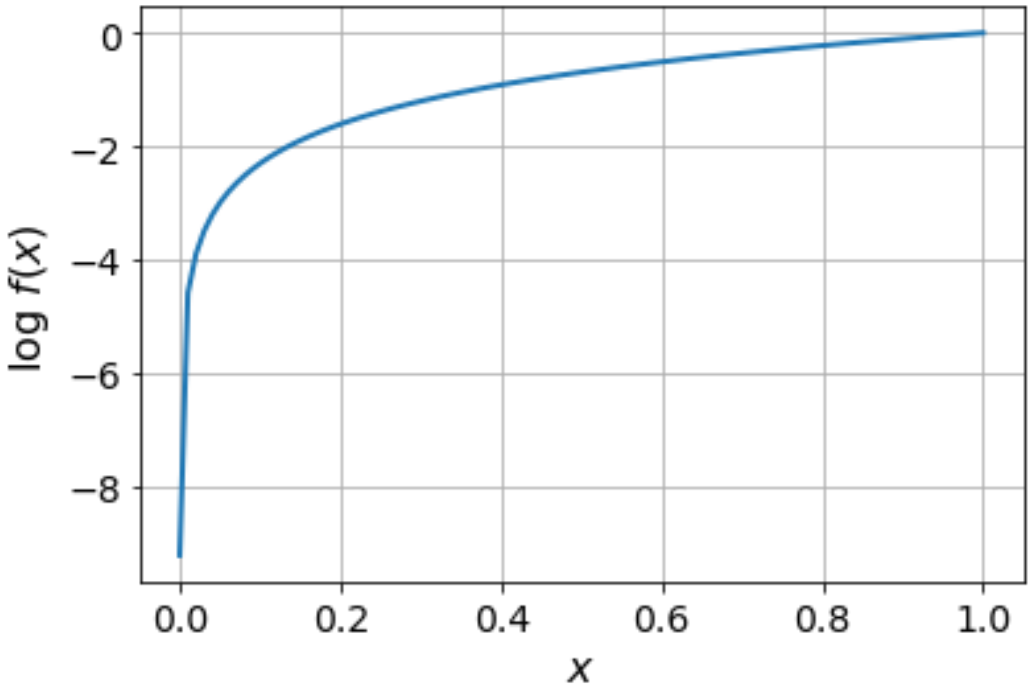


$$f_0(x_n; \theta)) + f_1(x_n; \theta) = 1$$

Basic concepts – cross entropy

$$\mathcal{L}(y_n, f(x_n;\theta)) = -\log(f_0(x_n;\theta))\mathbf{1}(y_n = 0) - \log(f_1(x_n;\theta)\mathbf{1}(y_n = 1)$$

$$f_0(x_n;\theta)) + f_1(x_n;\theta) = 1$$



Ground truth label y_n	Predicted probability $f_1(x_n;\theta)$	Cost $L(y_n, f(x_n;\theta)$
1	0.7	-log(0.7)
1	0.3	-log(0.3)
0	0.3	-log(0.7)
1	1	-log(1.0)
0	0.8	-log(0.2)

Total cost is the sum of all individual costs

Basic concepts - prediction

The model and the best parameters are determined

Prediction for a new sample also depends on your task

For regression:

$$\hat{\mathbf{y}} = f(\mathbf{x}; \theta^*)$$

Basic concepts - prediction

The model and the best parameters are determined

Prediction for a new sample also depends on your task

For classification:

$$\hat{\mathbf{y}} = \arg_k \max f_k(\mathbf{x}; \theta^*)$$

Choose the class with maximum probability

Basic concepts – prediction error

Prediction will have errors

For regression, the most used: Mean Squared Error (MSE), Mean Absolute Error (MAE)

$$MSE = \frac{1}{T}\sum_{t}^{T} \|\hat{\mathbf{y}}_t - \mathbf{y}_t\|_2^2, \;\; MAE = \frac{1}{T}\sum_{t}^{T} |\hat{\mathbf{y}}_t - \mathbf{y}_t|$$

For classification, the most used: Classification error (Cerr)

$$Cerr = \frac{1}{T}\sum_{t}^{T} \mathbf{1}(\hat{\mathbf{y}}_t \neq \mathbf{y}_t)$$

Basic concepts - Three sets:
Training, validation and test

1. Determining the best model parameters
2. Determining the hyper-parameters
3. Estimating generalization accuracy

A. Training set

B. Validation set

C. Test set

Simple models

Simplest ML model: Linear Regression

Linear model is the main building block

Assumes a linear relationship between features and labels

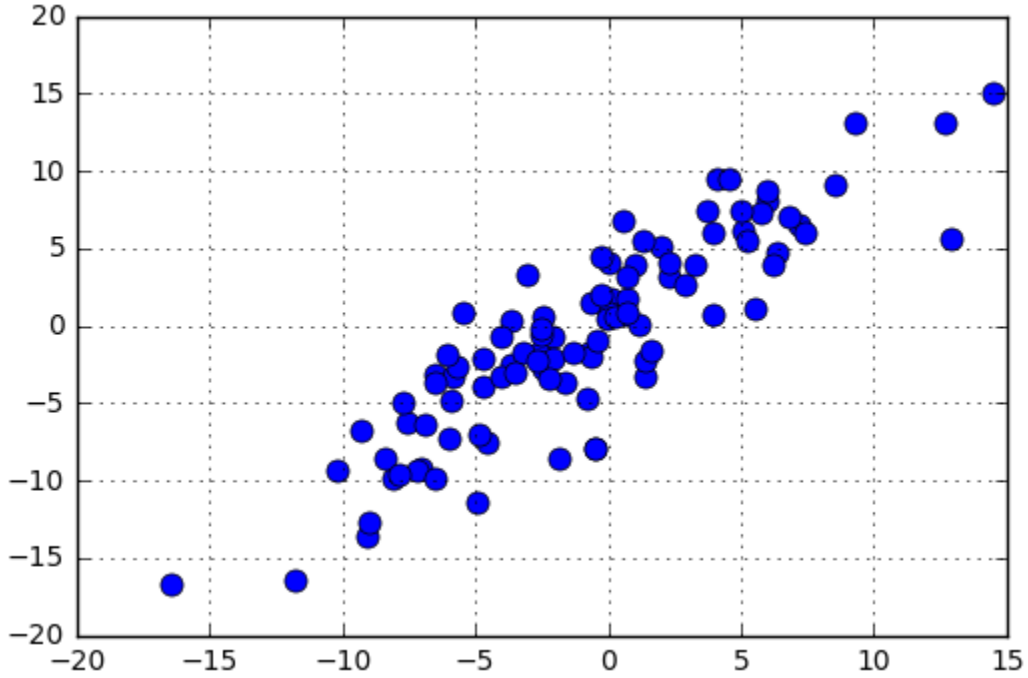
For simplicity, let us assume one-dimensional label: $\mathbf{y} = y$

And one-dimensional feature: $\mathbf{x} = x$

$$y = ax + b$$

Parameters of the linear model: $\theta = \{a, b\}$

Linear regression model



Dataset

$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$$

Model

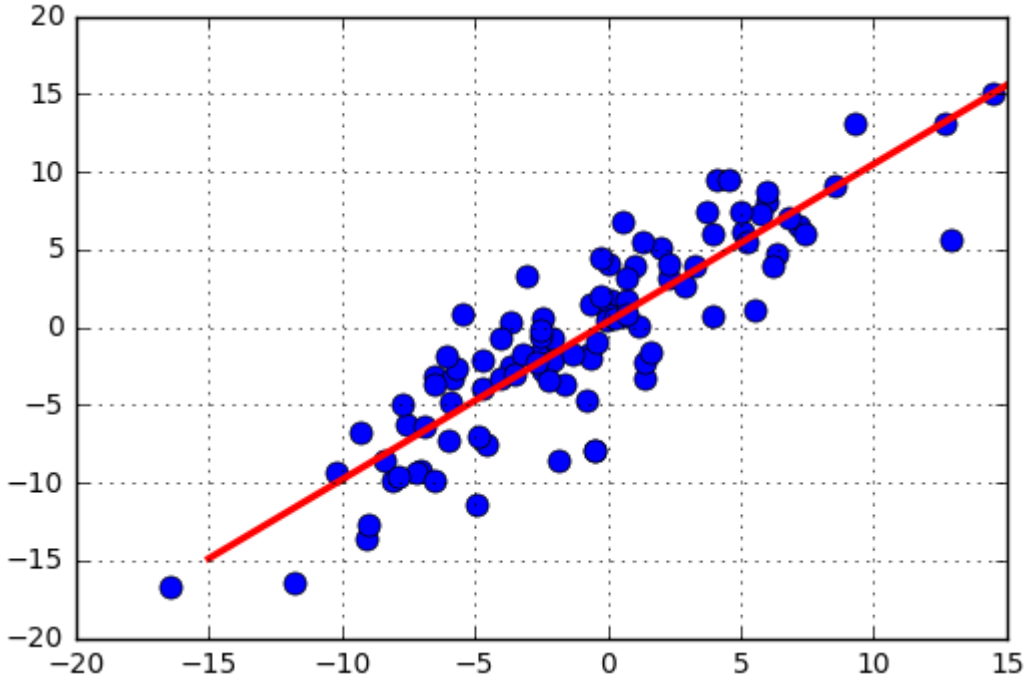
$$y = f(x;\theta) = f(x;a,b) = ax + b$$

Learning

$$a^*, b^* = \arg_{a,b} \min \sum_{n=1}^{N} \|y_n - ax_n - b\|_2^2$$

a^*, b^* : Optimal model parameters for this dataset

Linear regression model



Dataset

$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$$

Model

$$y = f(x;\theta) = f(x;a,b) = ax + b$$

Learning

$$a^*, b^* = \arg_{a,b} \min \sum_{n=1}^{N} \|y_n - ax_n - b\|_2^2$$

a^*, b^* : Optimal model parameters for this dataset

Generalization: multiple features

One feature was easy $y = ax + b$

Generalizing to multiple features is also easy

For simplicity, let us assume one dimensional label: $\mathbf{y} = y$

$$\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = \theta^T \mathbf{x}$$

Parameters of the linear model: θ

Linear model is for continuous labels

Linear model for Classification: Logistic Regression Models

Linear model is the main building block

Assumes a linear relationship between features and labels

For simplicity, let us assume one dimensional label: $\mathbf{y} = y$

$$\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = \theta^T \mathbf{x}$$

Parameters of the linear model: θ

Linear regression model was for continuous labels

Logistic regression is the extension to binary labels

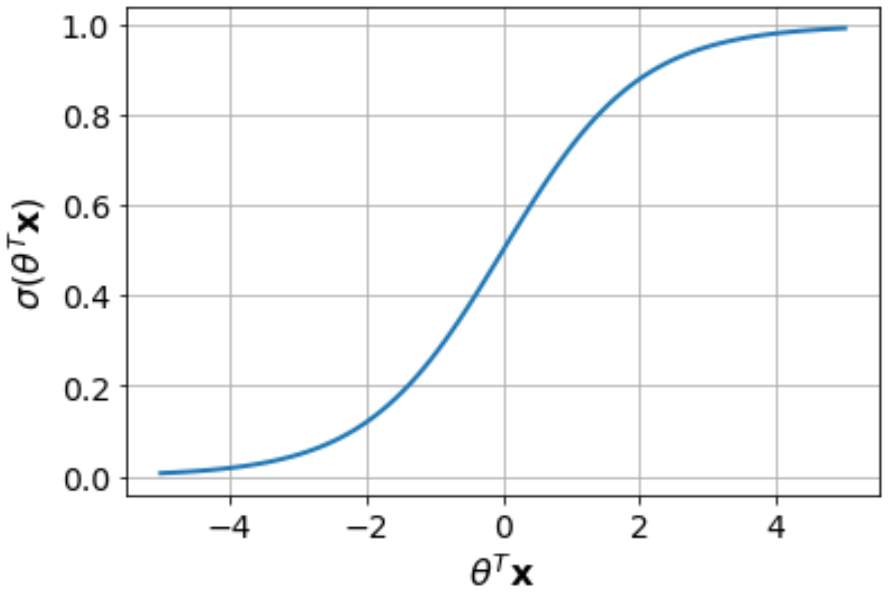
Extracting probabilities

$$\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = \theta^T \mathbf{x}$$

Continuous values that can go from -∞ to ∞

For binary classification we want probabilities

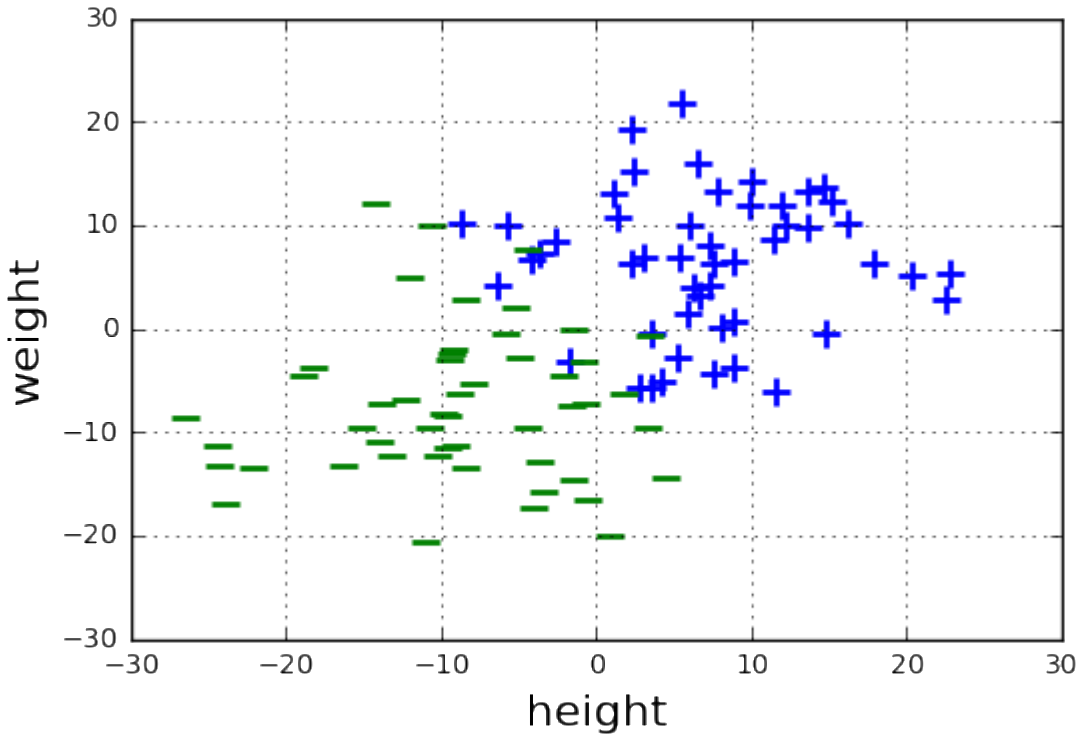
$$0 \leq f(\mathbf{x}; \theta) \leq 1$$



- Sigmoid function maps it to a probability

$$f(\mathbf{x}; \theta) = \sigma(\theta^T \mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

Logistic regression model



Dataset

$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$$

Model

$$p(y = 1) = f(\mathbf{x}; \theta) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

Learning

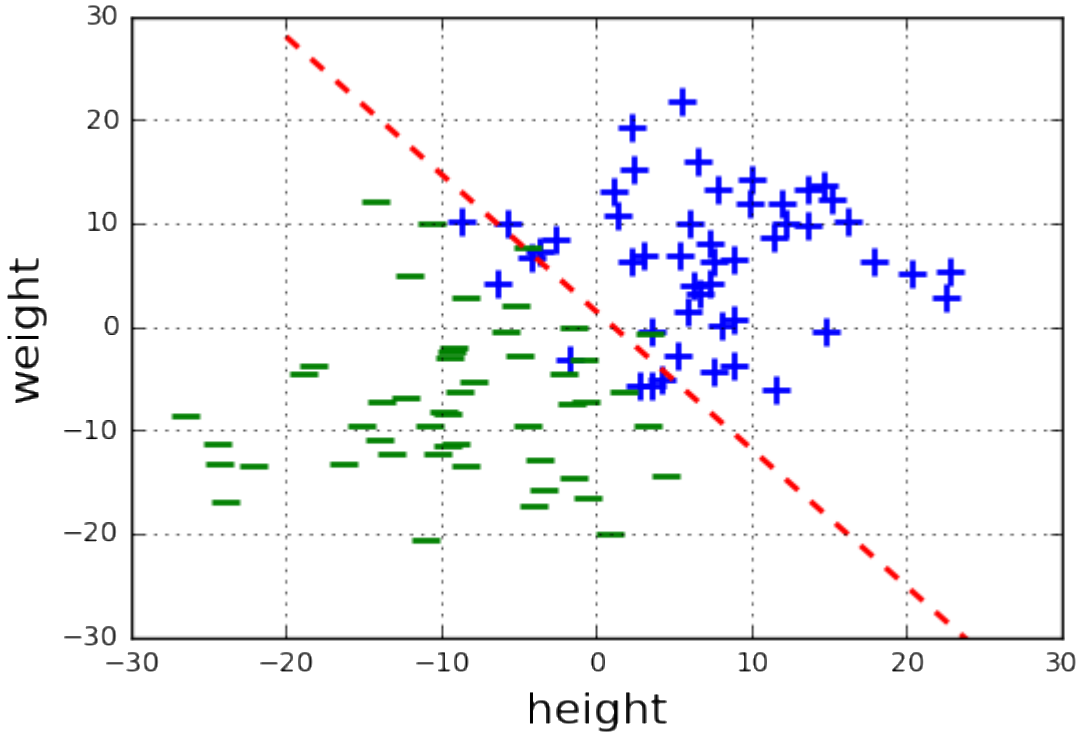
$$\theta^* = \arg_\theta \min \sum_{n=1}^{N} - \log f(\mathbf{x}_n; \theta) y_n$$

$$- \log(1 - f(\mathbf{x}; \theta))(1 - y_n)$$

Logistic regression model

Logistic regression is the building block of the perceptron model

Line denotes the points where p(y=1)=0.5



Dataset

$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$$

Model

$$p(y = 1) = f(\mathbf{x}; \theta) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

Learning

$$\theta^* = \arg_\theta \min \sum_{n=1}^{N} - \log f(\mathbf{x}_n; \theta) y_n$$

$$- \log(1 - f(\mathbf{x}; \theta))(1 - y_n)$$

Optimization - Learning

Whether classification or regression

$$\theta^* = \arg_\theta \min \mathcal{L}(\theta) \qquad \theta = [\theta_1, \theta_2, \ldots]$$

The main idea is to start from an initial estimate: θ^0

"Wiggle each parameter a bit" to find the direction that **minimizes the loss the most:** $\mathbf{v}^0$

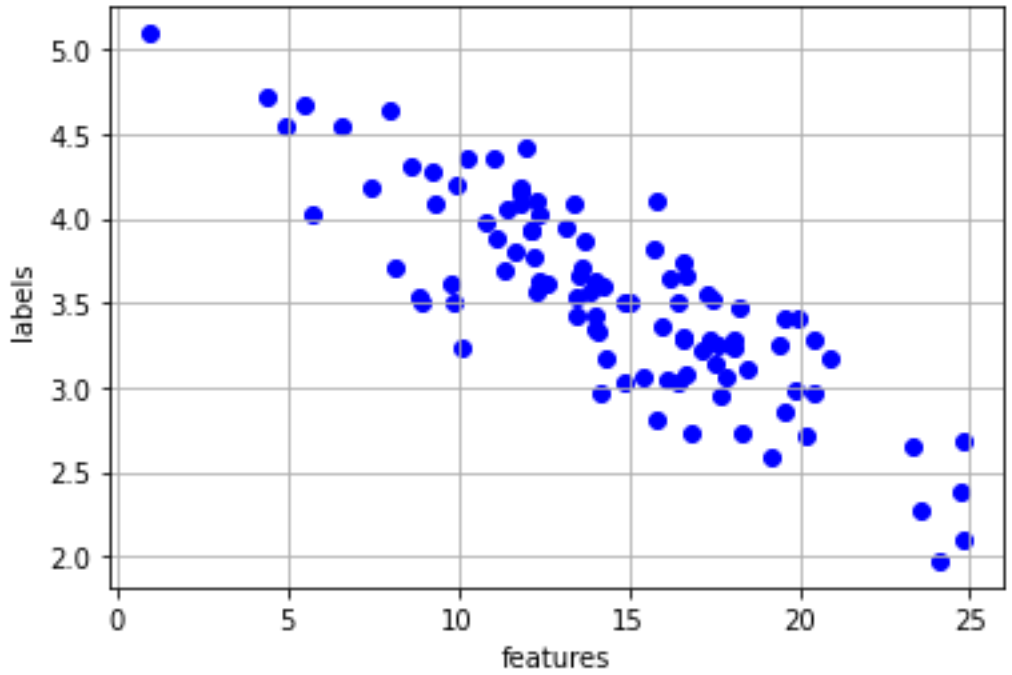
Update the parameters: $\theta^1 = \theta^0 + \mathbf{v}^0$

Continue doing this: $\theta^{t+1} = \theta^t + \mathbf{v}^t$

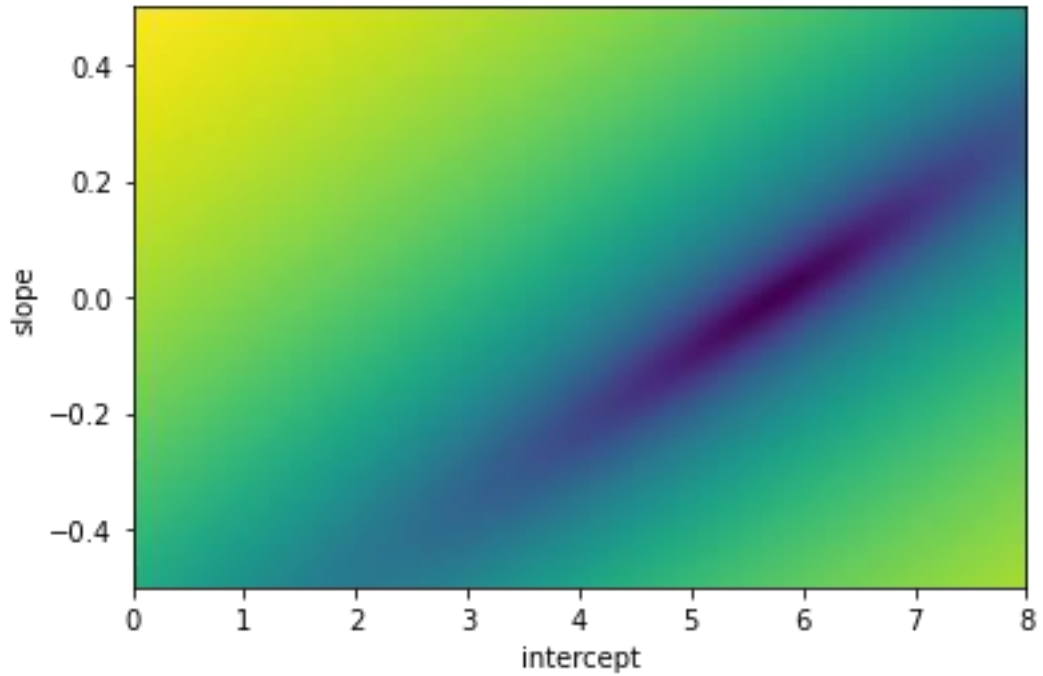
Stop when you can no longer minimize the loss

Let’s see it in action

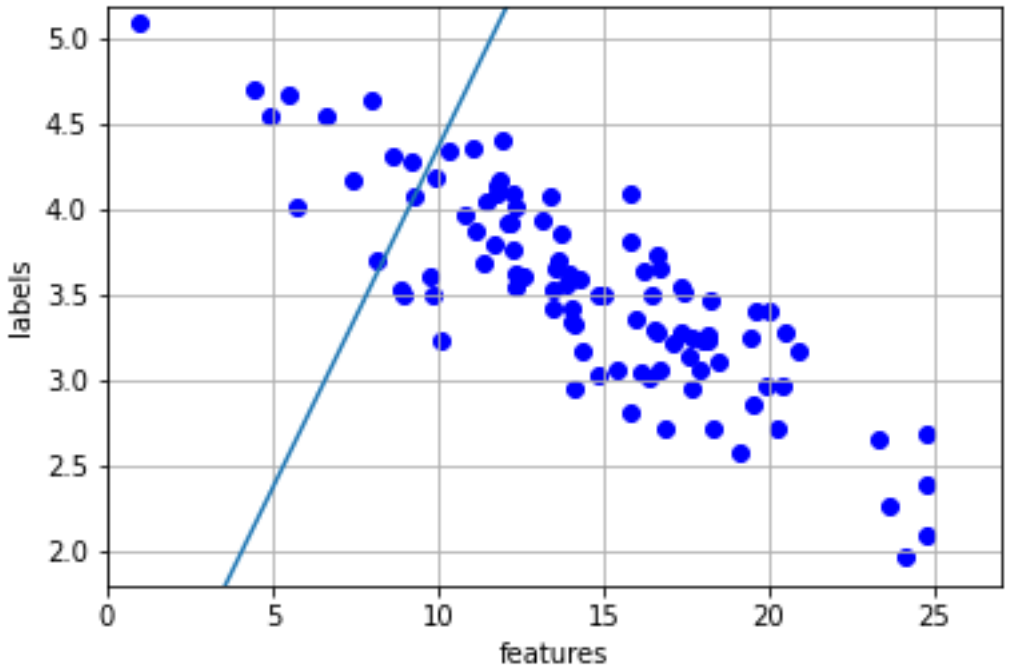
Samples

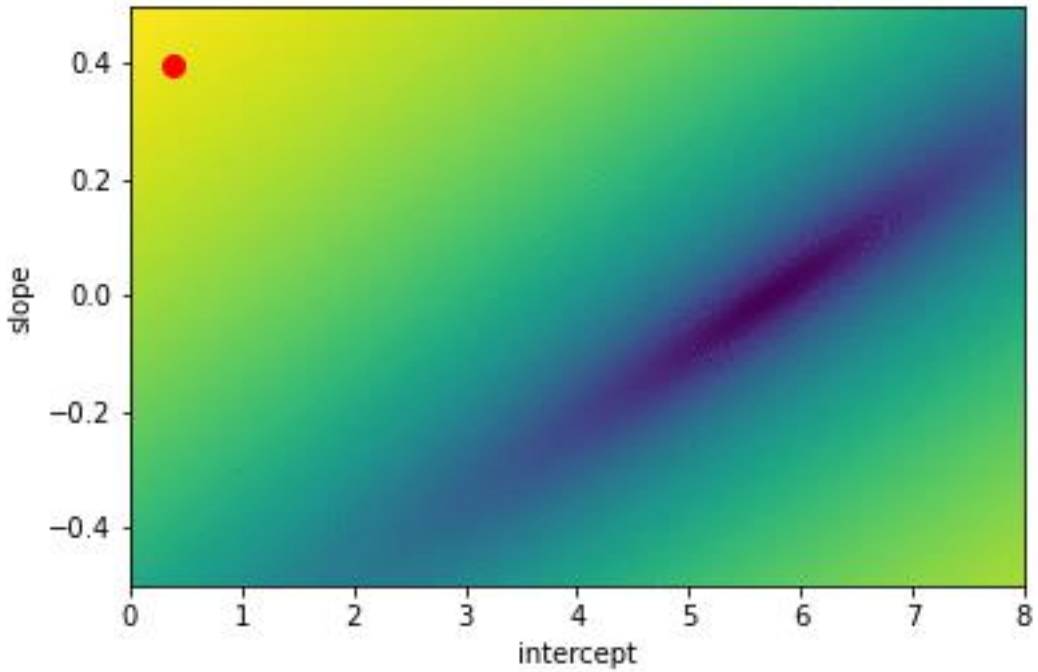


Loss function



$$\theta^{t+1} = \theta^t + \alpha \nabla \mathcal{L}(\theta) = \begin{bmatrix} \theta_0^t \\ \theta_1^t \\ \vdots \\ \theta_d^t \end{bmatrix} + \alpha \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \frac{\partial \mathcal{L}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \theta_d} \end{bmatrix}$$

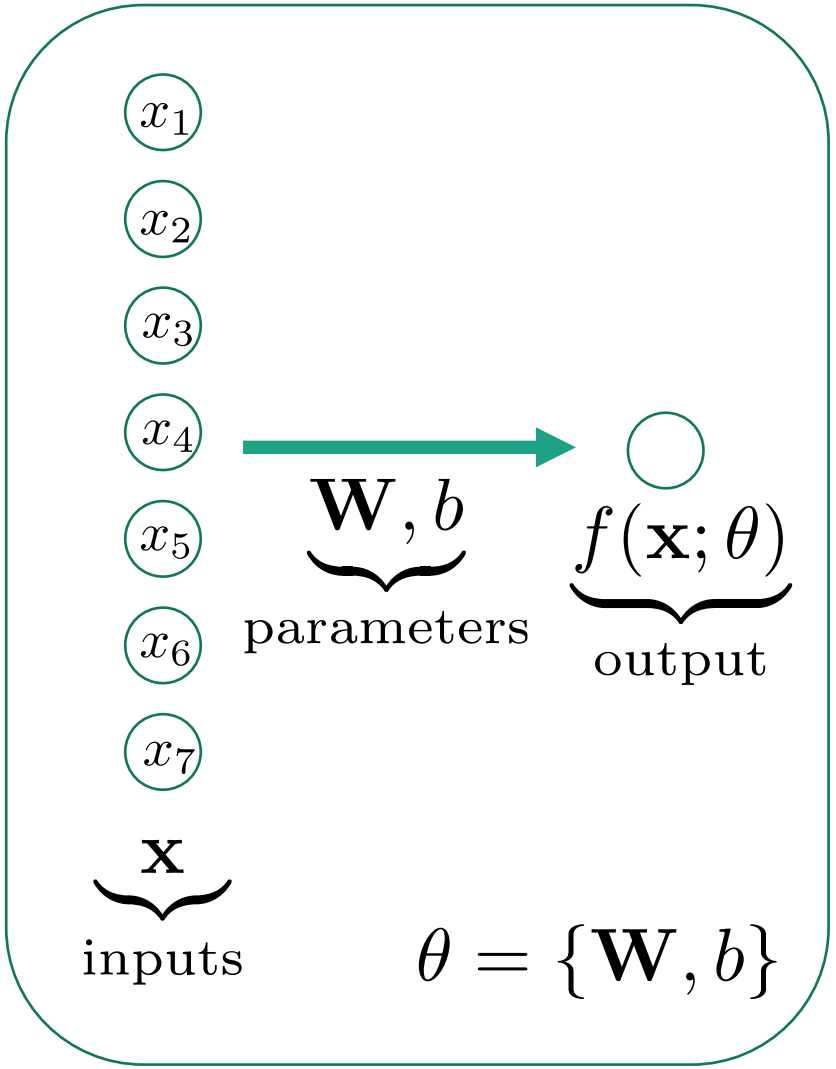




Perceptron model and multilayer extension

Artificial Neural Networks (ANNs)

Basic perceptron model



Model of binary classification

$$p(y=1) = f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$$

Formed of two different parts

1. Activation

$$\underbrace{a}_{\text{activation}} = \underbrace{\mathbf{W}}_{\text{weights}}\mathbf{x} + \underbrace{b}_{\text{bias}}$$

2. Nonlinearity $\quad f(\mathbf{x};\theta) = \sigma(a)$

Activation

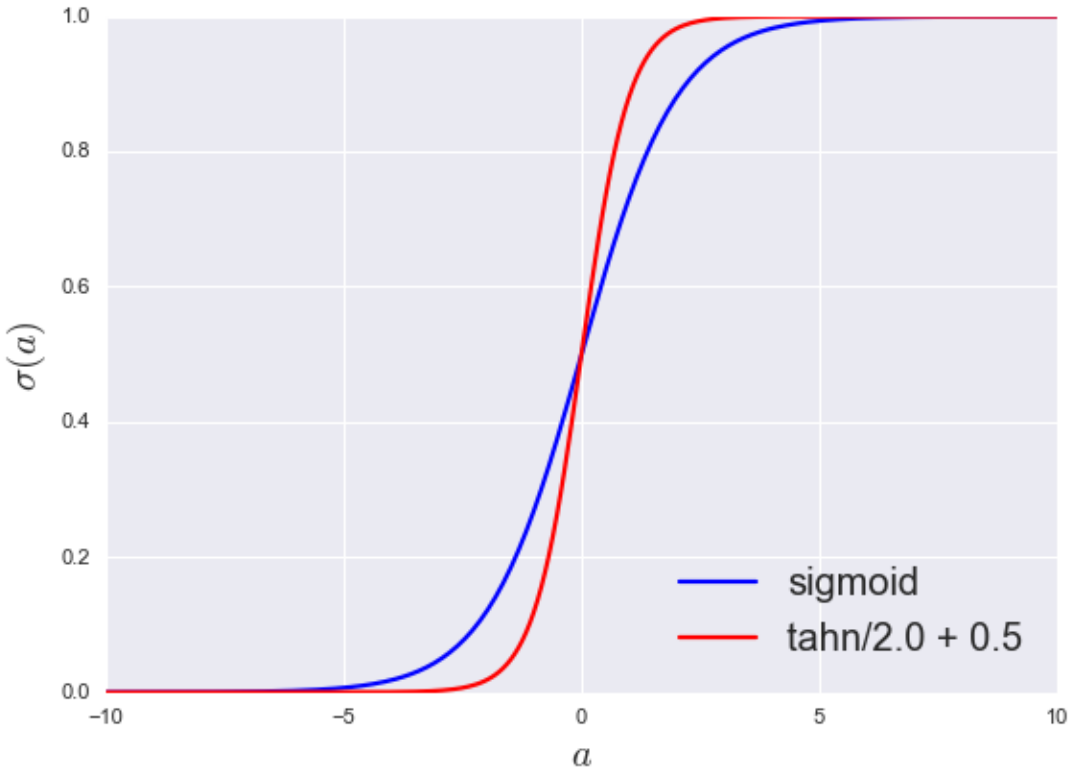
$$\underbrace{a}_{\text{activation}} = \underbrace{\mathbf{W}}_{\text{weights}}\,\mathbf{x} + \underbrace{b}_{\text{bias}} \longrightarrow a = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

- Linear transformation of the features
- d + 1 number of parameters $\mathbf{W} \in \mathbb{R}^{1 \times d}$ $b \in \mathbb{R}$

Non-linearity

Maps real line to probabilities, i.e.

Element-wise application



$$\sigma : \mathbb{R} \mapsto (0, 1)$$

Sigmoid function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

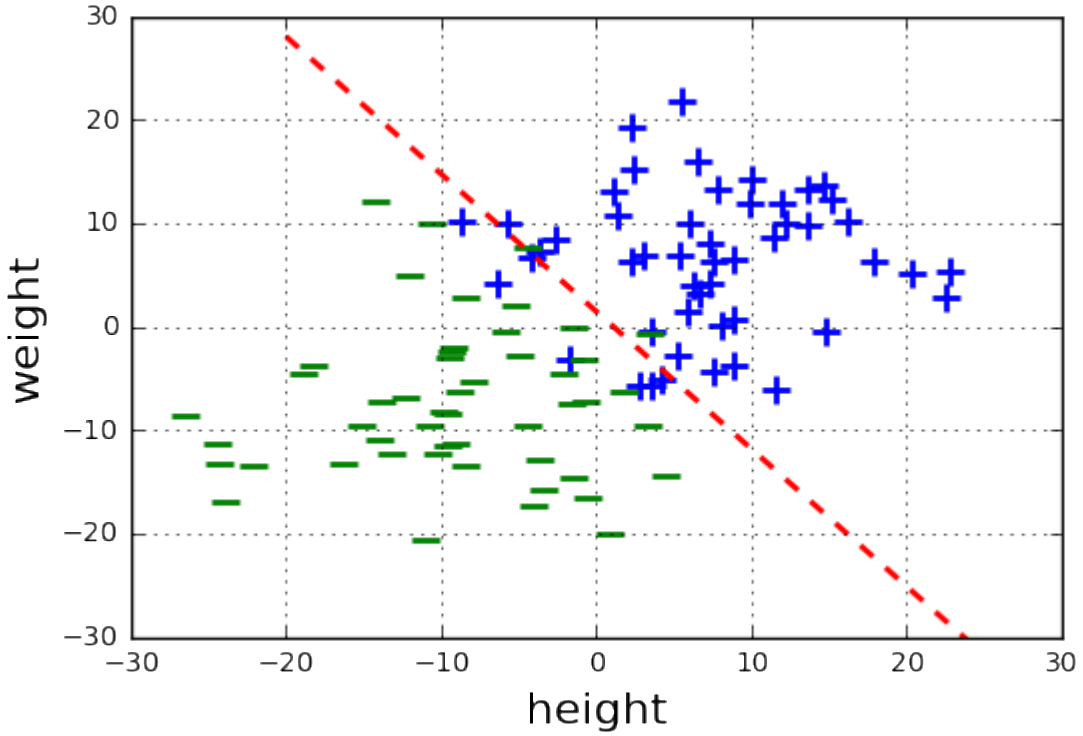
Tangent Hyperbolic

$$\sigma(a) = \tanh(a)/2.0 + 0.5$$

Remember - Logistic regression model

Logistic regression is the building block of the perceptron model

Line denotes the points where p(y=1)=0.5



Dataset

$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$$

Model

$$p(y = 1) = f(\mathbf{x}; \theta) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

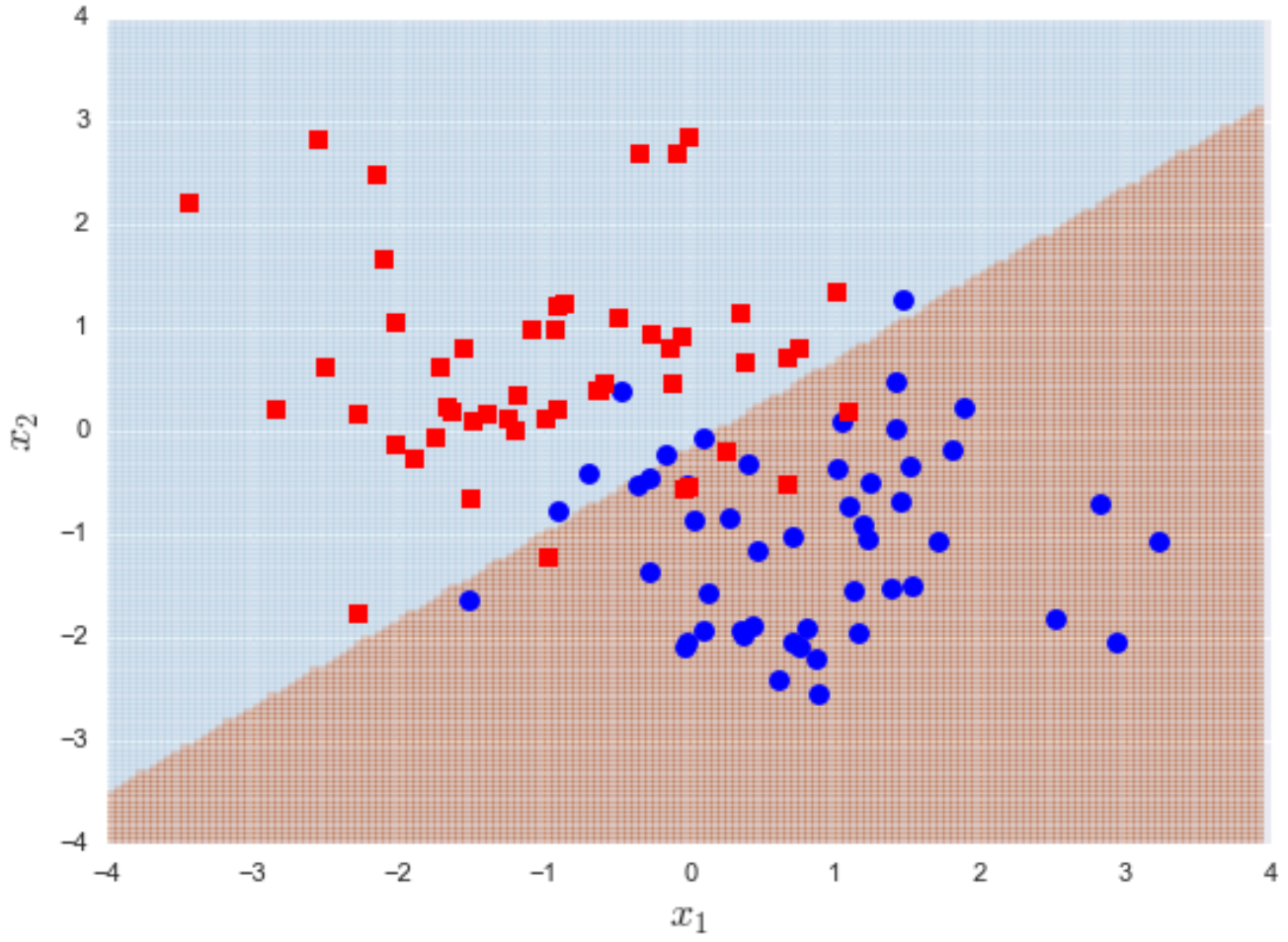
Learning

$$\theta^* = \arg_\theta \min \sum_{n=1}^{N} - \log f(\mathbf{x}_n; \theta) y_n$$

$$- \log(1 - f(\mathbf{x}; \theta))(1 - y_n)$$

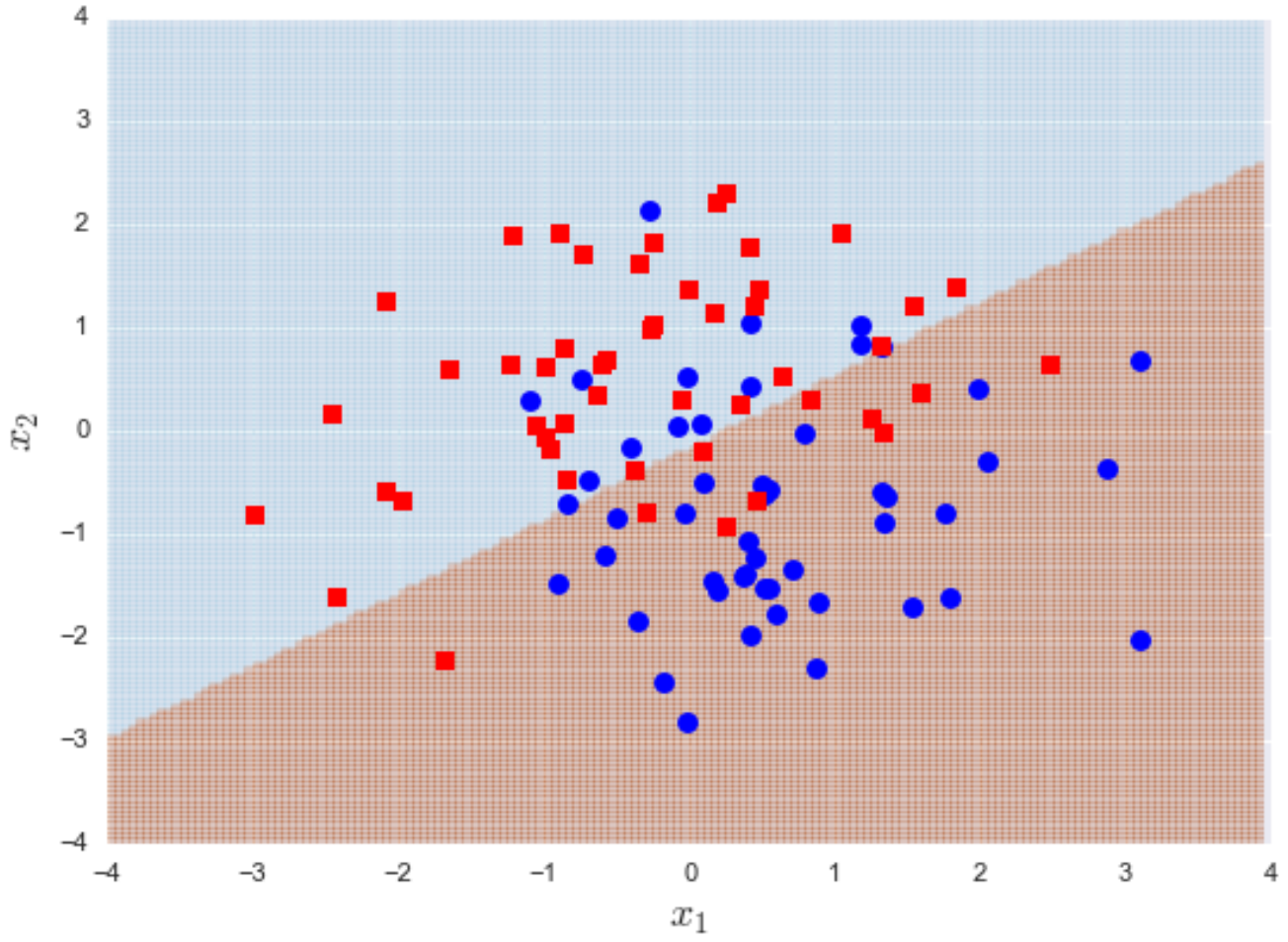
Decision boundary

$$f(\mathbf{x};\theta) = 0.5$$



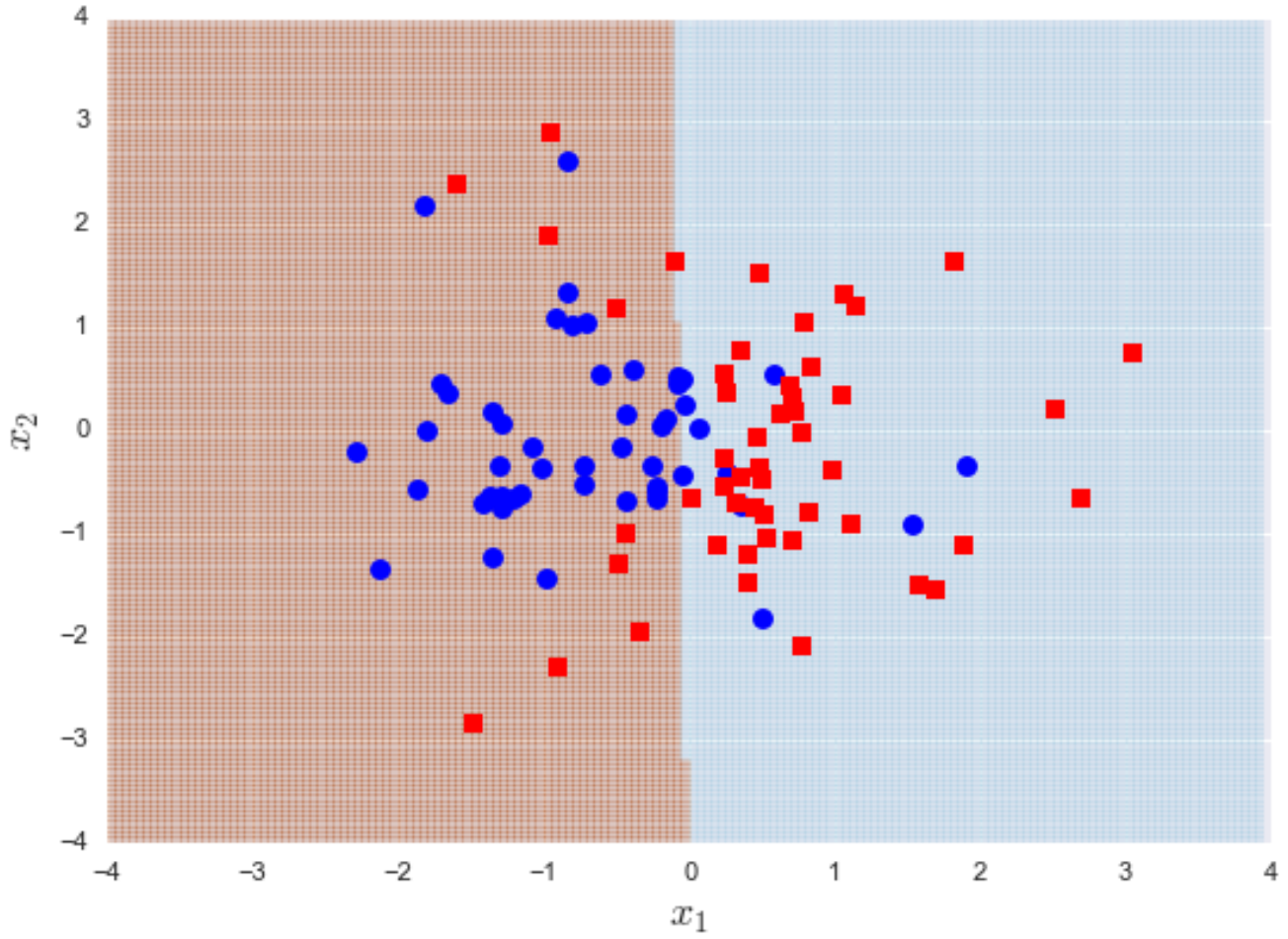
Decision boundary II

$$f(\mathbf{x};\theta) = 0.5$$



Decision boundary III

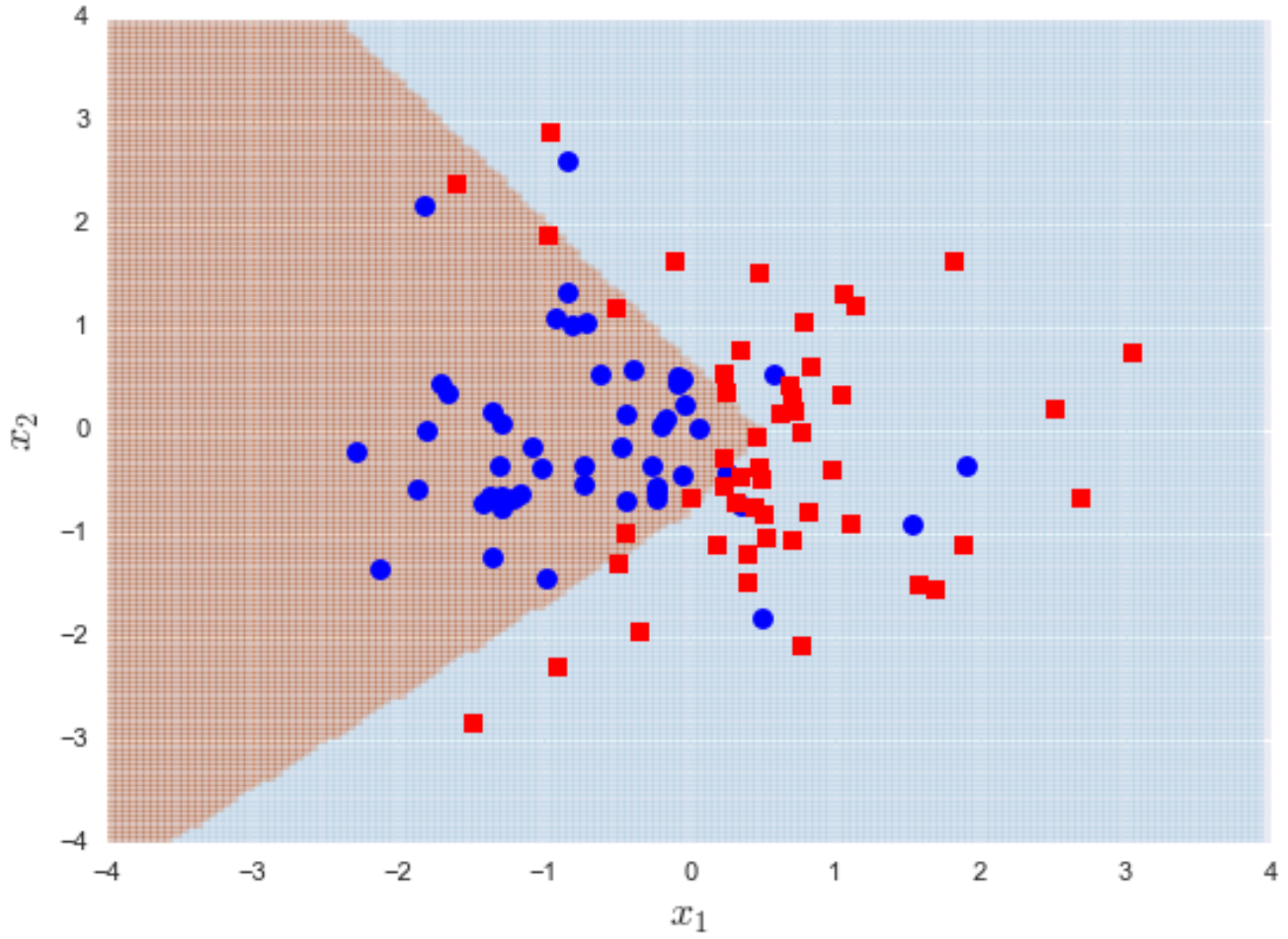
$$f(\mathbf{x};\theta) = 0.5$$



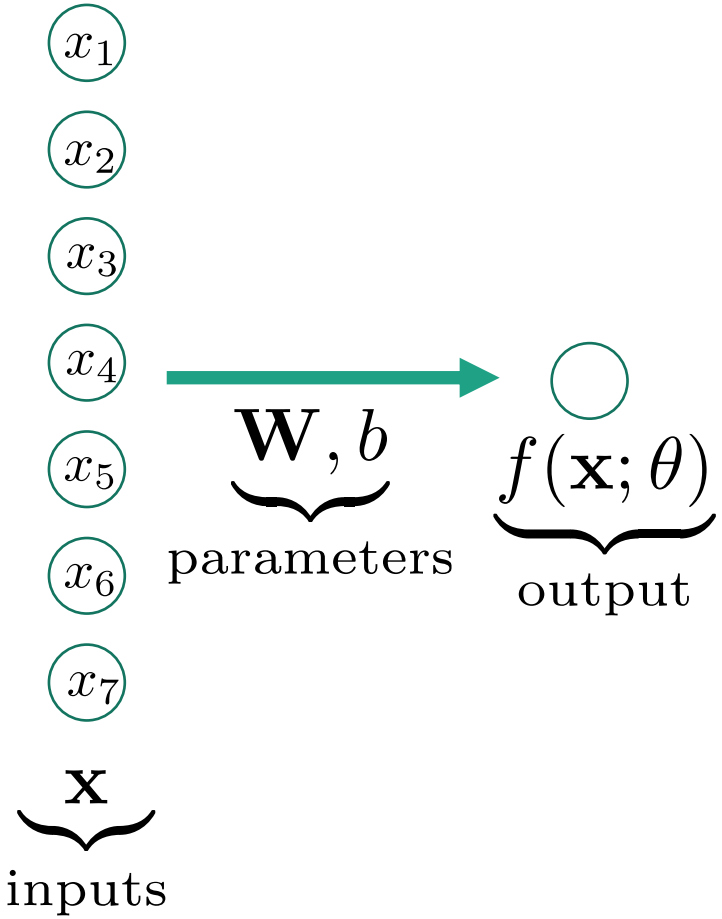
Notes on perceptron model

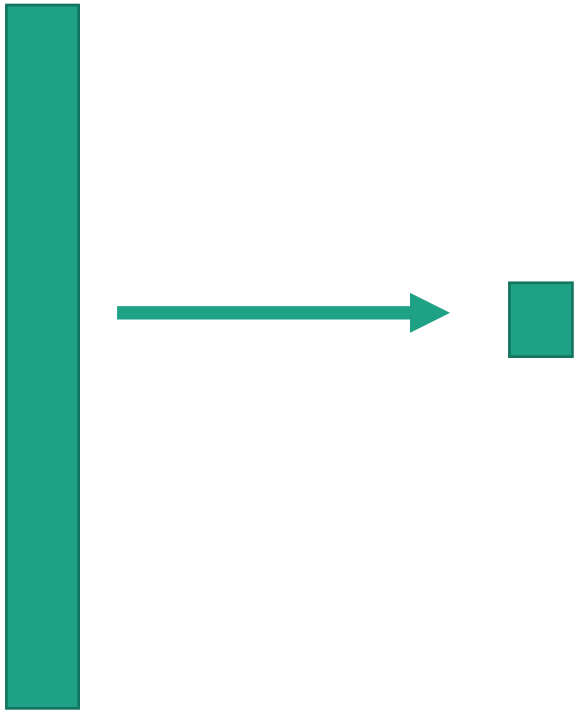
- Logistic regression is the building block
- Essentially perceptron is a linear model
- Linear decision boundary
- Cannot model more complicated decision boundaries
- Building block for more complicated models
- We have not seen training yet, will come back that
- First let’s see more complicated models

It would be better if



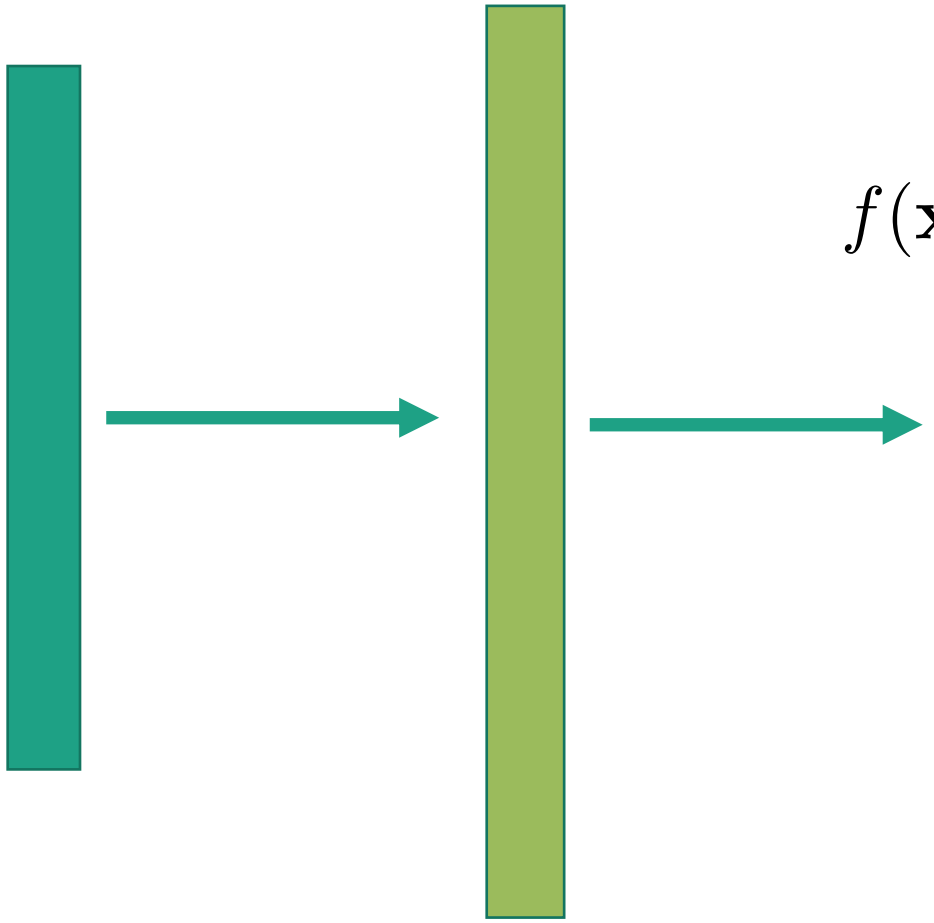
Simpler graphical representation





Simpler graphical representation

Multilayer perceptron (MLP)



$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$$

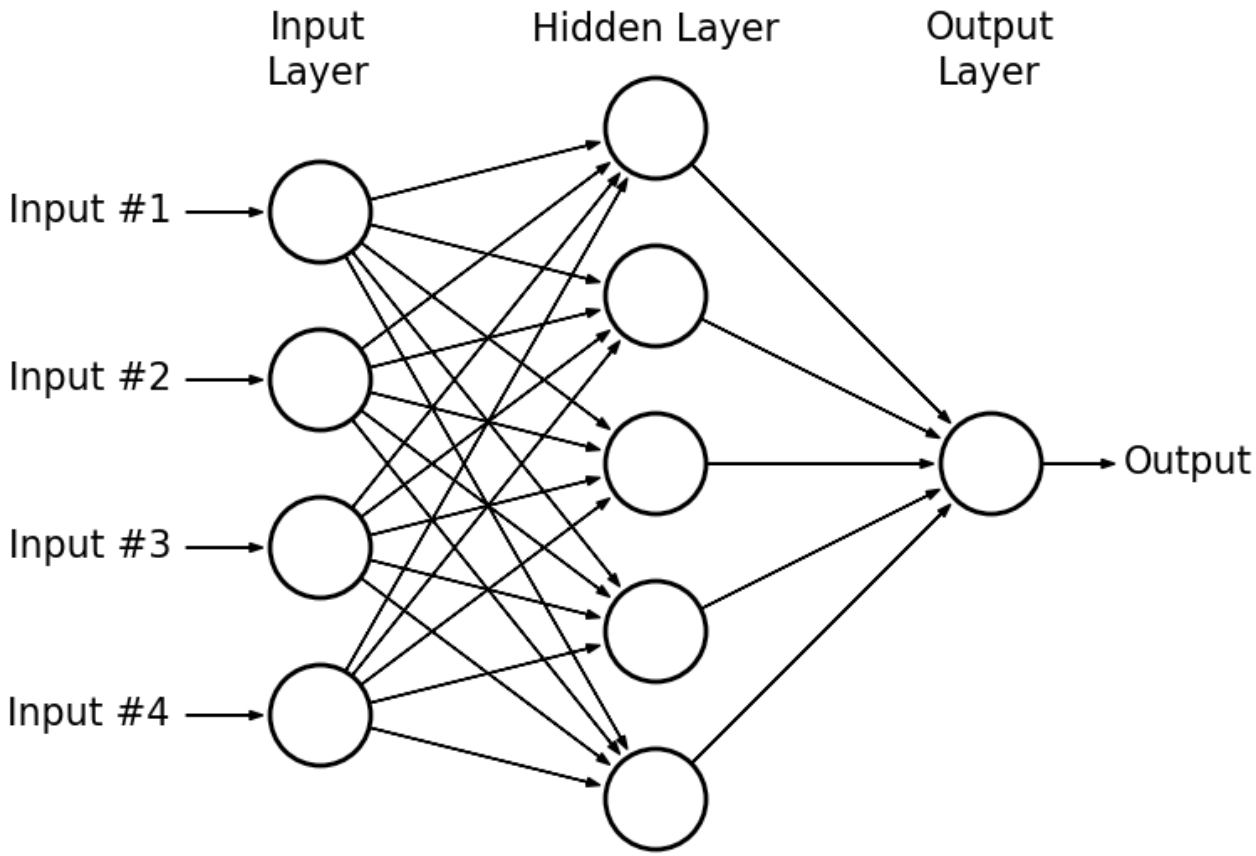
$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + b_1) + b_2)$$

$$f(\mathbf{x};\theta) = f_2(f_1(\mathbf{x};\theta_1);\theta_2)$$

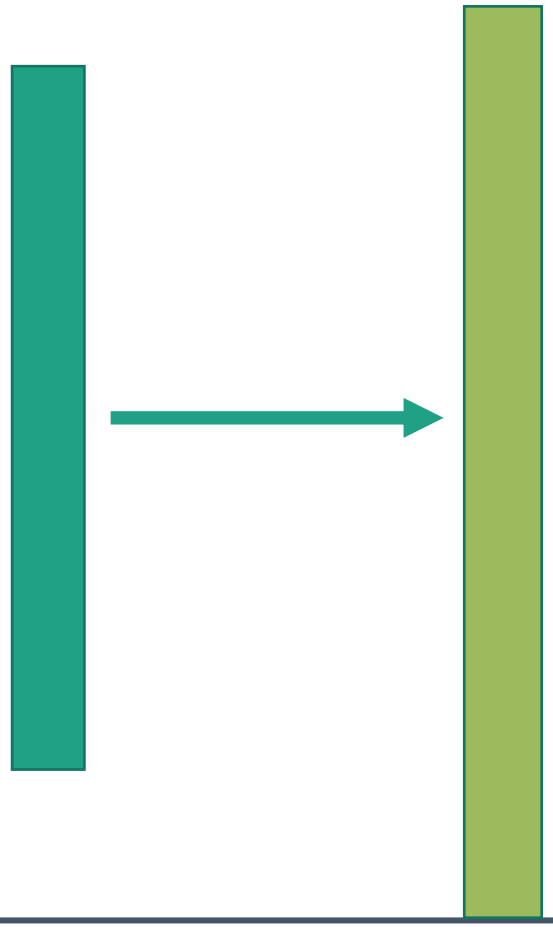
$$\theta = \{\theta_1, \theta_2\} = \{\mathbf{W}_1, \mathbf{W}_2, b_1, b_2\}$$

Short-hand notation

$$f(\mathbf{x};\theta) = f_2 \circ f_1 \circ \mathbf{x}$$


Input
Layer
Hidden Layer
Output
Layer
Input #1
Input #2
Input #3
Input #4
Output

The first layer transition becomes a matrix-vector product



$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + b_1) + b_2)$$

$$f(\mathbf{x};\theta) = f_2(f_1(\mathbf{x};\theta_1);\theta_2)$$

$$\mathbf{h} = f_1(\mathbf{x};\theta_1),\ \theta_1 = \{\mathbf{W}_1, b_1\}$$

$$h_1 = W^{(1)}_{1,1}x_1 + W^{(1)}_{1,2}x_2 + \cdots + W^{(1)}_{1,d}x_d + b_1$$

$$h_2 = W^{(1)}_{2,1}x_1 + W^{(1)}_{2,2}x_2 + \cdots + W^{(1)}_{2,d}x_d + b_1$$

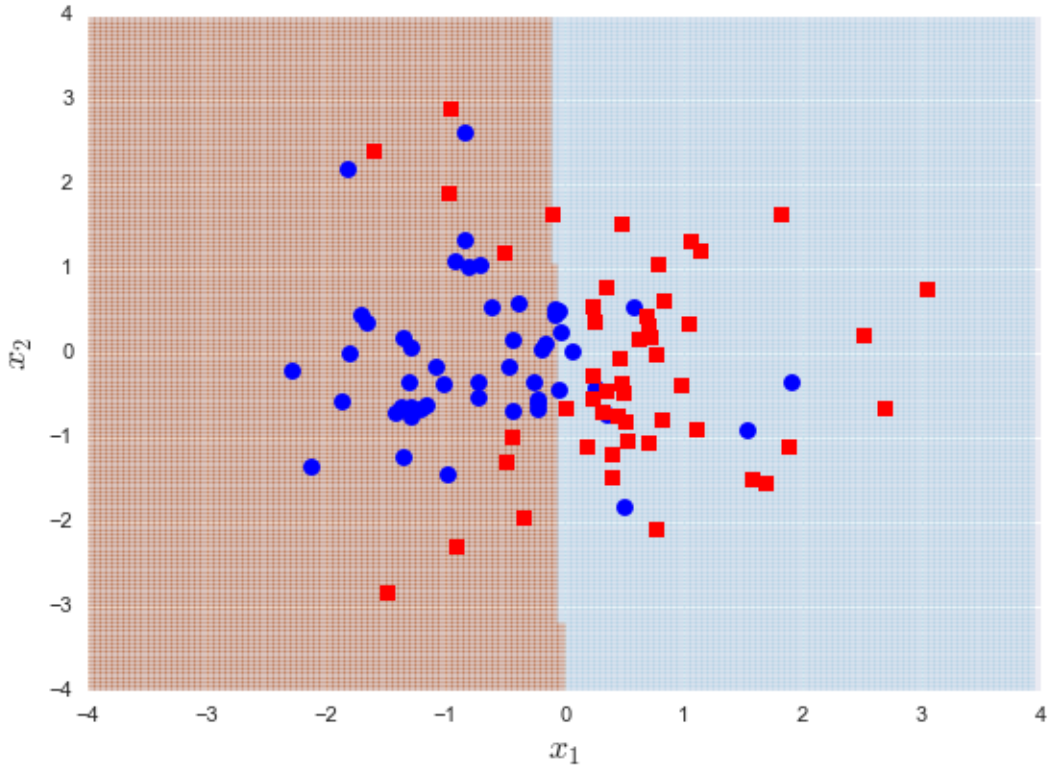
$$\vdots$$

$$h_{d_1} = W^{(1)}_{d_1,1}x_1 + W^{(1)}_{d_1,2}x_2 + \cdots + W^{(1)}_{d_1,d}x_d + b_1$$

Multilayer perceptron (MLP)

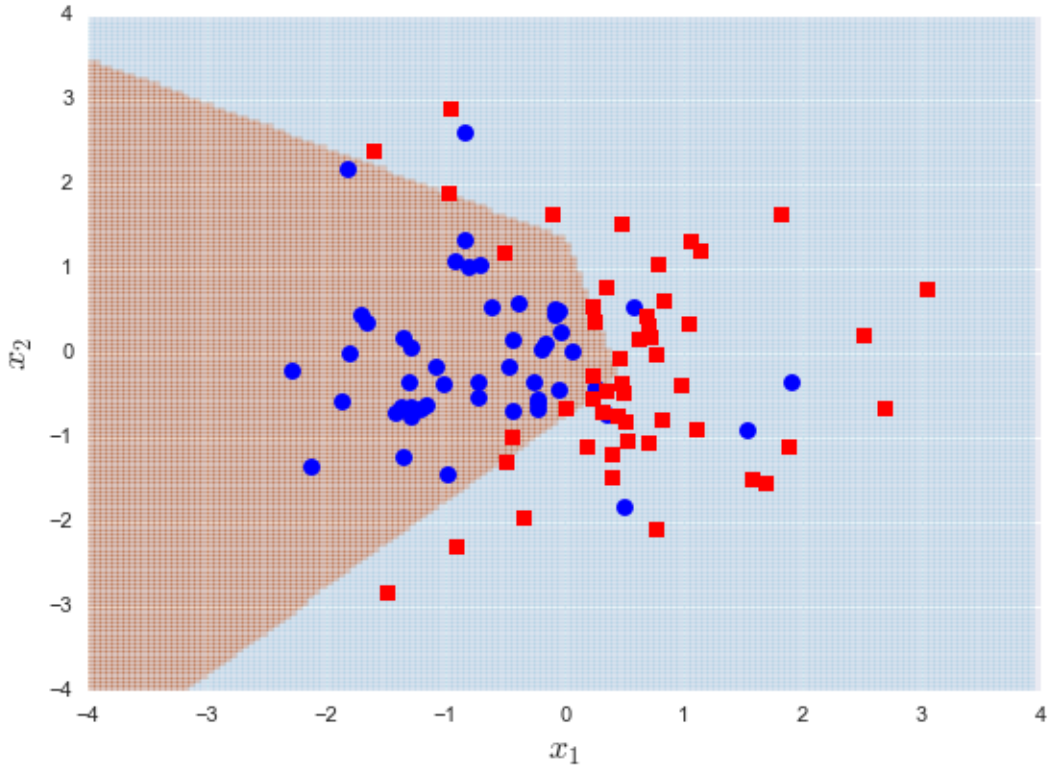
$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$$

Linear separation



$$f(\mathbf{x};\theta) = f_2(f_1(\mathbf{x};\theta_1);\theta_2)$$

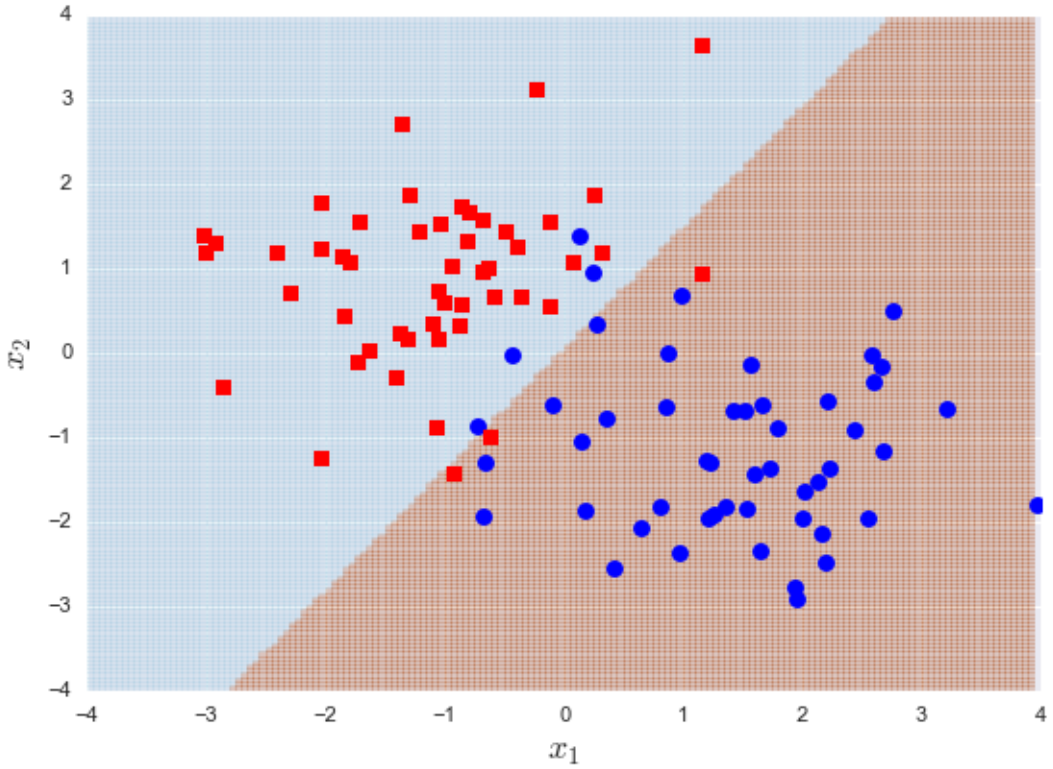
Non-linear separation



Multilayer perceptron (MLP)

$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$$

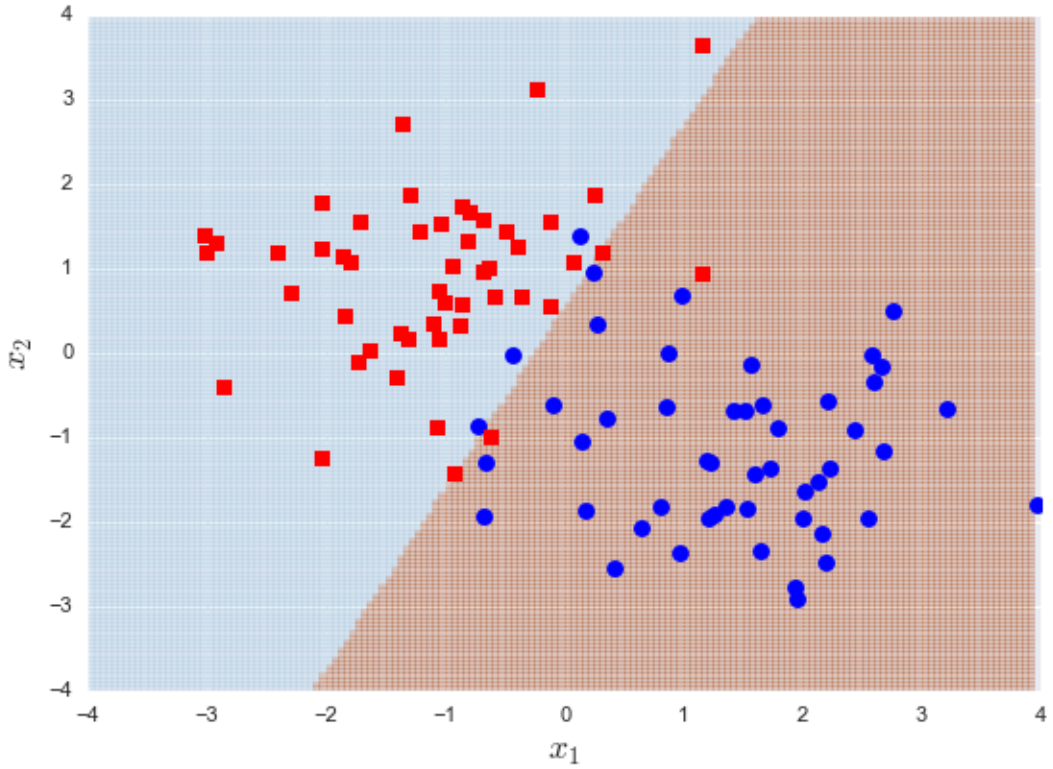
Linear separation



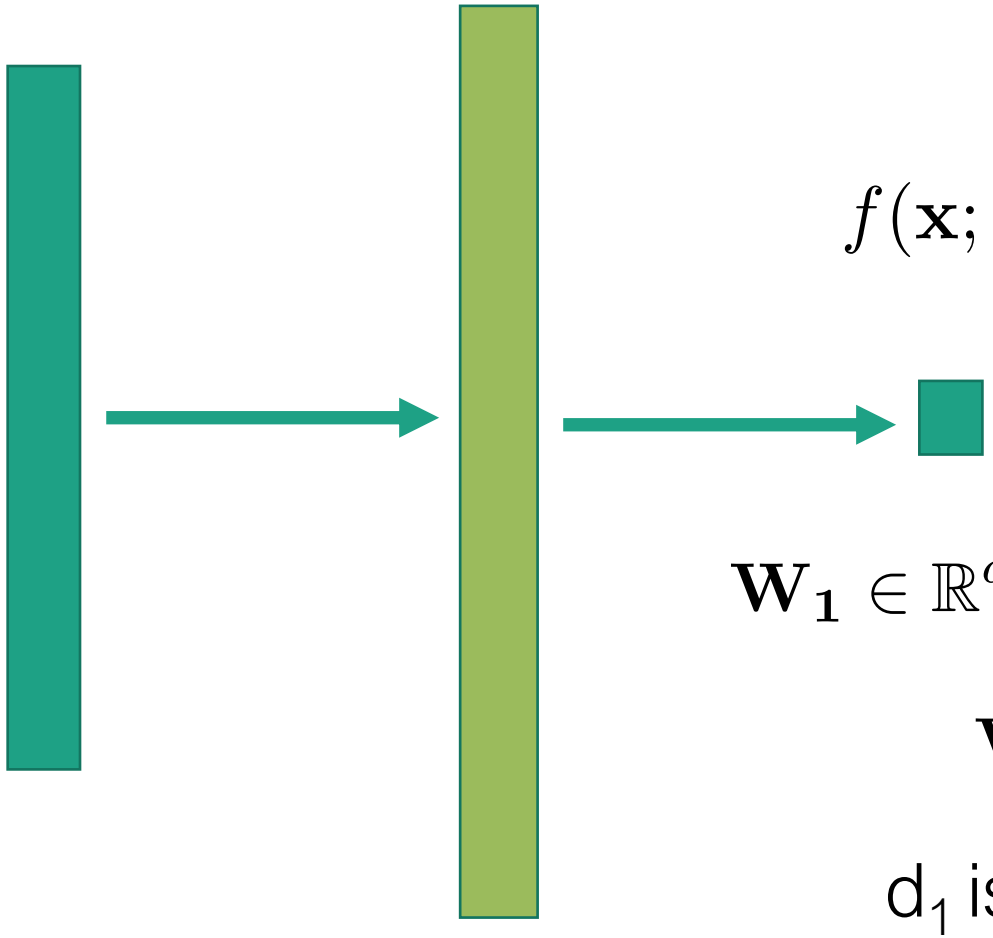
More powerful model

$$f(\mathbf{x};\theta) = f_2(f_1(\mathbf{x};\theta_1);\theta_2)$$

Able to do linear separation



MLP - width



$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$$

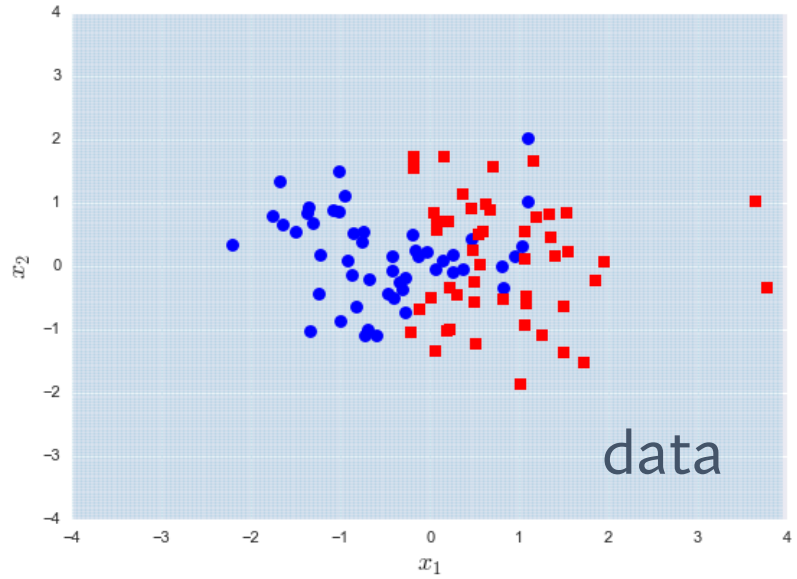
$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + b_1) + b_2)$$

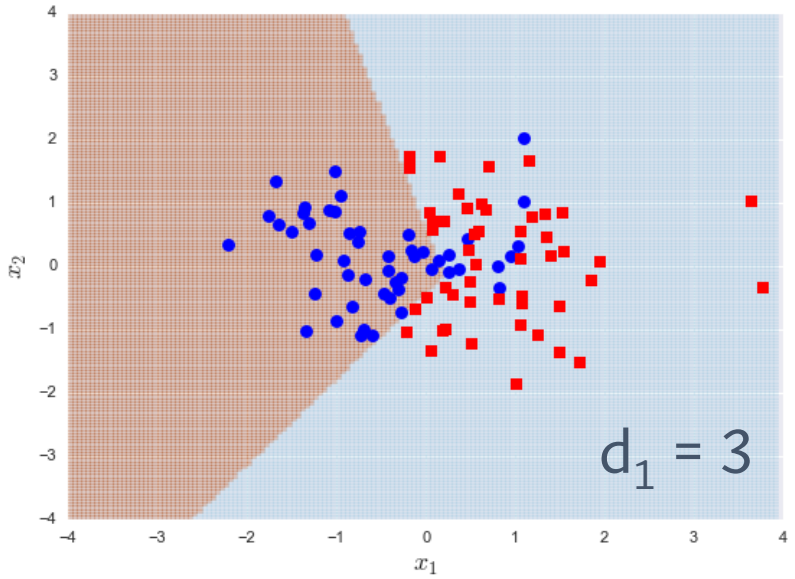
$$\mathbf{W_1} \in \mathbb{R}^{d_1 \times d} \implies \sigma(\mathbf{W}_1\mathbf{x} + b_1) \in \mathbb{R}^{d_1}$$

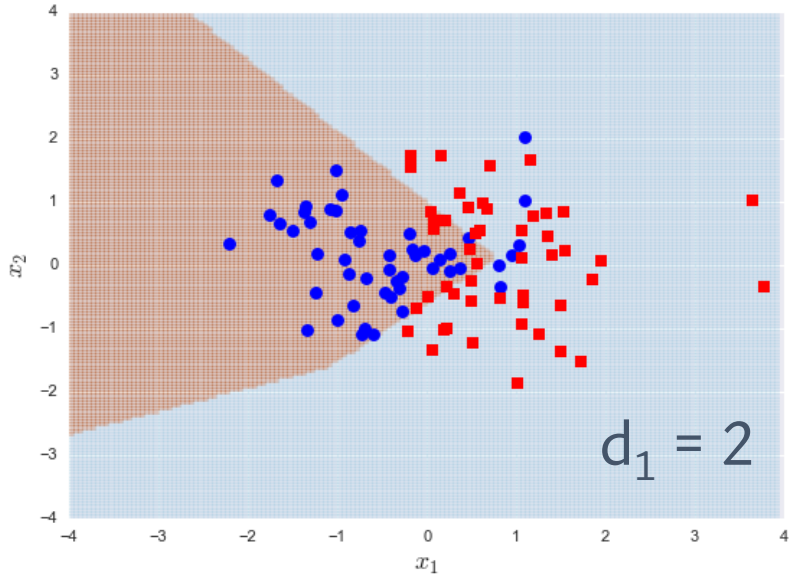
$$\mathbf{W_2} \in \mathbb{R}^{1 \times d_1} \implies f(\mathbf{x};\theta) \in \mathbb{R}$$

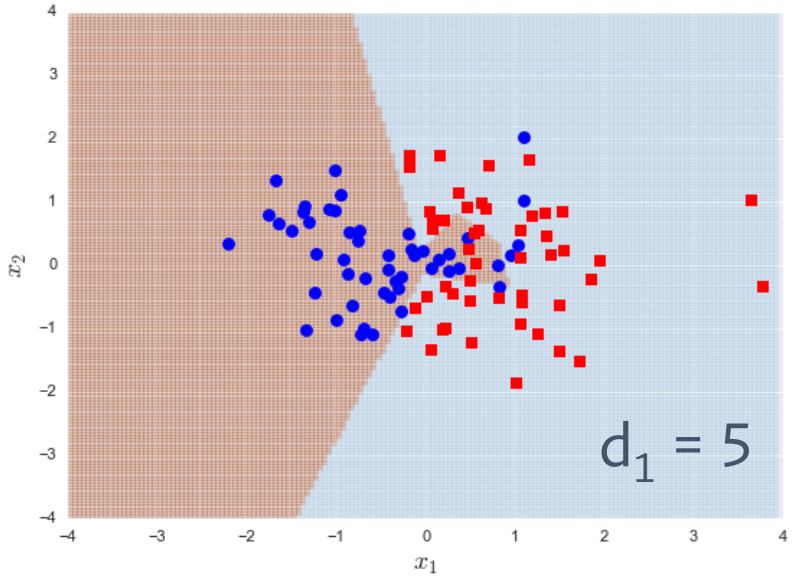
d_1 is the width of the hidden layer

Decision boundary at different width

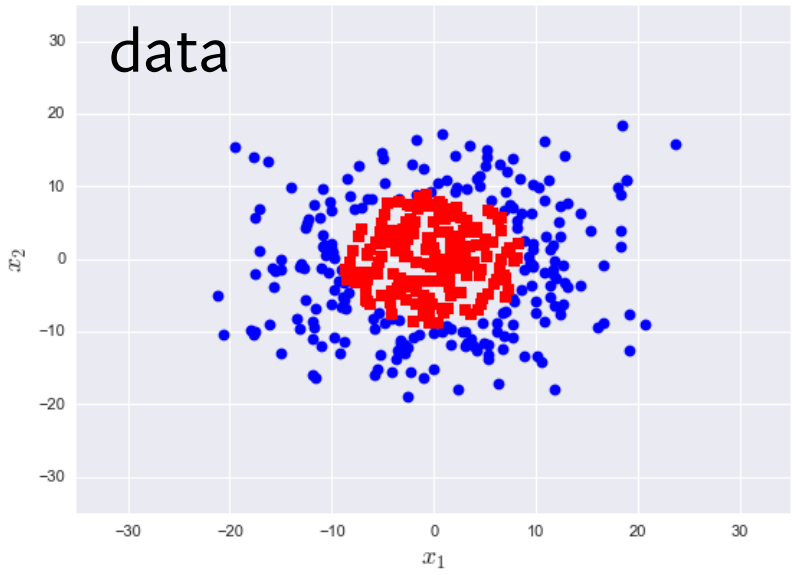


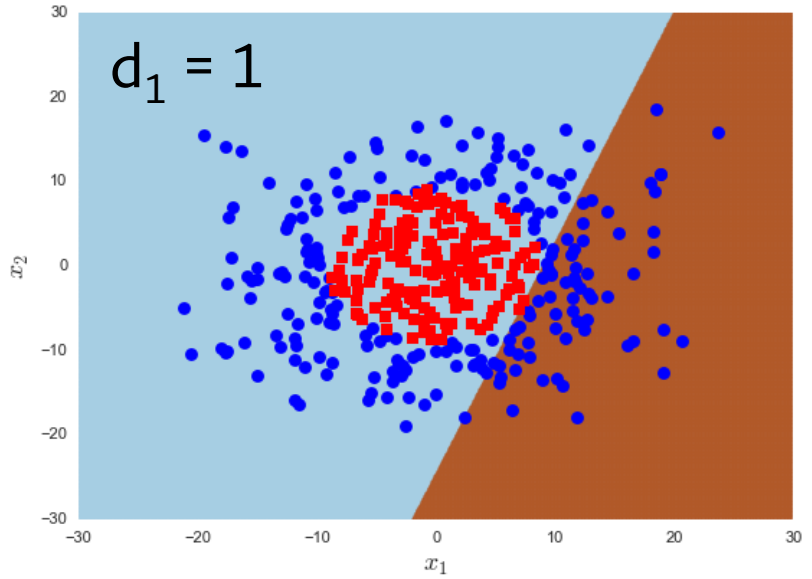


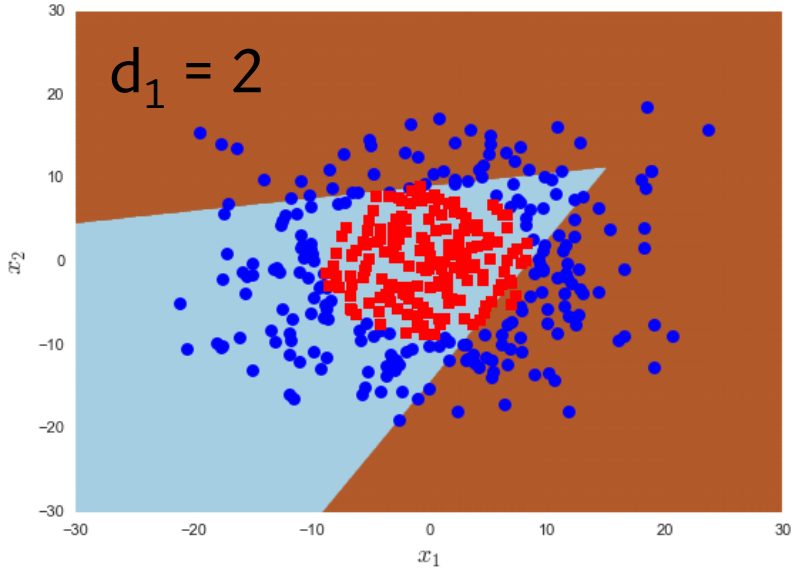


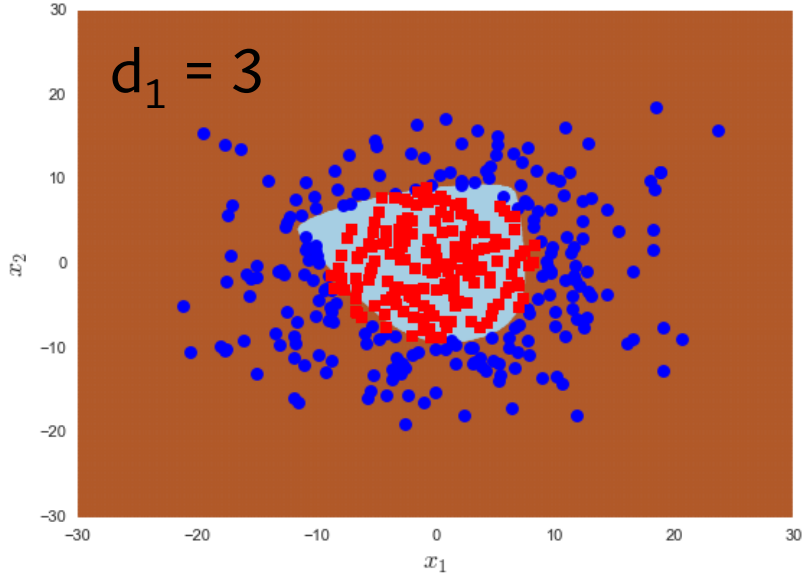


Decision boundary at different width





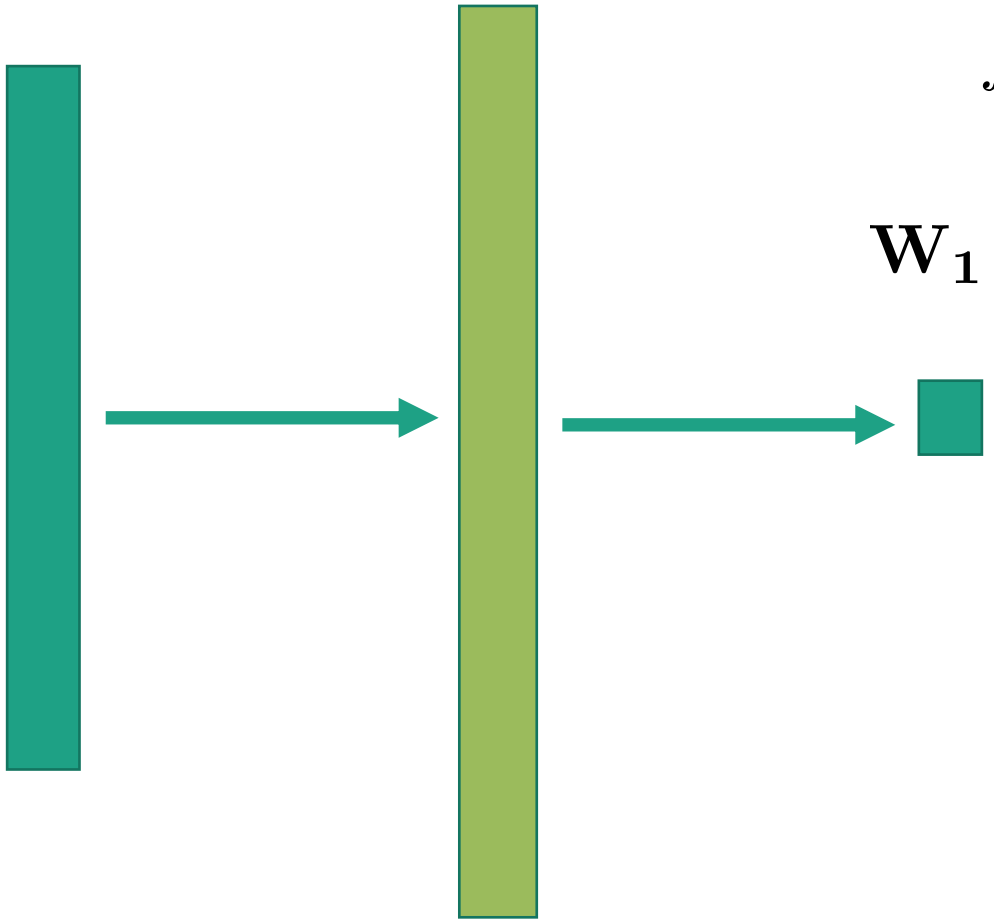




MLP – notes on width

- $d_1 = d$ -- only nonlinear transformation
- $d_1 > d$ -- mapping to a higher dimensional space
  - often it becomes easier to separate classes in higher dimensions
  - able to model complicated decision boundaries
  - Larger number of model parameters
- $d_1 < d$ – compression / bottleneck – possible information loss
  - becomes interesting for determining low-dimensional representations

Number of parameters increased



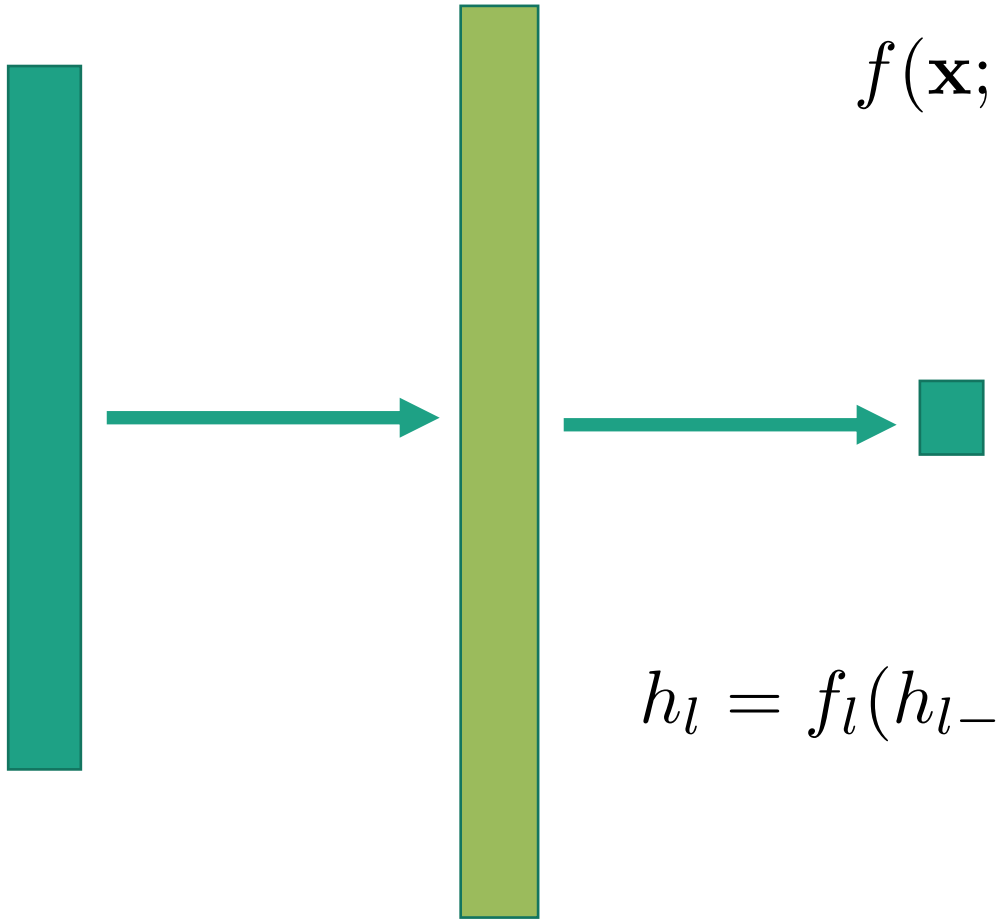
$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + b_1) + b_2)$$

$$\mathbf{W_1} \in \mathbb{R}^{d_1 \times d} \implies \sigma(\mathbf{W}_1\mathbf{x} + b_1) \in \mathbb{R}^{d_1}$$

$$\mathbf{W_2} \in \mathbb{R}^{1 \times d_1} \implies f(\mathbf{x};\theta) \in \mathbb{R}$$

W_1: has $d_1$xd parameters
b_1: is only 1 parameter
W_2: has d_1x1 parameters
b_2: is only 1 parameter
Total: $d_1$xd + d_1 + 2 parameters

MLP - depth



Network with one hidden layer

$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + b_1) + b_2)$$

Hidden layers

$$h_l = \sigma(\mathbf{W}_l h_{l-1} + b_l)$$

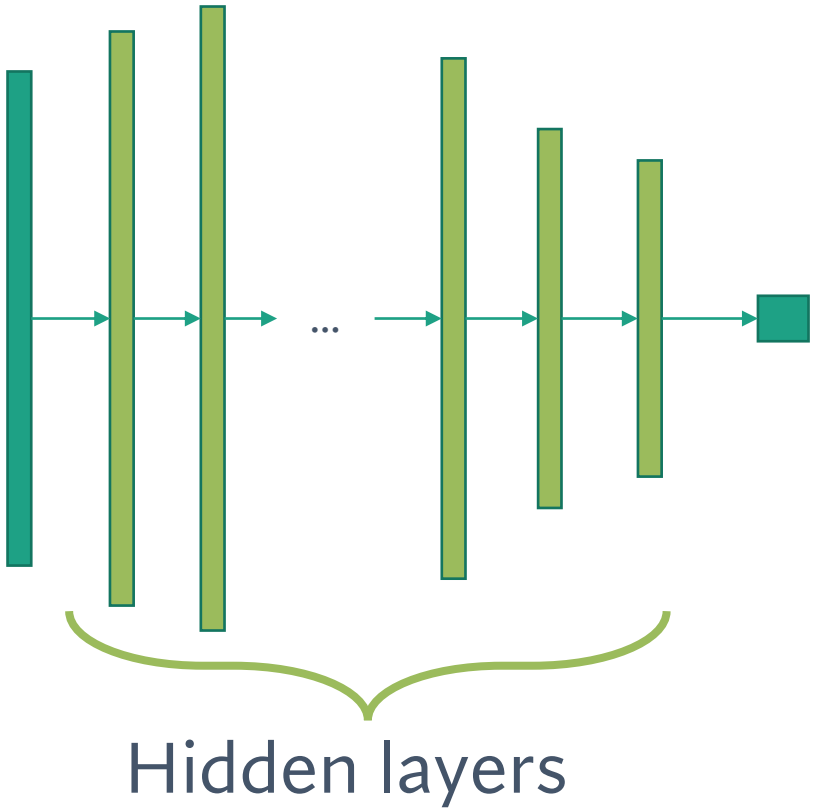
$$h_0 = \mathbf{x},\ h_2 = f(\mathbf{x};\theta)$$

Function view

$$h_l = f_l(h_{l-1};\theta_l) = f_l(f_{l-1}(h_{l-2},\theta_{l-1});\theta_l)$$

$$h_2 = f(\mathbf{x};\theta) = f_2 \circ f_1(\mathbf{x})$$

MLP – increasing depth



Network with L-1 hidden layer

Hidden layers

$$h_l = \sigma(\mathbf{W}_l h_{l-1} + b_l)$$

$$h_0 = \mathbf{x},\ \ h_L = f(\mathbf{x};\theta)$$

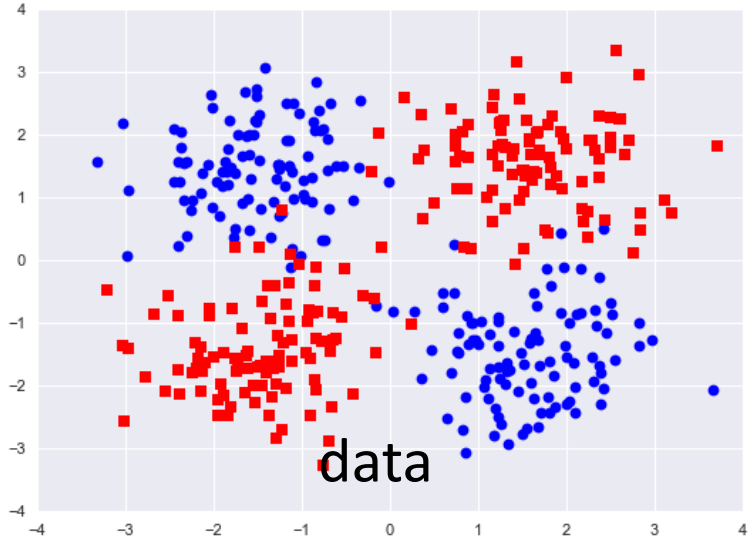
Function view

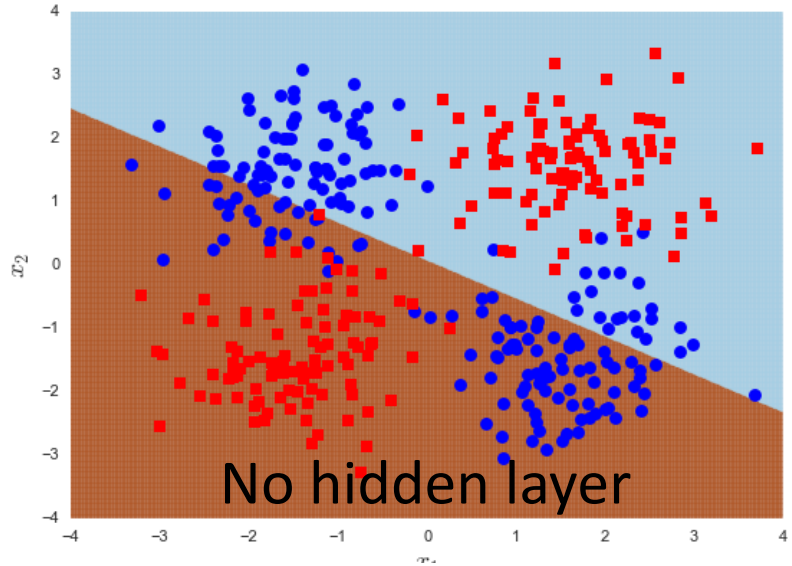
$$f_l(h_{l-1};\theta_l) = \sigma(\mathbf{W}_l h_{l-1} + b_l)$$

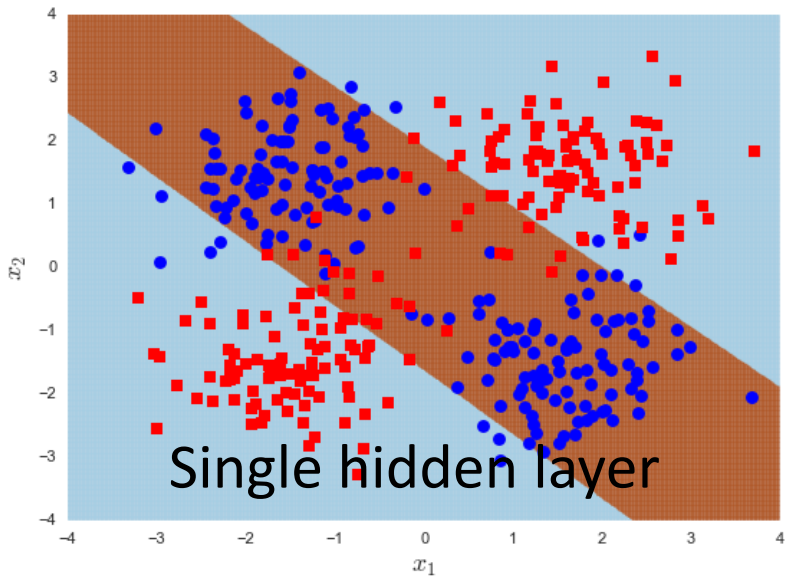
$$f(\mathbf{x};\theta) = f_L \circ \cdots \circ f_2 \circ f_1(\mathbf{x})$$

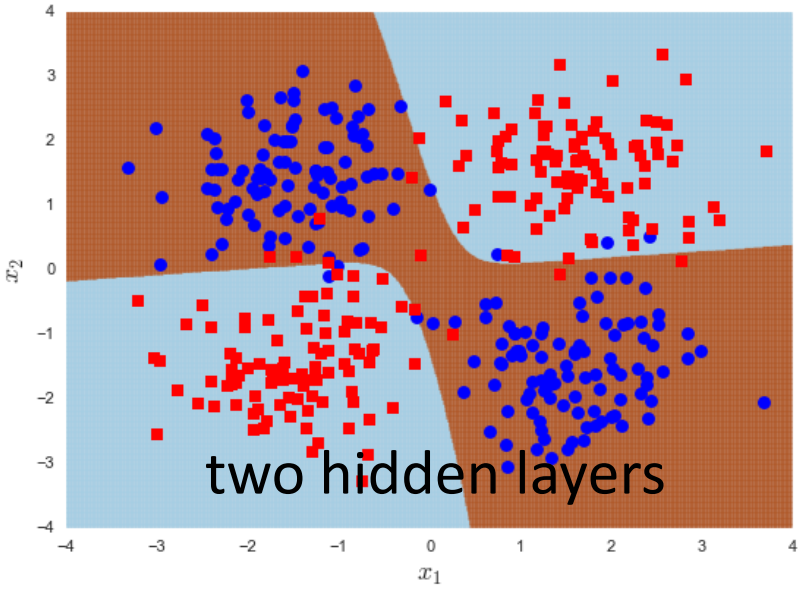
$$\theta = \{\theta_L, \ldots, \theta_1\}$$

Decision boundary at different depths [width = 2 at all depths]





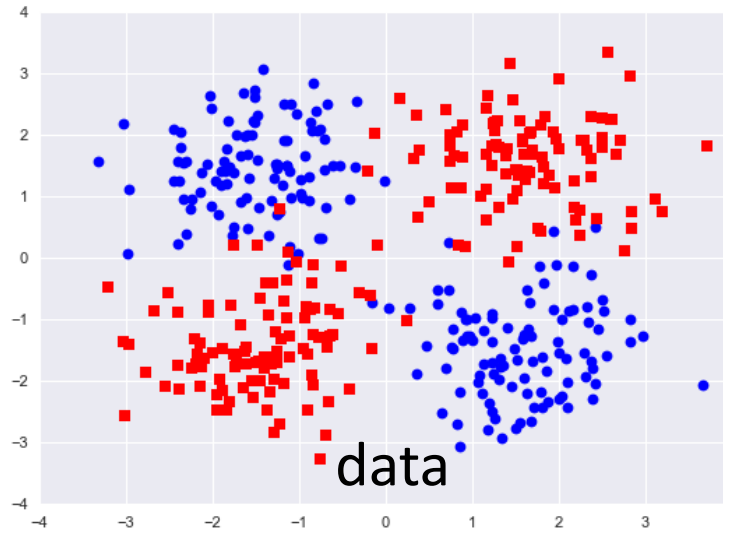


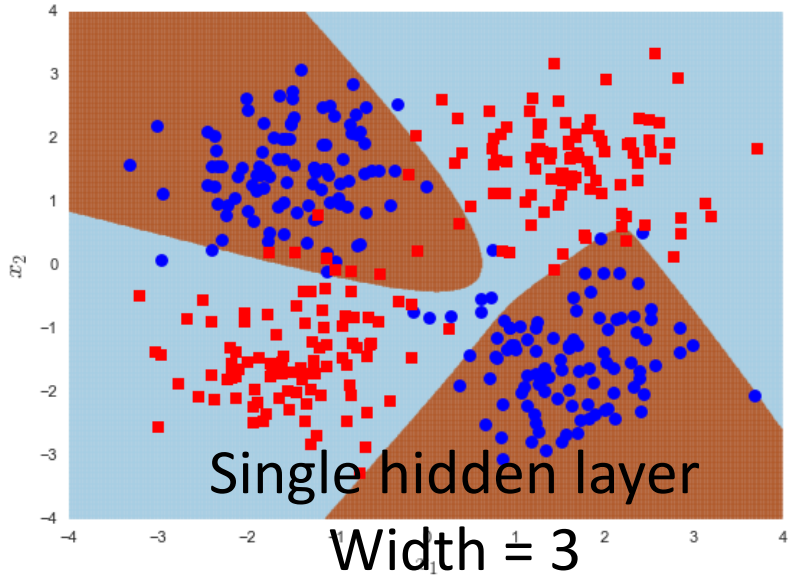


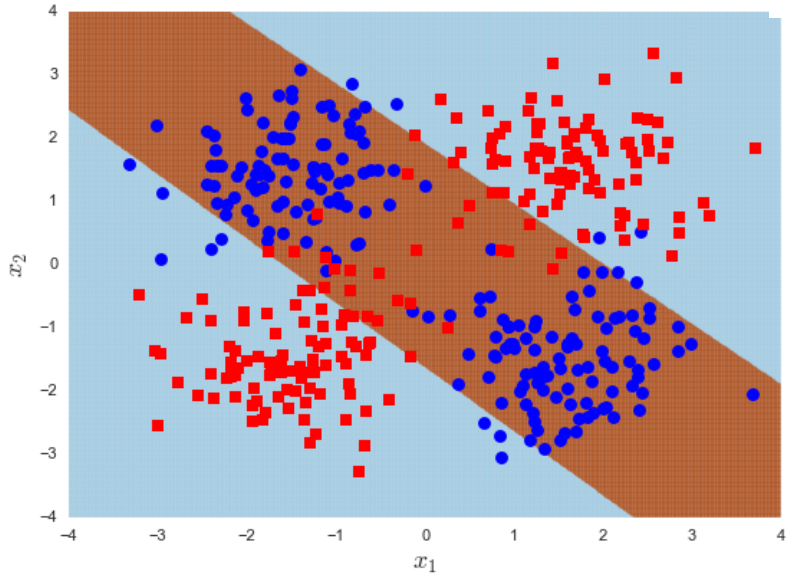
MLP – notes on depth

- No hidden layer – perceptron / logistic regression / linear
- Increasing depth
  - Allows more complicated decision boundaries
  - More powerful models
  - Leads to larger number of model parameters
  - Needs more samples to fit reliably
- May become difficult to train

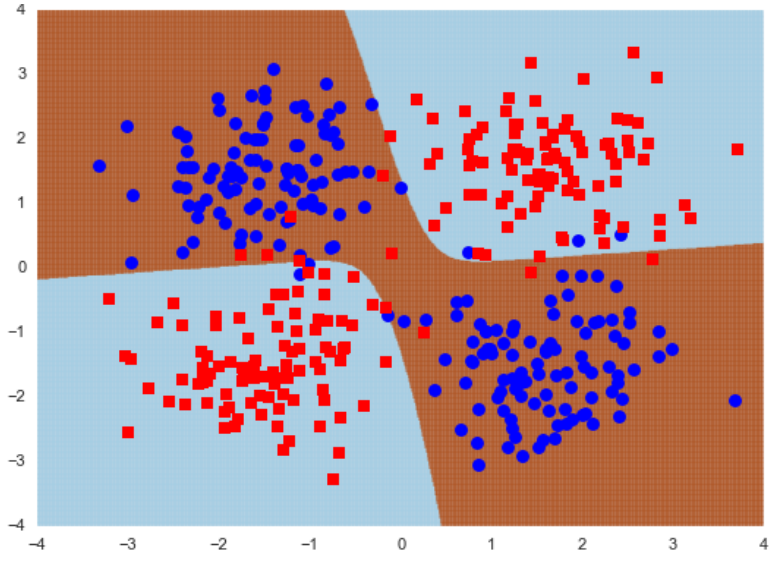
Depth / width interaction







Single hidden layer
Width = 2



Two hidden layers
Widths = 2

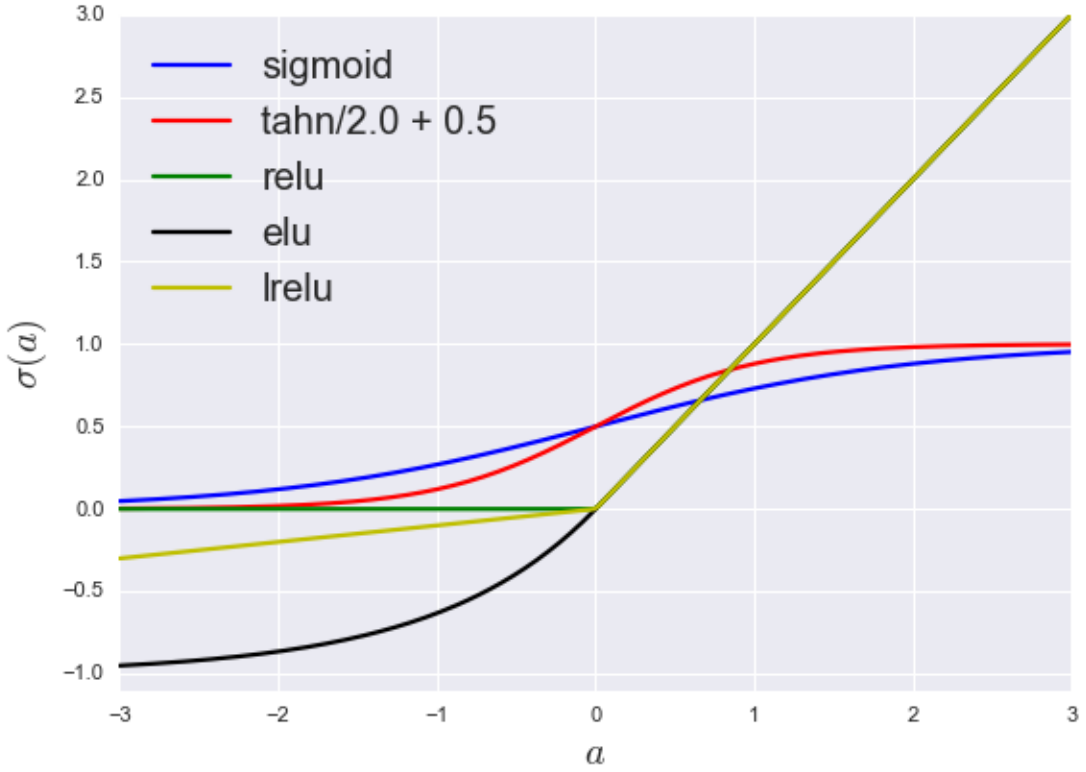
MLP – notes on depth and width

- Complicated interaction between depth and width
- Both allow modeling more complicated decision boundaries
- Choices of depth and width are *architectural* design choices.
- No accepted, widely used method for automatically determining architecture for a given problem
- Problem specific – mostly trial and error-based strategy
- A huge search space
- Engineering / intuition / art
- Prediction accuracy on a validation set

Non-linearity – more recent models

Often used in connections between internal layers

Element-wise application



Rectified Linear Unit (Relu)

$$\sigma(a) = \begin{cases} a, & a \geq 0 \\ 0, & a < 0 \end{cases}$$

Leaky Relu

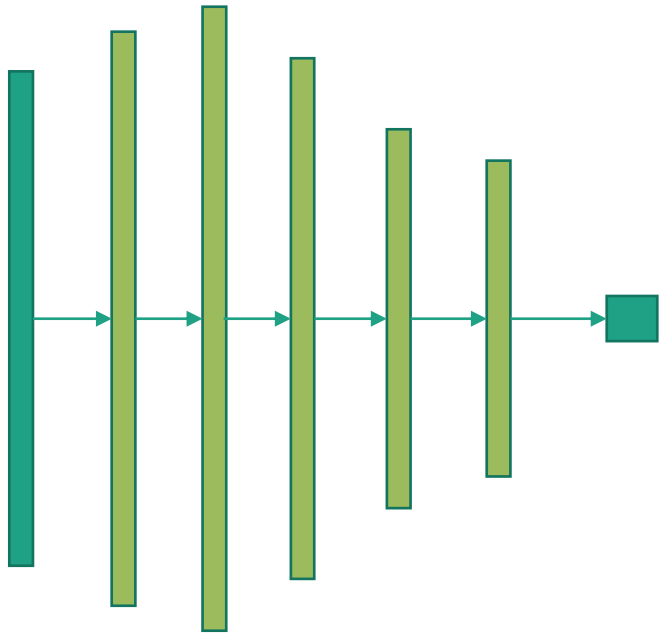
$$\sigma(a) = \begin{cases} a, & a \geq 0 \\ \alpha a, & a < 0 \end{cases}$$

Exponential Linear Unit (Elu)

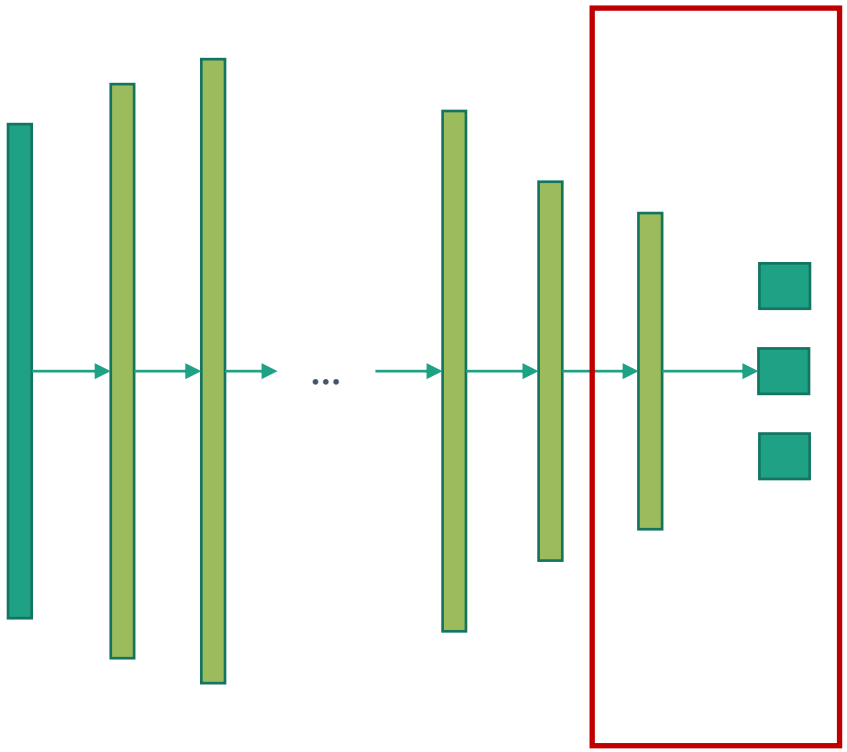
$$\sigma(a) = \begin{cases} a, & a \geq 0 \\ \exp(a) - 1, & a < 0 \end{cases}$$

Fully connected architecture

- This is called a fully connected architecture.
- Every neuron in a layer is connected to every neuron in the subsequent layer.



Fully Connected Classification Network



- Multilayer perceptron model with multiple outputs
- In the generic case, applies to multi-label classification
- Last layer of the network is constructed to make the network perform binary or multi-class classification
- So, the final layer is either a sigmoid function or a *soft-max* function that is designed for multi-label problems

Soft-max function

Binary classification: $a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}$

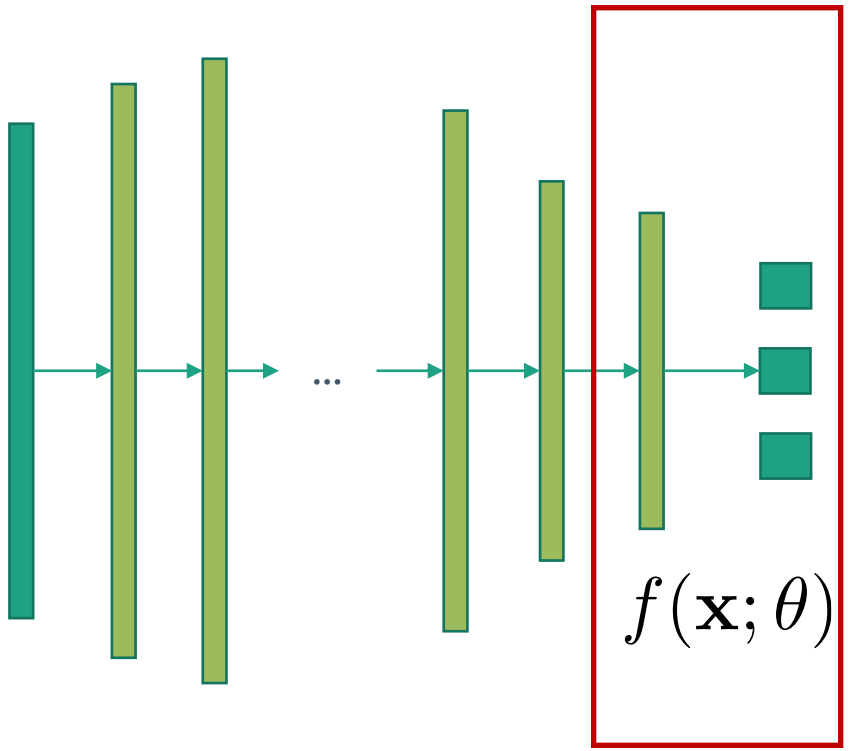
$$p(y=1) = f(\mathbf{x};\theta) = \frac{1}{1+e^{-a_L}},\ p(y=0) = 1 - p(y=1)$$

Multi-label classification – K classes: $a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}^K$

$$p(y=k) = f_k(\mathbf{x};\theta) = \frac{e^{a_{L,k}}}{\sum_{k' \in \mathcal{C}} e^{a_{L,k'}}} \quad \sum_{k \in \mathcal{C}} p(y=k) = 1$$

Soft-max function

Fully Connected Regression Network

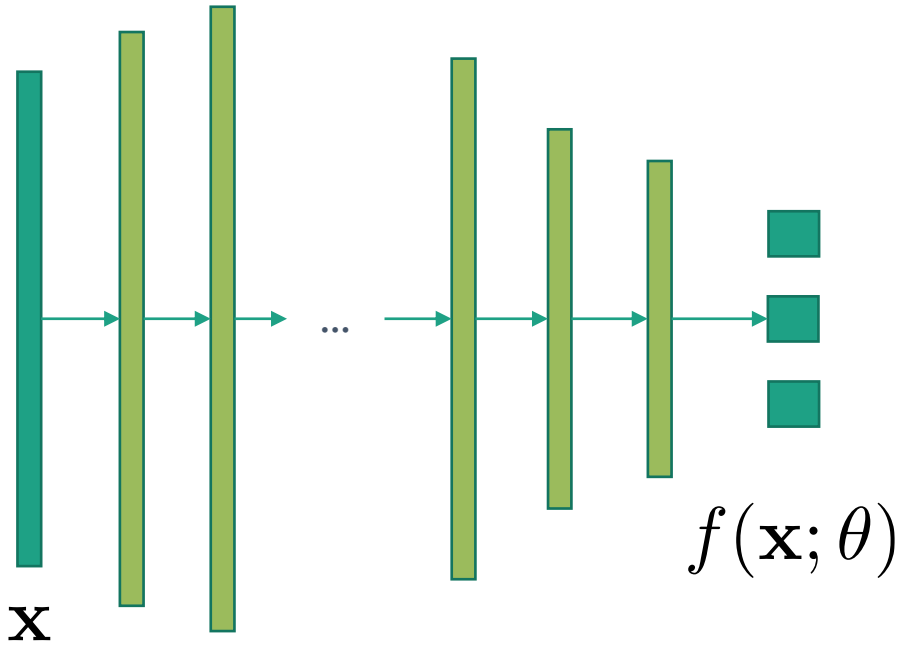


- Regression network is very similar to the classification network
- For an M dimensional regression problem

$$y \in \mathbb{R}^M \qquad f(\mathbf{x};\theta) \in \mathbb{R}^M$$

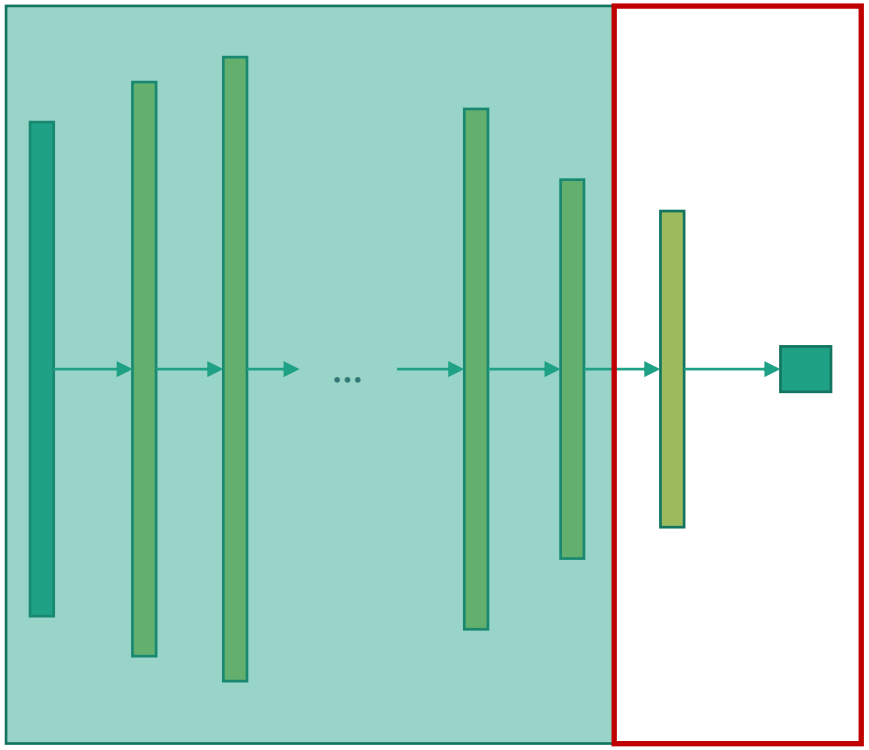
- It is common to not use any non-linear activation at the final layer

Information flow is forward while predicting



- Flow of information if *forward* when predicting
- The input is fed through the layers successively until the outputs
- *Feed-forward network architecture*

MLP – feature space view



- There is a linear model at the very final layer

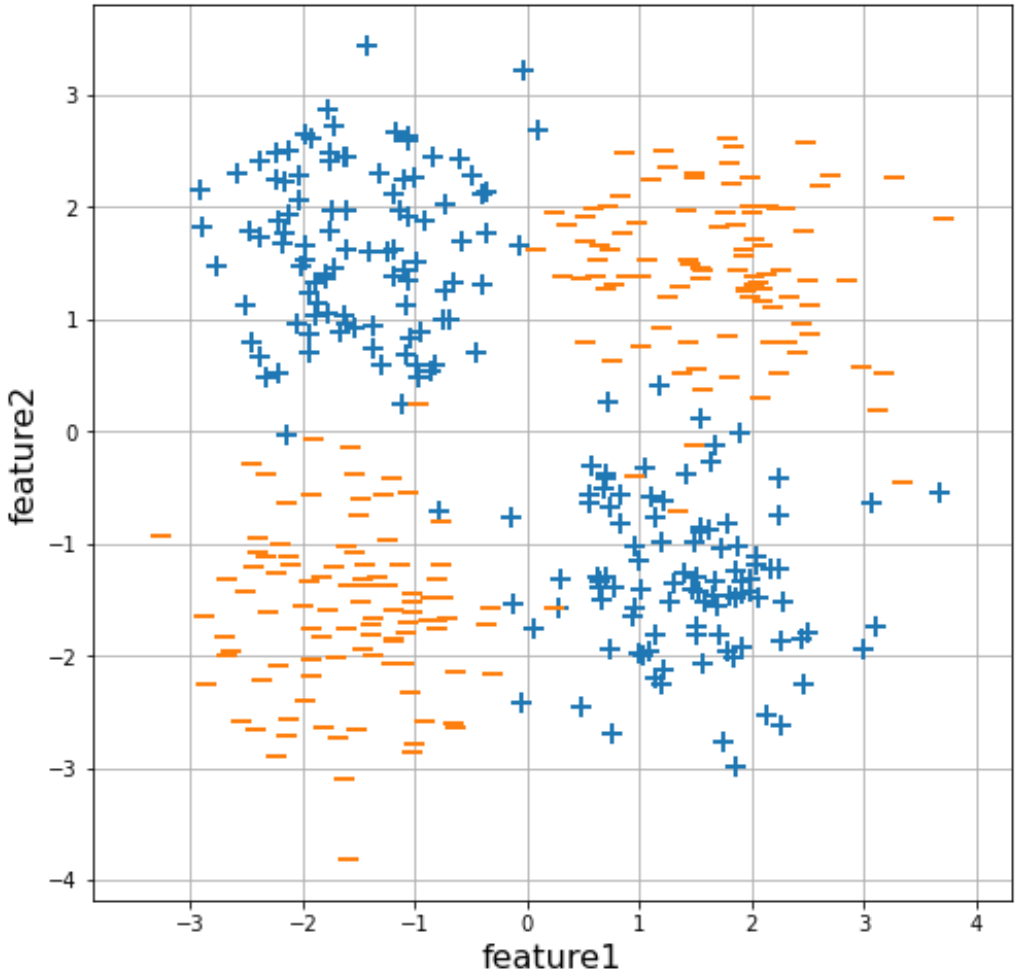
$$f(\mathbf{x};\theta) = f_L(\phi(\mathbf{x});\theta_L) = \sigma(\mathbf{W}_L\phi(\mathbf{x}) + b_L)$$

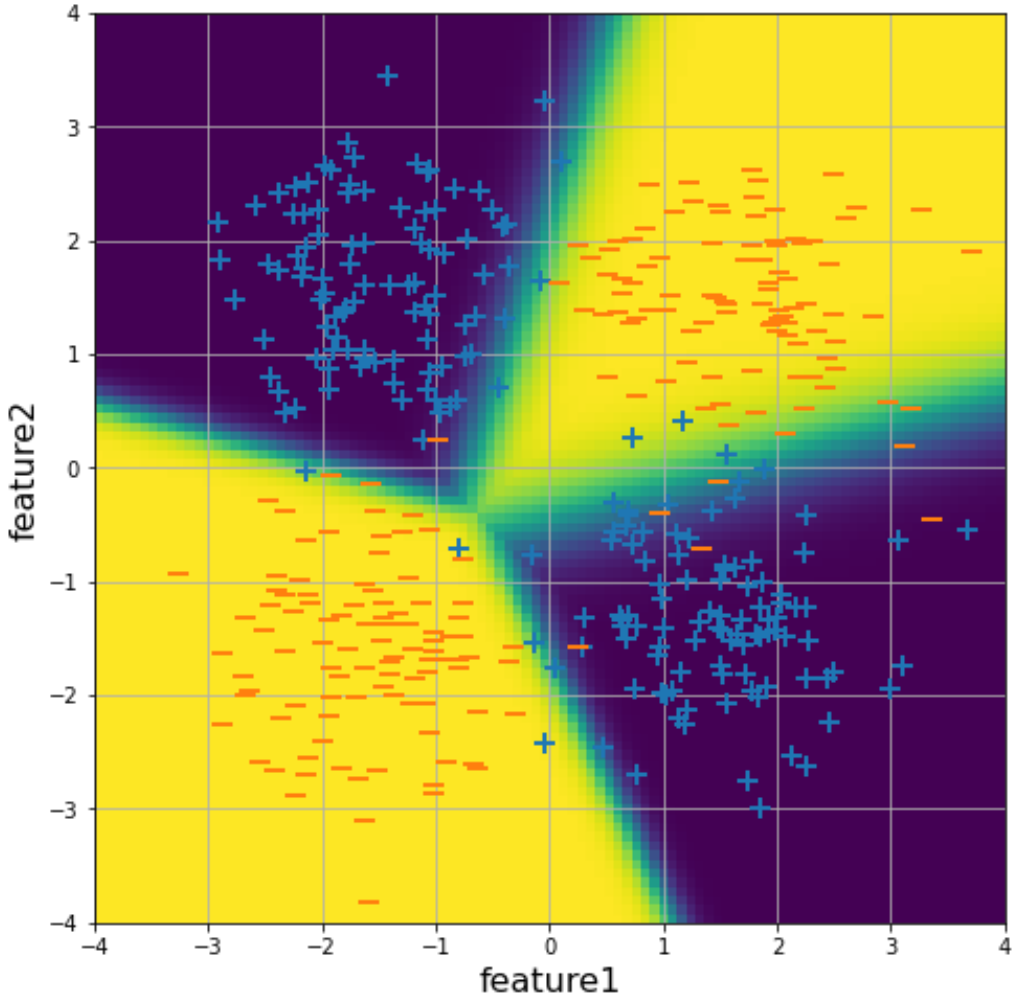
- There is a feature map that transforms the input

$$\phi(\mathbf{x}) : \mathbb{R}^d \mapsto \mathbb{R}^{d_{L-1}}$$

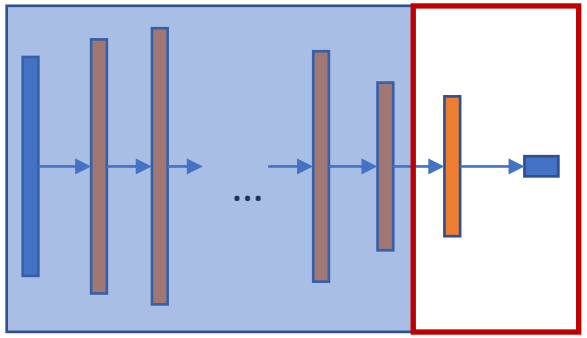
- Automatically determined non-linear feature map

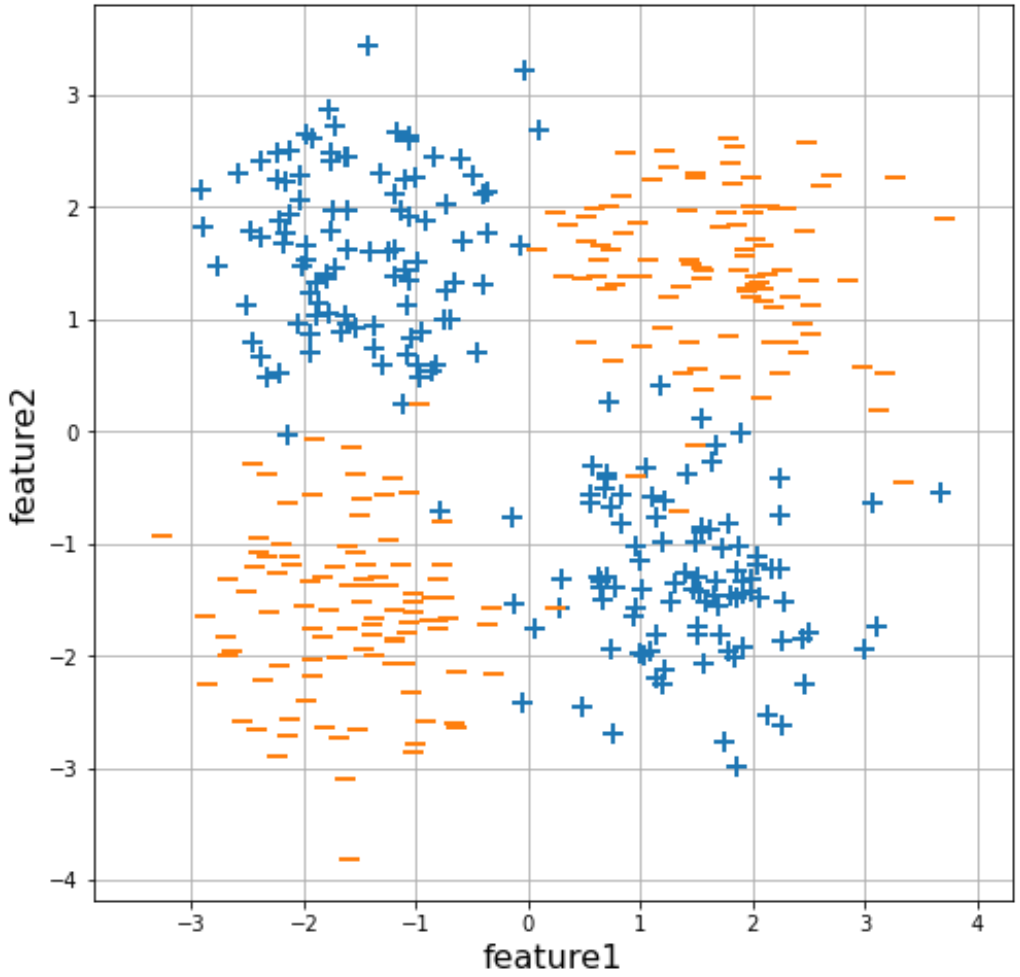
What does that mean?

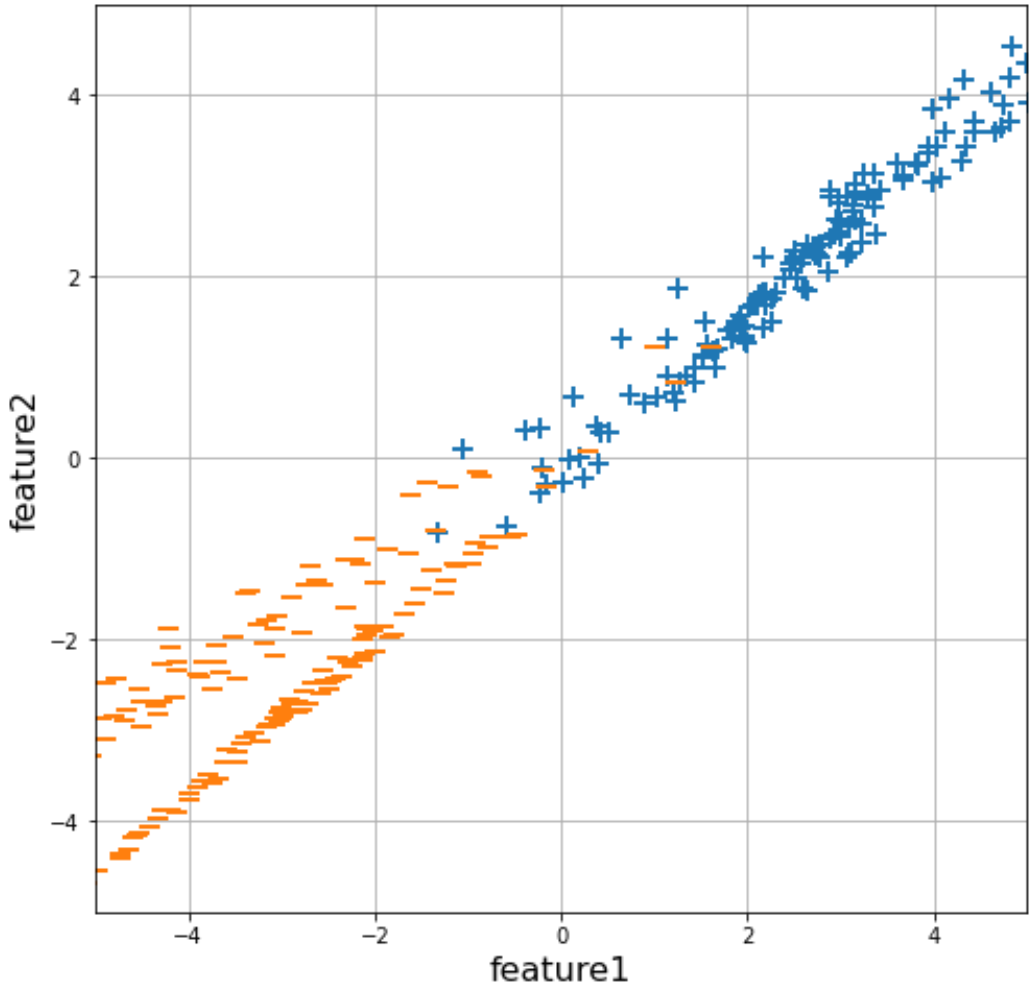




What does that mean?







MLP – why non-linear feature map

- Non-linearity is due to the activation functions $\sigma(\cdot)$
- What happens if all activations were linear functions?

$$\sigma(a) = \mathbf{V}a + c$$

$$f_l(h_{l-1}) = \mathbf{V}\left(\mathbf{W}_l h_{l-1} + b\right) + c = \tilde{\mathbf{W}} h_{l-1} + \tilde{b}$$

$$y = f_L \circ f_{L-1} \circ \cdots \circ f_1(\mathbf{x}) = \hat{\mathbf{W}}\mathbf{x} + \hat{b}$$

- The final map is effectively linear – same as a single linear model, such as logistic regression
- Non-linearity is extremely important for complicated models

The power of ANNs

- Universal approximation theorem: a feedforward network can accurately approximate any continuous function from one finite dimensional space to another, **given enough hidden units** (Hornik et al. 1989, Cybenko 1989).

- Therefore, ANNs have the potential to be universal approximators.

- However - universal approximation theorem does not provide any guarantee that training finds this representation. Subject to model tuning and computational power.

- In addition, there are many variants of ANN that are well-suited for different types of data (tableau, image, time series, etc.) without requiring data transformation. This is called end-to-end learning. Will discuss in the lecture of ‘unstructured data’.

Textbooks and tutorials

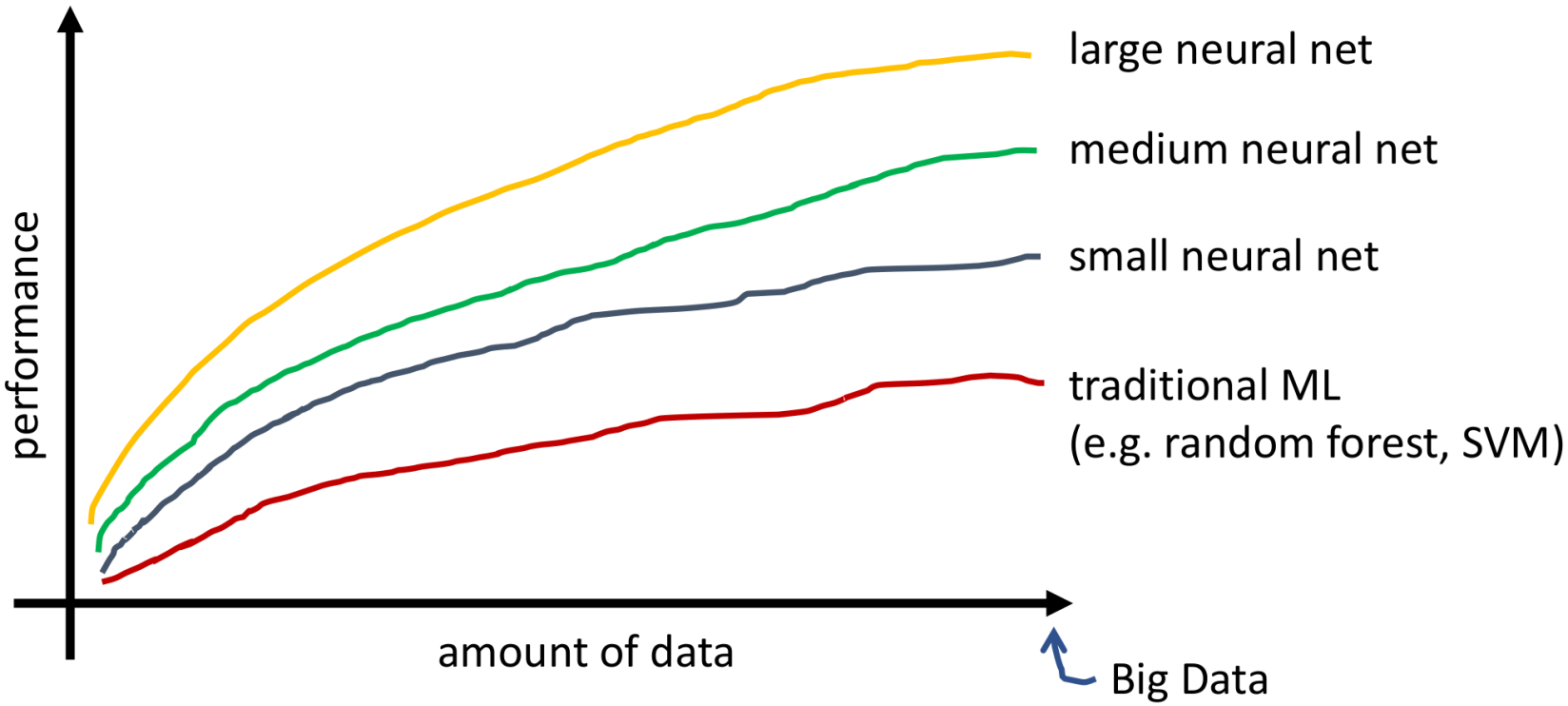
- VanderPlas, "Python data science handbook", O'Reilly, 2017, ISBN 9781491912058 (Example code)

- Geron (2nd Edition), "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow", O'Reilly, 2019, ISBN 9781492032649 (Example code)

- Scikit-Learn tutorial, VanderPlas

Deep learning and data size

The performance of DL increases rapidly with the size of the data.



Deep learning for images

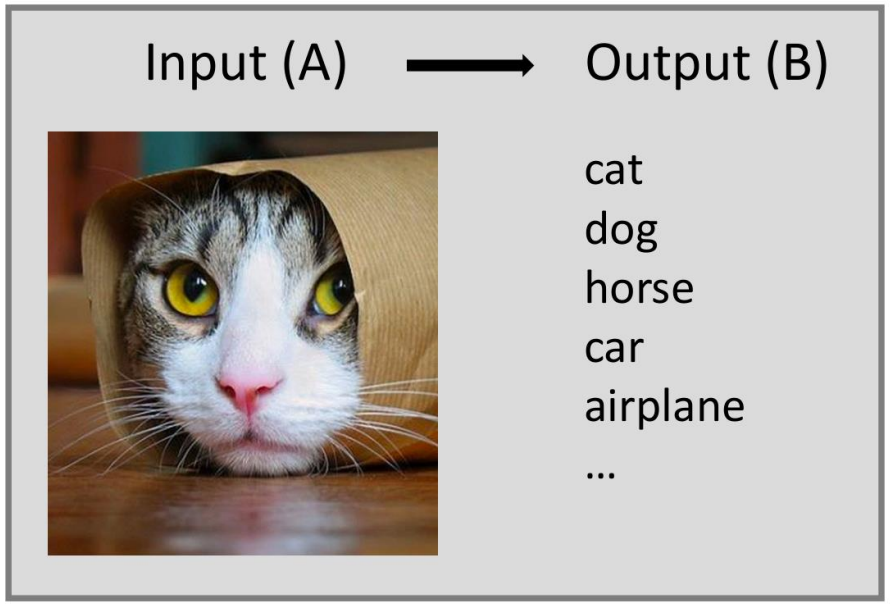
Convolutional Neural Networks (CNNs)

Deep learning for images (computer vision)

Applications

1. Image Classification
2. Image Classification with Localization
3. Object Detection
4. Semantic Segmentation
5. Instance Segmentation

Image Classification



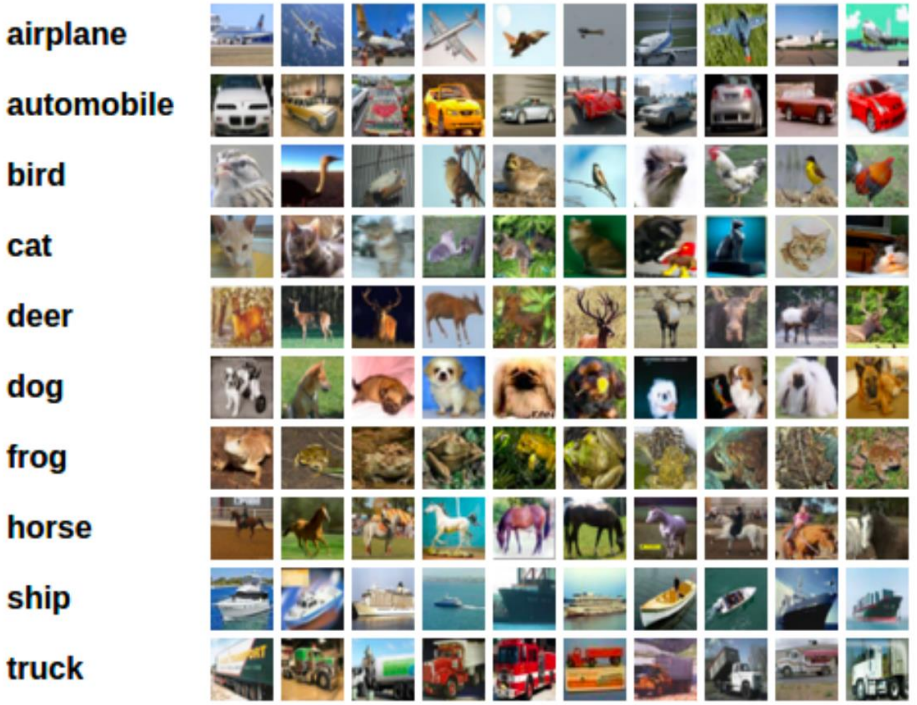
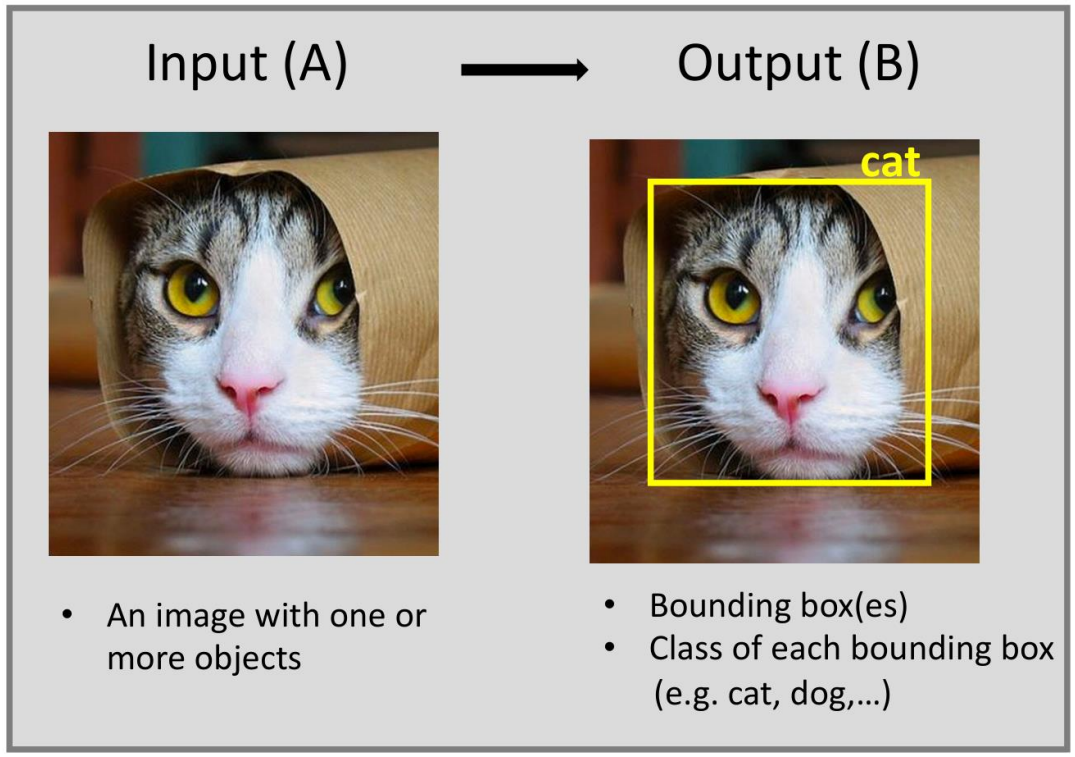


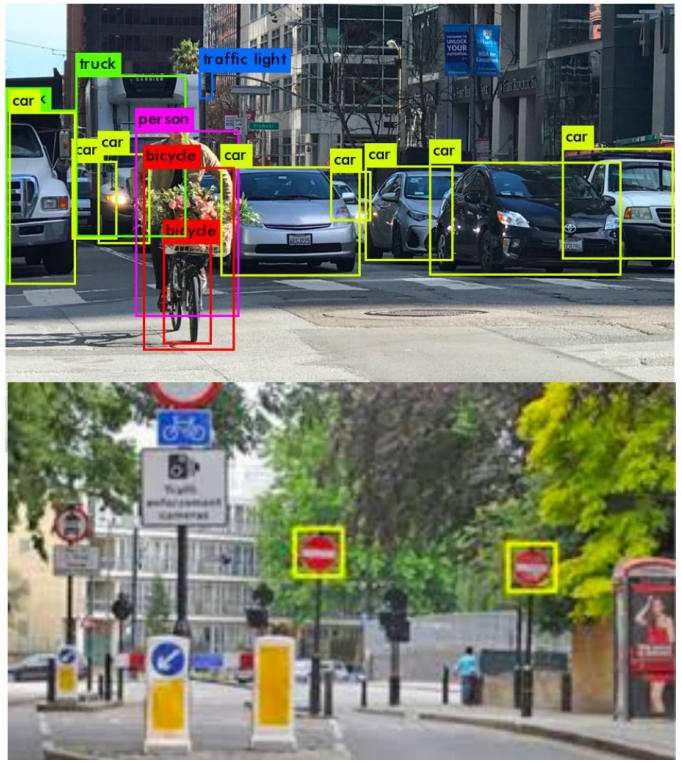
Image Classification with Localization

- Given an image we want to learn the class of the image and the class location in the image. We want a class and a rectangle of where that object is.
- Usually only one object is presented.



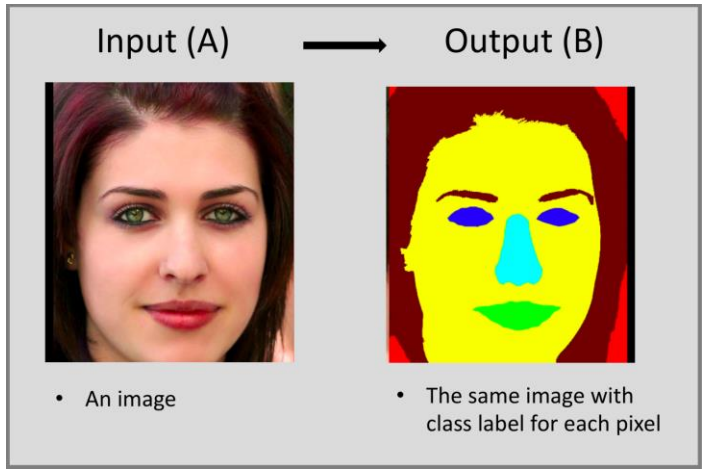
Object Detection

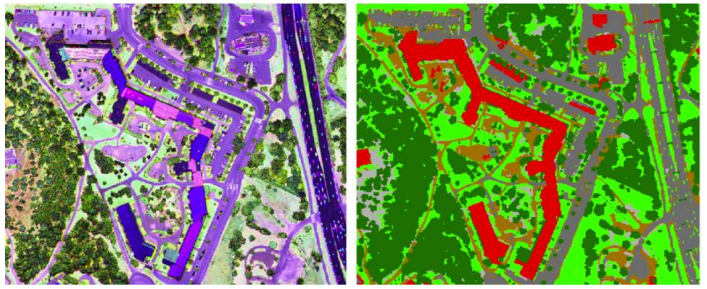
- We want to detect all objects in the image that belong to a specific class and give their boundary box.
- An image can contain more than one object with different classes.



Semantic Segmentation

- We want to label each pixel in the image with a category label.
- Semantic Segmentation don't differentiate instances, only care about pixel labels.
- It detects no objects just pixels – it does not tell you #cars in the image.
- If two objects of the same class are intersected, we won't be able to separate them.

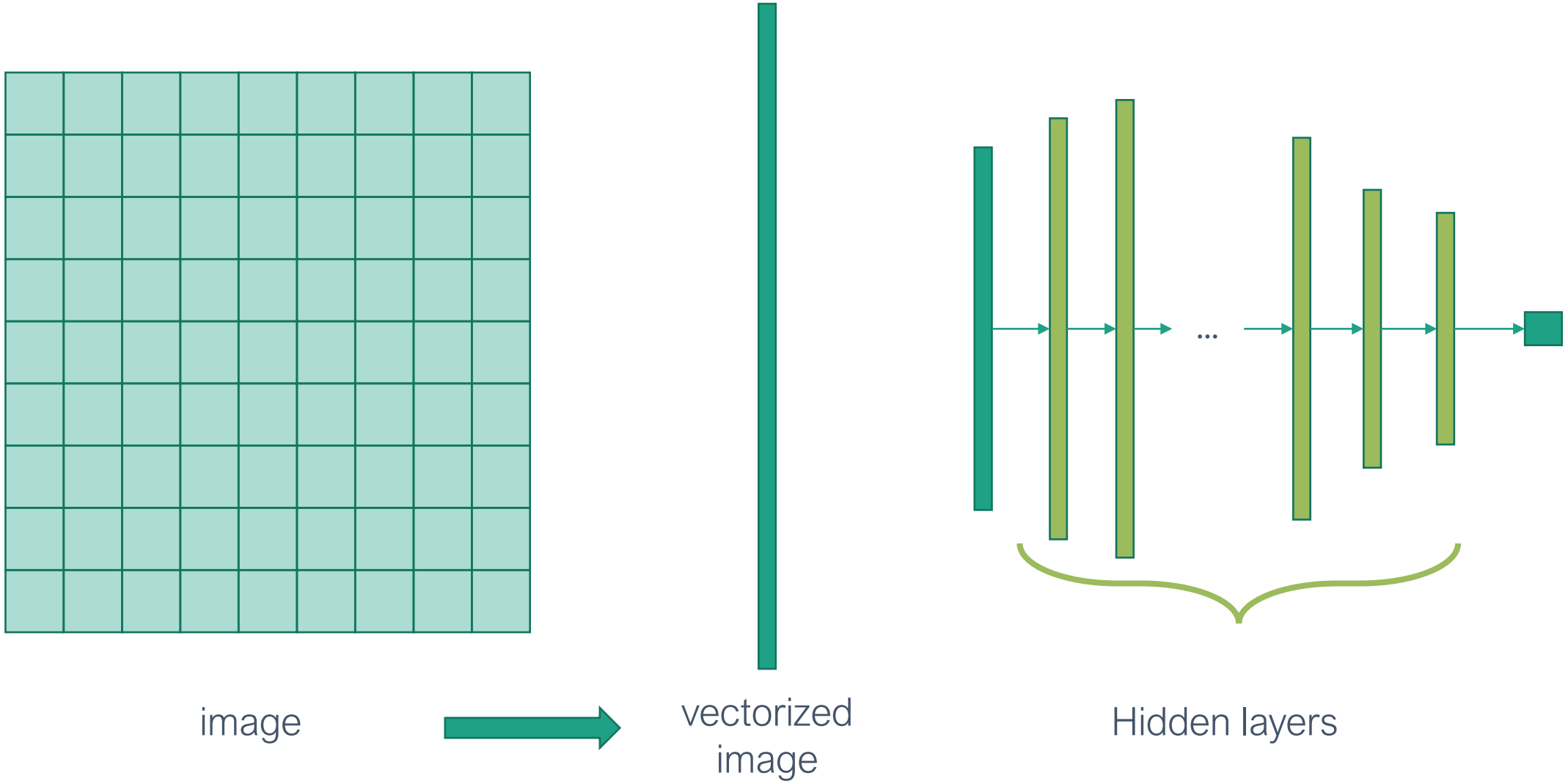




Instance Segmentation

- This looks like the full problem and the most challenging.
- We want to identify and distinguish all objects.

Naïve approach



Convolutional Neural Networks (CNNs) for images

Images can be large. CNN is fast and accurate to work with images.

E.g. Comparing FCNN and CNN with similar number of output features (~1000)

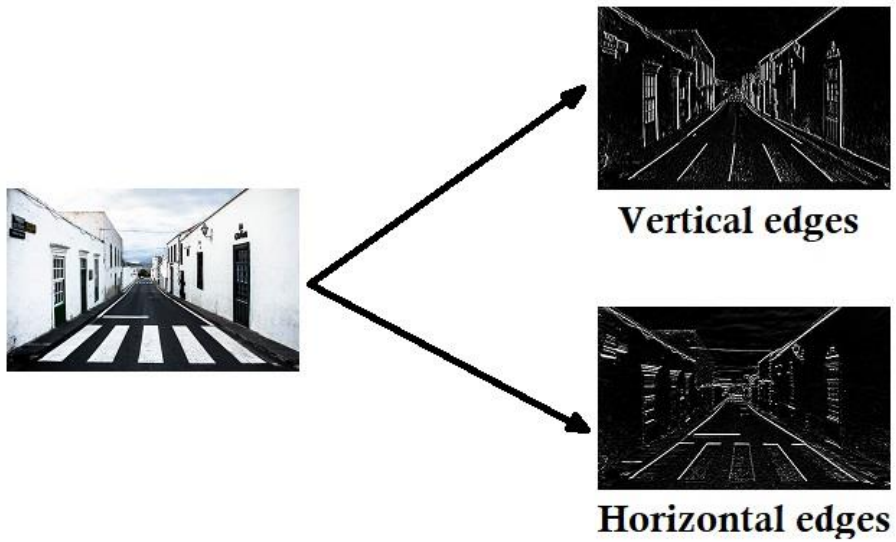
Given a 1000*1000 image (having 10e6 pixels/features), if a FCNN is used (1 hidden layer with 1000 neurons), then there are 10e9 parameters to learn. Very challenging.

In contrast, if we use a CNN with 967*967 filter, the output has 33*33 features (close to 1000) and only 10e6 parameters to learn.

CNNs for images

Parameter sharing

A filter is actually a feature detector. A filter that is useful in one part of the image is probably useful in another part of the image.

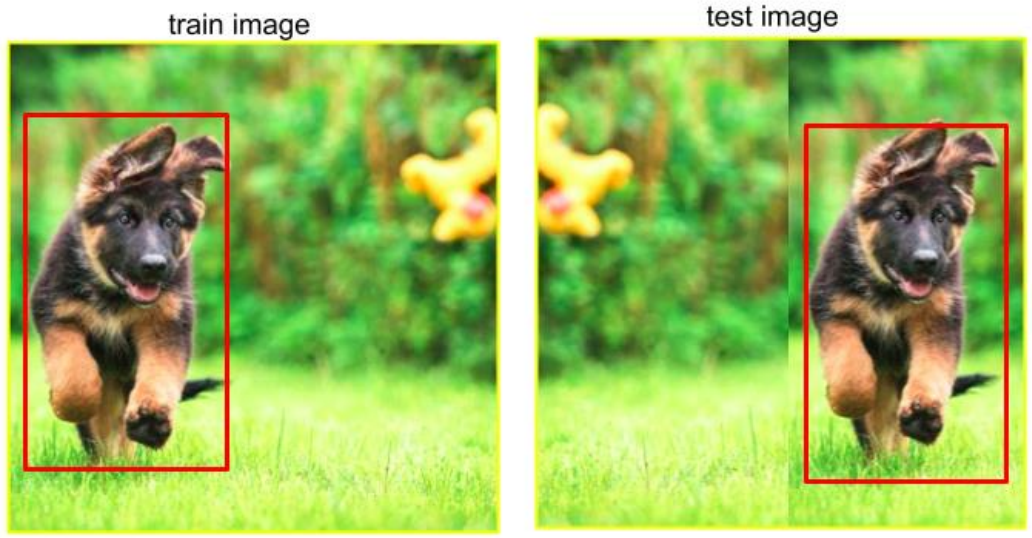


E.g. a vertical edge filter is useful in many parts of this image

CNNs for images

Translation invariance

In each layer, each output value depends only on a small number of inputs, which makes it translation invariance. In other words, if a pattern/object is translated, it is still identifiable by CNN.



CNN would detect the ‘dog’ object regardless of its location. This is called translation invariance.

The non-linearity will remain the same

Fully connected architecture

$$a_{l,k} = \sum_j w_{l,kj} h_{l-1,j} + b_{l,k}$$

$$h_{l,k} = \sigma \left(\sum_j w_{l,kj} h_{l-1,j} + b_{l,k} \right)$$

Convolutional architecture

$$a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$$

$$h_{l,k} = \sigma \left(\sum_j w_{l,kj} * h_{l-1,j} + b_{l,k} \right)$$

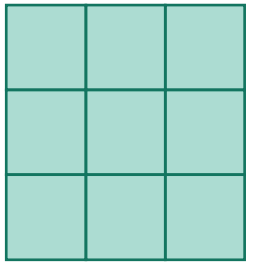
Convolution

$$w * x$$

Image: x

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

Convolution kernel: w



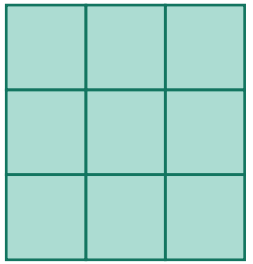
Convolution

$$w * x$$

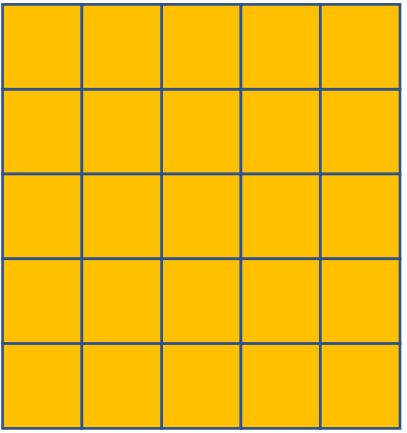
Image: x

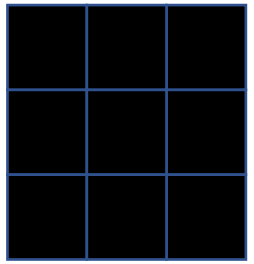
544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

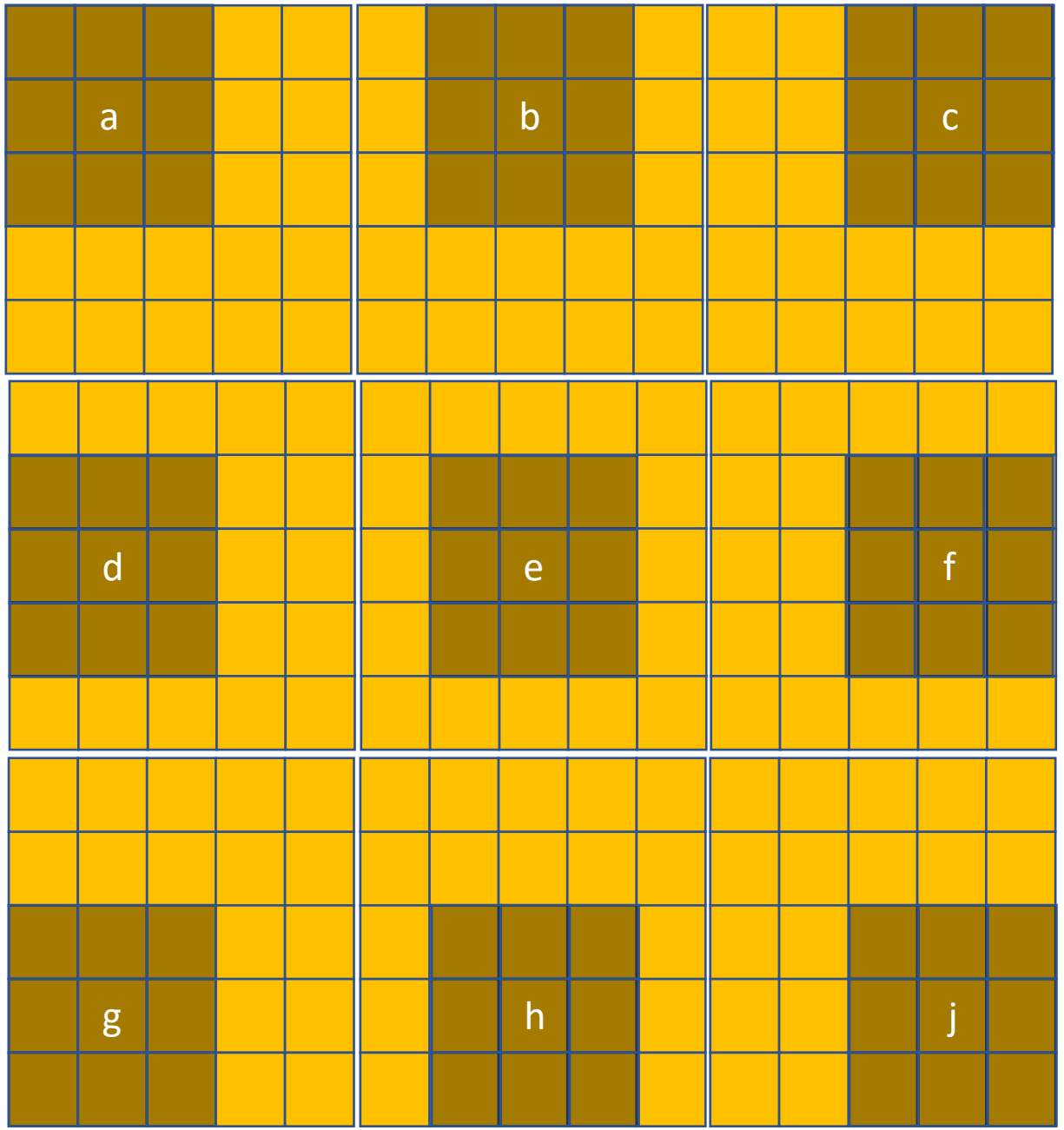
Convolution kernel: w



$$a_{ij} = \sum_p \sum_q x_{(i-p)(j-q)} w_{(p)(q)}$$





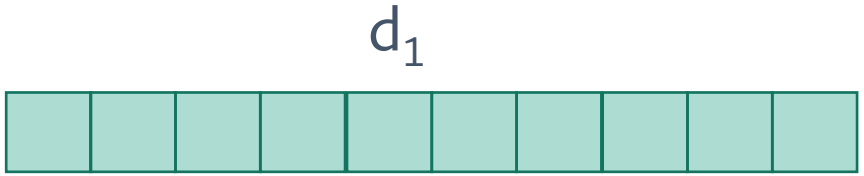


a	b	c
d	e	f
g	h	j

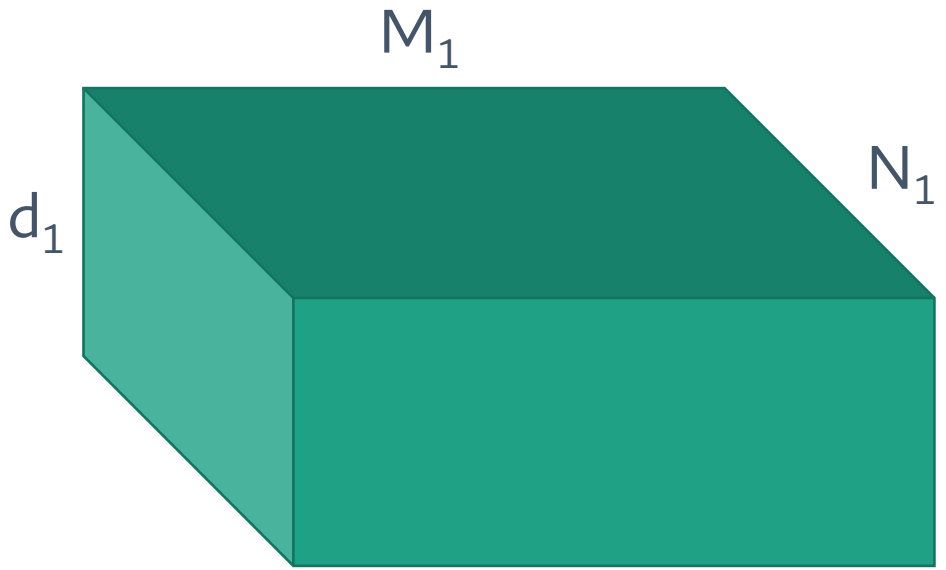
Larger hidden layers

Image: $x \in \mathbb{R}^{M_0 \times N_0}$

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557



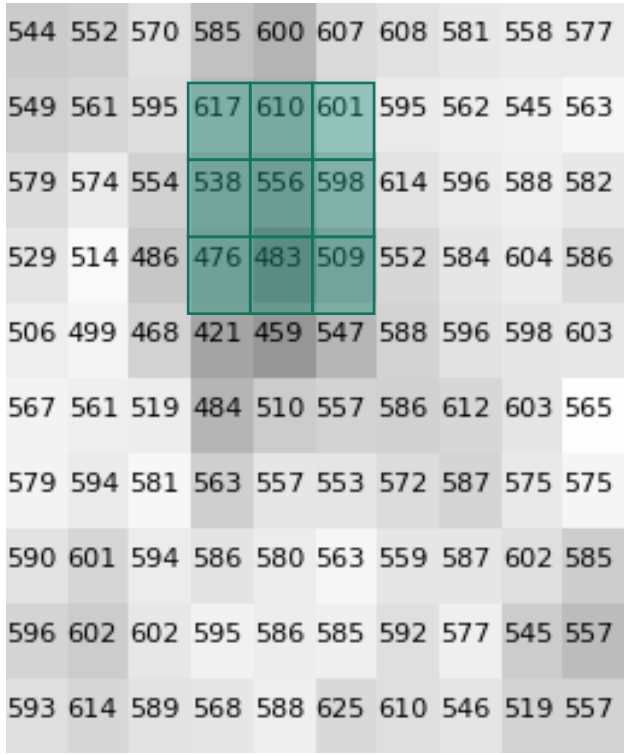
Fully connected: $h_1 \in \mathbb{R}^{d_1}$



Convolutional: $h_1 \in \mathbb{R}^{d_1 \times (N_1 \times M_1)}$

Channel size

Channel size is linked with kernel size and the type of convolution

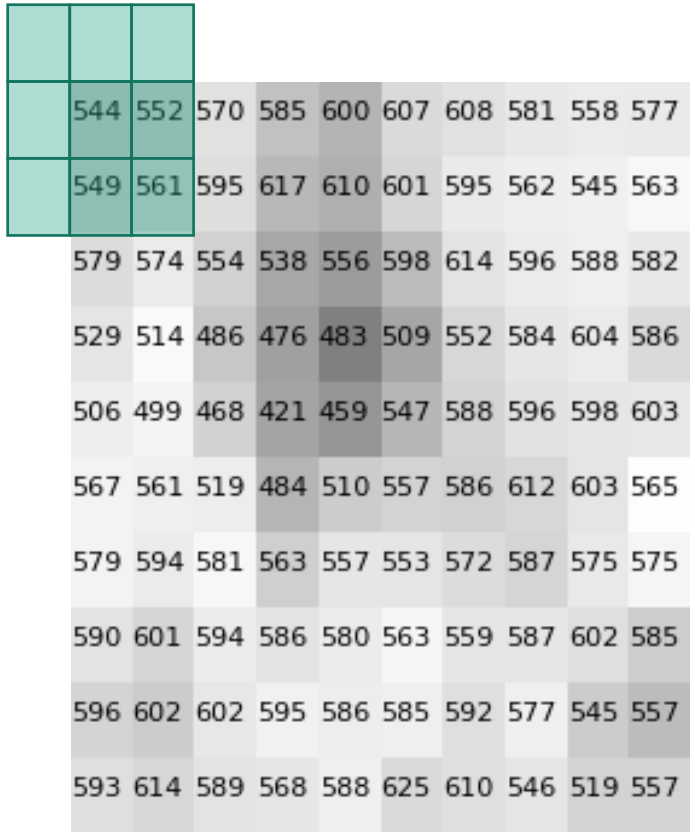


$$a_{ij} = \sum_p \sum_q x_{(i-p)(j-q)} w_{(p)(q)}$$

- When the kernel is placed in the image – no problem

Channel size

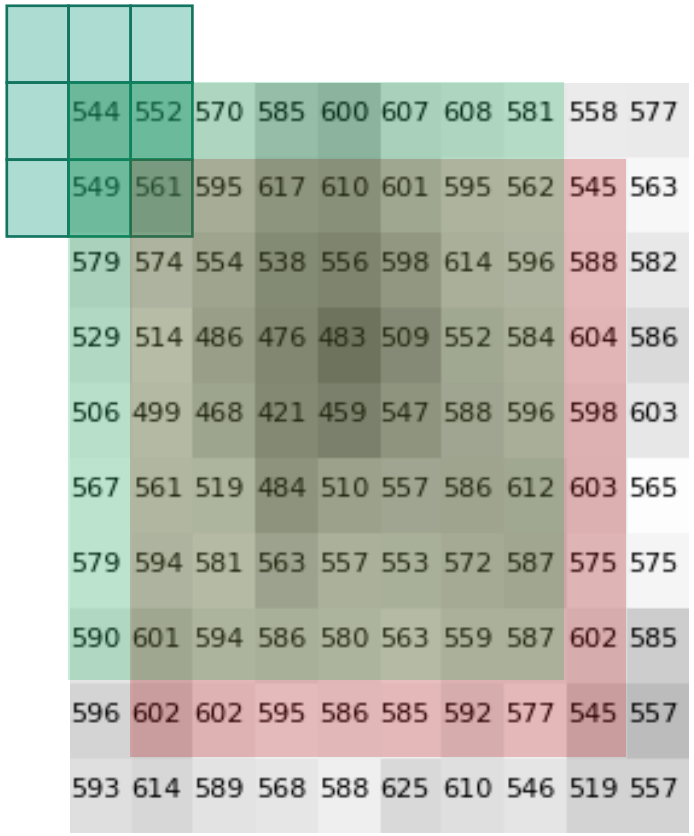
Channel size is linked with kernel size and the type of convolution



$$a_{ij} = \sum_p \sum_q x_{(i-p)(j-q)} w_{(p)(q)}$$

- When the kernel is placed in the image – no problem
- When it is placed on the boundary – it is not well defined
- Out-of-boundary values are not defined
- Two options:
  1. Valid convolution: only evaluate convolution when all the elements are defined
  2. Padding (Same): pad the boundaries so that result of the convolution will have the same size

Valid convolution



- If the kernel is centered, i.e. $w_{(0)(0)}$ is the center of the kernel, then convolution can only be evaluated within the red area
- You loose a pixel at each end of the picture
- If the kernel center is the top left corner, then the green area is the valid area
- You loose two pixels at the bottom and right of the image

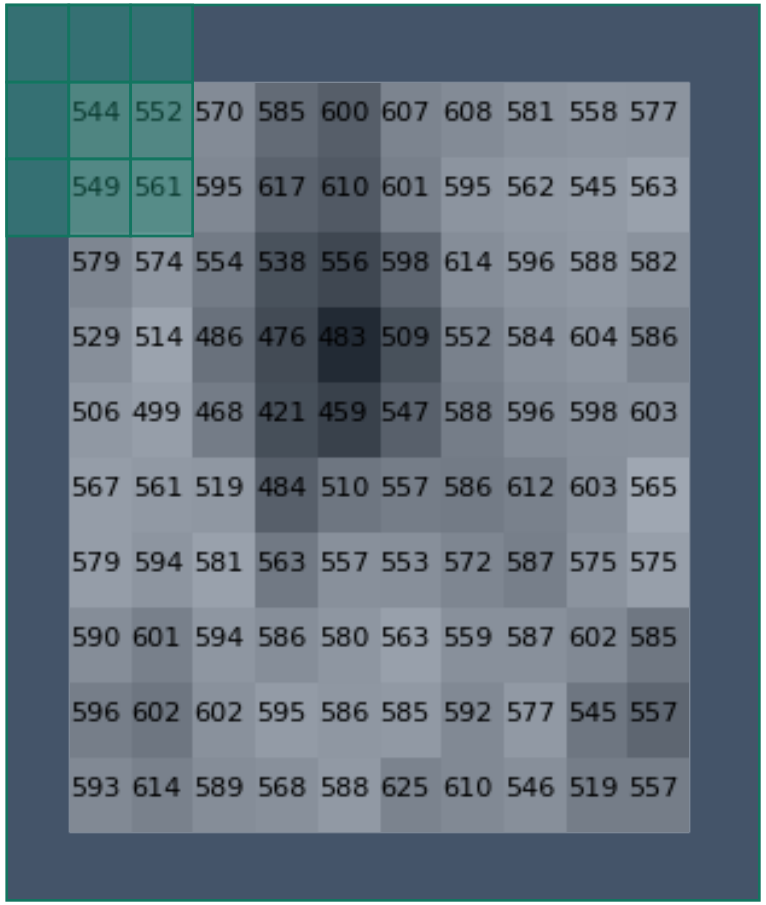
$$h_{l-1} \in \mathbb{R}^{d_{l-1} \times (M_{l-1} \times N_{l-1})}$$

$$w_{l,..} \in \mathbb{R}^{k_1 \times k_2}$$

$$h_l \in \mathbb{R}^{d_l \times (M_l \times N_l)}$$

$$M_l = M_{l-1} - k_1 + 1 \qquad N_l = N_{l-1} - k_2 + 1$$

Same padding



Alternatively you can pad the image on the boundaries so that channels will have the same size across layers

$$h_{l-1} \in \mathbb{R}^{d_{l-1} \times (M_{l-1} \times N_{l-1})}$$

$$h_{l-1} \in \mathbb{R}^{d_{l-1} \times ((M_{l-1}+k_1-1) \times (N_{l-1}+k_2-2))}$$

$$h_l \in \mathbb{R}^{d_l \times (M_l \times N_l)}$$

$$M_l = M_{l-1}\, N_l = N_{l-1}$$

Where you pad depends on where the center of the kernel is.

Commonly you would use centered kernels – padding around the image as shown on the left

The value you pad is a parameter, 0 is used often but you can use symmetric padding for certain applications

Convolution operation and filter (kernel)

vertical edge filter

1	0	-1
1	0	-1
1	0	-1

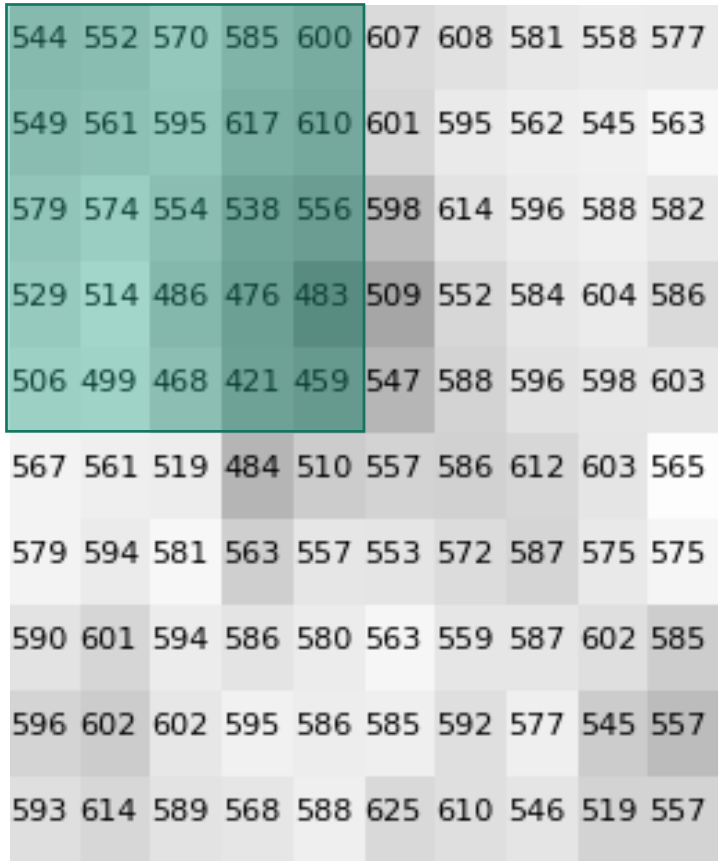
Vertical edges

Horizontal edges

1	1	1
0	0	0
-1	-1	-1

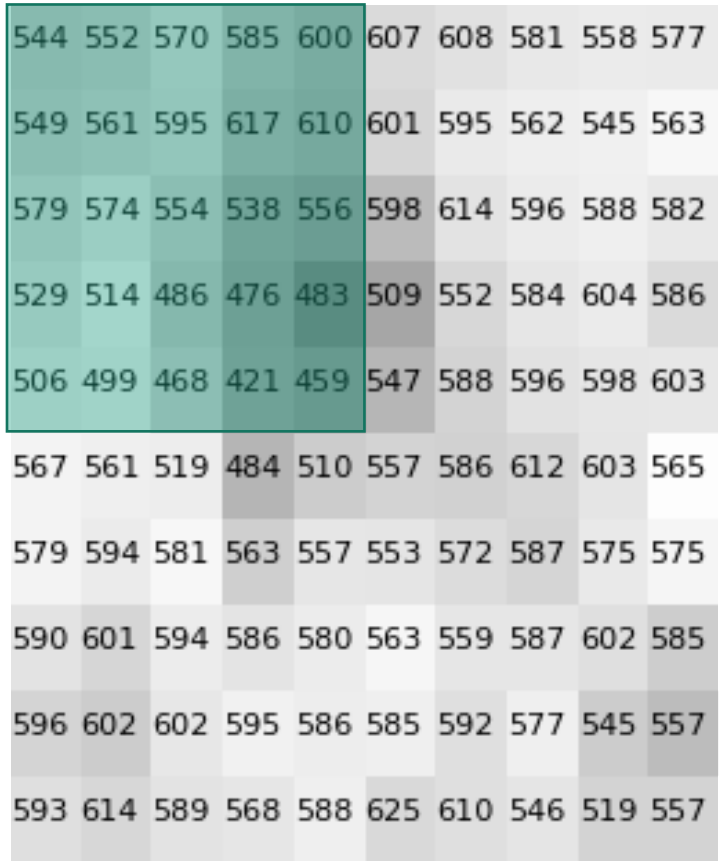
horizontal edge filter

Pooling



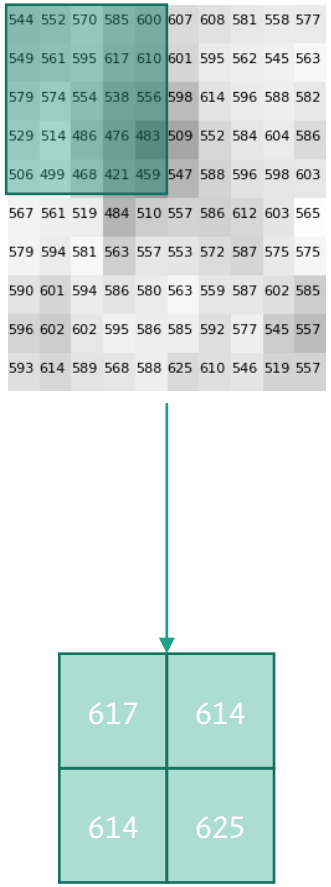
- Pool information in a neighborhood
- Represents the region with one number >> summarizes information
- Applied to each channel separately
- **Max-pooling** – maximum of the activation values
- **Min-pooling** – minimum of the activation values
- Both are non-linear operations, like median filtering
- **Averaging pooling** – linear operator
- Max-pooling is the most commonly used version

Max pooling



- Represents the entire region with the neuron that achieves the highest activation
- Leads to partial local translation invariance
- 617 in the highlighted area can be in any of the neurons, the pooled value will not change
- Does not lead to complete translation invariance
- Often applied with strides equal to the size of the kernel

Dimensionality reduction



- Leads to a substantial dimensionality reduction
- Even when the pooling kernel is of size 2x2, it can halve the image!
- As the size of the pooling kernel increase, the reduction increases as well
- Non-linear dimensionality reduction
- Only the most prominent activation is transmitted to the next layer
- More advanced pooling mechanisms exist CapsuleNets [Sabour, Frosst and Hinton 2017]

Essential blocks lead to powerful algorithms

Convolutional layers and pooling are the essential blocks

They have been used to create complicated networks

First one

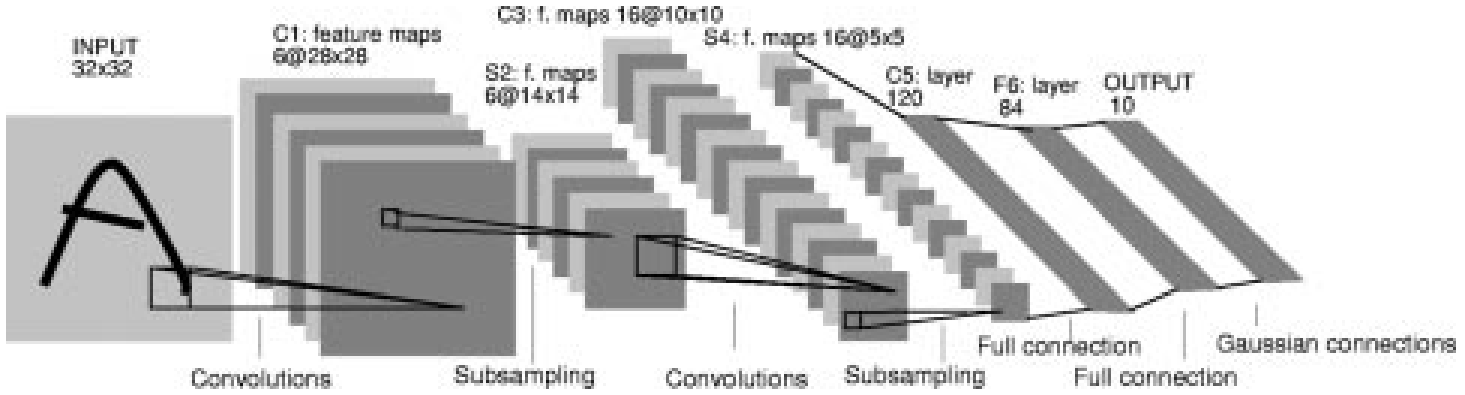


Fig. 2. Architecture of LeNet-5, a convolutional NN, here used for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical.

[Lecun, Bottou, Bengio and Haffner; Gradient-based learning applied to document recognition; 1998]

Then silence for a long time

Why silence

Models had too many parameters

They overfit for small datasets

We did not have very large datasets

Even for large sets, we did not have enough computation power to train the models until…

General purpose Graphical Processing Units (GPUs)
Allowed parallel processing

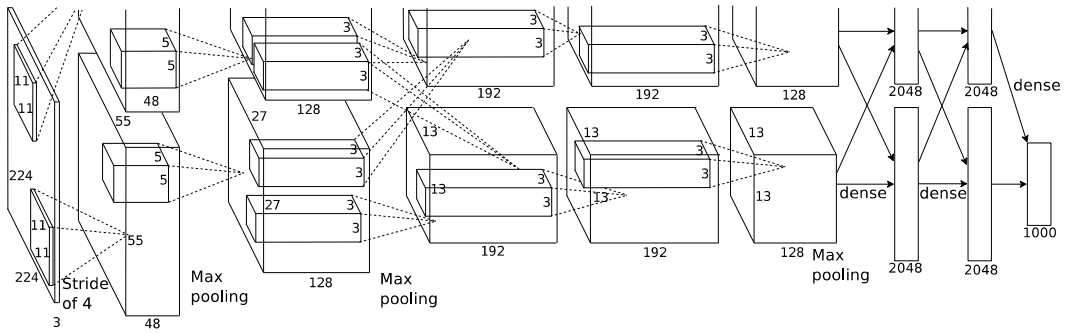
Then in 2012

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

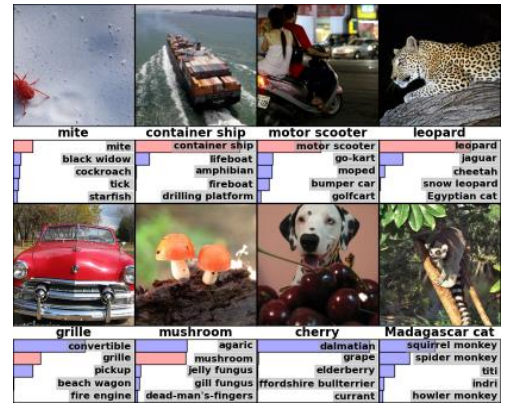
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

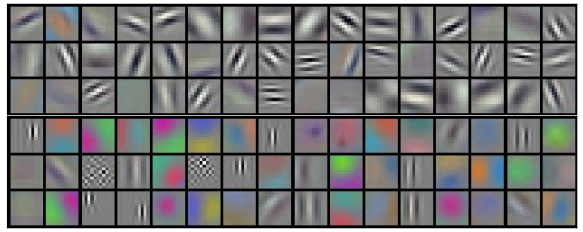


Network

Krizhevsky et al. almost halved the error rate in the ImageNet challenge

A simple CNN





First layer filters 11x11

A word about the ImageNet challenge

ImageNet: A Large-Scale Hierarchical Image Database

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei
Dept. of Computer Science, Princeton University, USA
{jiadeng, wdong, rsocher, jial, li, feifeili}@cs.princeton.edu

[CVPR 2009]

International Journal of Computer Vision
December 2015, Volume 115, Issue 3, pp 211–252 | Cite as

ImageNet Large Scale Visual Recognition Challenge

Authors Authors and affiliations

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei

Large scale object recognition challenge started in 2010

1.2 million training images

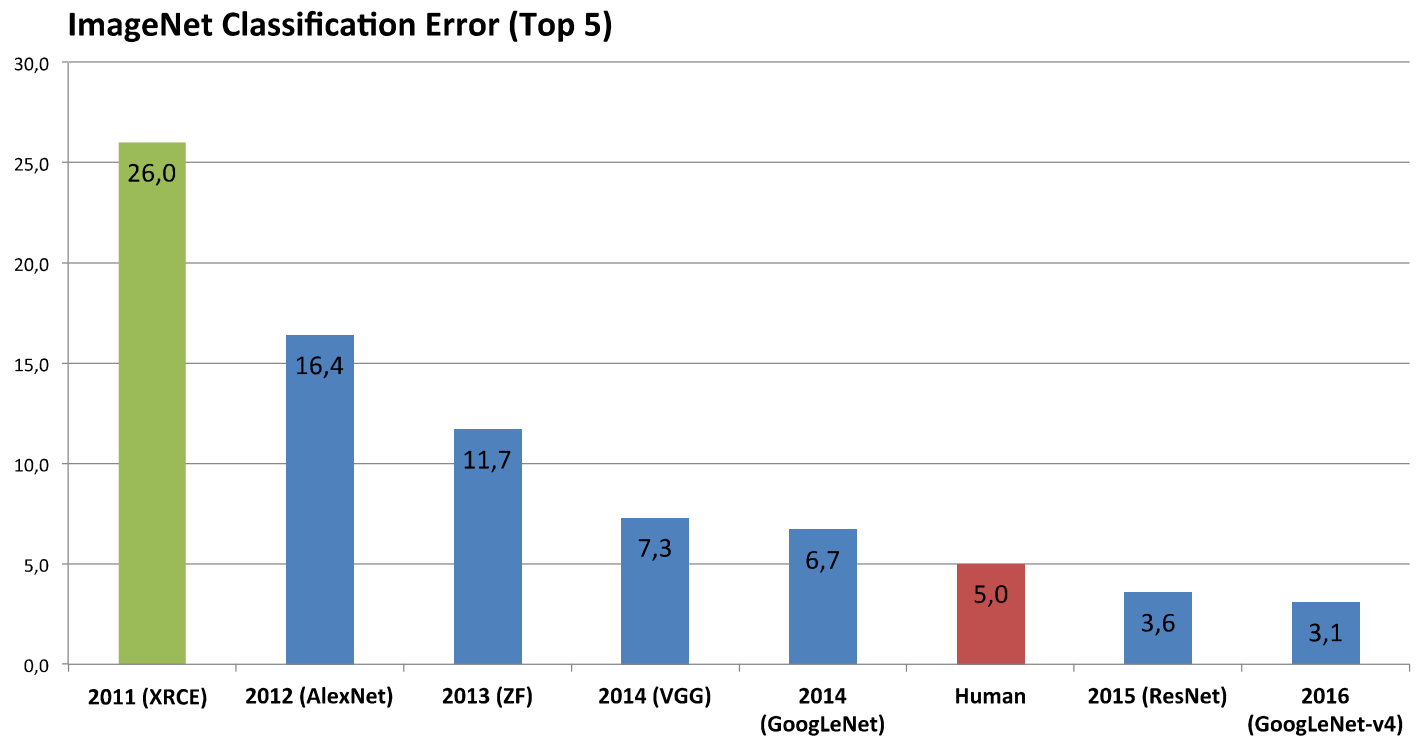
1000 object categories

200k validation and test images

2017 – 3 challenges:

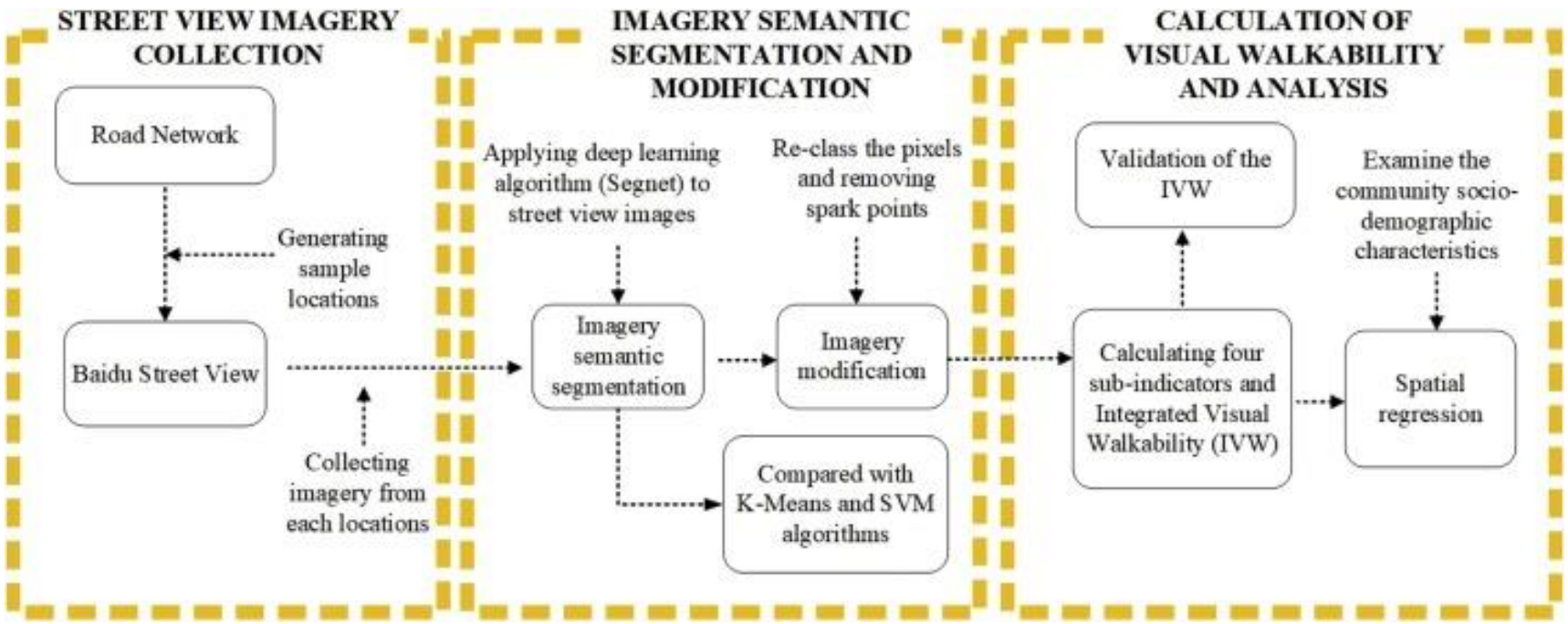
- Object localization
- Object detection
- Object detection from video

Historical evolution of the ImageNet Challenge



CNNs for Geospatial Research

Example 1: Visual walkability



Zhou et al., 2019

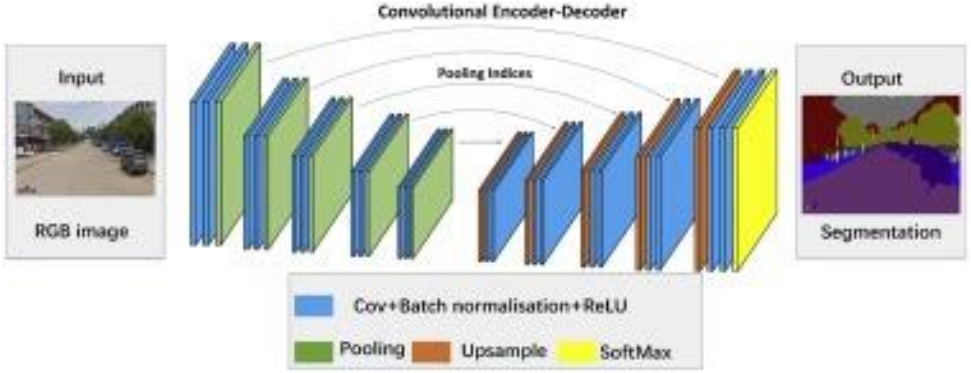
Visual walkability

- These indicators are derived from street images and used to build the IVW index.

Indicators	Definition	Formula	Explanation
Psychological Greenery	Extent to which the visibility of street vegetation can influence pedestrian psychological feelings	$G_i = \frac{\left(\sum_1^6 T_n\right)}{6*Sum}$	T_n is the number of tree pixels; Sum is the total pixel number;
Visual Crowdedness	Extent to which the visibility of obstacles can influence pedestrian experiences	$C_i = \frac{\left(\sum_1^6 C_n\right)}{6*Sum}$	C_n is the number of obstacles pixels; Sum is the total pixel number;
Outdoor Enclosure	How the room-like outdoor space is (the ratio of vertical objects to horizontal features)	$S_i = \frac{\sum_1^6 B_n + \sum_1^6 T_n}{\sum_1^6 P_n + \sum_1^6 R_n + \sum_1^6 F_n}$	B_n is the number of building pixels; T_n is the number of tree pixels; P_n refers to the number of pavement pixels; R_n refers to the number of road pixels; F_n refers to the number of fence pixels
Visual Pavement	Psychological impacts of the proportion of road and sideway on pedestrian experience	$D_i = \frac{\sum_1^6 P_n + \sum_1^6 F_n}{\sum_1^6 R_n}$	R_n refers to the number of road pixels; P_n refers to the number of pavement pixels; F_n refers to the number of fence pixels;
Integrated Visual Walkability (IVW)	Integrated Visual Walkability index	IVW=(G-level + C-level + S-level + D-level)*5	

Zhou et al., 2019

Visual walkability



(a). Structure of SegNet

(a). Segmentation sample of Segnet

(b). Segmentation sample after reclassed

(c). Street view image and corresponding labelled training image

Zhou et al., 2019

Example 2: measuring urban inequality

Measurements of inequalities: based on census costly and infrequent

High spatial and temporal resolution – challenging even for data-rich cities and countries

Opportunities to leverage emerging large-scale datasets: street-level images, satellite data, etc.

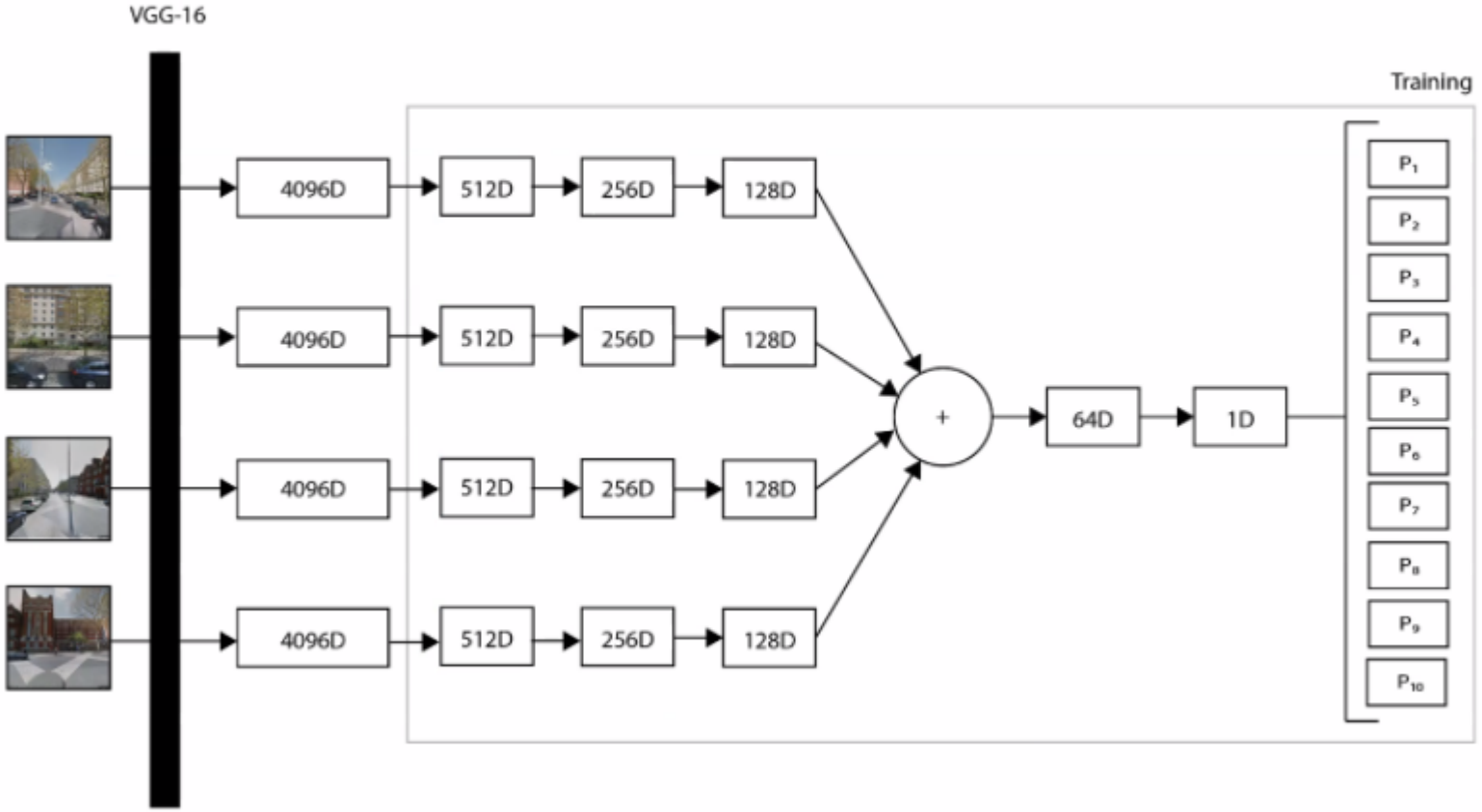
Example 2: measuring urban inequality

Case study: using street images to predict the decile of index of multiple deprivation (IMD)

~525,000 images from ~180,000 postcodes in London

Suel et al., 2019

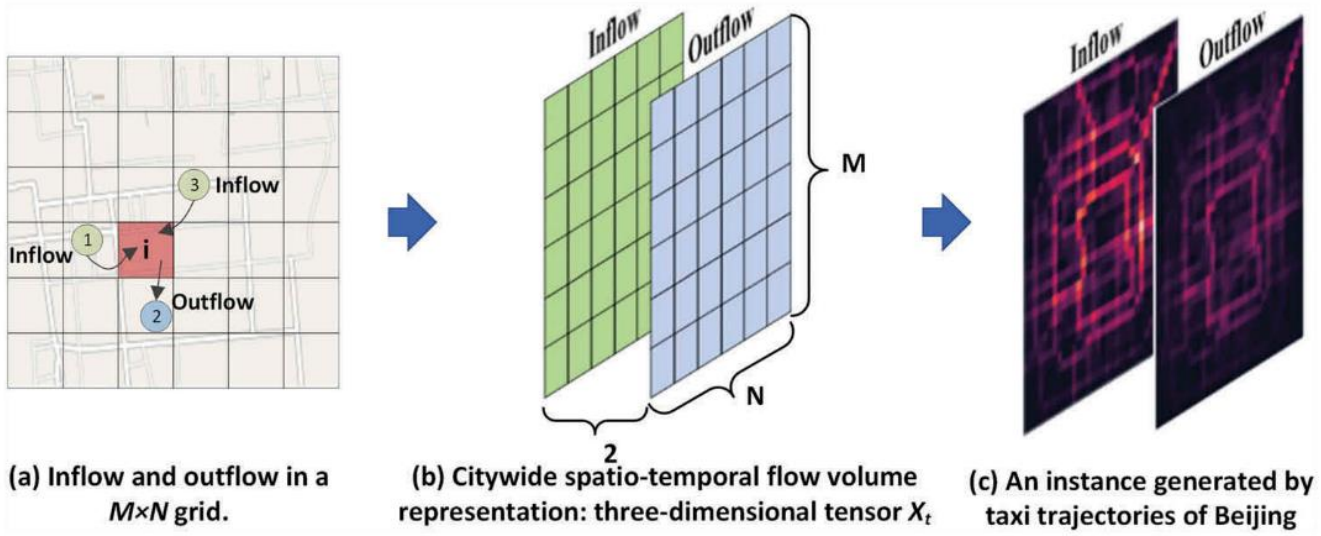
Example 2: measuring urban inequality



Suel et al., 2019

Example 3: spatio-temporal prediction of traffic flows in urban cells

The idea is to transform traffic flow into image-shape data and then apply CNN to predict flows.



(a) Inflow and outflow in a $M \times N$ grid.

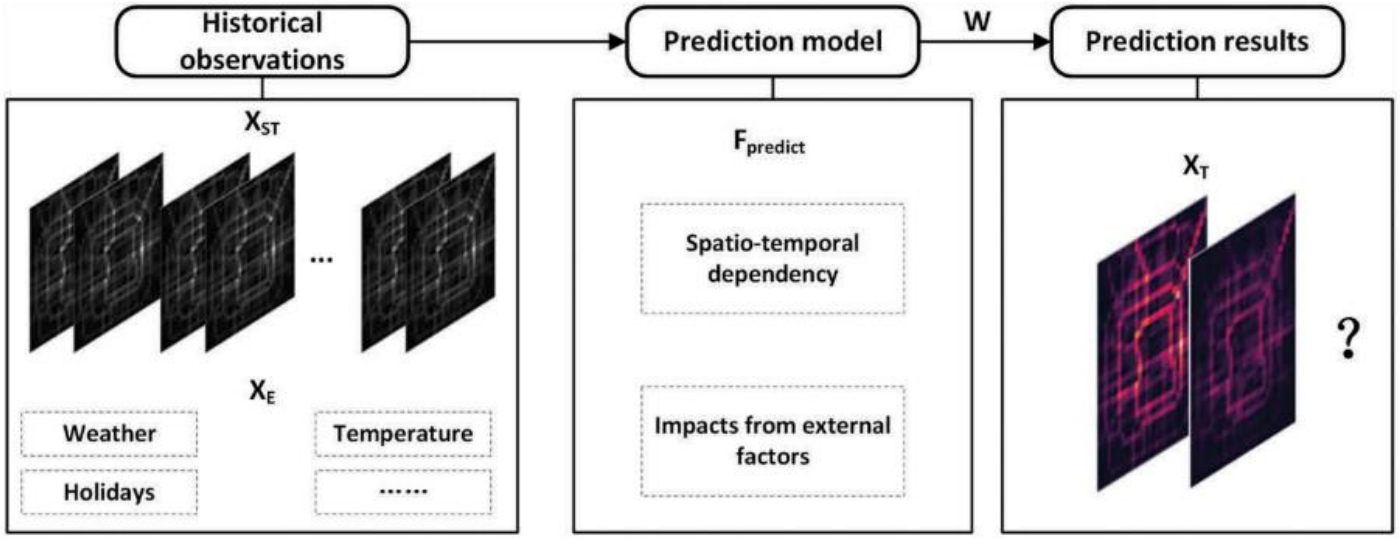
(b) Citywide spatio-temporal flow volume representation: three-dimensional tensor X_t

(c) An instance generated by taxi trajectories of Beijing

Ren et al., 2020

Example 3: spatio-temporal prediction of traffic flows in urban cells

The idea is to transform traffic flow into image-shape data and then apply CNN to predict flows.



Ren et al., 2020

Summary – deep learning for images

DL provides powerful CNN-based tools to extract patterns and knowledge from images for varied purposes.

Most models are open-source and easy to reuse. Most models have similar structure (incl. Conv, Pooling, FC layers).

For most applications, you don’t need to design a CNN from scratch. It is advisable to test classical models and adjust the models if needed.

Potential Problems with CNN

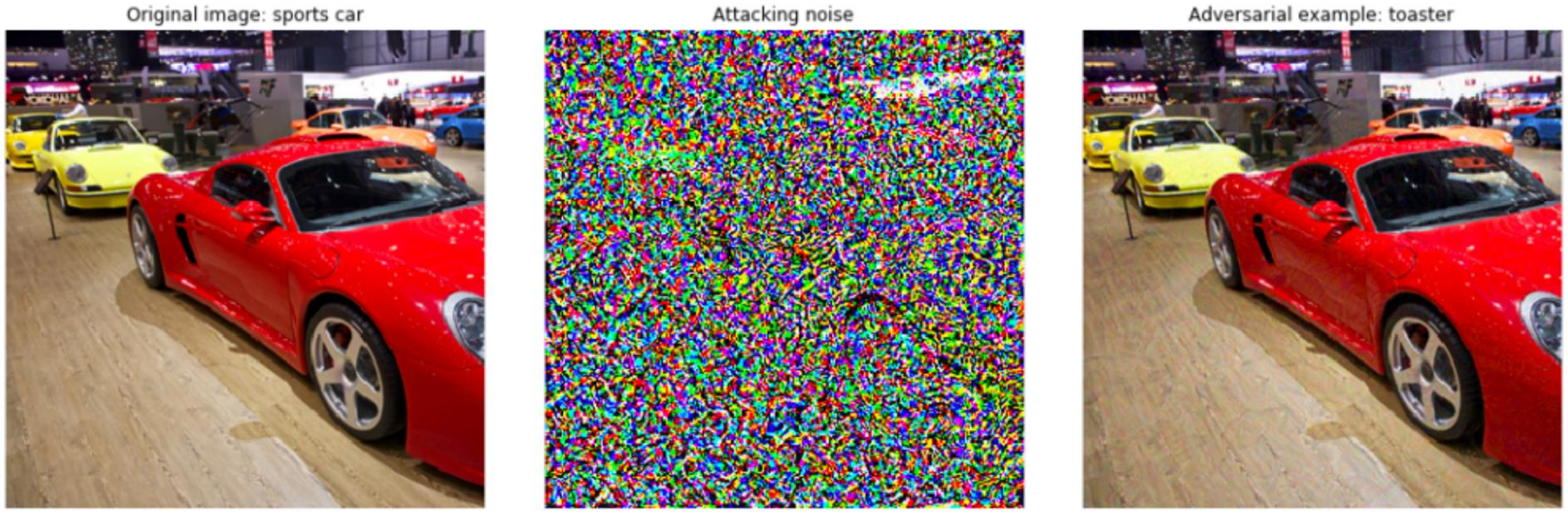
Garbage in, garbage out

If labels are not reliable, the model trained would be problematic.

“A flock of birds flying in the air”

“A group of flowers in a field”

Adversarial Attacks



- You can fool a neural network classifier by adding noise to data.
- The noise might not be even visible to a human eye.

Adversarial Attacks

- Stop sign is no longer recognised by a neural network after a few stickers are placed on it.

Adversarial Attacks

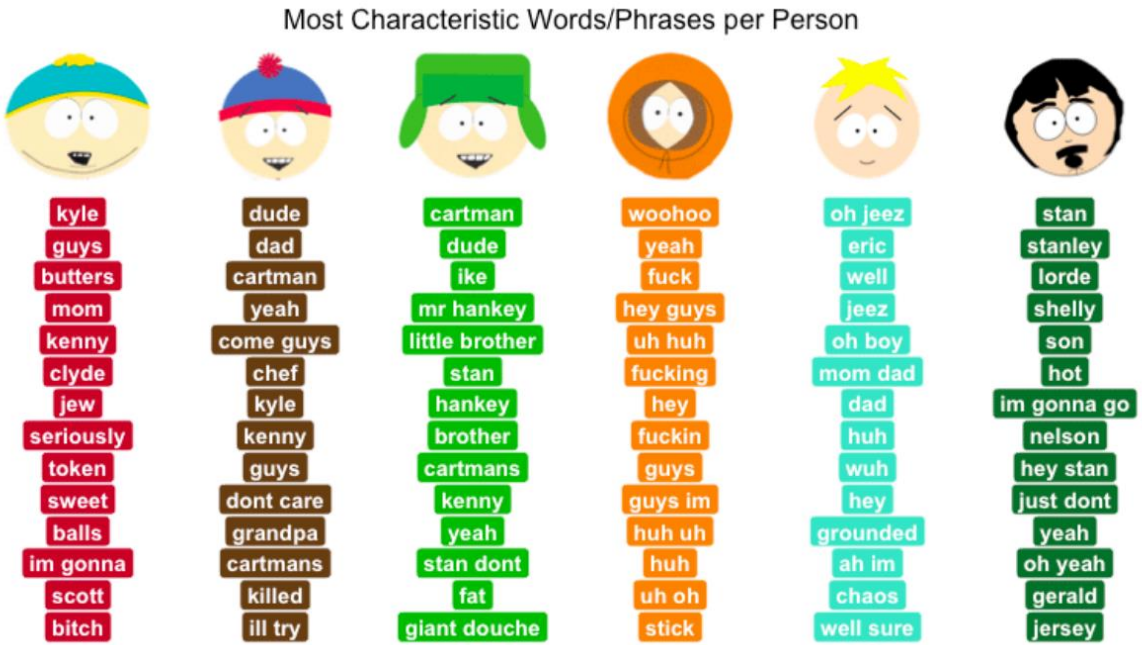
- Microsoft's chatbot is turned into an anti-Semitic extremist after chatting with a group of people feeding it with racist information.

Deep learning for text data

Keyword Detection

- Keyword extraction identifies the most important tokens or n-grams within a piece of text

- Input: a piece of text
- Output: list of most important tokens (words), optionally grouped by topics



Sentiment Analysis

- Sentiment Analysis assigns a sentiment score to each word in a piece of text

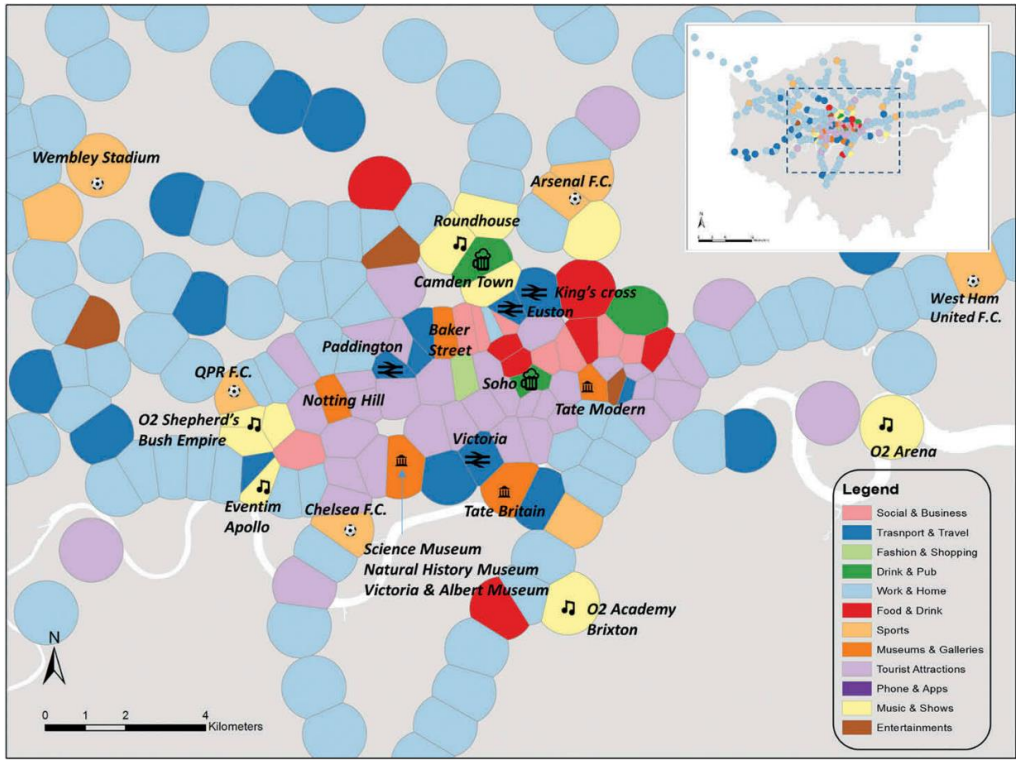
I love data science, and our teachers are awesome	+4 (Strongly positive)
Beer is disgusting, why do people even like it?	-1 (Weakly negative)
It's so great that my train is late every single day	+1 (Weakly positive)

- Input: a piece of text
- Output: a piece of text with a sentiment score assigned to each word

Topic Extraction

- Topic modelling extracts topics from a collection of documents.
- Input: a collection of documents
- Output: a list of topics with words that belong to them

Example:
Dominant topics on Tweets around the stations



Lai, J., Cheng, T., and Lansley, G. (2017) Improved targeted outdoor advertising based on geotagged social media data. Annals of GIS, ,1–14. https://doi.org/10.1080/19475683.2017.1382571

Summary

- Unstructured data are common in the big data era, including images, text, etc.
- Deep learning provides powerful tools for utilising unstructured datasets. Most tools are open-source and ready to use.
- For applications, you don't need to fully understand the mathematics of these DL models.
- We cover the foundation and classical models of CNN here. More details can be found in the Coursera module "Convolutional Neural Networks".

https://www.coursera.org/learn/convolutional-neural-networks

Workshop

In this week's workshop you will learn how to train your own deep learning models in Python for a range of tasks:

- Image classification
- Face recognition
- Semantic image segmentation