Convolutional Neural Networks

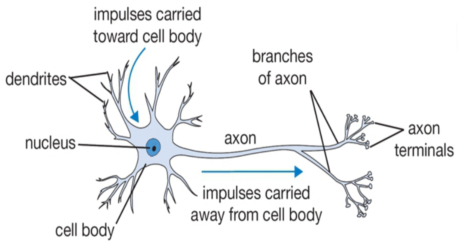
Julius Taylor
5210444
s8423760@stud.uni-frankfurt.de
Shawn Cala
4921431
shawn.cala@gmail.com

February 5, 2017

Table of Contents

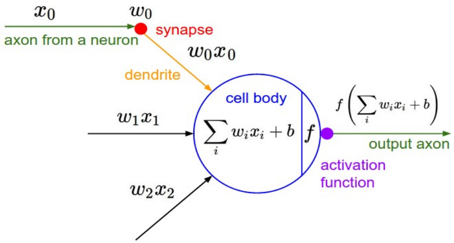
Neural Networks

Primary motivation: Neural Networks mathematically simulate biological functionalities of the human brain



impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

(a) biological model



(b) mathematical representation

Figure: neuronal model and computational abstraction

Structure

Neural Networks generally contain:

- an n-dimensional input
- one or many layers of interconnected neurons
- an output-layer

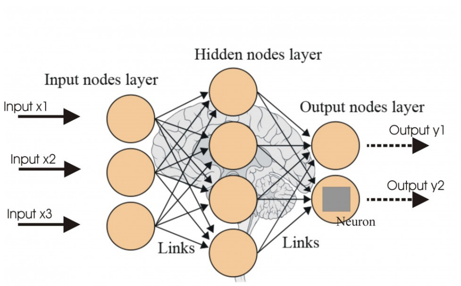


Figure: Basic concept of a Neural Network

The Problem Space

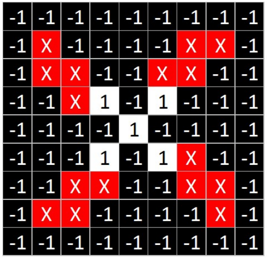
- image classification: Image $\rightarrow$ class that best describes the image (eg. 'Cat', 'Ship')
- recognizing patterns and generalizing from prior knowledge
- figure out features, that describe things

Problems with image classification

- naive approach: comparing pixels one by one
- very sensitive to small changes and does not generalize

$$\begin{matrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{matrix}
\overset{?}{=}
\begin{matrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{matrix}$$

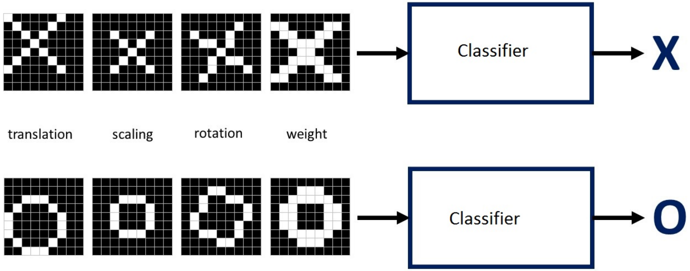
Problems with image classification



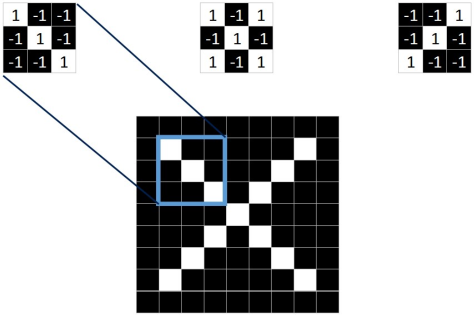
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	-1	X	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure: Visualizing what computers see

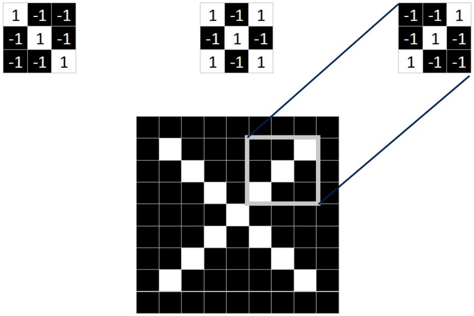
Problems with image classification



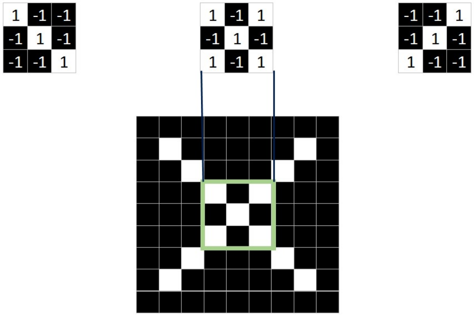
What makes us conclude that two pictures match?



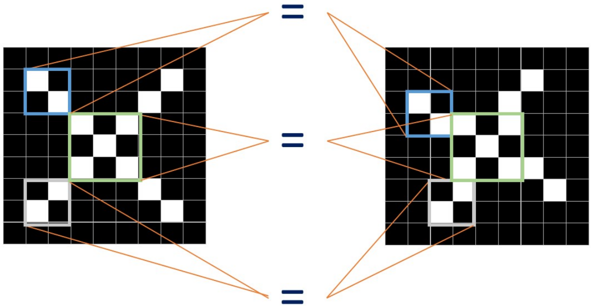
What makes us conclude that two pictures match?



What makes us conclude that two pictures match?



What makes us conclude that two pictures match?



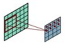
Solution: CNN's

Convolutional Neural Networks (CNNs) are a subtype of Neural Networks:

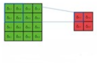
- is a type of feed-forward network
- structure inspired by the animal visual cortex
- uses many identical copies of the same neuron
- express computationally extensive models while keeping weights to learn small
- breakthrough results in pattern recognition problems

Structure

Convolution

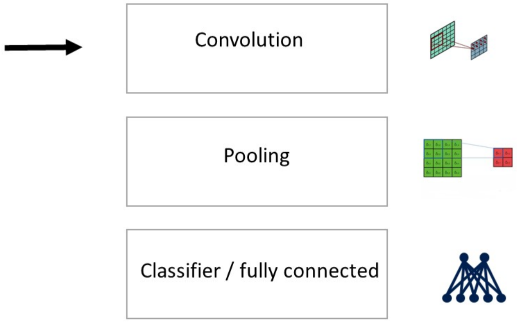


Pooling



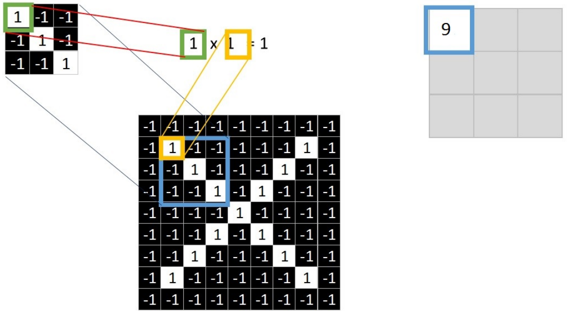
Classifier / fully connected

Structure



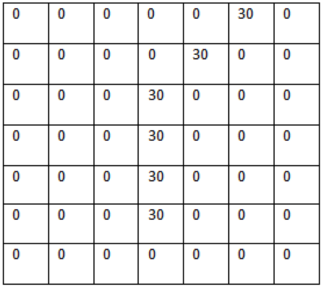
Finding Features

- We find features in a picture by filtering



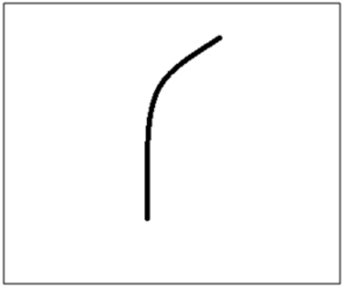
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Finding Features



0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

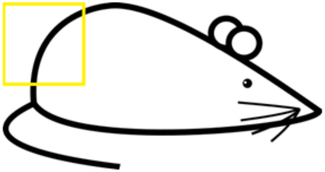
Pixel representation of filter



Visualization of a curve detector filter

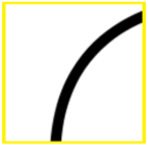
Finding Features

Original image

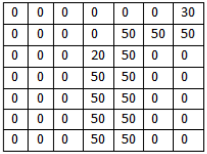


Visualization of the filter on the image

Finding Features



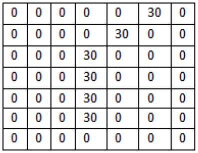
Visualization of the receptive field



0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

$*$



0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Finding Features

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

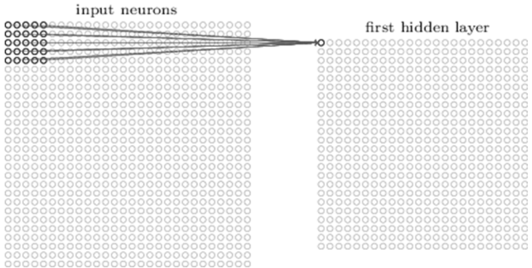
$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

Finding Features



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

$$(f * g)(c_1, c_2) = \sum_{a_1,a_2} f(a_1, a_2) \cdot g(c_1 - a_1,\ c_2 - a_2)$$

First Trick: Convolutions

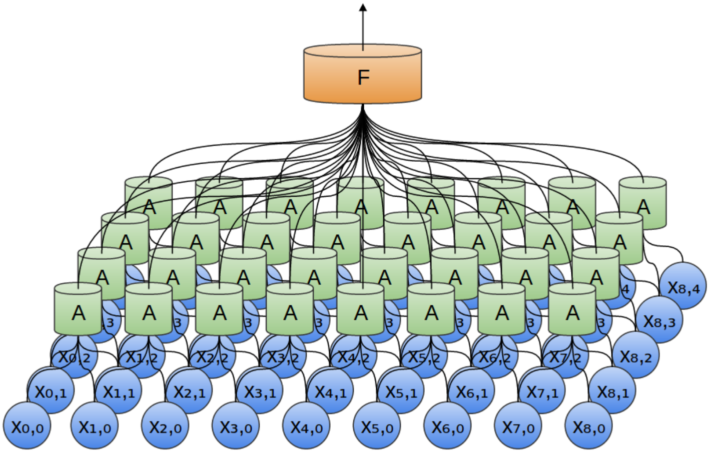
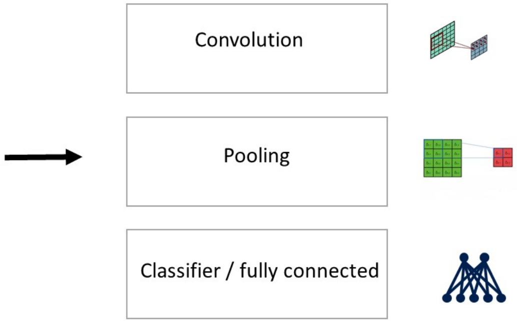

F
A
X0,0 X1,0 X2,0 X3,0 X4,0 X5,0 X6,0 X7,0 X8,0
X0,1 X1,1 X2,1 X3,1 X4,1 X5,1 X6,1 X7,1 X8,1
X0,2 X1,2 X2,2 X3,2 X4,2 X5,2 X6,2 X7,2 X8,2
X8,3
X8,4

Figure: A convolutional layer fed into a fully connected layer

Structure



Second Trick: Pooling

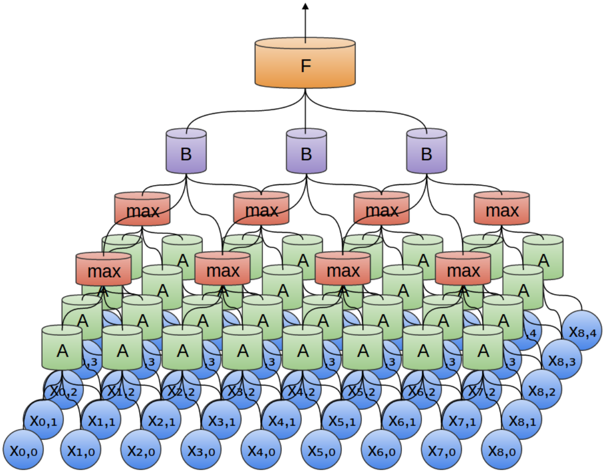
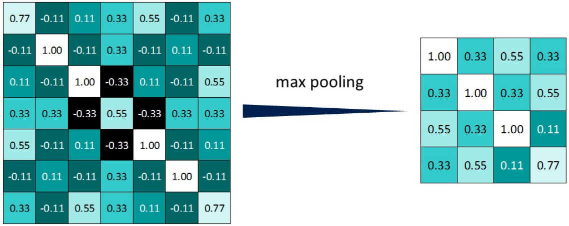


Figure: Added max pooling to previous net

Second Trick: Pooling

- 'zooms' out
- small patch after pooling corresponds to much larger patch before it
- makes the network a little more invariant to location of features

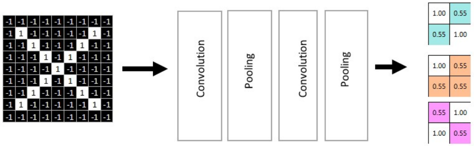


0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

Stacking Layers

- layers are composable
- we can feed the output of one layer into the input of another
- with each layer, we can detect higher level, more abstract features



Stacking Layers

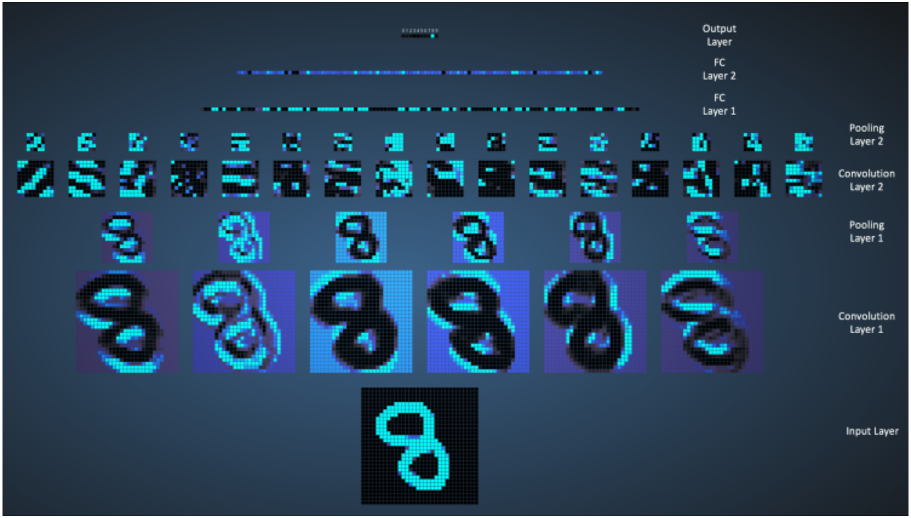
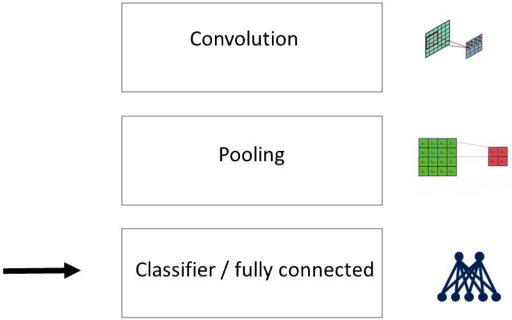


Figure: Visualizing a ConvNet trained on handwritten digits

Structure



Third Trick: Fully Connected

- every value in a fully connected layer gets a 'vote' which affects the result of classification
- values in certain positions are associated with a certain class

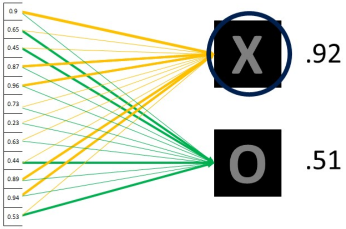
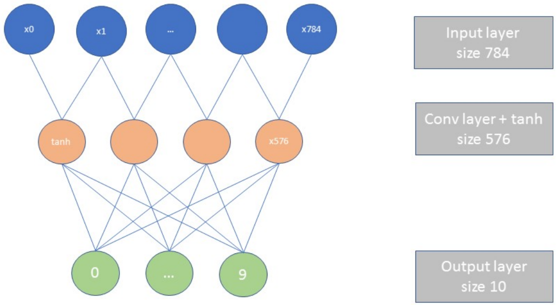
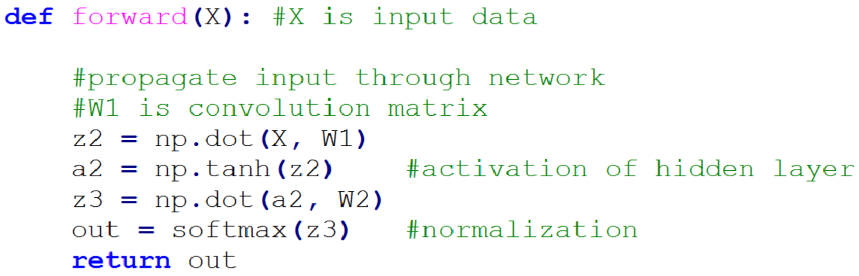


Figure: Weights in a fully connected layer contributing to classification

Our approach



Our approach - forward propagation



```
def forward(X): #X is input data

    #propagate input through network
    #W1 is convolution matrix
    z2 = np.dot(X, W1)
    a2 = np.tanh(z2)      #activation of hidden layer
    z3 = np.dot(a2, W2)
    out = softmax(z3)     #normalization
    return out
```

Activation function

Activation-functions are typically used to add nonlinearity to the network

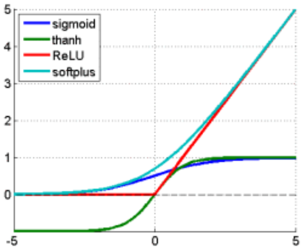


Figure: Different activation-functions

- activation functions must be non-linear and differentiable
- the classification network uses the tangens-hyperbolicus function

Fully connected layer

The program's output layer is fully connected:

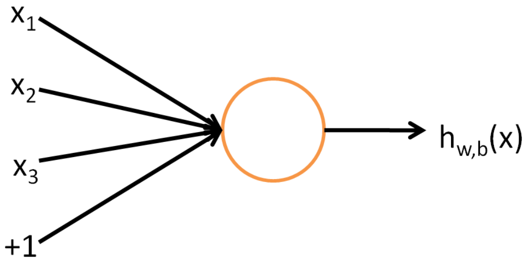
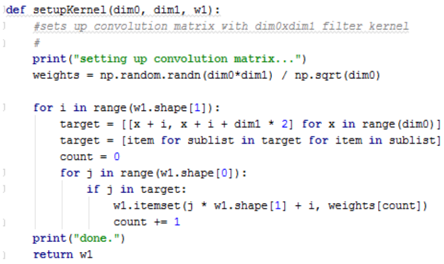


Figure: Single neuron in a fully connected layer

Output: $(\sum_i w_i x_i)$

Convolutional layer

Since iterating over the image each time is computationally inefficient, a mathematical representation is formed:



```
def setupKernel(dim0, dim1, w1):
    #sets up convolution matrix with dim0xdim1 filter kernel
    #
    print("setting up convolution matrix...")
    weights = np.random.randn(dim0*dim1) / np.sqrt(dim0)

    for i in range(w1.shape[1]):
        target = [[x + i, x + i + dim1 * 2] for x in range(dim0)]
        target = [item for sublist in target for item in sublist]
        count = 0
        for j in range(w1.shape[0]):
            if j in target:
                w1.itemset(j * w1.shape[1] + i, weights[count])
                count += 1
    print("done.")
    return w1
```

- Input: $conv = x * W_1 + b_1$
- Output: $out = tanh(conv)$

Training

The program was trained with backpropagation:

Backpropagation is the key algorithm which makes training neural networks feasible by greatly increasing the efficiency of learning algorithms.
To understand backpropagation, it is necessary to introduce the concept of forward propagation:

Forward propagation

Forward propagation is the process of computing the output of a network for a given input:

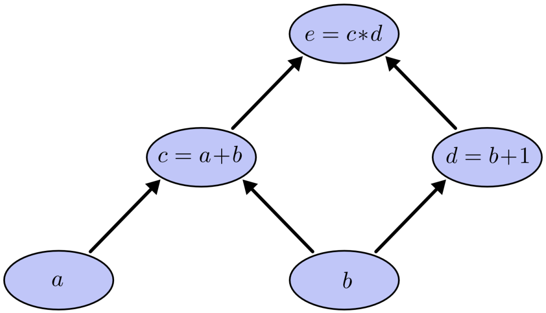


Figure: Computational graph of $e = (a+b) \cdot (b+1)$

Forward propagation

The input is arbitrarily set to $a = 2$ and $b = 1$

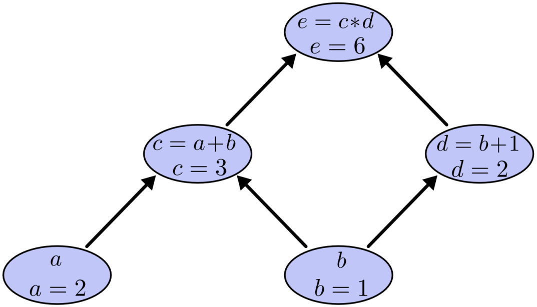


Figure: Computational graph of $e = (a + b) \cdot (b + 1)$

Forward propagation

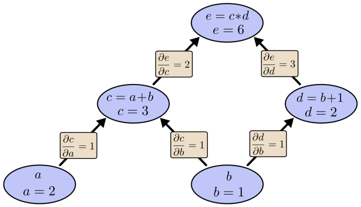
- To train network weights one must determine how the output is influenced by its layer weights.
- This is achieved by finding the respective partial derivatives

c is only indirectly influenced by a and b. To derive the exact influence, the chain rule can be applied:

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a}$$

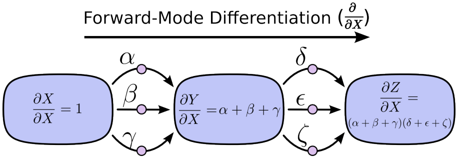
$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial d} \cdot \frac{\partial e}{\partial c} \cdot \frac{\partial d}{\partial b} \cdot \frac{\partial c}{\partial b}$$

Forward propagation



$e = c*d$
$e = 6$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$c = a+b$
$c = 3$
$d = b+1$
$d = 2$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
a
$a = 2$
b
$b = 1$

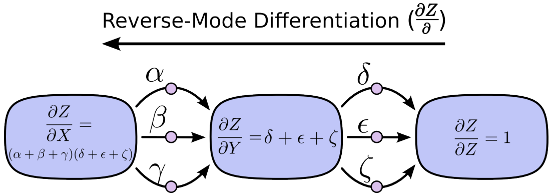
- This approach works for training networks
- Problem: exponential number of derivatives might be necessary



Forward-Mode Differentiation ($\frac{\partial}{\partial X}$)
$\frac{\partial X}{\partial X} = 1$
α
β
γ
$\frac{\partial Y}{\partial X} = \alpha + \beta + \gamma$
δ
ϵ
ζ
$\frac{\partial Z}{\partial X} =$
$(\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$

Solution: Backpropagation

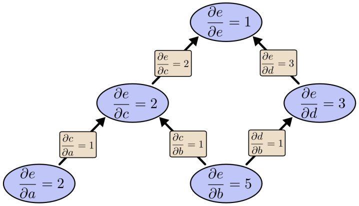
Instead of determining the influence of each input on the output, the influence of the layers on the output is influenced by the previous layers.



Reverse-Mode Differentiation ($\frac{\partial Z}{\partial}$)
α
β
γ
δ
ϵ
ζ
$\frac{\partial Z}{\partial X} = (\alpha+\beta+\gamma)(\delta+\epsilon+\zeta)$
$\frac{\partial Z}{\partial Y} = \delta+\epsilon+\zeta$
$\frac{\partial Z}{\partial Z} = 1$

Backpropagation

In the previous example executing a backward differentation immediately gives us the derivative of the output relative to every input:



$\frac{\partial e}{\partial e} = 1$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial e}{\partial a} = 2$
$\frac{\partial e}{\partial b} = 5$

This allows for massive speedup in networks with many inputs

Backpropagation

Math behind the classification program:

Convolution:	$conv = x \cdot W_1 + b_1$
Convolutional layer output:	$out = tanh(conv)$
Fully connected layer output:	$\hat{y} = out \cdot W_2 + b_2$
Loss function:	$L = -\sum_N y \cdot \ln \hat{y}$

The training function will attempt to minimize the loss function in regards to the parameters.

Backpropagation

The four trainable parameters in the program are:

1. W_1: weight of convolutional layer
2. W_2: weight of fully connected layer
3. b_1: offset of convolutional layer
4. b_2: offset of fully connected layer

W_2: Weight of fully connected layer

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W_2}$$

$$\frac{\partial L}{\partial \hat{y}} = \sum_N y \cdot \frac{\partial \ln \hat{y}}{\partial \hat{y}} = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial W_2} = \frac{\partial out \cdot \hat{W}_2 + b_2}{\partial W_2} = out^T$$

$$= (\hat{y} - y) * out^T$$

b_1, b_2: Offset of both layers

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial out} \cdot \frac{\partial out}{\partial conv} \cdot \frac{\partial conv}{\partial b_1} =$$

$$= (1 - tanh^2 conv) \cdot (y - \hat{y}) \cdot W_2 \cdot 1$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_2}$$

$$= (\hat{y} - y) \cdot 1$$

W_1: Weight of convolutional layer

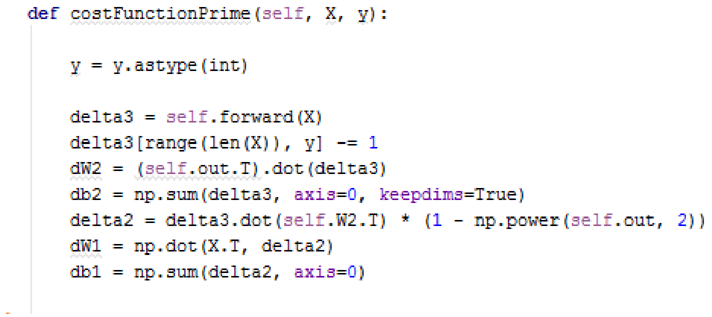
$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial out} \cdot \frac{\partial out}{\partial conv} \cdot \frac{\partial conv}{\partial W_1}$$

$$\frac{\partial \hat{y}}{\partial out} = \frac{\partial out \cdot \hat{W}_2 + b_2}{\partial out} = W_2$$

$$\frac{\partial out}{\partial conv} = \frac{\partial tanh(conv)}{\partial conv} = 1 - \tanh^2 conv \quad \frac{\partial conv}{\partial W_1} = x^T$$

$$\frac{\partial L}{\partial W_1} = x^T \cdot (1 - tanh^2 conv) \cdot (y - \hat{y}) \cdot W_2$$

Backpropagation in the program



```
def costFunctionPrime(self, X, y):

    y = y.astype(int)

    delta3 = self.forward(X)
    delta3[range(len(X)), y] -= 1
    dW2 = (self.out.T).dot(delta3)
    db2 = np.sum(delta3, axis=0, keepdims=True)
    delta2 = delta3.dot(self.W2.T) * (1 - np.power(self.out, 2))
    dW1 = np.dot(X.T, delta2)
    db1 = np.sum(delta2, axis=0)
```

Figure: Backpropagation in the program

Results

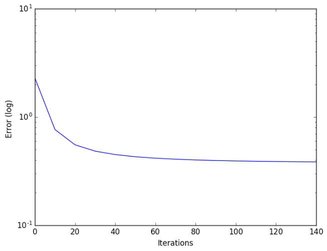


Figure: Training results

- The classification program was trained with 60000 images of digits from the MNIST database
- Training results were tested by classifying 10000 other images
- After training the program classifies images with an accuracy of approximately 92%

Results

Epoch	Loss	Accuracy % (test)	Accuracy % (training)
0	2.30317399	6.69	6.42
1	2.1117	50.58	50.291
2	1.9537	75.89	75.178
3	1.80576	76.53	75.8
4	1.666671	77.33	76.208
5	1.54040	76.426	77.63
6	1.4305	77.96	76.723
7	1.337965	78.56	77.095
8	1.26149	78.88	77.538
9	1.1987319	79.24	77.98
10	1.14714	79.72	78.426
...			
30	0.408457	86.27	86.713
...			
45	0.397866	88.46	88.99
...			
150	0.38695	91.555	91.718

Figure: Training results

Application

Visual Object Recognition

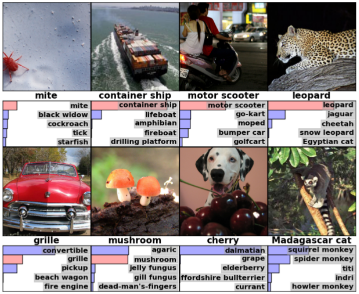


Figure: ImageNet Classification with Deep Convolutional Neural Networks

ImageNet by Krizhevsky et al (2012) classified 1.2 million high-resolution images in the ImageNet LSVRC-2010 into 1000 different classes. It achieves error rates of 37.5% for the top result and 17.0% for the top-5 results

Application

Text Classification

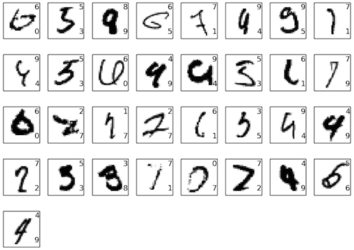


Figure: Excerpt of MNIST

http://blog.christianperone.com/2015/08/convolutional-neural-networ
http://cs231n.github.io/neural-networks-1/#bio
http://futurehumanevolution.com/artificial-intelligence-future-huma
http://colah.github.io/posts/2014-07-Conv-Nets-Modular/
https://bfeba431-a-62cb3a1a-s-sites.googlegroups.com/site/deeplearn
http://briandolhansky.com/blog/2013/9/27/artificial-neural-networks
http://neuralnetworksanddeeplearning.com/chap2.html
http://colah.github.io/posts/2014-07-Understanding-Convolutions/
http://neuralnetworksanddeeplearning.com/chap6.html
http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetwo
https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearities
http://colah.github.io/posts/2014-07-Understanding-Convolutions/
https://colah.github.io/posts/2015-08-Backprop/
http://cs231n.github.io/convolutional-networks/
http://colah.github.io/posts/2014-07-Conv-Nets-Modular/
http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

Additional Content which was overkill for the main presentation:

Overfitting

Overfitting occurs when the complexity of the model relative to the training size is too high.
The model begins memorizing the training data rather than the underlying principle and easily loses its predictive power when the input is slightly altered.
Overfitting is prevented in a number of ways:

1. max-pooling layers
2. dropout layers
3.

Components - Neuron in convolutional layer

To understand the concept behind Convolution we first demonstrate it in an example:

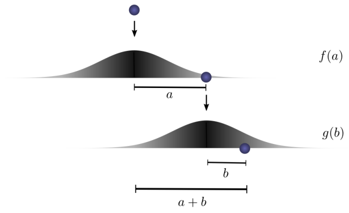


Figure: Probability distribution of ball thrown twice

We want to calculate the likelihood of the ball traveling a distance c after being thrown twice in a row.

Input

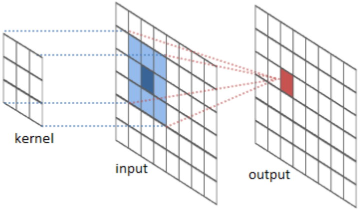
Input layer

The input layer consists of an n-dimensional matrix, which contain the information to be processed

- Our example uses a 2-dimensional input which represents the pixels of the images
- asd

Input:
a, b: length of first and second throw $f(a), g(b)$: probability distribution of the balls location If we want $c = a + b$ the probability of this event is $f(a) \cdot g(b)$
Since only the actual outcome is relevant to us, the actual values of a and b can be varied.
The total likelihood of the ball landing at c can also be summed over all probability distributions where $a + b = c$:
$f * g(c) = \sum_{a+b=c} f(a) \cdot g(a)$
The result of this sum is a convolution between a and b

Image processing with a kernel function can also be described as a



convolution:

Figure: Convolution between an input image and a kernel function

$$out = \sum_a \sum_b w_{ab} \cdot x$$

Layers - Convolution

The neurons of the Convolutional Layer is connected to the output of the previous layer through the kernel-function.

RELU/tanh layer

because it is not bounded or continuously differentiable. The rectified linear activation function is given by, f(z)=max(0,x).
To make backpropagation easier we used a differentiable function similar to RELU: $y = ln(1 + e^x)$

Max-Pooling layer

The Max-Pooling layer partitions the image, returns the maximum value of each partition and returns a compressed version of the input. This is done to a) reduce overfitting and b) downsampling the input

Dropout Layer

The dropout layer is a fully connected layer. It reduces overfitting and downsamples the input by randomly dropping neurons and their connections during training.

Dense layer

The dense layer is identical to a normal neural network layer in that it is fully interconnected with every neuron of the previous layer.
image here

Output layer

The output layer is fully connected and unambigously assigns each neuron of the output to one of the expected results.
image here