Deep Learning for Computer Vision Part I: Introduction

Computer Vision

Introduction

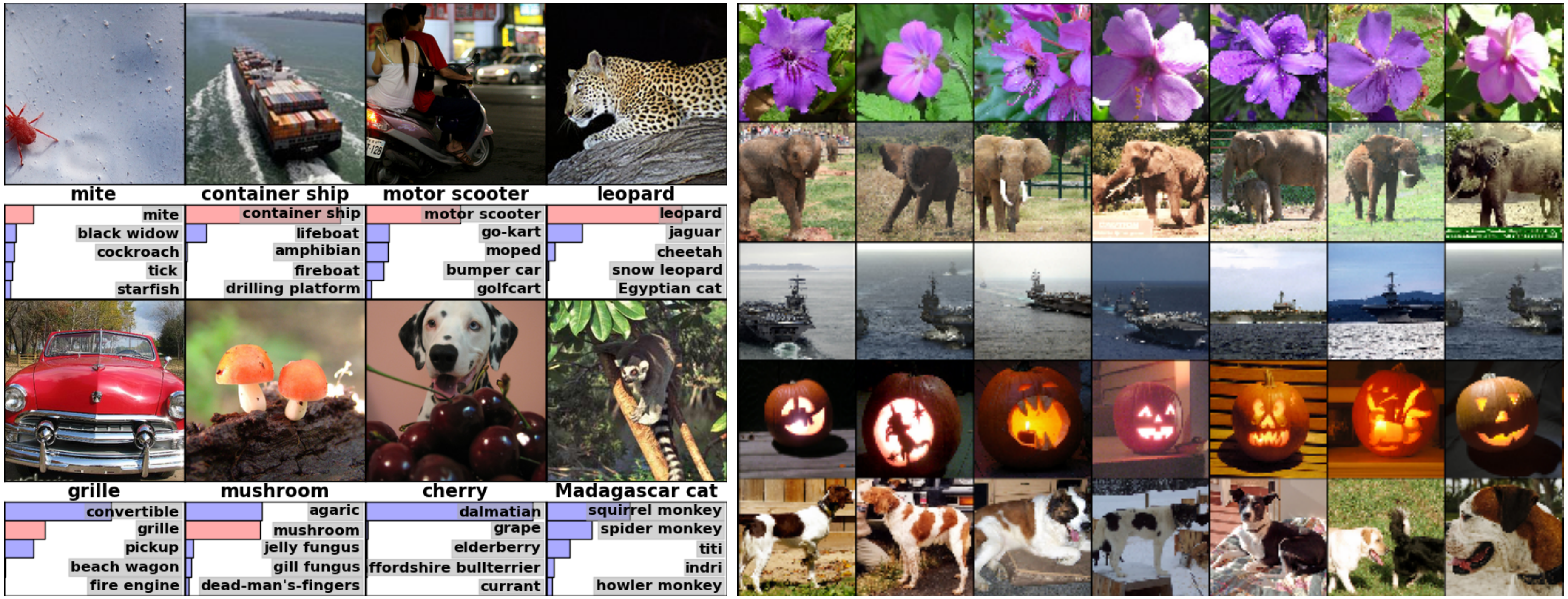
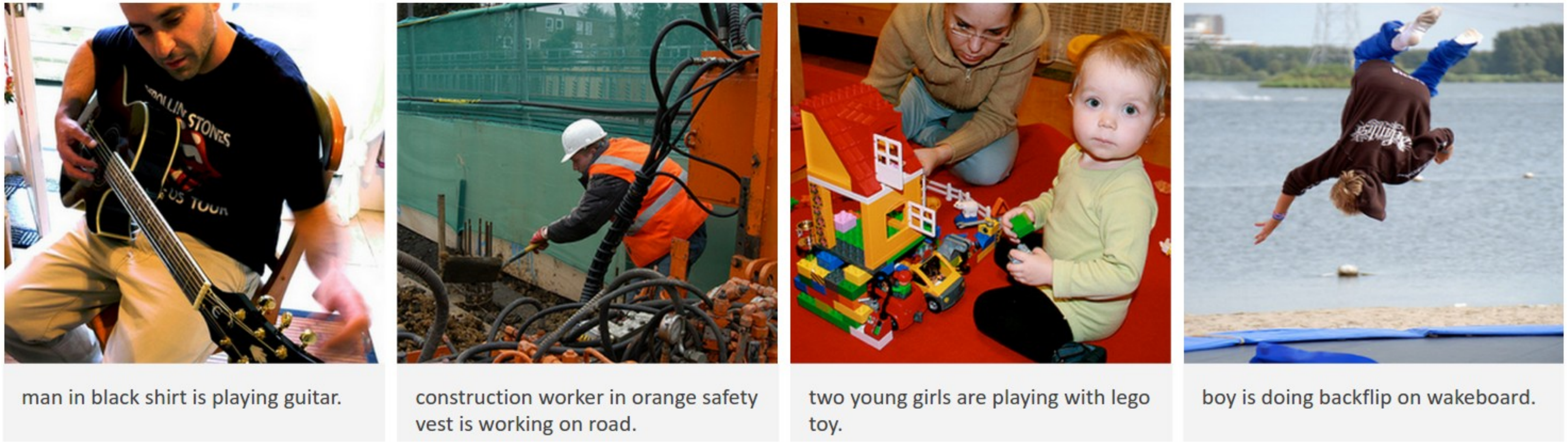


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

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

Computer Vision

Introduction



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.

Figure 6. Example sentences generated by the multimodal RNN for test images. We provide many more examples on our project page.

[Figure from Karpathy and Fei-Fei 2015 – Automatically generating image captions]

Computer Vision

Introduction

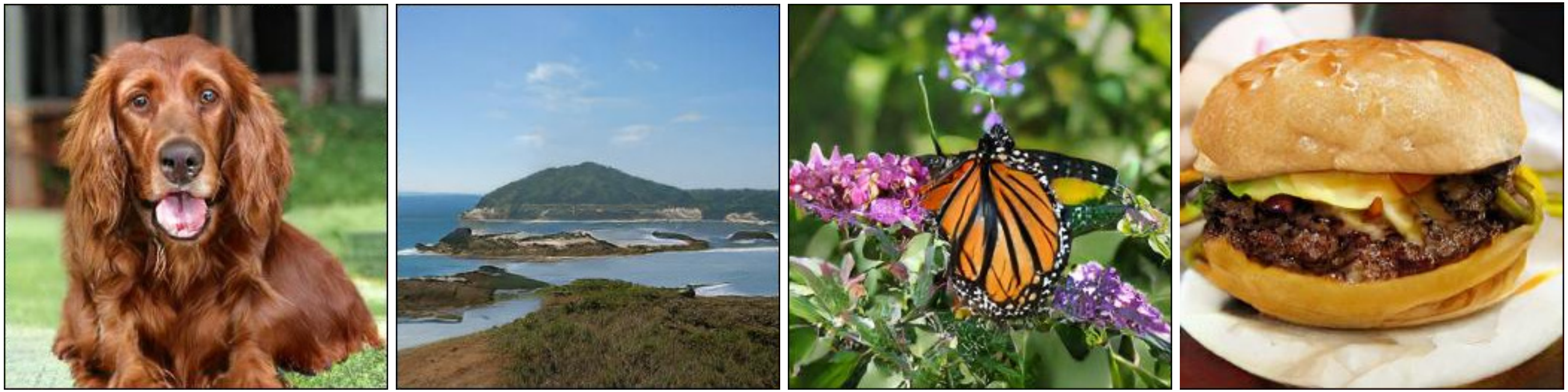
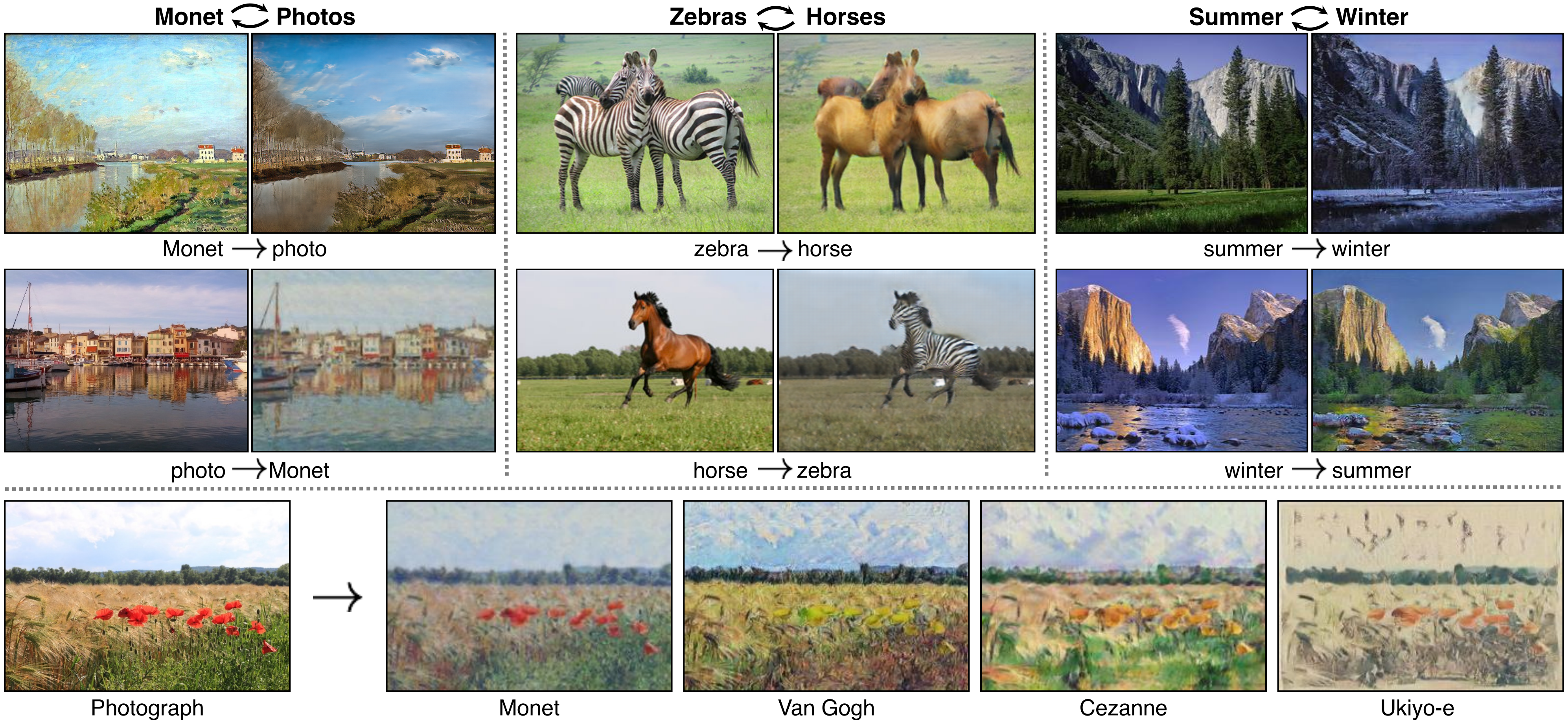


Figure 1: Class-conditional samples generated by our model.

[Figure from Brock, Donahue and Simonyan 2018 – Class conditional generation of images]

Computer Vision

Introduction



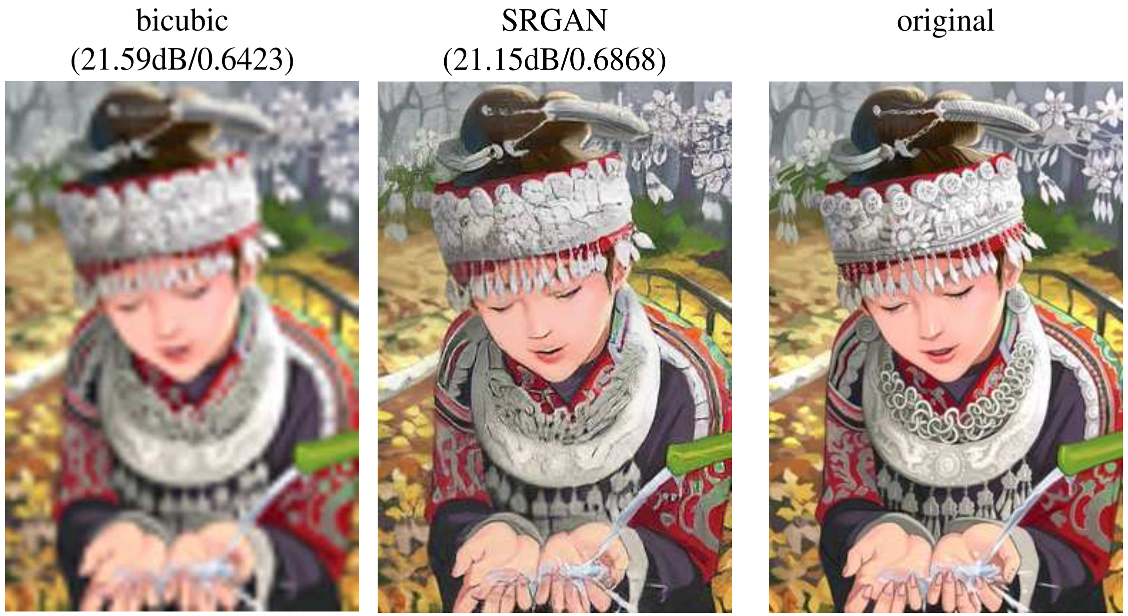
[Figure from Zhu, Park, Isola and Efros 2017 – CycleGAN]

Computer Vision

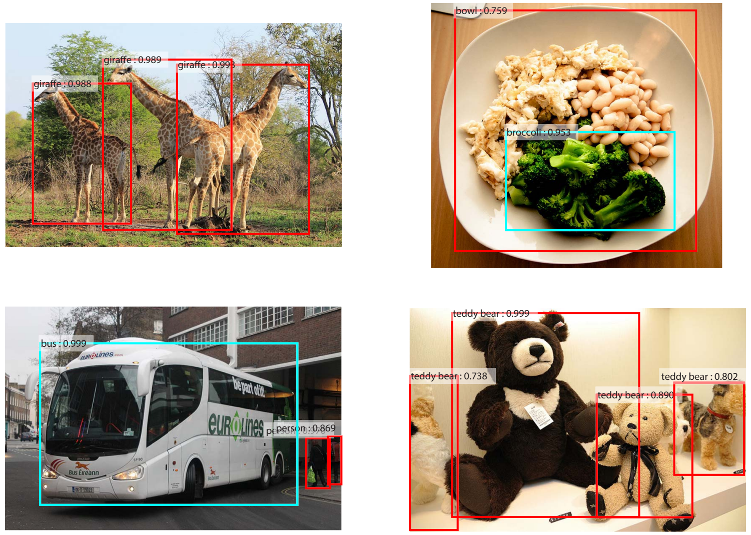
Introduction

Today

- Multilayered neural networks is an essential tool in computer vision and image analysis
- Enhancement, detection, segmentation, recognition,…



[Figure from Ledig et al. 2017]



[Figure from Ren et al. 2016 - Faster R-CNN]

Outline

1. Introduction to neural networks – this week
   Basics of neural networks

2. Convolutional neural networks
   Basic applications of deep learning to image analysis and computer vision

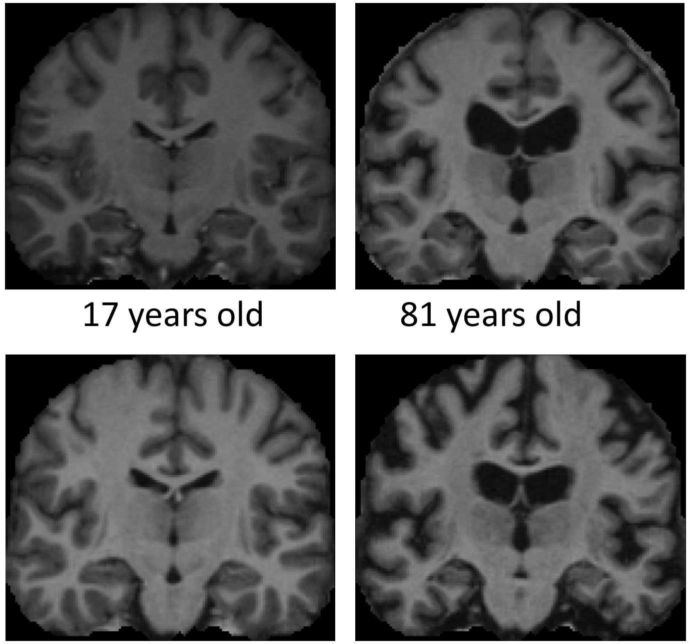
3. Advanced topics and applications
   Unsupervised learning and recurrent neural networks

Computer Vision

Introduction
Learning basics

Machine learning for visual perception

- Discovering and leveraging **patterns**



17 years old

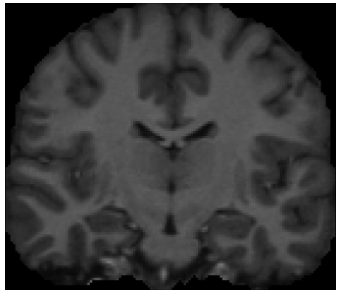
81 years old

30 years old

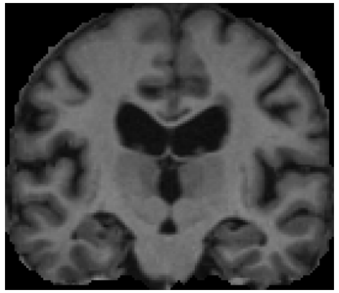
74 years old

Machine learning for visual perception

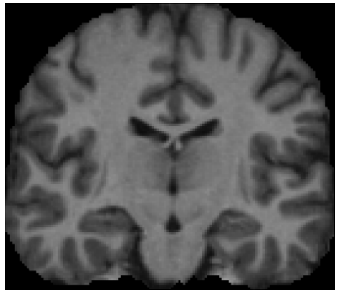
- Discovering and leveraging **patterns**
- Pattern recognition by seeing lots of **examples**
- Predictions based on visible patterns
- Algorithms that can
  - determine **useful** patterns, e.g. shape, texture, size and constellation of objects, …
  - **useful** : helps you accomplish the task
  - ignore unrelated variation, e.g. pose, illumination, contrast, …



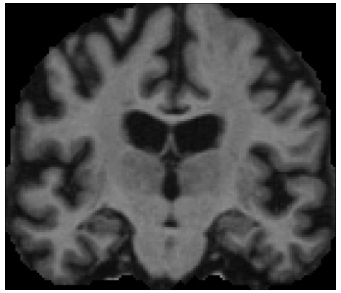
17 years old



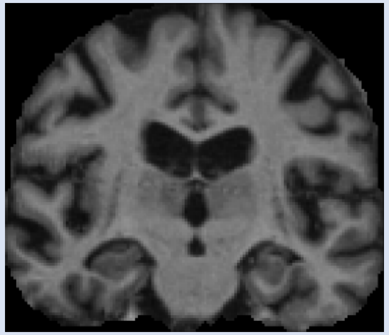
81 years old



30 years old



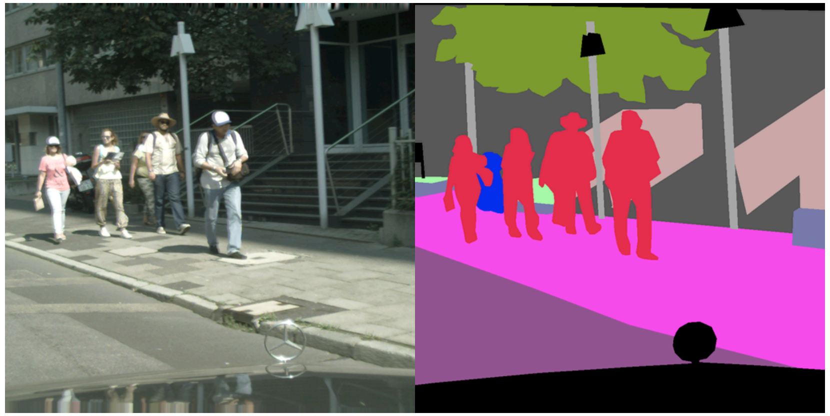
74 years old



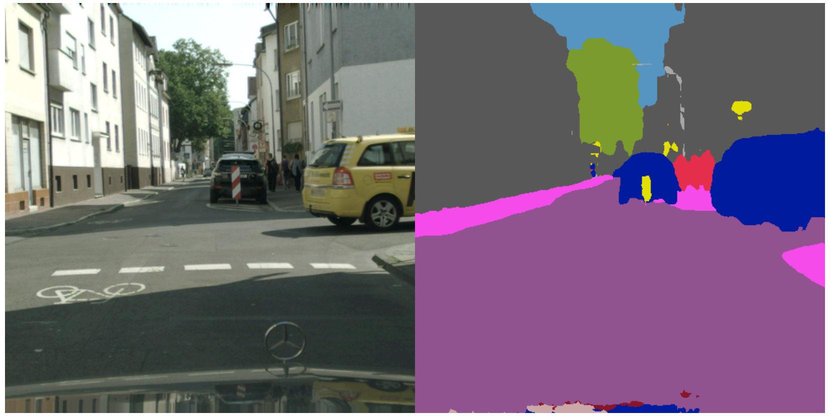
How old is she?

Two main types of learning: Supervised

- There is a **task**, e.g. segmentation, detection, recognition, …
- Examples have both **features** (images) and **labels** related to the task
- At prediction you only have the images
- Example task: segmentation
  - Appearance variation across classes
  - Ignore variance within instances of the same class
- Examples are extremely important!



Examples to learn from



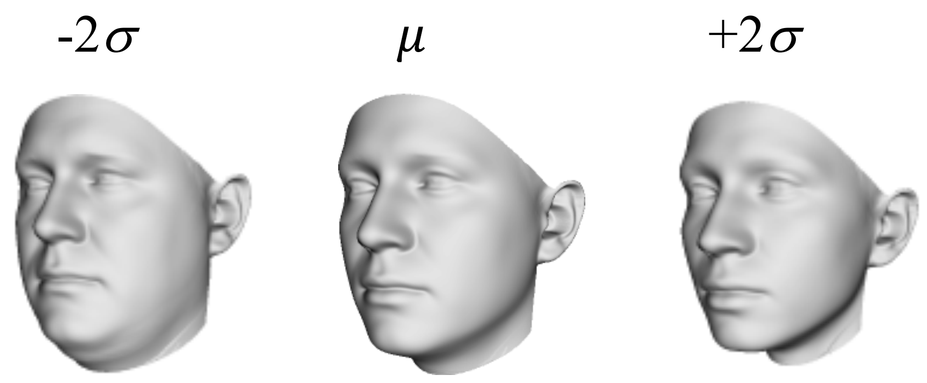
Test image

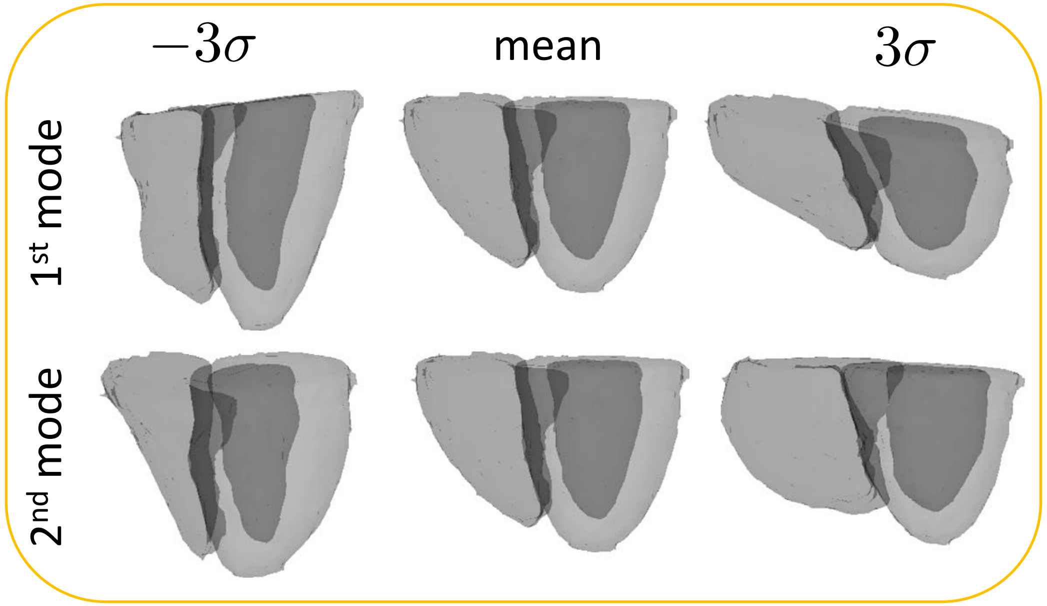
Prediction

Two main types of learning: Unsupervised

- Model / represent / describe the variability within the data
- There is no specific task
- Examples only have images (or features more generally)
- Prior information / regularization for a variety of inverse problems in image analysis, e.g. noise removal, super-resolution, deconvolution,

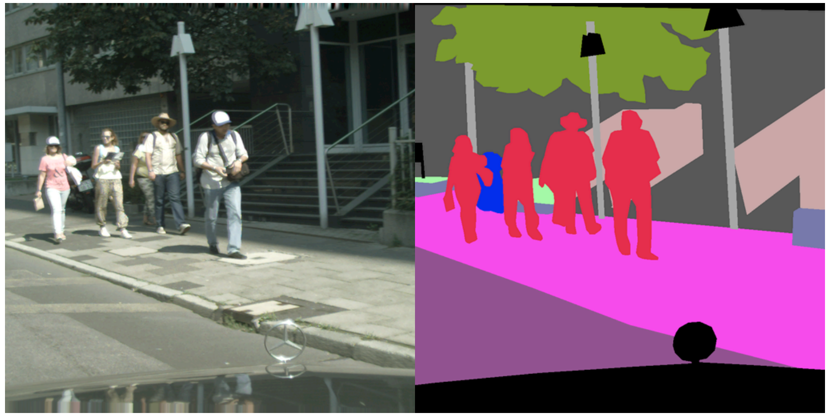
Linear model with PCA



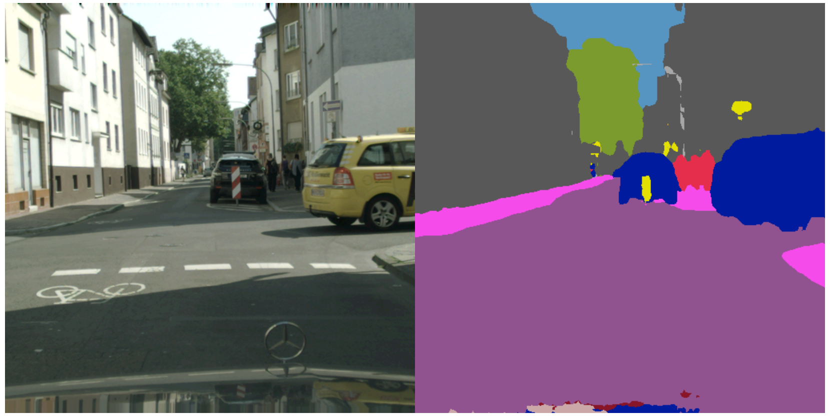


Two main types of learning: Supervised

- There is a **task**, e.g. segmentation, detection, recognition, …
- Examples have both **features** (images) and **labels** related to the task
- At prediction you only have the images
- Example task: segmentation
  - Appearance variation across classes
  - Ignore variance within instances of the same class
- Examples are extremely important!



Examples to learn from



Test image

Prediction

Section I: Mathematical basics

Basic notation

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

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

Features

- Observed information
- Numerical or categorical
- Hand-crafted explicit features such as HoG or SIFT
- Haar features, integral images
- Raw intensities, RGB values, gray levels
- Multispectral images can have even more raw intensity channels

Labels

- Unobserved information
- Numerical (regression) or categorical (classification)
- Semantic segmentation labels – pixel-wise
- Object categories – image-wise
- Unblurred intensities
- Denoised intensities
- Diagnosis, clinical outcomes

Computer Vision

Introduction
Learning basics
Math. basics

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

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

- Non-parametric mappings are also possible, think about nearest neighbor models

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

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

Basic concepts – cost function

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

- We model the definition of “best” using the cost function
- Task dependent and ideally defined for your final goal
- Regression: e.g. general p-norm for regression

$$\mathcal{L}\left(\mathbf{y}_n, f(\mathbf{x}_n;\theta)\right) = \|y_n - f(\mathbf{x}_n;\theta)\|_p$$

- Popular choices for p=1, 2 corresponding to L_1 and L_2 norms.

Basic concepts – cost function

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

- 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

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

- 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 - prediction

- The model and the best parameters are determined
- Prediction for a new sample also depends on your task
- For regression:

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

- Prediction will have errors
- Most commonly used: Mean Squared Error (MSE), Mean Absolute Error (MAE)

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

Basic concepts - prediction

- The model and the best parameters are determined
- Prediction for a new sample also depends on your task
- For classification:

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

- Prediction will have errors
- Most commonly used: Classification error (Cerr)

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

Notes on basic concepts

- Features:
  - Remember SIFT, Gradient Histograms, Integral images
  - Neural networks will focus on using directly the images
- Mapping:
  - Lots of research in this area
  - We will further focus on neural networks
- Cost functions:
  - There is on-going research
  - L_1 loss, Adversarial loss, Dice loss for segmentation
- Error computation:
  - Very important area
  - Estimation of generalization error is a difficult task
  - Statistical methods: cross-validation, bootstrapping, …

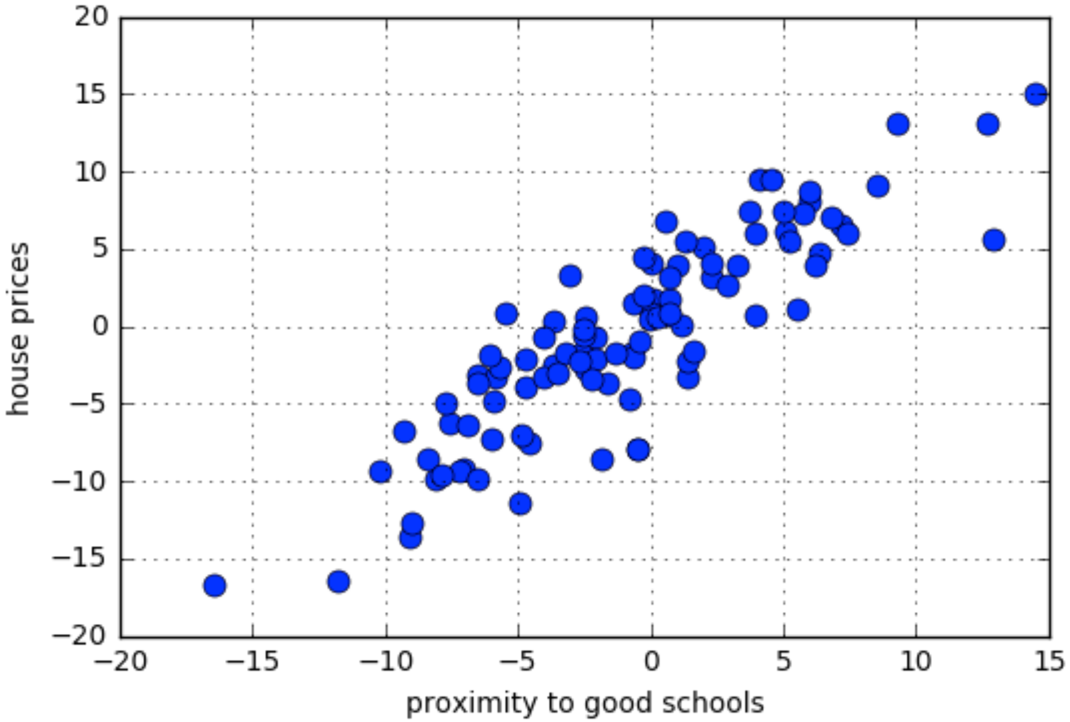
Linear and Logistic Regression Models

- 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 model is for continuous labels

Linear regression model



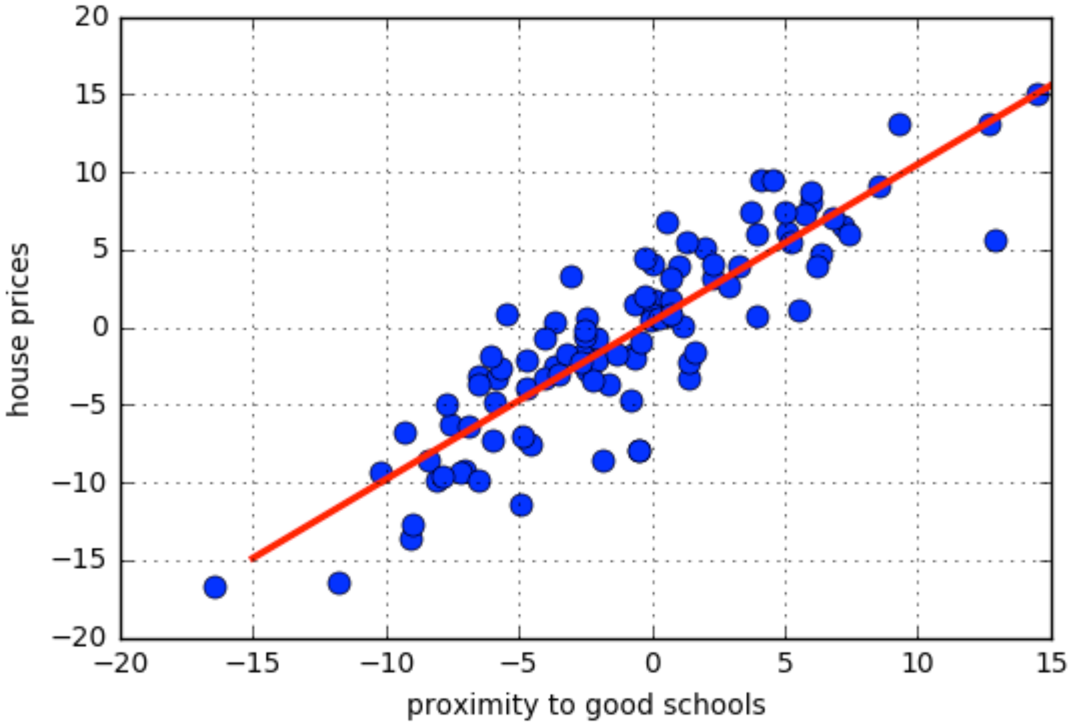
$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$$

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

Learning

$$\theta^* = \arg_\theta \min \sum_{n=1}^{N} \|y_n - \theta^T \mathbf{x}\|_2^2$$

Linear regression model



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

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

Learning

$$\theta^* = \arg_\theta \min \sum_{n=1}^{N} \|y_n - \theta^T \mathbf{x}\|_2^2$$

Linear and 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: $\mathbf{y} = y$

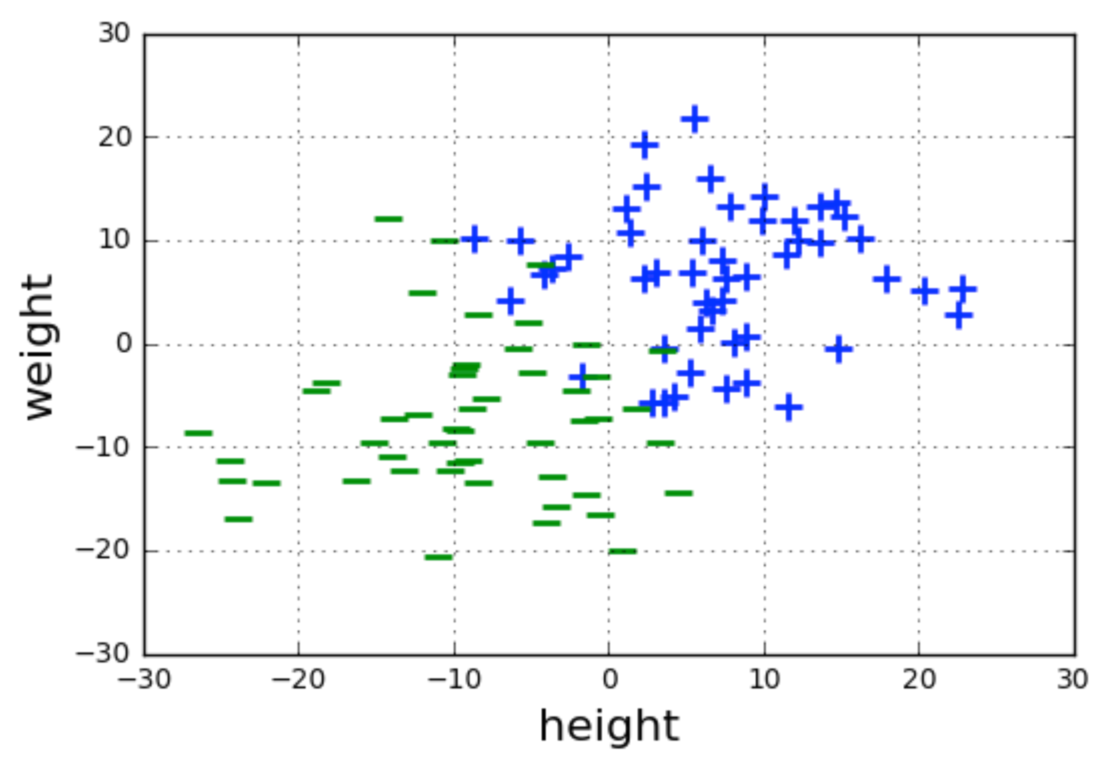
$$\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = \theta^T \mathbf{x}$$

- Parameters of the linear model: θ
- Linear model is for continuous labels
- Logistic regression is the extension to binary labels

Logistic regression model

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

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



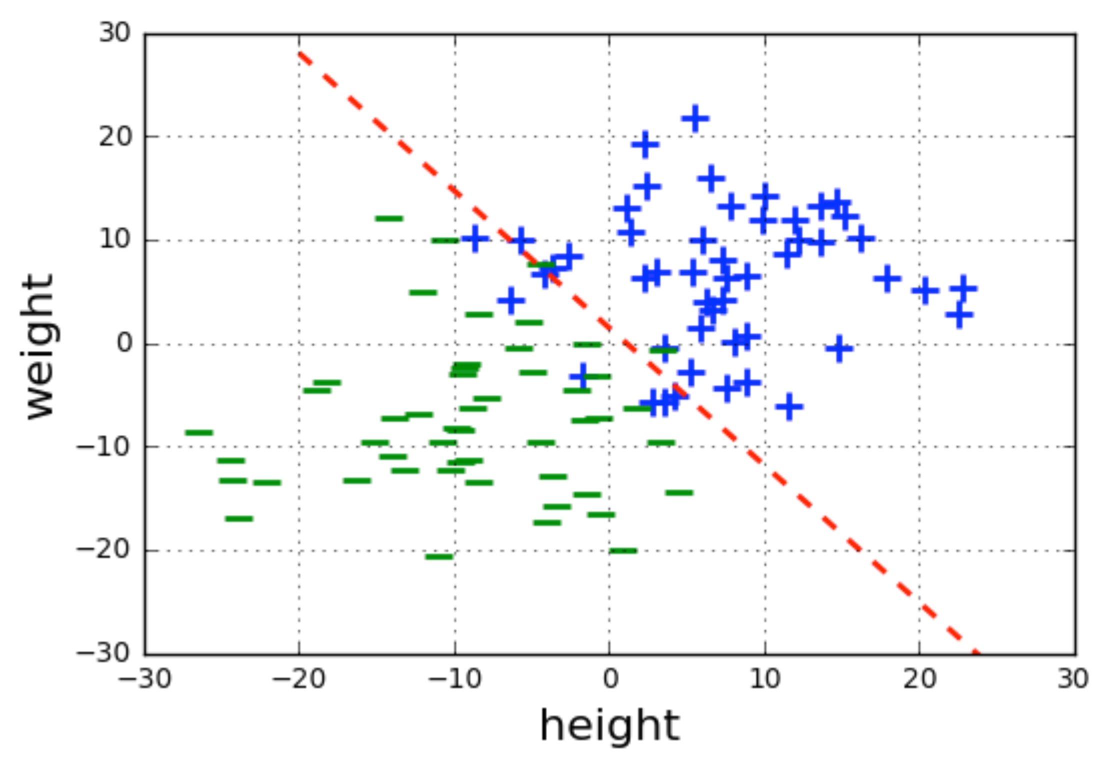
Learning

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

Computer Vision

Introduction
Learning basics
Math. basics

Logistic regression model



Line is the point where p(y=1)=0.5

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

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

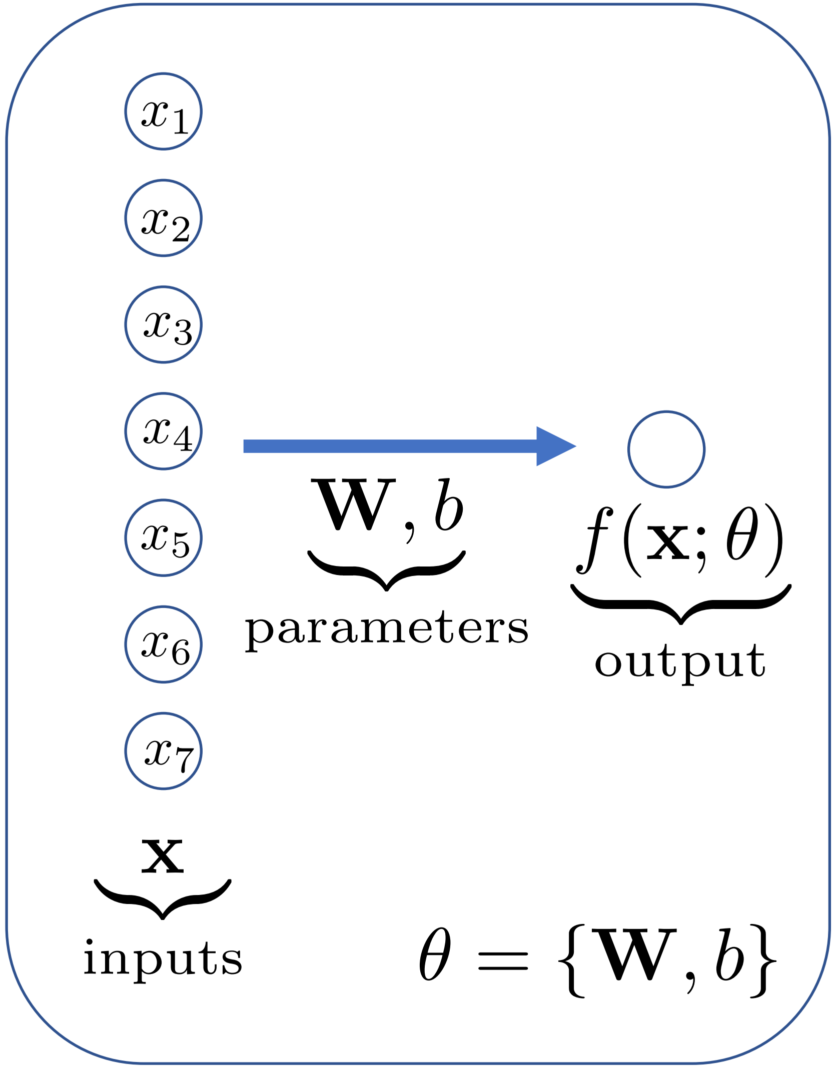
Learning

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

Logistic regression is the building block of the perceptron model

Section 2: Perceptron model and Multilayer extension

Basic perceptron model


x_1
x_2
x_3
x_4
x_5
x_6
x_7
$\underbrace{\mathbf{W}, b}_{\text{parameters}}$
$\underbrace{f(\mathbf{x}; \theta)}_{\text{output}}$
$\underbrace{\mathbf{x}}_{\text{inputs}}$
$\theta = \{\mathbf{W}, b\}$

Model of binary classification

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

Formed of two different parts

1. Activation

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

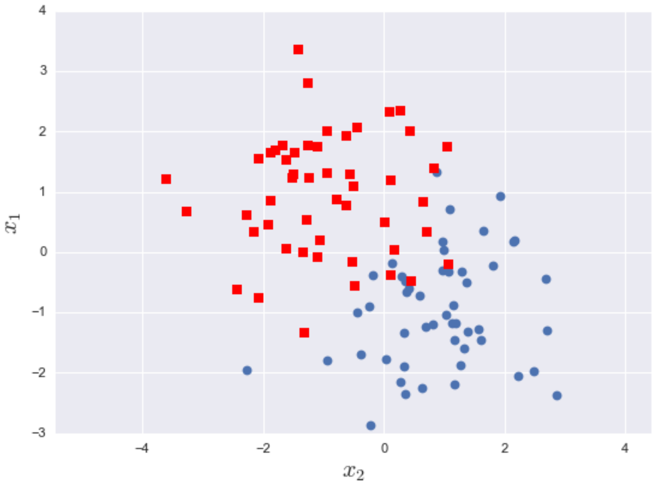
2. Nonlinearity

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

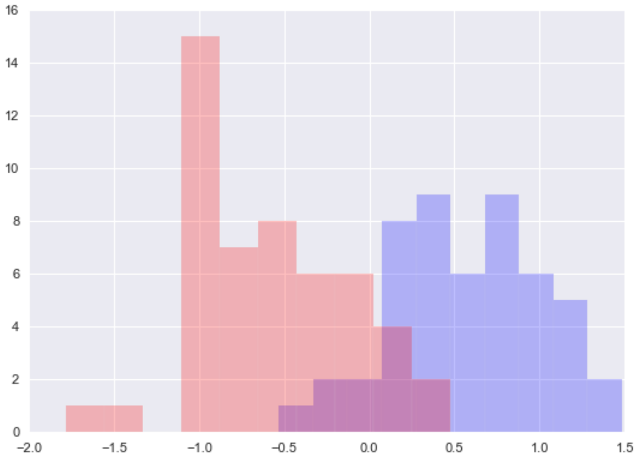
Activation

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

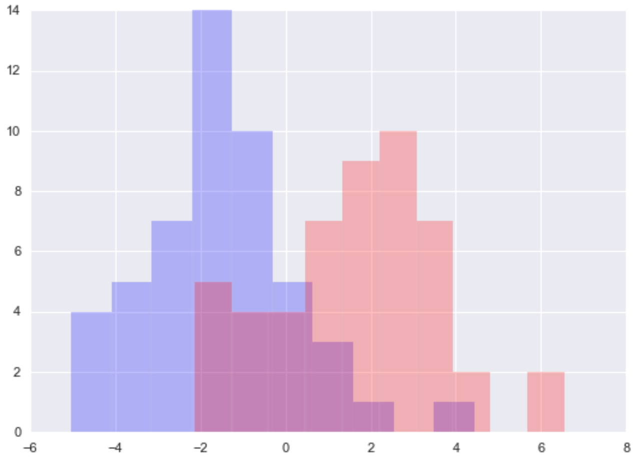
- Linear transformation of the features
- d + 1 number of parameters $\mathbf{W} \in \mathbb{R}^{1 \times d}$ $b \in \mathbb{R}$



Features and classes



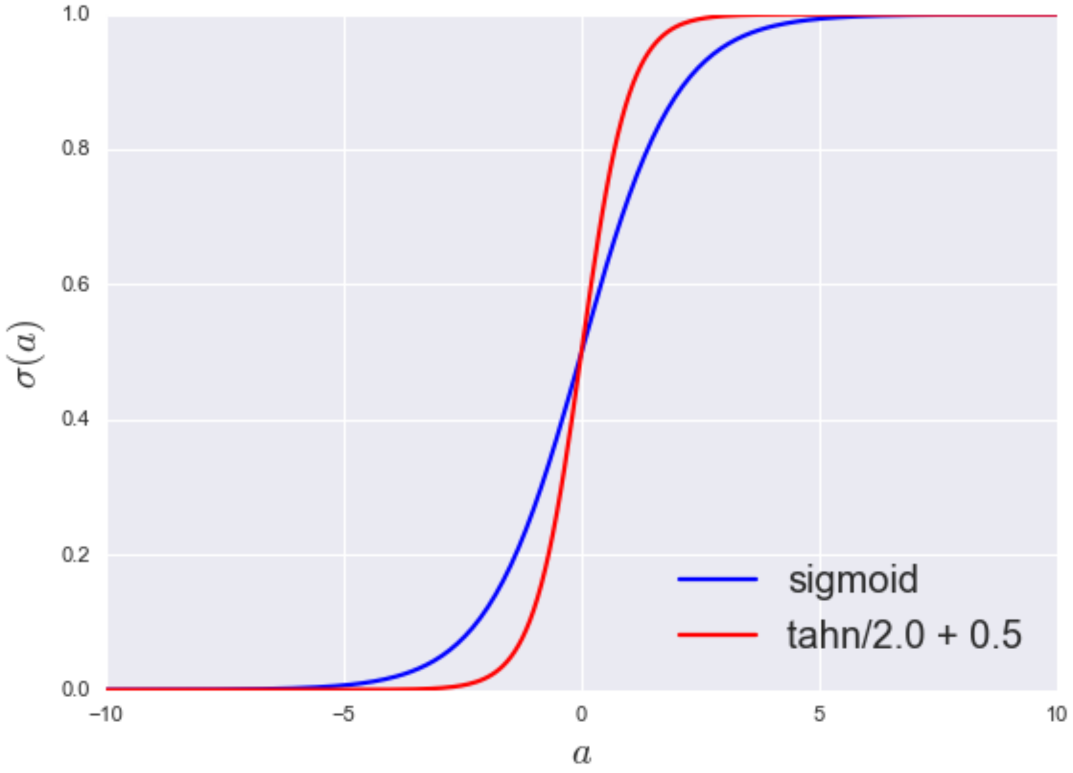
Transformation # 1



Transformation # 2

Non-linearity

- Maps real line to probabilities, i.e. $\sigma : \mathbb{R} \mapsto (0, 1)$
- Element-wise application



Sigmoid function

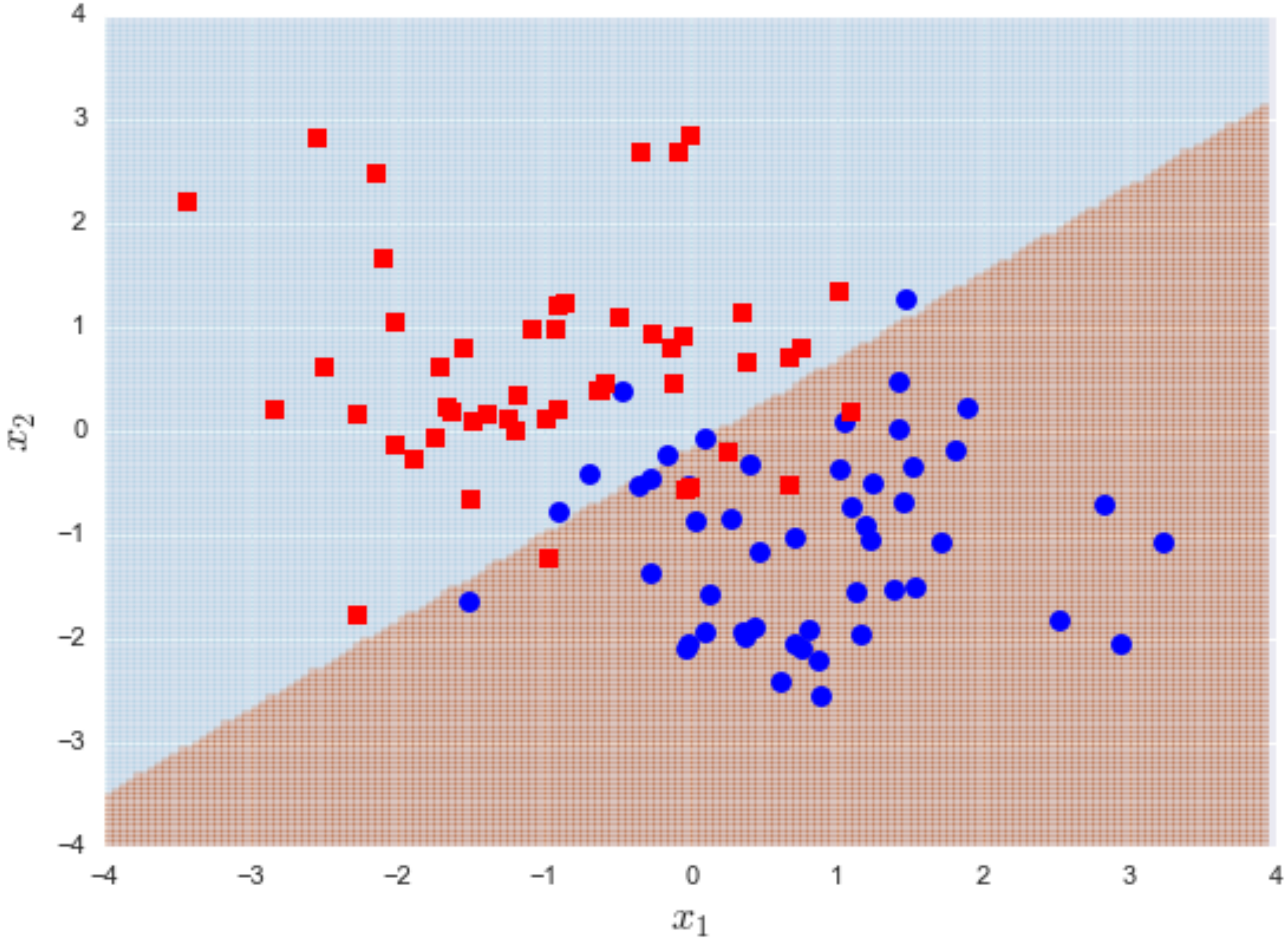
$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Tangent Hyperbolic

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

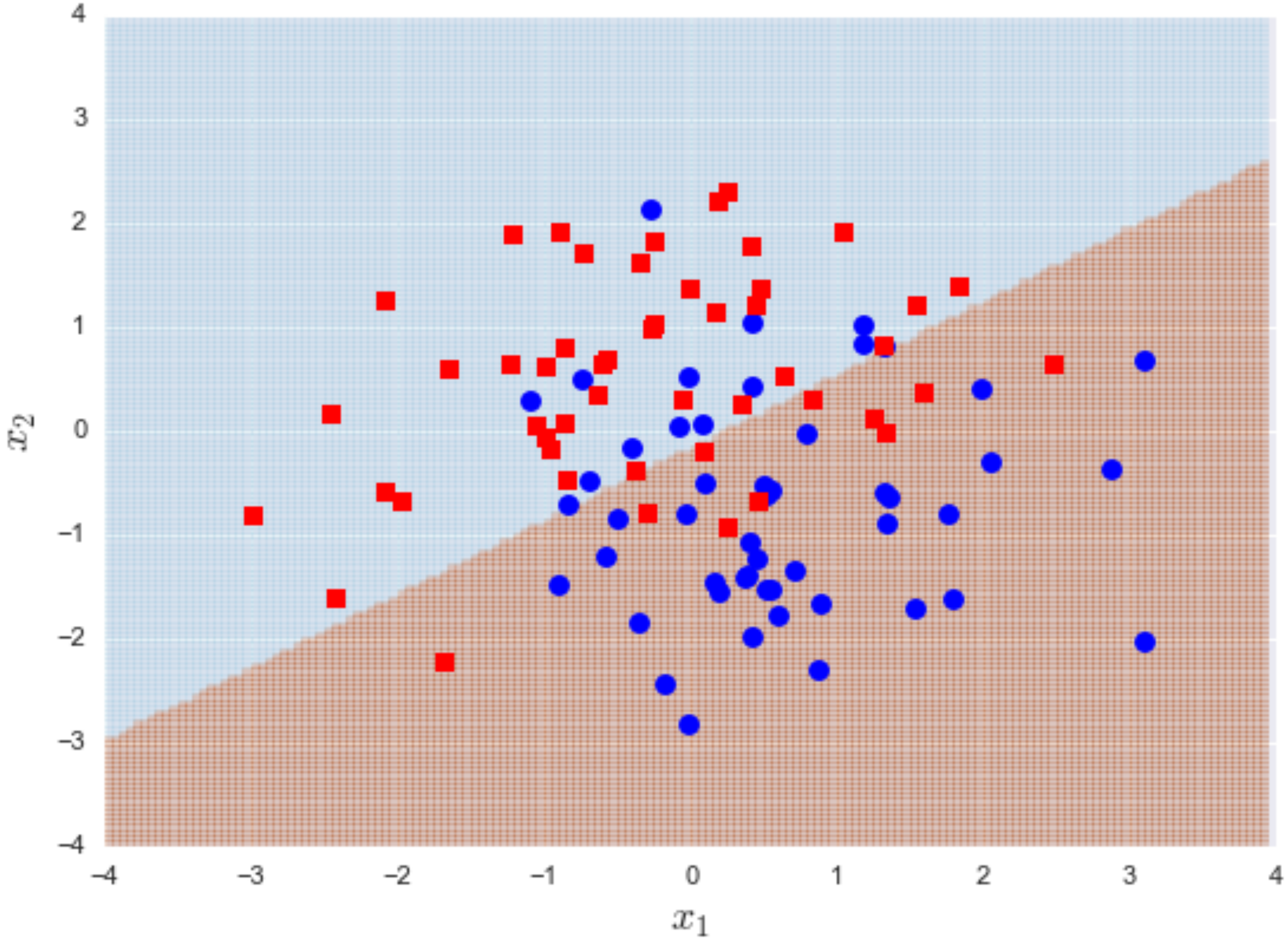
Decision boundary

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



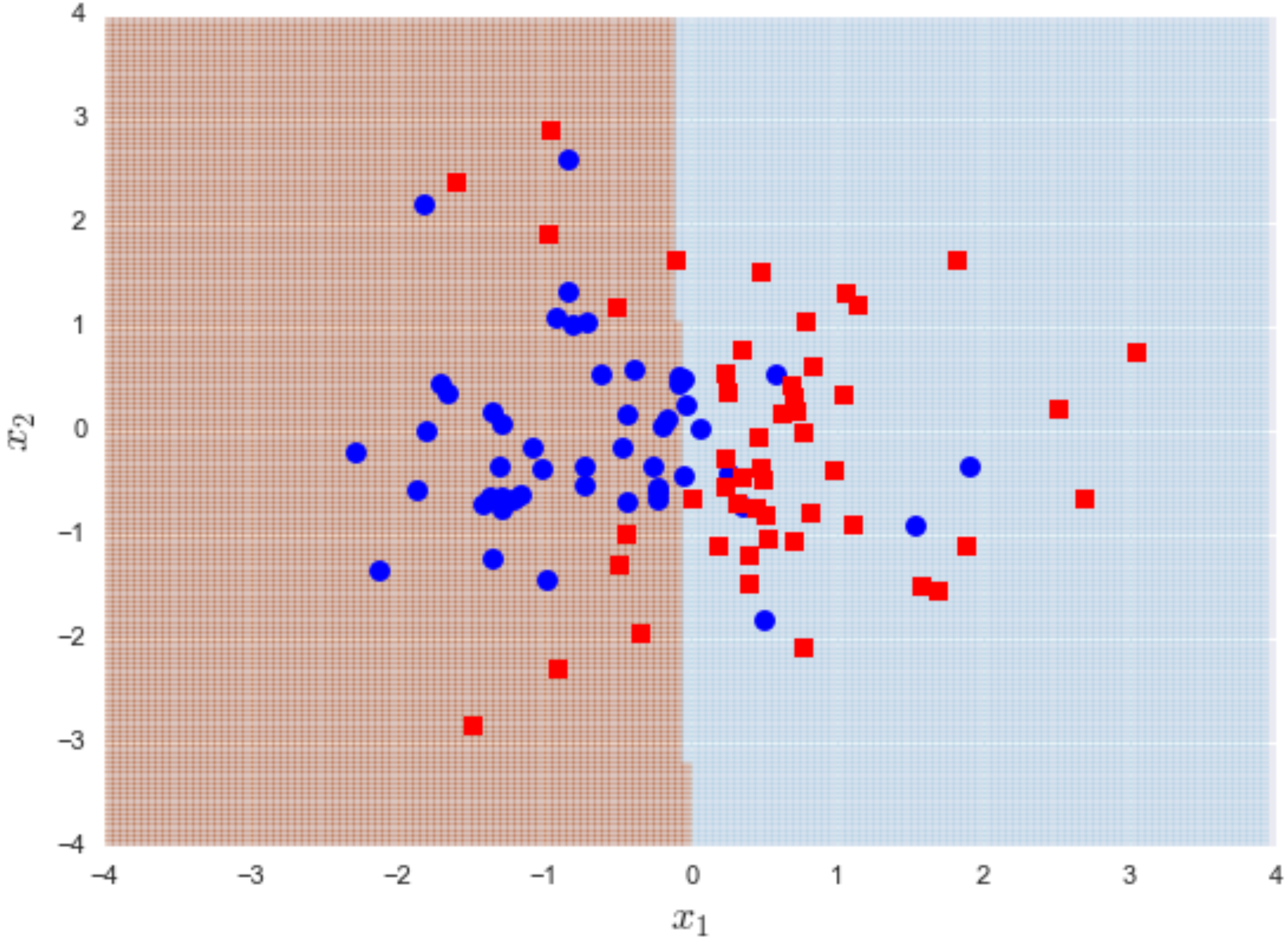
Decision boundary II

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



Decision boundary III

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



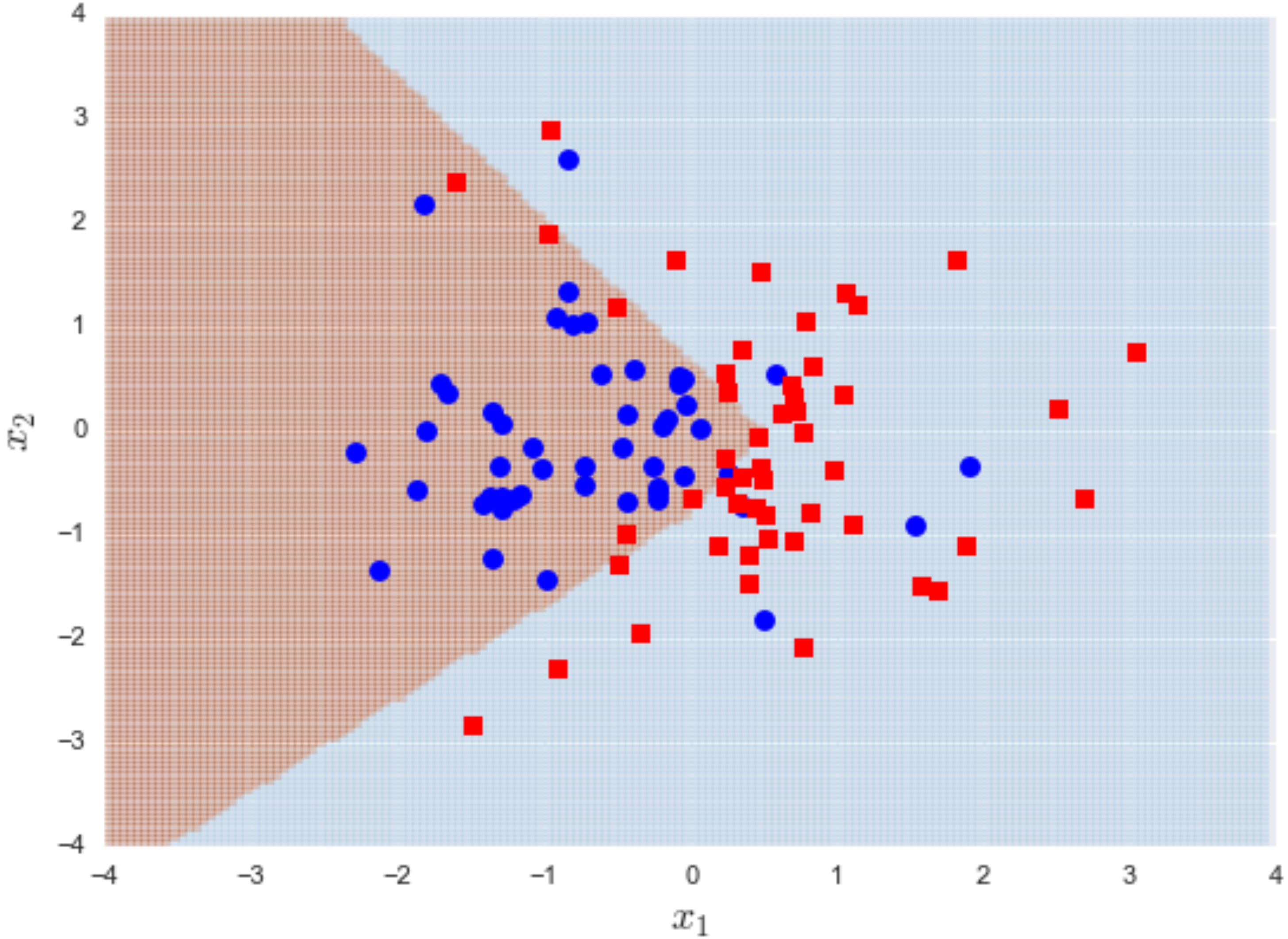
Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model

Notes on perceptron model

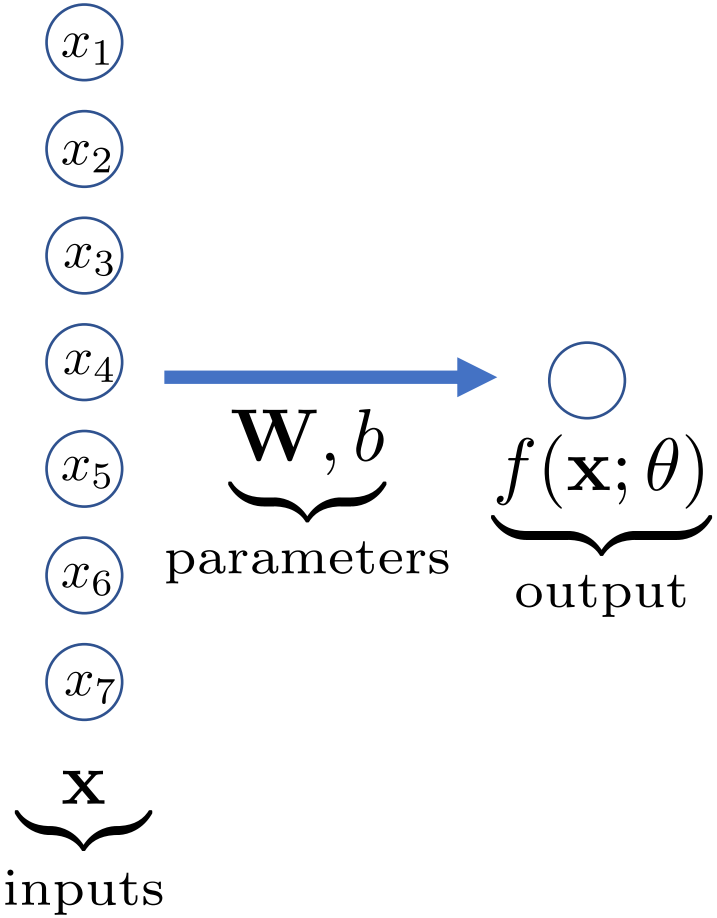
- Logistic regression is the building block
- Essentially a linear model
- Linear boundary separation and cannot model more complicated separation
- Building block for more complicated models
- We have not seen training yet, will come back that
- First let’s see more complicated models

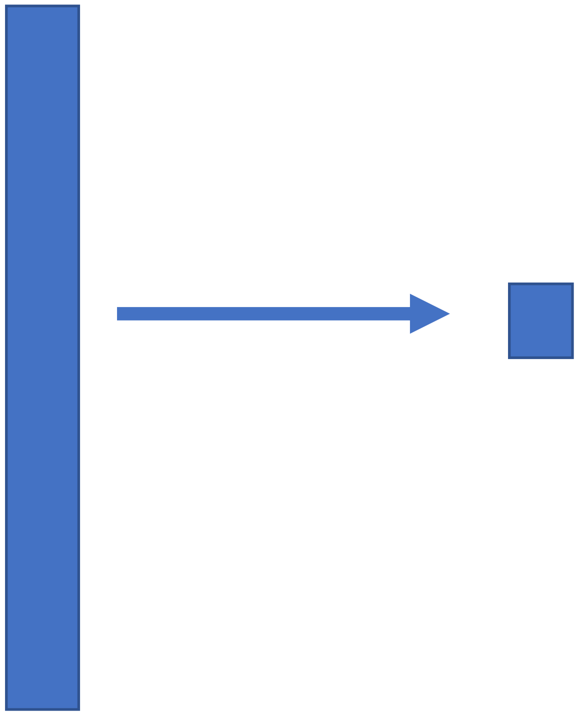
It would be better if



Tak

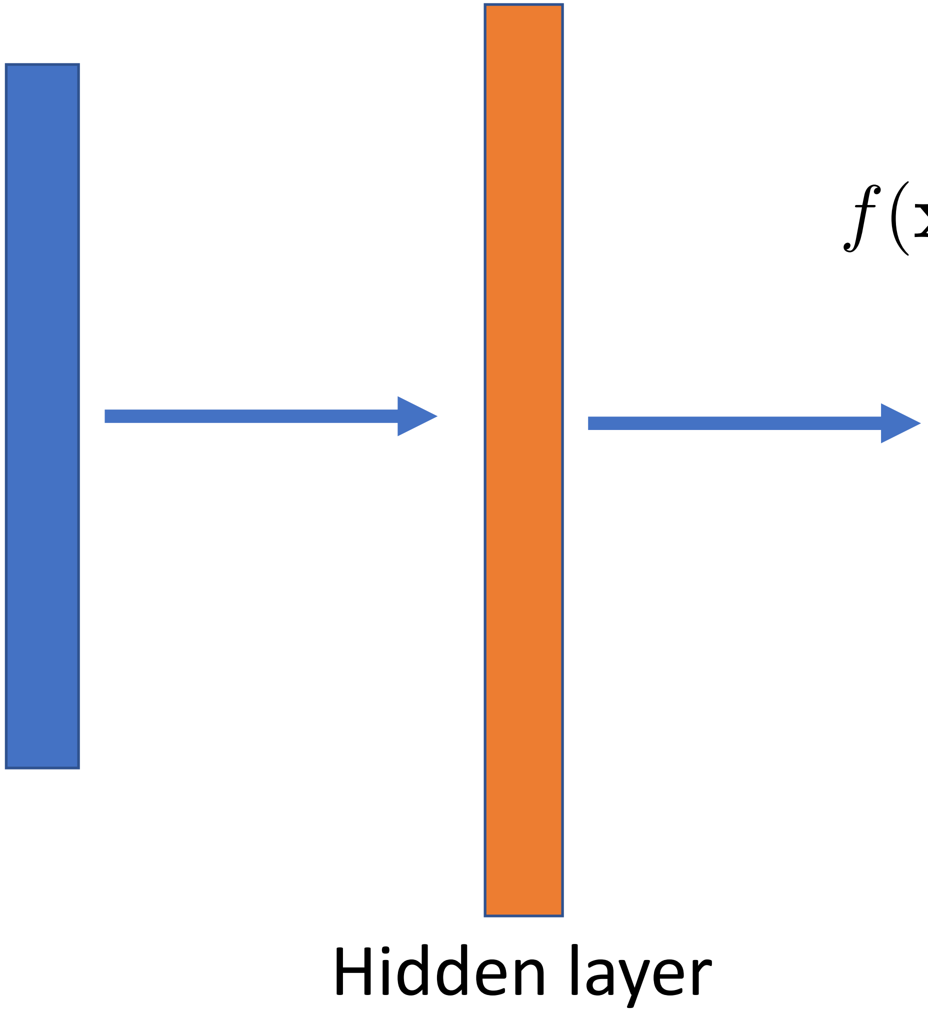
Simpler graphical representation


x_1
x_2
x_3
x_4
x_5
x_6
x_7
$\underbrace{\mathbf{x}}_{\text{inputs}}$
$\underbrace{\mathbf{W}, b}_{\text{parameters}}$
$\underbrace{f(\mathbf{x};\theta)}_{\text{output}}$



Simpler graphical representation

Multilayer perceptron (MLP)



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

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

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

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

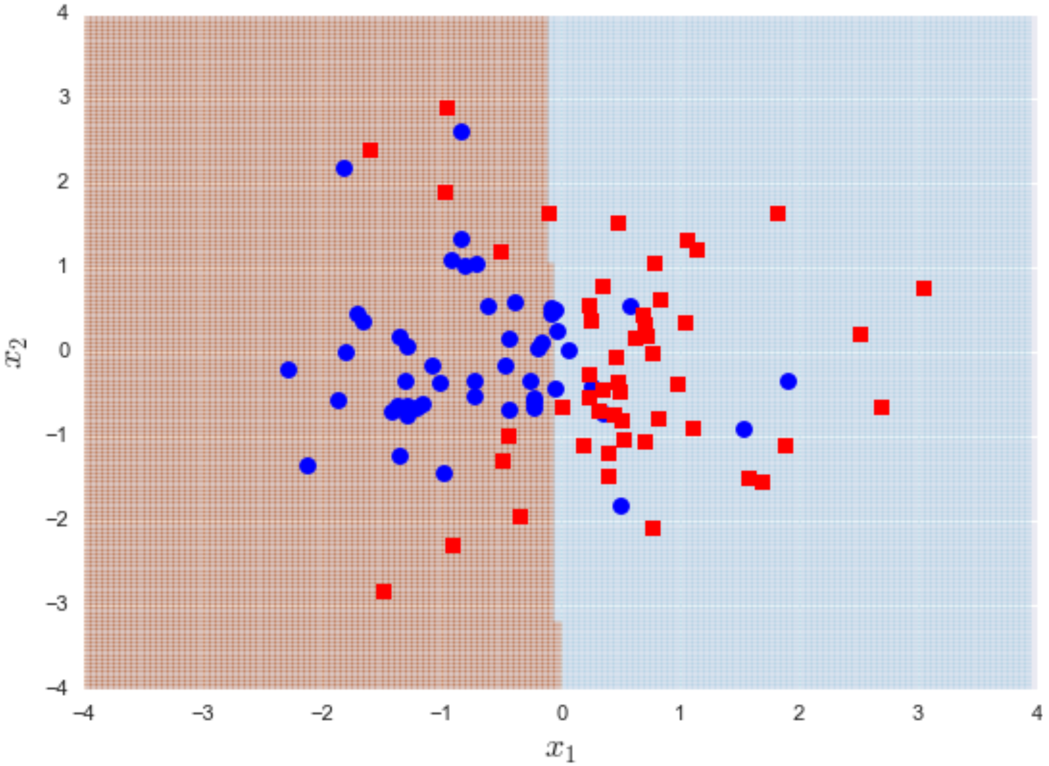
Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP

Multilayer perceptron (MLP)

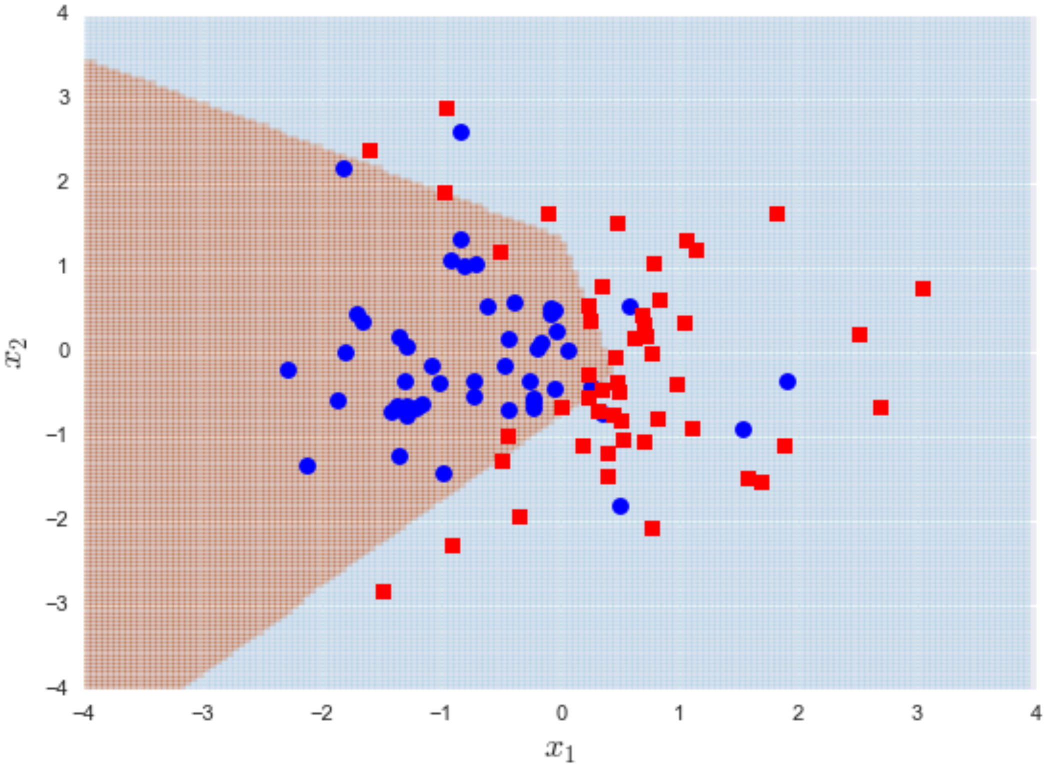
$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$$

Linear separation



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

Non-linear separation



Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP

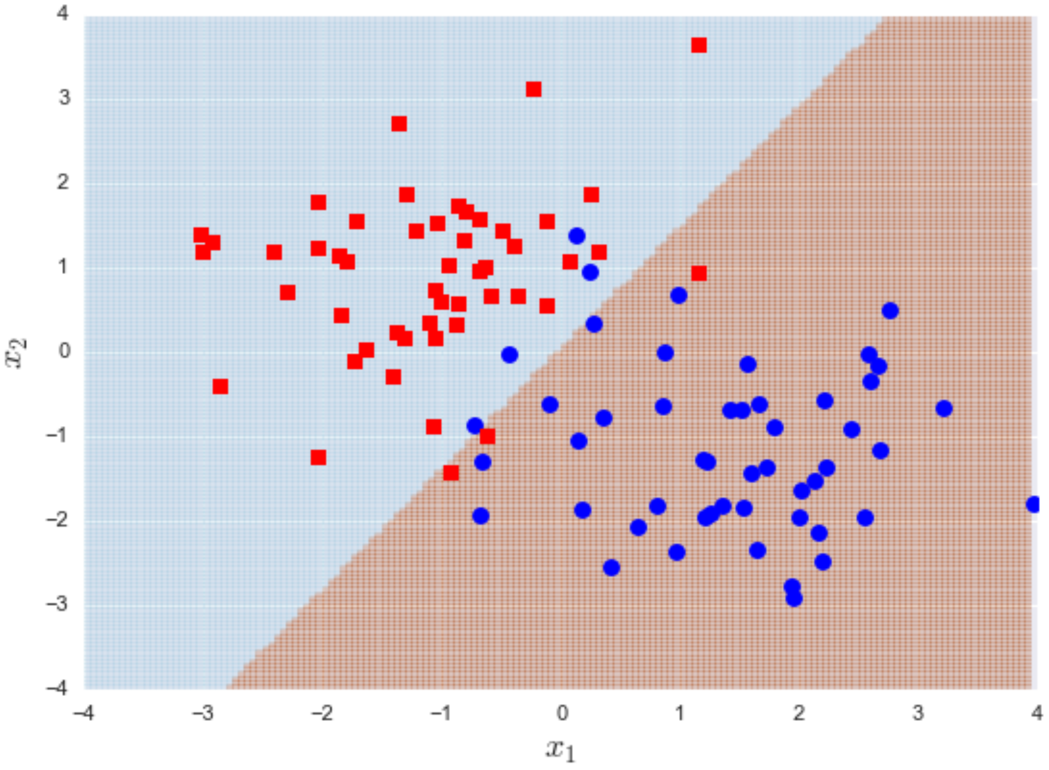
Multilayer perceptron (MLP)

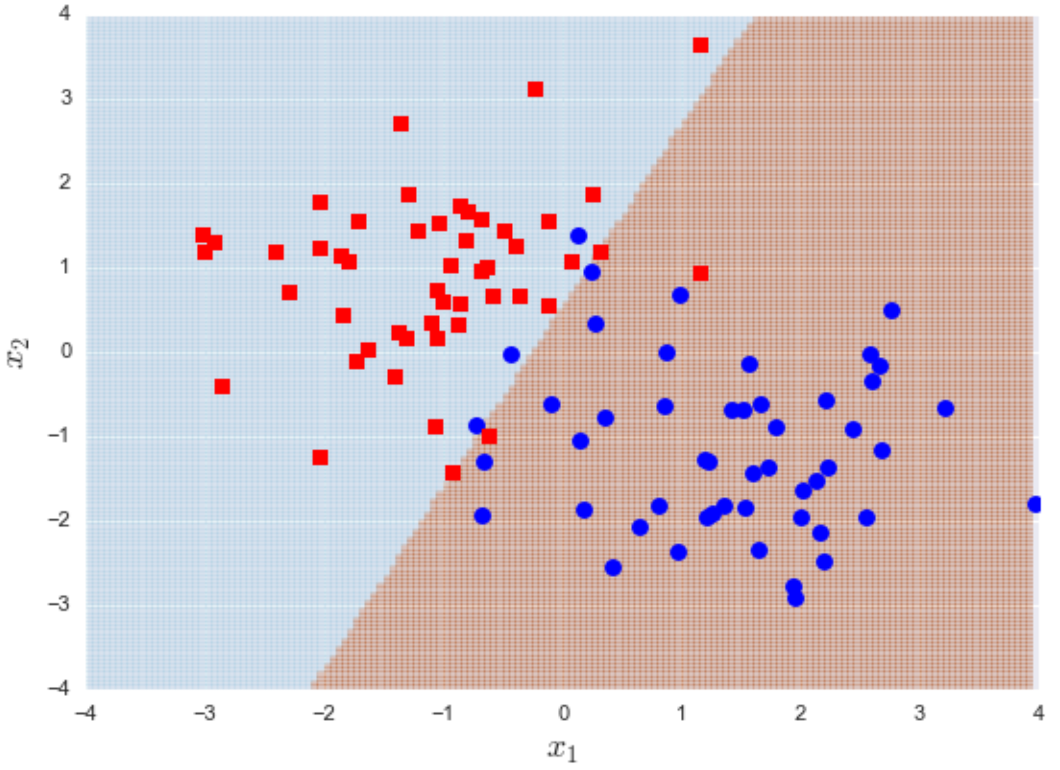
$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x} + b)$$

Linear separation

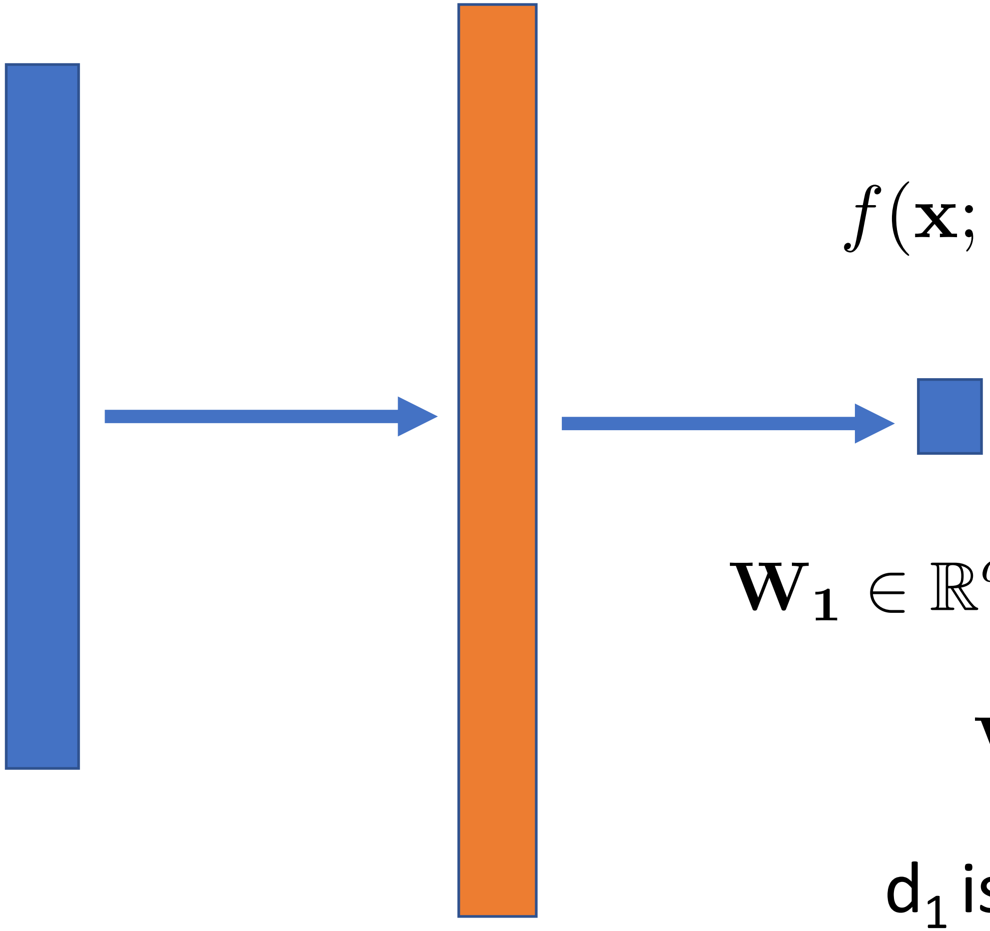
$$f(\mathbf{x};\theta) = f_2(f_1(\mathbf{x};\theta_1);\theta_2)$$

Able to do linear separation





MLP - width



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

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

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

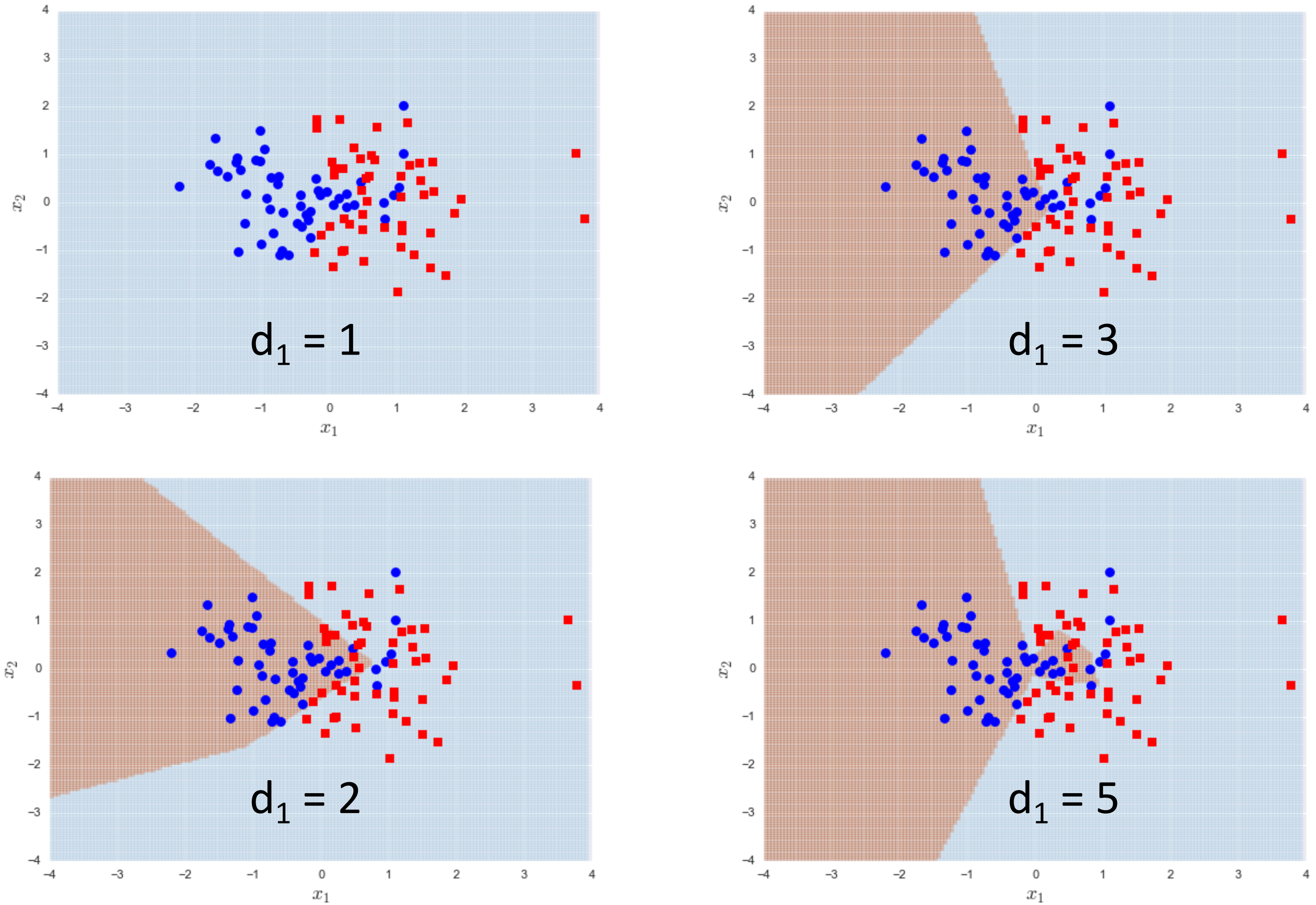
$$\mathbf{W_2} \in \mathbb{R}^{1 \times d_1} \implies f(\mathbf{x};\theta) \in \mathbb{R}$$

d_1 is the width of the hidden layer

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP

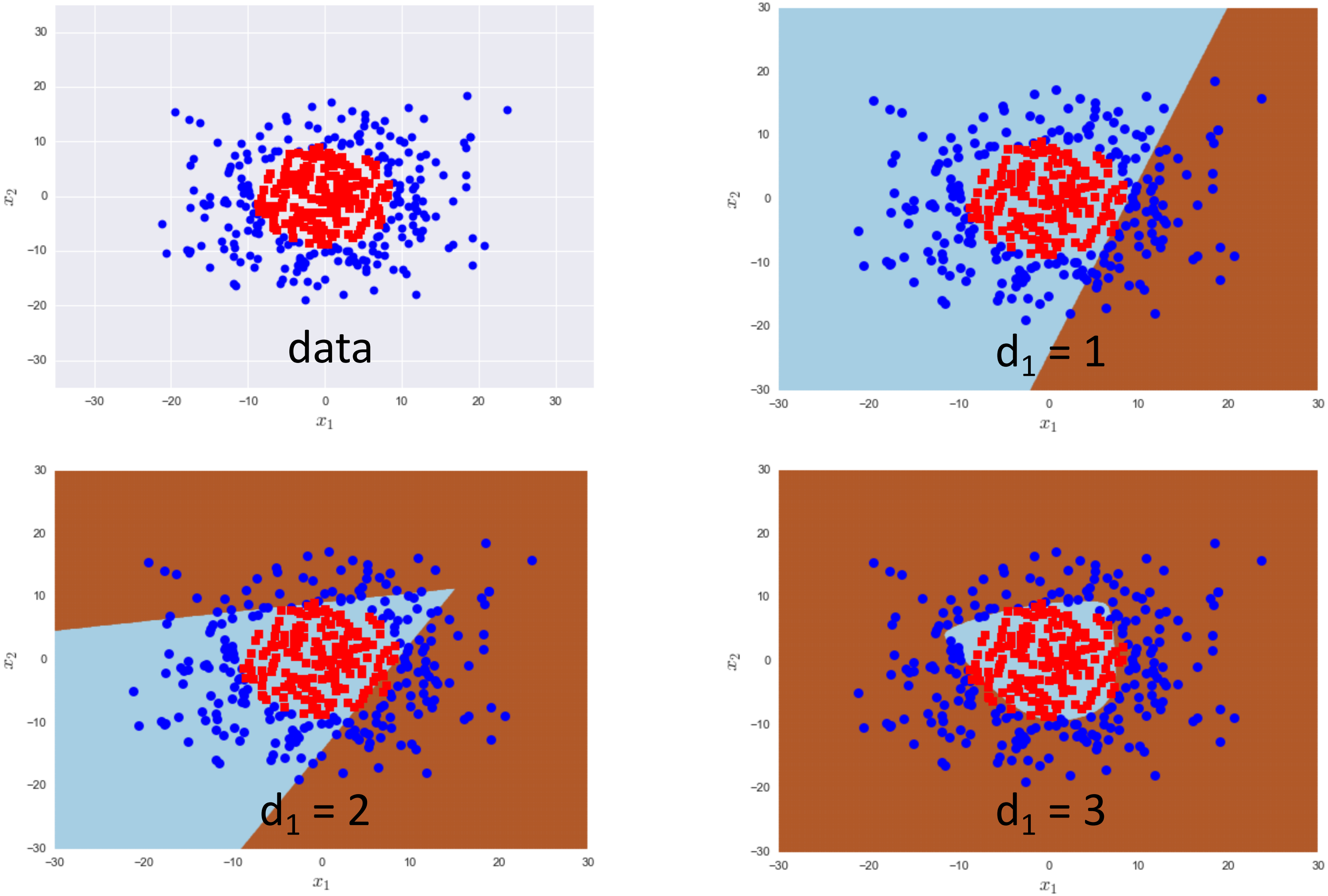
Decision boundary at different width



Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP

Decision boundary at different width



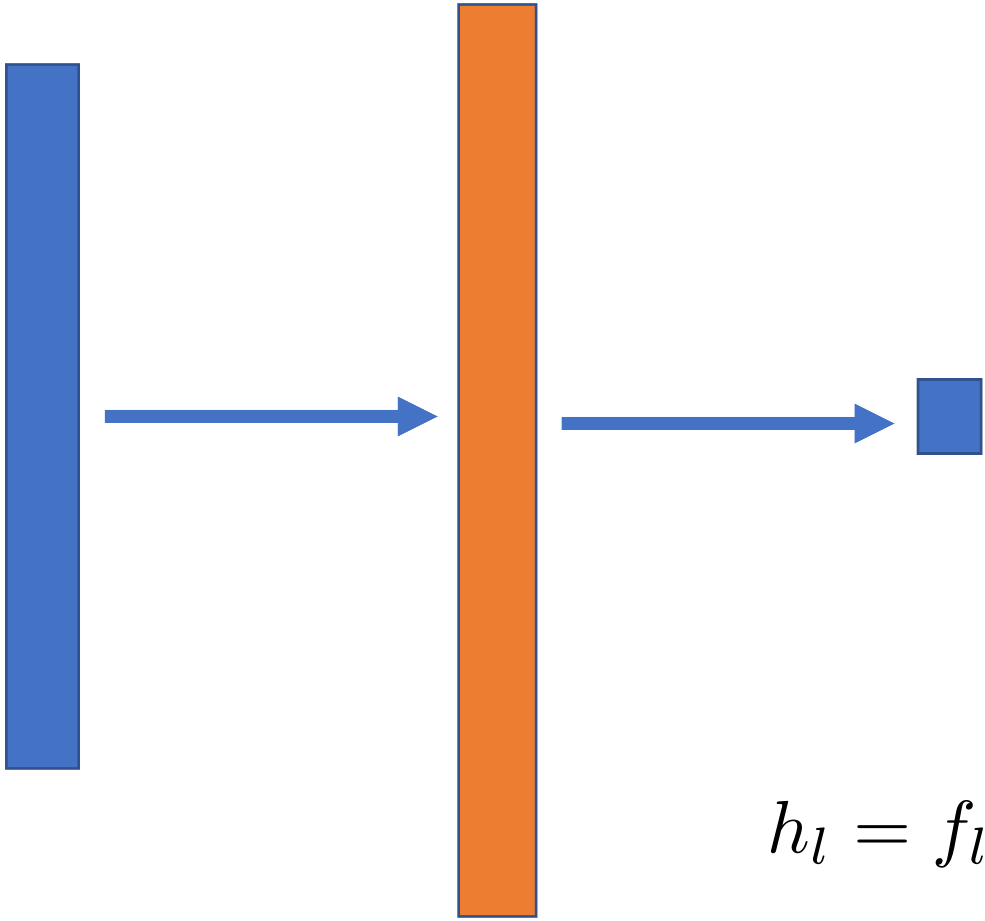
MLP – notes on width

- $d_1 = d$ -- 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, remember PCA? – will talk about this later

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP

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

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

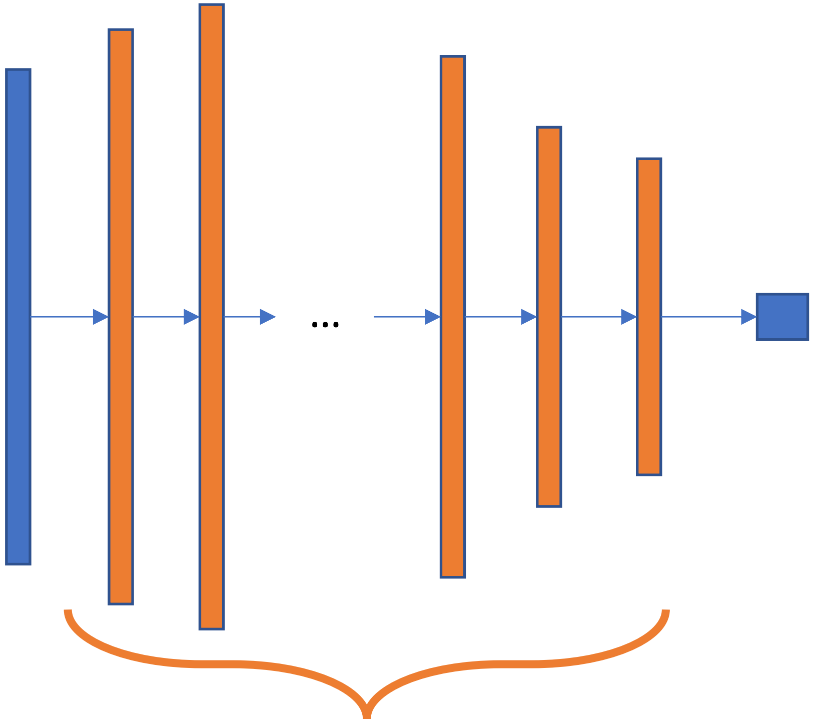
$$h_0 = \mathbf{x},\ h_L = f(\mathbf{x};\theta)$$

Function view

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

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

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



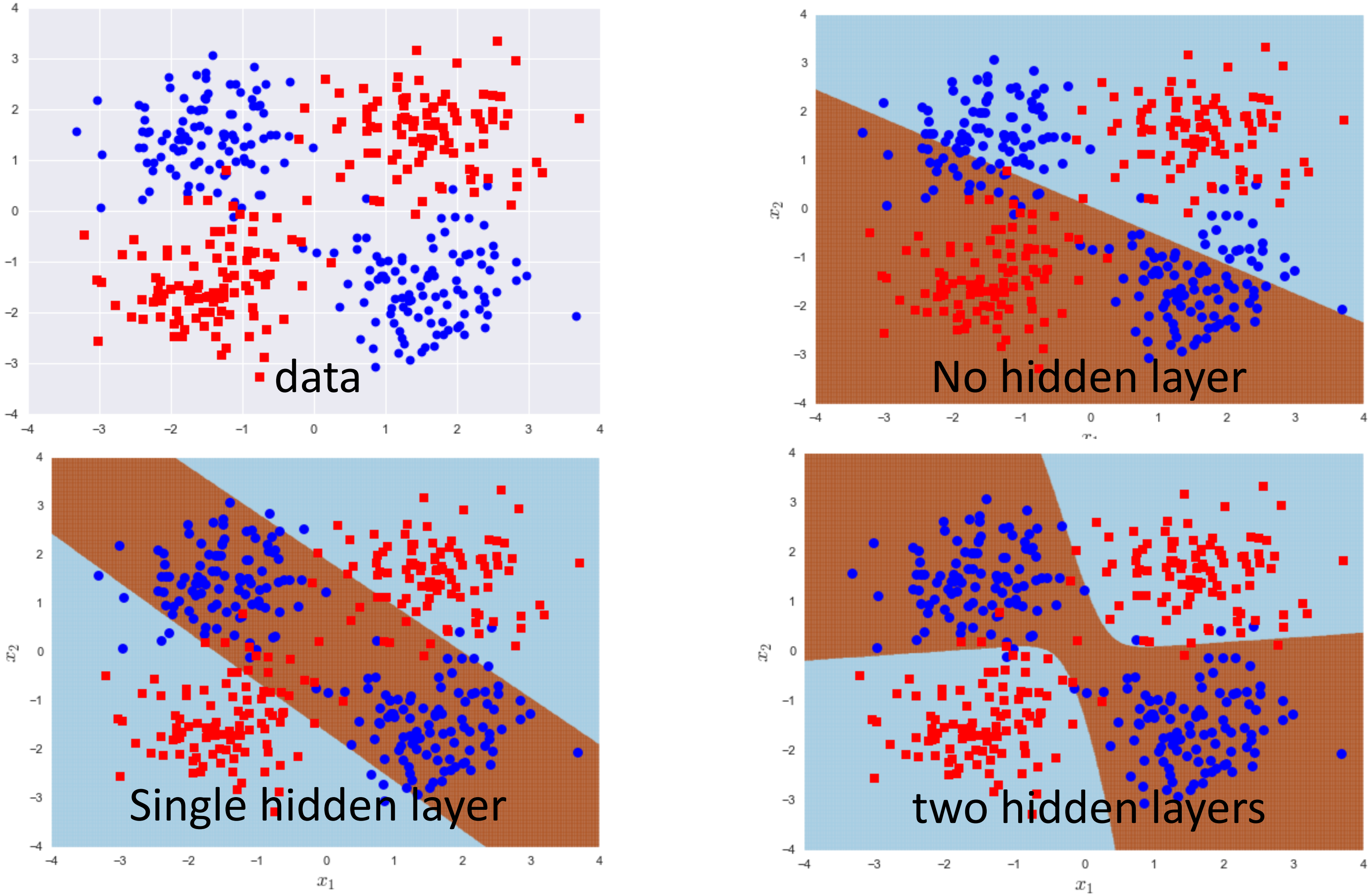
Hidden layers

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP

Decision boundary at different depths

[width = 2 at all depths]



data

No hidden layer

Single hidden layer

two hidden layers

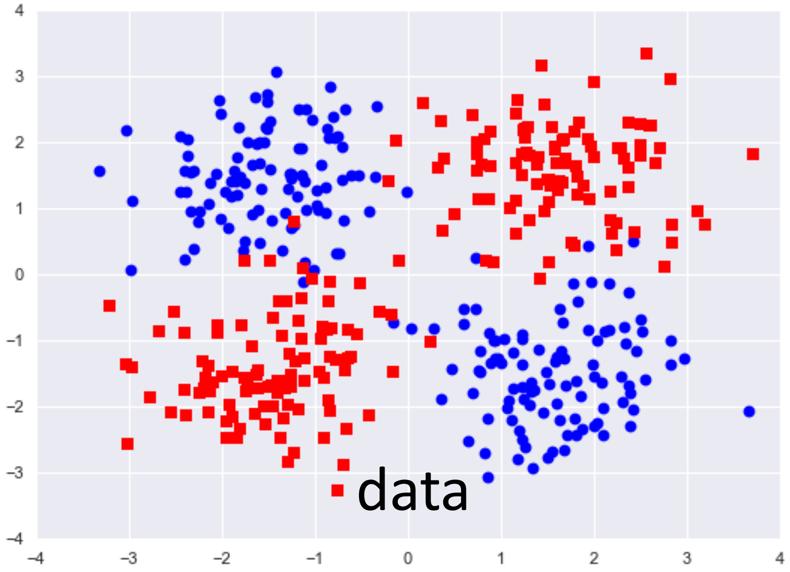
MLP – notes on depth

- No hidden layer – logistic regression / linear
- Increasing depth
    - Allows more complicated models
    - Leads to larger number of model parameters
    - Needs more samples to fit reliably
- May become difficult to train – will talk about it later

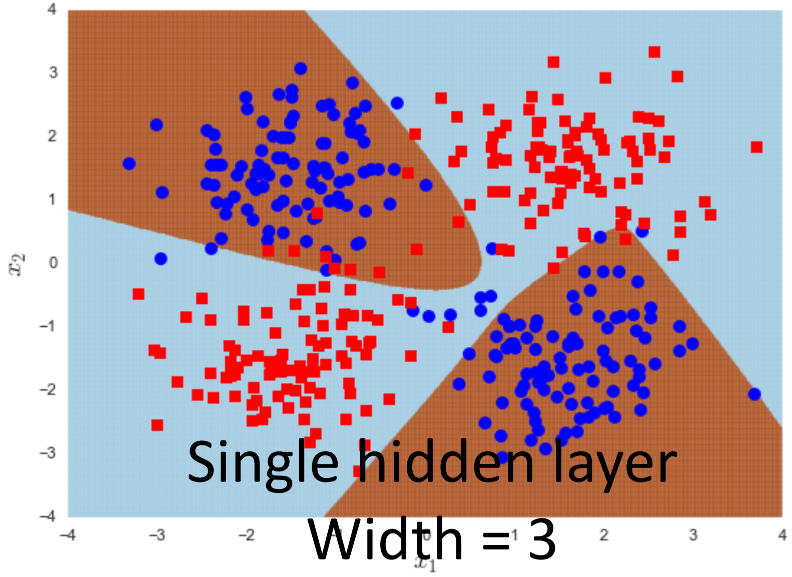
Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP

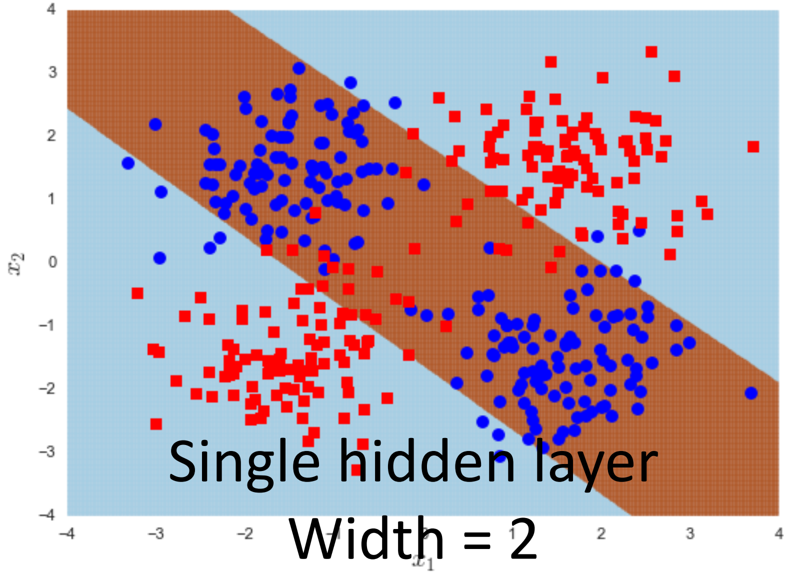
Depth / width interaction



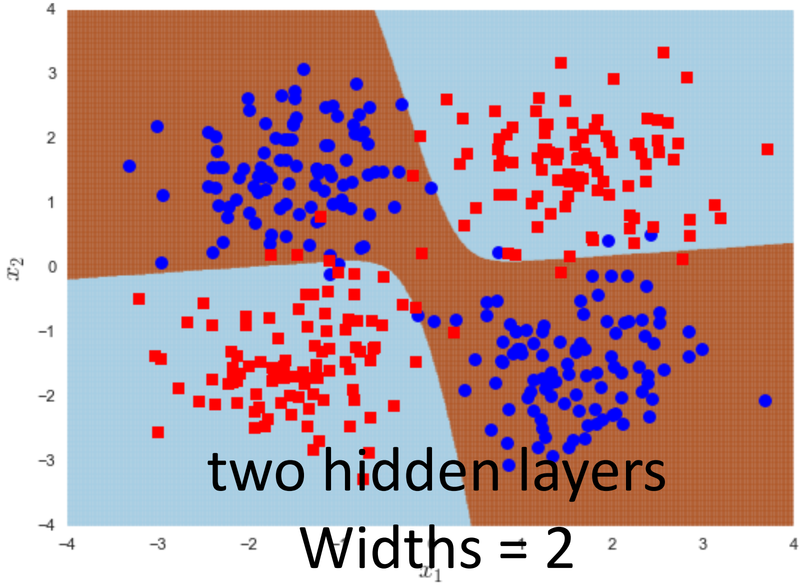
data



Single hidden layer
Width = 3



Single hidden layer
Width = 2



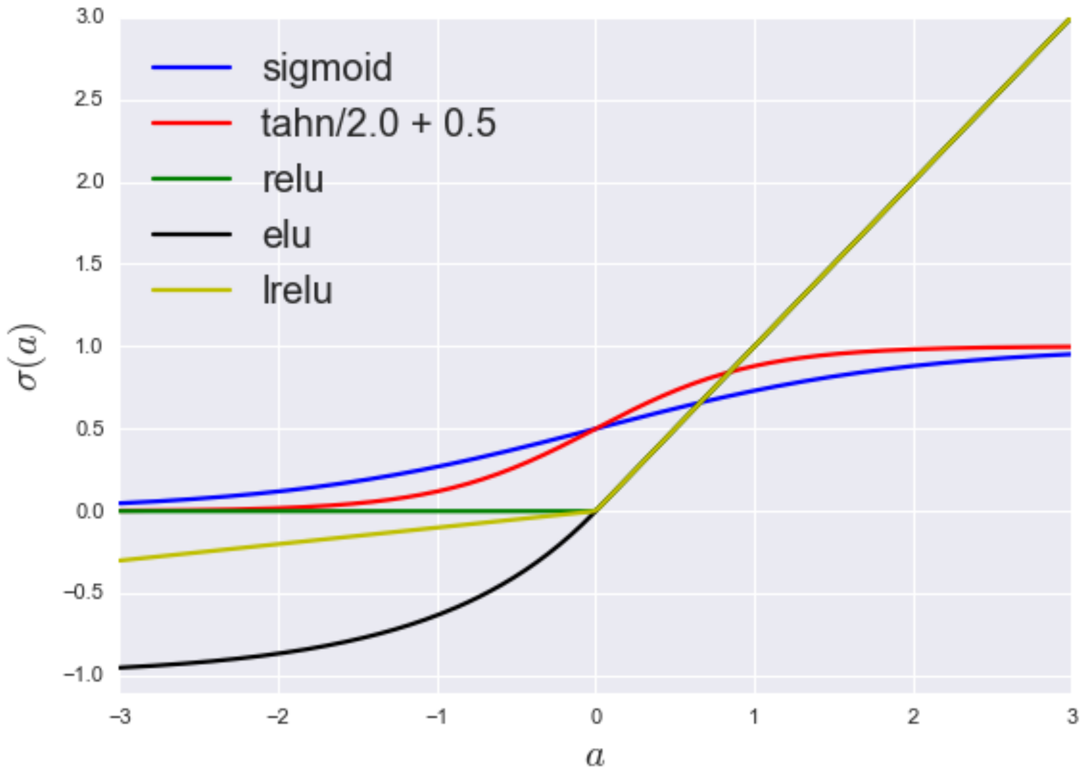
two hidden layers
Widths = 2

MLP – notes on depth and width

- Complicated interaction between depth and width
- Both allow modeling more complicated class separation
- 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
- 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

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

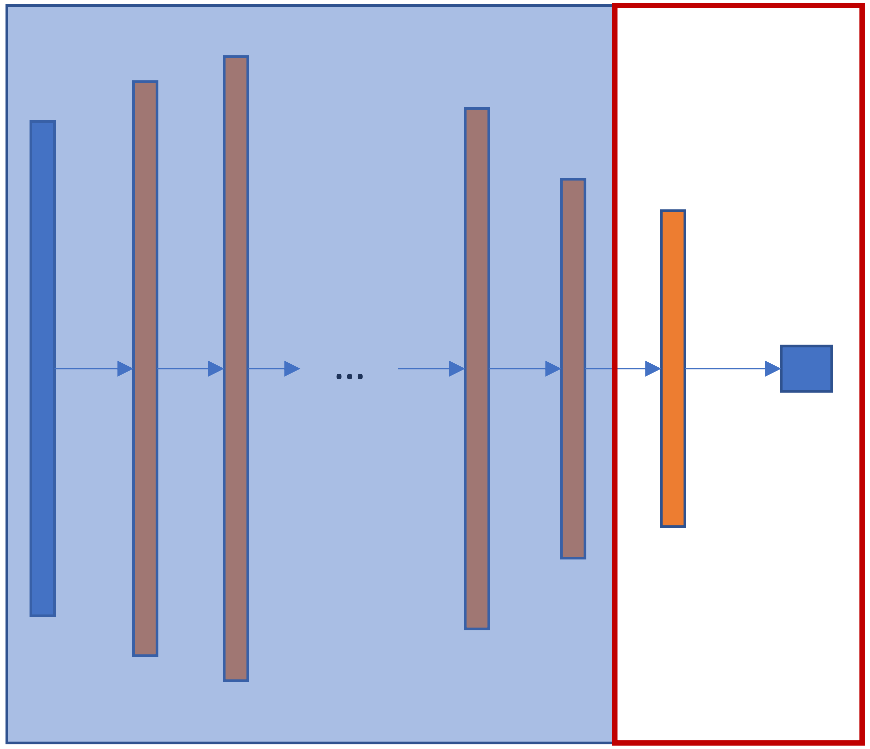
Exponential Linear Unit (Elu)

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

Notes on non-linear functions

- Main difference between non-linear functions is their derivatives
- Sigmoid and tanh saturate for high values, their derivatives diminish
- Relu has constant derivative for positive and 0 for negative values, it does not saturate for positive values [Hahnloser et al. 2000, Hahnloser and Seung 2001]
- Elu [Clevert, Unterthiner and Hochreiter 2015], Leaky Relu and Parametric Relu [He et al. 2015] are variations on the Relu idea.
- Relu is the activation of infinite number of binary sigmoid nodes combined together [Nair and Hinton 2010]
- Relu, Elu and Leaky Relu are yield good empirical performance

MLP – feature space view



Feature extraction

Linear classification

- There is a linear model at the very final layer

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

- 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

MLP – why non-linear feature map

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

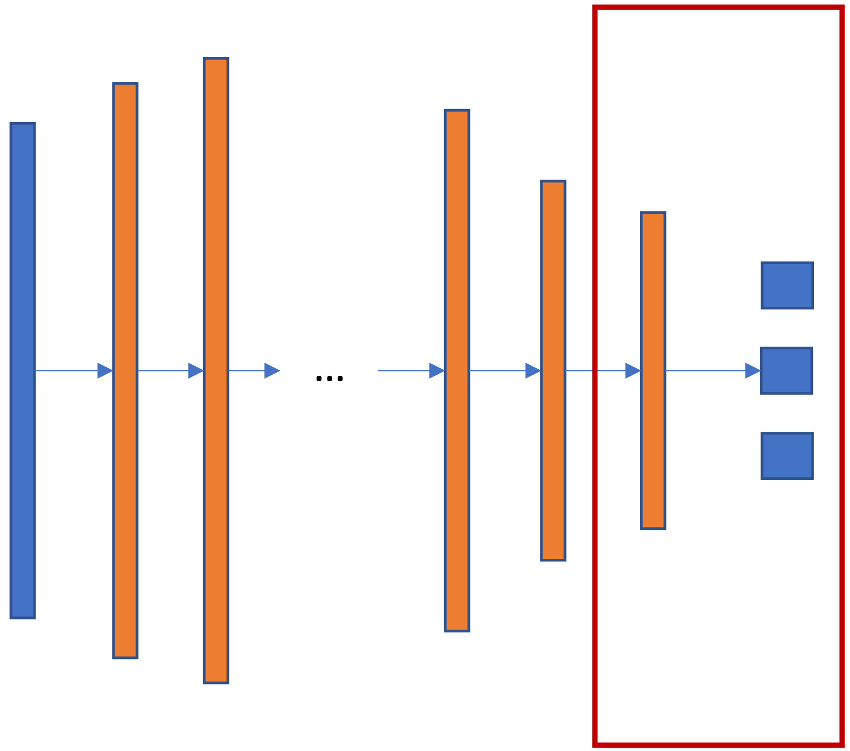
$$\sigma(a) = \mathbf{V}a + c$$

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

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

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

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

Fully Connected Classification Network

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

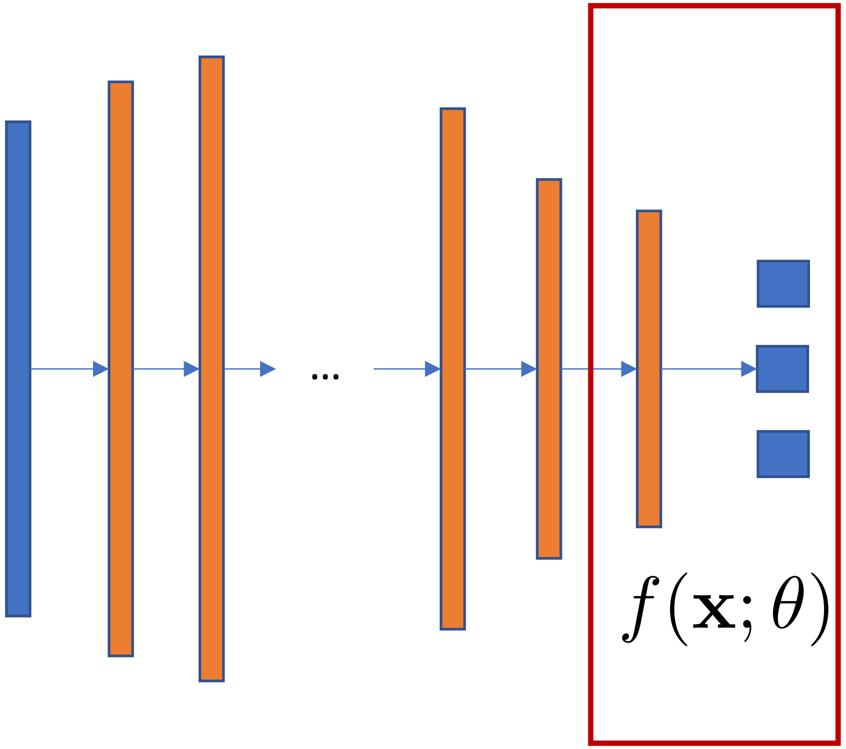
$$p(y=k) = f_k(\mathbf{x};\theta) = \frac{e^{a_{L,k}}}{\sum_{k' \in \mathcal{C}} e^{a_{L,k'}}} \quad \sum_{k \in \mathcal{C}} p(y=k) = 1$$

- Soft-max function

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP

Fully Connected Regression Network

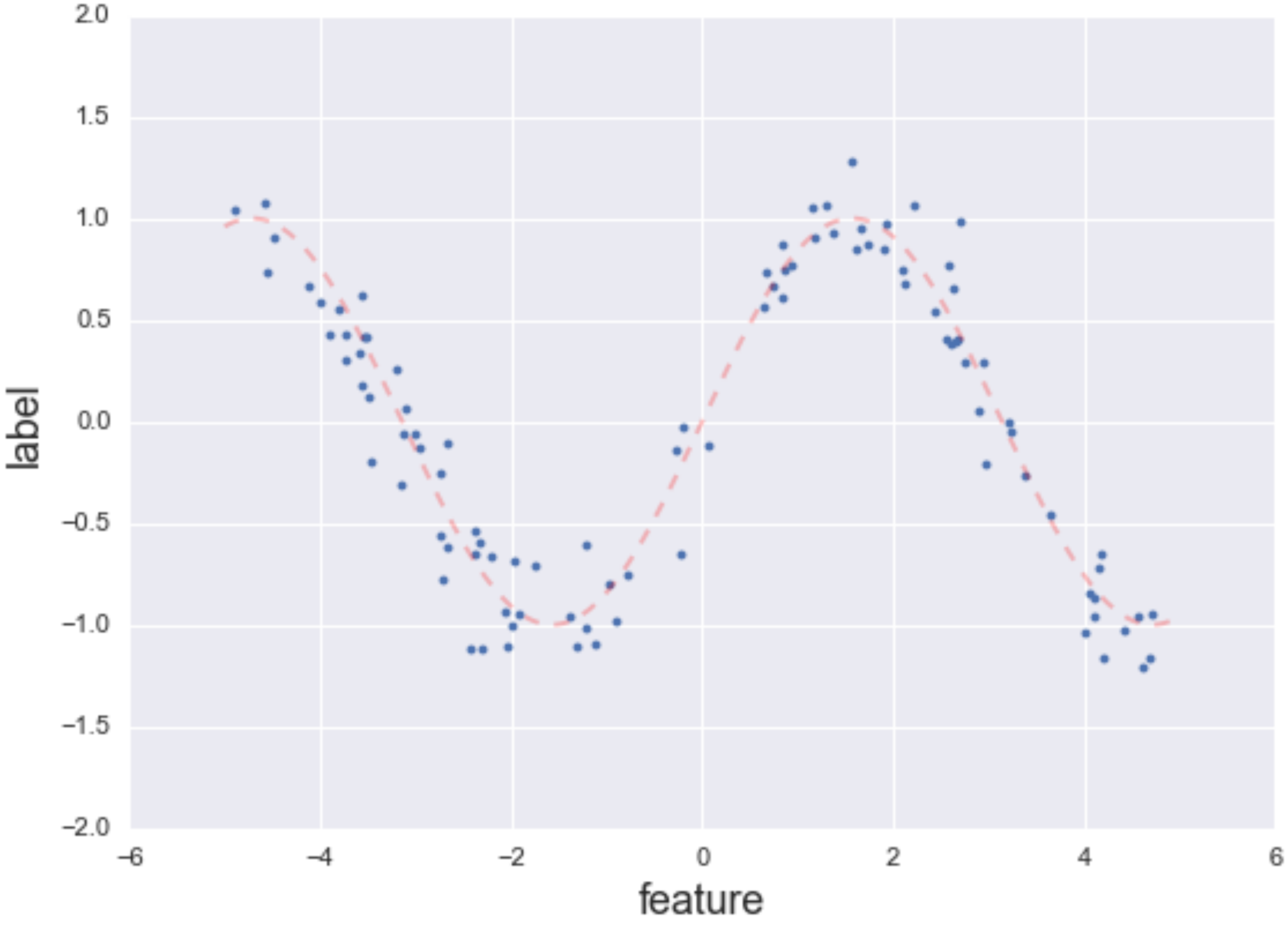


- Regression network is very similar to the classification network
- For an M dimensional regression problem

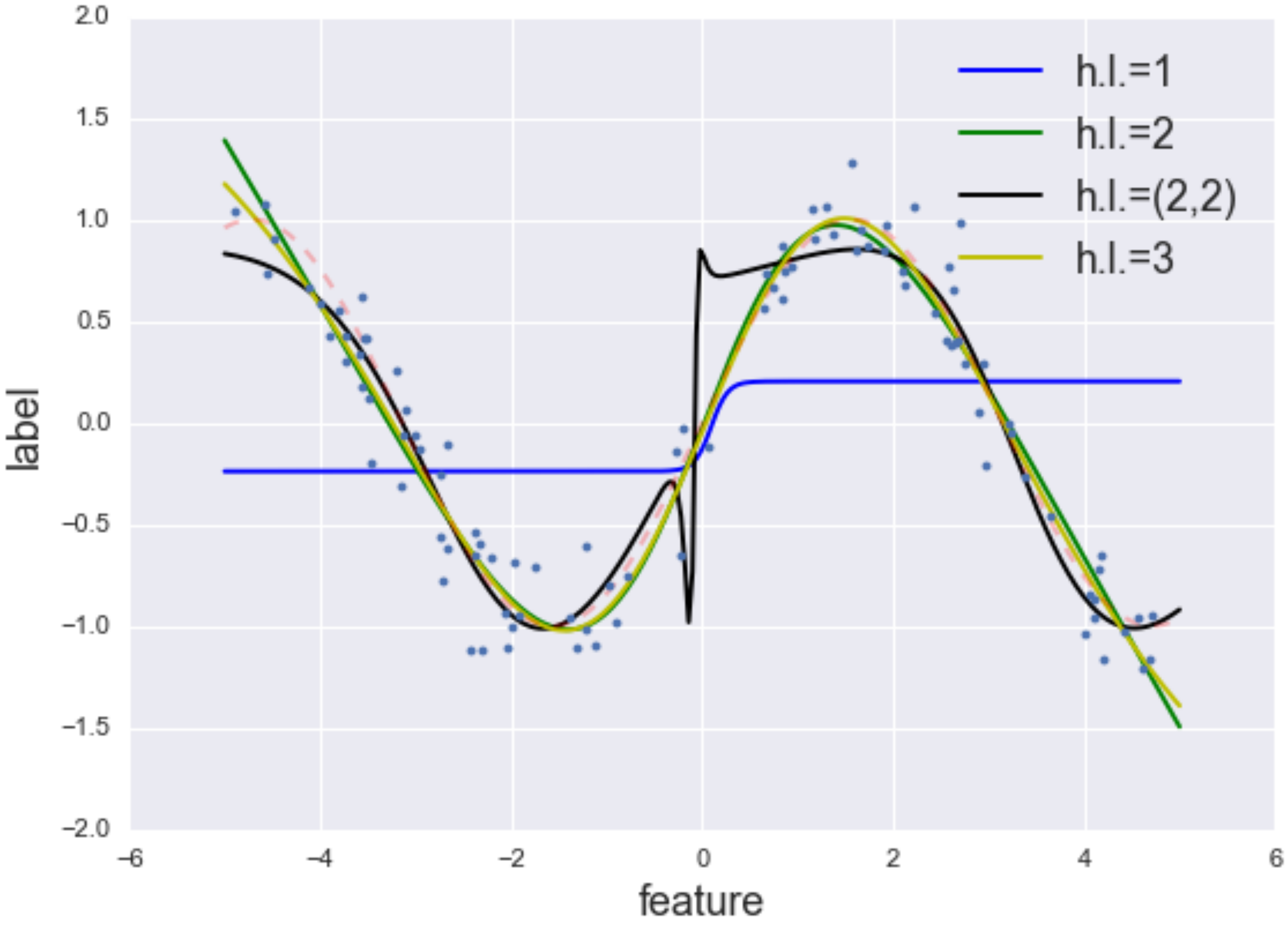
$$y \in \mathbb{R}^M \qquad f(\mathbf{x};\theta) \in \mathbb{R}^M$$

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

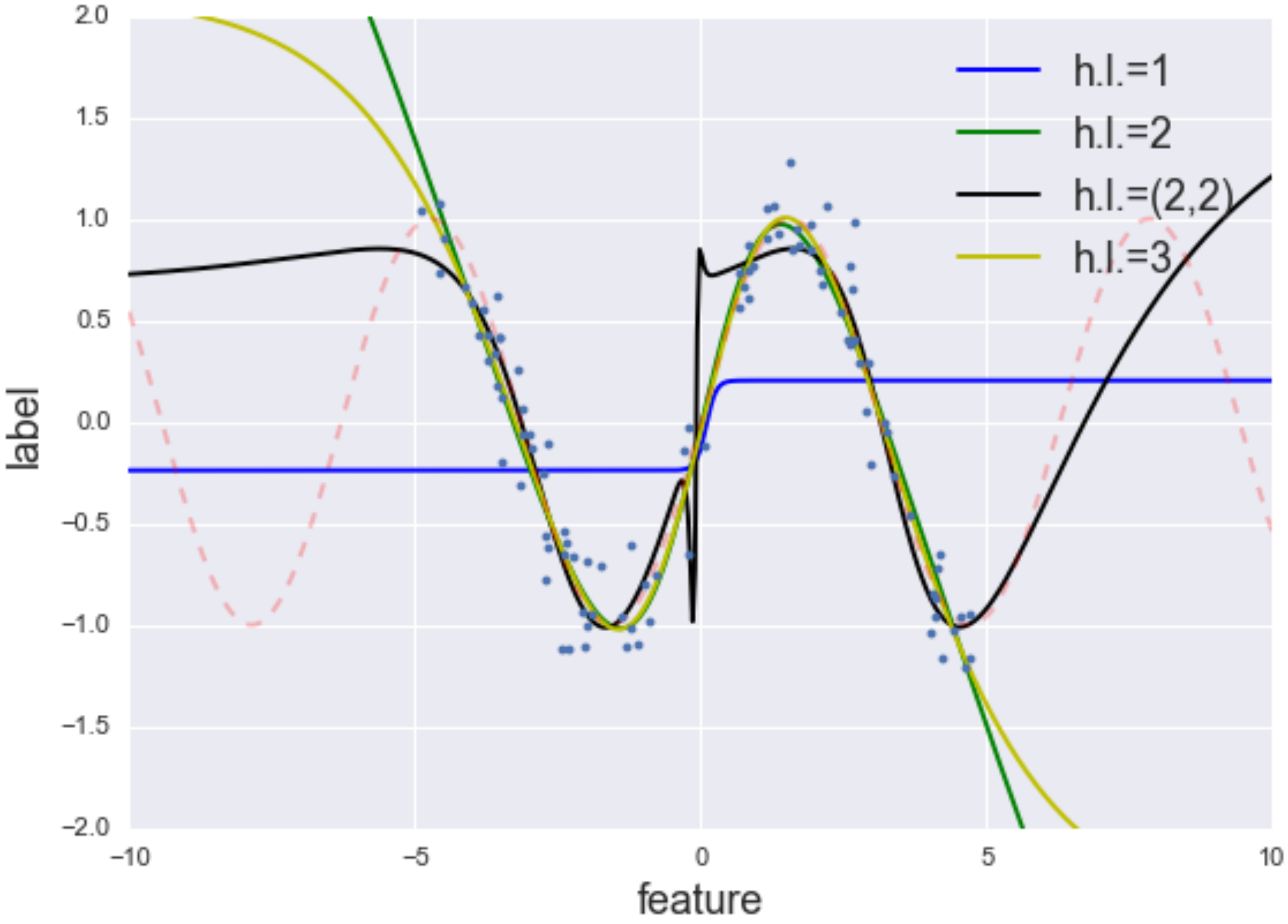
Regression example



Regression example



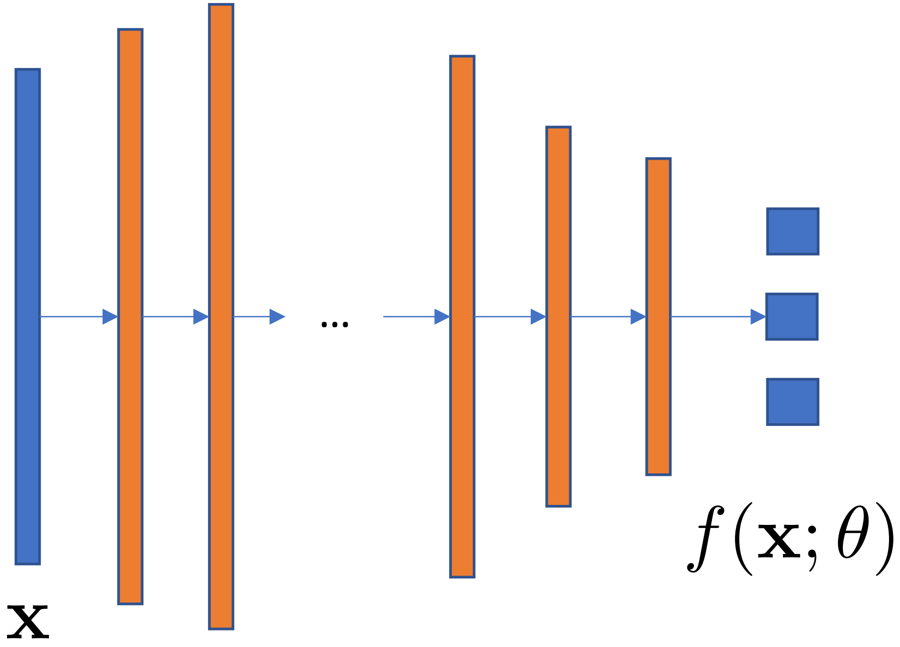
Can it extrapolate?



Notes on prediction and extrapolation

- Fully connected networks are very powerful generic models that can represent very complicated functions
- It is important to choose the network architecture correctly for the best prediction accuracy
- This is done on a validation set
- Models can interpolate between samples they have seen
- They do not extrapolate well – that requires knowledge about the underlying system

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*

Training of multilayered networks

- Training / validation / test set
- Cost functions
- Backpropagation
- Stochastic optimization
- Initialization
- Avoiding over-fitting

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

Training set

- **Main role:** Determining the best model parameters

$$\mathcal{D}_{\text{training}} = \{\mathbf{x}_n, \mathbf{y}_n\}$$

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

- Model parameters that minimize a cost for the training set
- Size of the training set is **important**. Larger number of training examples are needed for models with larger number of parameters.

Validation set

- **Main role:** Determining the hyper-parameters

$$\mathcal{D}_{\text{validation}} = \{\mathbf{x}_n, \mathbf{y}_n\}$$

- Hyper-parameters of a model
  - Architecture – number of layers, width of layers, non-linearities, …
  - Number of training iterations
  - Using / Not using regularization, regularization coefficients
  - Batch size used during training
- Ideally no overlap

$$\mathcal{D}_{\text{validation}} \cap \mathcal{D}_{\text{training}} = \emptyset$$

Test set

- **Main role:** Estimating model prediction accuracy

$$\mathcal{D}_{\text{test}} = \{\mathbf{x}_n, \mathbf{y}_n\}$$

- Absolutely no overlap

$$\mathcal{D}_{\text{test}} \cap \mathcal{D}_{\text{training}} = \emptyset, \;\; \mathcal{D}_{\text{test}} \cap \mathcal{D}_{\text{validation}} = \emptyset$$

- Model parameters / hyper-parameters should **not** be changed based on test accuracy
- Ideally should come from the same distribution as training and validation sets
- Variations in distributions can reduce prediction accuracy

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets

Notes

- Generalization accuracy, computed on the test set, will often be lower than the one on training set.
- Too much difference between test and training accuracy points out to “over-fitting”, where the model fits to noise in training set that is by chance correlated with labels
- For small datasets one often uses cross-validation, bootstrap resampling, jack knife resampling, …

Training of multilayered networks

- Training / validation / test set
- Cost functions
- Backpropagation
- Stochastic optimization
- Initialization
- Avoiding over-fitting

Cost functions - classification

- Depends on the task
- Important to choose an appropriate cost function
- Sums are over training samples

$$\mathcal{L} = \sum_{n=1}^{N} -y_n \ln f(\mathbf{x}_n;\theta) - (1 - y_n)\ln(1 - f(\mathbf{x}_n;\theta))$$

Binary classification

$$\mathcal{L} = \sum_{n=1}^{N} \sum_{k \in \mathcal{C}} -\mathbf{1}(y_n = k) \ln f_k(\mathbf{x}_n;\theta)$$

Multi-label classification

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions

Cost functions - regression

- Sums are over training samples

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(y_n - f(\mathbf{x}_n;\theta))^T(y_n - f(\mathbf{x}_n;\theta))$$

Mean Squared Error, L_2 loss

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{M}|y_{n,j} - f_j(\mathbf{x}_n;\theta)|$$

Mean Absolute Error, L_1 loss

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation

Training of multilayered networks

- Training / validation / test set
- Cost functions
- Backpropagation
- Stochastic optimization
- Initialization
- Avoiding over-fitting

Optimization

- Whether classification or regression

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

- The weights and biases of the network
- Even the simplest network may lead to complicated optimization surface, most likely non-convex
- Gradient based optimization – gradient descent (ascent if maximization)
- In its simplest form: $\theta_i^{t+1} = \theta_i^t - \eta \dfrac{\partial \mathcal{L}}{\partial \theta_i}$

Gradients via chain rule

$$\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}(f(\mathbf{x};\theta))}{\partial \theta_i} \qquad f(\mathbf{x};\theta) = f_L \circ \cdots \circ f_2 \circ f_1(\mathbf{x})$$

$$f_l = \sigma(a_l) = \sigma(\mathbf{W}_l f_{l-1} + b_l)$$

- Let's focus on $w_{ij,l}, b_{i,l}$, one of the weight and one bias in layer l, the partial derivatives are given as

$$\frac{\partial \mathcal{L}}{\partial w_{ij,l}} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}} \frac{\partial a_{l,i}}{\partial w_{ij,l}} \qquad \frac{\partial \mathcal{L}}{\partial b_{i,l}} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}} \frac{\partial a_{l,i}}{\partial b_{i,l}}$$

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation

Gradients via chain rule

$$\frac{\partial \mathcal{L}}{\partial w_{ij,l}} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}} \frac{\partial a_{l,i}}{\partial w_{ij,l}}$$

$$\frac{\partial f_{l,i}}{\partial a_{l,i}} = \sigma'(a_{l,i}) \qquad \frac{\partial a_{l,i}}{\partial b_{i,l}} = 1 \qquad \frac{\partial a_{l,i}}{\partial w_{ij,l}} = f_{l-1,j}$$

Let us define the error signal

$$\delta_{l,i} = \frac{\partial \mathcal{L}}{\partial a_{l,i}} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}}$$

Gradients with respect to the error signal are

$$\frac{\partial \mathcal{L}}{\partial w_{ij,l}} = \delta_{l,i} f_{l-1,j}$$

$$\frac{\partial \mathcal{L}}{\partial b_{i,l}} = \delta_{l,i}$$

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation

Error signal propagates backwards

$$\delta_{l,i} = \frac{\partial \mathcal{L}}{\partial f_{l,i}} \frac{\partial f_{l,i}}{\partial a_{l,i}} = \left(\sum_k \frac{\partial \mathcal{L}}{\partial f_{l+1,k}} \frac{\partial f_{l+1,k}}{\partial a_{l+1,k}} \frac{\partial a_{l+1,k}}{\partial f_{l,i}} \right) \frac{\partial f_{l,i}}{\partial a_{l,i}}$$

$$= \left(\sum_k \frac{\partial \mathcal{L}}{\partial f_{l+1,k}} \frac{\partial f_{l+1,k}}{\partial a_{l+1,k}} w_{ki,l+1} \right) \frac{\partial f_{l,i}}{\partial a_{l,i}}$$

$$= \left(\sum_k \delta_{k,l+1} w_{ki,l+1} \right) \frac{\partial f_{l,i}}{\partial a_{l,i}}$$

$$= \left(\sum_k \delta_{k,l+1} w_{ki,l+1} \right) \sigma'(a_{l,i})$$

Error signal at layer l is a function of error signal and the weights at layer l+1

Backpropagation

[Rumelhart, Hinton and Williams 1986, Nature]

We start at the very final layer L

$$\delta_{L,i} = \frac{\partial \mathcal{L}}{\partial f_{L,i}} \sigma'(a_{L,i}), \; \frac{\partial \mathcal{L}}{\partial w_{ij,L}} = \delta_{L,i} f_{L-1,j}, \; \frac{\partial \mathcal{L}}{\partial b_{i,L}} = \delta_{L,i}$$

Propagate errors backwards and compute gradients

$$\delta_{l,i} = \left(\sum_k \delta_{l+1,k} w_{ki,l+1} \right) \sigma'(a_{l,i})$$

$$\frac{\partial \mathcal{L}}{\partial w_{ij,l}} = \delta_{l,i} f_{l-1,j} \qquad \frac{\partial \mathcal{L}}{\partial b_{i,l}} = \delta_{l,i}$$

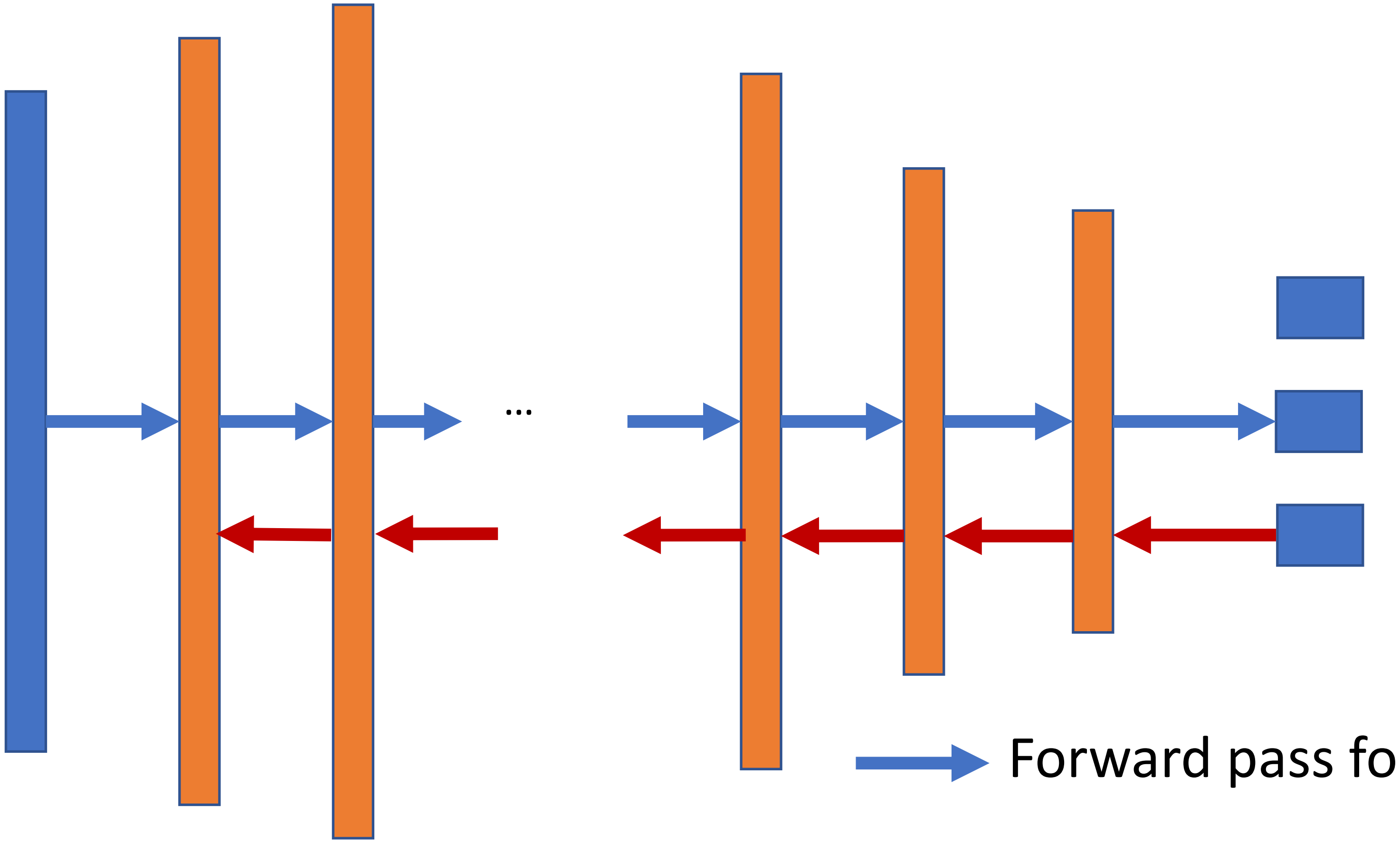
A matrix form for the error signals

$$\delta_l = \underbrace{\begin{pmatrix} \delta_{l+1,1}, \ldots, \delta_{l+1,d_{l+1}} \end{pmatrix}}_{\substack{\dfrac{\partial \mathcal{L}}{\partial a_{l+1}} \\ \text{Row vector}}} \mathbf{W_{l+1}} \underbrace{\begin{pmatrix} \sigma'(a_{l,1}) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma'(a_{l,d_l}) \end{pmatrix}}_{\text{Diagonal matrix}}$$

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation

Error information flowing backwards for training



→ Forward pass for prediction

← Backward pass for computing gradients

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation

Training of multilayered networks

- Training / validation / test set
- Cost functions
- Backpropagation
- Stochastic optimization
- Initialization
- Avoiding over-fitting

Direct optimization can be slow

- Training with very large number of samples

$$\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}}{\partial \theta_i}|_{\theta_i^t} = \theta_i^t - \eta \sum_n \frac{\partial \mathcal{L}_n}{\partial \theta_i}|_{\theta_i^t}$$

- Forward and backward pass required for each sample for each update step
- This would be very slow as the number of samples reach even thousands of images.

Stochastic gradient descent (SGD)

- Instead at every step a random batch of training samples are chosen, and parameters are updated accordingly
- A random batch of training samples are chosen at each update step

$$\mathcal{B} \subset \mathcal{D}_{\text{training}}$$

- Gradient with respect to parameters are approximated

$$\sum_{n} \frac{\partial \mathcal{L}_n}{\partial \theta_i}\Big|_{\theta_i^t} \approx \sum_{n_m \in \mathcal{B}} \frac{\partial \mathcal{L}_{n_m}}{\partial \theta_i}\Big|_{\theta_i^t}$$

$$\theta_i^{t+1} = \theta_i^t - \eta \sum_{n_m \in \mathcal{B}} \frac{\partial \mathcal{L}_{n_m}}{\partial \theta_i}$$

Notes on optimization techniques

- Learning rate is crucial - not too low, not too high – often set by empirical tests on validation set
- Many variations present, including momentum and second order gradient approximations
  - RMSProp [Hinton]
  - Adagrad [Duchi et al. 2011]
  - Adam [Ba and Kingma 2014]
  - …

Basic update model with momentum

$$\Delta\theta_i = \alpha\Delta\theta_i - (1-\alpha)\eta\frac{\partial\mathcal{L}}{\partial\theta_i}$$

$$\theta_i^{t+1} = \theta_i^t + \Delta\theta_i$$

- Different algorithms have different convergence properties, e.g.
  - Adam works well out of the box with minimal parameter tuning
  - SGD may lead to better results but needs more tuning [Wilson et al. 2017, NIPS]

Training of multilayered networks

- Training / validation / test set
- Cost functions
- Backpropagation
- Stochastic optimization
- Initialization
- Avoiding over-fitting

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Initialization

Initialization

$$\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}}{\partial \theta_i}|_{\theta_i^t} = \theta_i^t - \eta \sum_n \frac{\partial \mathcal{L}_n}{\partial \theta_i}|_{\theta_i^t}$$

- The optimization need to start from somewhere

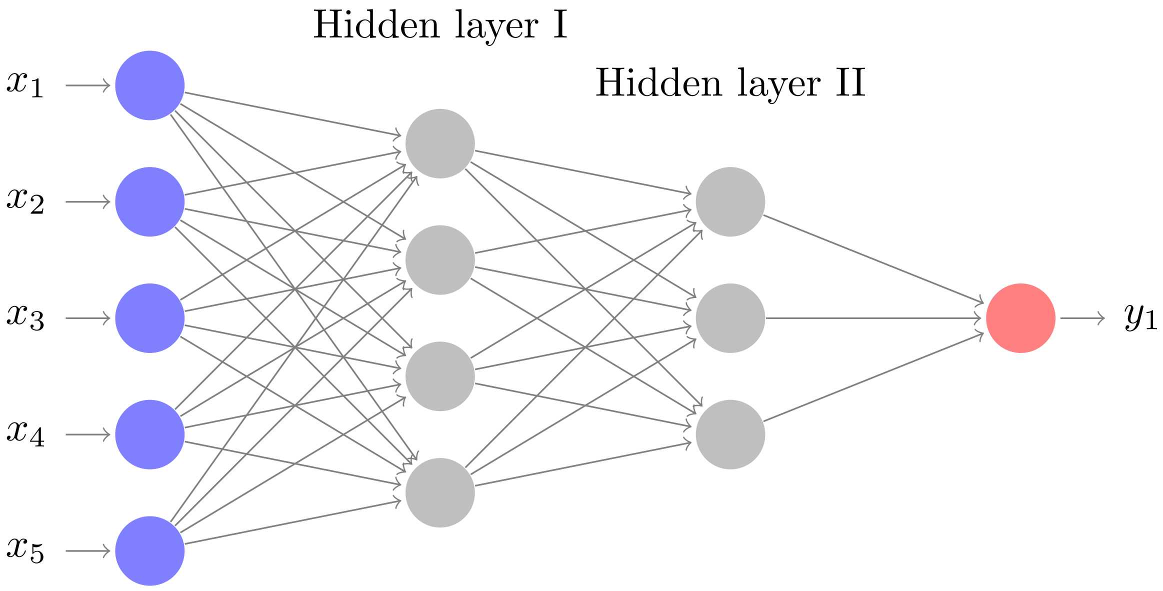
$$w_{ij,l}^0 =?,\ b_{i,l}^0 =?$$

- It turns out that the initialization is quite important

Very Naïve Initialization

- Zero (constant) initialization

$$w_{ij,l}^{0} = 0,\ b_{i,l}^{0} = 0$$



Very Naïve Initialization

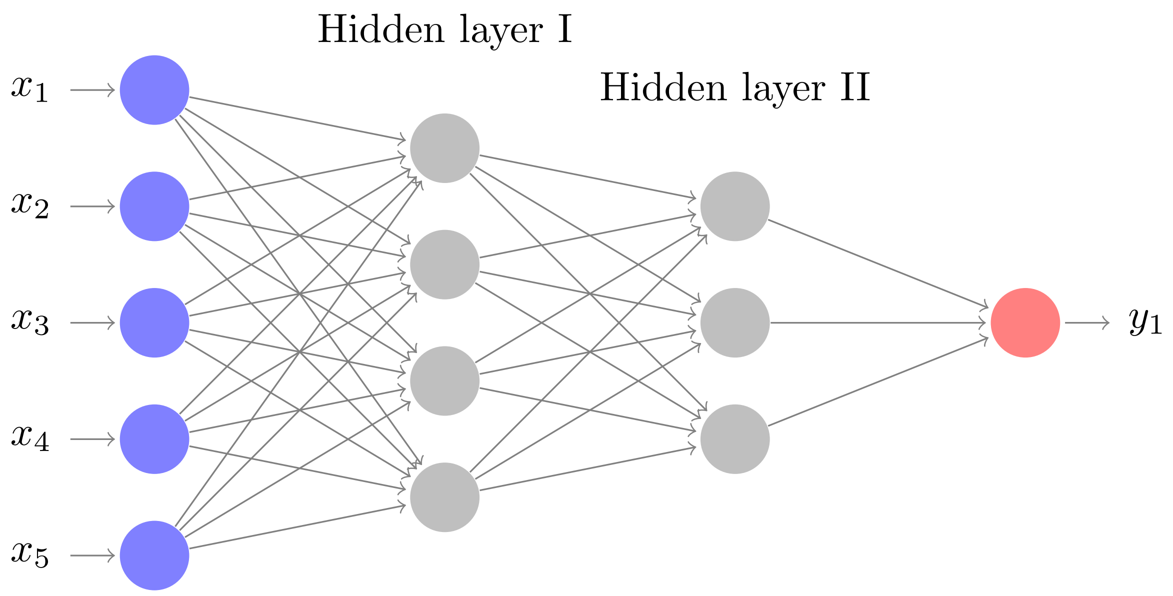
- Zero (constant) initialization

$$w^0_{ij,l} = 0,\ b^0_{i,l} = 0$$

- Initialization with constant weights is problematic
- Complete symmetry in the network
- No matter what the input is the values of the hidden units will be the same
- All the weights in the one layer will get the same updates
- Effectively, one neuron thick networks
- Initializing bias with 0 is used commonly – does not cause symmetry problems

Random initialization

- Initialize weights with random values biases with zeros



- Not symmetric any more
- Different weights will get different updates
- Used quite often with random samples from uniform or normal

Other heuristics

- Random initialization is the common approach
- The main question is what distribution should one use
- Normal and uniform are the obvious choices but with what standard deviation?
- [Glorot and Bengio 2010]

$$w_{ij,l} \propto \mathcal{U}\left[-\frac{1}{\sqrt{d_{l-1}}}, \frac{1}{\sqrt{d_{l-1}}}\right]$$

- [He et al. 2015]

$$w_{ij,l} \propto \mathcal{N}\left(0, \sqrt{\frac{2}{d_{l-1}}}\right) \qquad w_{ij,l} \propto \mathcal{N}\left(0, \sqrt{\frac{2}{d_{l+1}}}\right)$$

- Both considering that variance of a signal at the input and output of a layer should be similar

Training of multilayered networks

- Training / validation / test set
- Cost functions
- Backpropagation
- Stochastic optimization
- Initialization
- Avoiding over-fitting

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Intialization
Over-fitting

Number of parameters in a network

- Let us assume the following network

$$\underbrace{10}_{\text{Input dimesion}} \rightarrow \underbrace{20 \rightarrow 30 \rightarrow 10}_{\text{Hidden dimensions}} \rightarrow \underbrace{1}_{\text{Output dimension}}$$

- Total number of parameters:
  10x20 + 20 + 20x30 + 30 + 30x10 + 10 + 10 = 1170
- To determine the large number of parameters we need large number of samples
- Modern networks have many number of parameter, reaching millions

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Intialization
Over-fitting

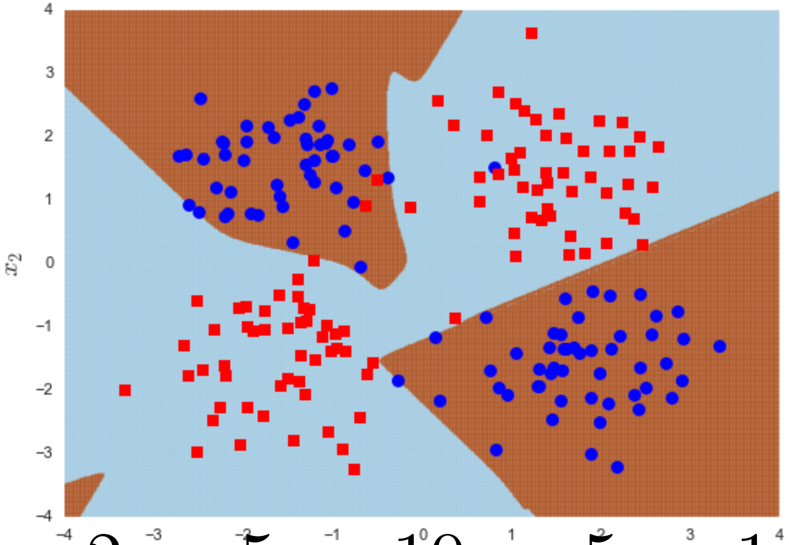
Over-fitting

- When the model has too many parameters and not enough training samples
- Model can learn noise in the training samples
- Perfect prediction on training data
- Bad prediction in validation data
- Will not be able to generalize

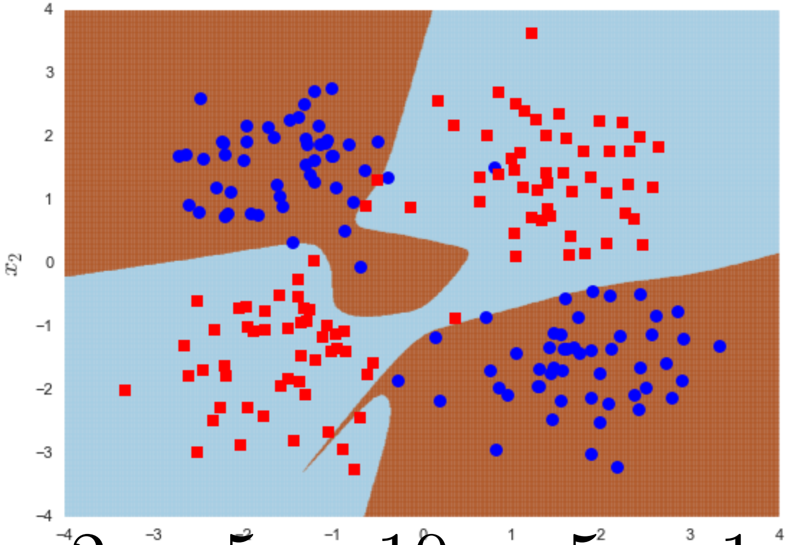
Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Intialization
Over-fitting

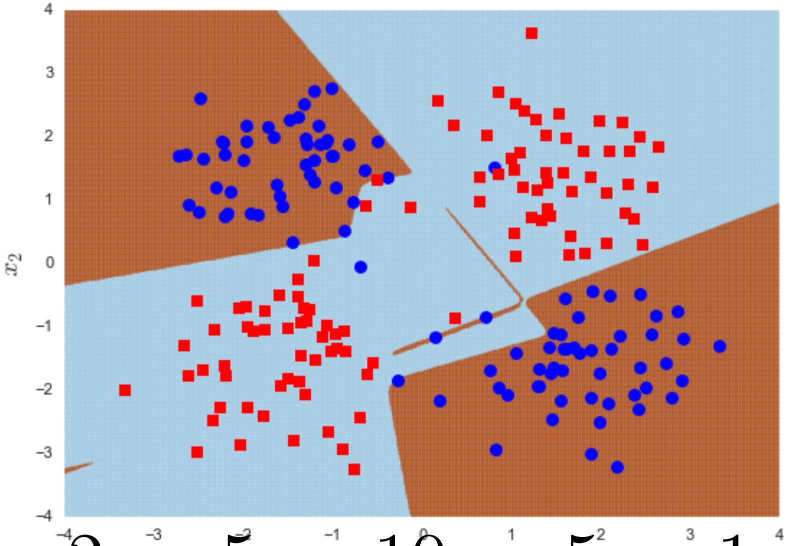
Over-fitting – examples on the toy data



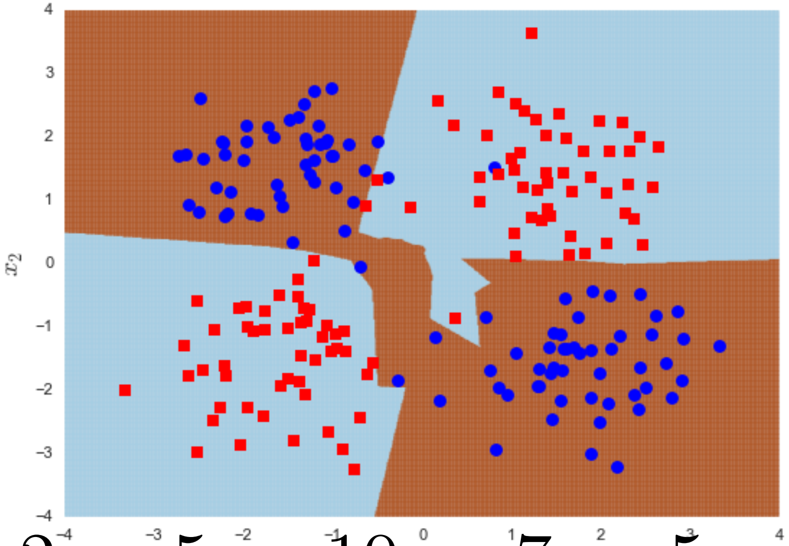
$2 \rightarrow 5 \rightarrow 10 \rightarrow 5 \rightarrow 1$



$2 \rightarrow 5 \rightarrow 10 \rightarrow 5 \rightarrow 1$



$2 \rightarrow 5 \rightarrow 10 \rightarrow 5 \rightarrow 1$



$2 \rightarrow 5 \rightarrow 10 \rightarrow 7 \rightarrow 5 \rightarrow 1$

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Intialization
Over-fitting

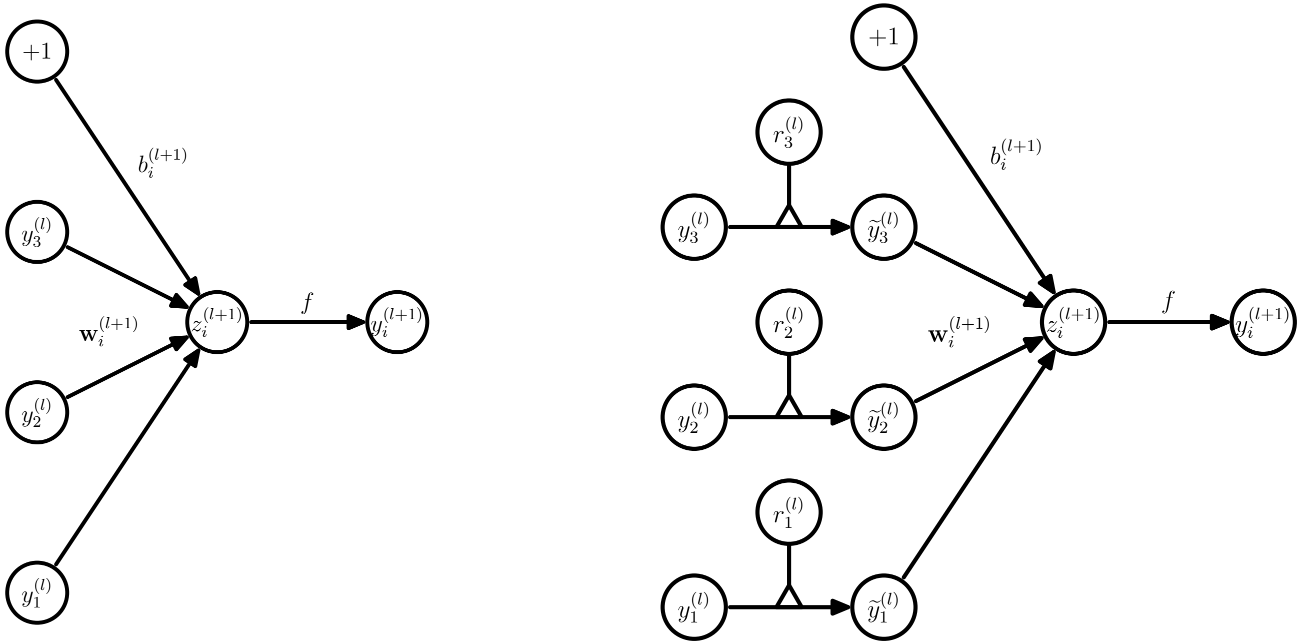
Over-fitting – how to overcome it?

- **Best strategy is to have more data**
- Reduce the size of your network, i.e. less parameters
- Regularization [Krogh and Hertz, NIPS 1992]
- Similar strategy is taken for many other learning algorithms, ridge regression, SVM, sparse regression,…

$$\min \mathcal{L} + \lambda \mathcal{R}\left(\theta\right) \qquad \mathcal{R}(\theta) = \frac{1}{2} \sum \theta_i^2 \qquad \mathcal{R}(\theta) = \sum |\theta_i|$$

Drop-out

- Drop-out [Hinton et al. 2012, Srivastava, Hinton, Krizhevsky, JMLR 2014] [Figure from latter]
- Randomly set some of the activations to zero during training



(a) Standard network

(b) Dropout network

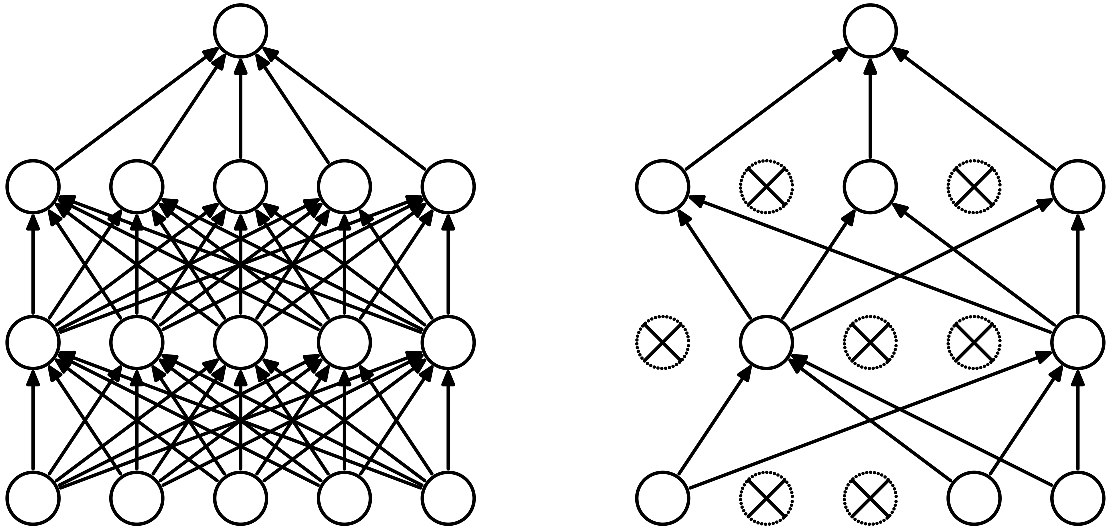
Figure 3: Comparison of the basic operations of a standard and dropout network.

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Intialization
Over-fitting

Drop-out

- [Figure from Srivastava, Hinton, Krizhevsky, JMLR 2014]
- Network builds redundancies, need to create multiple paths



(a) Standard Neural Net

(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Training on another task with more data

- There may be tasks with more data
- Unsupervised pre-training
- Initial supervised training with other related tasks and initializing the network weights with other network's values – transfer learning
- We will talk about this more in CNN lectures

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Intialization
Over-fitting

Other solutions that empirically help

- Data augmentation – we’ll talk about that more
- Adding random noise to the weight / data [An, 1996; Kim, Giles and Horne, 1996; application in Graves, Mohamed and Hinton, 2013]

Not necessarily for over-fitting but helps convergence:

- Batch-normalization [Ioffe and Szegedy 2015]
- Layer normalization [Ba, Kiros and Hinton 2017]

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Intialization
Over-fitting

Tools

- Python is arguably the dominant programming language
- Tensorflow
- PyTorch
- A lot of fantastic tools for Matlab also exist
- For C++ lovers – Caffe
- In the exercises you will continue using python

Computer Vision

Introduction
Learning basics
Math. basics
Perceptron model
MLP
Training
Three sets
Cost functions
Backpropagation
Intialization
Over-fitting

MLP - Analogy to brain