HANDS-ON AI II

Unit 1: Recap of Hands-on AI I

Rainer Dangl

Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

Copyright Statement

Content of Unit 1

- **Recap data types**
  - Tabular data
  - Images
  - Sequences
- **Recap supervised ML**
  - Workflow
  - Training
  - Generalization
- **Recap NNs and CNNs**
  - Optimization
  - PyTorch
  - Concepts of neural networks
  - Concepts of convolutional neural networks

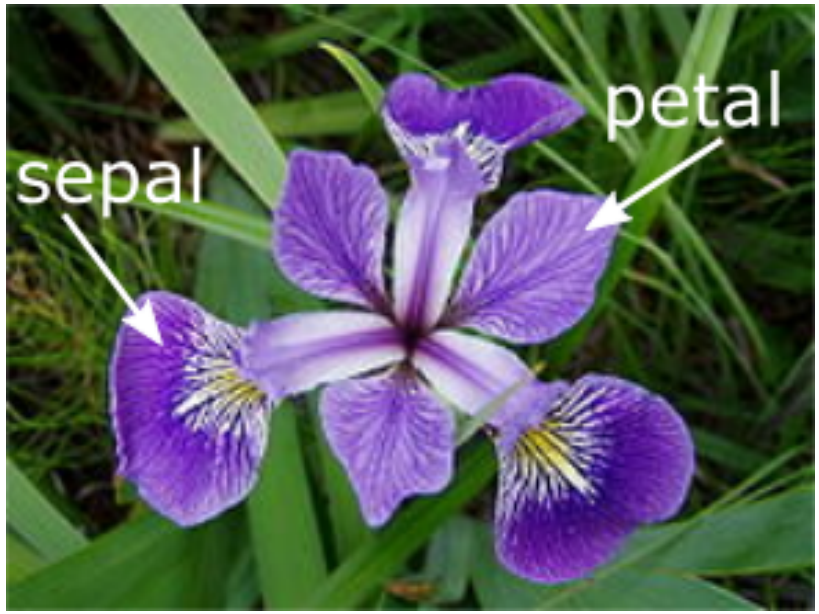
Tabular Data

- Data is structured in a tabular form.
- Data elements are arranged in vertical columns and horizontal rows.
- Each column and row is uniquely numbered.
- Tabular data has a virtually infinite range for mass data storage (can always add rows).
- Tabular databases include the following key properties:
  - Share the same set of properties per record, i.e., every row has the same column titles.
  - Each column is (usually) assigned with a header title (metadata).
  - Access through identifiers, i.e., each object can be retrieved by a query through key values.

Example: Iris Data Set

We have the following $d = 4$ **features**:

- Sepal length in cm
- Sepal width in cm
- Petal length in cm
- Petal width in cm



Terminology

sep-len	sep-width	pet-len	pet-width	species
6.7	3.1	4.7	1.5	versicolor
6.7	3.1	4.4	1.4	versicolor
6.5	3.2	5.1	2.0	virginica
5.0	3.6	1.4	0.2	setosa
6.5	3.0	5.8	2.2	virginica
...	...	...	...	...

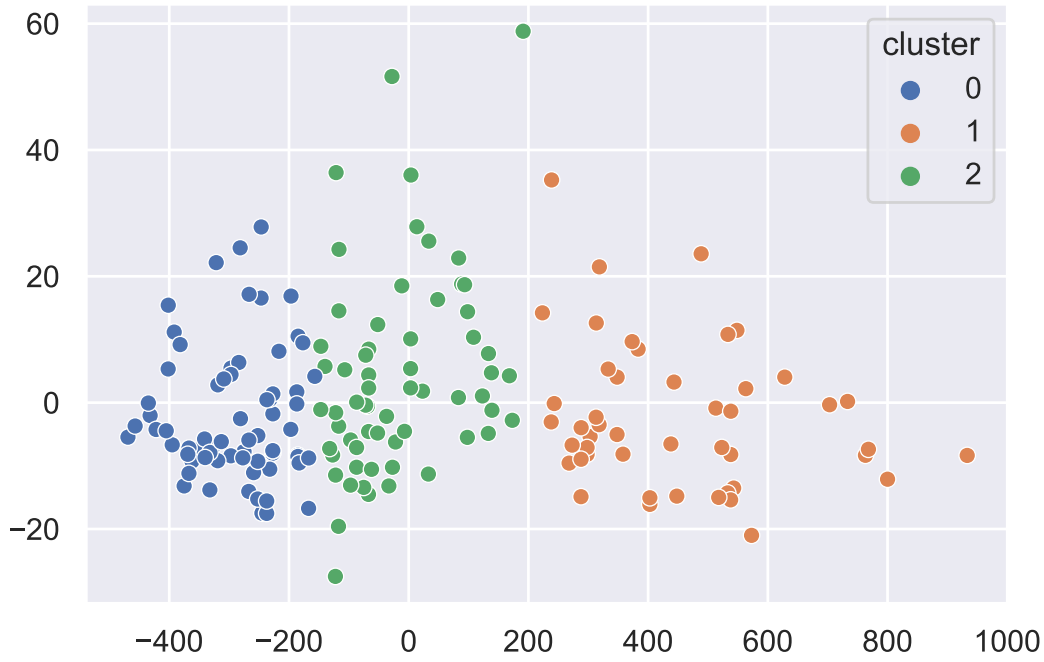
- Every flower entry is referred to as a **sample**.
- Every sample is described by 4 **features** (sep-len, sep-width, pet-len, pet-width), which can be represented as a **feature vector**, e.g.: $\boldsymbol{x} = (6.7, 3.1, 4.7, 1.5)$.
- There are 3 species (setosa, virginica, versicolor), which means that there are 3 **classes**.
- Every sample lists the species/class via its **label**, e.g.: $y =$ versicolor.

Dimensionality Reduction

- Problem: Too many features to see anything in the data.
- Often, data is described with hundreds (or thousands) of features $\rightarrow$ visualization is a common problem.
- Idea: **Reduce dimensionality** of the data set, while still preserving as much information as possible.
- Popular algorithms are PCA (principal component analysis) or t-SNE (t-distributed stochastic neighbor embedding).
- Can reduce n-dimensional data to, e.g., 2-dimensional data $\rightarrow$ can be easily visualized.

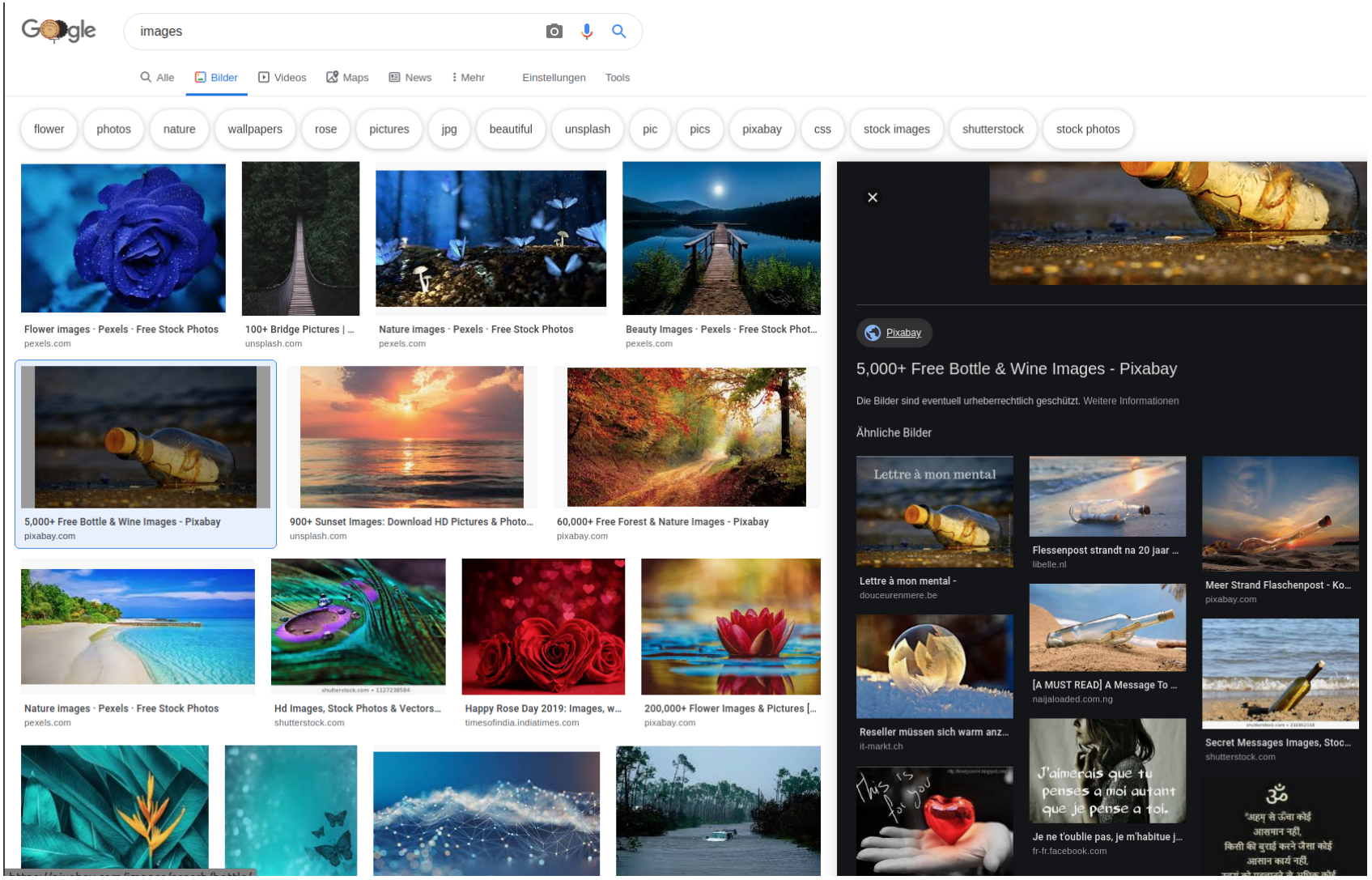
Clustering Algorithms

- So far, all our data was labeled.
- Imagine now that the data is unlabeled and we still want to find out which data belongs together.
- We can now use so-called **clustering algorithms** that try to group samples into “similar” and “dissimilar” samples.[1]



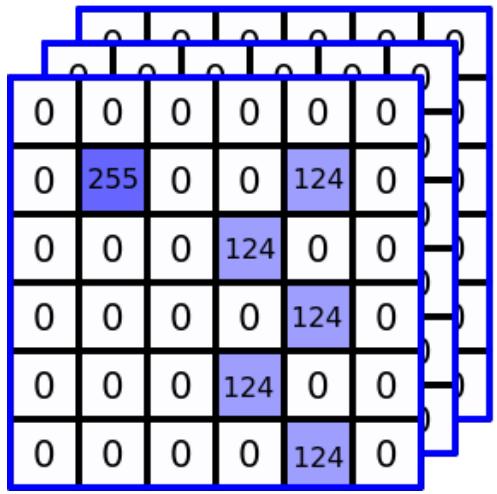
[1] What is considered “similar” highly depends on the algorithm.

Images


images
Alle
Bilder
Videos
Maps
News
Mehr
Einstellungen
Tools
flower
photos
nature
wallpapers
rose
pictures
jpg
beautiful
unsplash
pic
pics
pixabay
css
stock images
shutterstock
stock photos
Flower images · Pexels · Free Stock Photos
pexels.com
100+ Bridge Pictures | ...
unsplash.com
Nature images · Pexels · Free Stock Photos
pexels.com
Beauty Images · Pexels · Free Stock Phot...
pexels.com
5,000+ Free Bottle & Wine Images - Pixabay
pixabay.com
900+ Sunset Images: Download HD Pictures & Photo...
unsplash.com
60,000+ Free Forest & Nature Images - Pixabay
pixabay.com
Nature images · Pexels · Free Stock Photos
pexels.com
Hd Images, Stock Photos & Vectors...
shutterstock.com
Happy Rose Day 2019: Images, w...
timesofindia.indiatimes.com
200,000+ Flower Images & Pictures [...
pixabay.com
Pixabay
5,000+ Free Bottle & Wine Images - Pixabay
Die Bilder sind eventuell urheberrechtlich geschützt. Weitere Informationen
Ähnliche Bilder
Lettre à mon mental
Lettre à mon mental -
douceurenmere.be
Flessenpost strandt na 20 jaar ...
libelle.nl
Meer Strand Flaschenpost - Ko...
pixabay.com
Reseller müssen sich warm anz...
it-markt.ch
[A MUST READ] A Message To ...
naijaloaded.com.ng
Secret Messages Images, Stoc...
shutterstock.com
This is for you
J'aimerais que tu penses à moi autant que je pense à toi.
Je ne t'oublie pas, je m'habitue j...
fr-fr.facebook.com
ॐ
"अहम् से ऊँचा कोई आसमान नहीं,
किसी को बुराई करने जैसा कोई आसान कार्य नहीं.

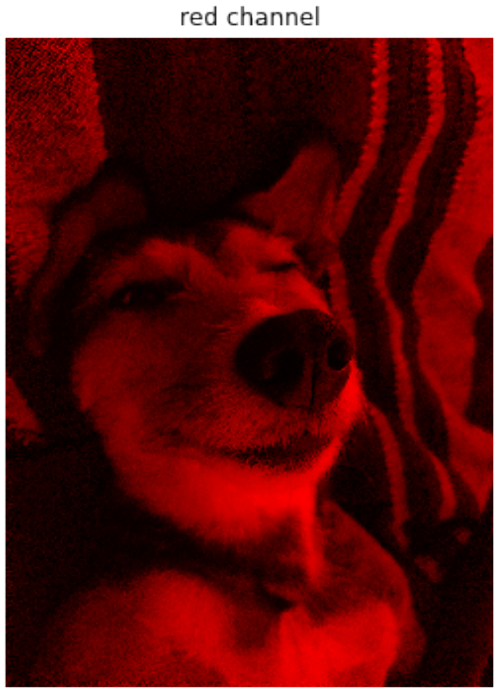
Representation of Images

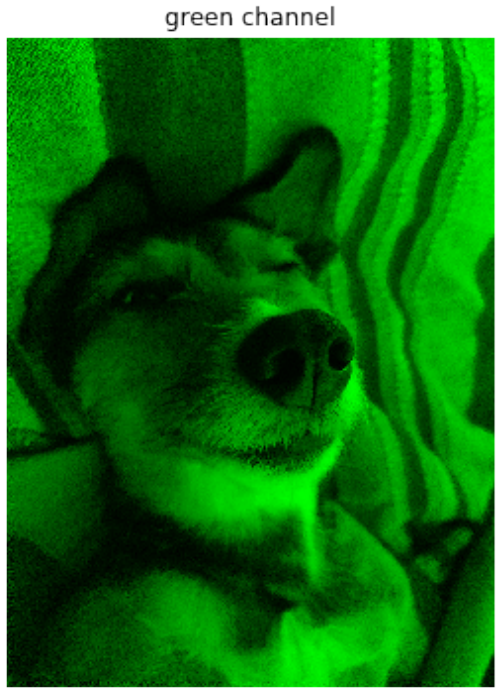
- Images are usually represented in **3 dimensions**:
  - Height
  - Length (or width)
  - Channels (often: red, green, blue)
- **Color depth** refers to the number of possible values for each channel of a pixel.
  - Color depths of **8-bit** ($2^8 = 256$ values) and **16-bit** ($2^{16} = 65536$ values) are common.
  - Higher values mean increased intensity.

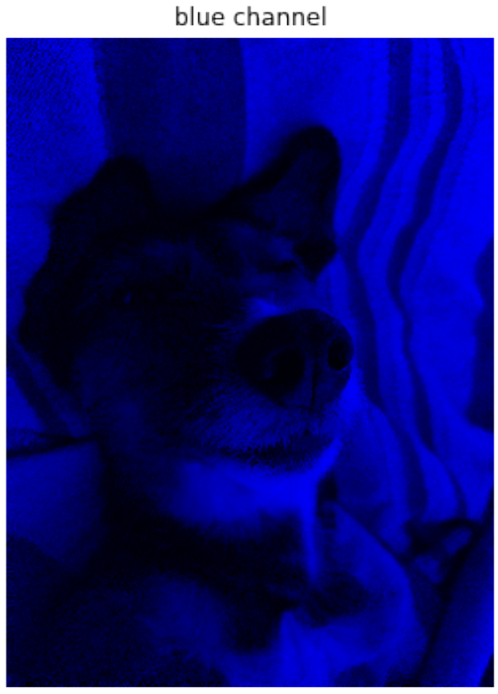


RGB Model

- The RGB (red, green, blue) model is the most important model for colored images.
- 3 channels: **red, green, blue**
- Adding up the 3 color channels results in the final image.







RGB Model

- The RGB (red, green, blue) model is the most important model for colored images.
- 3 channels: **red, green, blue**
- Adding up the 3 color channels results in the final image.

Data Augmentation

- In many cases, we can augment data artificially without collecting new real samples: Create “new” **artificial samples** by modifying existing samples.
- Pros:
  - Can increase the number of data points by a large factor with little effort (often on-the-fly).
  - Reduces overfitting, increases robustness of model.
- Cons:
  - Can introduce artifacts or change the task.
  - Heavily dependent on data, model and task.

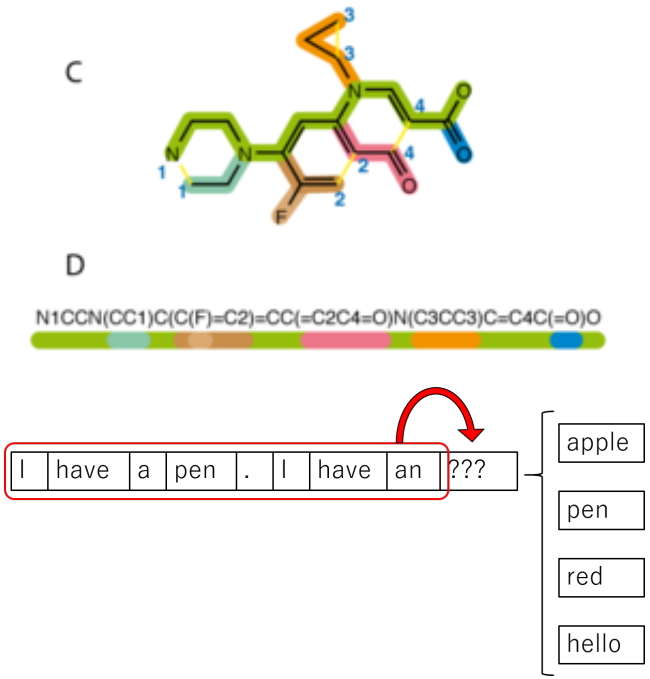
Data Augmentation Techniques

- There exist a lot of data augmentation **techniques**.
- Some can be applied across different fields (e.g., adding noise), some are specific and only applicable in certain scenarios.
- Here are some common examples in the area of **image augmentation**:
  - Rotation
  - Flipping
  - Zooming/Cropping
  - Blurring
  - Noise
  - Input Dropout
  - Distortion Effects
  - Color Jittering
  - ...

Sequences

- Sequence: $a_1, a_2, a_3, a_4, \ldots, a_n$
- Can every data set be represented as a sequence? Yes!
- Does it always make sense? No!
- Sequences can be useful when the data can be ordered in a useful way. What is useful depends on task ...

Sequences



- **Time series**: Ordered in time, e.g., weather, finance, pandemics $\rightarrow$ forecasting, classification
- Can also be order along other parameters, e.g., position:
  - **Molecule representations** in automated drug design
  - Symbol and word order in **languages**

Introduction to Machine Learning

Supervised Machine Learning

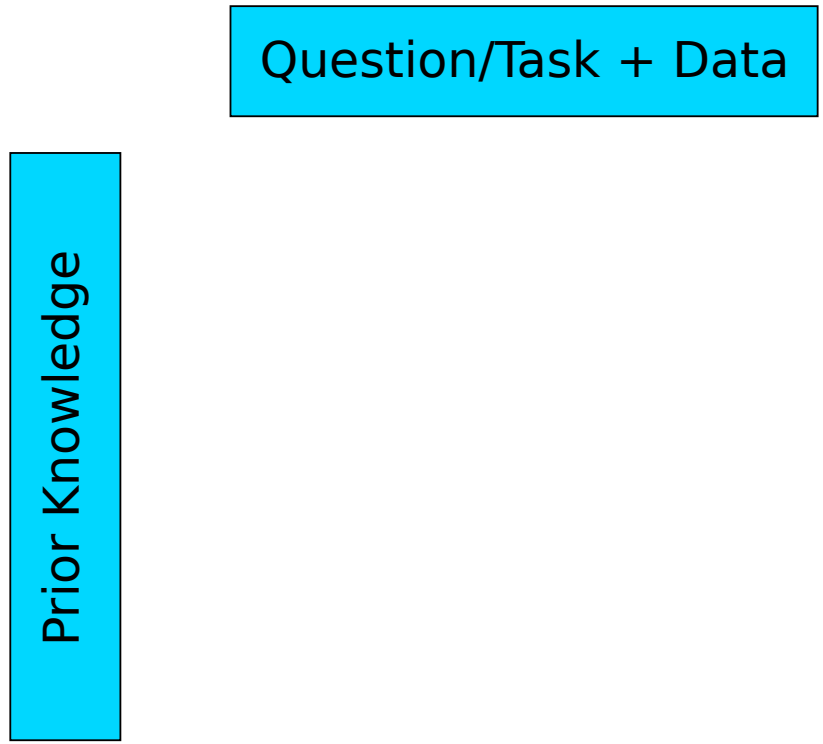
Supervised Machine Learning:

- Learning from **input values** and corresponding **target values**:
  - E.g., image + object type, DNA sequence + phenotype, ...
- Typical usage: **predictive modeling**
  - **Train** model on data set with input + target values.
  - Use trained model to **predict** target values for other (new) inputs where the targets are not known yet.
- **Classification**: target value is class label
- **Regression**: target value is numerical value

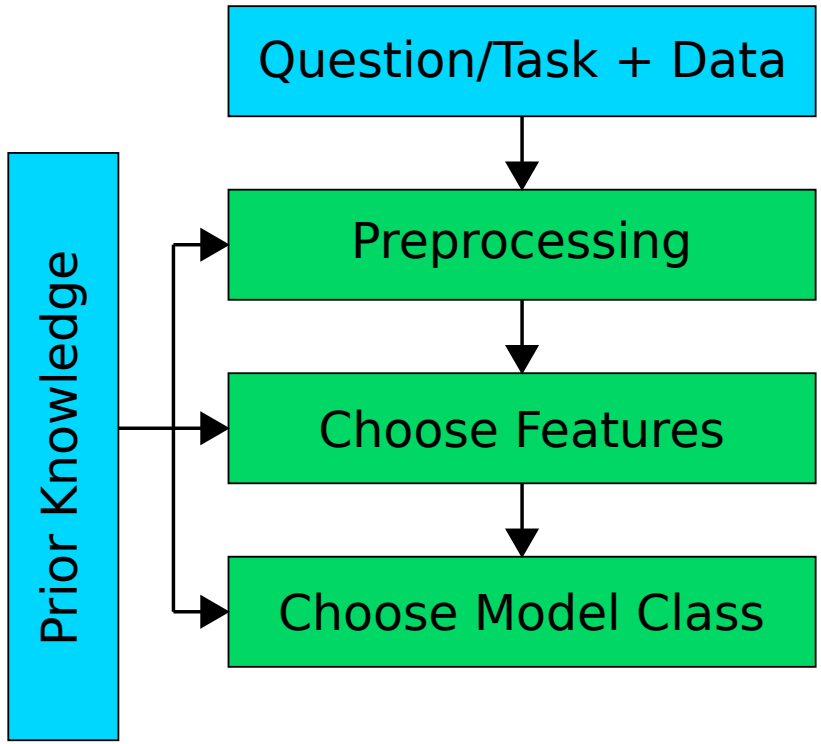
Terminology

- **Model**: parameterized function/method with specific parameter values (e.g., a trained neural network)
- **Model class**: the class of models in which we search for the model (e.g., neural networks, SVMs, . . .)
- **Parameters**: what is adjusted during training (e.g., network weights)
- **Hyperparameters**: settings controlling model complexity or the training procedure (e.g., network learning rate)
- **Model selection/training**: process of finding a model (optimal parameters) from the model class

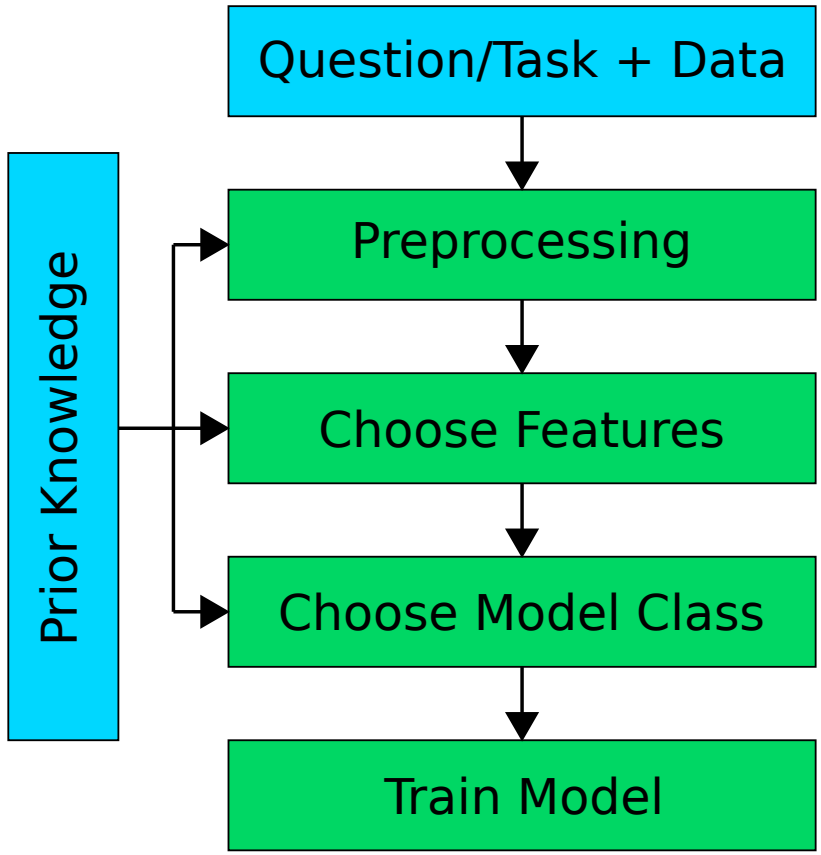
Basic Data Analysis Workflow



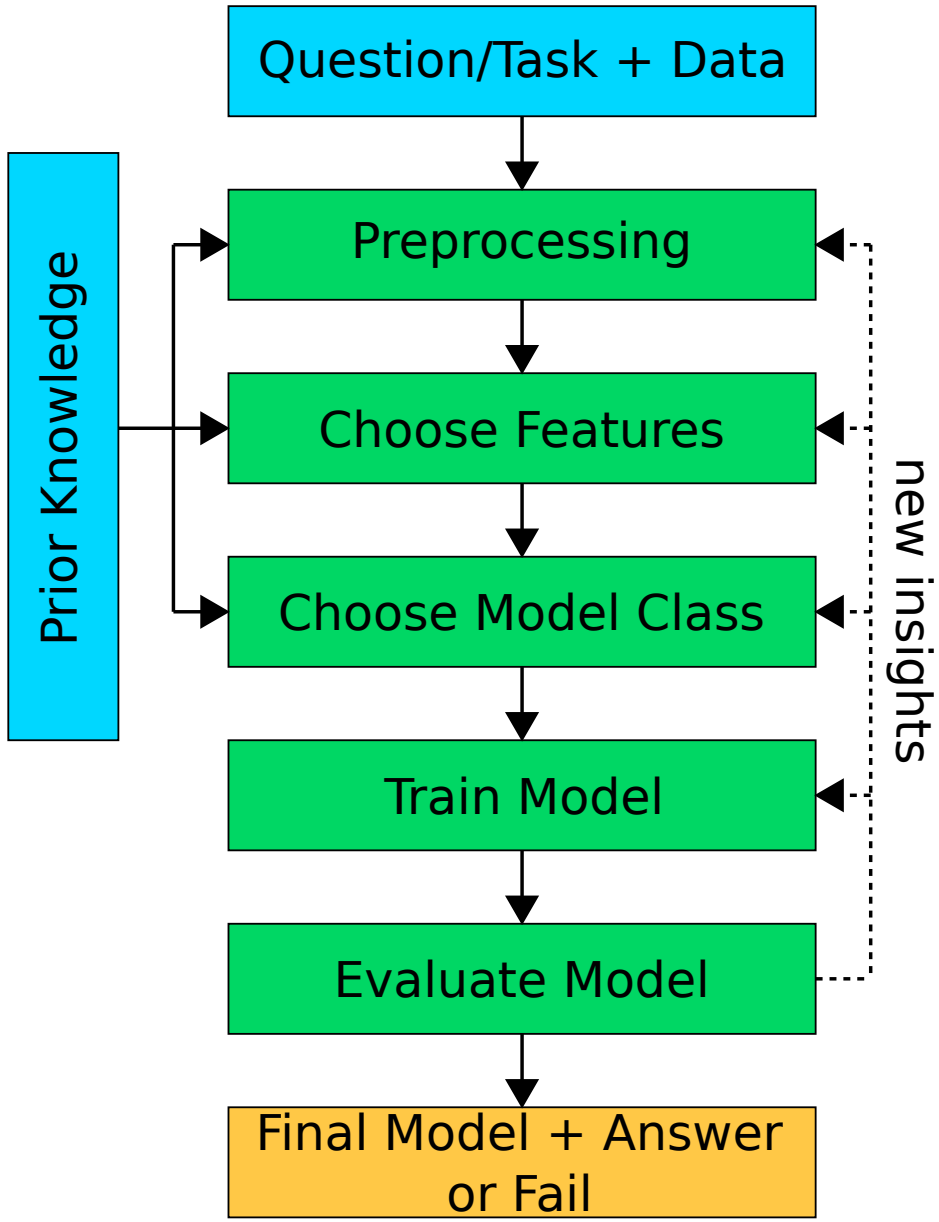
Basic Data Analysis Workflow



Basic Data Analysis Workflow



Basic Data Analysis Workflow



Introductory Example: Fish Recognition

- Example from

  R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification*. 2nd edition. John Wiley & Sons, 2001. ISBN 0-471-05669-3.*

- Automated system to sort fish in a fish-packing company: salmons must be distinguished from sea bass optically.
- **Given**: a set of pictures with fish labels.
- **Goal**: distinguish between salmons and sea bass.

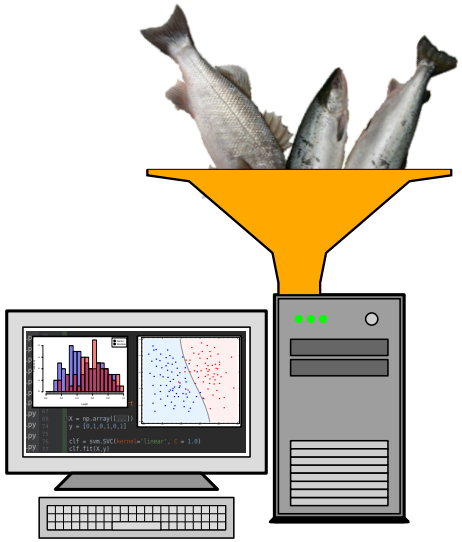
→ **Classification** task with two labels (salmon vs. sea bass, or, alternatively, salmon vs. not salmon)

Our Data (Two Sample Images)

Salmon:

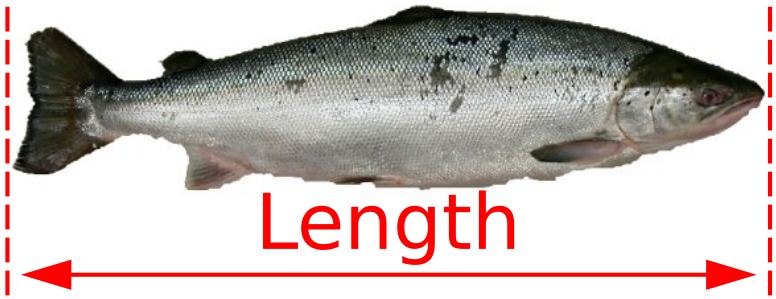
Sea bass:

How can we distinguish these two kinds of fish?



Back to Our Data

Salmon:



Sea bass:

- Assume we use **length** and **brightness** as features.
- For simplicity, also assume that some person extracted these features for us.

$\rightarrow$ How do we express/represent these features?

Input Representation

- We can represent an object by a vector $\boldsymbol{x}$ of feature values (=**feature vector**) of length d and **label** y:

$$\boldsymbol{x} = (x_1, \ldots, x_d) \qquad y$$

  - Fish example: A fish is represented as feature vector with two values *length* and *brightness* (i.e., $d = 2$) and one label ($y =$ "salmon" or $y =$ "sea bass").

- An object described by one feature vector and one label is referred to as **sample**: $(\boldsymbol{x}, y)$.

Input Representation

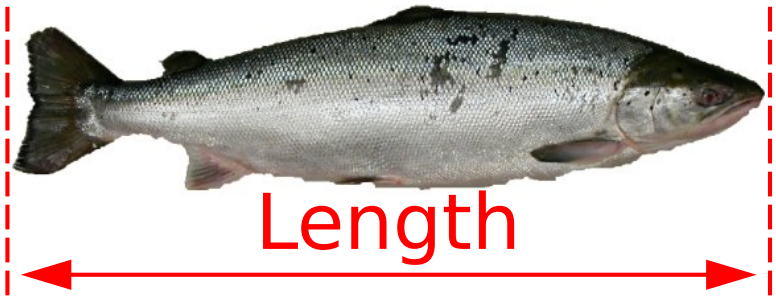
- Assume our data set consists of n objects with feature vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of length d, and each object has a corresponding label $y_1, \ldots, y_n$.
- Then, we can write the feature vectors of all objects in a **matrix of feature vectors** $\boldsymbol{X}$ and the labels in a corresponding labels vector $\boldsymbol{y}$:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

- Our labeled data is thus described by: $(\boldsymbol{X}, \boldsymbol{y})$.

Back to Our Data

Salmon:

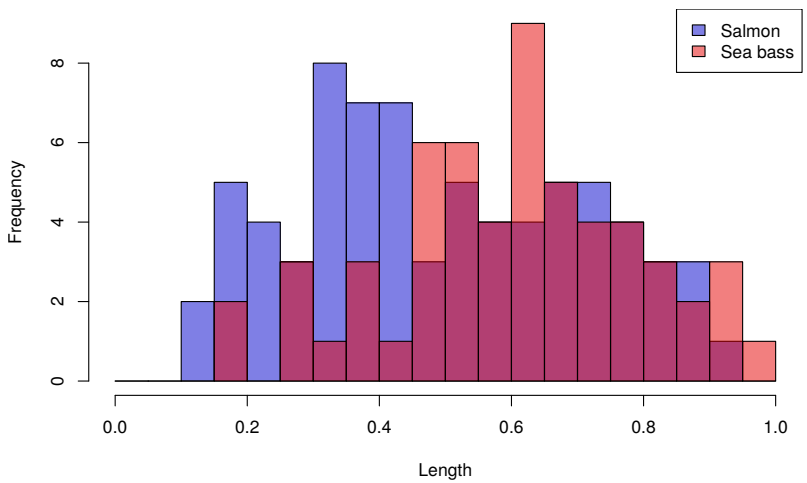


Sea bass:

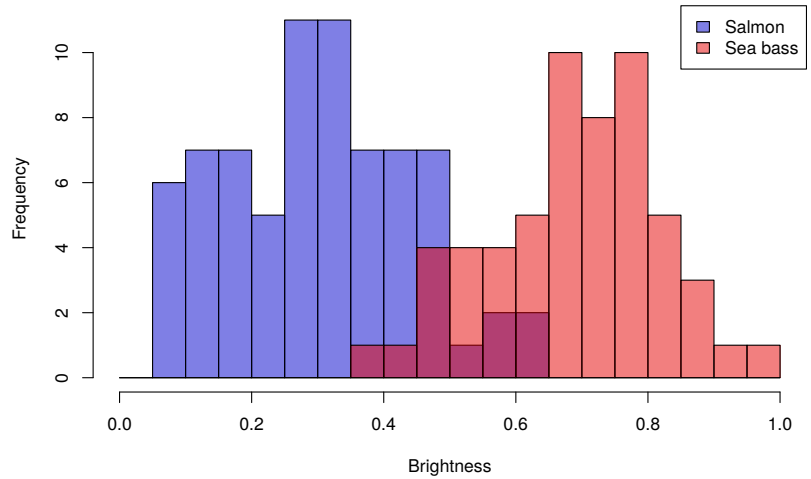
- We now know how to represent our data (i.e., using **features** and **labels**) and will take a look at it via histograms.

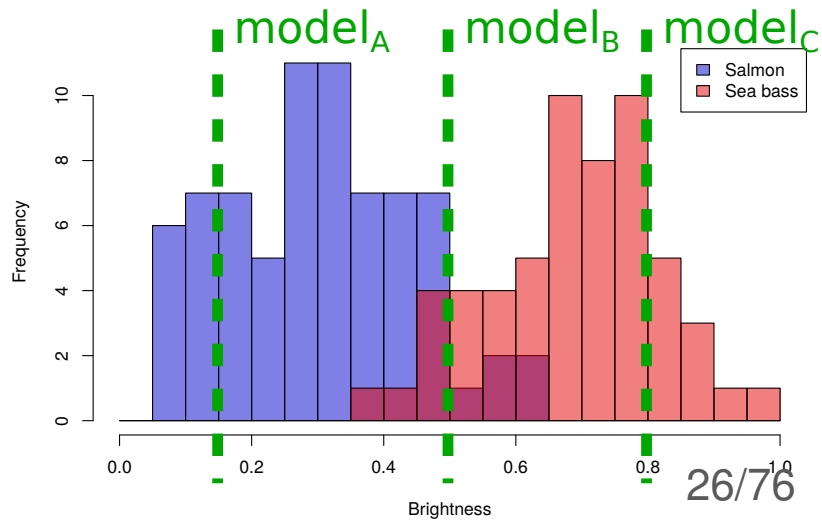
Back to Our Data

Length:



Brightness:





Scoring Our Models: Loss Function

- Assume we have a model g, parameterized by $\boldsymbol{w}$.
- $g(\boldsymbol{x}; \boldsymbol{w})$ maps an input vector $\boldsymbol{x}$ to an output value $\hat{y}$.
- We want $\hat{y}$ to be as close as possible to the true target value y.
- We can use a **loss function**

$$L(y, g(\boldsymbol{x}; \boldsymbol{w})) = L(y, \hat{y})$$

  to measure how close our prediction is to the true target for a given sample with $(\boldsymbol{x}, y)$.
- **The smaller the loss/cost, the better our prediction**.

Generalization Error/Risk and ERM

- The **generalization error** or **risk** is the expected loss on future data for a given model $g(.; \boldsymbol{w})$:

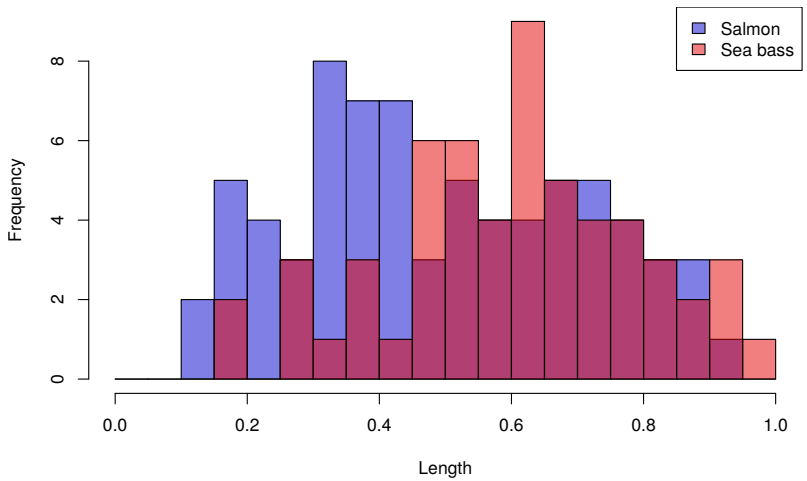
$$R(g(.; \boldsymbol{w})) = \int_{\boldsymbol{X}} \int L(y, g(\boldsymbol{x}; \boldsymbol{w})) \cdot p(\boldsymbol{x}, y) dy d\boldsymbol{x}$$

- $R(g(\boldsymbol{x}; \boldsymbol{w}))$ denotes the **expected loss** for input $\boldsymbol{x}$, and $p(\boldsymbol{x}, y)$ is the joint probability distribution for $\boldsymbol{x}$ and y.
- In practice, we hardly have any knowledge about $p(\boldsymbol{x}, y)$, but we have access to a subset of n data samples (=our dataset $(\boldsymbol{X}, \boldsymbol{y})$) which we can use to **estimate the risk**.
- We can minimize the **empirical risk** R_E on our data set (=**Empirical Risk Minimization** (ERM)):

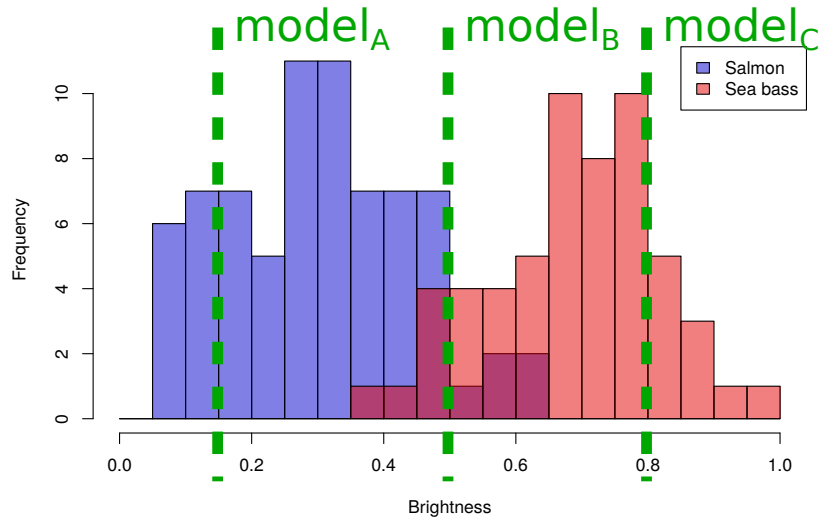
$$R_E(g(.; \boldsymbol{w}), (\boldsymbol{X}, y)) = \frac{1}{n} \cdot \sum_{i=1}^{n} L(y_i, g(\boldsymbol{x}_i; \boldsymbol{w}))$$

Back to Our Data

Length:



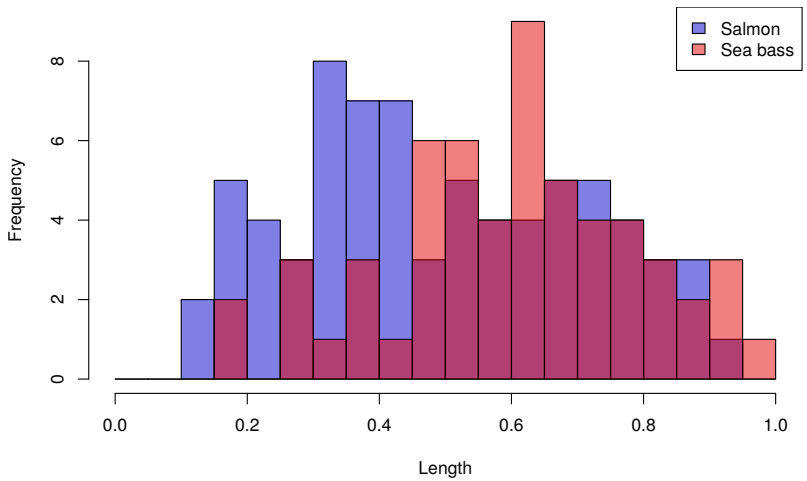
Brightness:



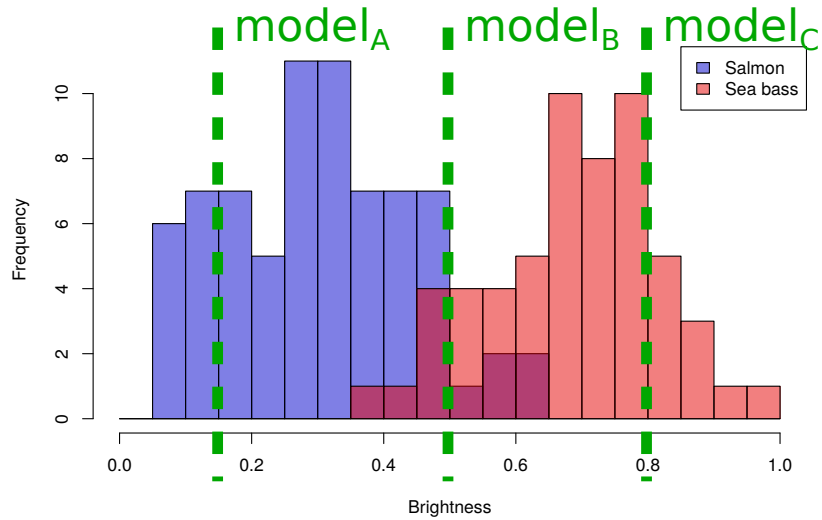
- How do we get the “best” model?
  - How does our model perform on our data?
    - **Loss function** ✓
  - How will it perform on (unseen) future data?
    - **Generalization error/risk** ✓

Back to Our Data

Length:



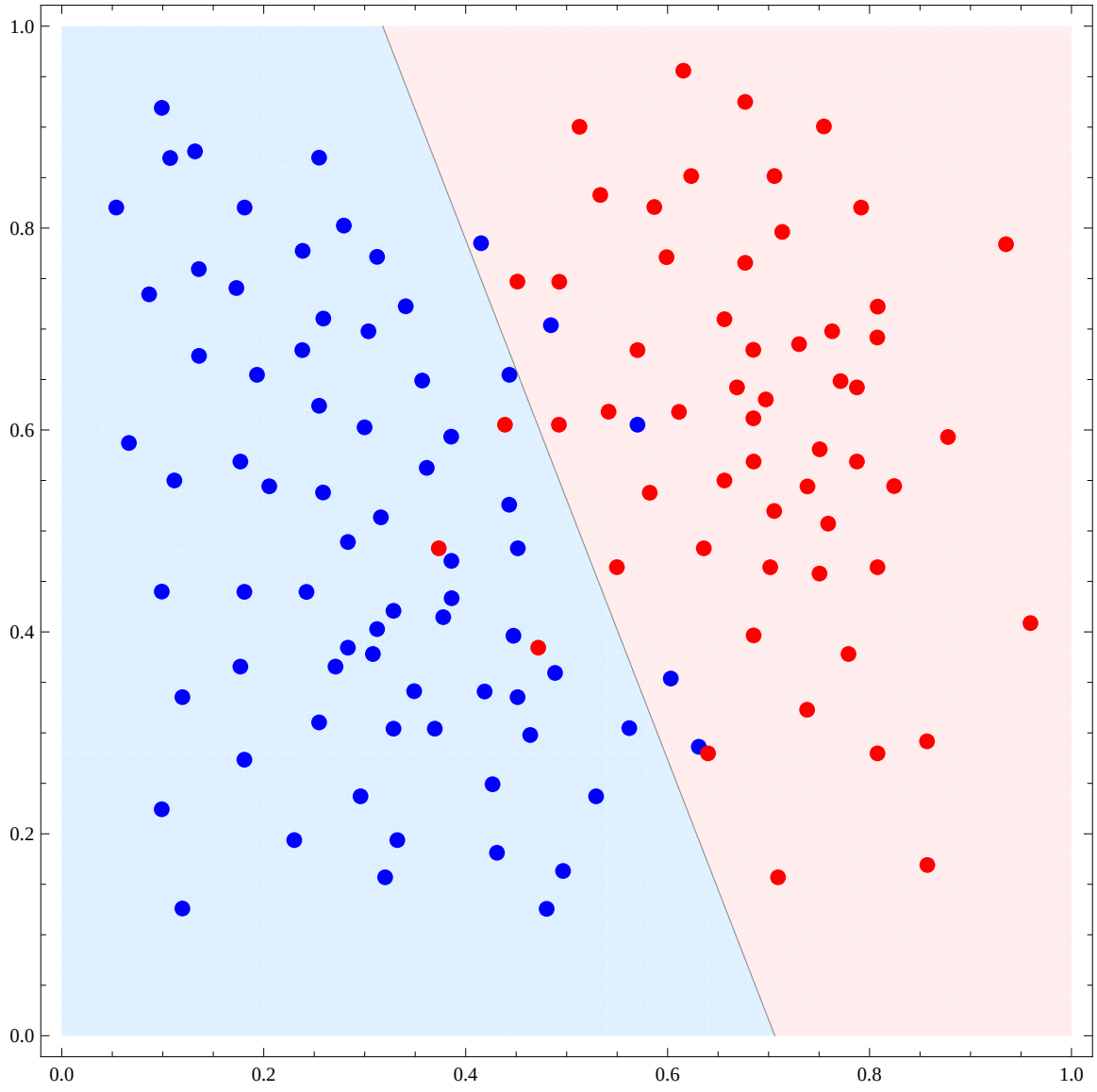
Brightness:



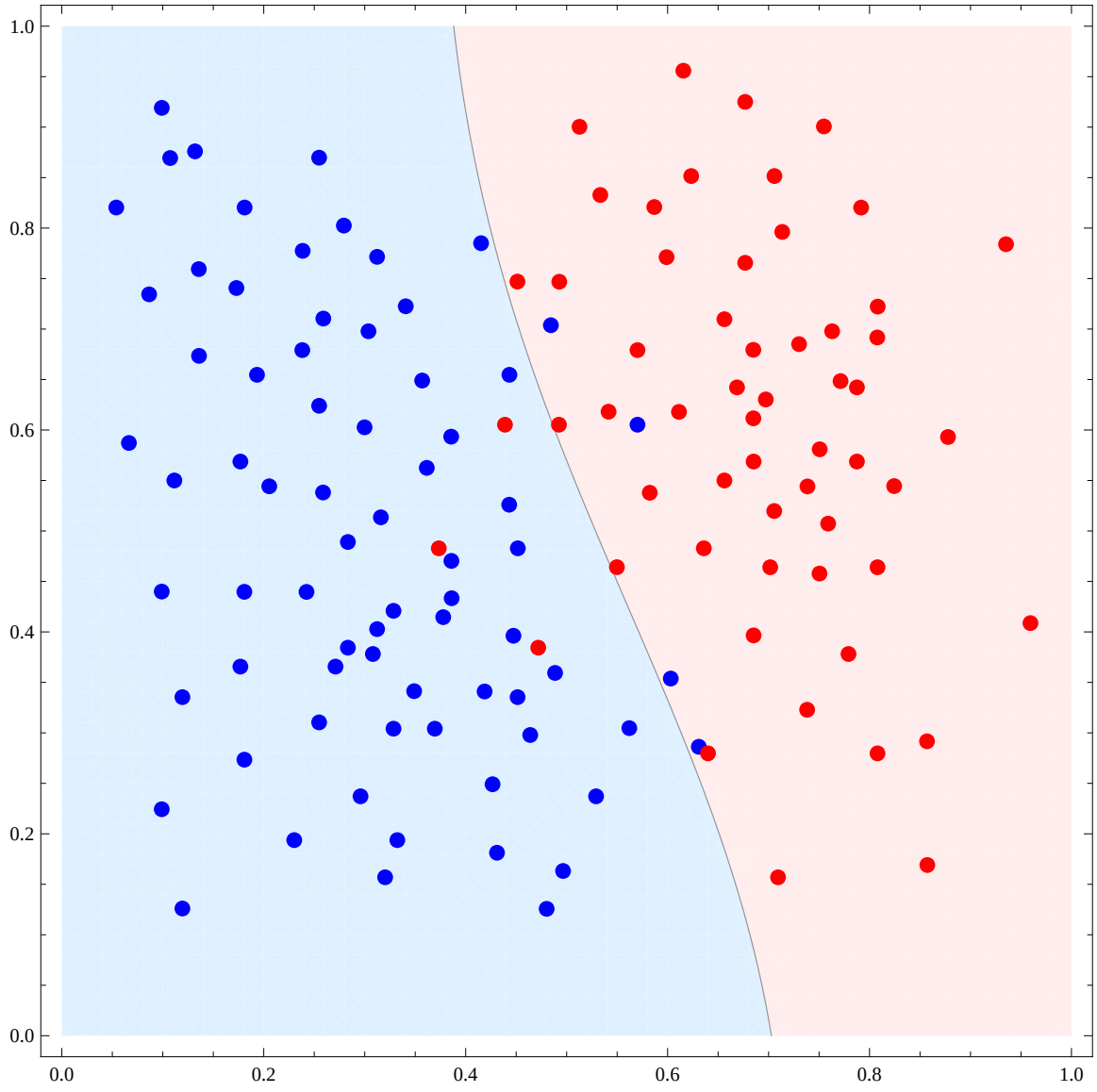
- We can now optimize our model by minimizing the risk on our (training) data set.
- But the individual features (especially length) do not separate the classes well.

→ **Combine our features** and use a different model class.

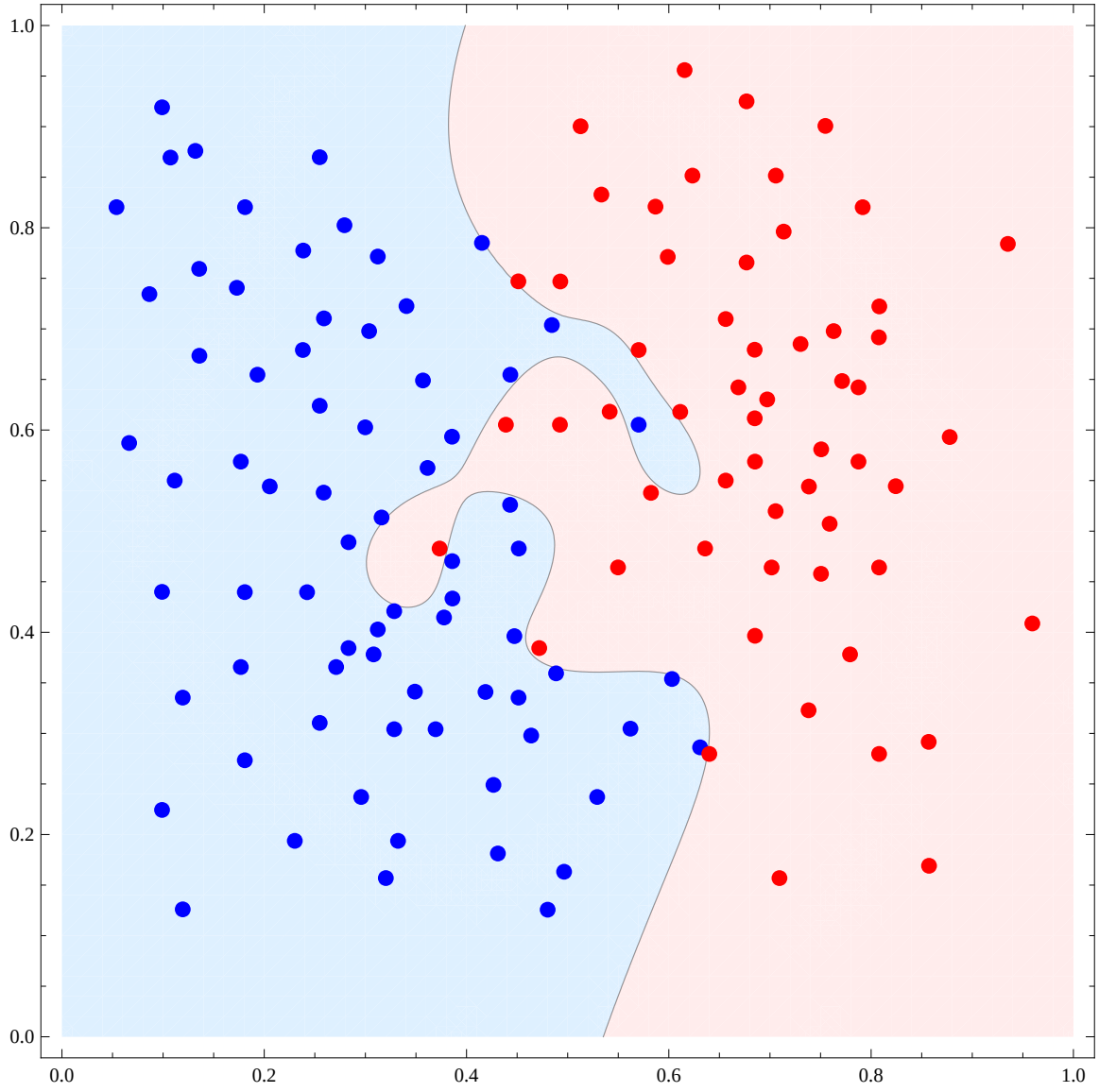
Combination: Linear Separation



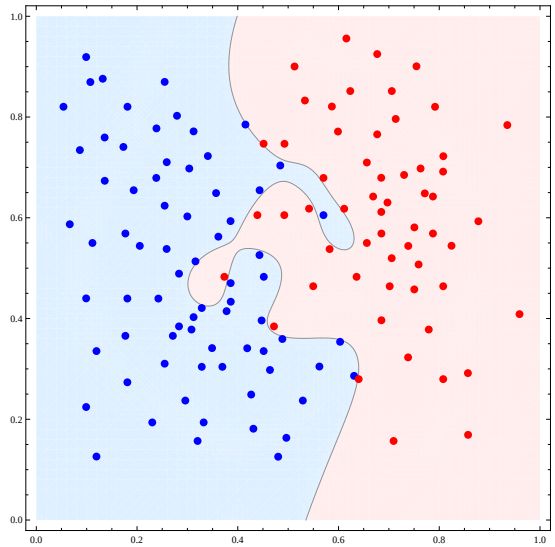
Combination: Mildly Non-linear Separation



Combination: Highly Non-linear Separation



The Problem of Overfitting



- With ERM, we can optimize our model by minimizing the loss on our data set.
- Problem: We might fit our parameters to noise specific to our data set (=**overfitting**).

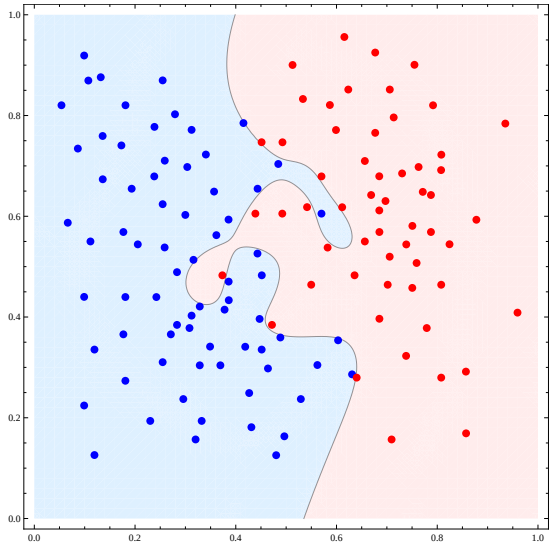
→ We need to get a better estimate for the (true) risk.

Risk Estimation: Test Set Method

- Assume our data samples are **independently and identically distributed (i.i.d.)**[2]
- We can split our data set of n samples into **two non-overlapping subsets**:
  - **Training set**: a subset with l samples we perform ERM on (i.e., optimize parameters on)
  - **Test set**: a subset with m samples we use to estimate the risk (test data = approximation of future, unseen data)
- Our estimate R_E on the test set will show if we overfit to noise in the training set.

[2] i.i.d.: Each sample has the same probability distribution as the others, and all samples are mutually independent.

Back to Our Data



- Now, we can use **ERM** to optimize a model on our **training data set** (optionally, including a validation set).
- A held-out **test set** will allow us to get an **estimate** about the performance on future data.
- If overfitting is detected, we can reduce the model complexity via hyperparameters.

From Linear Regression to Logistic Regression

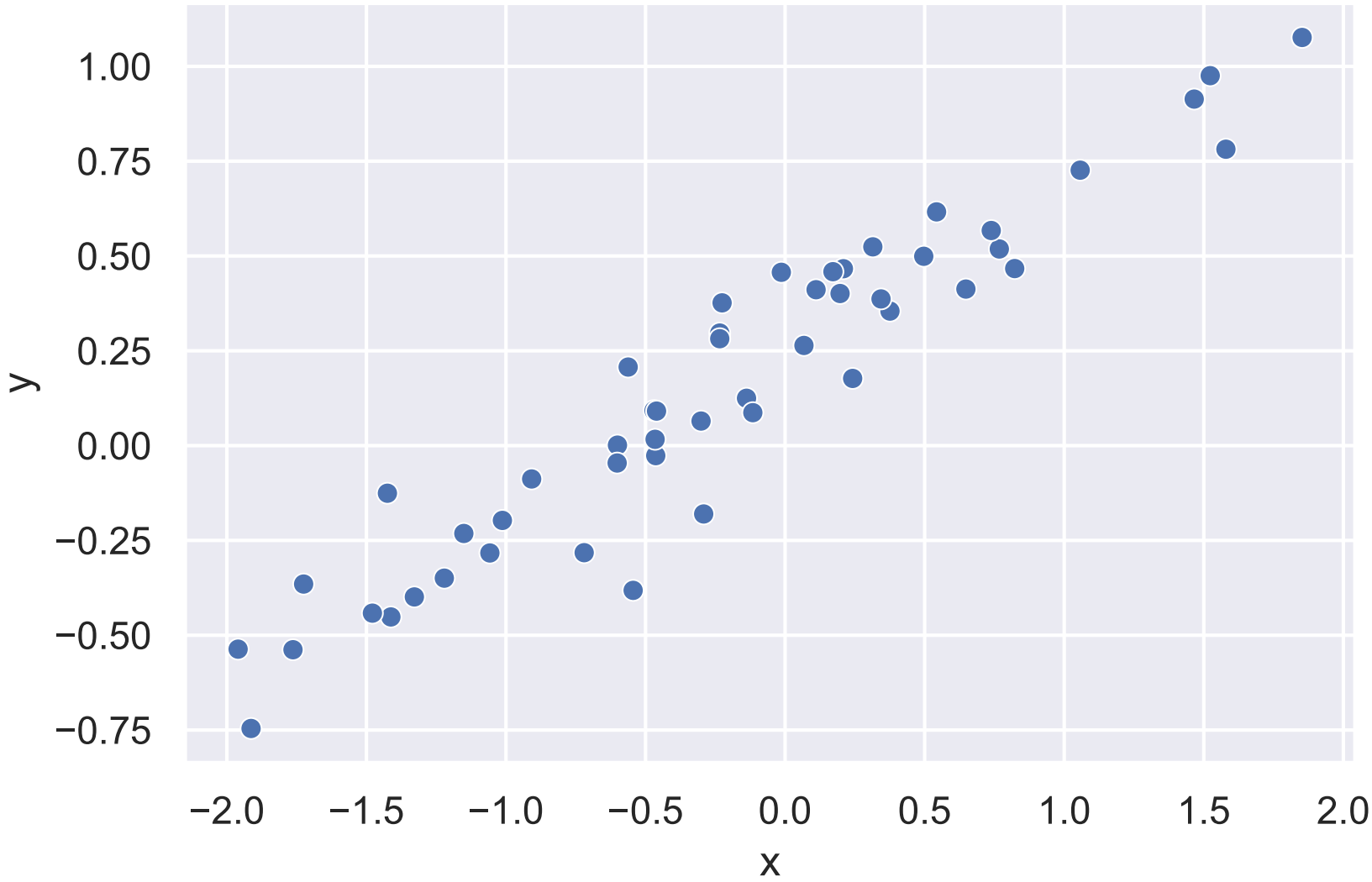
- Linear Regression is one of the simplest ML algorithms.
- Given labeled data $(\boldsymbol{X}, \boldsymbol{y}) = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n))$
  - $\boldsymbol{x}_i$: feature vector
  - y_i: corresponding label
  - n: number of samples (dataset size)
- Find model $g(\boldsymbol{x}; \boldsymbol{w})$ such that $\forall i : g(\boldsymbol{x}_i; \boldsymbol{w}) \approx y_i$.
- Simplest approach: use something linear:

$$g(\boldsymbol{x}_i; \boldsymbol{w}) = g(\boldsymbol{x}_i; a, b) = a + b \cdot \boldsymbol{x}_i$$

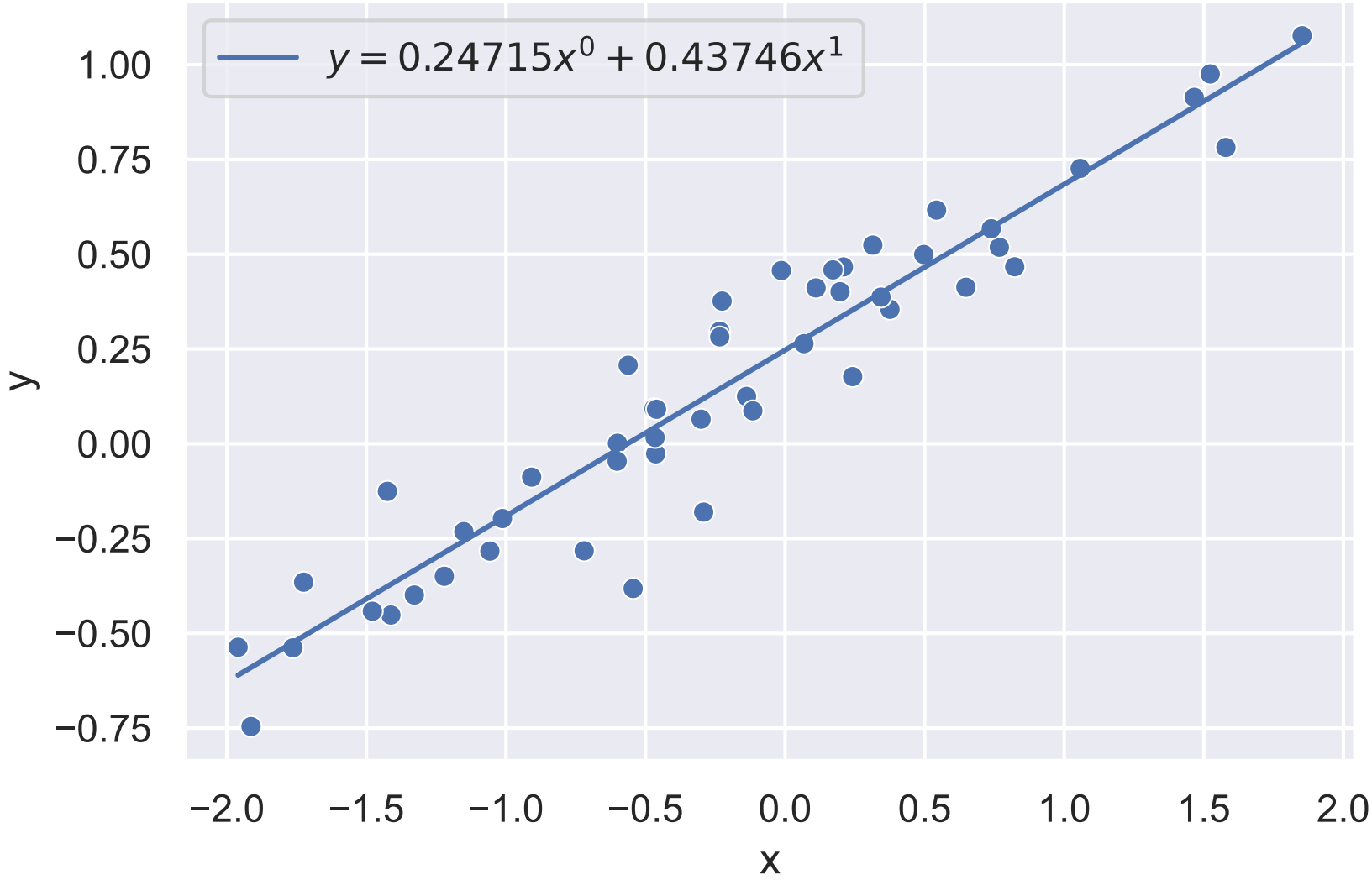
- Idea: **Minimize mean-squared error loss**:

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - g(\boldsymbol{x}_i; \boldsymbol{w}) \right)^2$$

Linear Regression



Linear Regression

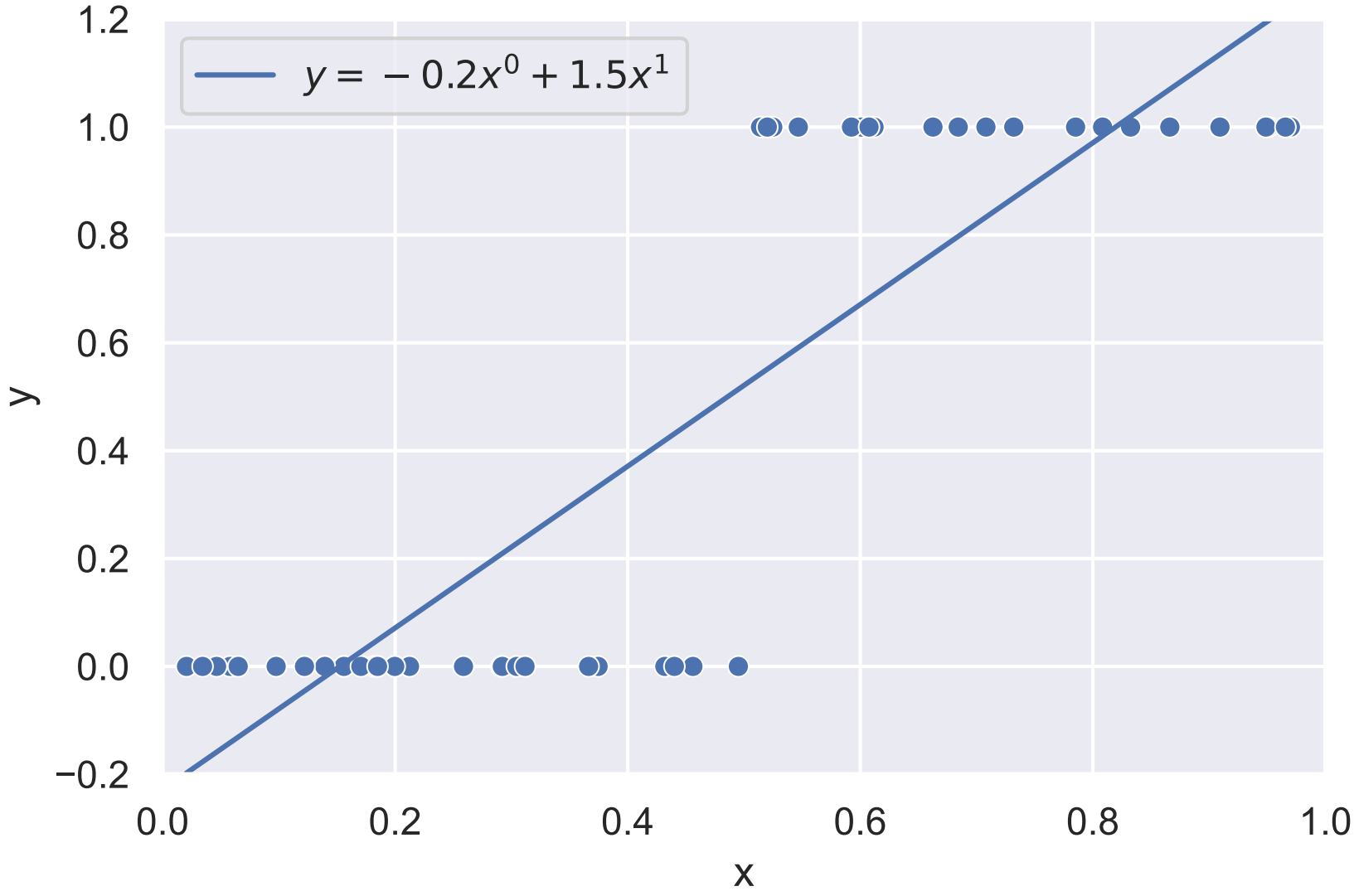


Problems With Linear Regression

- Given: n datapoints $\boldsymbol{x}_i$ with labels $y_i \in \{0, 1\}$
- Task: find $g(\boldsymbol{x}; \boldsymbol{w})$ such that $g(\boldsymbol{x}_i; \boldsymbol{w}) = y_i$
  $\Rightarrow$ **Classification task**
- First (bad) idea: fit a linear regression line
- Then, get classes based on some threshold:

$$y_i = \begin{cases} 0 & g_{\text{LinReg}}(\boldsymbol{x}_i; \boldsymbol{w}) < 0.5 \\ 1 & g_{\text{LinReg}}(\boldsymbol{x}_i; \boldsymbol{w}) \geq 0.5 \end{cases}$$

Problems With Linear Regression



Logistic Regression

- The relationship between features $\boldsymbol{x}_i$ and labels y_i is **not linear**, instead we apply the logistic/sigmoid function

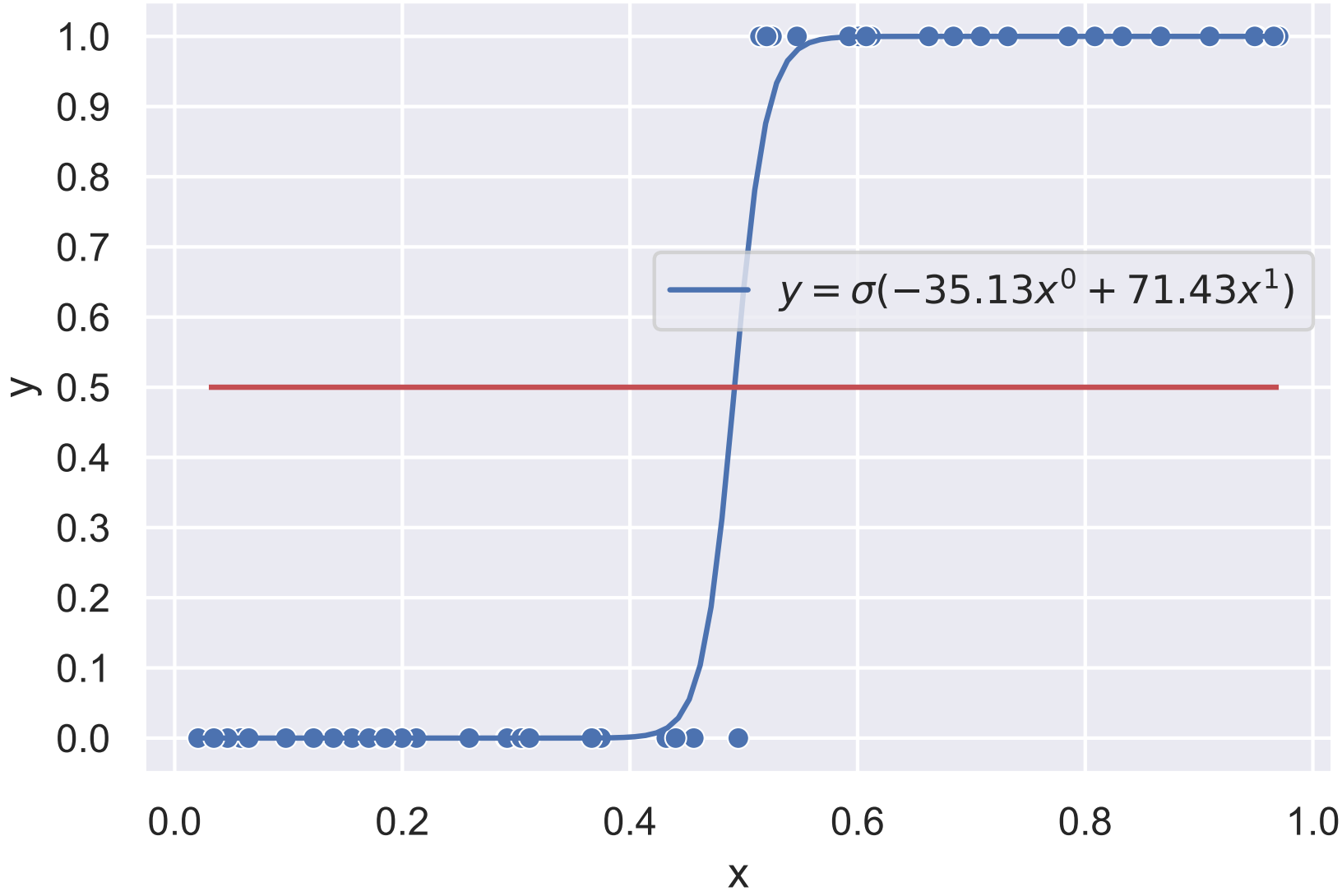
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is a linear function of features (in our case, this will be the "raw"/linear model output). Example:

$$g(\boldsymbol{x}_i; \boldsymbol{w}) = \sigma(a + b \cdot \boldsymbol{x}_i) \qquad \text{with} \quad \boldsymbol{w} = \{a, b\}$$

- Logistic regression is a **regression model** because it estimates the **probability of class membership** (output y is a real numeric value, i.e., $y \in [0, 1] \in \mathbb{R}$).
- Classification: Use some **decision rule** (e.g., threshold).

Logistic Regression



Softmax

- Generalization of the sigmoid function.
- Suitable for **multi-class** classification.
- For K classes with $y \in \{1, \ldots, K\}$ the probability of $\boldsymbol{x}$ belonging to class i is:

$$p(y = i \mid \boldsymbol{x}) = \sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

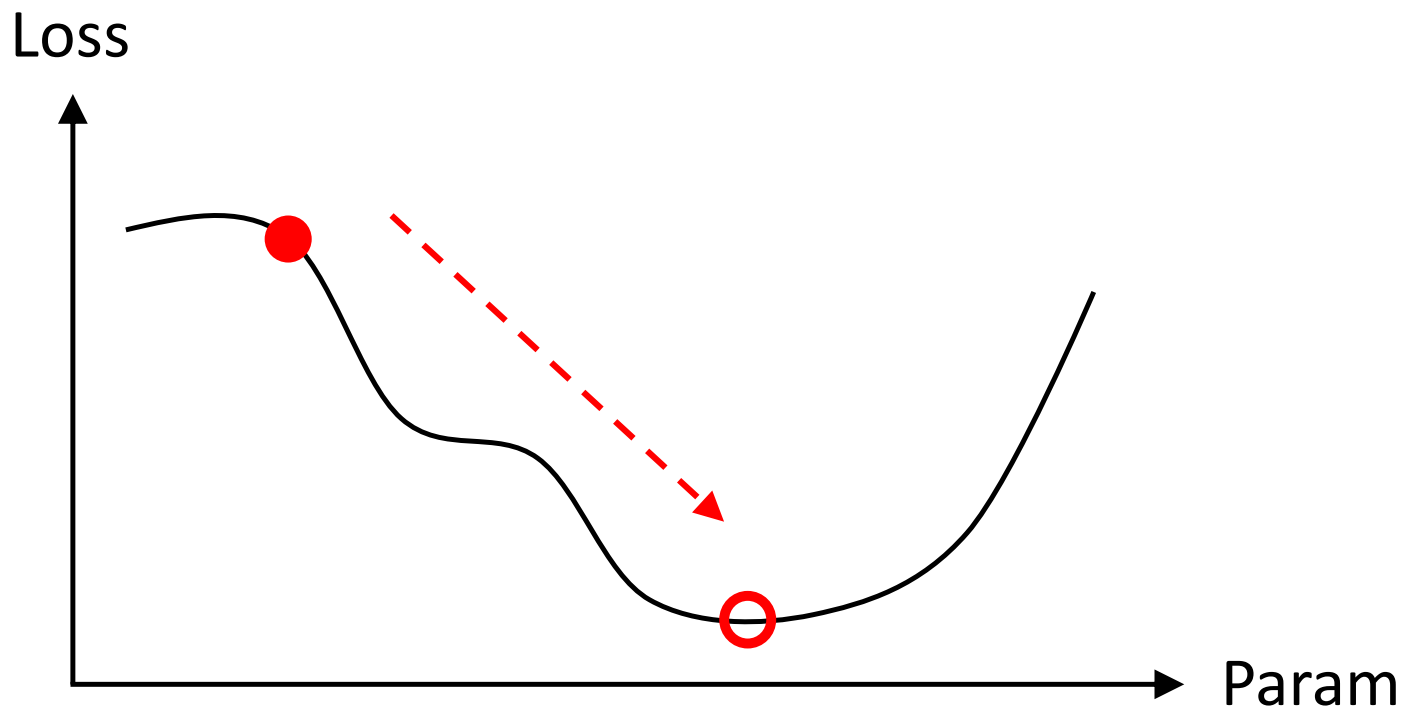
  where $\boldsymbol{z} = (z_1, \ldots, z_K)$ is the vector of "raw" model outputs, i.e., there is an output for each class (see example later).
- While not necessary, as a regular sigmoid function would suffice, this also works for binary classification, i.e., $K = 2$.

Logistic Regression Has No Closed Form Solution

- Output of logistic regression are now probabilities: Use **(Binary) Cross Entropy** as appropriate loss function.
- **No closed-form solution** = Minimum of the loss cannot be calculated directly (no analytical solution).
- Iterative methods have to be used for minimizing the loss.
- One prominent example is **gradient descent**.

Gradient Descent

- Given: a function $f(x)$
- Task: find x that maximizes (or minimizes) $f(x)$
- Idea: start at some value x_0, and take a small step η in the direction in which the function decreases strongest



Gradient Descent in Logistic Regression

- The minimization of the loss function $L(.;\theta)$ can be done by gradient descent:

$$\theta_{n+1} = \theta_n - \eta \frac{\partial L}{\partial \theta}$$

where η is the learning rate and θ is the parameter or set of parameters to be optimized.

- In the case

$$g(\boldsymbol{x}_i; \boldsymbol{w}) = \sigma(a + b \cdot \boldsymbol{x}_i)$$

the set of parameters is $\theta = \boldsymbol{w} = \{a, b\}$.

PyTorch

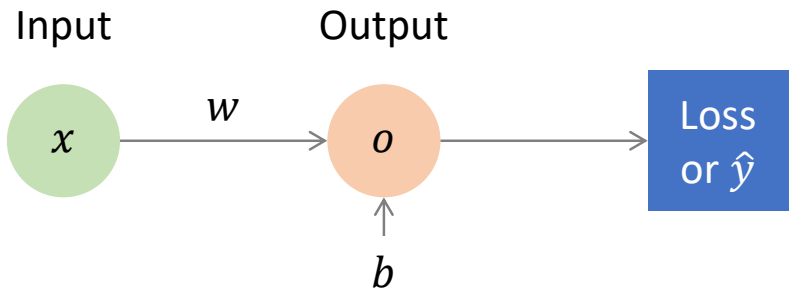
Overview

- An **open source deep learning platform for Python** that is very well suited for experimental approach and prototyping as well as production.
- **Automatic computation of gradients** (for learning) through the creation of **computational graphs**.
- Graphs are created dynamically based on connected building blocks (nodes) that include data (e.g., input features) and operations (e.g., loss function).

From Logistic Regression towards Neural Networks

- Given binary labeled 1D data $((x_1, y_1), \ldots (x_n, y_n))$, where $x \in \mathbb{R}^1$ and $y \in \{0, 1\}$ (i.e., two classes 0 and 1).
- A matching logistic regression model would look like $o = g(x; \{d, k\}) = \sigma(z) = \sigma(d + k \cdot x)$, where z is the "raw" model output and o the final model output (a probability) for some given input x and the model parameters d and k.
- As preparation for the following steps, let's substitute the model parameters as follows: $d \to b, k \to w$, so our model now looks like $g(x; \{b, w\}) = \sigma(b + w \cdot x)$. Also, for brevity, we can optionally write just $g(x)$, i.e., we do not explicitly include the parameters for a more compact notation.
- We will call b the **bias** and w the **weight**.

Visualization and Output Calculation



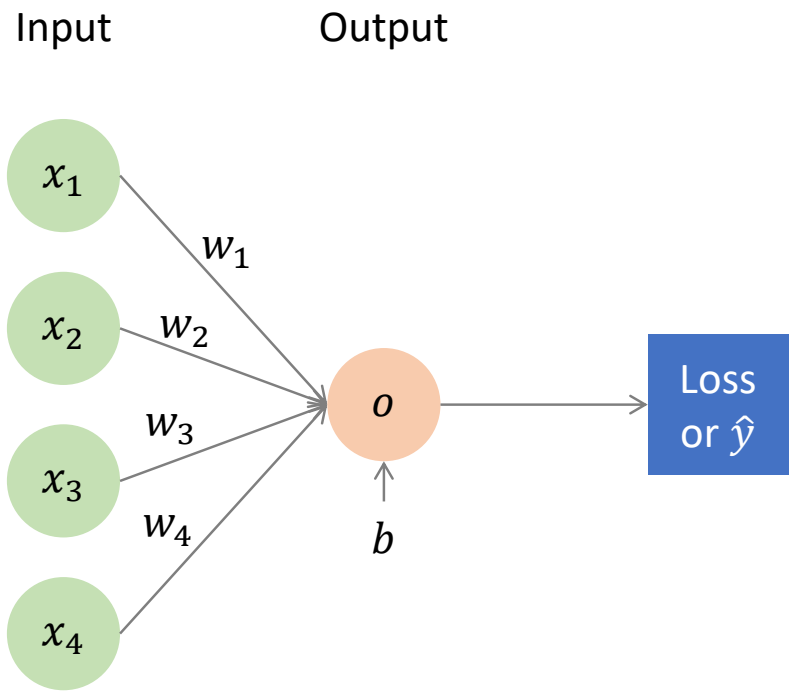
- The above is a simplified visualization of the data flow through our model: $o = \sigma(z) = \sigma(a + b \cdot x)$.
- The output o can then be used for two things:
  - Calculating the **loss** during gradient descent to repeatedly update our model parameters (bias b and weight w) to decrease the loss.
  - Determining the class **prediction** $\hat{y}$ (recall: we have to apply some decision rule, e.g., via a threshold).

Multi-Dimensional Example

- Given binary labeled M-dimensional data $((\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n))$, where $\boldsymbol{x} \in \mathbb{R}^M$ and $y \in \{0, 1\}$ (i.e., two classes 0 and 1).
- A matching logistic regression model would look like $o = g(\boldsymbol{x}; \{b, \boldsymbol{w}\}) = \sigma(z) = \sigma(b + \boldsymbol{w} \cdot \boldsymbol{x})$, where z is the "raw" model output and o the final model output (a probability) for some given input $\boldsymbol{x}$ and the model parameters b and $\boldsymbol{w}$.
- Note that the parameter $\boldsymbol{w}$ is now a vector,[3] and $\boldsymbol{w} \cdot \boldsymbol{x}$ is the scalar/dot product, i.e., $\sum_{i=1}^{M} w_i x_i$.

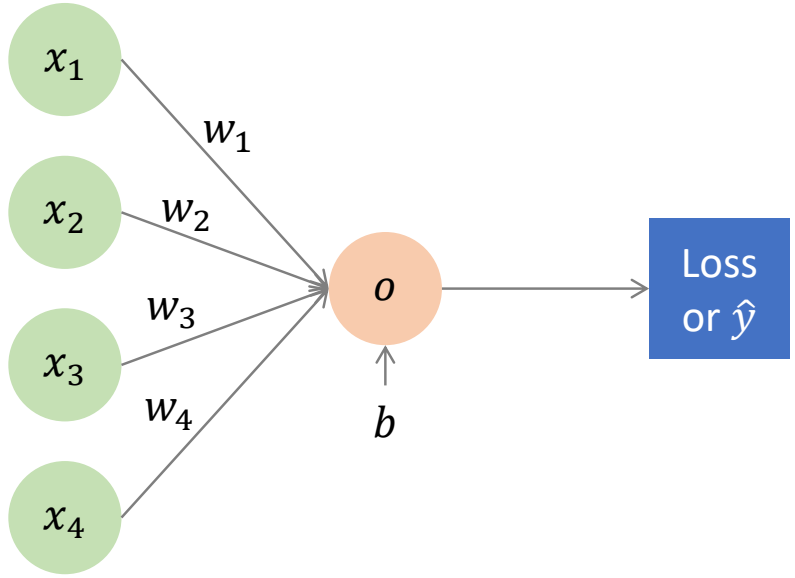
[3] Strictly speaking, b is internally also a vector since $b + \boldsymbol{w} \cdot \boldsymbol{x}$ is actually $\boldsymbol{b} \cdot \boldsymbol{x}^0 + \boldsymbol{w} \cdot \boldsymbol{x}^1$. However, because the computation does not depend on the input $\boldsymbol{x}$, i.e., $\boldsymbol{b} \cdot \boldsymbol{x}^0 = \boldsymbol{b} \cdot \boldsymbol{1} = \sum_{i=1}^{M} b_i \cdot 1 = \sum_{i=1}^{M} b_i$, we simplify it to just b.

Visualization and Output Calculation



- The above is a simplified visualization of the data flow through our model with an $M = 4$-dimensional input size. See the next slide for how the final model output is calculated.

Output Calculation



- Model parameters b and $\boldsymbol{w}$:

$$b, \boldsymbol{w} = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix}$$

- "Raw"/linear model output z given $M = 4$-dimensional input $\boldsymbol{x}$ (with $\boldsymbol{w} \cdot \boldsymbol{x}$ being the scalar product):

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, z = b + \boldsymbol{w} \cdot \boldsymbol{x} = b + \sum_{i=1}^{4} w_i x_i$$
$$= b + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$$

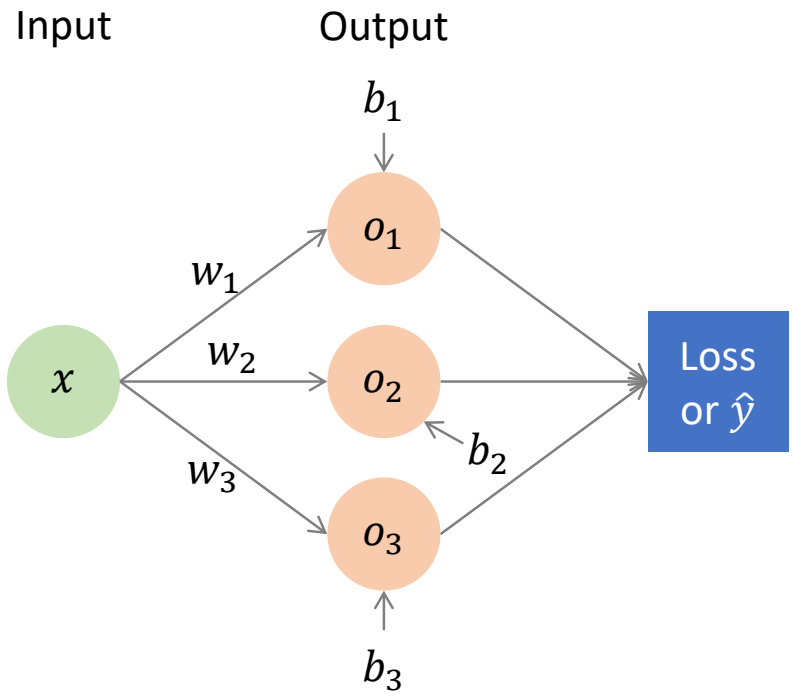
- Final model output o (class probability):

$$o = \sigma(z)$$

Multi-Class Example

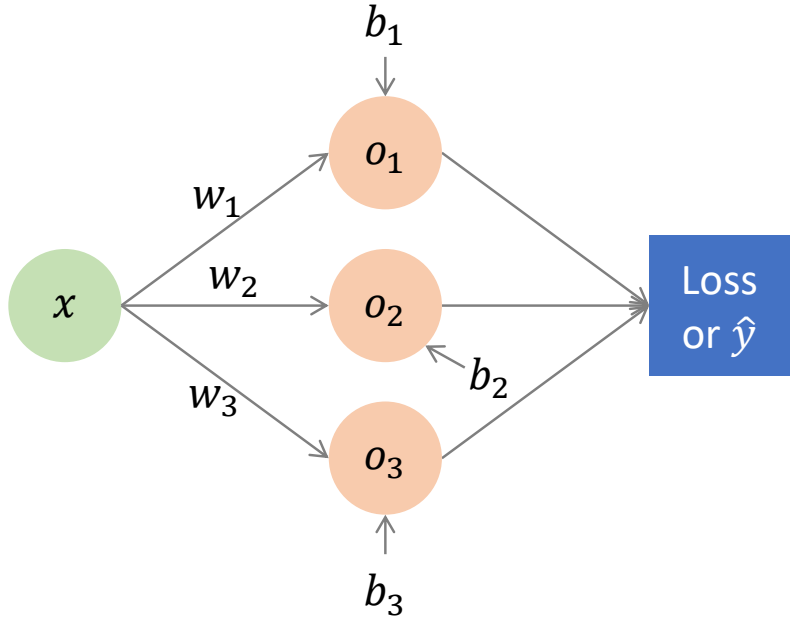
- Given K-multi-class labeled 1D data $((x_1, y_1), \ldots (x_n, y_n))$, where $x \in \mathbb{R}^1$ and $y \in \{1, \ldots, K\}$.
- Recall that the softmax function is suitable for multi-class classification, so we replace our logistic function $\sigma(z)$ with the softmax function $\sigma(\boldsymbol{z})_i$, where i means the i-th class and $\boldsymbol{z} = (z_1, \ldots, z_K)^\top$ is the vector of "raw" model outputs.
- A matching logistic regression model would look like $\boldsymbol{o} = (o_1, \ldots, o_K)^\top = g(x; \{\boldsymbol{b}, \boldsymbol{w}\}) = (\sigma(\boldsymbol{z})_1, \ldots, \sigma(\boldsymbol{z})_K)^\top$, where $\boldsymbol{o}$ is the final vector of K class probabilities for some given input x and the model parameters $\boldsymbol{b}$ and $\boldsymbol{w}$.
- Note that the parameters $\boldsymbol{b}$ and $\boldsymbol{w}$ are now vectors.

Visualization



- The above is a simplified visualization of the data flow through our model with 1D input size and $K = 3$ classes. See the next slide for how the final model output is calculated.

Output Calculation



- Model parameters $\boldsymbol{b}$ and $\boldsymbol{w}$:

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

- "Raw"/linear model output $\boldsymbol{z}$ given input x:

$$x, \boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \boldsymbol{b} + \boldsymbol{w} \cdot x = \begin{bmatrix} b_1 + w_1 x \\ b_2 + w_2 x \\ b_3 + w_3 x \end{bmatrix}$$

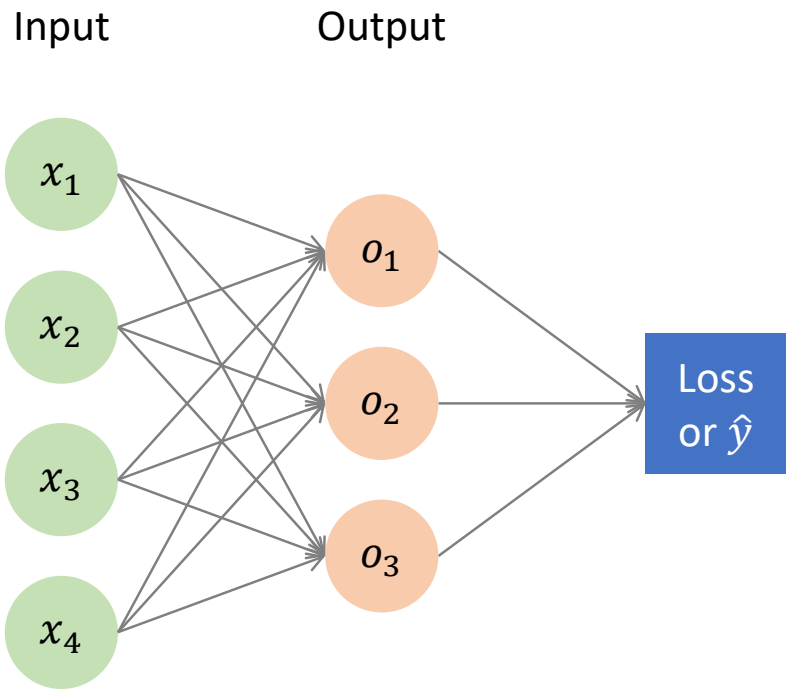
- Final model output $\boldsymbol{o}$ (class probability vector):

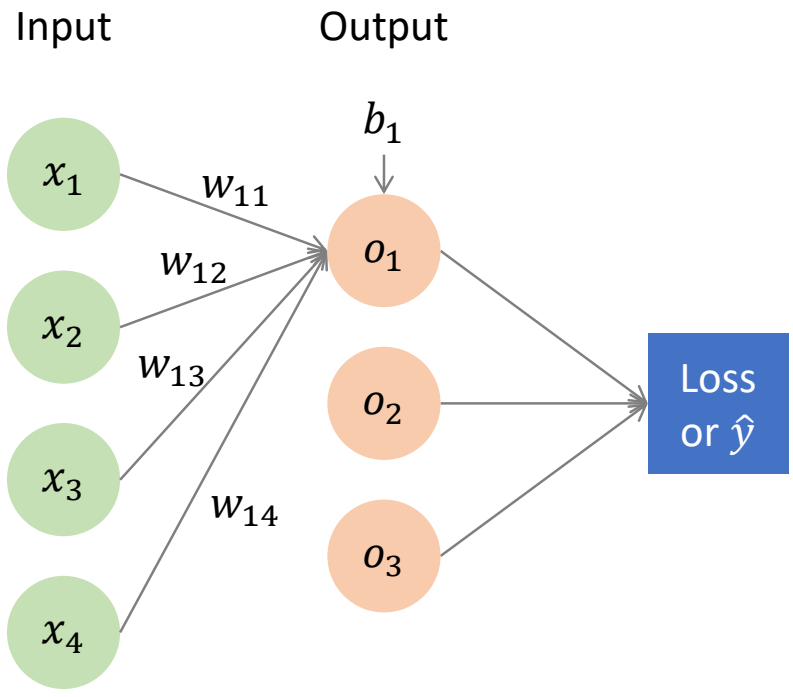
$$\boldsymbol{o} = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \end{bmatrix} = \begin{bmatrix} \sigma(\boldsymbol{z})_1 \\ \sigma(\boldsymbol{z})_2 \\ \sigma(\boldsymbol{z})_3 \end{bmatrix}$$

Multi-Dimensional and Multi-Class Example

- Given K-multi-class labeled M-dimensional data $((\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n))$, where $\boldsymbol{x} \in \mathbb{R}^M$ and $y \in \{1, \ldots, K\}$.
- A matching logistic regression model would look like $\boldsymbol{o} = (o_1, \ldots, o_K)^\top = g(\boldsymbol{x}; \{\boldsymbol{b}, \boldsymbol{W}\}) = (\sigma(\boldsymbol{z})_1, \ldots, \sigma(\boldsymbol{z})_K)^\top$, where $\boldsymbol{o}$ is the final vector of K class probabilities for some given input $\boldsymbol{x}$ and the model parameters $\boldsymbol{b}$ and $\boldsymbol{W}$.
- Note that the parameter $\boldsymbol{b}$ is now a vector and the parameter $\boldsymbol{W}$ a matrix.

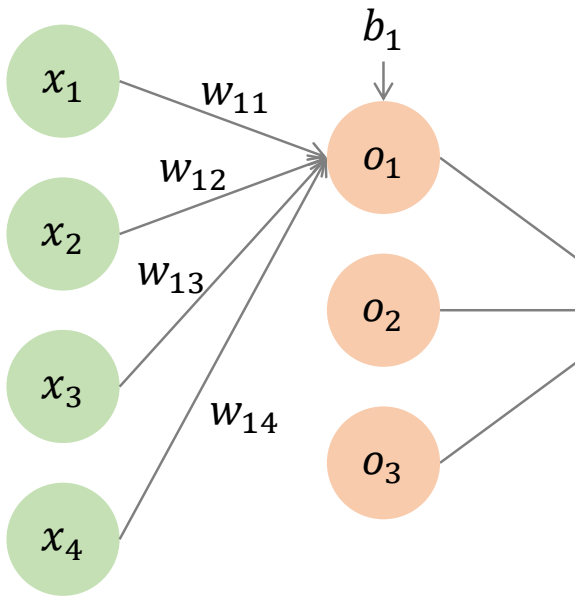
Visualization





- The above is a simplified visualization of the data flow through our model with $M = 4$-dimensional input size and $K = 3$ classes. See the next slide for how the final model output is calculated.

Output Calculation



- Model parameters $\boldsymbol{b}$ and $\boldsymbol{W}$:

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, \boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}$$

- "Raw"/linear model output $\boldsymbol{z}$ given $M = 4$-dimensional input $\boldsymbol{x}$ (with $\boldsymbol{W}\boldsymbol{x}$ being the matrix product):

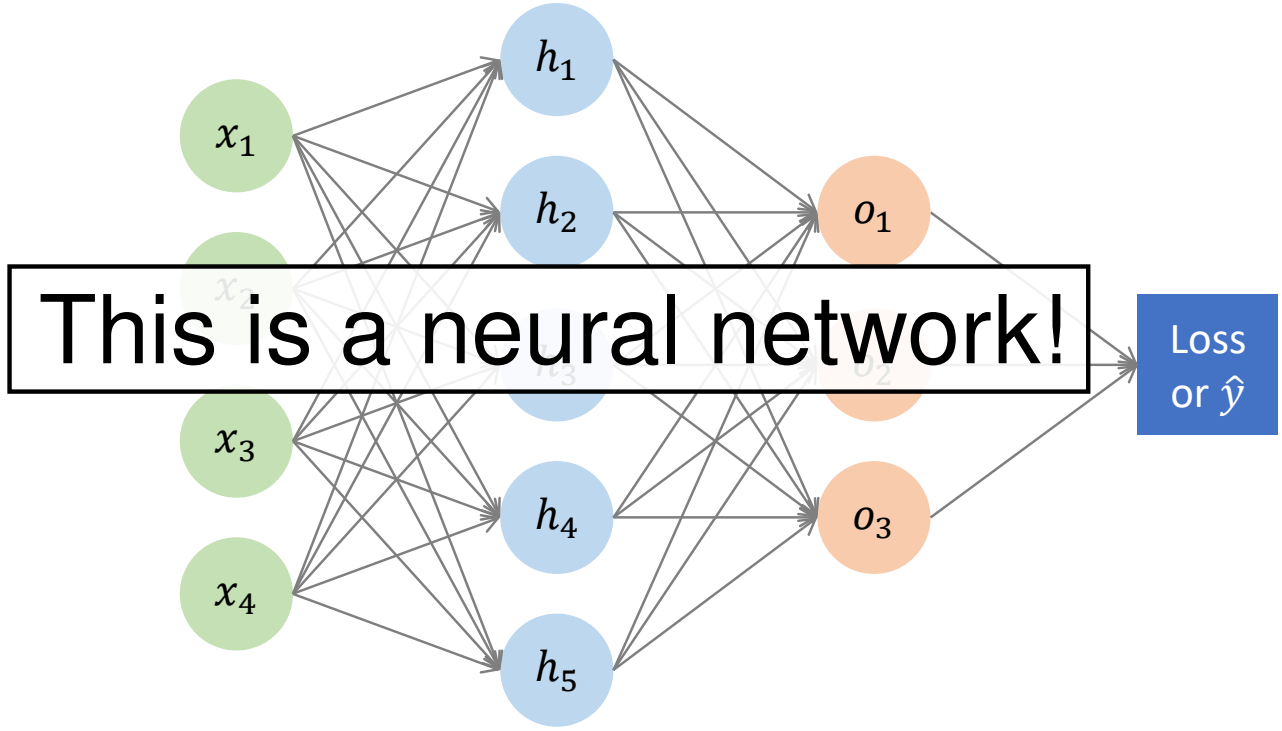
$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{x} = \begin{bmatrix} b_1 + \sum_{i=1}^{4} w_{1i}x_i \\ b_2 + \sum_{i=1}^{4} w_{2i}x_i \\ b_3 + \sum_{i=1}^{4} w_{3i}x_i \end{bmatrix}$$

- Final model output $\boldsymbol{o}$ (class probability vector):

$$\boldsymbol{o} = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \end{bmatrix} = \begin{bmatrix} \sigma(\boldsymbol{z})_1 \\ \sigma(\boldsymbol{z})_2 \\ \sigma(\boldsymbol{z})_3 \end{bmatrix}$$

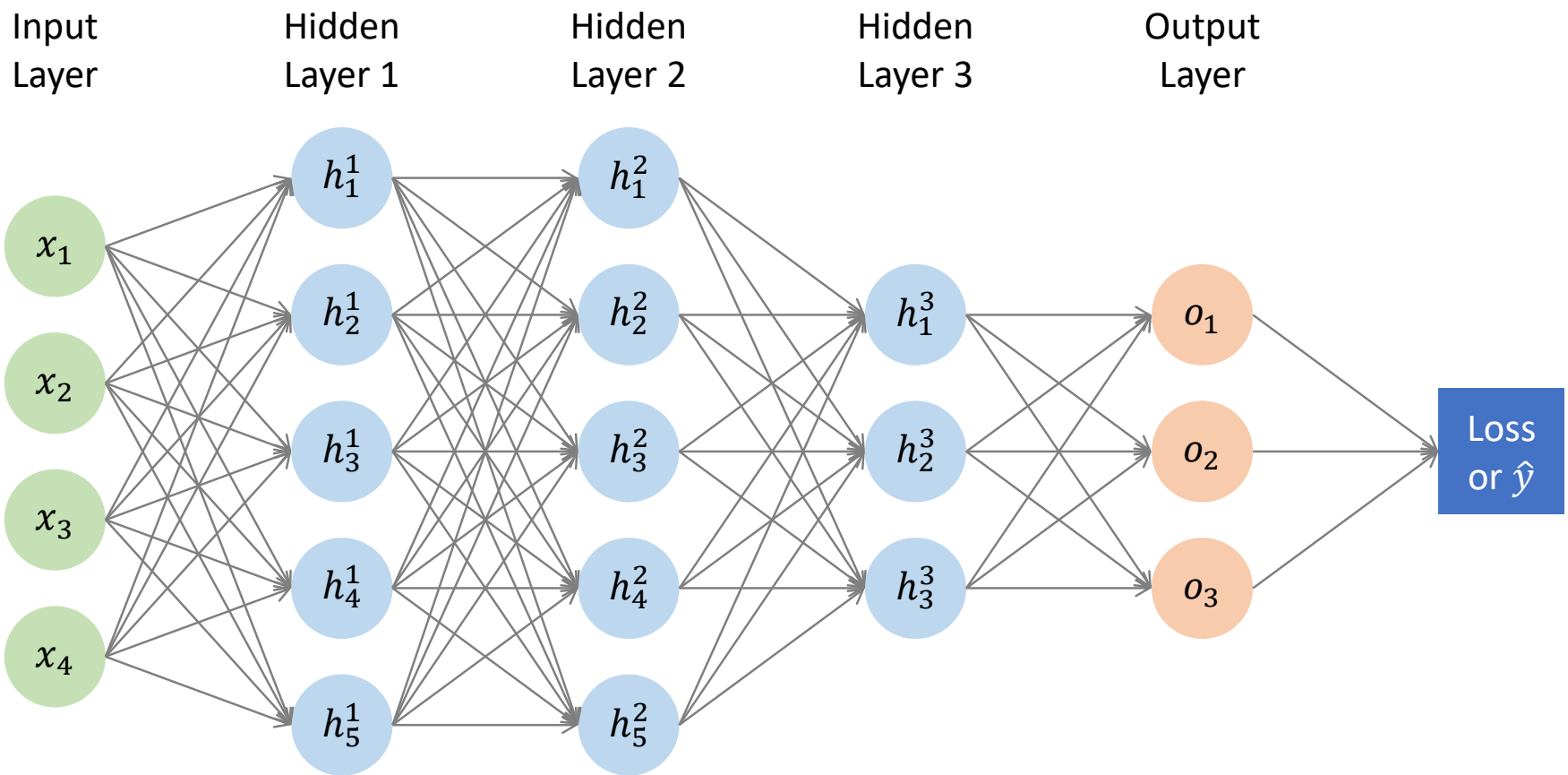
Logistic Regression $\Rightarrow$ Neural Networks

- Again as preparation, let's rename *Input* and *Output*:
- What if we made our model g **more complex/powerful**?
- We could just reuse the idea of our bias-weight calculation by **adding** an intermediate/hidden **layer** h with output $\boldsymbol{h} = h(\boldsymbol{x}; \{\boldsymbol{b}, [illegible]$ $\boldsymbol{x}))$:



Neural Networks

- We can add even more such layers, and the number of **nodes** (a.k.a. **units** or **neurons**) can also vary.
- E.g., 3 layers with a final output of $\boldsymbol{o} = g(h^3(h^2(h^1(\boldsymbol{x}))))$:



Non-linearity

- Simply stacking linear layers does not work, since multiple linear transformations can be collapsed into a single linear transformation.
- To get rid of this problem, we thus add a **non-linear function** f after each layer (same idea we had when going from linear regression to logistic regression).

$$\text{Before: } h(\boldsymbol{x}) = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}$$

$$\text{After: } h(\boldsymbol{x}) = f(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x})$$

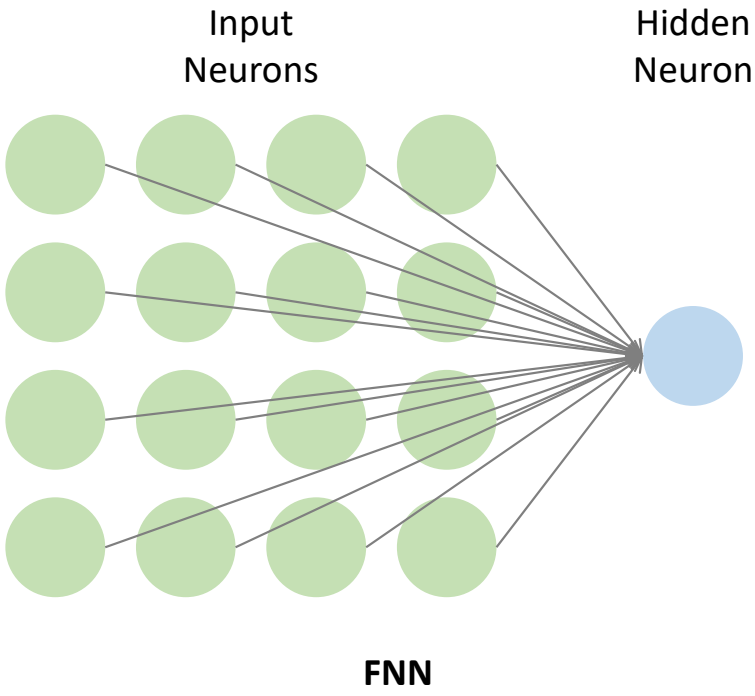
- These functions are also called **activation functions**. Examples: Sigmoid, ReLU, Tanh, etc.

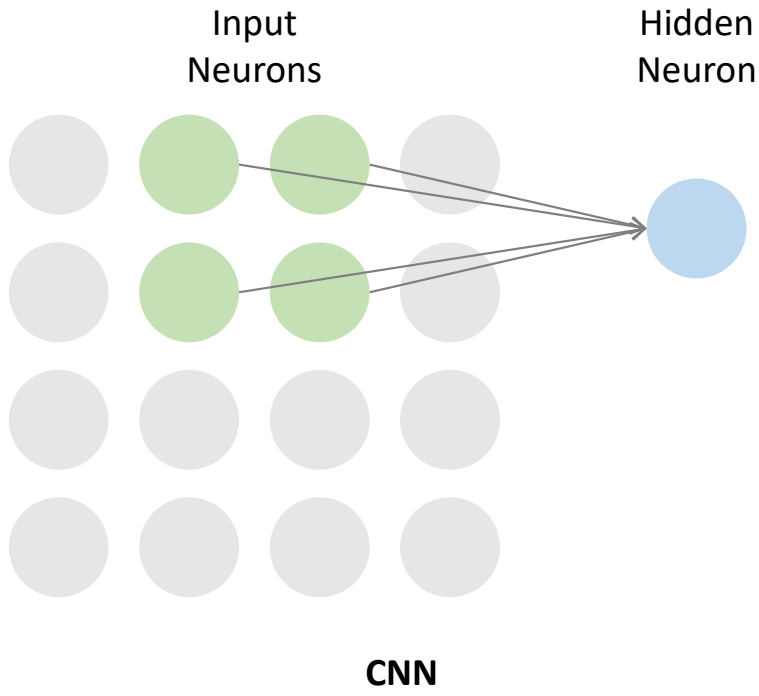
Convolutional Neural Networks (CNNs)

- In contrast to fully-connected NNs, CNNs only connect parts of the input to a neuron of the following layer: **receptive field** with **shared** weight matrix (=**kernel**).
- CNNs are useful whenever there is "local structure" in the data, e.g., pixel/audio/voxel data.
- Example: image-like data:
  - Extremely high-dimensional.
  - Pixels that are near each other are **highly correlated**.
  - Same basic patches (e.g., edges, corners) appear on all positions of the image.
  - Often **invariances to certain variations** are desired (e.g., translation invariance).

Receptive Field

- **FNN**: In a feed-forward neural network, each hidden neuron is connected to all neurons of the previous layer.
- **CNN**: In a convolutional neural network, a hidden neuron is only connected to a few neurons in the previous layer.





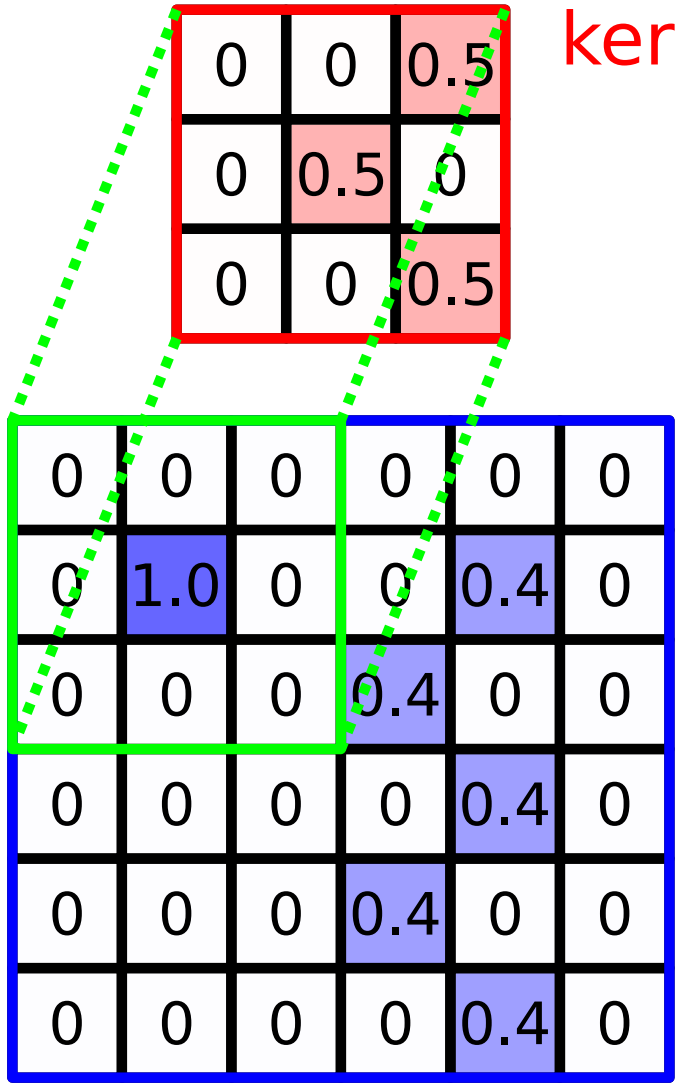
Convolution and Weight Sharing

- Same basic patches (e.g., edges, corners) appear on all positions of the image.
- Often, **invariances to certain variations** are desired (e.g., translation invariance).

$\rightarrow$ We can reuse the receptive field at all positions of the image to produce the new output (=feature/activation map).

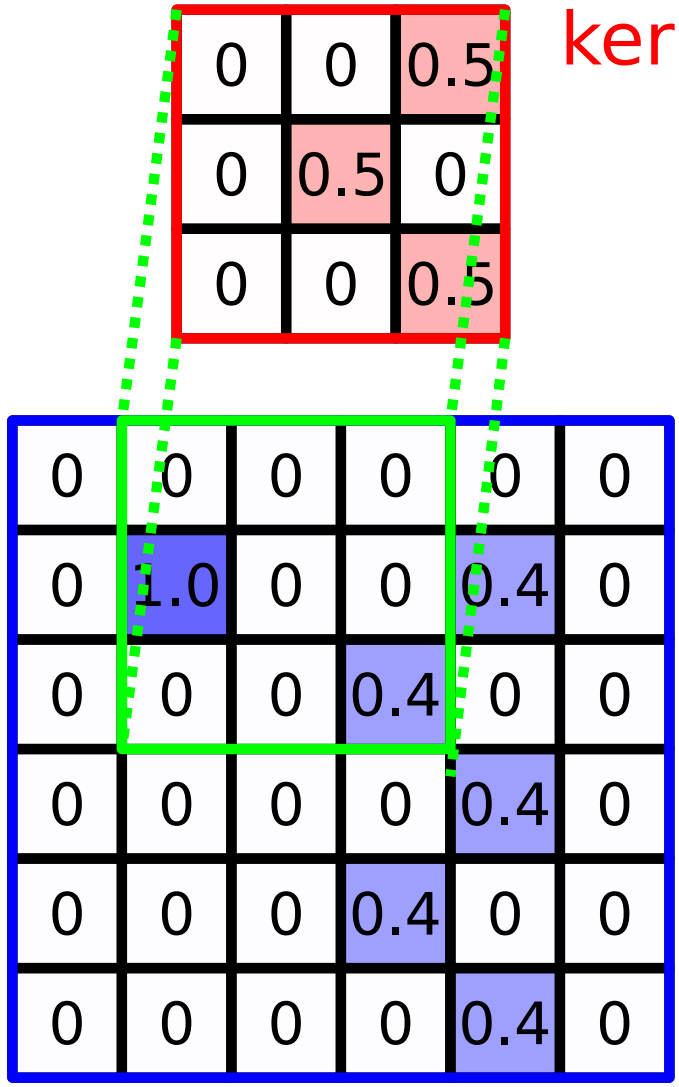
- Reusing the kernel weight matrix is called **weight sharing**.
  - We apply our kernel to all image positions while keeping the weights the same (=**convolution** operation).
  - This **significantly reduces** the number of model **parameters**.
  - Interactive kernel demo: `https://setosa.io/ev/image-kernels/`

Weight Sharing (k: 3×3, i: 6×6, o: 4×4)



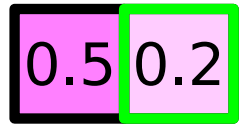
$$
\begin{aligned}
&0 \cdot 0 + \quad 0 \cdot 0 \quad + 0.5 \cdot 0 \\
&+ 0 \cdot 0 + 0.5 \cdot 1.0 + \quad 0 \cdot 0 \\
&+ 0 \cdot 0 + \quad 0 \cdot 0 \quad + 0.5 \cdot 0 \\
&= 0.5
\end{aligned}
$$

Weight Sharing (k: 3×3, i: 6×6, o: 4×4)

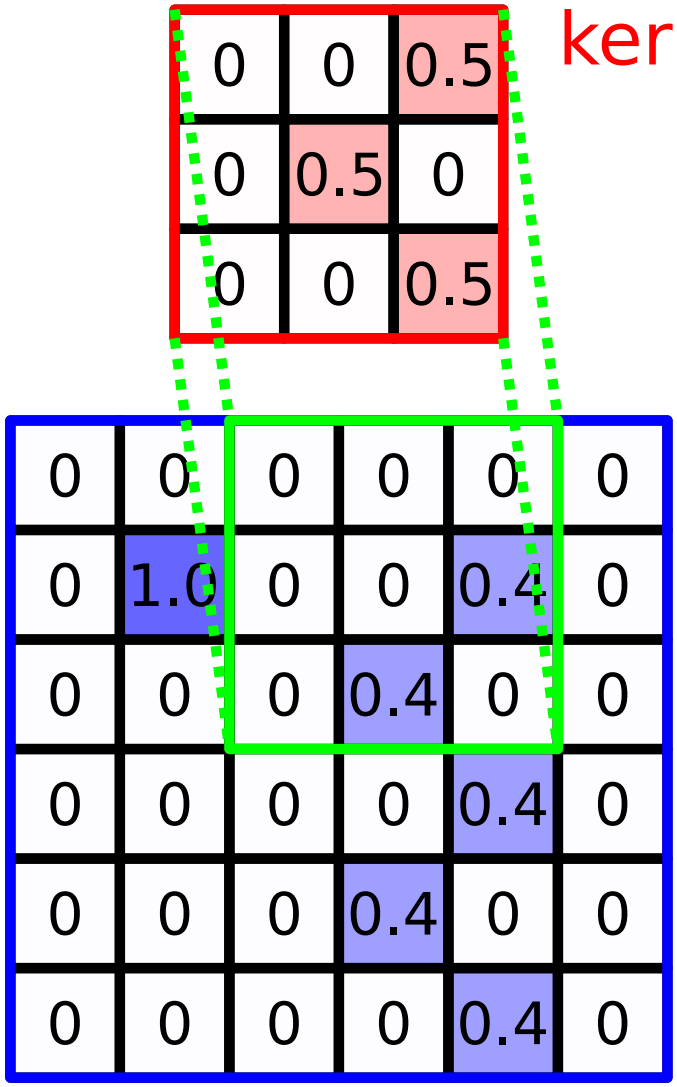


$$\begin{aligned} &0 \cdot 0 + 0 \cdot 0 + 0.5 \cdot 0 \\ &+ 0 \cdot 1.0 + 0.5 \cdot 0 + 0 \cdot 0 \\ &+ 0 \cdot 0 + 0 \cdot 0 + 0.5 \cdot 0.4 \\ &= 0.2 \end{aligned}$$

output

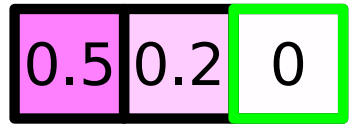


Weight Sharing (k: 3×3, i: 6×6, o: 4×4)

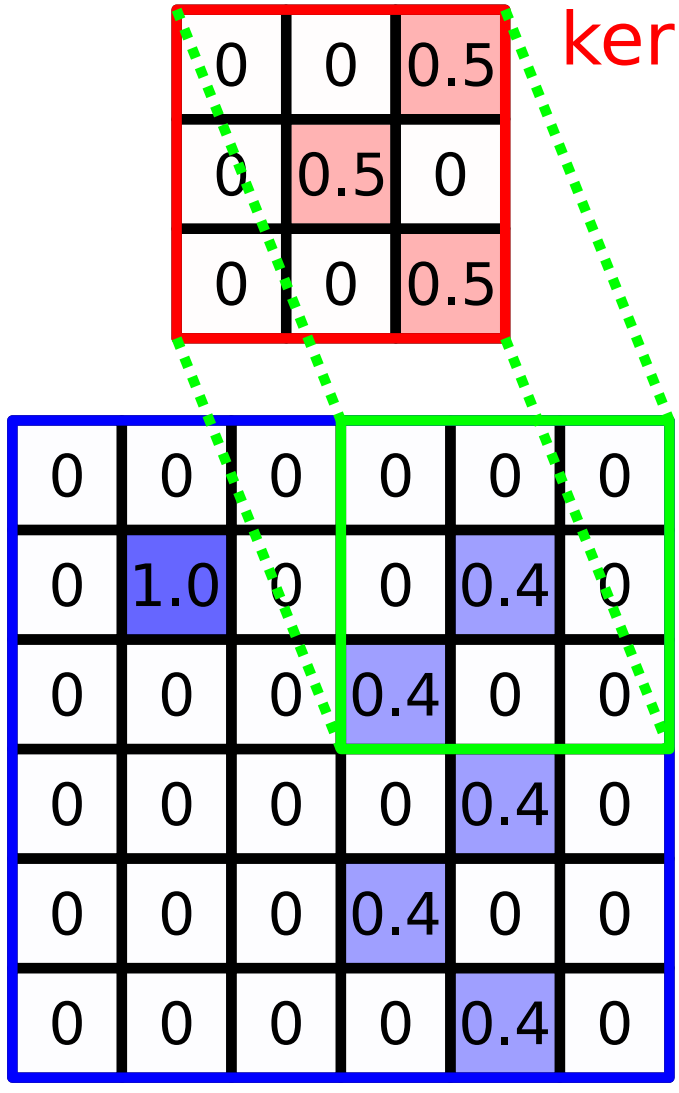


$$
\begin{aligned}
&0 \cdot 0 + \ 0 \cdot 0 \ + 0.5 \cdot 0 \\
&+ 0 \cdot 0 + 0.5 \cdot 0 + 0 \cdot 0.4 \\
&+ 0 \cdot 0 + 0 \cdot 0.4 + 0.5 \cdot 0 \\
&= 0
\end{aligned}
$$

output

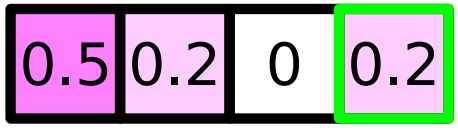


Weight Sharing (k: 3×3, i: 6×6, o: 4×4)

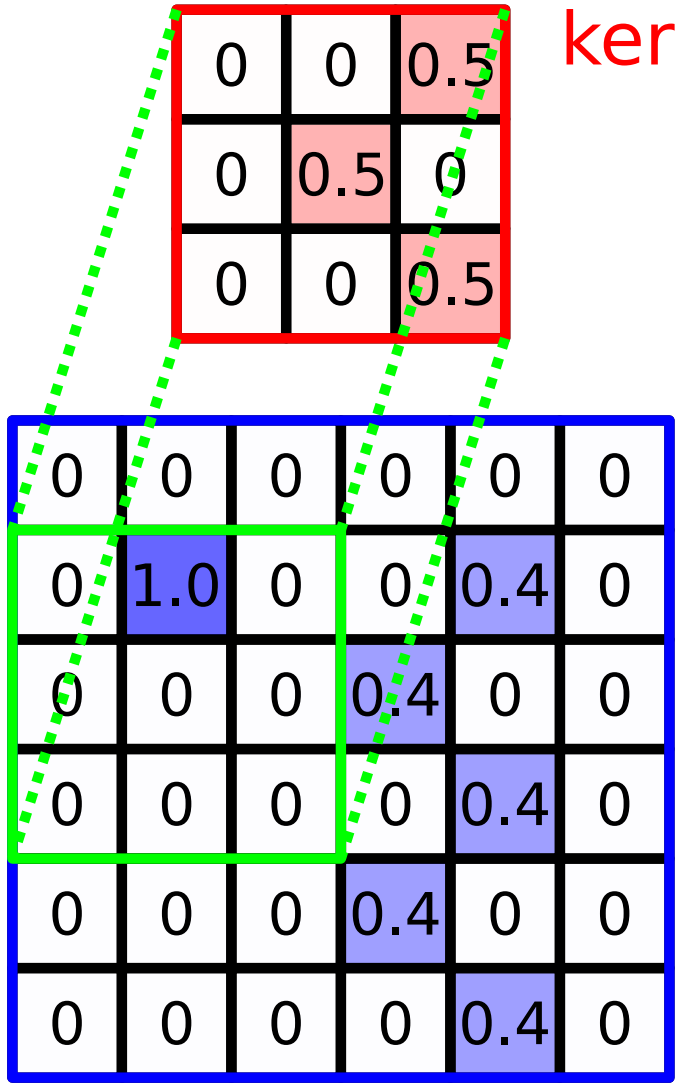


$$\begin{aligned} & 0 \cdot 0 + 0 \cdot 0 + 0.5 \cdot 0 \\ + & \ 0 \cdot 0 + 0.5 \cdot 0.4 + 0 \cdot 0 \\ + & \ 0 \cdot 0.4 + 0 \cdot 0 + 0.5 \cdot 0 \\ = & \ 0.2 \end{aligned}$$

output

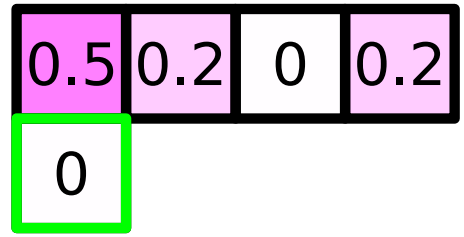


Weight Sharing (k: 3×3, i: 6×6, o: 4×4)

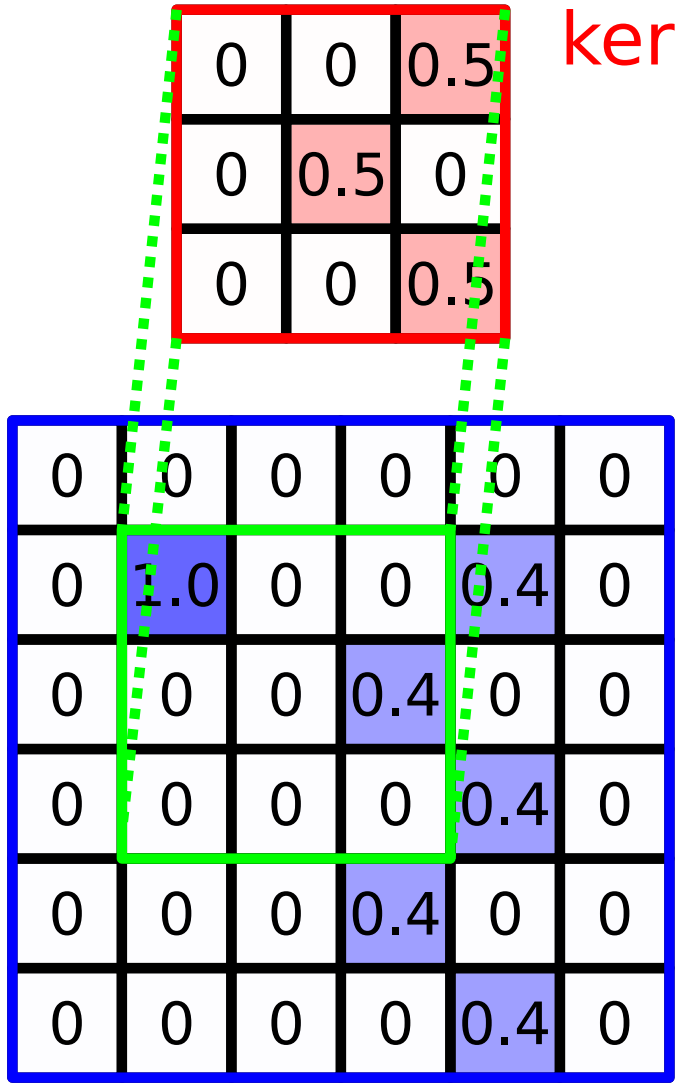


$$
\begin{aligned}
&0 \cdot 0 + 0 \cdot 1.0 + 0.5 \cdot 0 \\
+\,&0 \cdot 0 + 0.5 \cdot 0 + \ 0 \cdot 0 \\
+\,&0 \cdot 0 + \ 0 \cdot 0 \ + 0.5 \cdot 0 \\
=\,&0
\end{aligned}
$$

output



Weight Sharing (k: 3×3, i: 6×6, o: 4×4)

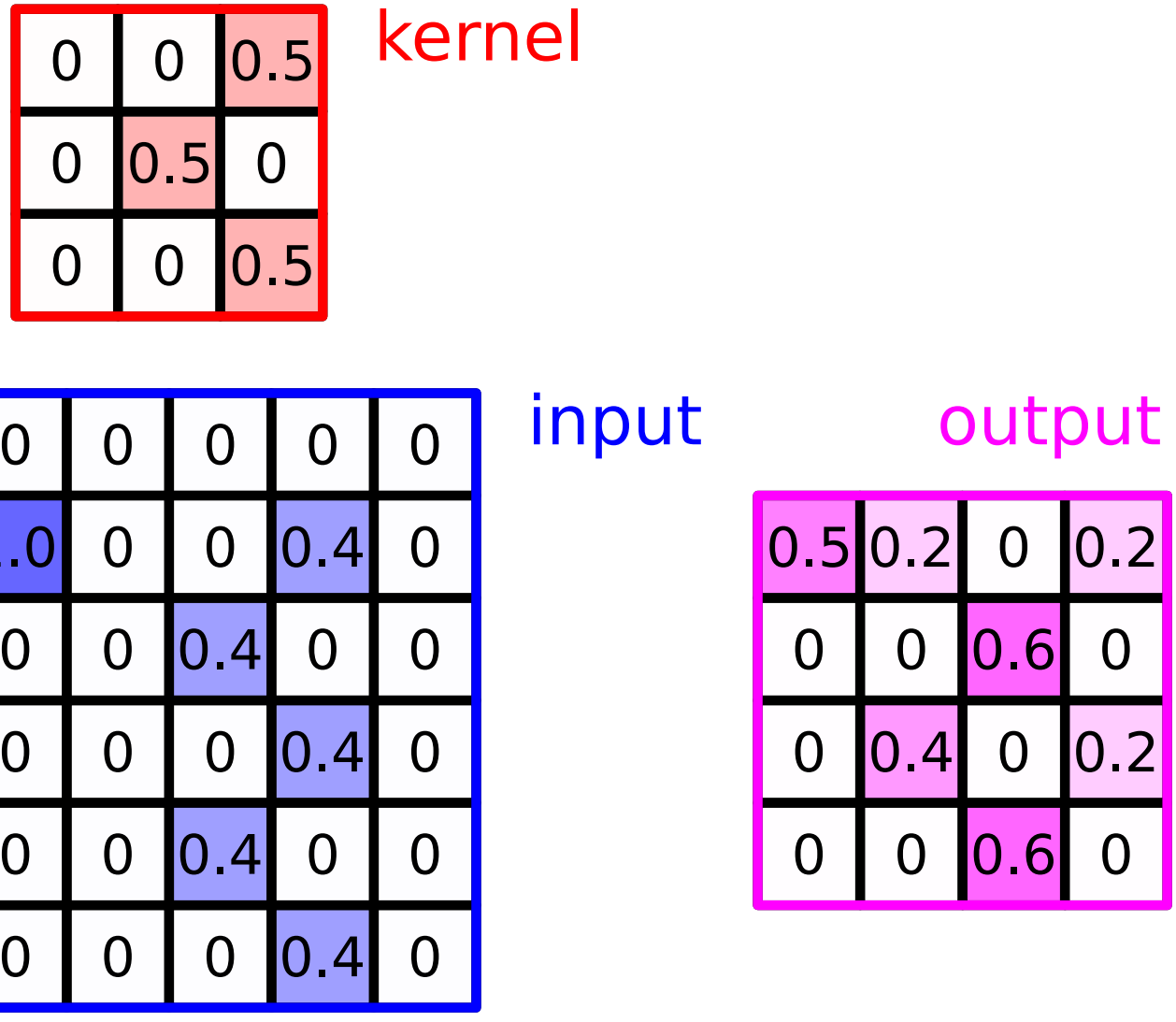


$$
\begin{aligned}
&0 \cdot 1.0 + \ 0 \cdot 0 \ + 0.5 \cdot 0 \\
+ \ &0 \cdot 0 \ + 0.5 \cdot 0 + 0 \cdot 0.4 \\
+ \ &0 \cdot 0 \ + \ 0 \cdot 0 \ + 0.5 \cdot 0 \\
= \ &0
\end{aligned}
$$

output

0.5	0.2	0	0.2
0	0		

Weight Sharing (k: 3×3, i: 6×6, o: 4×4)



Padding

- When applying a convolution with a kernel of size > 1, the output will be smaller than the input (see examples before).
- **Padding** is used to keep the input and output size the same:
  - **Zero-Padding**: Add zeros at borders of input
  - **Repeat-Padding**: Duplicate the border values
  - Other padding methods: Mean, weighted sum, ...
- Can be applied to input image or in between layers to keep the original input size

Striding

- **Striding** controls how much the kernels/filters are moved.
- The smaller the stride, the more the receptive filters overlap.
- Striding is one way of **downsampling** images.
- A stride > 1 will lead to loss of information (no problem if we keep the essential information) but will also reduce computational load and memory requirements.
- A stride > 1 will increase the receptive field through depth of network.

Pooling

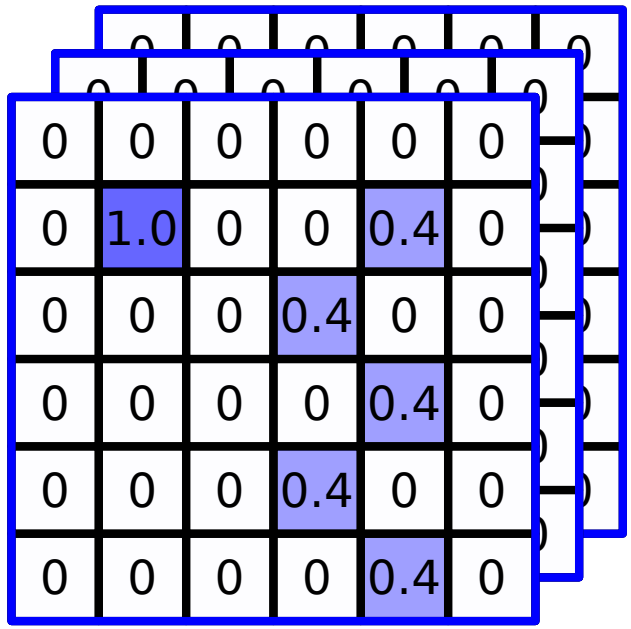
- Another way of **downsampling** images is **pooling**.
- There are different ways to perform pooling. Most popular:
  - **Average Pooling**: take the average value in a $k \times k$ field
  - **Max Pooling**: take the maximum value in a $k \times k$ field
  - **N-Max Pooling**: take the mean over the n maximum values in a $k \times k$ field
- Pooling will lead to loss of information (no problem if we keep the essential information) but will also reduce computational load and memory requirements.
- Pooling will increase the receptive field through depth of network.
- Pooling is a fixed operation compared to “strided” convolutions, i.e., there are no parameters to learn.

Inputs in CNNs

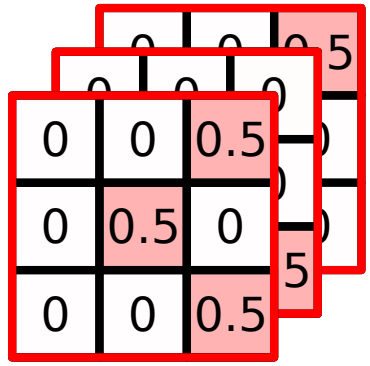
- Until now, we assumed grayscale images with 1 channel.
- RGB images have 3 **channels** for red, green, blue, typically stored in a shape of (width, height, 3).
- After the convolutional operations, channels are also called **feature maps** or **activation maps**.
- We need to make sure our kernel matches the number of channels (e.g., if the input is 3D, the kernel must be 3D as well).
- Regardless of the number of channels, a single feature map/channel will be produced (it just computes the sum of the channel-wise convolutions).

Inputs in CNNs

input (3 channels)

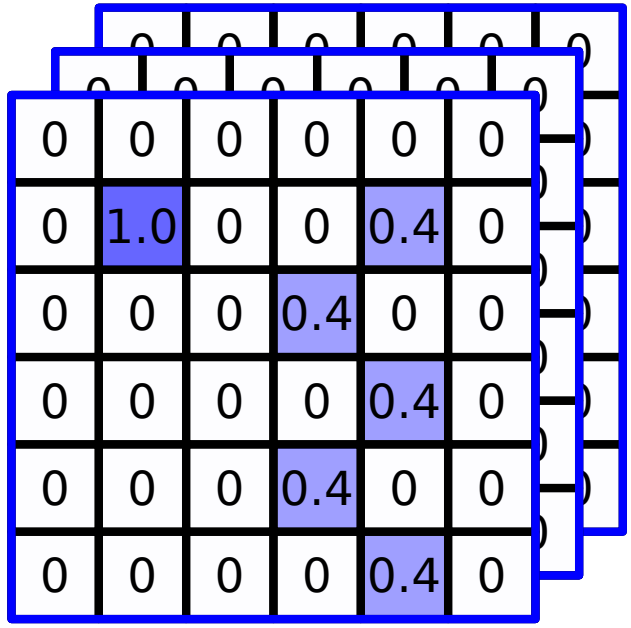


Inputs in CNNs

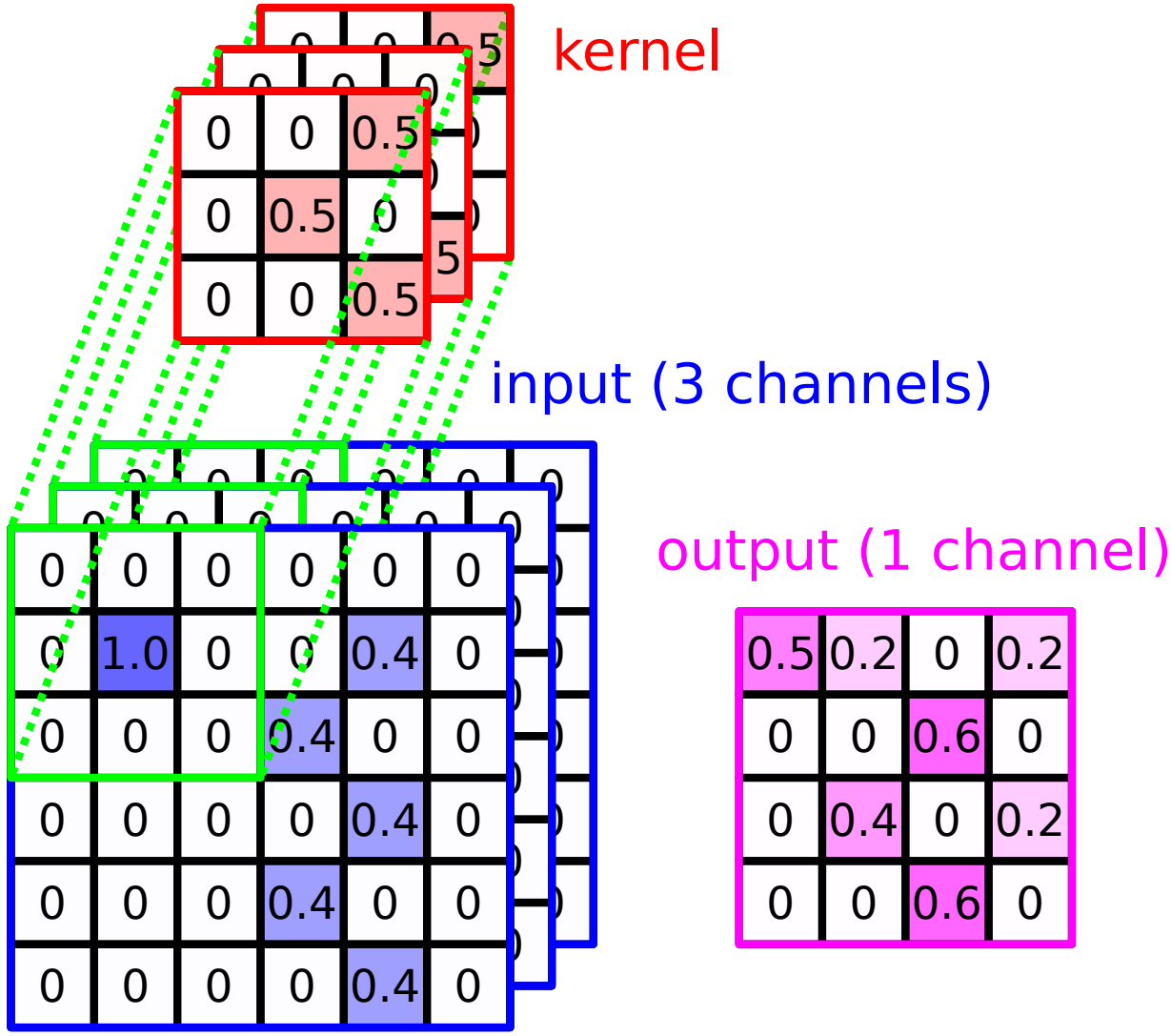


kernel

input (3 channels)



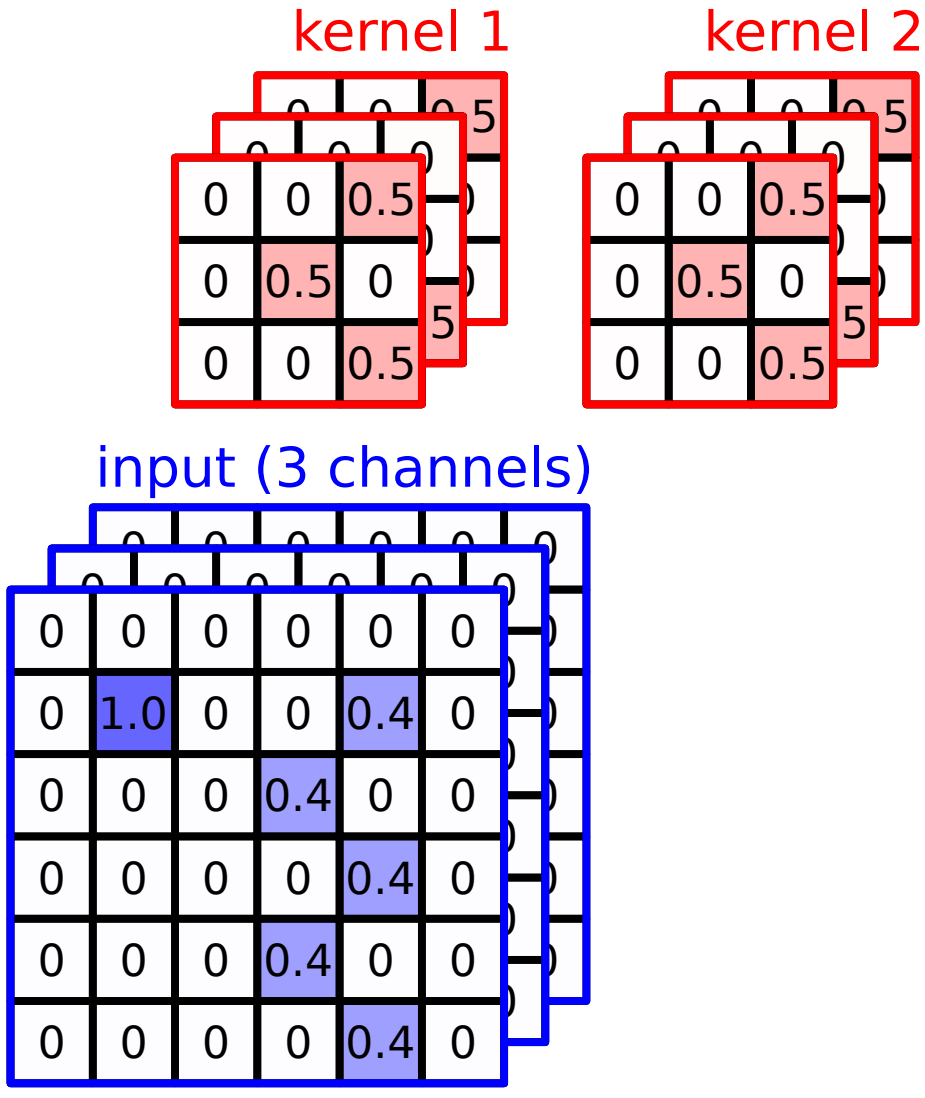
Inputs in CNNs



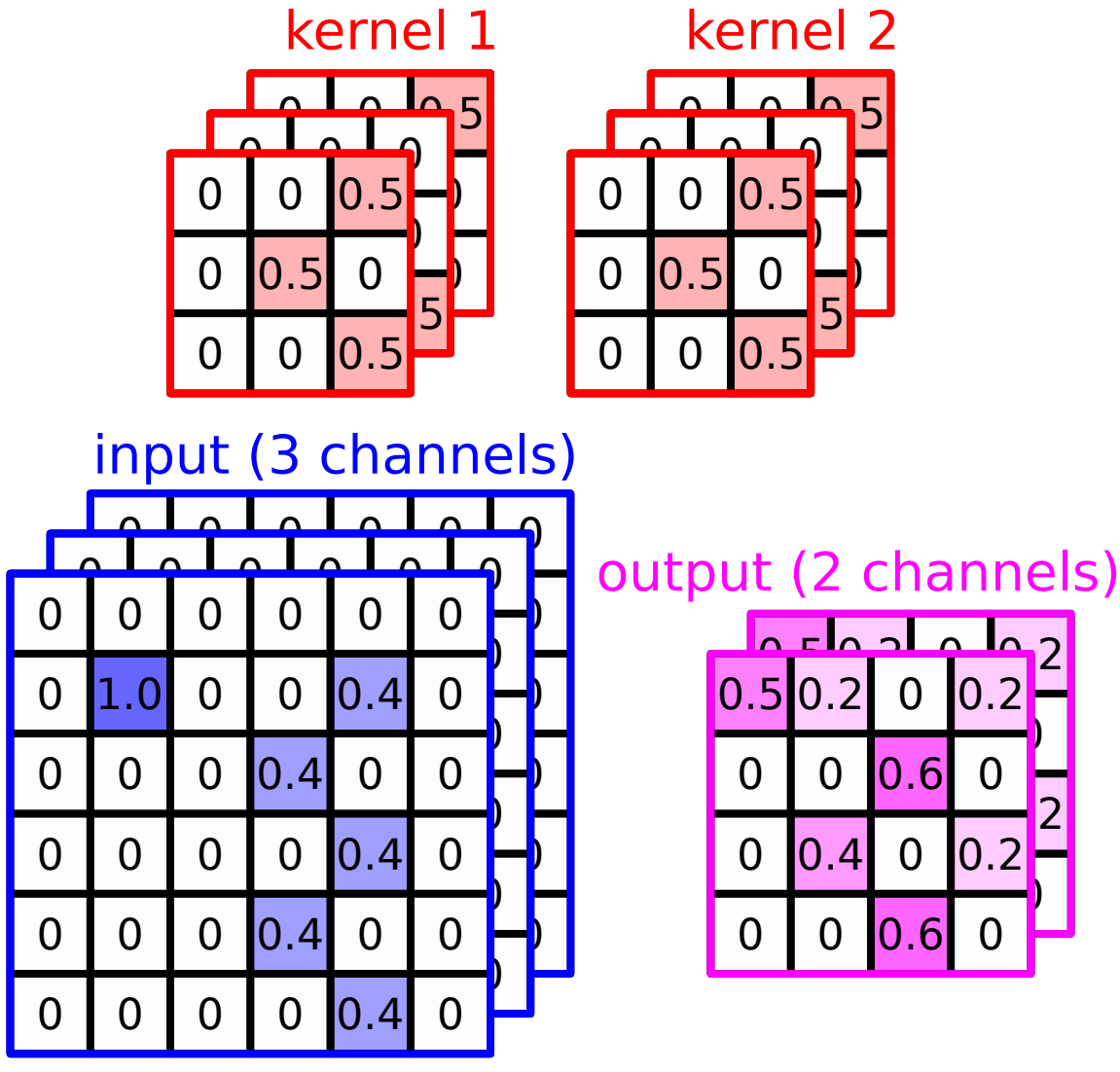
Outputs in CNNs

- We usually want to apply **multiple kernels** to the image.
- For each (multi-dimensional) kernel, we create a **feature map/channel** in the CNN output.

Outputs in CNNs



Outputs in CNNs



Multiple Levels of Convolutions

- A typical CNN architecture has **several layers** of:
  1. Convolution
  2. Non-linearity
  3. Pooling (optional)
- Each kernel produces a new feature map for the next layer.
- The complexity of detected features tends to increase layer by layer (e.g., first feature map does edge detection, later ones combine it to complex shapes).
- Depending on the task, we might need to perform some additional operations after the convolutions. E.g., for image classification, we flatten the final CNN output and use it as input for a regular neural network.
- Interactive CNN visualization demo:
  https://poloclub.github.io/cnn-explainer/

Ways to Improve a Network

- There are a couple of tricks that we can use to (potentially) improve the performance of our neural network models:
  - Data augmentation
  - Dropout
  - Batch normalization
  - Transfer learning
  - Deep networks
  - Learning rate schedules
- Note: There is no guarantee that these will improve the performance.