Introduction to Deep Learning

Deep Learning in Radiotherapy Course

Mitko Veta

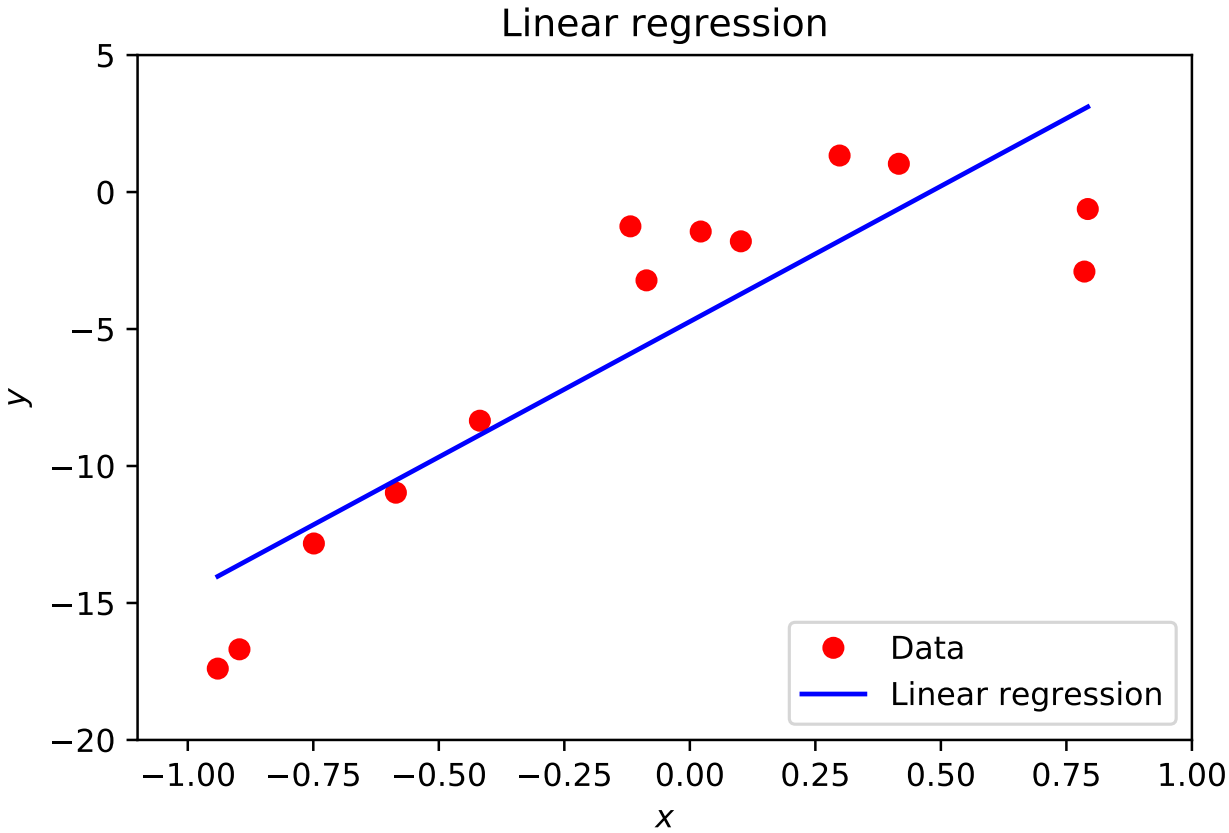
Eindhoven University of Technology
Department of Biomedical Engineering

2024

Learning goals

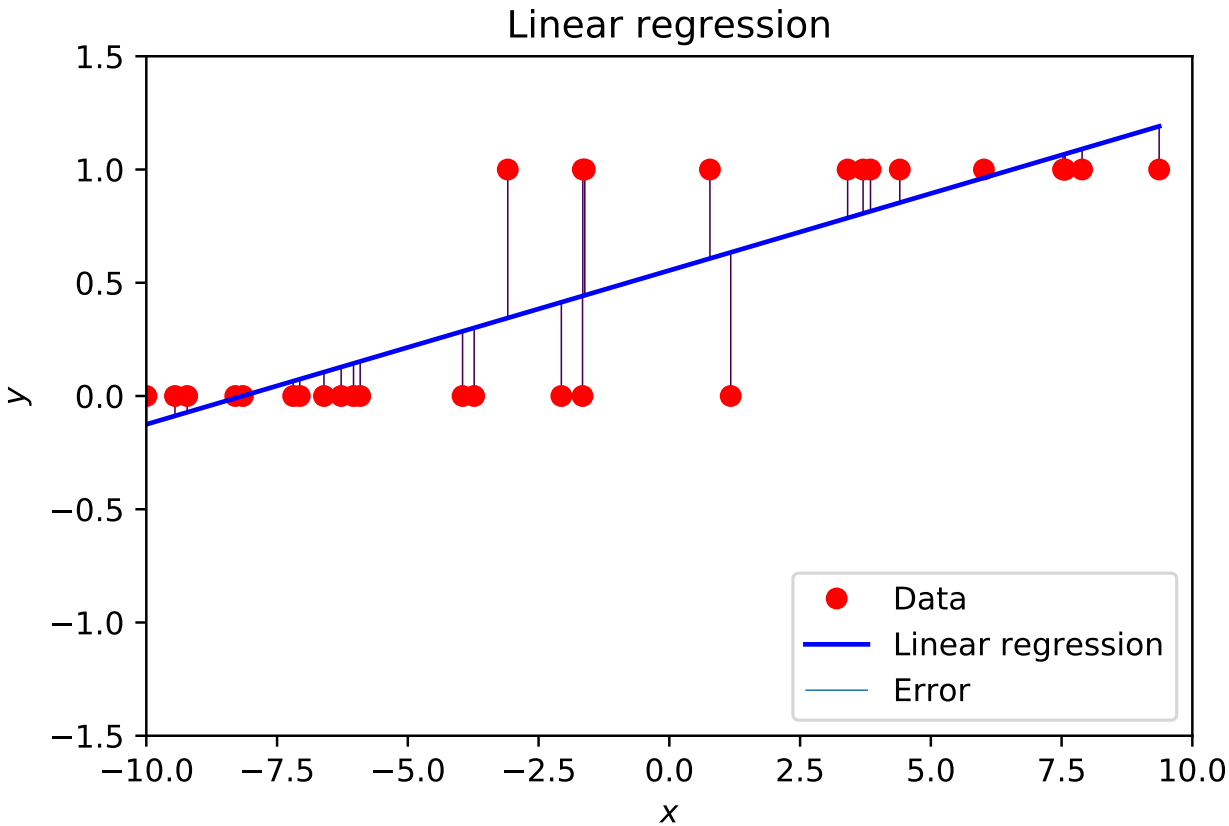
- Develop an understanding of how linear classification models can be combined to create more advanced (i.e. non-linear) models.
- Gain familiarity with the concept of neural networks as layered structures.
- Develop an understanding of how deep neural networks can be modified to be more suitable for image data processing.

Previously: linear model for regression



$$\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

Linear regression for a binary target



$$\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

Towards a linear model for classification

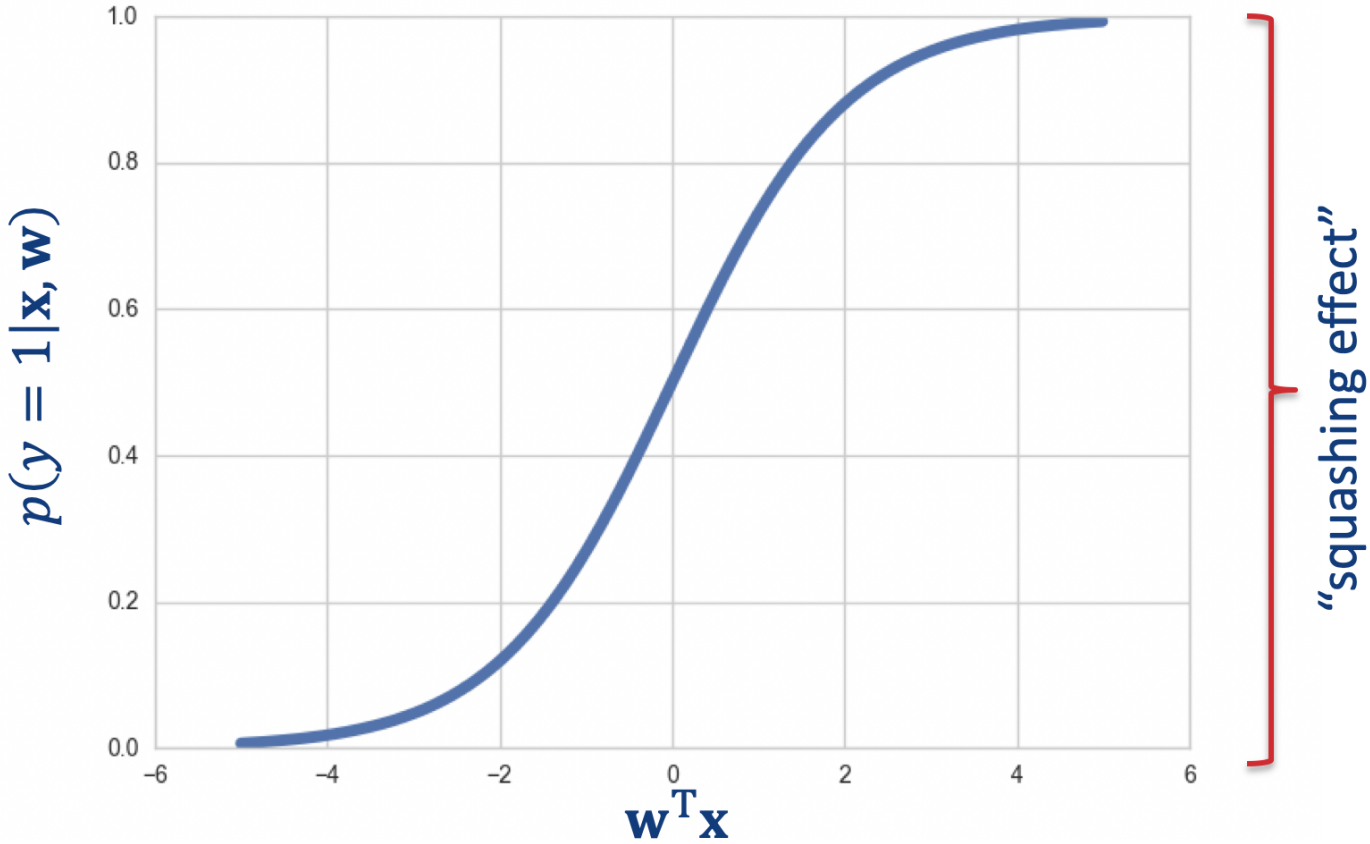
$$\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

The following changes need to be made to the linear model:

- Instead of directly predicting the value $\hat{y}$ (which is a categorical variable), predict the probability that the sample belongs to one of the classes, e.g. $p(y_i = 1|\mathbf{x}_i)$.
- $p(y_i = 1|\mathbf{x}_i)$ is a continuous value, however, it is bounded between 0 an 1.
- In order to interpret is as a probability, $\hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$ has to be "squashed" between 0 and 1.

The sigmoid function

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



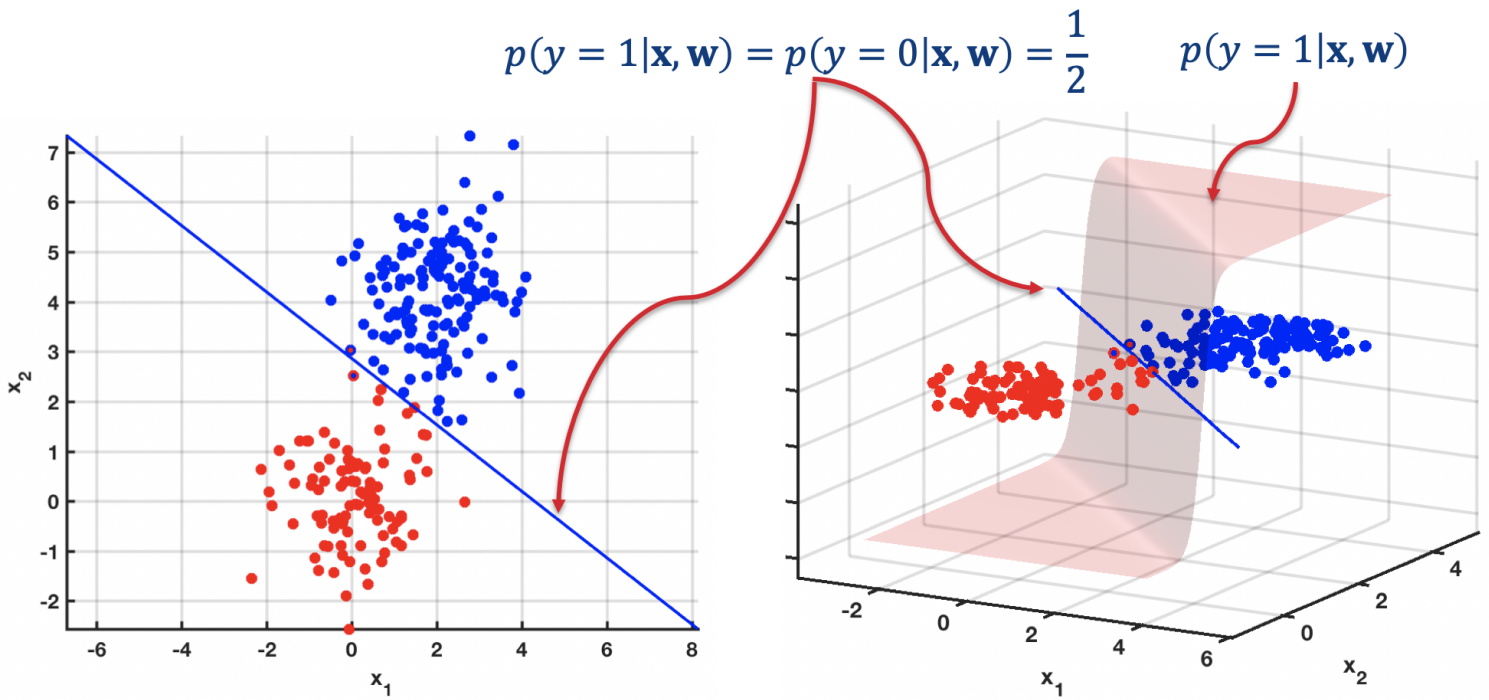
Logistic regression: a linear model for classification

Linear regression: $\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$

Logistic regression: $p(y_i = 1|\mathbf{x}_i) = \sigma(\hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j)$

Logistic regression: a linear model for classification

Logistic regression produces a linear decision boundary:



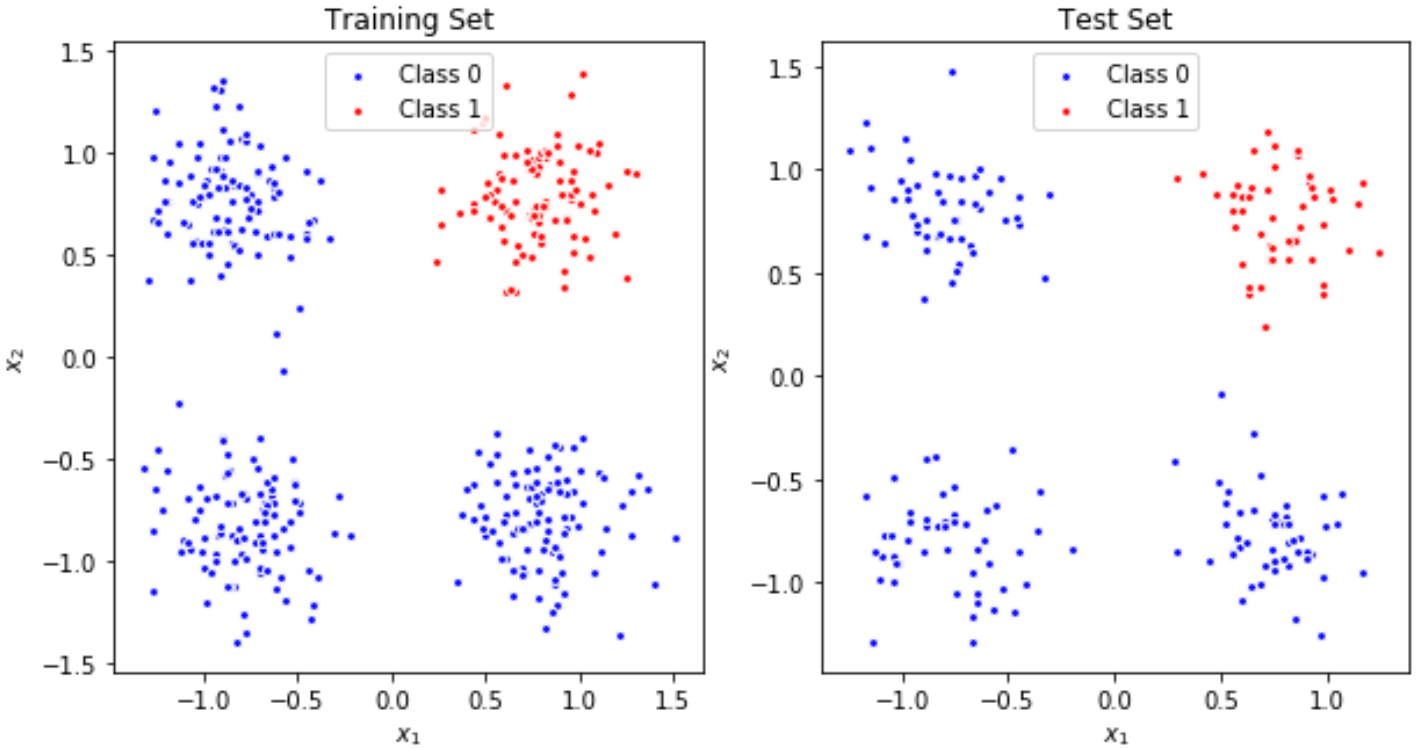
Softmax regression

The extension of logistic regression to multi-class problem (more than two classes) is straightforward and it is called softmax regression.

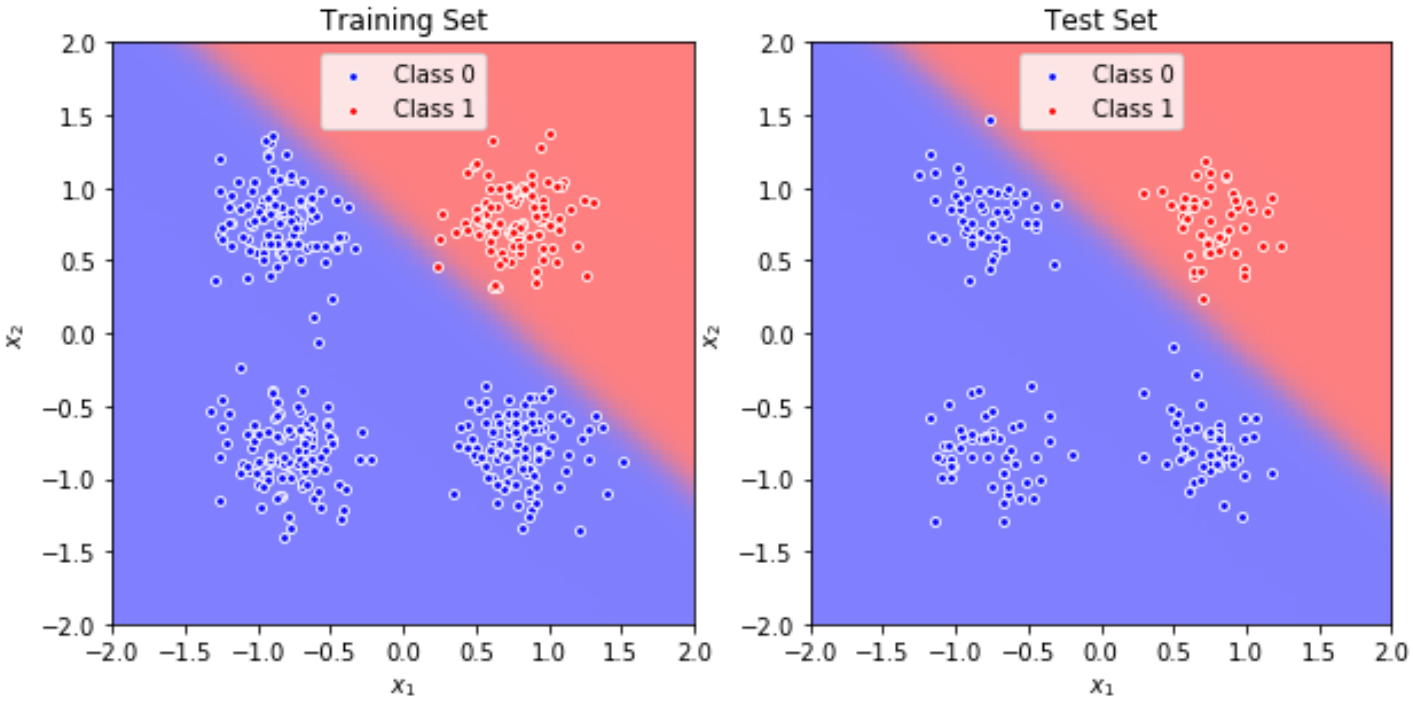
Essentially, for C number of classes, C number of linear models are and are converted to probabilities using the softmax function:

$$p(y = i|\mathbf{x}) = \frac{e^{\mathbf{x}^\intercal \mathbf{w}_i}}{\sum_k^C e^{\mathbf{x}^\intercal \mathbf{w}_k}}$$

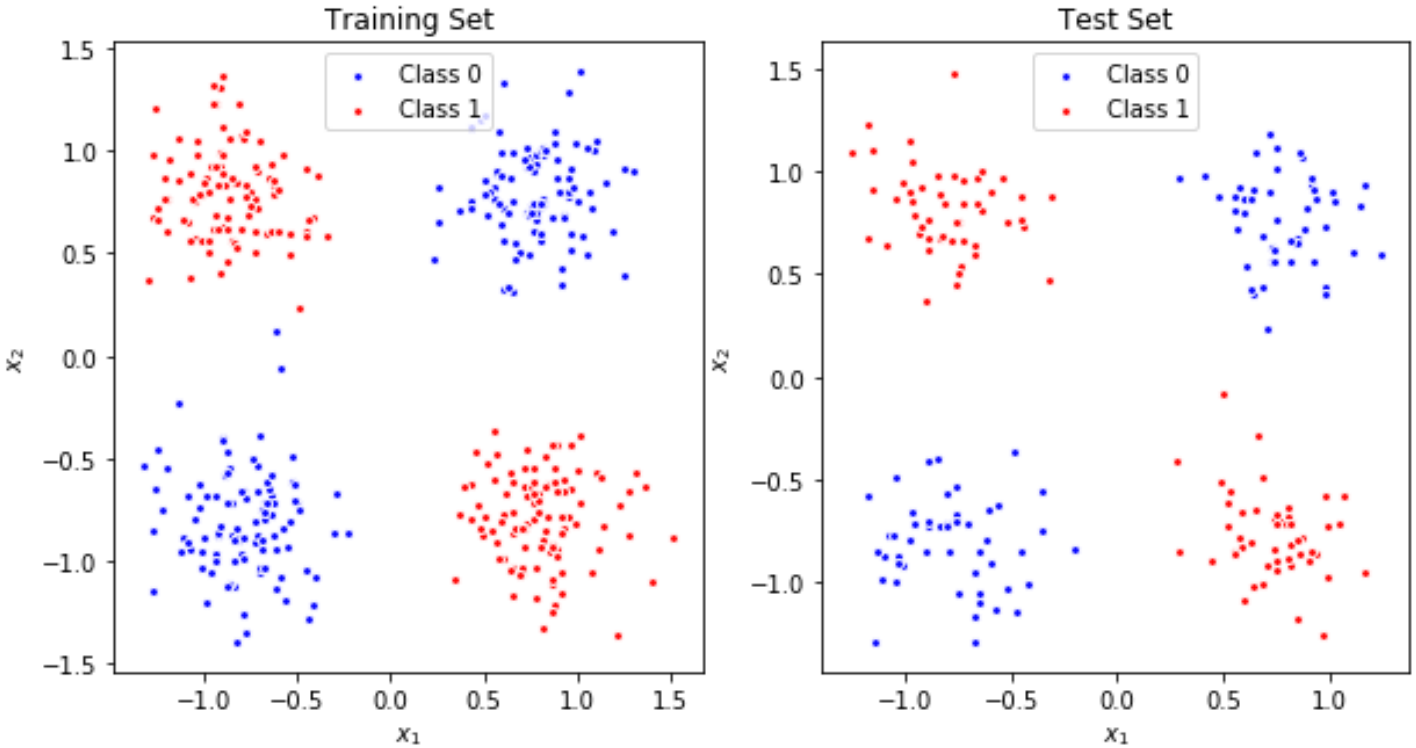
The AND classification problem



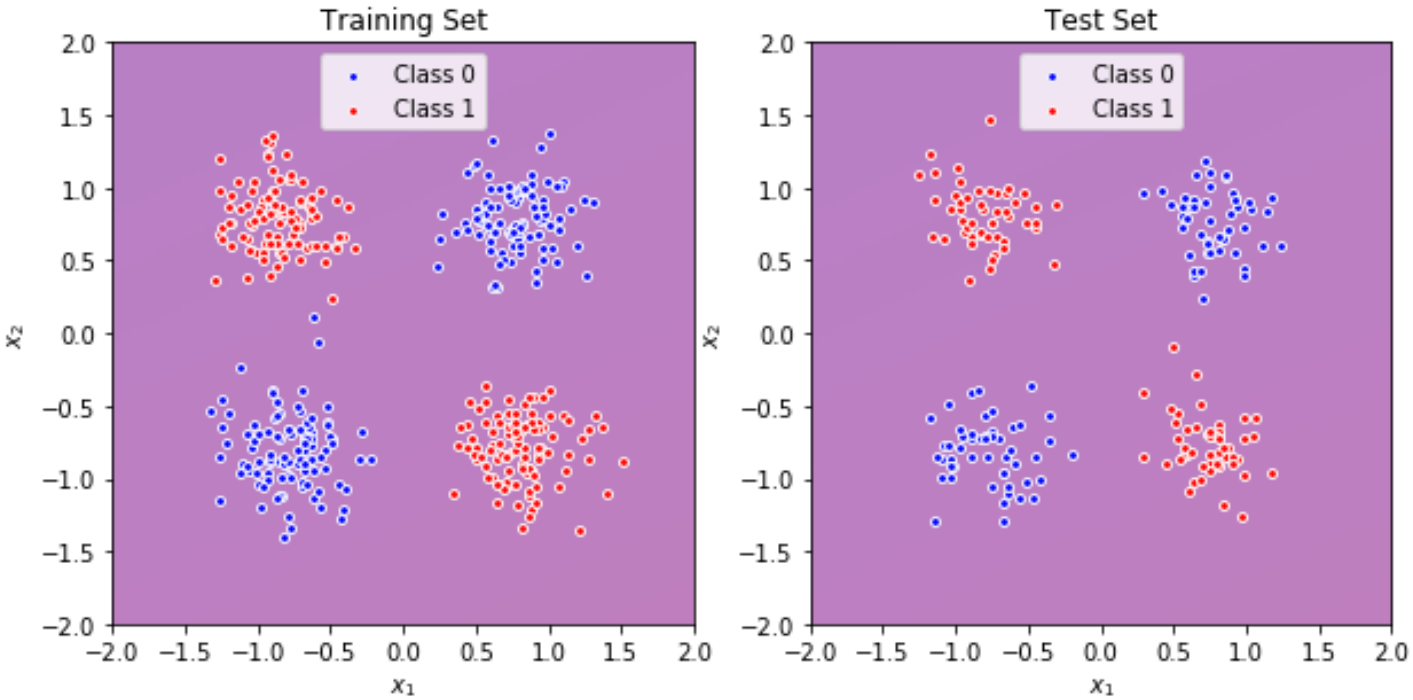
The AND classification problem: logistic regression



The XOR classification problem

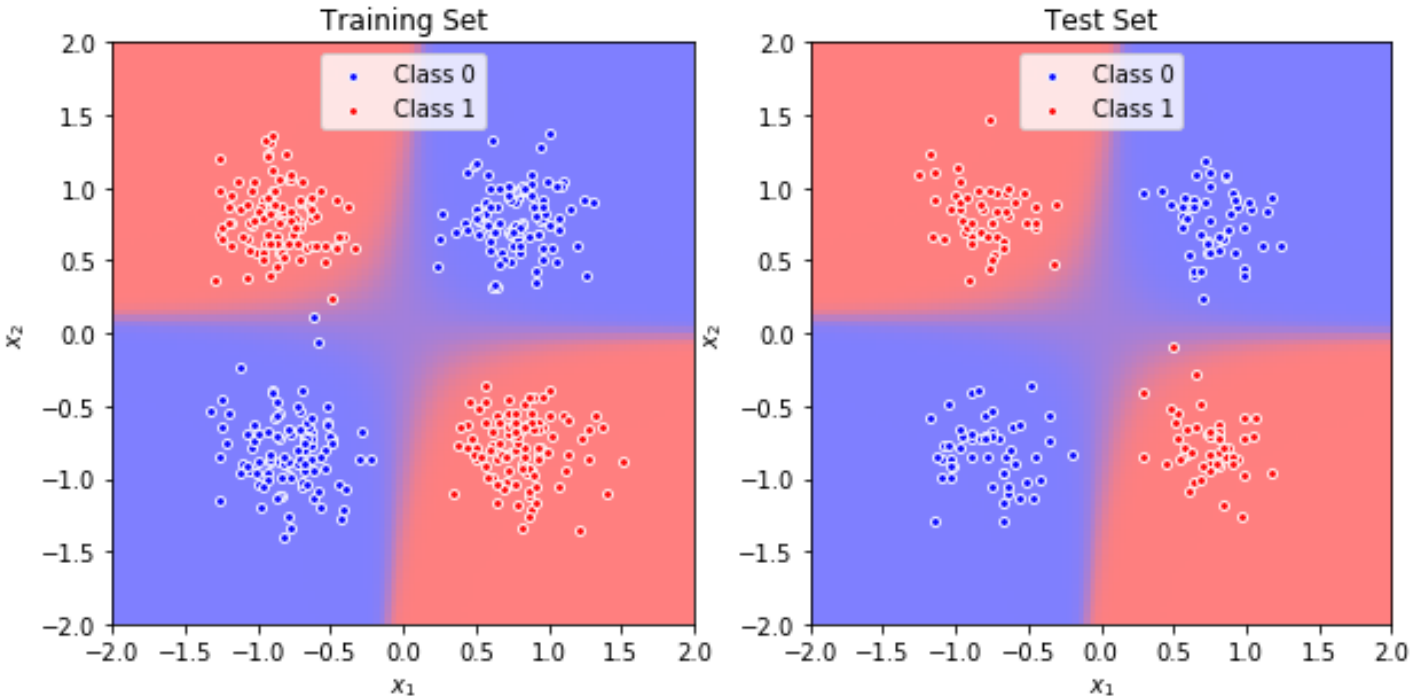


The XOR classification problem: logistic regression



The XOR classification problem: LR + polynomial transformation

$$\mathbf{x}_i = [x_1, x_2] \rightarrow \tilde{\mathbf{x}}_i = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$$



Feature transformation

Prior to 2010, most of the work on machine learning involved *relatively* simple classifiers in combination with extensive, "manual" feature engineering.

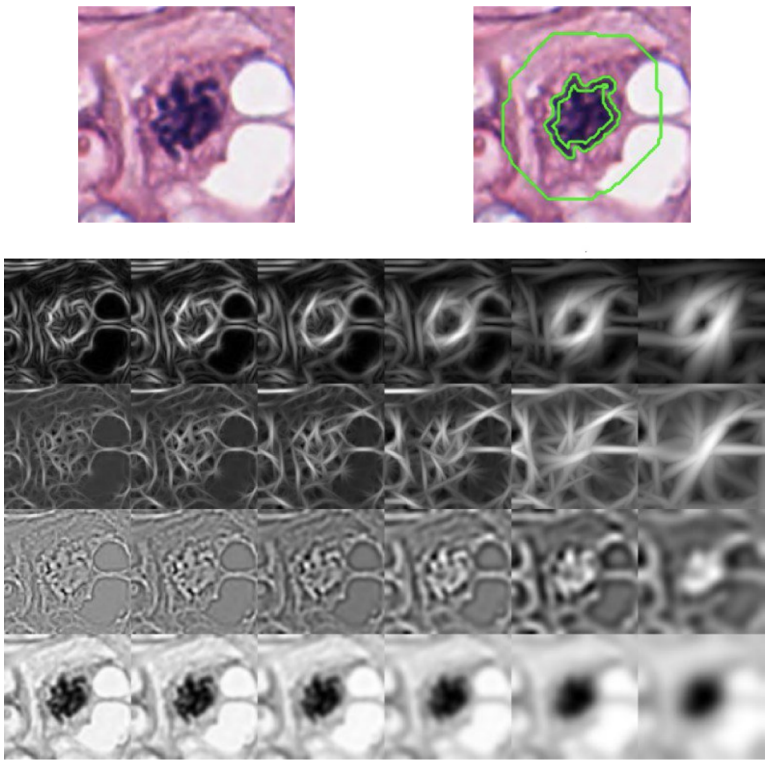
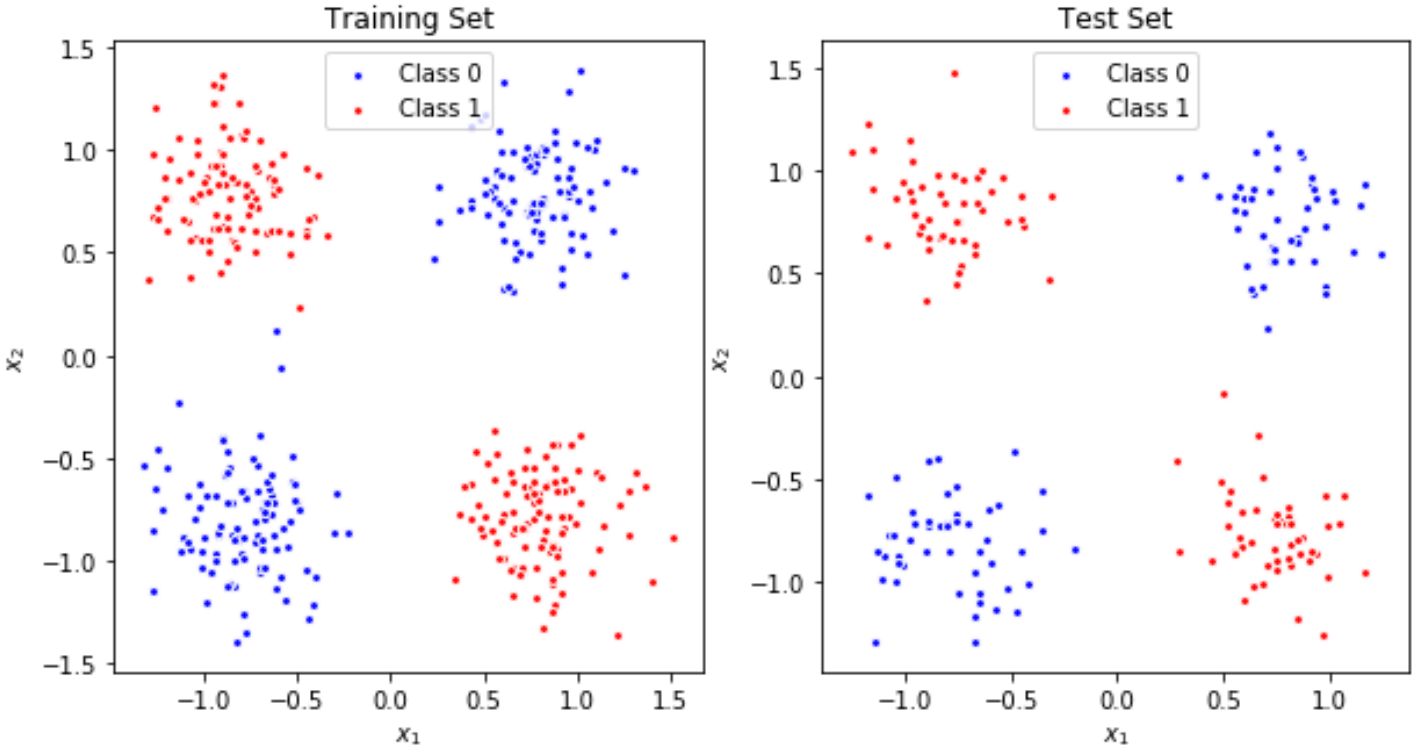


Figure source: Veta et al. SPIE MI 2012

A change of paradigm

Feed **raw** (or minimally processed) images directly to the machine learning models. Design the models in such a way that they can **learn the needed feature transformations** directly from the image data.

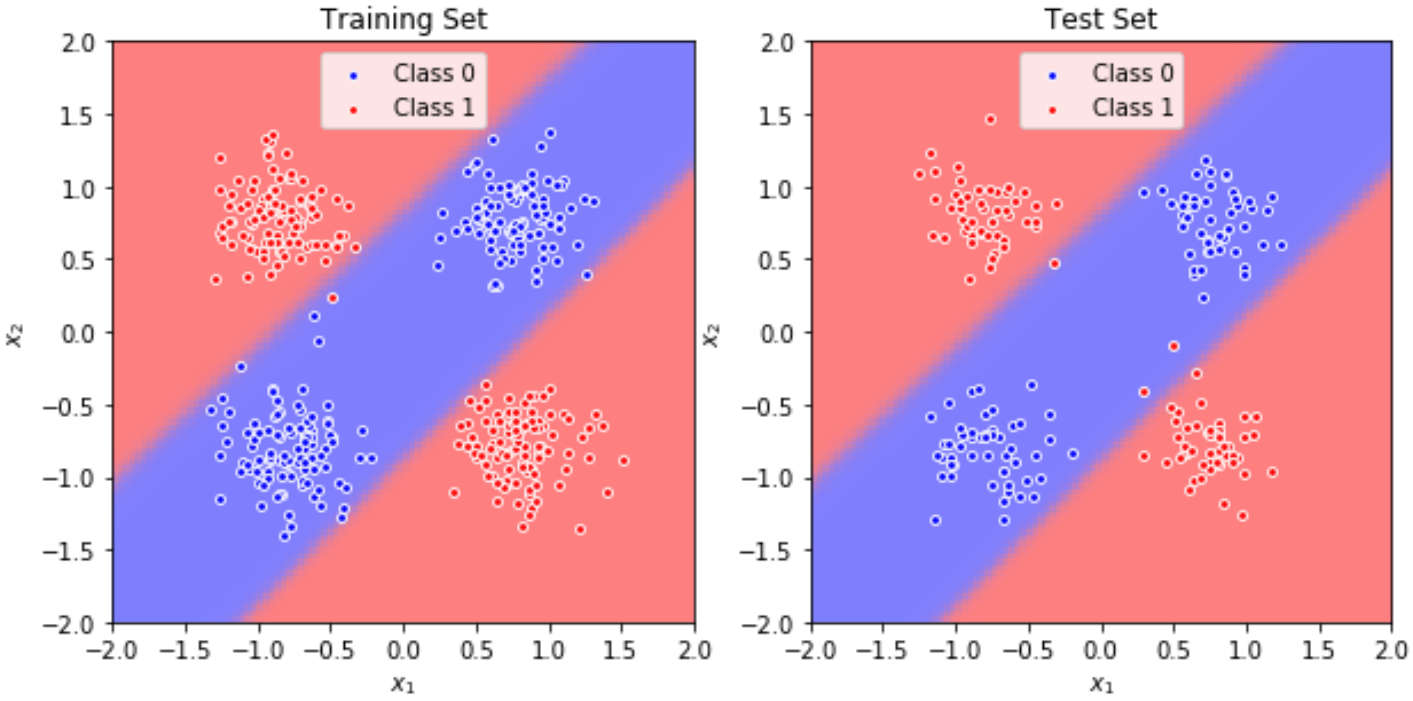
The XOR classification problem



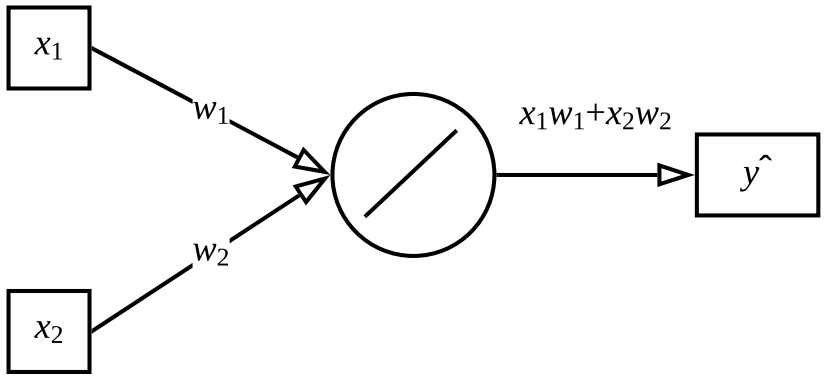
Question: How can the logistic regression classifier be modified to solve the XOR problem without explicit feature transformation?

Combining two linear classifiers

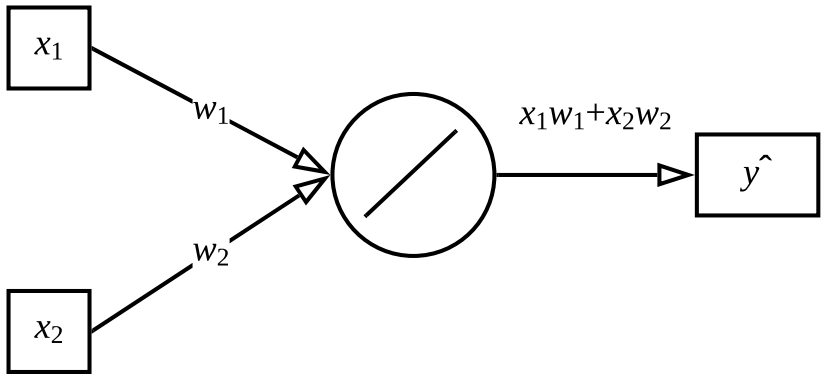
Answer: Use two logistic regression classifiers.

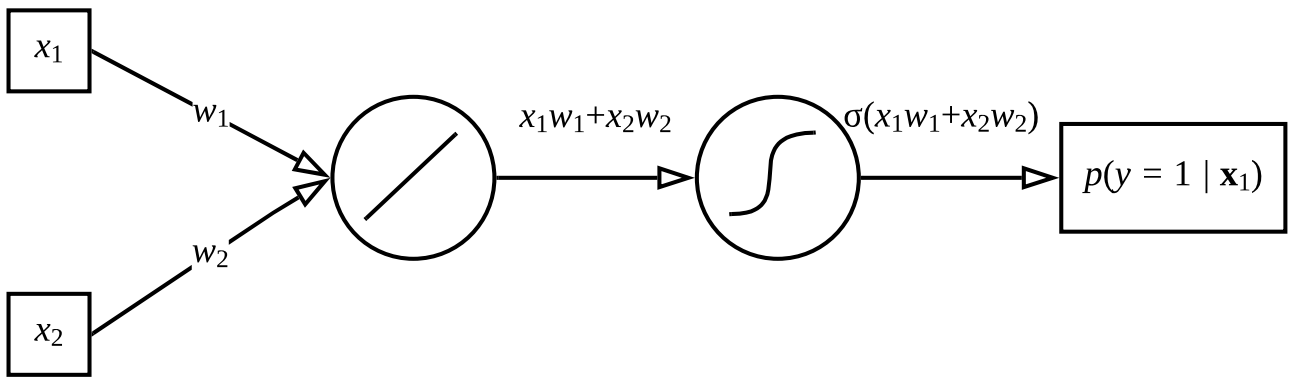


A graphical view of linear and logistic regression

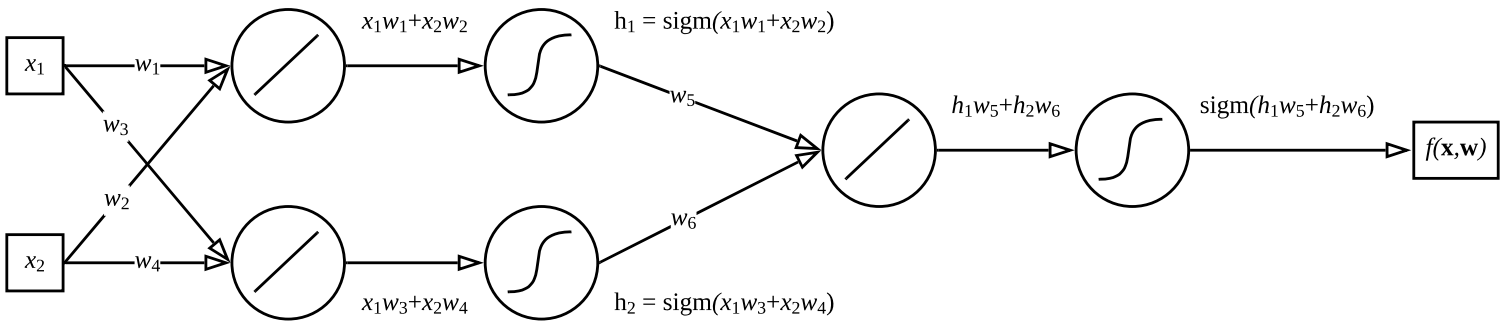


A graphical view of linear and logistic regression

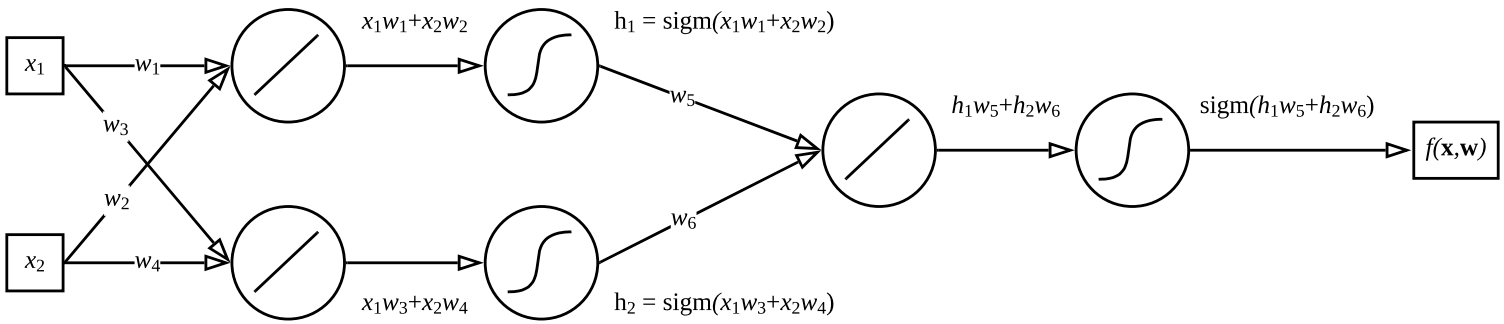




Combining two linear classifiers

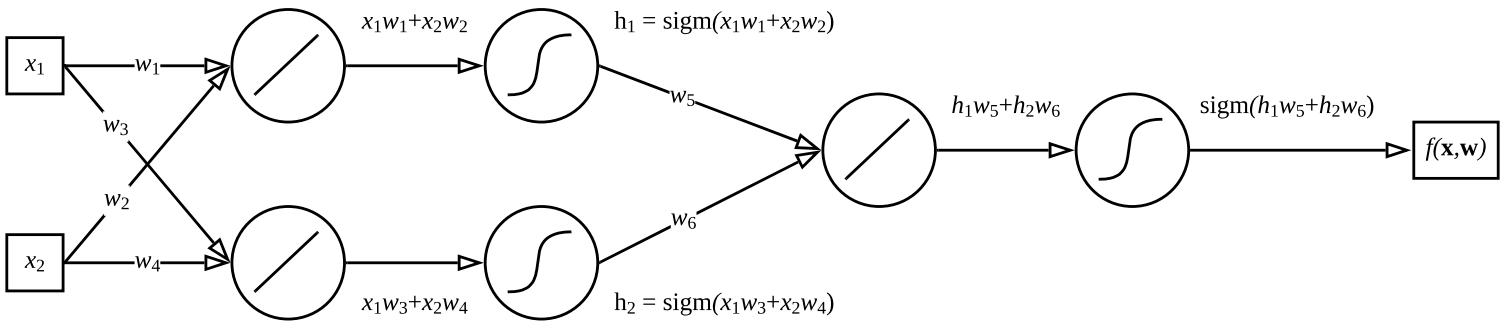


Combining two linear classifiers



This is a (small) feedforward neural network.

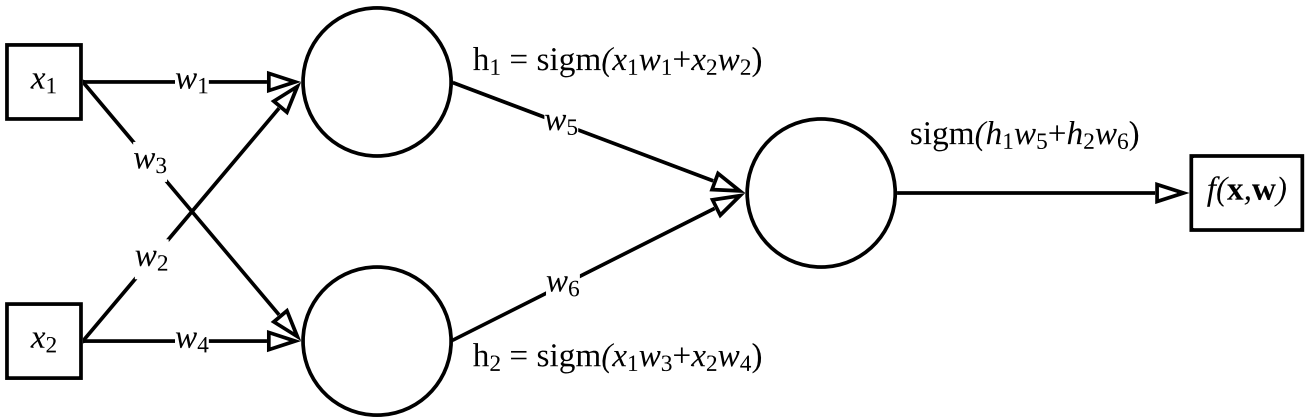
Combining two linear classifiers



This is a (small) feedforward neural network.

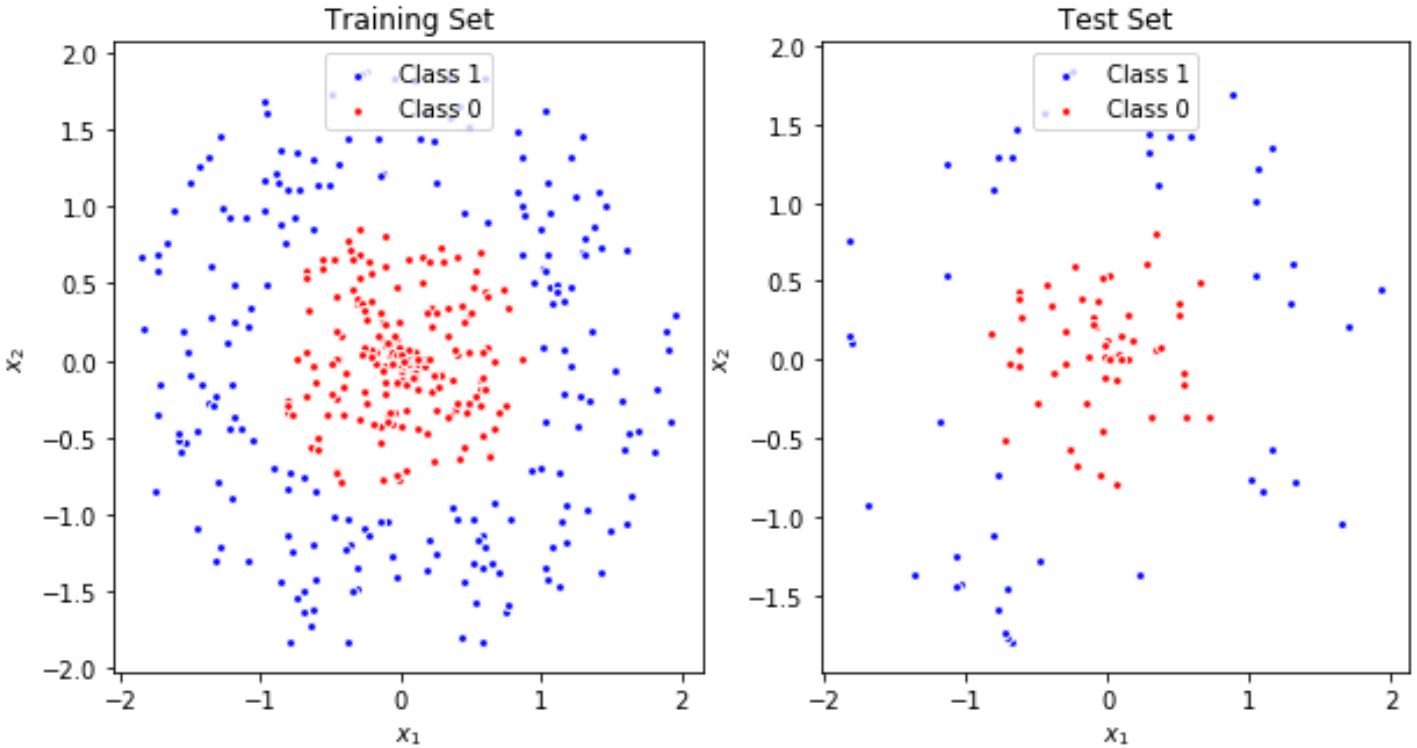
It is a composition of two functions $\mathbf{h}(\mathbf{x})$ and $f(\mathbf{x})$. $\mathbf{h}(\mathbf{x})$ is called a hidden layer. It can be seen as a **learned** feature representation of $\mathbf{x}$ (analogous to the hand-crafted features $\tilde{\mathbf{x}}$).

Combining two linear classifiers

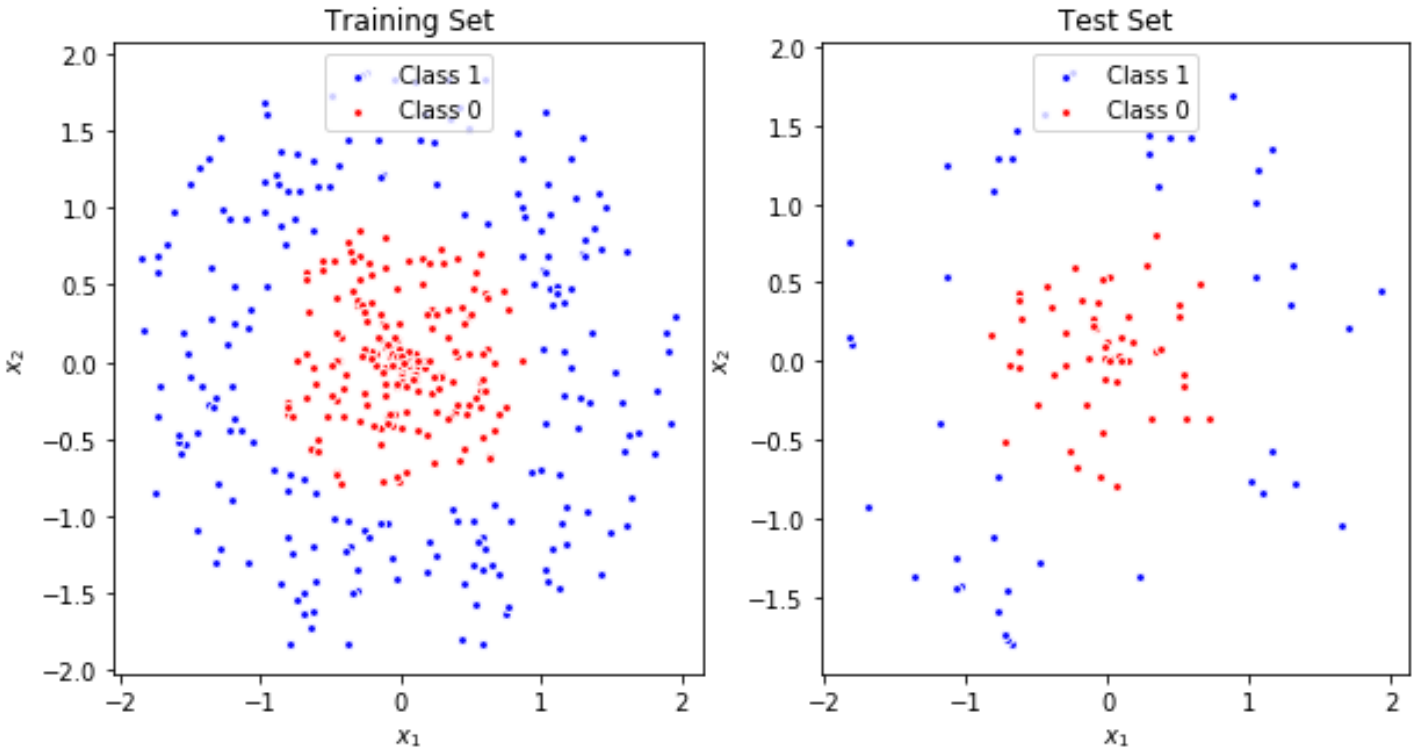


Often the sigmoid nonlinearity is not depicted in graphical representations (but it is there, and as shown later, it is crucial).

A somewhat more difficult problem

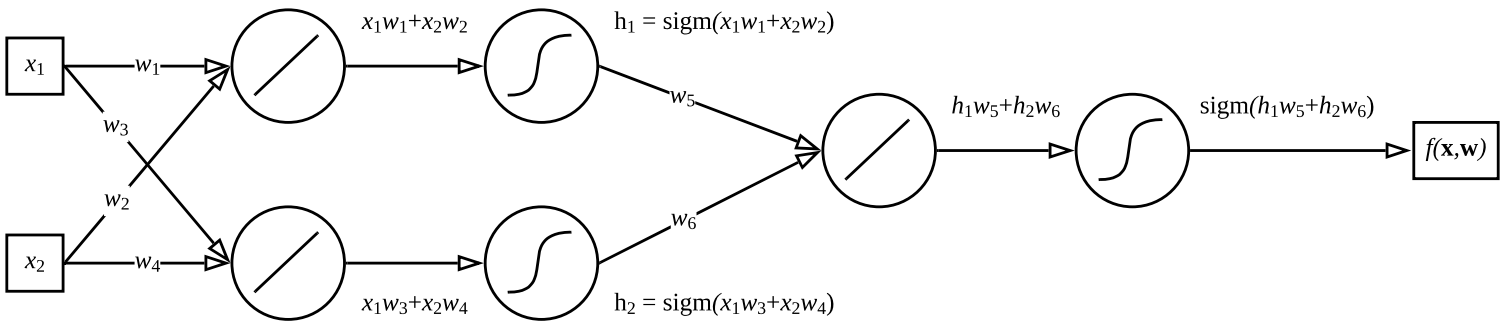


A somewhat more difficult problem



Example in Tensorflow Playground.

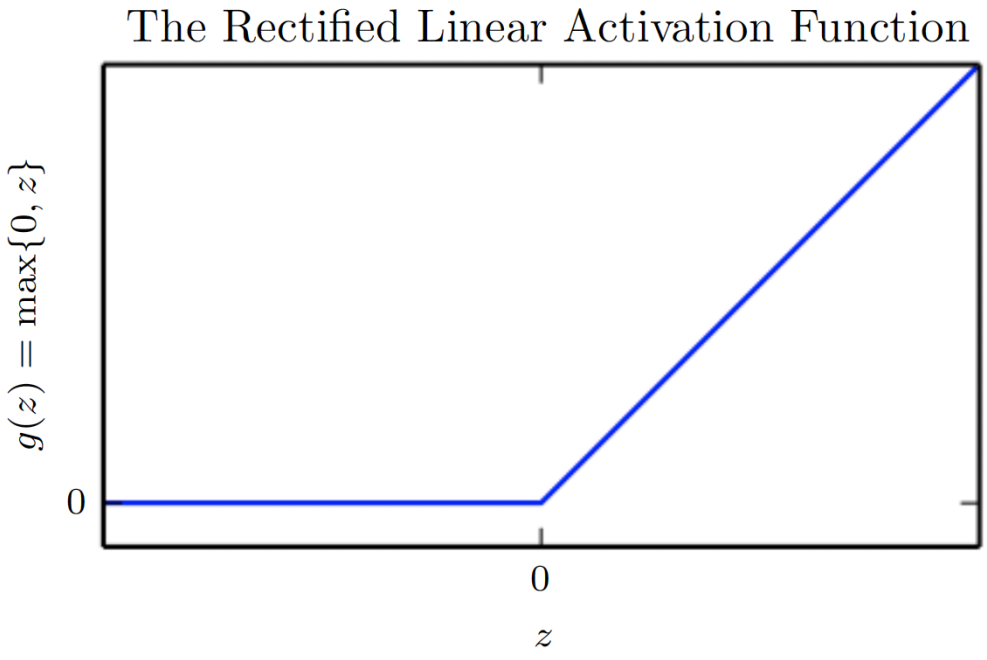
Why do we need nonlinearities



Without the nonlinearities, the network will be a linear combination of linear combinations of the input features, which collapses to a linear combination of the input features.

In other words, it will be no different than just logistic regression. **The network has no depth.**

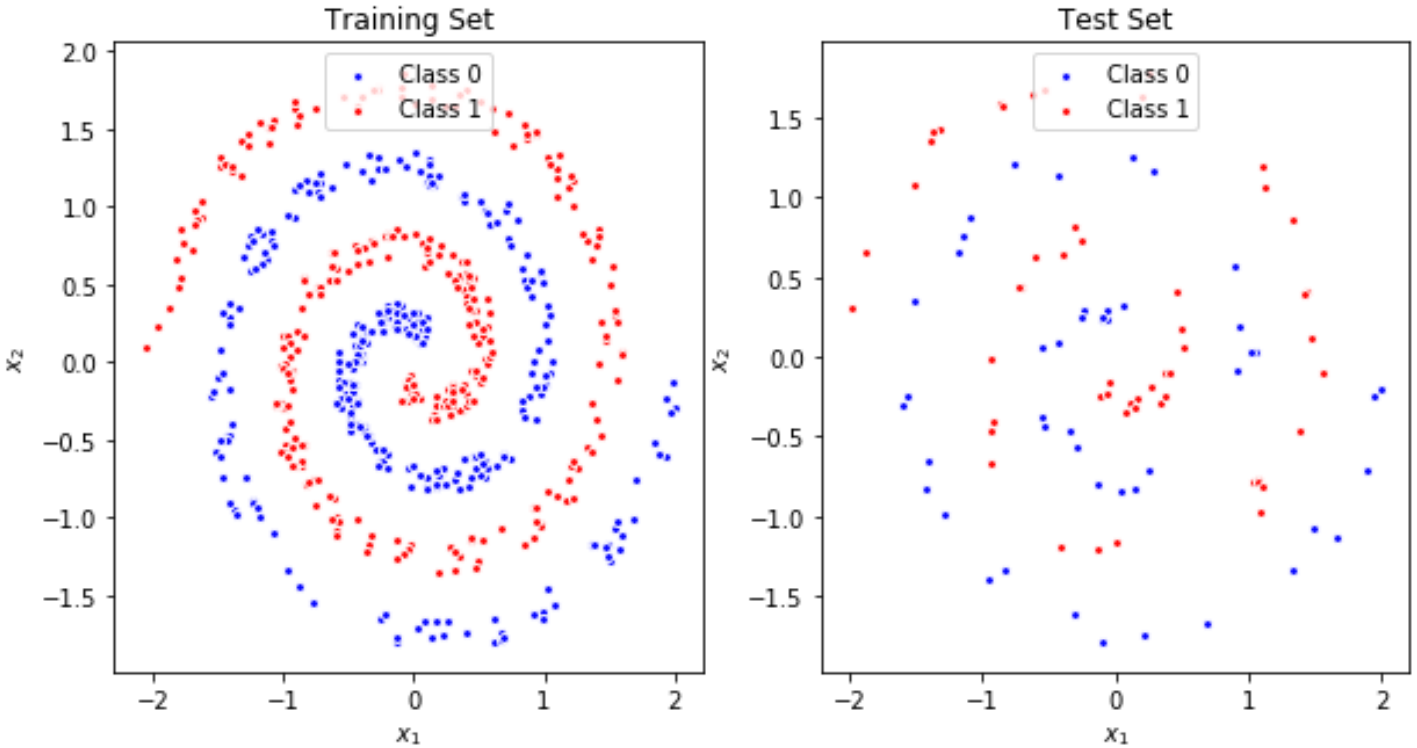
The ReLU nonlinearity



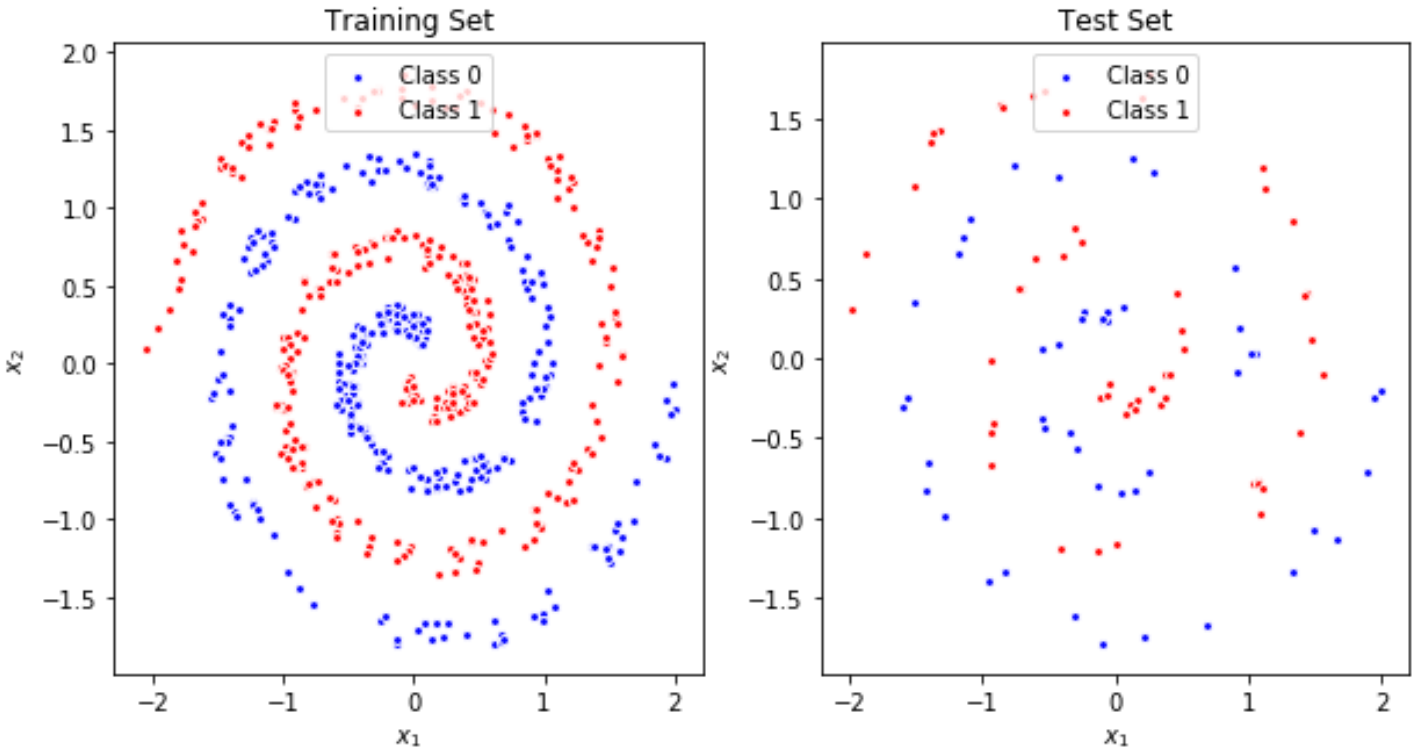
Instead of a sigmoid, the recommended default nonlinearity in modern neural network is the rectified linear unit. Because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimise with gradient-based methods.

Figure from: deeplearningbook.org

An even more difficult problem

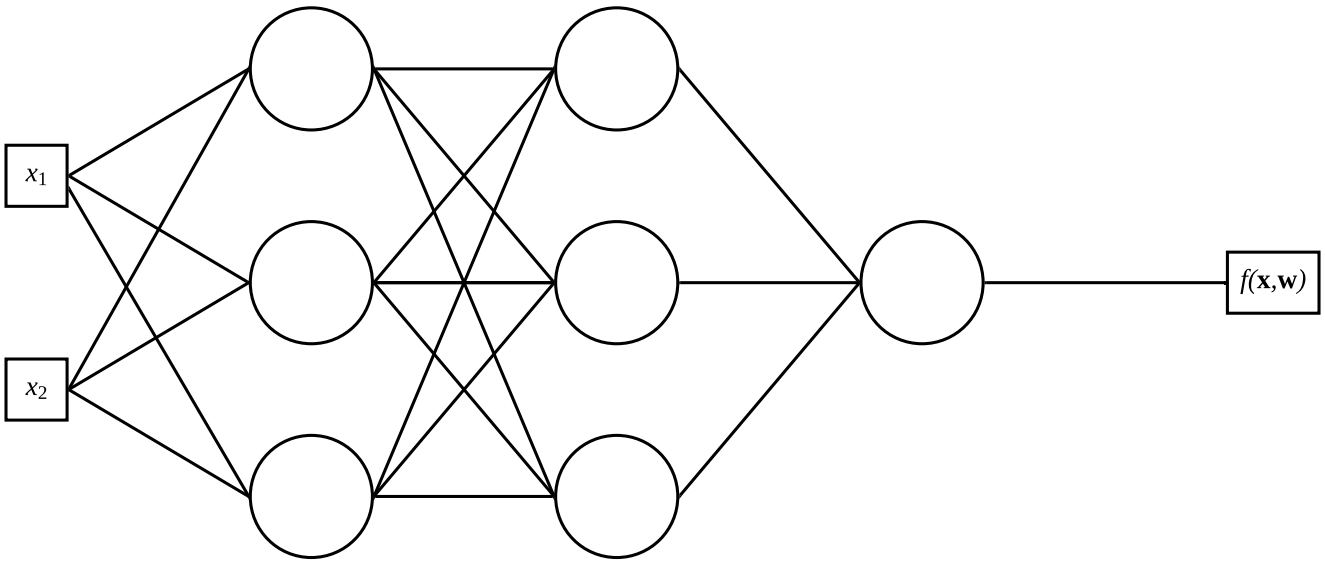


An even more difficult problem

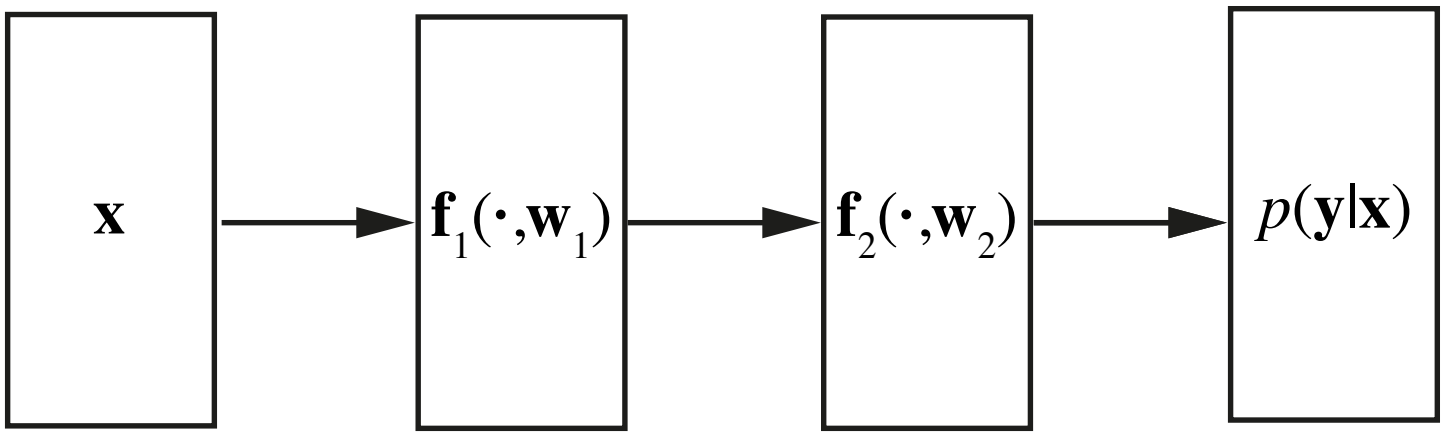


Example in Tensorflow Playground.

Multilayer neural networks



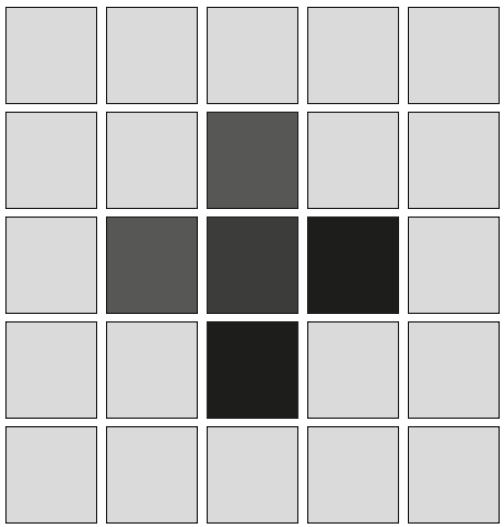
The layered view of neural networks



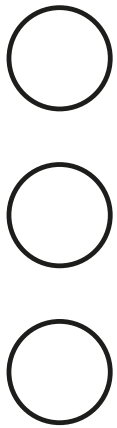
The neural network can be seen as a composition of the different layers: $f_d(\ldots f_2(f_1(\mathbf{x})))$. In modern deep learning frameworks layers (not neurons) are considered the basic building blocks.

The number of layers "d" is called the **depth** of the network. This is where the "deep" on deep learning comes from.

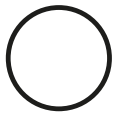
Images as inputs to a neural network



5×5 image

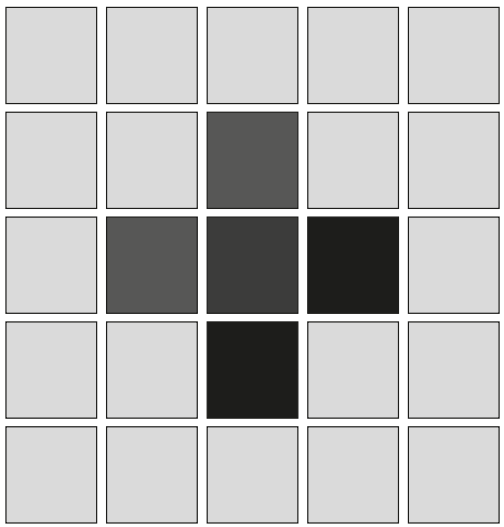


3 hidden neurons

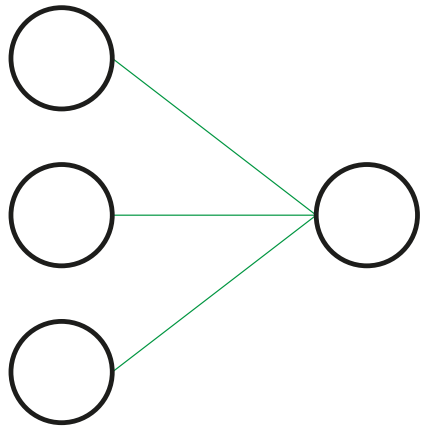


1 output neurons

Images as inputs to a neural network



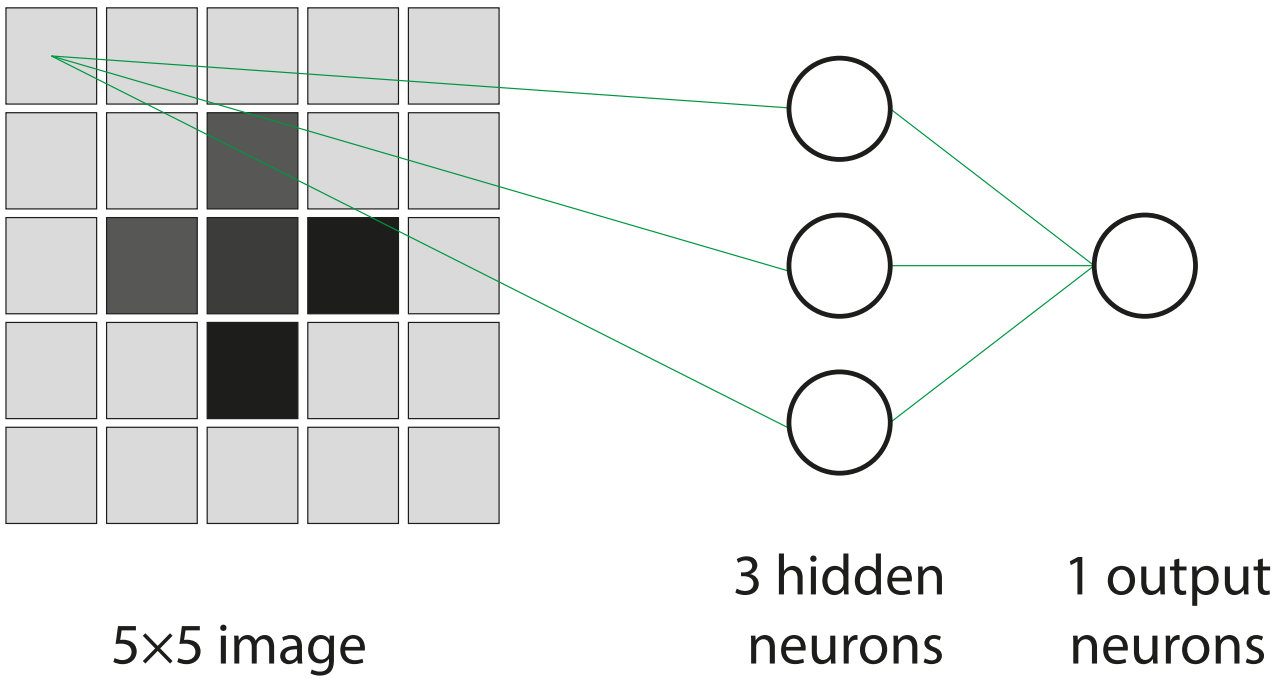
5×5 image



3 hidden neurons

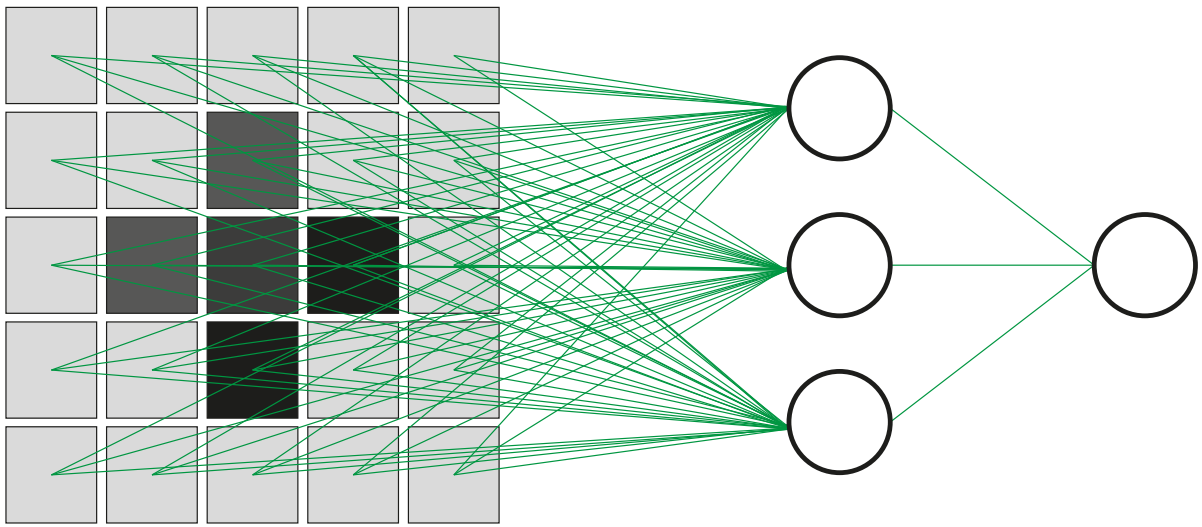
1 output neurons

Images as inputs to a neural network

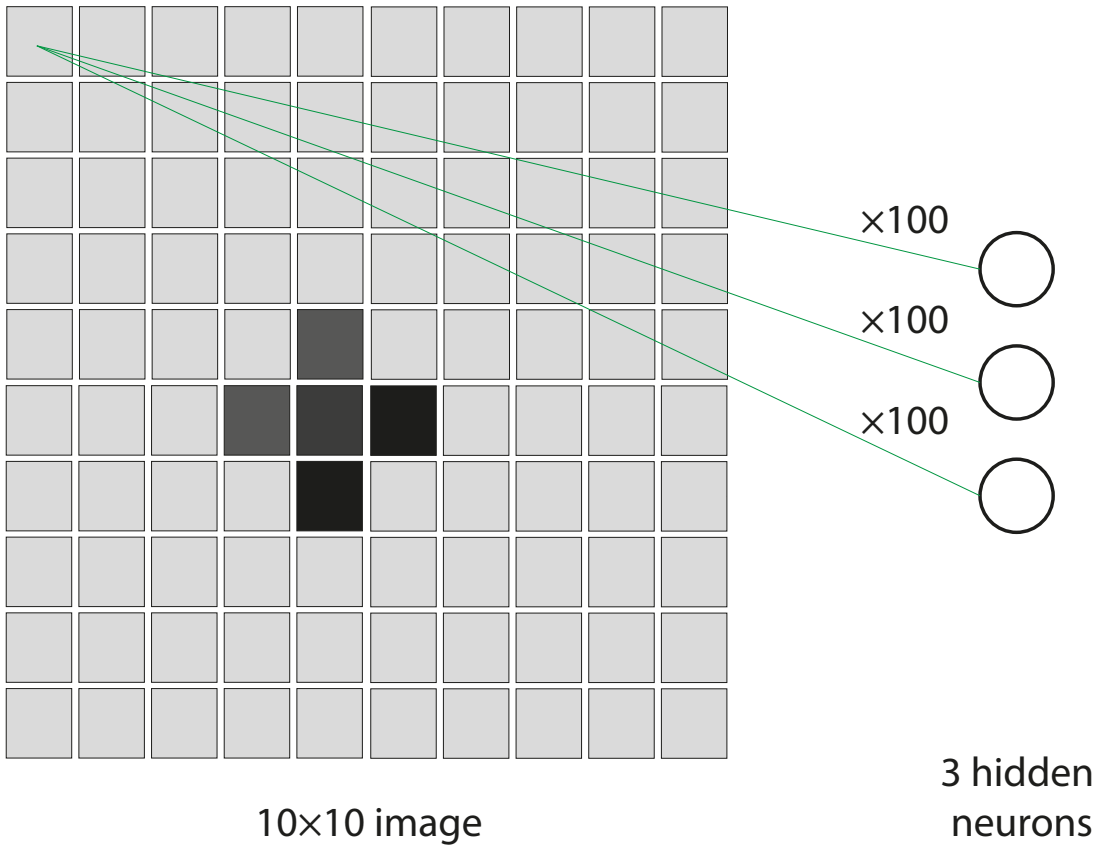


The biases $w_{i,0}$ are not shown.

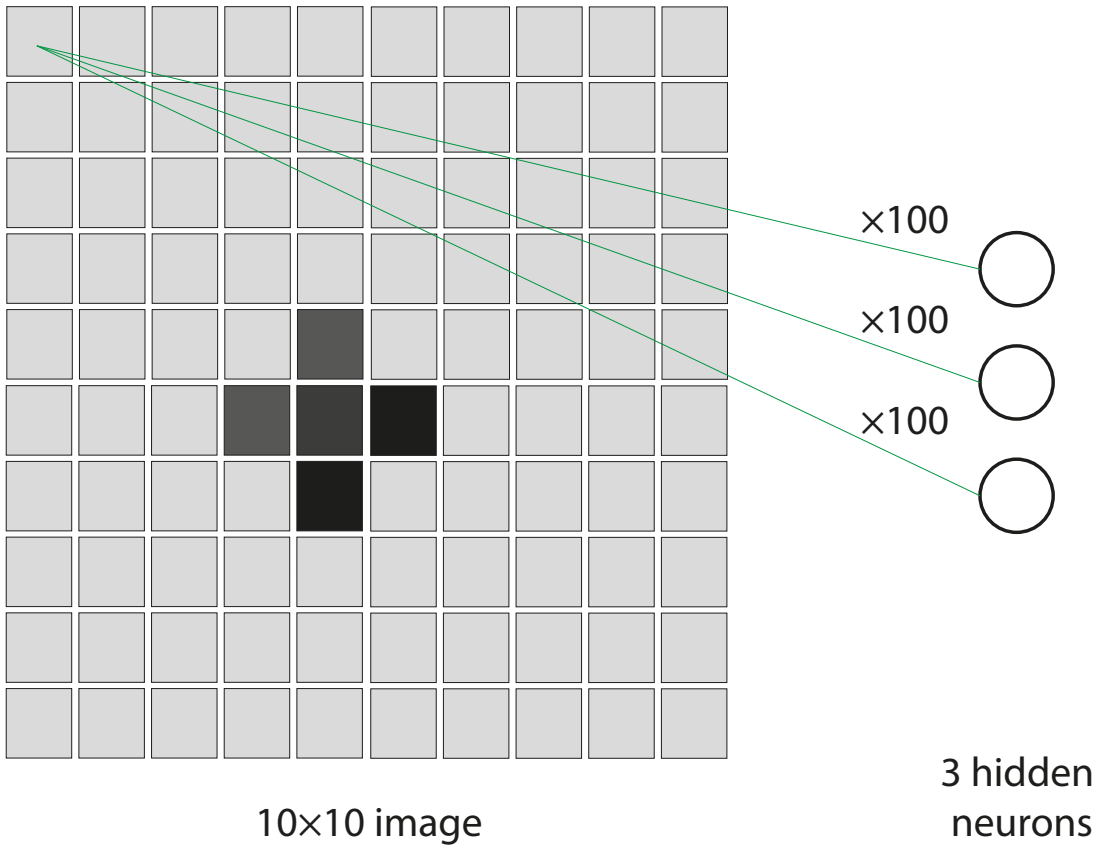
Images as inputs to a neural network



The number of parameters explodes with larger image sizes



The number of parameters explodes with larger image sizes

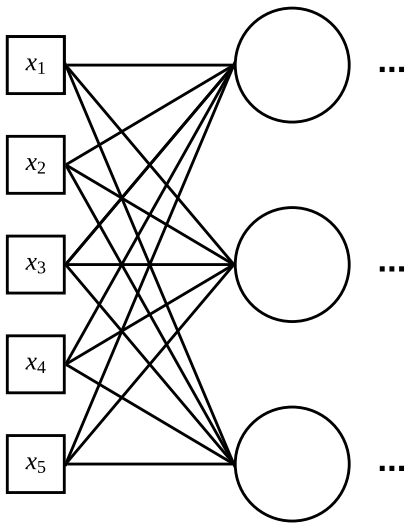


$$\#\ \text{parameters} = (\text{height} \times \text{width} \times \#\ \text{channels} + 1) \times \#\textit{neurons}$$

The "+1" comes from the biases $w_{i,0}$.

Towards convolutional neural networks

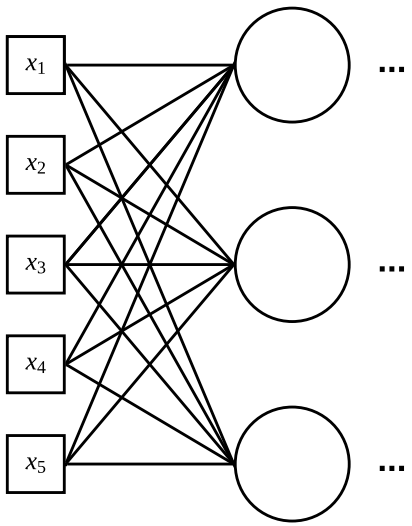
Example (1-D image for simplicity): 5×1 input image, 3 hidden neurons.



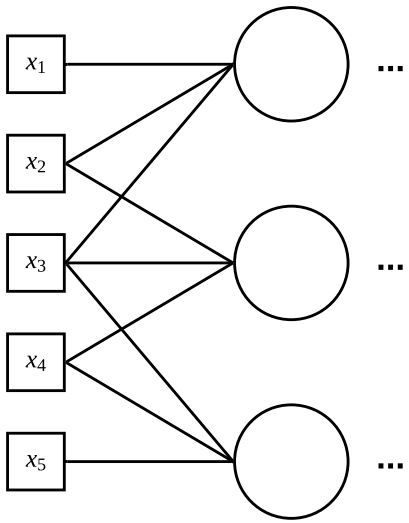
full connectivity: 15 parameters

Towards convolutional neural networks

Example (1-D image for simplicity): 5×1 input image, 3 hidden neurons.



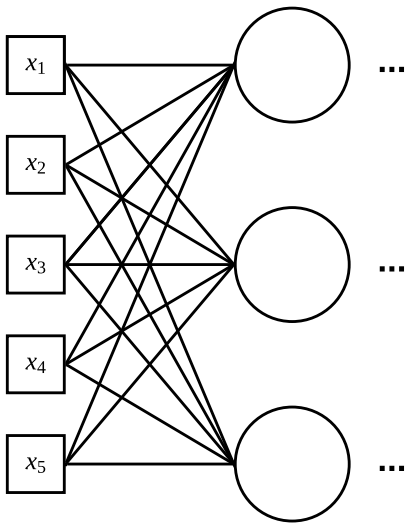
full connectivity: 15 parameters



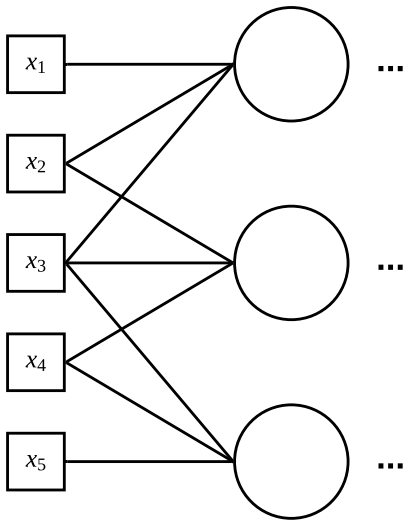
sparse connectivity: 9 parameters

Towards convolutional neural networks

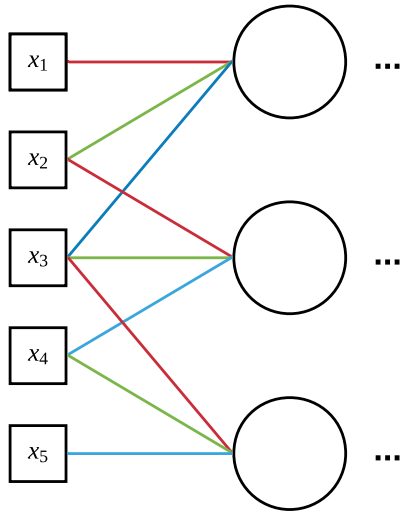
Example (1-D image for simplicity): 5×1 input image, 3 hidden neurons.



full connectivity: 15 parameters



sparse connectivity: 9 parameters

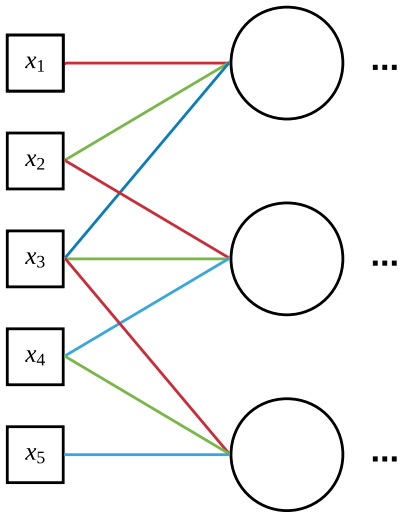


shared weights: 3 parameters

Note: the poor biases are again, ignored, but there are three of them in each case

Towards convolutional neural networks

Let the outputs of the three neurons be $\sigma(a_1), \sigma(a_2), \sigma(a_3)$. Then:



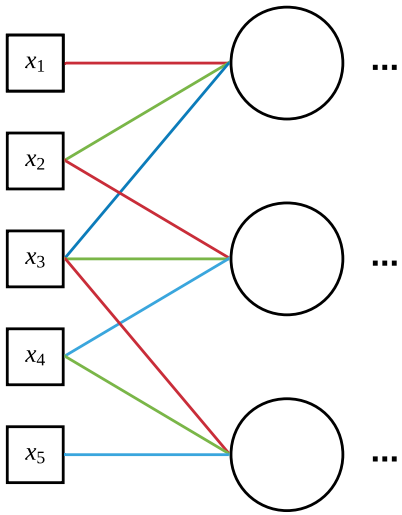
$$a_1 = x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$a_2 = x_2 w_1 + x_3 w_2 + x_4 w_3$$

$$a_3 = x_3 w_1 + x_4 w_2 + x_5 w_3$$

Towards convolutional neural networks

Let the outputs of the three neurons be $\sigma(a_1), \sigma(a_2), \sigma(a_3)$. Then:



$$a_1 = x_1 w_1 + x_2 w_2 + x_3 w_3$$

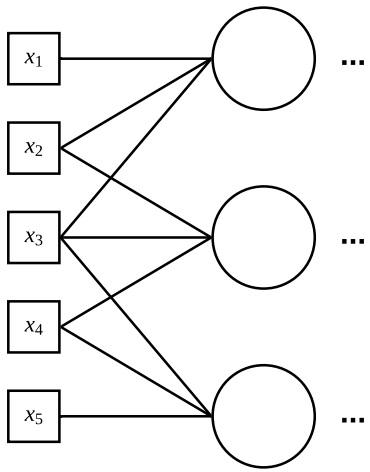
$$a_2 = x_2 w_1 + x_3 w_2 + x_4 w_3$$

$$a_3 = x_3 w_1 + x_4 w_2 + x_5 w_3$$

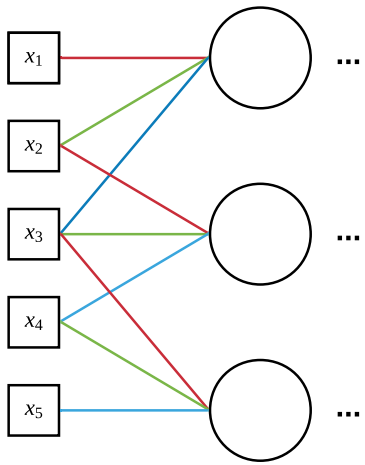
$$[a_1, a_2, a_3] = [x_1, x_2, x_3, x_4, x_5] * [w_3, w_2, w_1]$$

, where $*$ is the convolution operator, thus a **convolutional layer**.

Motivation (or rather a justification)

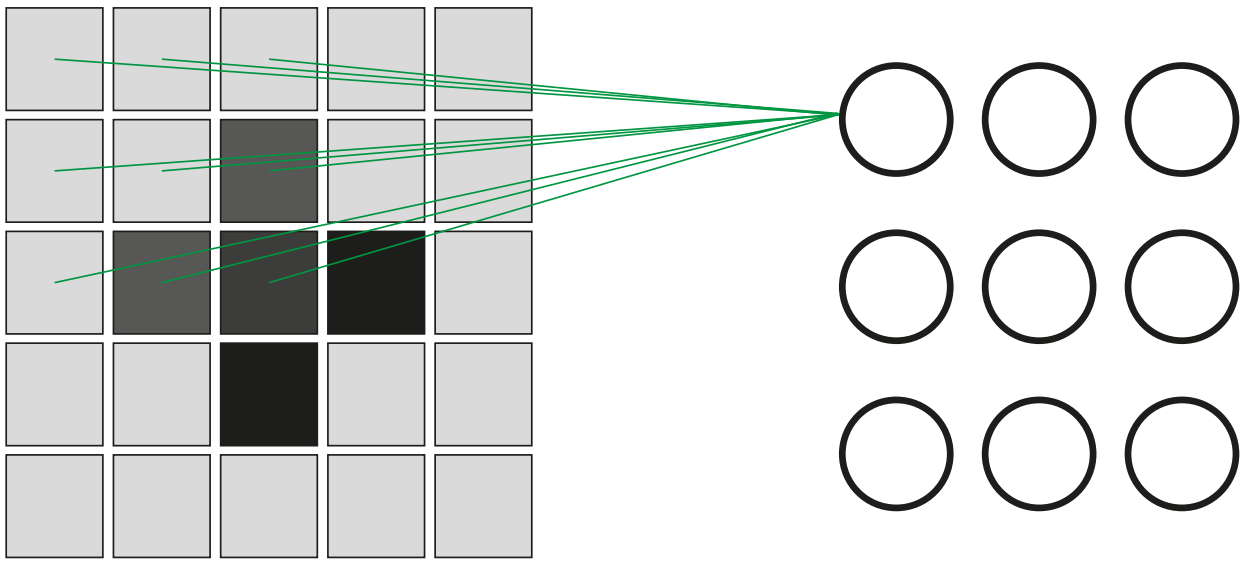


sparse connectivity
motivation: the features appear locally



shared weights
motivation: the features repeat throughout the image

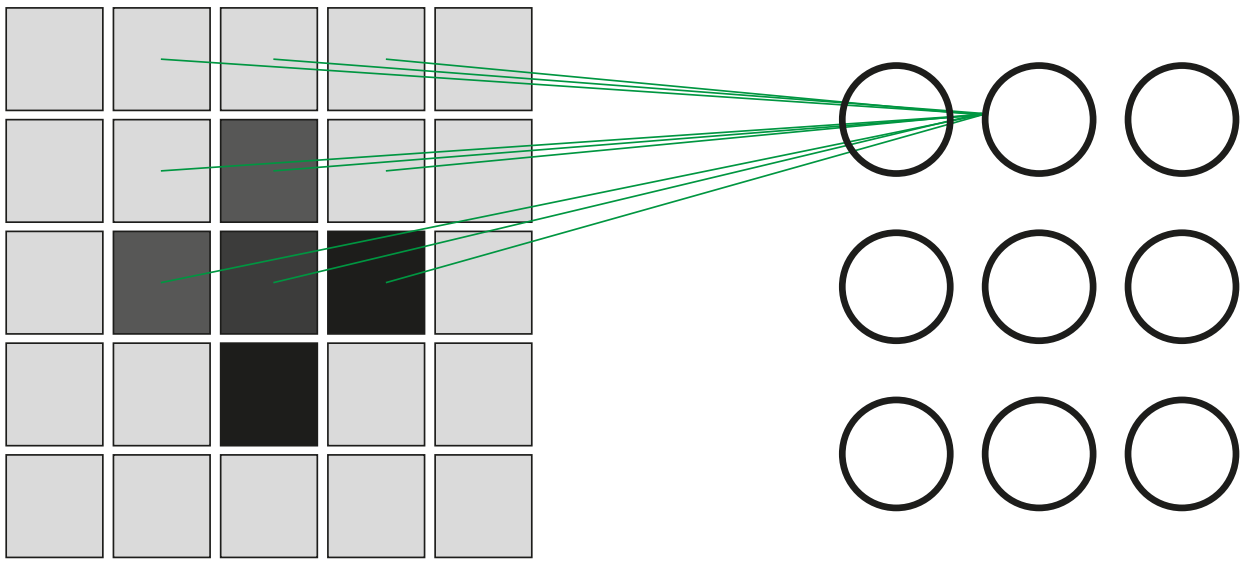
Towards convolutional neural networks in 2D



5×5 image

3×3 hidden layer

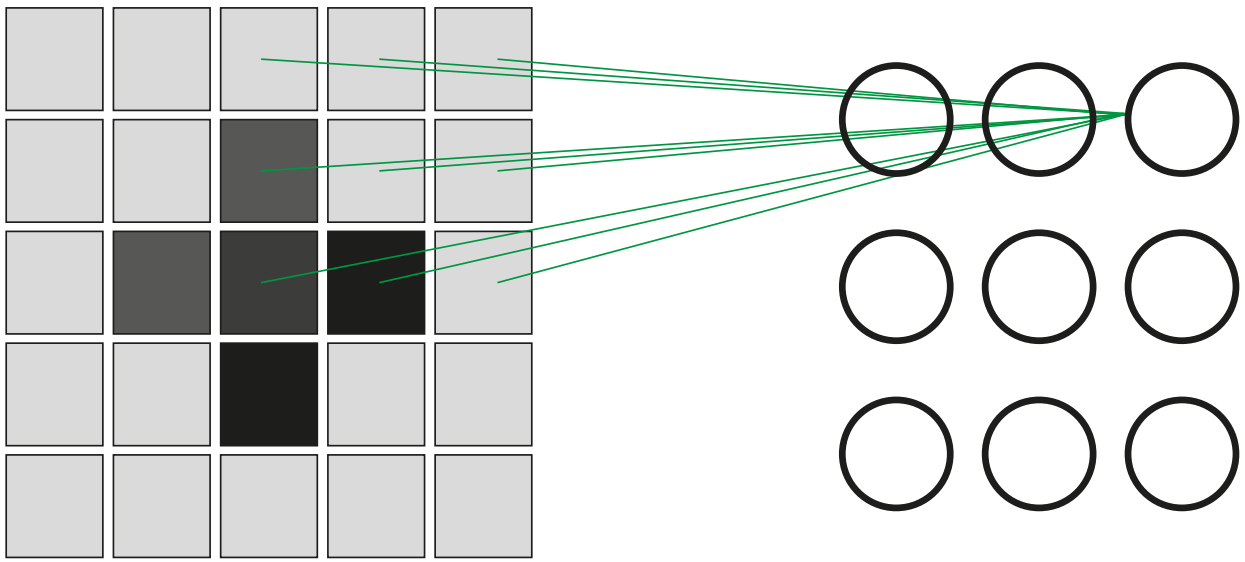
Towards convolutional neural networks in 2D



5×5 image

3×3 hidden layer

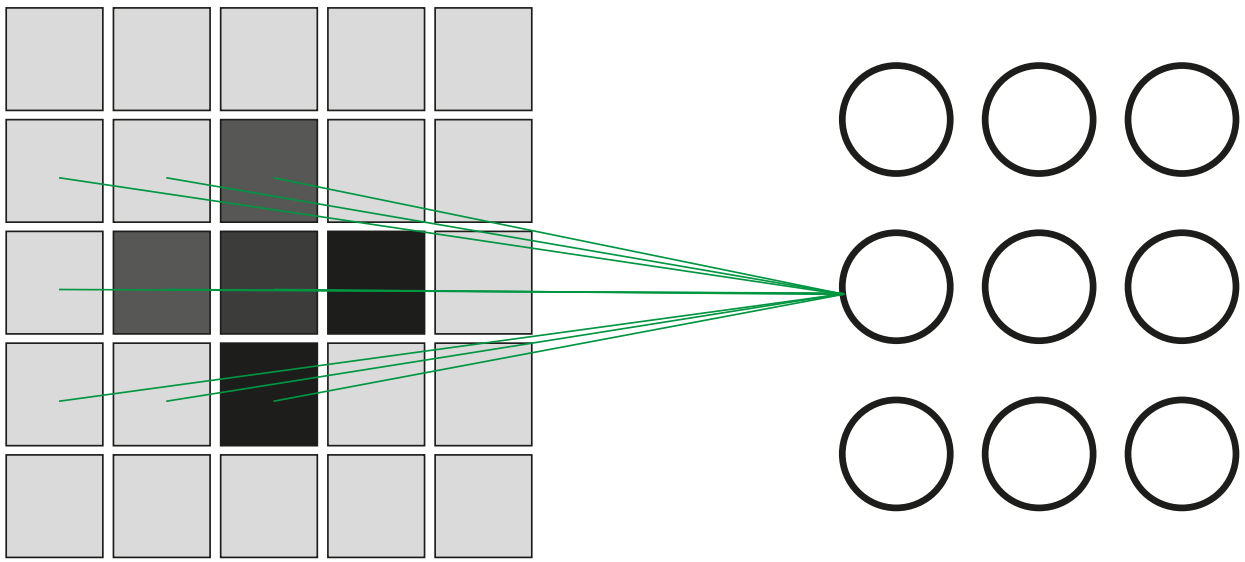
Towards convolutional neural networks in 2D



5×5 image

3×3 hidden layer

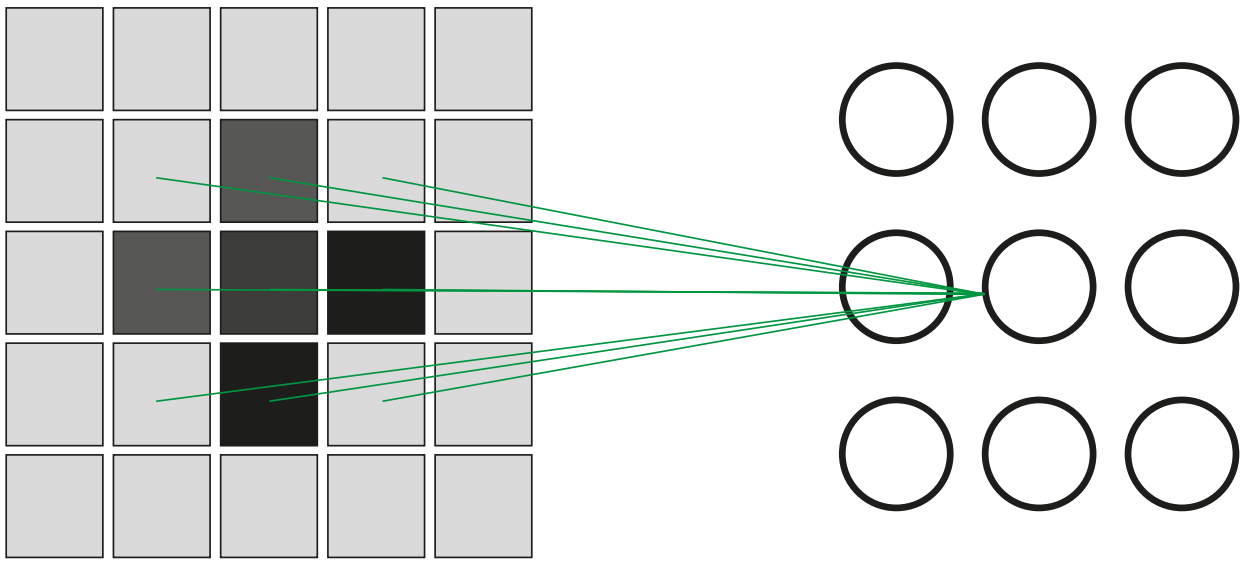
Towards convolutional neural networks in 2D



5×5 image

3×3 hidden layer

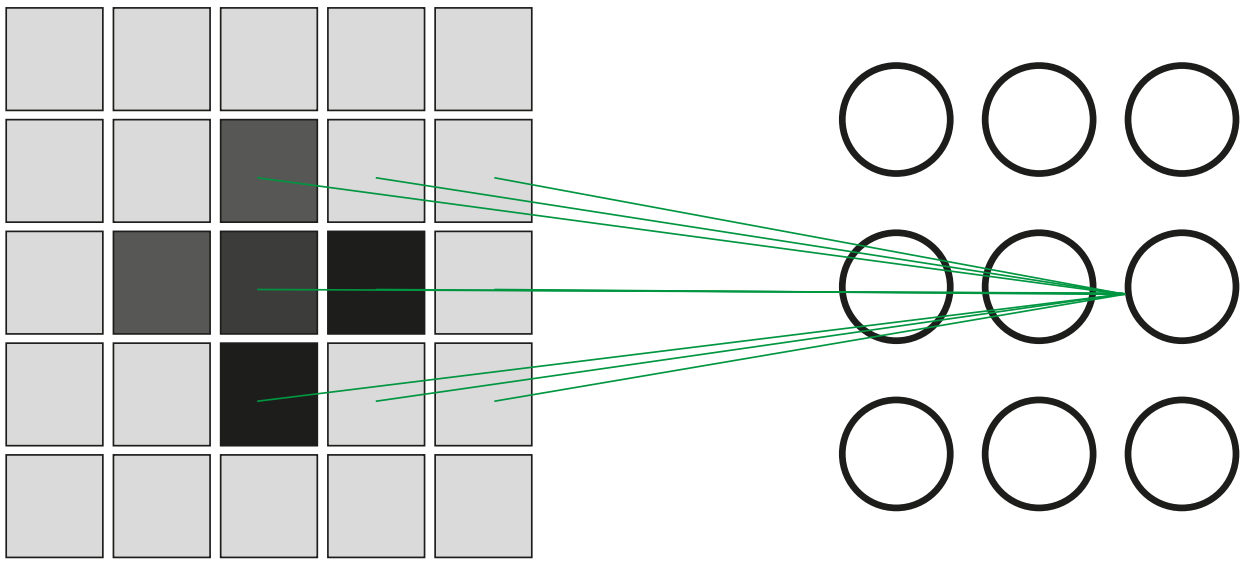
Towards convolutional neural networks in 2D



5×5 image

3×3 hidden layer

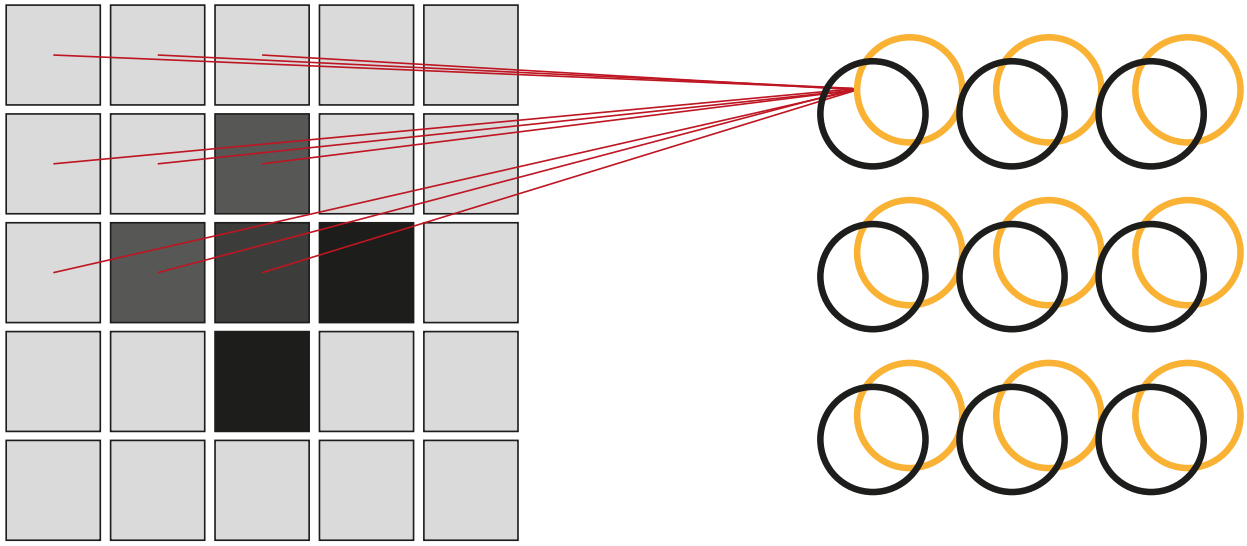
Towards convolutional neural networks in 2D



5×5 image

3×3 hidden layer

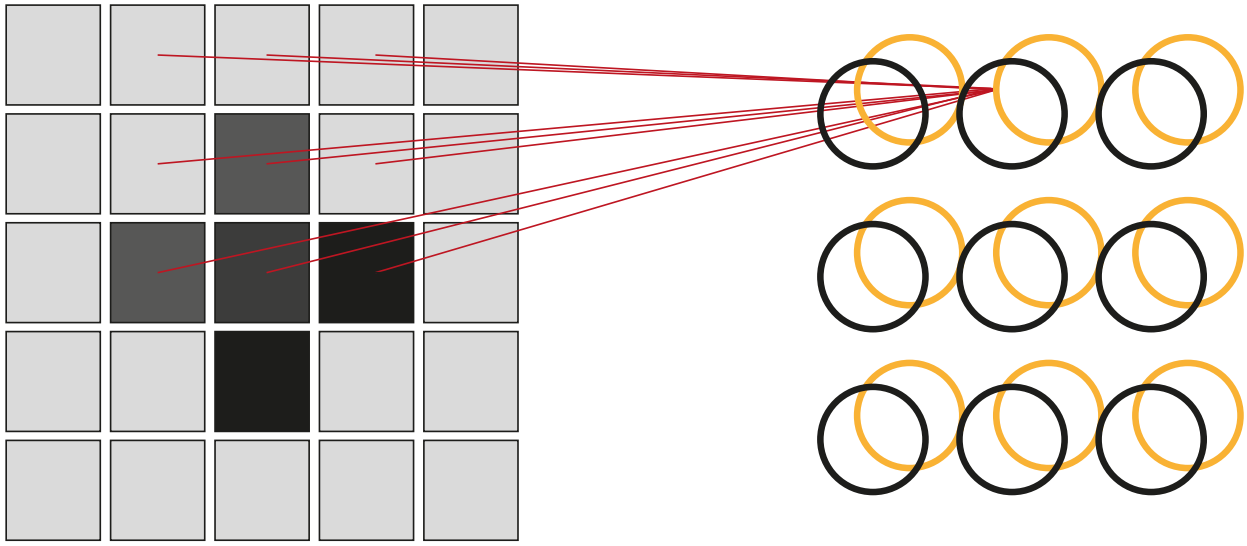
Adding a second feature map



5×5 image

3×3×2 hidden layer
(2 feature maps)

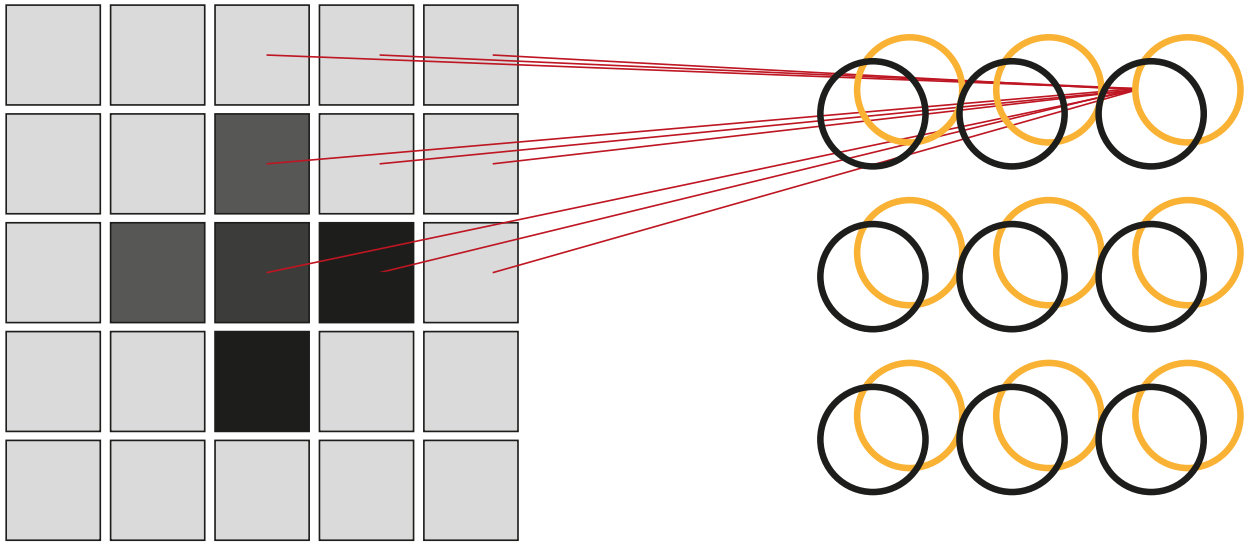
Adding a second feature map



5×5 image

3×3×2 hidden layer
(2 feature maps)

Adding a second feature map



5×5 image

3×3×2 hidden layer
(2 feature maps)

Convolution with padding

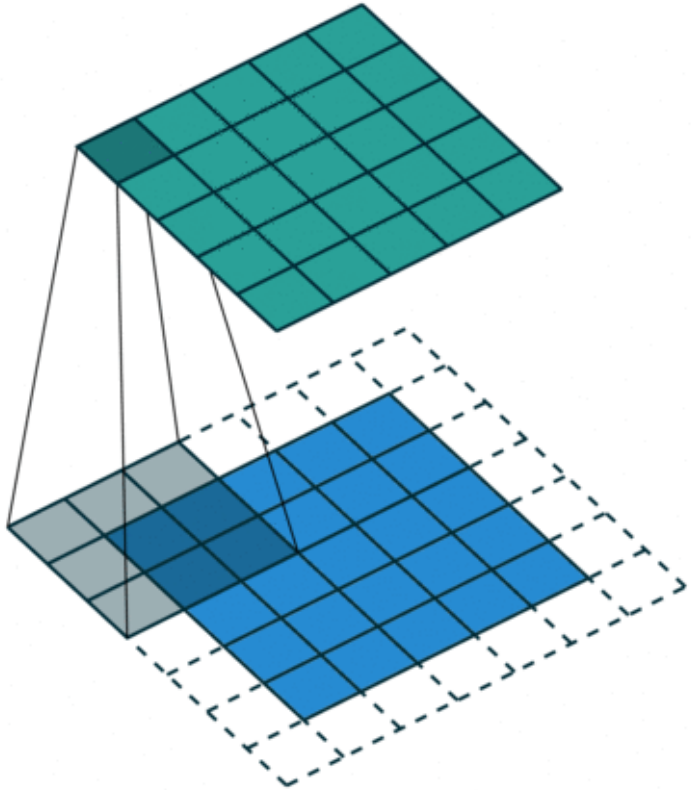
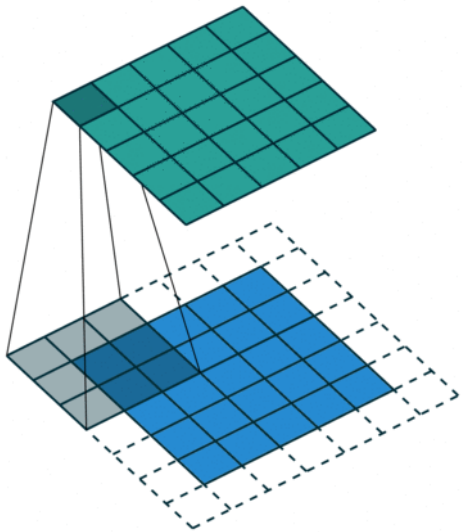


Figure source: https://github.com/vdumoulin/conv_arithmetic

Computing the output size

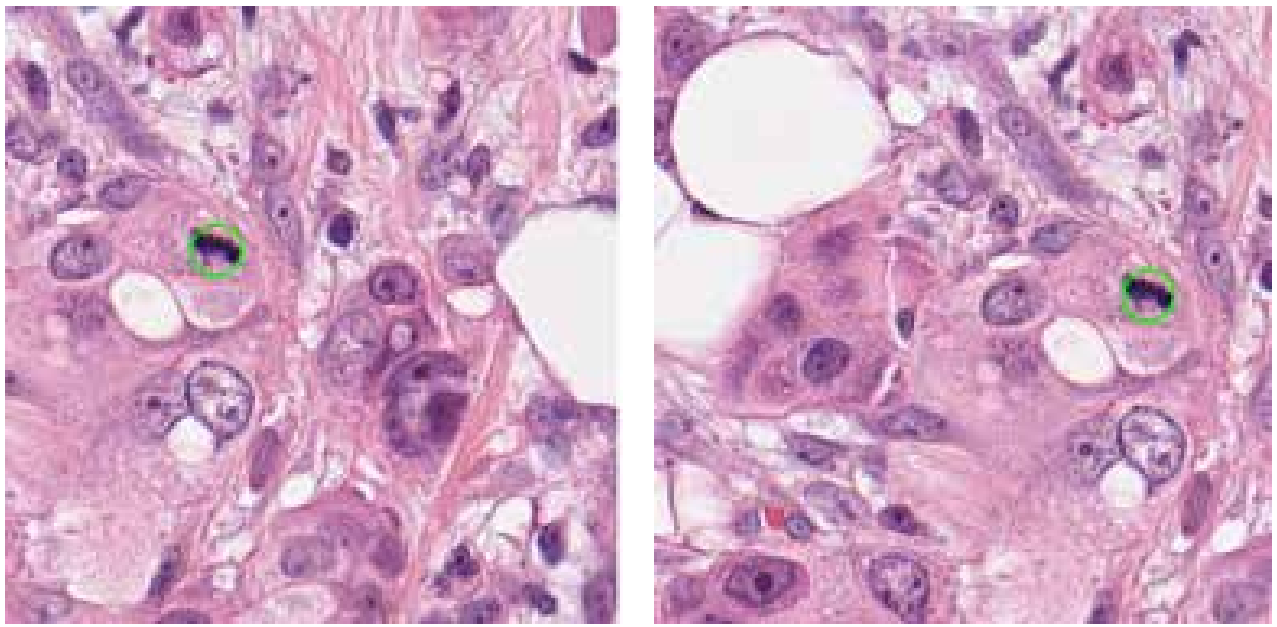


$$\text{output size} = \frac{\text{input size} - \text{kernel size} + 2 \times \text{padding}}{\text{stride}} + 1$$

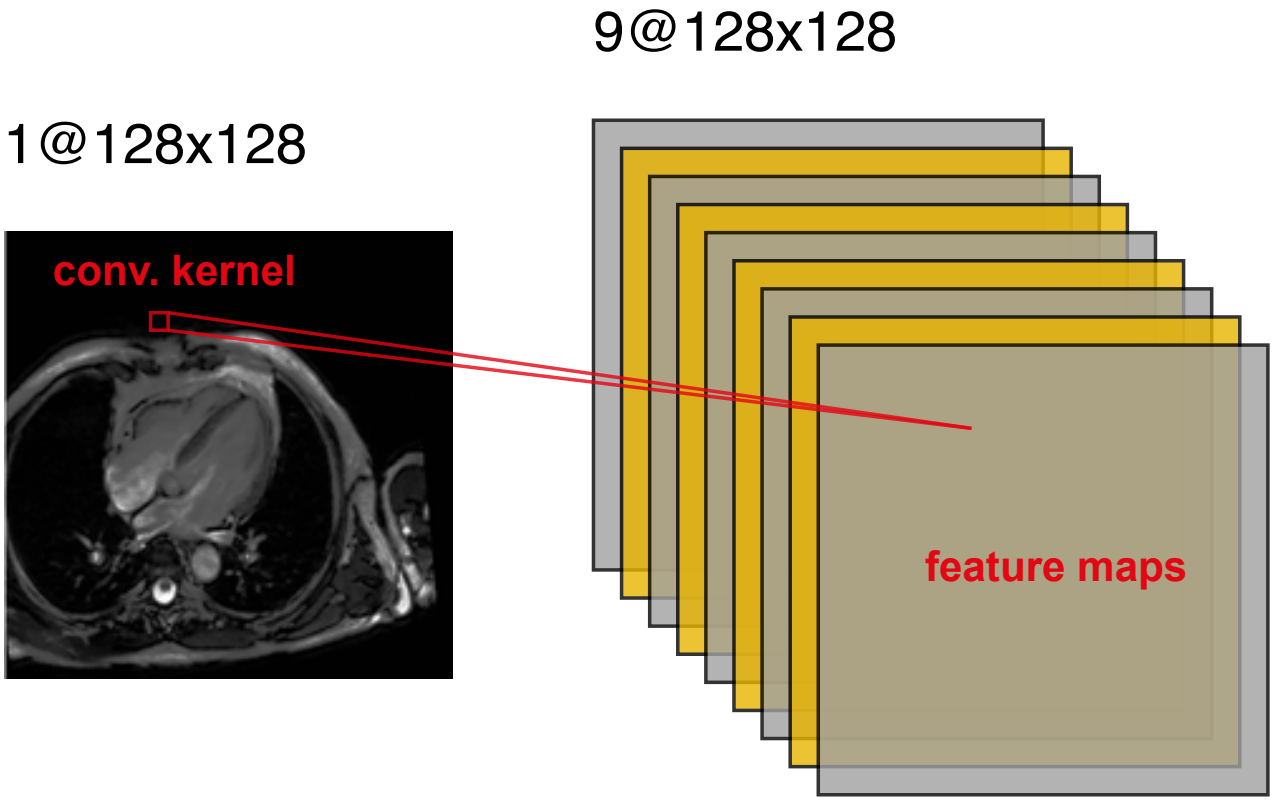
In this example: input size $= 5$, kernel size $= 3$, padding $= 1$, stride $= 1$.
The output size is $(5 - 3 + 2 \times 1)/1 + 1 = 5$.

Motivation (or rather justification) for CNNs

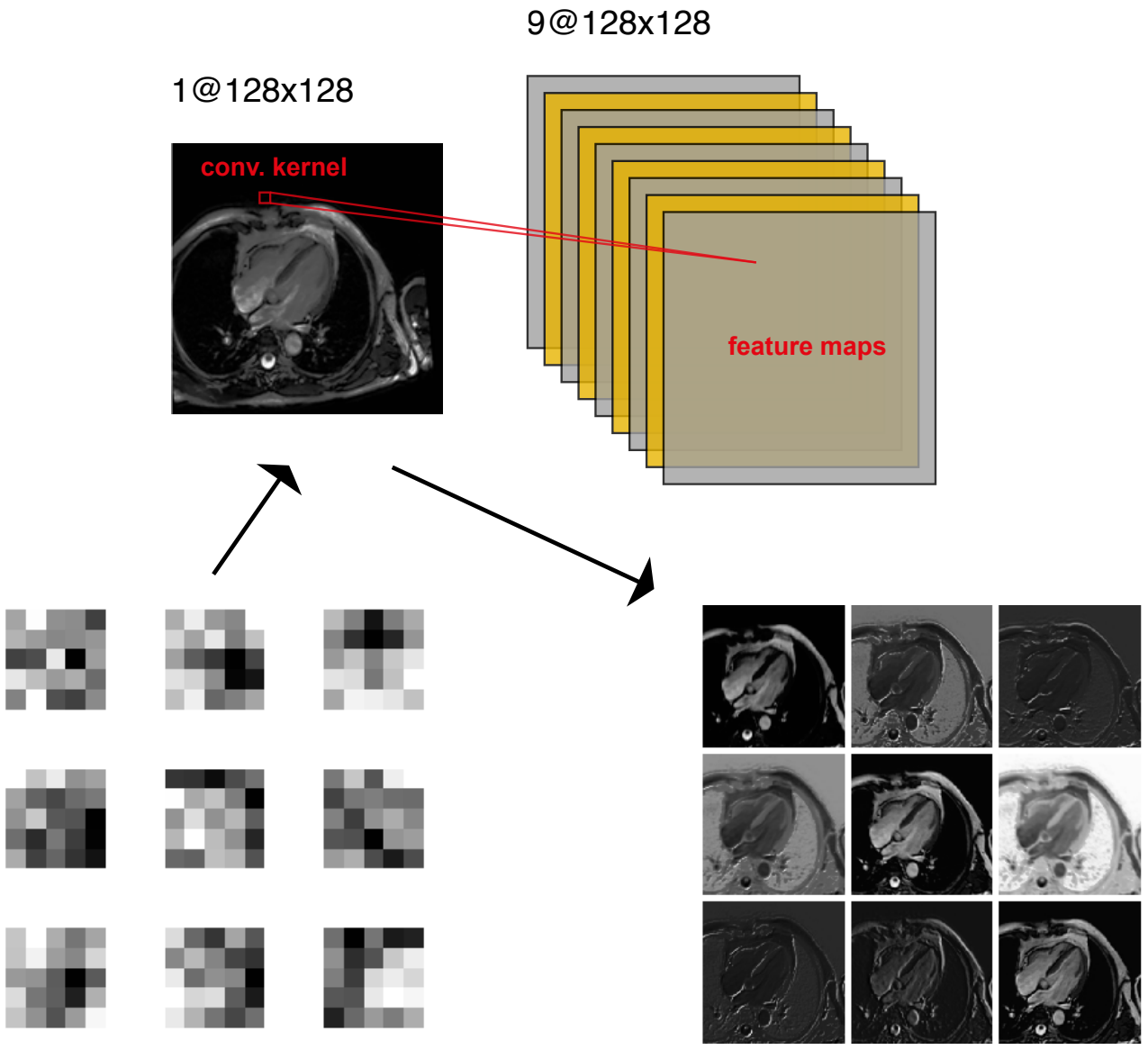
The features of interest can appear at different locations in the image.



Kernels and feature maps



Kernels and feature maps



Motivation (or rather a consequence) for deep CNNs

The network learns low-level features in the first layers, and builds up towards more complex features in the deeper layers: intensity $\rightarrow$ edges and colour blobs $\rightarrow$ junctions $\rightarrow$ shapes $\rightarrow$ etc.

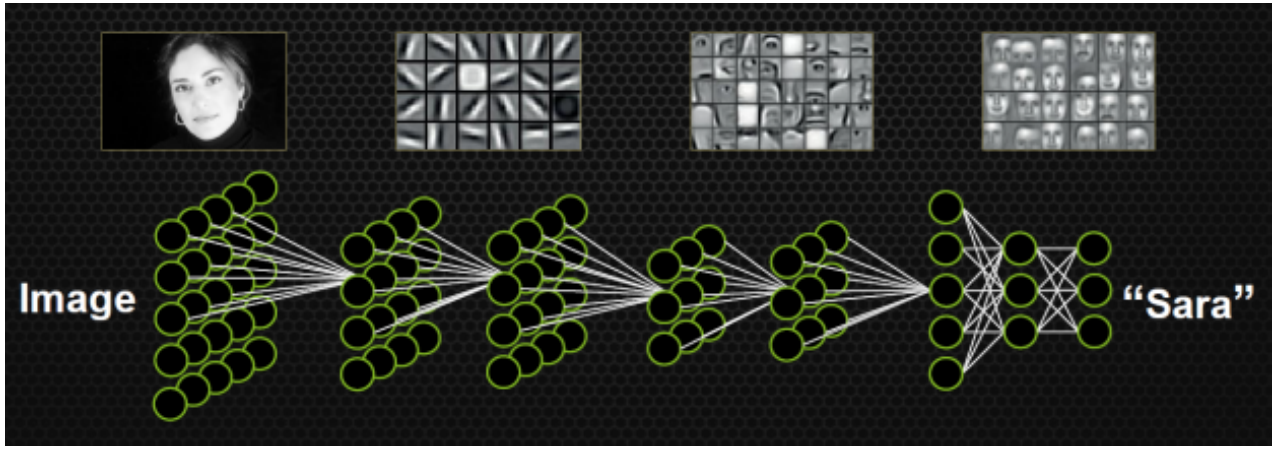


Figure source: nvidia.com

Equivariance and invariance to translation

The convolutional layers are **equivariant with translation**: as the input is translated, the output is translated in a predictable manner.

Equivariance and invariance to translation

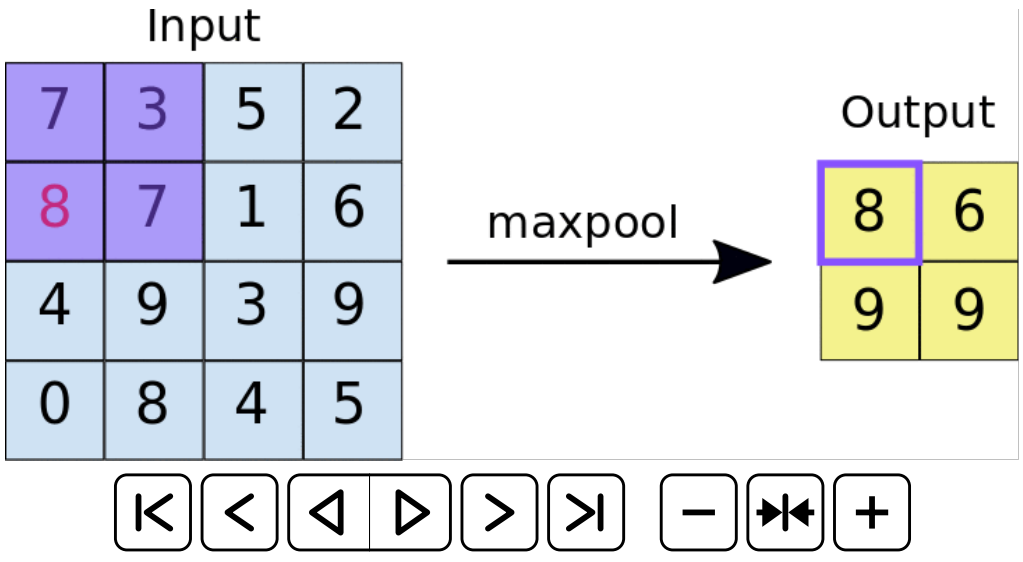
The convolutional layers are **equivariant with translation**: as the input is translated, the output is translated in a predictable manner.

A desired property of neural networks for classification is **invariance**: as the input is translated, the output remains the same.

Partial translational invariance of CNNs is achieved with the max-pooling operator.

Note: there are other types of invariance e.g. rotational.

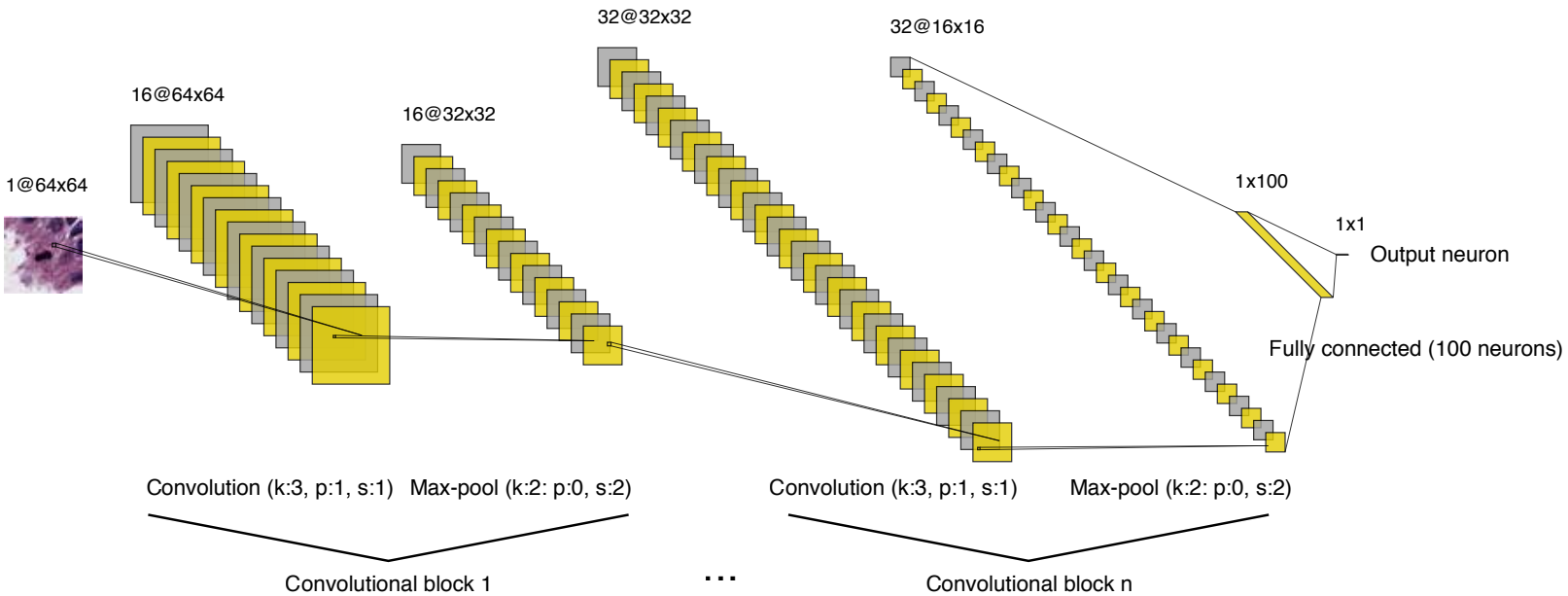
Max-pooling



A max-pool with a 2×2 kernel stride and size 2 (most common form) will reduce the image size by 2 in each dimension (a useful side-effect).

A "typical" CNN architecture for 2D image classification

Note that the convolution is a linear operation so non-linearities (such as ReLU) are still needed.



Training CNNs

Example by Andrej Karpathy.