Class 13: CNNs and Image Classification

Theme: Images

Computational Analysis of Text, Audio, and Images, Fall 2023

Aarhus University

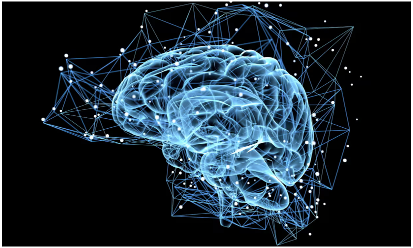
Mathias Rask (mathiasrask@ps.au.dk)

Aarhus University

Table of Contents

History

- The idea of neural networks dates back to the 1940s. Basic idea:
  - ⇝ Can we model how humans learn?
- Human brains consist of neurons, which are in turn connected with each other
- Neurons are connected to each other and propagate signals to one another
- Each neuron collects the signal and then activates it to a larger or lesser extent



- The activated neurons are then propagated further onto new neurons

⇝ This whole idea is encoded in artificial neural networks (ANNs)

Artificial Neural Networks (ANNs)

- NN is a mathematical function f that maps inputs to outputs $\mathbb{R} \to \mathbb{R}$ based on the structure and parameters of the network
- The goal is to have the NN *learn* the "right" parameters based on data

$\rightsquigarrow$ We approximate the function f

How does this relate to deep learning?

- In theory: NNs $\neq$ deep learning
- In practice: NNs $=$ deep learning

NN Components

Every NN consists of:

- Neurons
- Activations
- Parameters (weights and biases)
- Layers

How can we explain each of the components?

Linear Regression

Neural networks look complex when they get deep...

But they are actually just nested regressions...

A simple linear regression looks this this:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2$$

We can translate this into NN terminology:

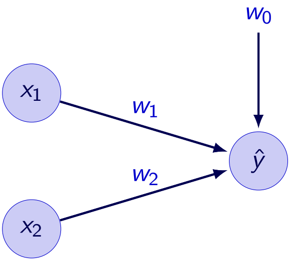
$$y = w_0 + w_1 x_1 + w_2 x_2$$

where:

- w_0 is the bias/intercept (also called b_0)
- w_1 is the weight/parameter/coefficient β_1 connecting $x1$ to y
- w_2 is the weight/parameter/coefficient β_2 connecting $x2$ to y

In math, this is called a linear combination

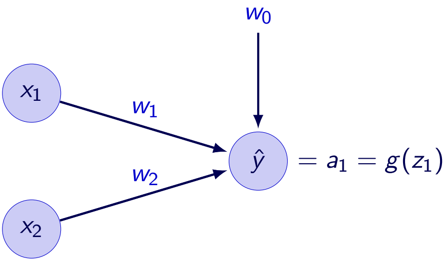
Graph Regression



$$= z_1 = w_0 + w_1 x_1 + w_2 x_2 = \begin{cases} 0 & \text{if } z_1 \leq 0 \\ 1 & \text{if } z_1 > 0 \end{cases}$$

Non-Linearity

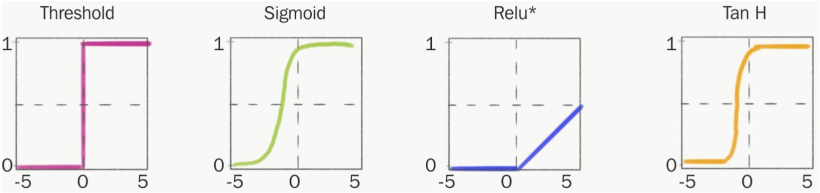
Real-world problems are often non-linear. To enable NNs to complex issues, we use activation functions:



The non-linearity arises from parsing the linear combination through a non-linear function g

$\rightsquigarrow$ without activation functions, NNs are limited to solving linear problems

Activation Functions



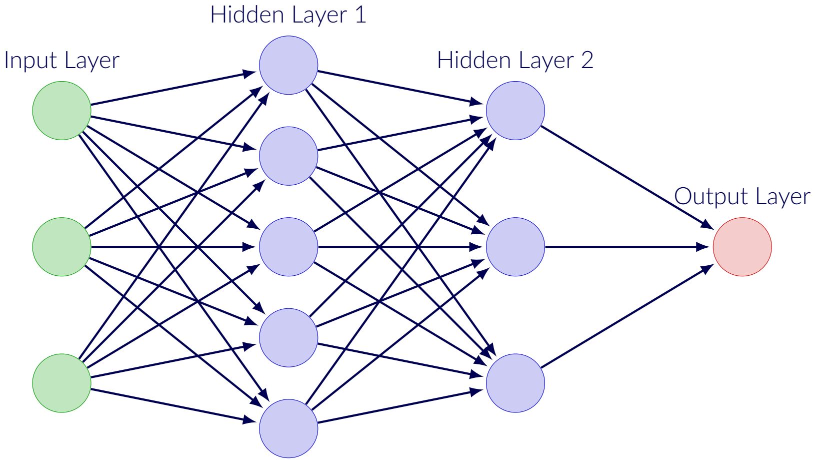
$$\text{Step Function:} \quad f(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$$

$$\text{Sigmoid:} \quad f(x) = \frac{1}{1 + e^{-x}}$$

$$\text{ReLU (Rectified Linear Unit):} \quad f(x) = \max(0, x)$$

$$\text{tanh:} \quad f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

A Vanilla NN



1. How many neurons do the network contains?
2. How many layers?
3. How many weights and biases?

Forward and Backward Passing

Each pass through the network consists of a forward pass and a backward pass.

- Forward pass: Compute output given x
  1. Collecting ($\sum$)
  2. Activating (g)
  3. Distributing ($\rightarrow$)

  $\rightsquigarrow$ Output: $\hat{y} = f(x \mid w)$
- Backward pass: Compute partial derivatives

NN Notation

Before we manually compute a forward pass, let's introduce some notation.

We use $w_{jk}^{(L)}$ to denote the weight of the connection from the k^{th} neuron in the $(L-1)^{th}$ layer to the j^{th} neuron in the L^{th} layer

Similarly, we denote the biases and activations as:

> $b_j^{(L)}$ is the bias for the j^{th} neuron in the L^{th} layer
>
> $a_j^{(L)}$ is the activation for the j^{th} neuron in the L^{th} layer

In this notation, we can write the relations between a hidden neuron j in layer L and the previous layer $L-1$ as:

$$z_j^{(L)} = \sum_k w_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)} \tag{1}$$

$$a_j^{(L)} = g\left(z_j^{(L)}\right) \tag{2}$$

where g is an activation function.

Nested Regressions

While basic sums are intuitive, computers compute the forward pass using matrix notation:

$$a^{(L)} = g(w^{(L)} a^{(L-1)} + b^{(L)}). \tag{3}$$

In this notation, $w^{(L)}$ contains the weights connecting to the L^{th} layer such that the j^{th} row and k^{th} column corresponds to $w_{jk}^{(L)}$. Note that $a^{(L-1)}$ and $b^{(L)}$ are just vectors.

$$w^{(L)} = \begin{bmatrix} w_{11}^{(L)} & w_{12}^{(L)} & w_{13}^{(L)} & w_{14}^{(L)} & w_{15}^{(L)} \\ w_{21}^{(L)} & w_{22}^{(L)} & w_{23}^{(L)} & w_{24}^{(L)} & w_{25}^{(L)} \\ w_{31}^{(L)} & w_{32}^{(L)} & w_{33}^{(L)} & w_{34}^{(L)} & w_{35}^{(L)} \end{bmatrix}' \quad b^{(L)} = \begin{bmatrix} b_1^{(L)} \\ b_2^{(L)} \\ b_3^{(L)} \end{bmatrix} \quad a^{(L-1)} = \begin{bmatrix} a_1^{(L-1)} \\ a_2^{(L-1)} \\ a_3^{(L-1)} \\ a_4^{(L-1)} \\ a_5^{(L-1)} \end{bmatrix}$$

What's the output dimension when we compute the product $w^{(L)} a^{(L-1)}$?

Matrix Multiplication

$$w^{(L)} = \begin{bmatrix} w_{11}^{(L)} & w_{12}^{(L)} & w_{13}^{(L)} & w_{14}^{(L)} & w_{15}^{(L)} \\ w_{21}^{(L)} & w_{22}^{(L)} & w_{23}^{(L)} & w_{24}^{(L)} & w_{25}^{(L)} \\ w_{31}^{(L)} & w_{32}^{(L)} & w_{33}^{(L)} & w_{34}^{(L)} & w_{35}^{(L)} \end{bmatrix}' \qquad a^{(L-1)} = \begin{bmatrix} a_1^{(L-1)} \\ a_2^{(L-1)} \\ a_3^{(L-1)} \\ a_4^{(L-1)} \\ a_5^{(L-1)} \end{bmatrix}$$

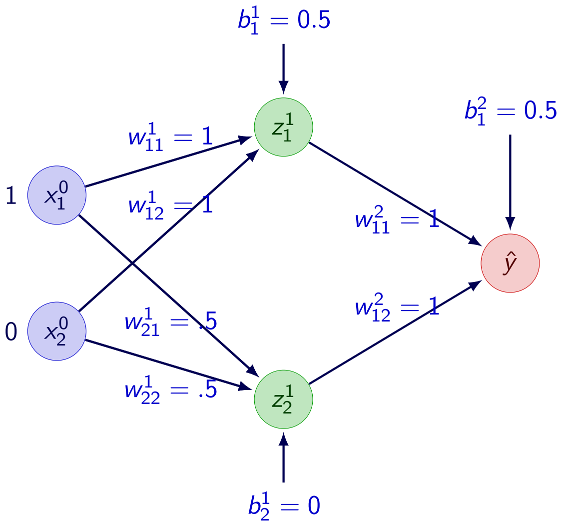
$$\begin{aligned} z_1^{(L)} &= w_{11}^{(L)} a_1^{(L-1)} + w_{12}^{(L)} a_2^{(L-1)} + w_{13}^{(L)} a_3^{(L-1)} + w_{14}^{(L)} a_4^{(L-1)} + w_{15}^{(L)} a_5^{(L-1)} + b_1^{(L)} \\ &= \sum_k w_{1k}^{(L)} a_k^{(L-1)} + b_1^{(L)} \\ z_2^{(L)} &= w_{21}^{(L)} a_1^{(L-1)} + w_{22}^{(L)} a_2^{(L-1)} + w_{23}^{(L)} a_3^{(L-1)} + w_{24}^{(L)} a_4^{(L-1)} + w_{25}^{(L)} a_5^{(L-1)} + b_2^{(L)} \\ &= \sum_k w_{2k}^{(L)} a_k^{(L-1)} + b_2^{(L)} \\ z_3^{(L)} &= w_{31}^{(L)} a_1^{(L-1)} + w_{32}^{(L)} a_2^{(L-1)} + w_{33}^{(L)} a_3^{(L-1)} + w_{34}^{(L)} a_4^{(L-1)} + w_{35}^{(L)} a_5^{(L-1)} + b_3^{(L)} \\ &= \sum_k w_{3k}^{(L)} a_k^{(L-1)} + b_3^{(L)} \end{aligned}$$

Forward Passing By Hand

- Data point: $(x_1, x_2) = (1, 0)$
- Two layers: One hidden and one output layer
- Hidden layer has two neurons
- Output layer has one neuron
- Target is $y = 1$

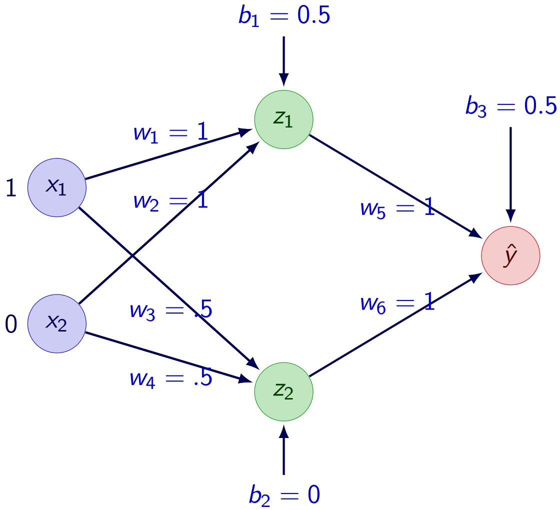
We "randomly" initialize the weights

Our Network



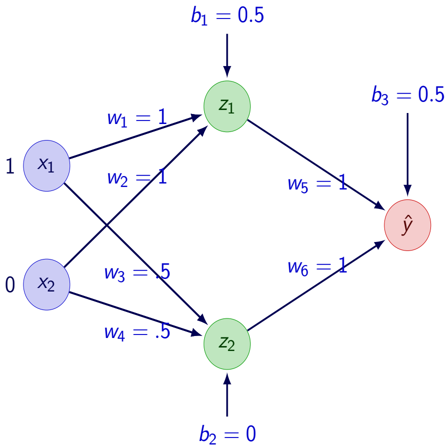
$b_1^1 = 0.5$
$b_1^2 = 0.5$
$w_{11}^1 = 1$
$w_{12}^1 = 1$
$w_{21}^1 = .5$
$w_{22}^1 = .5$
$w_{11}^2 = 1$
$w_{12}^2 = 1$
1
0
x_1^0
x_2^0
z_1^1
z_2^1
$\hat{y}$
$b_2^1 = 0$

Our Network



Complete the forward pass: What's $\hat{y}$?

Forward Calculations



$$\begin{aligned} z_1 &= w_1 x_1 + w_2 x_2 + b_1 \\ &= 1 \cdot 1 + 1 \cdot 0 + 0.5 \\ &= 1 + 0 + 0.5 \\ &= 1.5 \end{aligned}$$

$$\begin{aligned} z_2 &= w_3 x_1 + w_4 x_2 + b_2 \\ &= 0.5 \cdot 1 + 0.5 \cdot 0 + 0 \\ &= 0.5 + 0 + 0 \\ &= 0.5 \end{aligned}$$

$$\begin{aligned} \hat{y} &= w_5 z_1 + w_6 z_2 + b_3 \\ &= 1 \cdot 1.5 + 1 \cdot 0.5 + 0.5 \\ &= 1.5 + 0.5 + 0.5 \\ &= 2.5 \end{aligned}$$

Cost Functions

Great. We now completed our forward pass but we need an additional step to evaluate how good or bad our output is: cost function.

A popular choice for regression problems is the sum of squares (quadratic cost):

$$C = \frac{1}{2n} \sum_x \|y(x) - \hat{y}(x)\|^2$$

For a single example x, it becomes: $C_x = \frac{1}{2}\|y - \hat{y}\|^2$.

Assume the target is $y = 2$. Compute the cost for our output $\hat{y} = 2.5$ (you can ignore the $\frac{1}{2}$)

Gradient Descent

The objective of our NN is to correctly learn the function $y = f(x \mid w)$. Why? Because it then means that we can map x to y

How do we learn f?
$\rightsquigarrow$ by minimizing the cost function

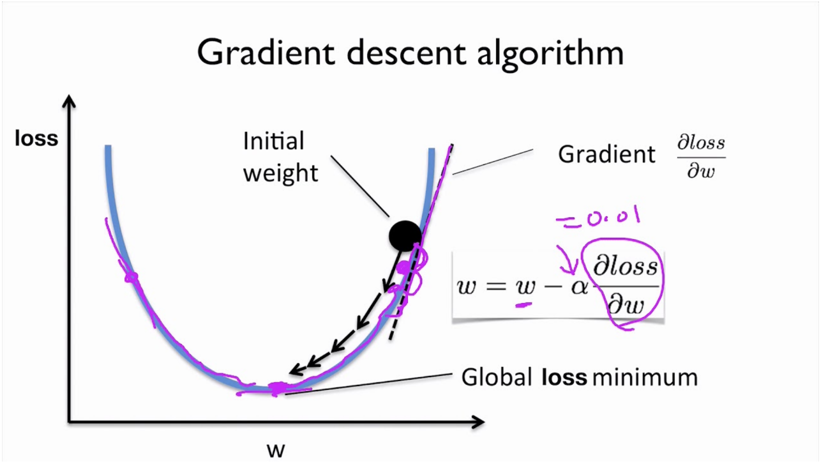
NN minimizes the cost by using a method called gradient descent.

- Randomly initialize the weights
- Repeat:
    1. Compute gradient based on the **all data points** to figure out in what way the weights should be adjusted (up or down) to minimize the loss
    2. Update weights with step η based on the gradient
    3. Terminate when weights give proper solution

In practice:

- Stochastic Gradient Descent: **one data point**
- Mini-batch Gradient Descent: **one small batch**

Gradient Descent Illustration



source: https://github.com/dshahid380/Gradient-descent-Algorithm

Backpropagation

Gradient Descent involves (duh...) a gradient

In a nutshell, the gradient encodes information about how we should adjust our weights to improve our output

To compute the gradient, we use the famous backpropagation algorithm

- Basic idea: Understand how changing the weights and biases changes the C
- Computes this by letting the cost from the forward pass C flow backward in the network to compute gradient

Derivatives

- Calculus is the mathematical study of continuous change
- Derivatives quantify the sensitivity of change of a function's output with respect to the input
  - $\rightarrow$ we find this by differentiation which we know, but (can't remember) from high school
- It's typically written as $f(x) = f'(x)$ or $\frac{d}{dx}$

Differentation Exercise

Example 1: $f(x) = x^2$

Example 3: $x^2 + x^3$

Example 2: $f(x) = 1/x$

Hints:

- the power rule: $x^n = nx^{n-1}$
- the sum rule: $f + g = f' + g'$

Differentation Exercise

Solution 1: $f(x) = x^2$

$$\begin{aligned}\frac{d}{dx}x^2 &= 2x^{(2-1)}\\ &= 2x^1\\ &= 2x\end{aligned}$$

Solution 2: $f(x) = 1/x$

$$\begin{aligned}\frac{d}{dx}1/x &= \frac{d}{dx}x^{-1}\\ &= -1x^{-1-1}\\ &= -x^2\\ &= \frac{-1}{x^2}\end{aligned}$$

Solution 3: $x^2 + x^3$

$$\begin{aligned}\frac{d}{dx}x^2 + x^3 &= \frac{d}{dx}x^2 + \frac{d}{dx}x^3\\ &= 2x^{2-1} + 3x^{3-1}\\ &= 2x + 3x^2\end{aligned}$$

The Chain Rule

The chain rule of calculus is used to find the derivative of a function that itself is a function
$\rightsquigarrow$ The backpropagation algorithm is designed to use the chain rule efficiently.

The rule states that:

$$f(g(x)) = f'(g(x))g'(x)$$

Example: $(5x-2)^3$

$f(g) = g^3$
$g(x) = 5x - 2$

Derivatives:

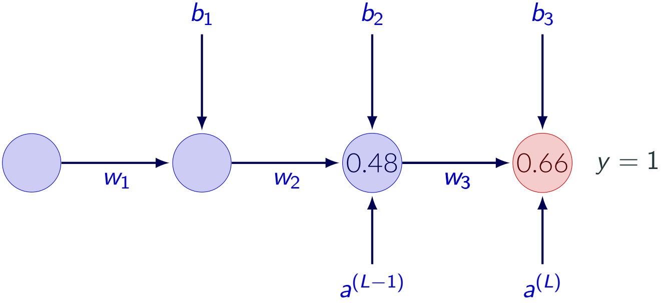
$f'(g) = 3g^2$
$g'(x) = 5$

We can then plugin and calculate the expression:

$$\begin{aligned}\frac{d}{dx}(5x-2)^3 &= (3g(x)^2)(5) \\ &= 15(5x-2)^2\end{aligned}$$

Backpropagation In Action

We imagine we have a simple NN with three layers each with a single neuron:



$$
\begin{aligned}
C &= (a^{(L)} - y)^2 \\
&= (0.66 - 1)^2
\end{aligned}
$$

$$
\begin{aligned}
z^{(L)} &= w^L \cdot a^{(L-1)} + b^L \\
a^{(L)} &= g(z^{(L)})
\end{aligned}
$$

Backpropagation In Action

How the cost function changes w.r.t. a single weight is captured by the partial derivative: $\frac{\partial C}{\partial w^L}$

We compute this using the chain rule!

We need to decompose the chain of action: w^L influences C through z^L and in turn a^L $\rightsquigarrow$ the chain rule:

$$\frac{\partial C}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C}{\partial a^L}$$

Backpropagation In Action

$$\frac{\partial C}{\partial a^L} = \frac{\partial (a^{(L)} - y)^2}{\partial a^L}$$
$$= 2(a^{(L)} - y)$$

$$\frac{\partial a^L}{\partial z^L} = g'\left(z^{(L)}\right)$$

$$\frac{\partial z^L}{\partial w^L} = a^{L-1}$$

$$\frac{\partial C}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C}{\partial a^L}$$
$$= a^{L-1} g'\left(z^{(L)}\right) 2(a^{(L)} - y)$$

From all this, we can go ahead and compute the gradient, which is how the NN learns (recall: gradient descent)

The Gradient

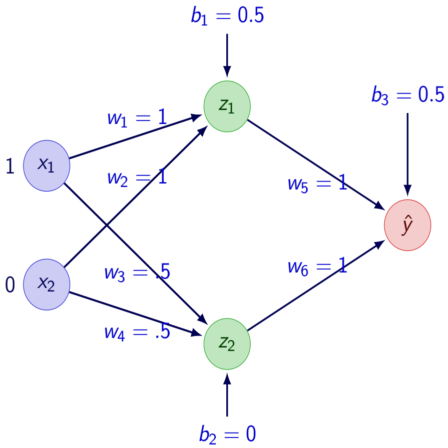
The gradient is a fancy way of saying: *how much should we change our weights to minimize the cost?*

$$-\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^1} \\ \frac{\partial C}{\partial b^1} \\ \frac{\partial C}{\partial w^2} \\ \frac{\partial C}{\partial b^2} \\ \vdots \\ \frac{\partial C}{\partial w^L} \\ \frac{\partial C}{\partial b^L} \end{bmatrix} = \begin{bmatrix} +0.31 \\ +0.03 \\ -1.25 \\ -0.10 \\ \vdots \\ +0.85 \\ +0.10 \end{bmatrix}$$

Each component tells us how much each weight should be:

1. nudged up $\uparrow$ or down $\downarrow$
2. and how much it should change

Backward Calculation



$$\begin{aligned} z_1 &= w_1 x_1 + w_2 x_2 + b_1 \\ &= 1 \cdot 1 + 1 \cdot 0 + 0.5 \\ &= 1 + 0 + 0.5 \\ &= 1.5 \end{aligned}$$

$$\begin{aligned} z_2 &= w_3 x_1 + w_4 x_2 + b_2 \\ &= 0.5 \cdot 1 + 0.5 \cdot 0 + 0 \\ &= 0.5 + 0 + 0 \\ &= 0.5 \end{aligned}$$

$$\begin{aligned} \hat{y} &= w_5 z_1 + w_6 z_2 + b_3 \\ &= 1 \cdot 1.5 + 1 \cdot 0.5 + 0.5 \\ &= 1.5 + 0.5 + 0.5 \\ &= 2.5 \end{aligned}$$

Table of Contents

Dense vs. Convolutional Layers

Dense layers:

- Fully connected: All neurons in layer L is connected to all neurons in layer $L-1$ and $L+2$
- High number of parameters
- No attention to spatial adjacency

Convolutional layers:

- Sparsely connected: We can compress the data with filters/kernels
- Parameter-sharing: Lower number of parameters
- Translation invariant (equivariance): position does not matter for feature representation

Convolution Neural Networks (CNNs)

CNNs is a specialized kind of architecture for processing grid-like data, e.g. images (2-d) or time series (1-d) used for (supervised) image classification

A CNN is defined as a network that uses at least one convolutional layer
$\rightsquigarrow$ A layer that uses convolution instead of general matrix multiplication

A typical convolutional layer has three steps:

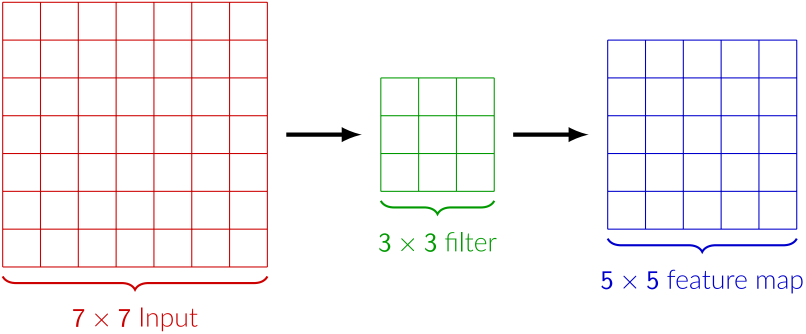
1. Convolution
2. Activation
3. Pooling

The Convolution Operation

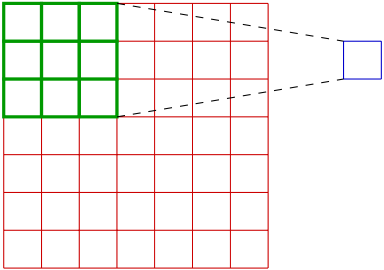
The convolution has three ingredients:

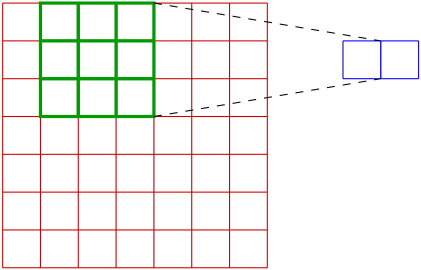
- Input image M with dimension $N \times N$
- Filter K
- Stride S

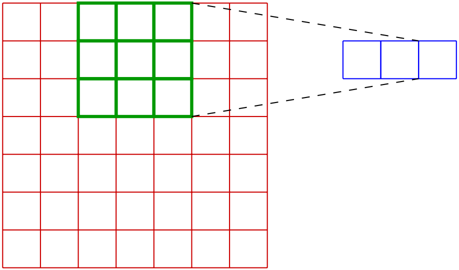
⇝ The filter is slided over M with stride S to produce feature map

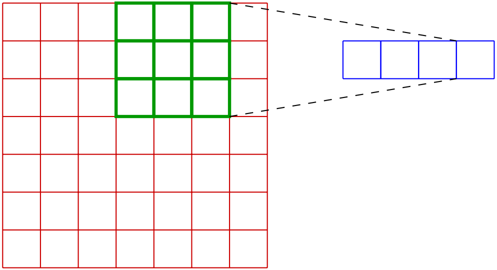


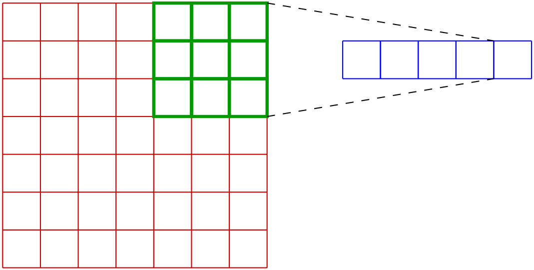
Convolution Operation

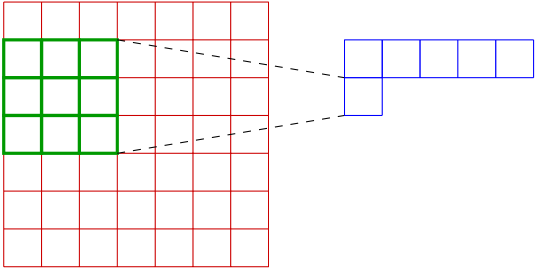


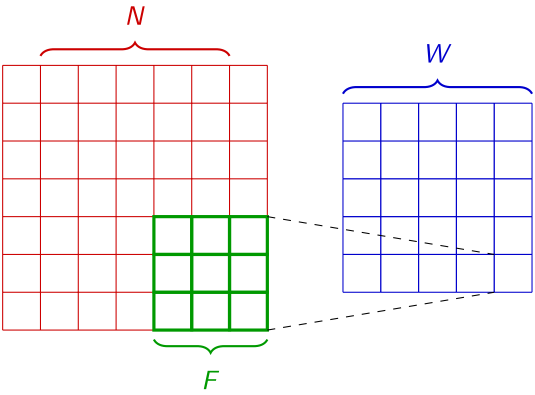












The width of the feature map is represented by the formula:

$$W = \frac{N + 2P - F}{S} + 1$$

where P is the width of the padding and S is the stride

Convolution Exercise

You have:

- $128 \times 128 \times 3$ input
- One 5×5 filter
- Padding: 2
- Stride: 1

$W = \frac{N+2P-F}{S} + 1$

Questions:

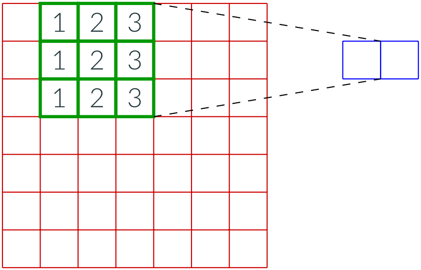
1. What's the output dimension of the feature map?
2. What's the number of parameters?
3. What if $F = N + 2P$?

Calculation

Filter:

0	0	1
0	1	0
1	0	0

Input matrix:



$$= \begin{bmatrix} 1\cdot 0 & 2\cdot 0 & 3\cdot 1 \\ 1\cdot 0 & 2\cdot 1 & 3\cdot 0 \\ 1\cdot 1 & 2\cdot 0 & 3\cdot 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 3 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$= 6$$

Filters/Kernels

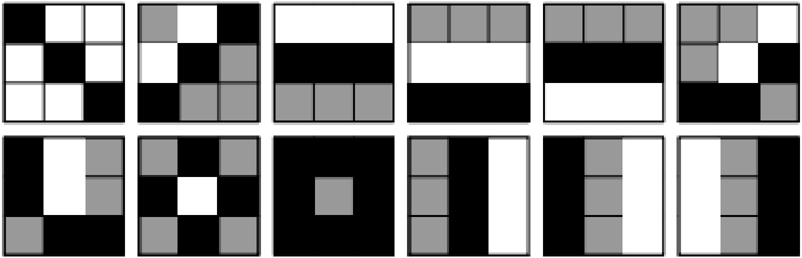
The filters are the core of the convolution ⇝ the weight matrices for CNNs

The purpose of filters is to simultaneously:

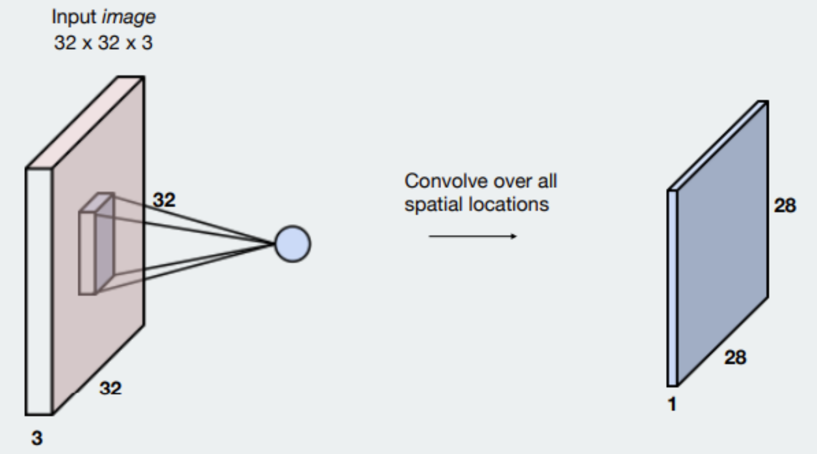
1. Compress the image
2. Learn features

⇝ Encoded by the feature map, which quantifies how much a spatial location resembles the filter (e.g. lines, corners, shapes)

Example of 3×3 filters:

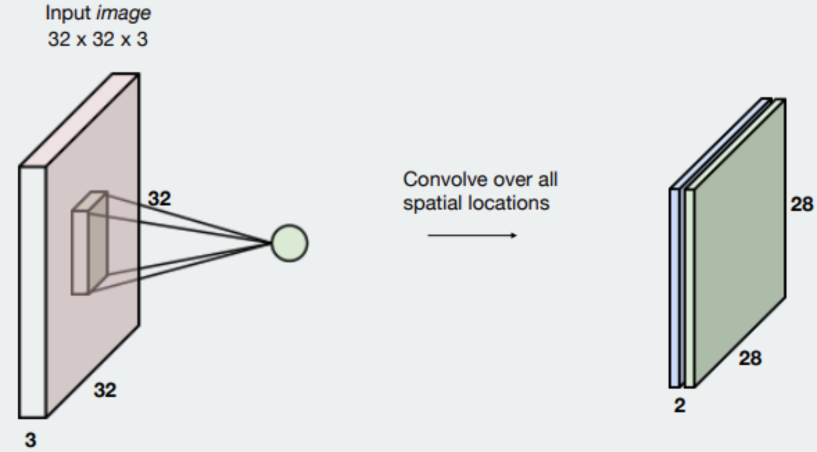


One Filter

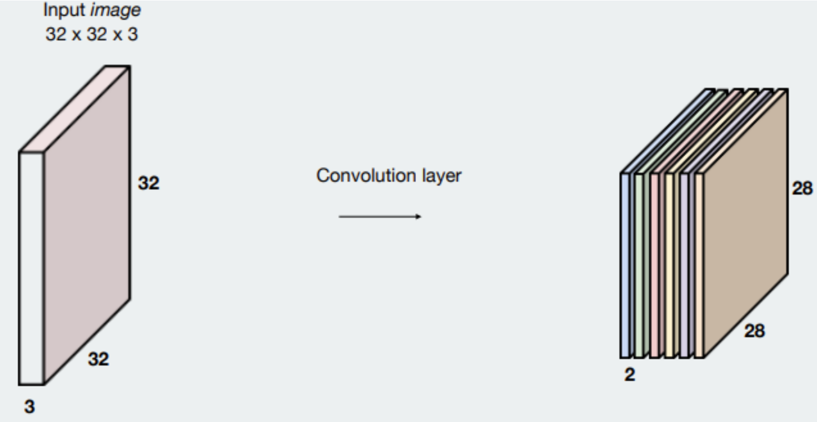


Input *image*
32 x 32 x 3
32
32
3
Convolve over all spatial locations
28
28
1

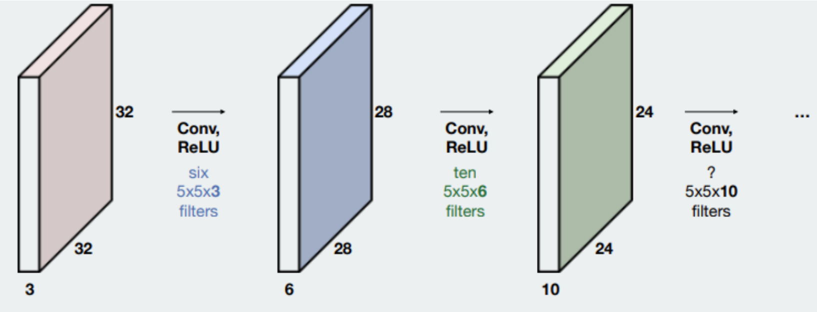
Two Filters


Input *image*
32 x 32 x 3
32
32
3
Convolve over all
spatial locations
28
28
2

Multiple Filters



Stacking Convolutional Layers



Pooling

After convolution, CNN typically uses a pooling operation, which compresses the image even more.

How the pooling operation compresses the output depends on whether we use max, min, or mean pooling.

The pooled feature map is achieved in a similar way as the convolution, namely by sliding a grid over the map.

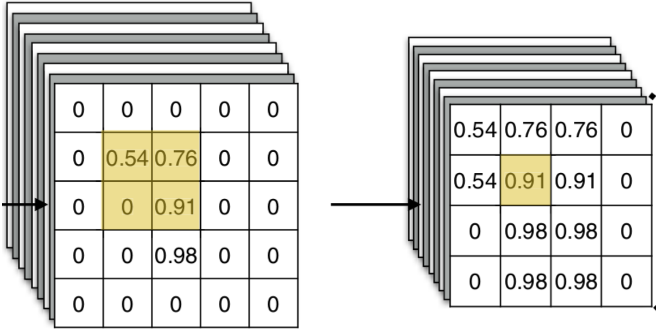
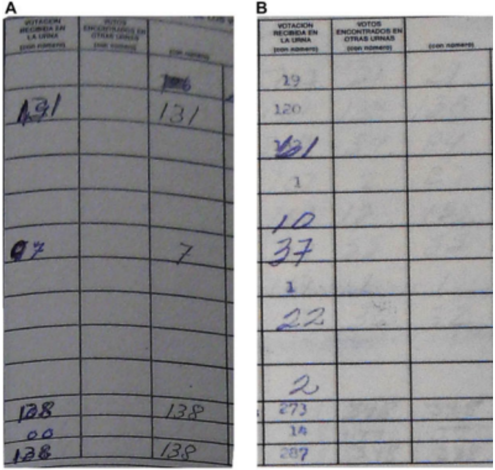


Table of Contents

Example: Handwriting Recognition and Electoral Fraud (Cantú, 2019)



Training a Neural Network

The task:
To train a model h using $\mathcal{D}_{\texttt{train}}$ that learns a function f to make good predictions on $\mathcal{D}_{\texttt{new}}$

The challenge:
h should approximate f in general and not just on $\mathcal{D}_{\texttt{train}}$

The solutions:

1. Put restrictions on the capacity of h $\rightsquigarrow$ i.e. regularize the learning
2. Reuse already good models $\rightsquigarrow$ i.e. transfer learning

Regularization

Regularization is an umbrella term for strategies used to reduce generalization error:

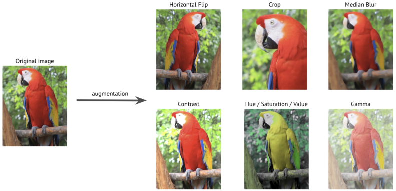
- Parameter norm penalty
- Dropout
- Augmentation
- Early stopping

Regularization Techniques

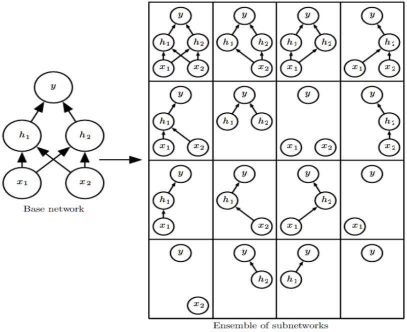
Norm-regularization:

$C = \text{Loss} + \text{Regularization}$

Augmentation:

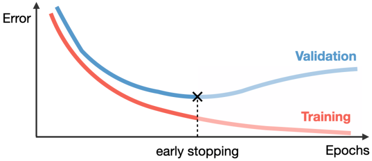


Dropout:



Early Stopping:

Stop training when performance on the validation dataset starts worsening



Transfer Learning

- Take pretrained model $h_{\texttt{pretrained}}$ and use on your data $\mathcal{D}_{\texttt{new}}$
  1. Reuse 1:1
  2. Freeze layers and retrain a few
- Makes training much faster
- Combats overfitting since we avoid local minima
  1. Makes training faster
  2. Avoids overfitting

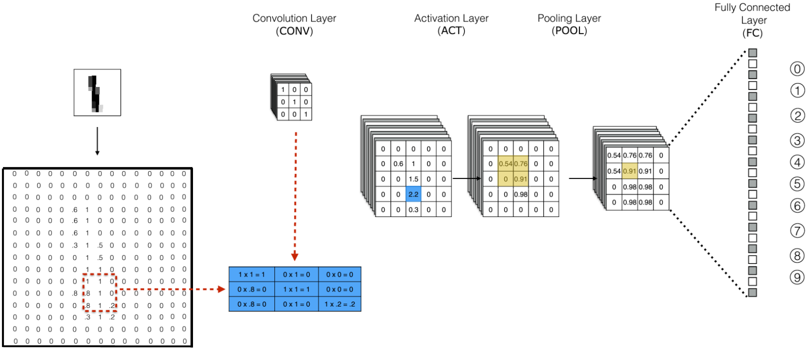


Table of Contents

See you next week!

Topic 4: Images

Computational Analysis of Text, Audio, and Images, Fall 2023

Aarhus University

References i

[1] F. Cantú, “The fingerprints of fraud: Evidence from mexico’s 1988 presidential election,” *American Political Science Review*, vol. 113, no. 3, pp. 710–726, 2019.