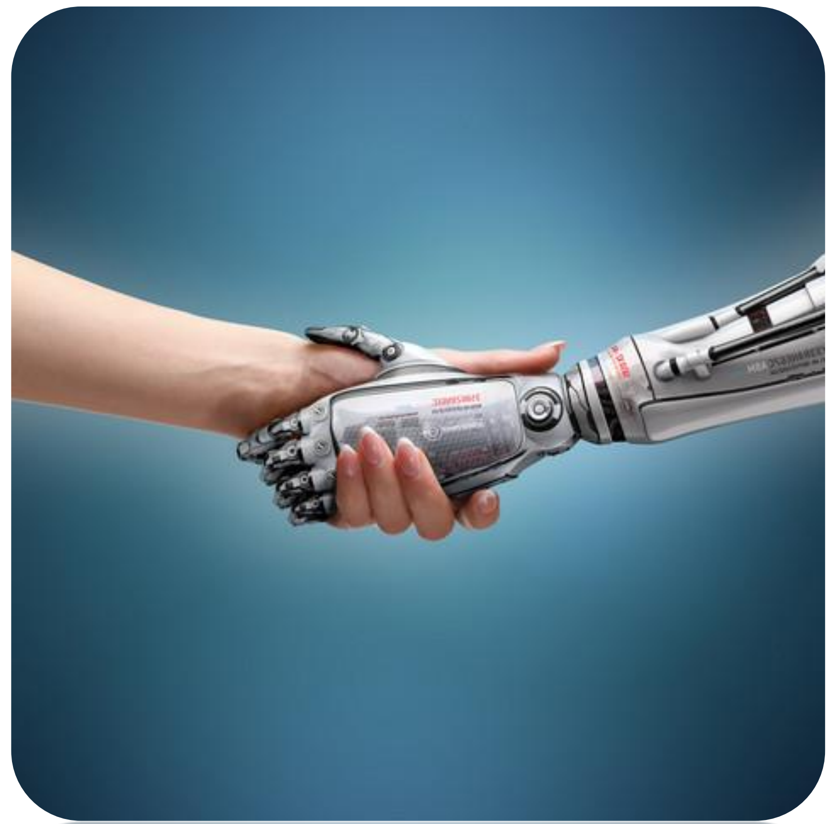
Zurich University
of Applied Sciences

Artificial Intelligence
V11: Generative Modeling with Neural Nets

Brief overview of neural networks
Generative Adversarial Nets
Use case: image inpainting



With material from
- Stuart Russell, UC Berkeley
- Arthur Juliani’s and Brandon Amos’s blog posts
- Ian Goodfellow, UC Berkeley COMPSCI 294 guest lecture

OpenAI

Zurich University of Applied Sciences and Arts
InIT Institute of Applied Information Technology (stdm)

Zurich University
of Applied Sciences

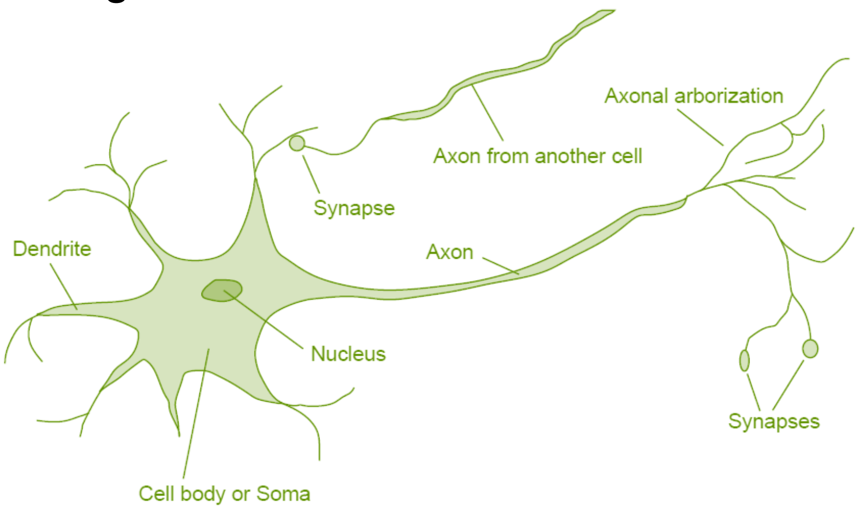
Educational objectives

- Have a **basic** understanding of the architecture and working **of neural networks**
- Know the **general idea** behind Generative Adversarial Nets (**GAN**s)
- Understand the **training** process (and inherent difficulties) **for GANs**
- Be able to start **working on open source GAN** code

1. BRIEF OVERVIEW OF NEURAL NETWORKS

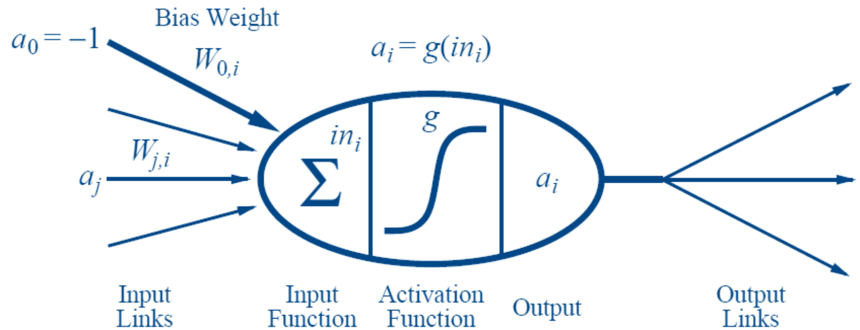
Neurons

Biological model



- 10^{11} neurons of >20 types
- 10^{14} synapses
- 1ms – 10ms cycle time
- Signals are noisy "spike trains" of electrical potential
- Organized in layers to form a brain

McColloch-Pitts "unit" (1943)

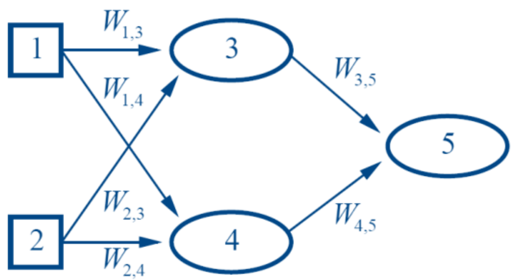


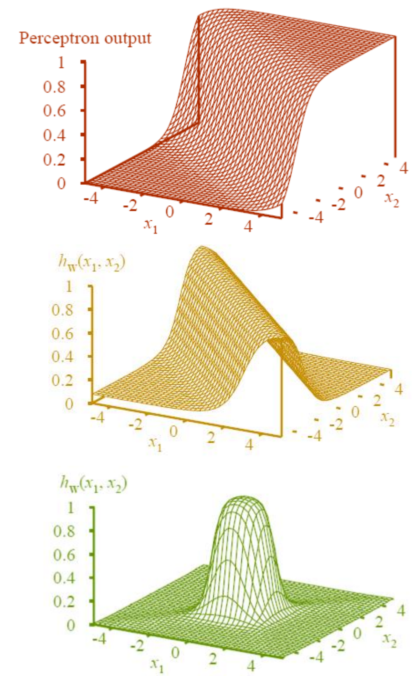
- Output is a **thresholded linear function** of the inputs: $a_i = g(in_i) = g\left(\sum_j W_{j,i} \cdot a_j\right)$
- Changing the bias weight $W_{0,i}$ moves the threshold location
- A **gross oversimplification** of real neurons!
- Purpose: develop understanding of what **networks of simple units** can do

Feed-forward network example

FNN: a parameterized family of nonlinear functions

- $a_5 = g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4)$
  $= g\left(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)\right)$





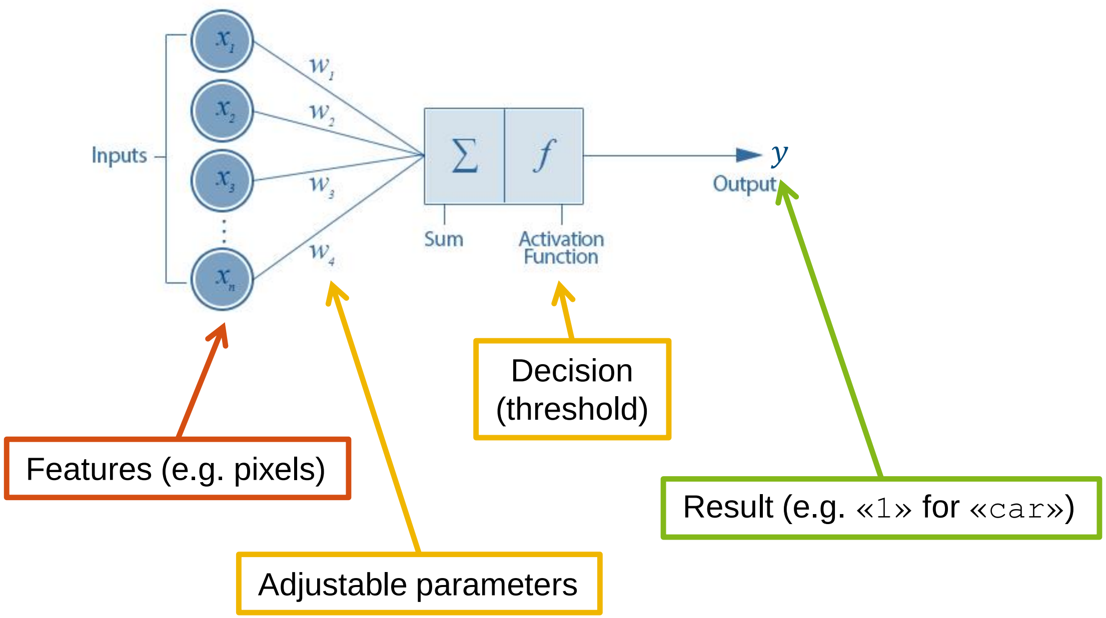
- **Adjusting weights** changes the function: **learning** works this way!
  (→ see appendix for first ideas)

Expressiveness of multilayer networks (multilayer perceptrons)

- All continuous functions w/ 2 layers, all functions w/ 3 layers
  - Combine two opposite-facing threshold functions to make a ridge
  - Combine two perpendicular ridges to make a bump
  - Add bumps of various sizes and locations to fit any surface

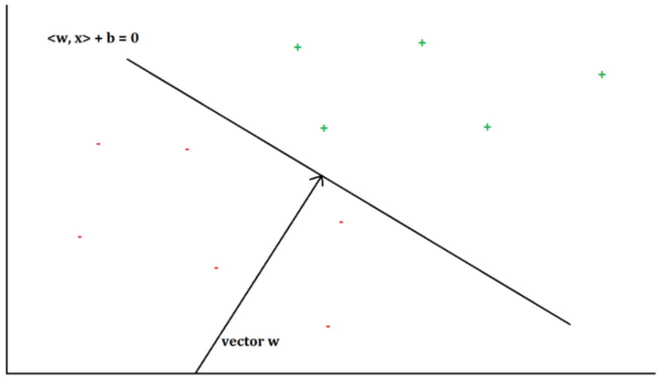
What is the effect of weight adjustment?

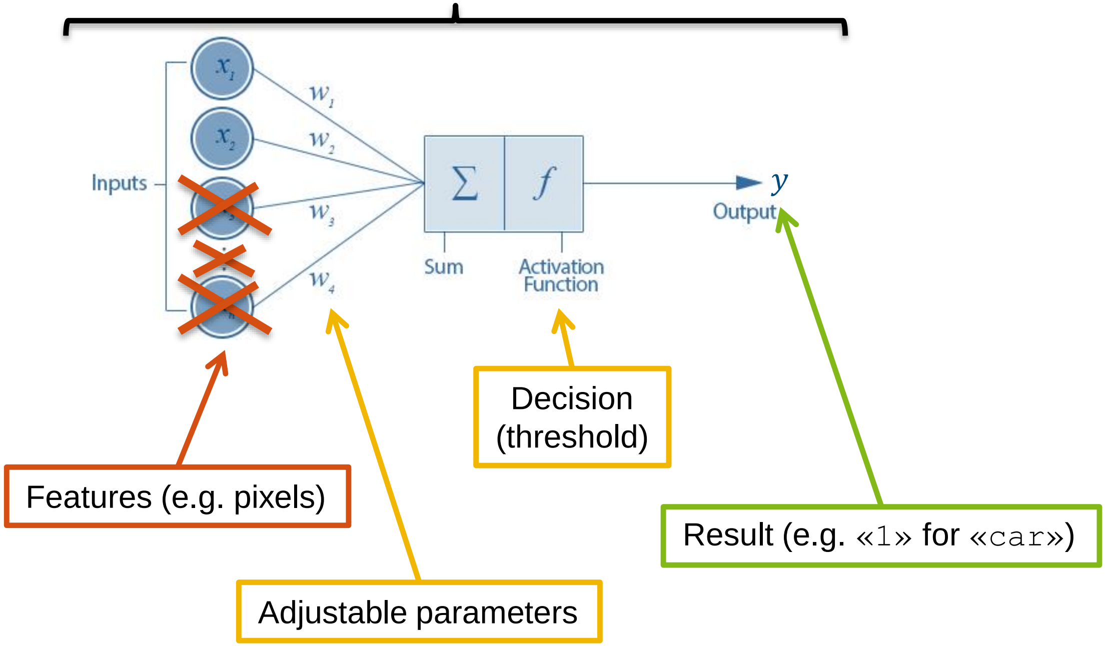
Neuron



What is the effect of weight adjustment?

Neuron



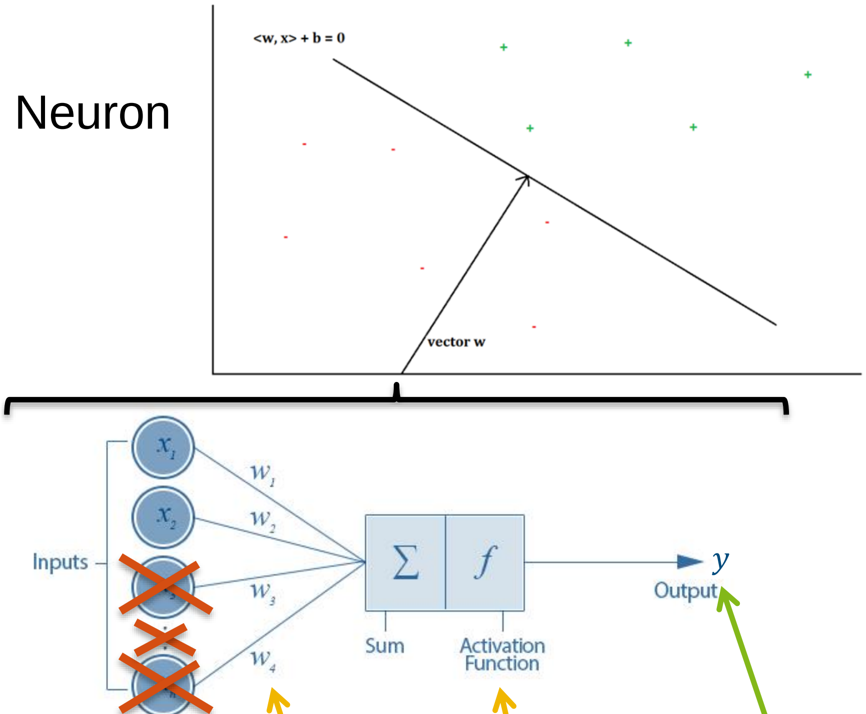


Zurich University
of Applied Sciences

What is the effect of weight adjustment?

Neuron

Neural Network

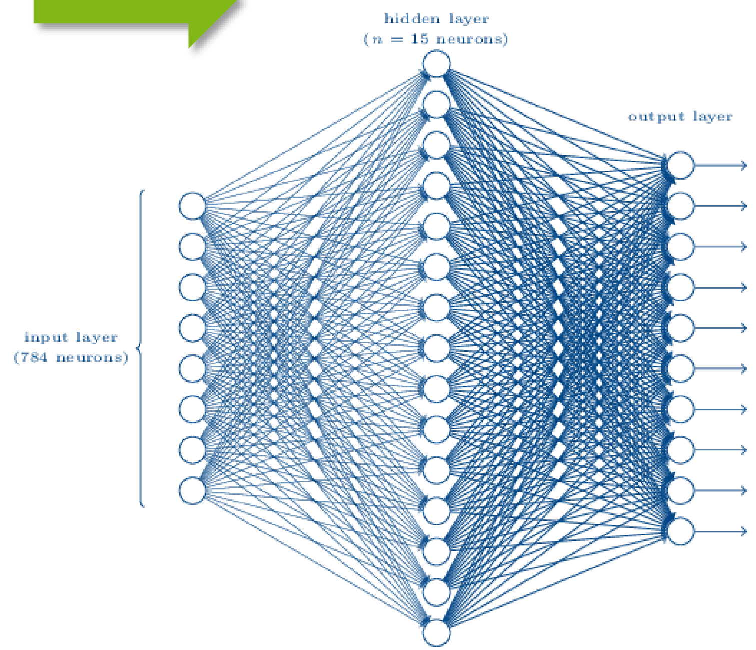


Features (e.g. pixels)

Adjustable parameters

Decision
(threshold)

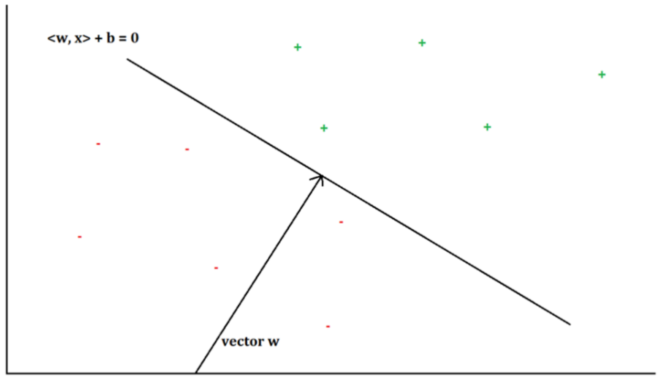
Result (e.g. «1» for «car»)

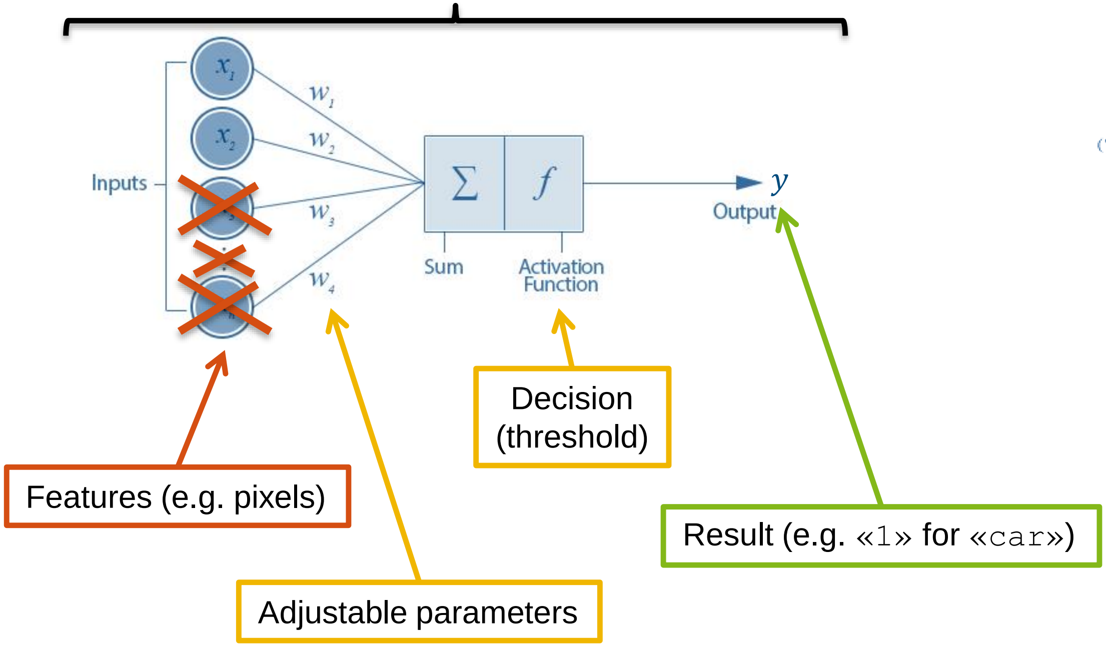


Zurich University
of Applied Sciences

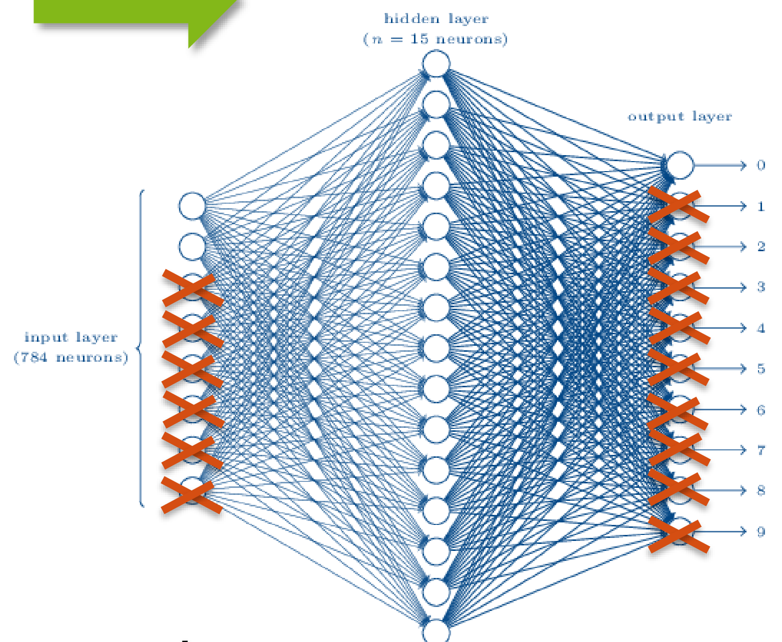
What is the effect of weight adjustment?

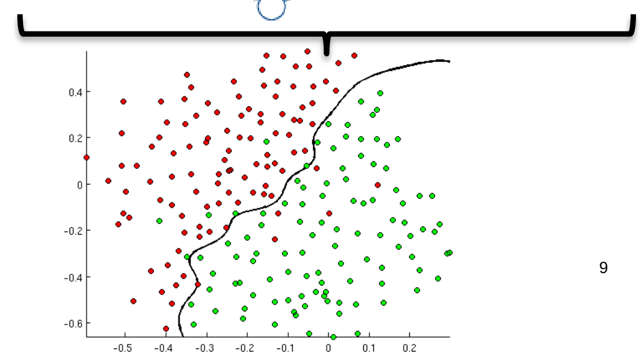
Neuron





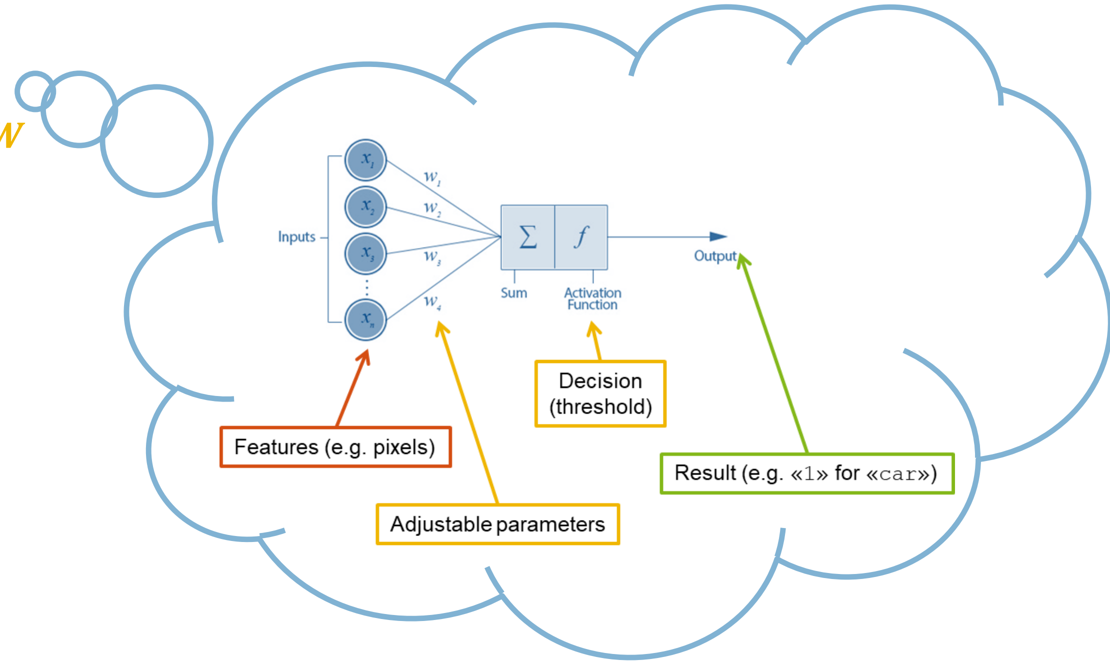
Neural Network





How are the weights adjusted?
First intuition

- Our example neural network: $f_{\boldsymbol{W}}(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \ldots\}$ initialized randomly)



Zurich University of Applied Sciences

How are the weights adjusted?
First intuition

- Our example neural network: $f_{\boldsymbol{W}}(x) = y$ with image x, ground truth y und parameters $\boldsymbol{W}$ ($\boldsymbol{W} = \{w_1, w_2, \ldots\}$ initialized randomly)

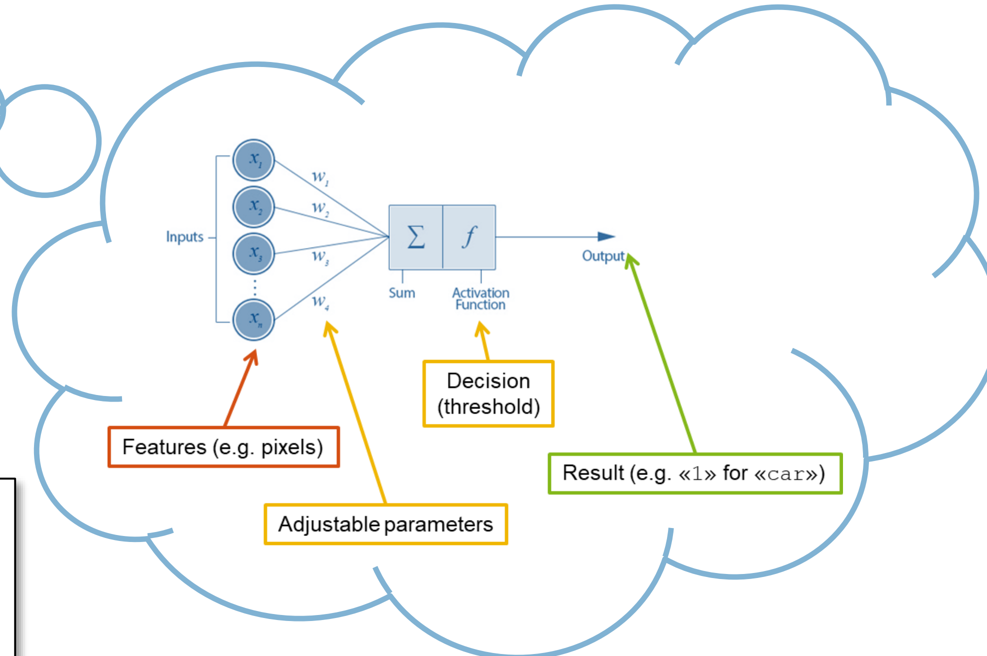
- Error measure: $L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N}(f_{\boldsymbol{W}}(x_i) - y_i)^2$ Average of quadratic difference on all images (loss function)

$$L(\boldsymbol{W}) = \underbrace{\frac{1}{N}\sum_{i=1}^{N}}_{\text{Average (of all examples)}} \underbrace{(f_{\boldsymbol{W}}(x_i) - y_i)}_{\text{Difference prediction – ground truth (error)}}\underbrace{{}^2}_{\text{Overproportional penalty for large errors}}$$

Average (of all examples)

Difference prediction – ground truth (error)

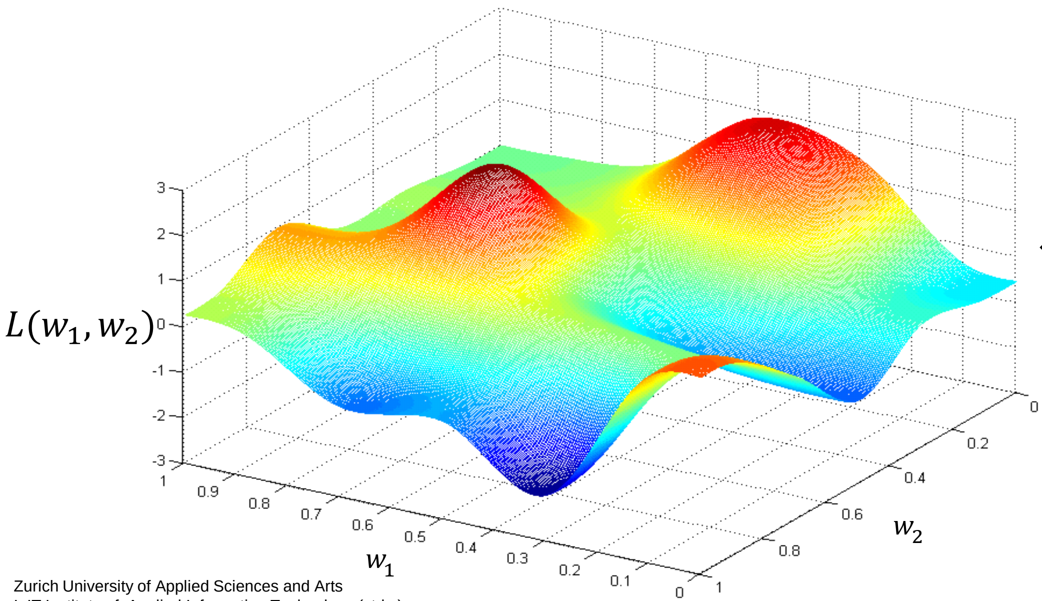
Overproportional penalty for large errors



How are the weights adjusted? (contd.)
First intuition

- Our example neural network: $f_W(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \dots\}$ initialized randomly)

- Error measure: $L(W) = \frac{1}{N}\sum_{i=1}^{N}(f_W(x_i) - y_i)^2$
  Average of quadratic difference on all
  images (loss function)



🡐 error «landscape»

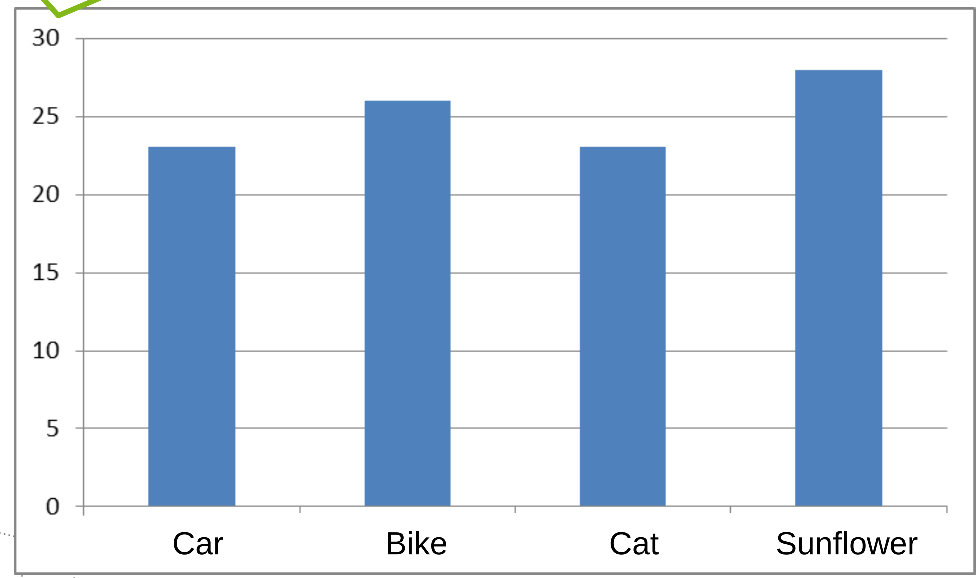
How are the weights adjusted? (contd.)

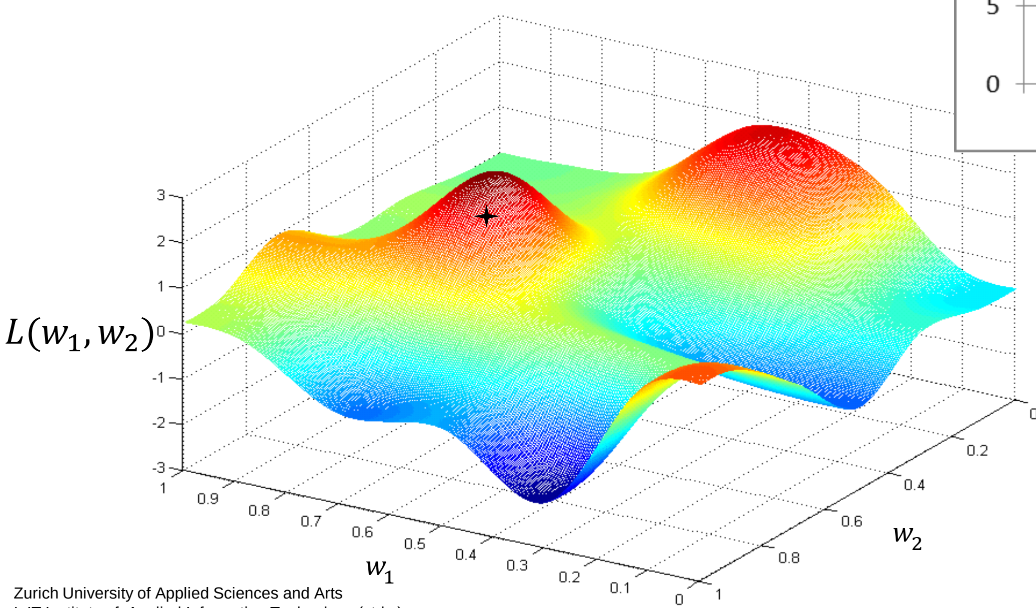
First intuition

- Our example neural network: $f_{\boldsymbol{W}}(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \ldots\}$ initialized randomly)

- Error measure: $L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N}(f_{\boldsymbol{W}}(x_i) - y_i)^2$
  Average of quadratic difference on all images (loss function)

Likelihood [%] of certain event





← error «landscape»

Zurich University of Applied Sciences

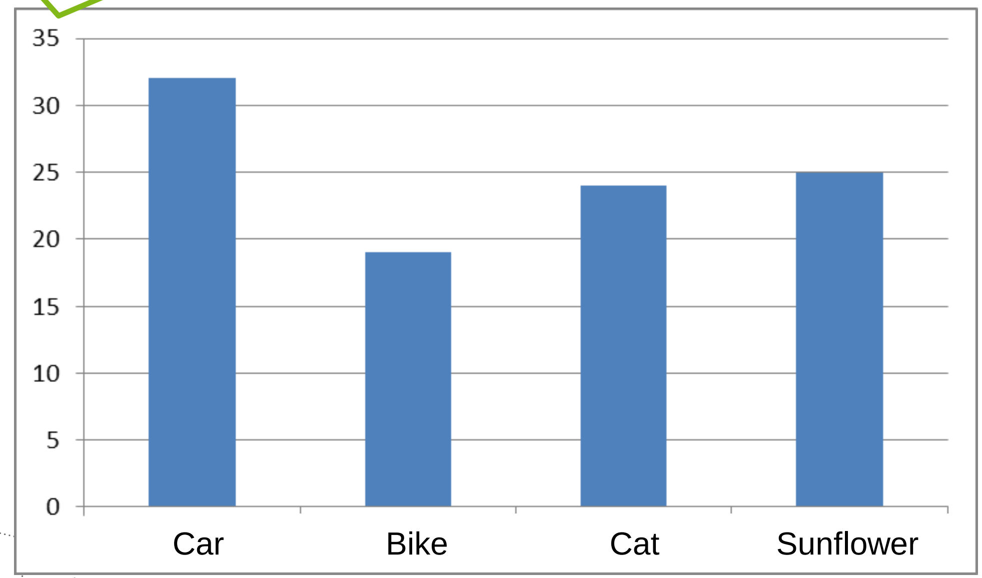
How are the weights adjusted? (contd.)

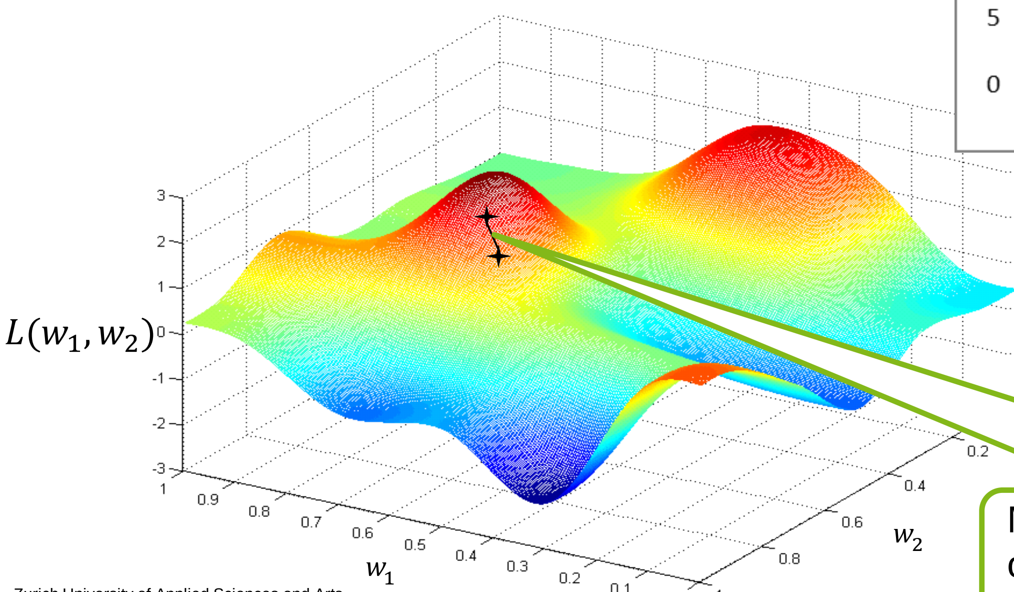
First intuition

- Our example neural network: $f_{W}(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \ldots\}$ initialized randomly)

- Error measure: $L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N}(f_{W}(x_i) - y_i)^2$
  Average of quadratic difference on all images (loss function)

Likelihood [%] of certain event





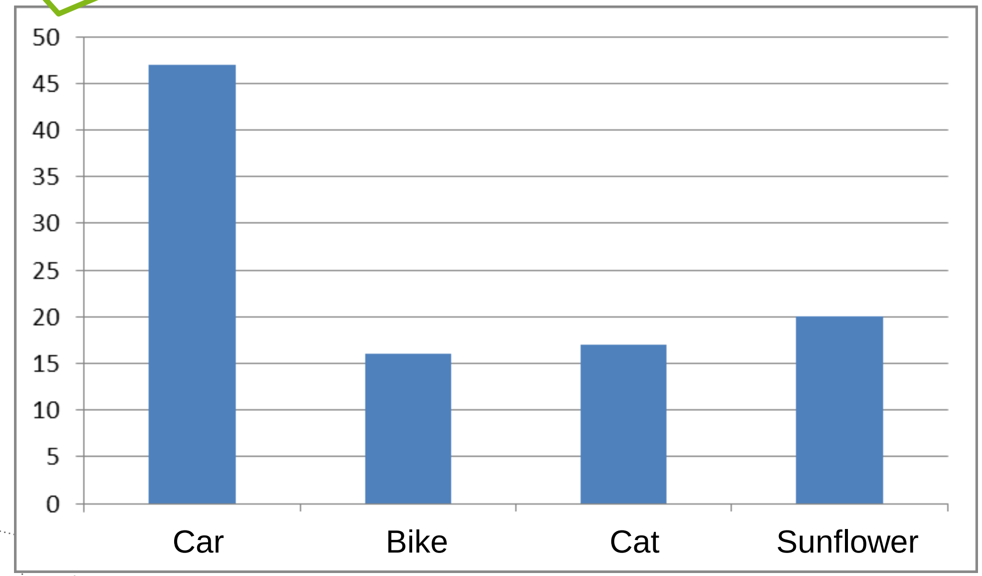
← error «landscape»

Method: adapt weights of f in the direction of the steepest descent (downwards) of L

How are the weights adjusted? (contd.)

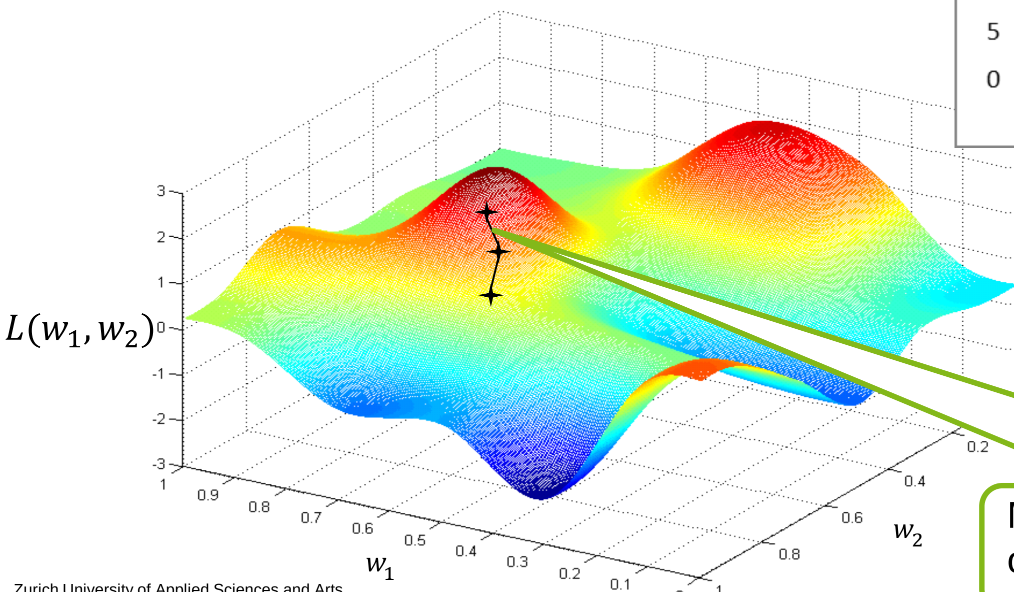
First intuition

Likelihood [%] of certain event



- Our example neural network: $f_{W}(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \ldots\}$ initialized randomly)

- Error measure: $L(W) = \frac{1}{N}\sum_{i=1}^{N}(f_{W}(x_i) - y_i)^2$
  Average of quadratic difference on all
  images (loss function)



🡐 error «landscape»

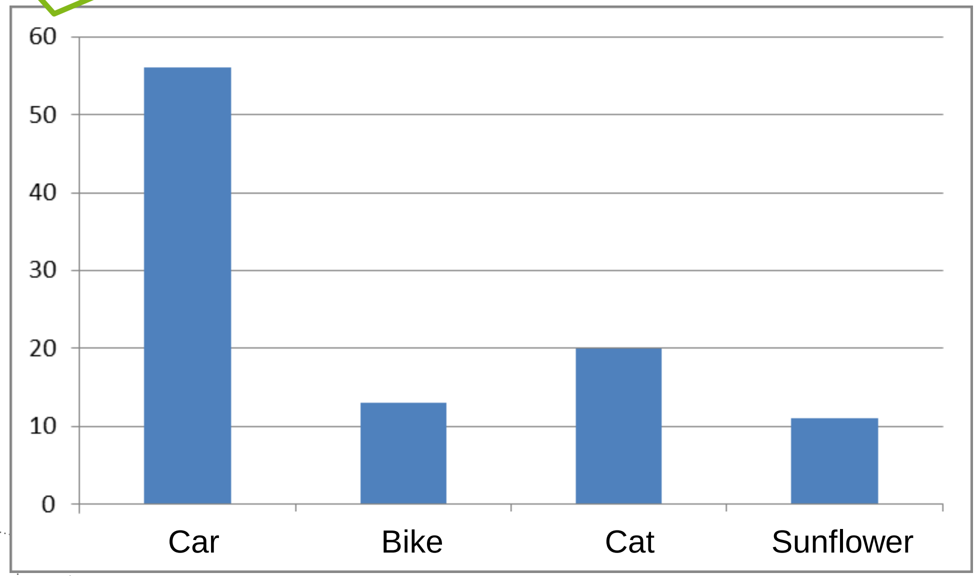
Method: adapt weights of f in the direction of the steepest descent (downwards) of L

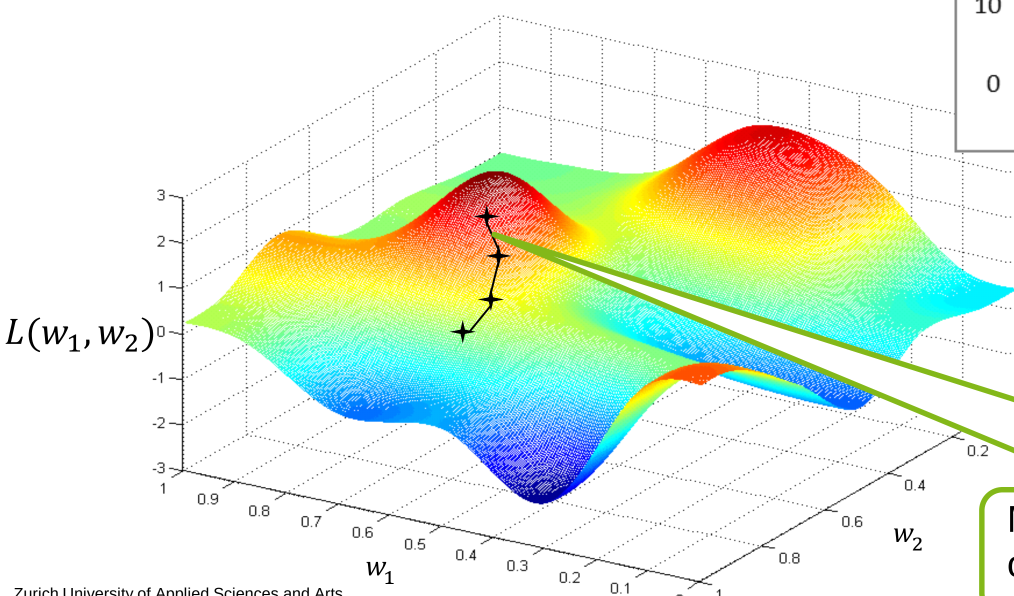
How are the weights adjusted? (contd.)
First intuition

- Our example neural network: $f_W(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \ldots\}$ initialized randomly)

- Error measure: $L(W) = \frac{1}{N}\sum_{i=1}^{N}(f_W(x_i) - y_i)^2$
  Average of quadratic difference on all images (loss function)

Likelihood [%] of certain event





← error «landscape»

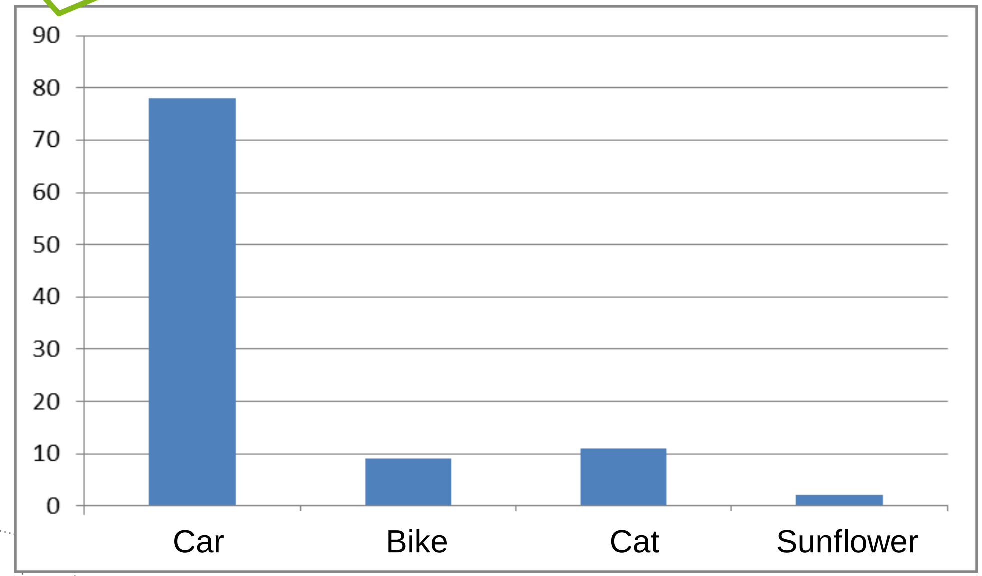
Method: adapt weights of f in the direction of the steepest descent (downwards) of L

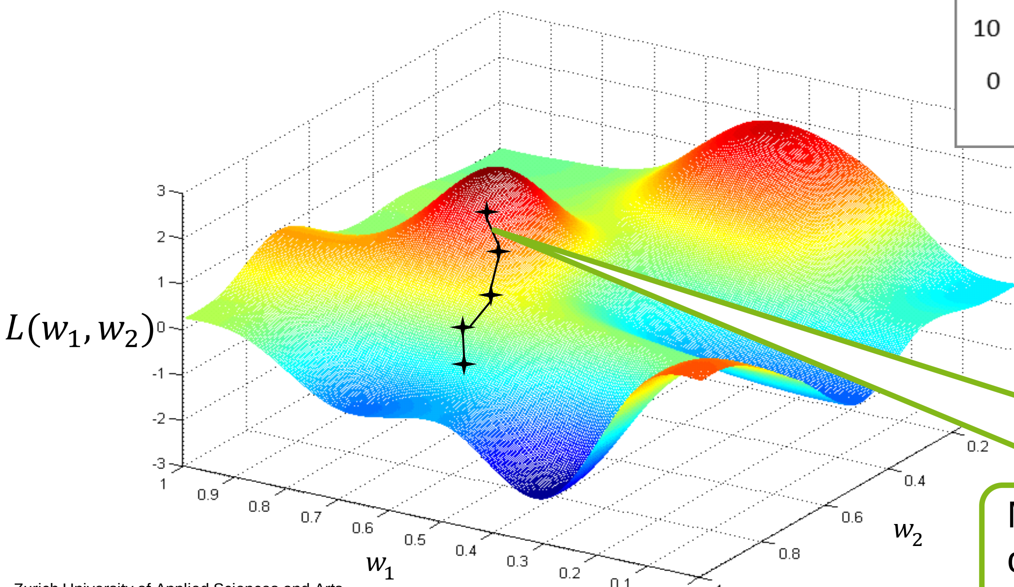
How are the weights adjusted? (contd.)
First intuition

- Our example neural network: $f_W(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \ldots\}$ initialized randomly)

- Error measure: $L(W) = \frac{1}{N}\sum_{i=1}^{N}(f_W(x_i) - y_i)^2$
  Average of quadratic difference on all
  images (loss function)

Likelihood [%] of certain event





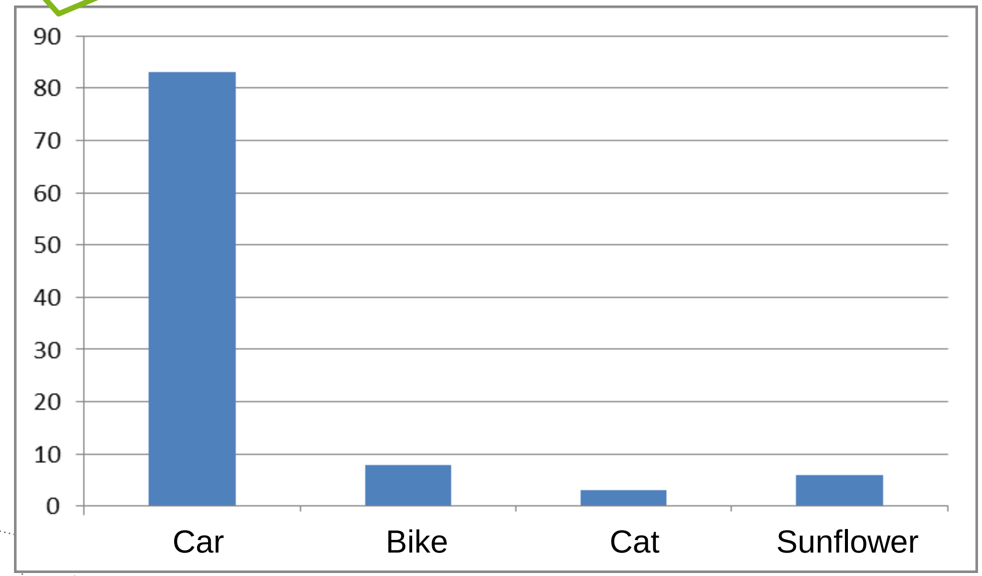
🡐 error «landscape»

Method: adapt weights of f in the direction of the steepest descent (downwards) of L

How are the weights adjusted? (contd.)

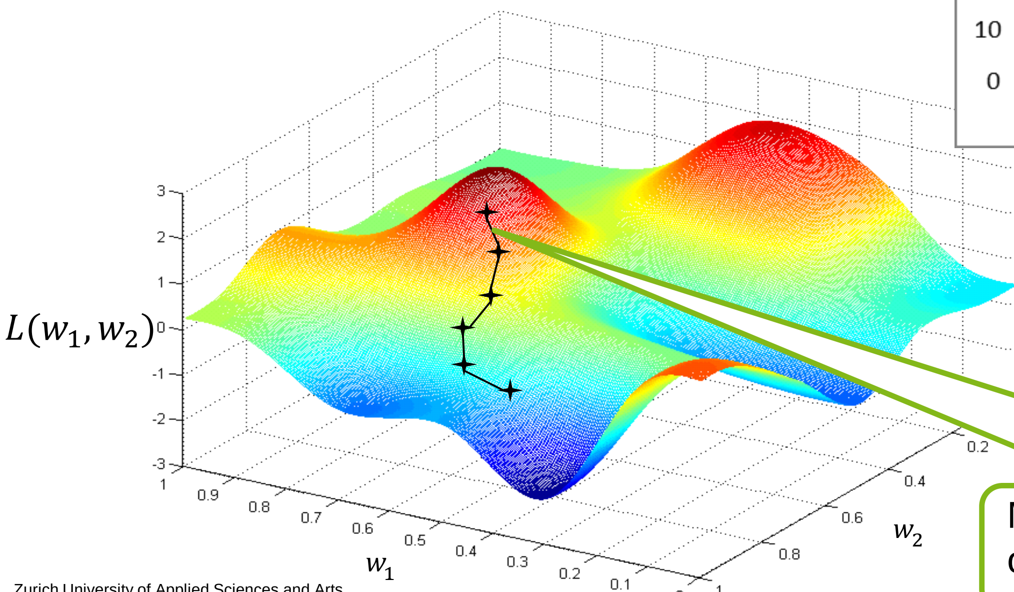
First intuition

Likelihood [%] of certain event



- Our example neural network: $f_W(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \dots\}$ initialized randomly)

- Error measure: $L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N}(f_W(x_i) - y_i)^2$
  Average of quadratic difference on all images (loss function)



🡐 error «landscape»

Method: adapt weights of f in the direction of the steepest descent (downwards) of L

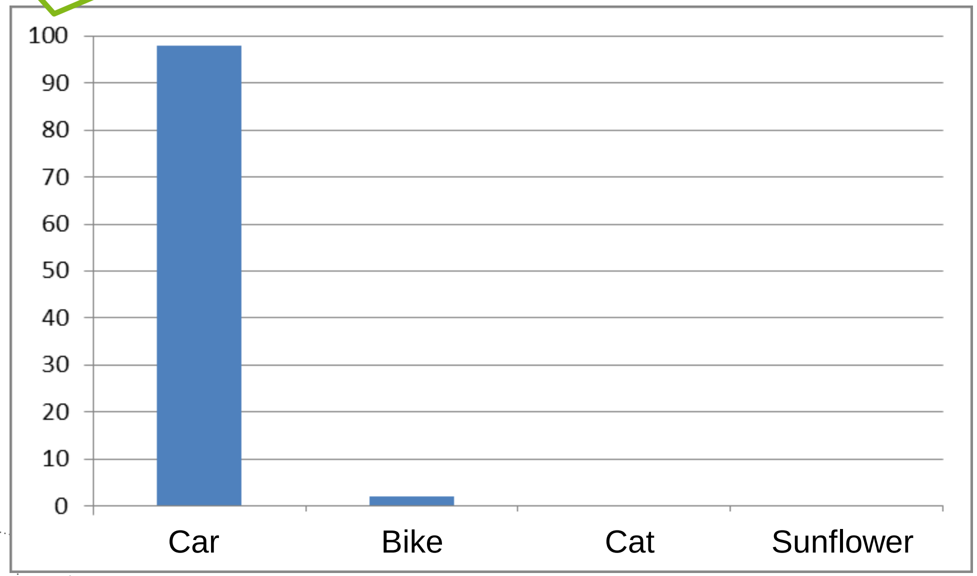
How are the weights adjusted? (contd.)

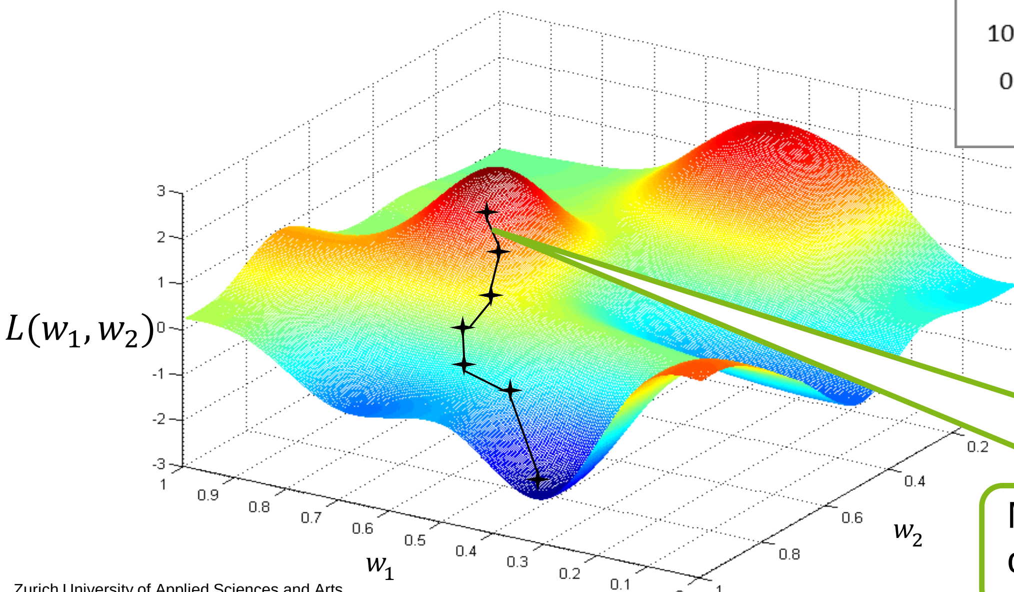
First intuition

- Our example neural network: $f_{W}(x) = y$
  with image x, ground truth y und parameters $\boldsymbol{W}$
  ($\boldsymbol{W} = \{w_1, w_2, \dots\}$ initialized randomly)

- Error measure: $L(W) = \frac{1}{N}\sum_{i=1}^{N}(f_{W}(x_i) - y_i)^2$
  Average of quadratic difference on all images (loss function)

Likelihood [%] of certain event





← error «landscape»

Method: adapt weights of f in the direction of the steepest descent (downwards) of L

How are the weights adjusted?

Neural network training ideas

→ see also https://stdm.github.io/downloads/papers/ADS_2019_DeepLearning.pdf

Trained by gradient descent (complete network is differentiable)

- Forward pass: calculation of loss function L for a mini batch of training examples
- Backward pass: calculation of $\frac{\partial L}{\partial W_{l,i}}$ for each weight $W_{l,i}$ on overall loss
  - Efficiently computable by layer-wise application of chain rule (backpropagation algorithm)

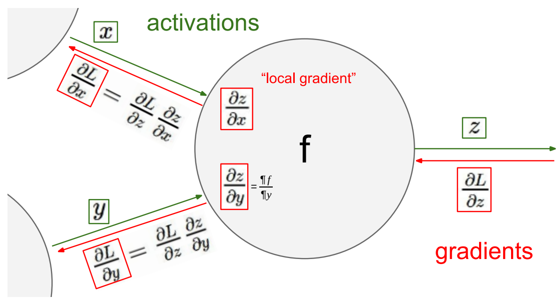
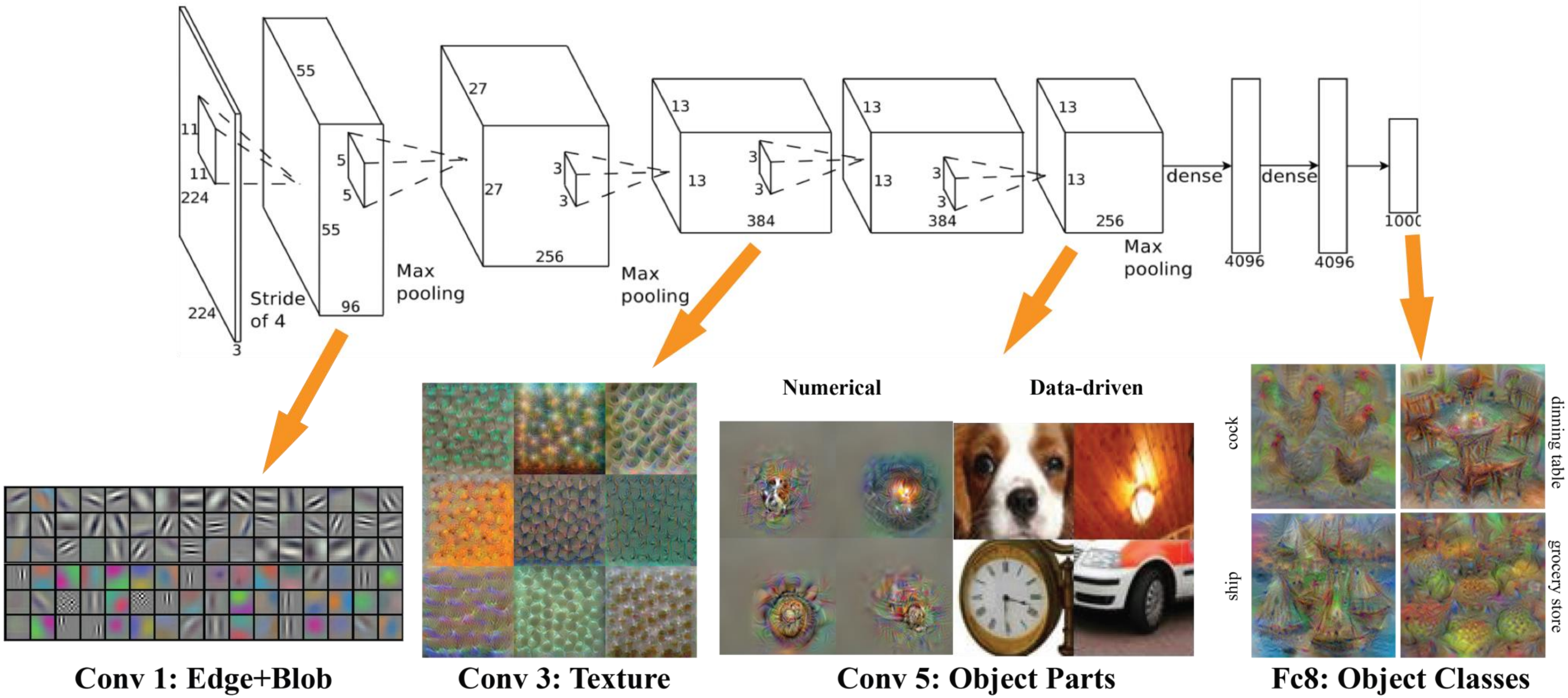


Illustration:
http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

Many details to be considered for training to work in practice

- Weight initialization: choose random initial weights according to the magnitude of the inputs
- Gradient flow: secure sufficient gradient magnitude for fast training convergence via batchnorm
- Learning rate: choose adaptive learning rates, e.g. using the ADADELTA optimizer
- Batch composition: care for sufficient randomness in the presentation order
- Regularization: use dropout to overcome the problem of more parameters then input data

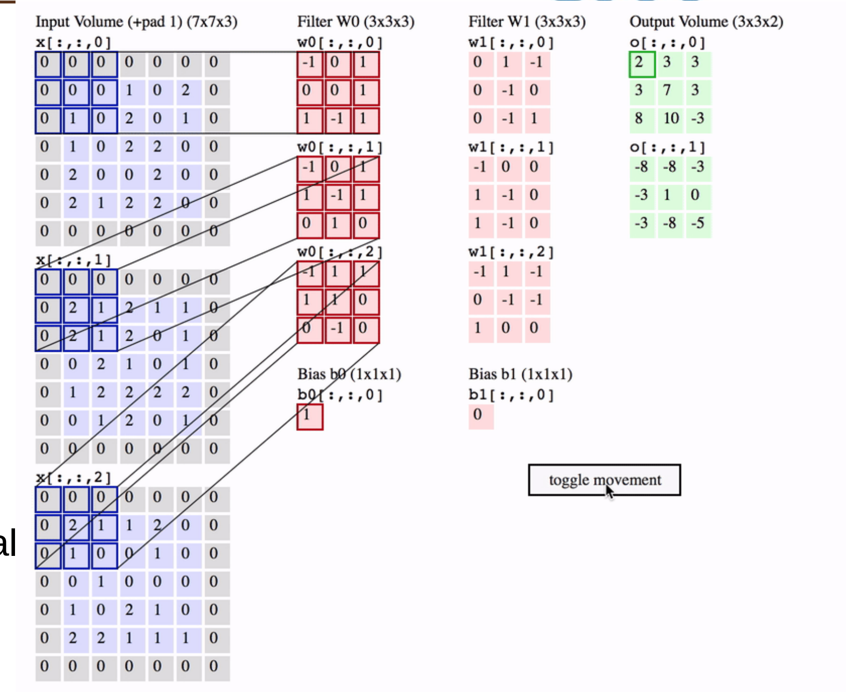
What does a neural network «see»?
A hierarchy of progressively complex features, visualized



Source: http://vision03.csail.mit.edu/cnn_art/data/single_layer.png

Convolutional Neural Networks

Intuition: cp. https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

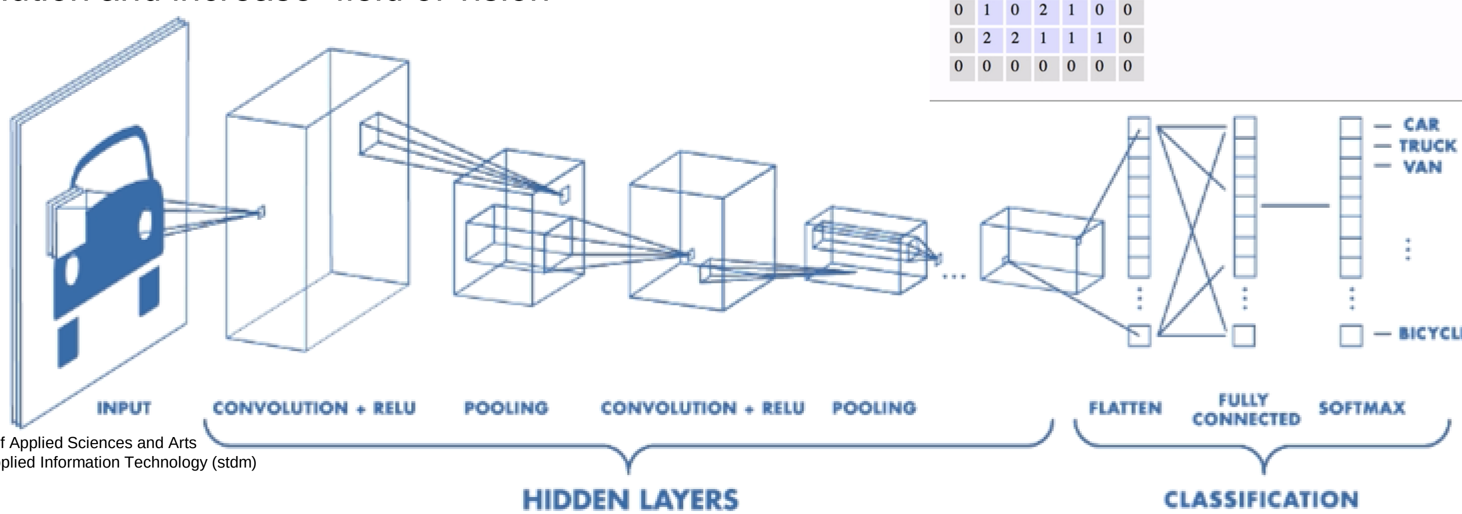


Goal: fewer free parameters → eases learning

Idea: exploit 2D-correlated local structure in (image) input data
→ inspired by mammal visual cortex

Principle

- A "**filter**" moves over every input pixel and calculates a feature that **describes** the **pixel's local context**
  → map result to same spatial location
  → filter weights (i.e., feature meaning) is trainable
- Have **several such** "filters" to encode different features
- After each filtering layer, **sub-sample** result to reduce spatial resolution and increase "field of vision"



2. GENERATIVE ADVERSARIAL NETS

Recap: Probability distributions as generative models

Terminology: its probability density function (pdf) is one way to describe a distribution.

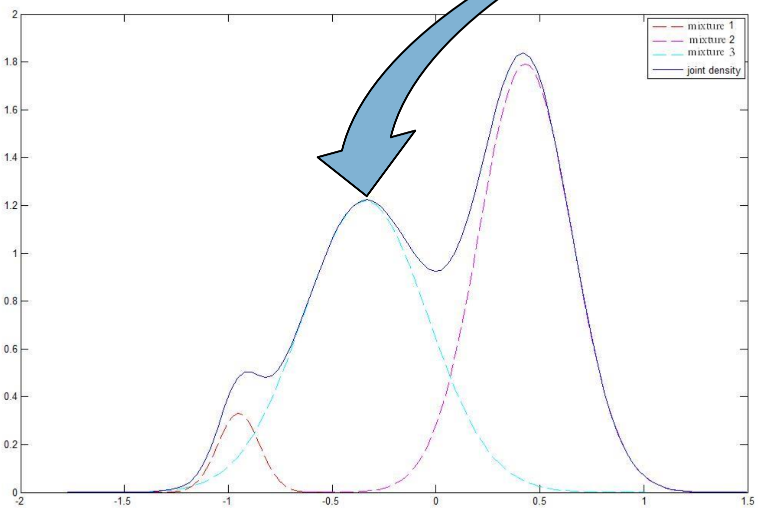
What does a pdf tell about a set of data?

➔ For data coming from some stochastic processes, the pdf tells **everything there is to know** about the data

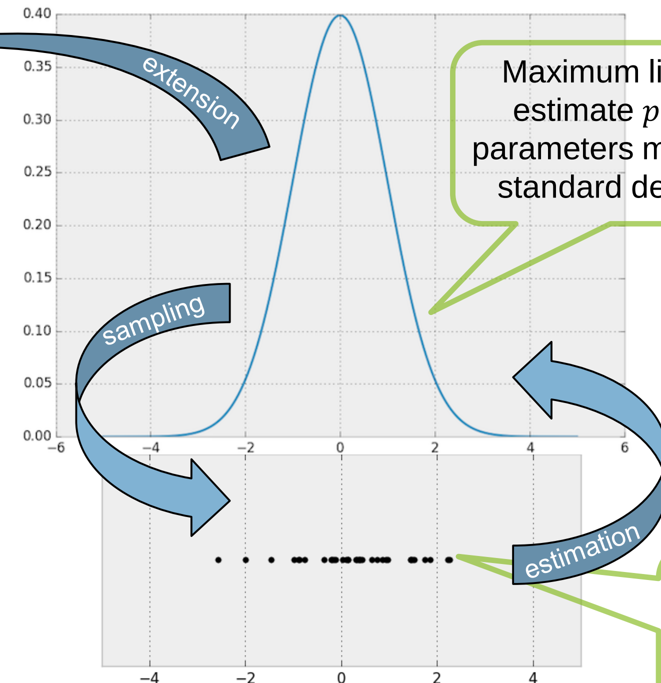
➔ **Allows for sampling** data from the underlying distribution

A Gaussian as base generative model

- Recovering a known, parametric pdf: The univariate Gaussian



Example of a multimodal (but univariate) distribution, approximated by a GMM with 3 mixtures.



Maximum likelihood estimate $p(x)$ with parameters mean μ and standard deviation σ

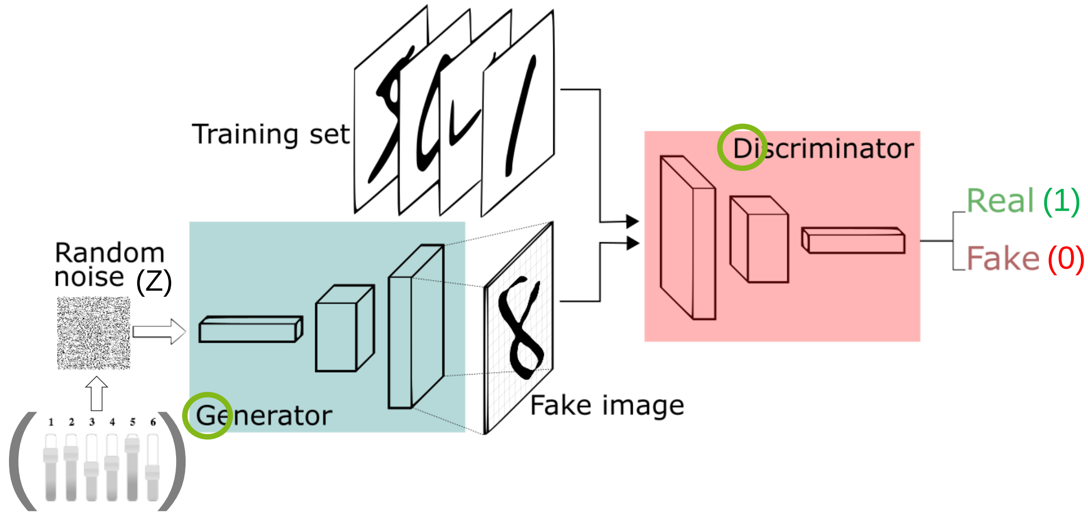
Given data points x_i; Assumption: $x_i \sim p(x;\theta) = N(x;\mu,\sigma)$

Source: Brandon Amos, «*Image Completion with Deep Learning in TensorFlow*», 2016, https://bamos.github.io/2016/08/09/deep-completion/

Adversarial nets
Bootstrapping implicit generative representations

Train 2 models simultaneously [1]

- G: Generator
  → learns to generate data
- D: Discriminator
  → learns $p(x\ not\ being\ generated)$



Sources: https://deeplearning4j.org/generative-adversarial-network; http://www.dpkingma.com/sgvb_mnist_demo/demo.html

➔ Both differentiable functions D&G learn while competing
➔ The latent space Z serves as a source of variation to generate different data points
➔ Only D has access to real data

[1] Schmidhuber, «Learning Factorial Codes by Predictability Minimization», 1992

No weenies allowed! How SpongeBob helps..

…to understand bootstrapping untrained (G)enerator & (D)iscriminator

Bouncer newbie (D) decides
on entry: for tough guys only

Source: Arthur Juliani, *«Generative Adversarial Networks Explained with a Classic Spongebob Squarepants Episode»*, 2016, https://medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39#.gcoxuaruk

No weenies allowed! How SpongeBob helps..

…to understand bootstrapping untrained (G)enerator & (D)iscriminator

Bouncer newbie (D) decides on entry: for tough guys only

SpongeBob (G) wants to appear tough to be admitted (i.e., synthesizes behavior)

Source: Arthur Juliani, «*Generative Adversarial Networks Explained with a Classic Spongebob Squarepants Episode*», 2016, https://medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39#.gcoxuaruk

No weenies allowed! How SpongeBob helps..

…to understand bootstrapping untrained (G)enerator & (D)iscriminator

Bouncer newbie (D) decides on entry: for tough guys only

SpongeBob (G) wants to appear tough to be admitted (i.e., synthesizes behavior)

In the beginning, D focuses on obvious things to discriminate: e.g., physical strength

Source: Arthur Juliani, «*Generative Adversarial Networks Explained with a Classic Spongebob Squarepants Episode*», 2016, https://medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39#.gcoxuaruk

No weenies allowed! How SpongeBob helps..

…to understand bootstrapping untrained (G)enerator & (D)iscriminator

Zurich University of Applied Sciences

Bouncer newbie (D) decides on entry: for tough guys only

SpongeBob (G) wants to appear tough to be admitted (i.e., synthesizes behavior)

In the beginning, D focuses on obvious things to discriminate: e.g., physical strength

So G tries to imitate that, but fails

Source: Arthur Juliani, «*Generative Adversarial Networks Explained with a Classic Spongebob Squarepants Episode*», 2016, https://medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39#.gcoxuaruk

No weenies allowed! How SpongeBob helps..

…to understand bootstrapping untrained (G)enerator & (D)iscriminator

Bouncer newbie (D) decides on entry: for tough guys only

SpongeBob (G) wants to appear tough to be admitted (i.e., synthesizes behavior)

In the beginning, D focuses on obvious things to discriminate: e.g., physical strength

So G tries to imitate that, but fails

By observation, G discovers more detailed features of tough guys: e.g., fighting

Source: Arthur Juliani, «*Generative Adversarial Networks Explained with a Classic Spongebob Squarepants Episode*», 2016, https://medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39#.gcoxuaruk

No weenies allowed! How SpongeBob helps..

…to understand bootstrapping untrained (G)enerator & (D)iscriminator

Bouncer newbie (D) decides on entry: for tough guys only

SpongeBob (G) wants to appear tough to be admitted (i.e., synthesizes behavior)

In the beginning, D focuses on obvious things to discriminate: e.g., physical strength

So G tries to imitate that, but fails

By observation, G discovers more detailed features of tough guys: e.g., fighting

So G learns to imitate that as well

Source: Arthur Juliani, «*Generative Adversarial Networks Explained with a Classic Spongebob Squarepants Episode*», 2016, https://medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39#.gcoxuaruk

No weenies allowed! How SpongeBob helps..
…to understand bootstrapping untrained (G)enerator & (D)iscriminator

Bouncer newbie (D) decides on entry: for tough guys only

SpongeBob (G) wants to appear tough to be admitted (i.e., synthesizes behavior)

In the beginning, D focuses on obvious things to discriminate: e.g., physical strength

So G tries to imitate that, but fails

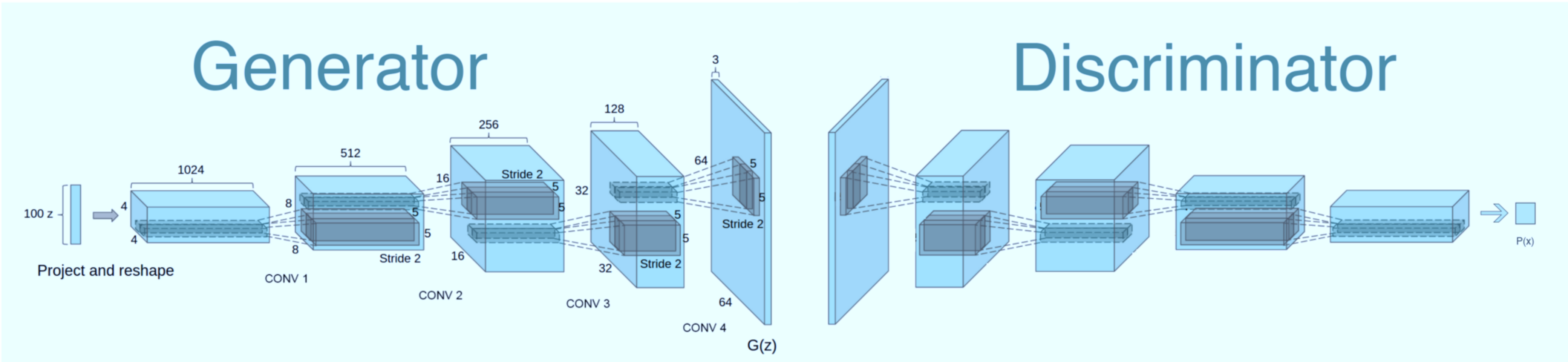
By observation, G discovers more detailed features of tough guys: e.g., fighting

So G learns to imitate that as well

…and eventually tricks D.

Source: Arthur Juliani, *«Generative Adversarial Networks Explained with a Classic Spongebob Squarepants Episode»*, 2016, https://medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39#.gcoxuaruk

GAN model formulation (improved)
Deep convolutional generative adversarial nets [2]



Implement both G and D as deep convnets (DCGAN)

- **No pooling**, only fractionally-strided convolutions (G) and strided convolutions (D)
- **No fully connected** hidden **layers** for deeper architectures
- Apply **batchnorm** in both
- **ReLU** activation **in G** (output layer: tanh)
- **LeakyReLU** activation in **D** (all layers)

[2] Radford, Metz, Chintala, «Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks», 2016

Zurich University of Applied Sciences

Model training [5]

Usually $k = 1$ (or ½)

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

[5] Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio, «**Generative Adversarial Nets**», 2014

Model training [5]

for number of training iterations **do**

Usually $k = 1$ (or ½)

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

change θ_D to **maximize**

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

[5] Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio, «**Generative Adversarial Nets**», 2014

Model training [5]

Zurich University of Applied Sciences

Usually $k = 1$ (or ½)

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the $\boxed{\text{discriminator}}$ by $\boxed{\text{ascending}}$ its stochastic gradient:

change θ_D to **maximize**

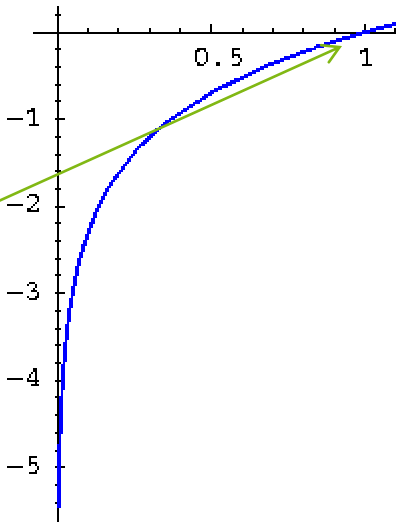
$$\nabla_{\theta_d} \underbrace{\frac{1}{m} \sum_{i=1}^{m}}_{\text{average}} \left[\underbrace{\log D\left(\boldsymbol{x}^{(i)}\right)}_{\substack{\text{log likelihood of } x \\ \text{being real} \rightarrow 0}} + \underbrace{\log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)}_{\substack{\text{log likelihood } G(z) \\ \textbf{not} \text{ being real} \rightarrow 0}} \right].$$

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for



[5] Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio, «**Generative Adversarial Nets**», 2014

Model training [5]

Usually $k = 1$ (or ½)

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

change θ_D to **maximize**

$$\nabla_{\theta_d} \underbrace{\frac{1}{m} \sum_{i=1}^{m}}_{\text{average}} \left[\underbrace{\log D\left(\boldsymbol{x}^{(i)}\right)}_{\text{log likelihood of } x \text{ being real} \to 0} + \underbrace{\log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)}_{\text{log likelihood } G(z) \text{ not being real} \to 0} \right].$$

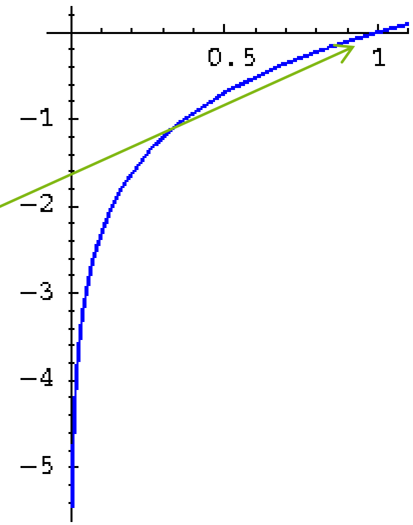
end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

change θ_G to **minimize**

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for



[5] Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio, «**Generative Adversarial Nets**», 2014

Model training [5]

Usually $k = 1$ (or ½)

for number of training iterations **do**

  for k steps **do**

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

  • Update the $\boxed{\text{discriminator}}$ by $\boxed{\text{ascending}}$ its stochastic gradient:

change θ_D to **maximize**

$$\nabla_{\theta_d} \underbrace{\frac{1}{m} \sum_{i=1}^{m}}_{\text{average}} \left[\underbrace{\log D\left(\boldsymbol{x}^{(i)}\right)}_{\text{log likelihood of } x \text{ being real} \to 0} + \underbrace{\log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)}_{\text{log likelihood } G(z) \text{ not being real} \to 0} \right].$$

  end for

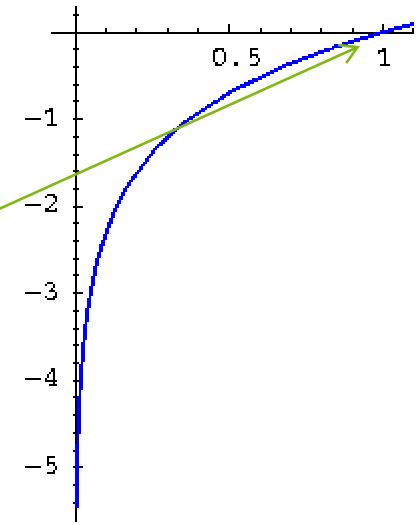
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Update the $\boxed{\text{generator}}$ by $\boxed{\text{descending}}$ its stochastic gradient:

change θ_G to **minimize**

$$\nabla_{\theta_g} \underbrace{\frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)}_{\text{G just get's gradients on how well it can fool D (no direct training labels)}}.$$

end for



[5] Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio, «**Generative Adversarial Nets**», 2014

3. USE CASE: IMAGE INPAINTING

Based on material from Brandon Amos,
«Image Completion with Deep Learning in TensorFlow», 2016

https://bamos.github.io/2016/08/09/deep-completion/

Image inpainting as a sampling problem

…approached by machine learning

Yeh et al., *«Semantic Image Inpainting with Perceptual and Contextual Losses»*, 2016

Training: Regard **images as samples of** some underlying probability distribution $\boldsymbol{p_G}$

1. Learn to represent this distribution using a GAN setup (G and D)

--

Testing: Draw a **suitable sample** from p_G by…

1. **Fixing parameters $\boldsymbol{\Theta_G}$** and Θ_D of G and D, respectively
2. **Finding** input $\boldsymbol{\hat{z}}$ to G such that $G(\hat{z})$ fits **two constraints**:
   a) **Contextual**: Output has to **match** the **known parts of** the **image** that needs inpainting
   b) **Perceptual**: Output has to **look generally «real»** according to D's judgment
3. …**by** using gradient-based **optimization on $\boldsymbol{\hat{z}}$**

Powerful idea: application of trained ML model may again involve optimization!

Reconstruction formulation

Given

- Uncomplete/corrupted image $x_{corrputed}$
- Binary mask M (same size as $x_{corrputed}$, 0 for missing/corrupted pixels)
- Generator network $G()$, discriminator network $D()$

Problem

- Find $\hat{z}$ such that $x_{reconstructed} = M \odot x_{corrputed} + (1 - M) \odot G(\hat{z})$
  ($\odot$ is the element-wise product of two matrices)

$$\underset{\text{Input}}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} \odot \underset{\text{Binary Mask}}{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}} = \underset{\text{Output}}{\begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}}$$

Solution

- Define contextual and perceptual loss as follows:
  $L_{contextual}(z) = \left\| M \odot G(z) - M \odot x_{corrupted} \right\|_1$ (distance between known parts of image and reconstruction)
  $L_{perceptual}(z) = \log\left(1 - D\left(G(z)\right)\right)$ (as before: log-likelihood of $G(z)$ **not** being real according to D)
  $L(z) = L_{contextual}(z) + \lambda \cdot L_{perceptual(z)}$ (combined loss)

➔ Optimize $\hat{z} = \arg\min_{z} L(z)$

Results

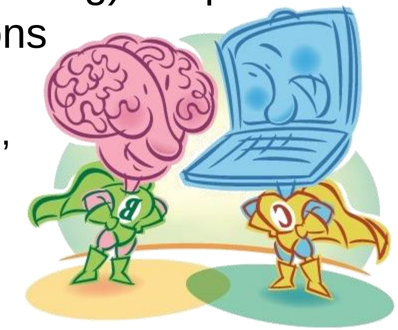
See it move: https://github.com/bamos/dcgan-completion.tensorflow

Where's the intelligence?
Man vs. machine

- **Learning** smooth approximations of complex probability **density** functions (PDF) enables us to **sample previously unseen examples**
  - That is, we can ***create*** new images, new music, …

Source: https://nerdist.com/nvidia-ai-headshots-fake-celebrities/.

- But isn't **creativity** more **the power to surprise**, i.e., (technically speaking) the power to **come up with new yet reasonable PDFs** instead of new instantiations from a given PDF?
  - That would mean that to create does not mean to know the PDF of «things», but the PDF of the «reasonableness of things». As this is unknown for novel things, it needs to be continually explored.



Review

- **Neural networks** with at least one hidden layer **are general function approximators**, trained by gradient descent

- **GANs** have been shown to **produce realistic output** on a wide range of (still smallish) image, audio and text generation tasks
- **Finding Nash equilibria** in high-dimensional, continuous, non-convex games **is an** important **open** research **problem**

- **Image inpainting works by optimizing the output** of a fully trained generator to fit the given context & realism criteria, **using** again **gradient descent**
  ➔ Applying machine learned models might involve optimization (~training) steps again
  ➔ **This is in line with human learning**: Once trained to draw, hand-copying a painting involves "optimization" on the part of the painter

Further reading: Goodfellow, *«NIPS 2016 Tutorial: Generative Adversarial Networks»*, 2016

APPENDIX

Recap: basic idea of deep learning
Add depth (layers ➔ capability) to learn features automatically

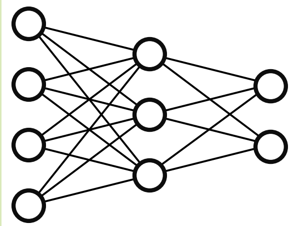
Classic computer vision

Feature extraction
(SIFT, SURF, LBP, HOG, etc.)

(0.2, 0.4, …)

(0.4, 0.3, …)

Classification
(SVM, neuronal net, etc.)



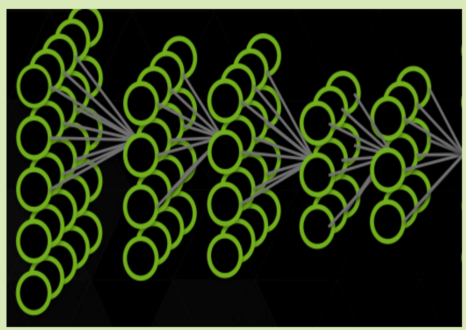
container ship

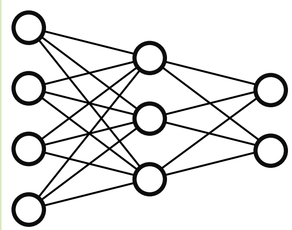
tiger

…

Convolutional neural networks (CNNs)

Takes raw pixels as input, learns good features automatically!





container ship

tiger

…

Pros and cons of generative models

Flavors of generative models

- **Statistical** models that directly model the pdf (e.g., GMM, hidden Markov model HMM)
- **Graphical** models with latent variables (e.g., Boltzmann machines RBM/DBM, deep belief networks DBN)
- **Autoencoders** (e.g. Kingma & Welling, *"Autoencoding Variational Bayes"*, 2013)

Promises

- Help **learning about** high-dimensional, complicated probability **distributions** *(even if pdf is not represented explicitly)*
- **Simulate** possible futures for planning or simulated RL
- Handle **missing data** (in particular, semi-supervised learning)
- Some applications actually require **generation** (e.g. sound synthesis, identikit pictures, content reconstruction)

Common drawbacks

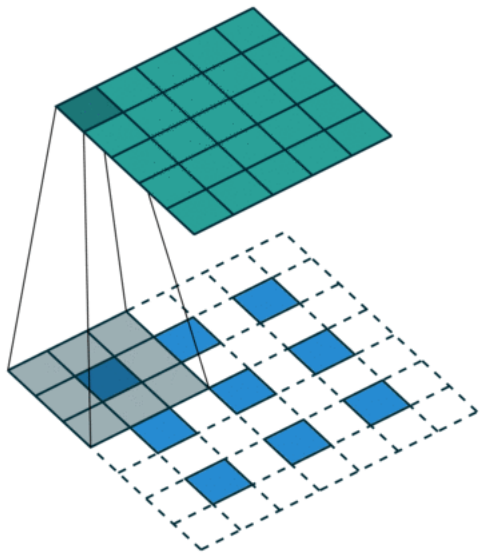
- Statistical models suffer severely from the **curse of dimensionality**
- Approximations needed for **intractable probabilistic computations** during ML estimation
- **Unbacked assumptions** (e.g., Gaussianity) and averaging e.g. in VAEs

Strided what? Convolutional arithmetic [3]

NN wiring to save weights while exploiting local structure

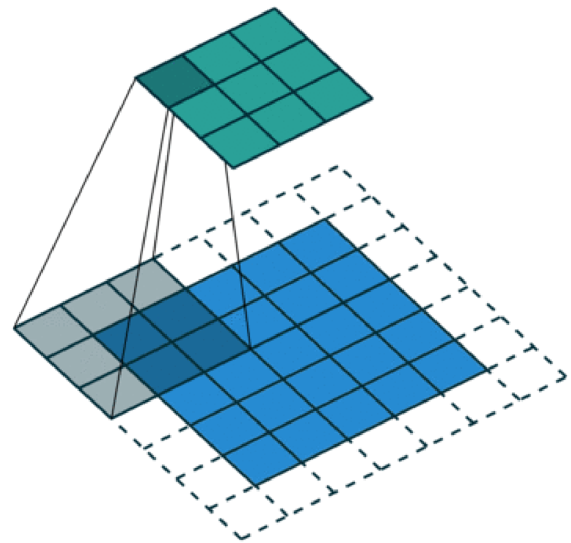
Fractionally-strided conv. in G

- Performing transposed convolution
- Used to «**up-sample**» from input (blue) to output (green)



Strided convolutions in D

- Stride (stepsize) = 2
- Used instead of (max) **pooling** [4]



[3] Dumoulin, Visin, «A guide to convolution arithmetic for deep learning », 2016
[4] Springenberg, Dosovitsiy, Brox, Riedmiller, «Striving for simplicity: The all convolutional net», 2014

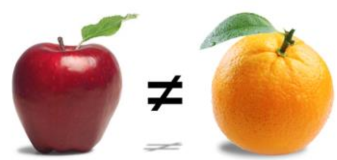
Visualizing the training process

Observations

- G starts with producing **random noise**
- Quickly arrives at what seems to be **pencil strokes**
- It takes a while for the network to produce **different images** for different z
- It takes nearly to the end before the synthesized **images per z stabilize** at certain digits

6x6 samples $G(z)$ from fixed z's every 2 mini batches (for 50k iterations). See https://dublin.zhaw.ch/~stdm/?p=400.

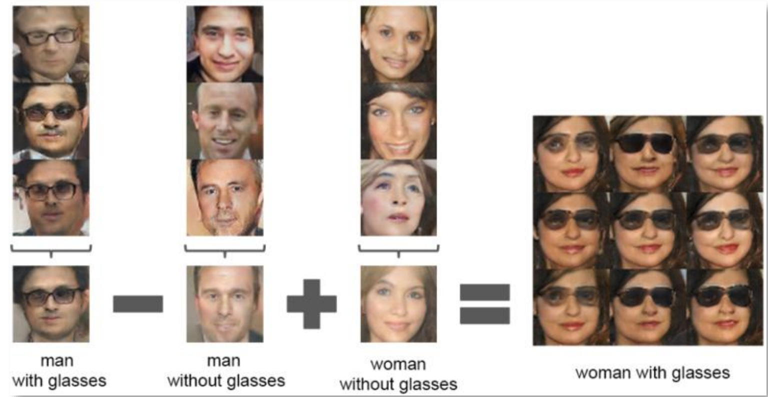
➔ Possible improvements?



Features of (DC)GANs

Learn semantically meaningful latent space

- Examples of z**-space vector arithmetic** from DCGAN paper [2]:



The z vectors in the left 3 columns have been averaged, then arithmetic has been performed. The middle image on the right is the output of $G(resulting\ z\ vector)$. The other 8 pictures are the result of adding noise to the resulting z vector (showing that smooth transitions in input space result in smooth transitions in output space).

Training is not guaranteed to converge

- D and G play a **game-theoretic game** against each other (in terms of slide 12: minimax)
- **Gradient descent isn't meant to find** the corresponding Nash Equilibria (saddle point of joint loss function, corresponding to minima of both player's costs) [6]
- How to **sync D's and G's training** is experimental (if G is trained too much, it may collapse all of z's variety to a single convincing output)
- The improvements of [2] and [7] make them **stable enough for first** practical **applications**
- **Research** on adversarial training of neural networks is still **in its infancy**

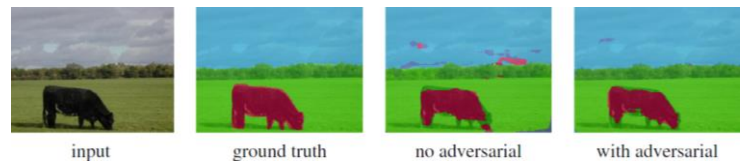
[6] Goodfellow, Courville, Bengio, «Deep Learning», ch. 20.10.4, 2016
[7] Salimans, Goodfellow, Zaremba, Cheung, «Improved Techniques for Training GANs», 2016

GAN use cases

Research is **gaining momentum very quickly**; see appendix for more!

- Generate images from text
  Reed et al., *«Generative Adversarial Text to Image Synthesis»*, 2016

- Segment images into semantically meaningful parts
  Luc et al., *«Semantic Segmentation using Adversarial Networks»*, 2016



- Complete missing parts in images
  Yeh et al., *«Semantic Image Inpainting with Perceptual and Contextual Losses»*, 2016
  ➔ see next slides

The GAN zoo as of April 2017

Avinash Hindupur's list at https://github.com/hindupuravinash

GAN - Generative Adversarial Networks
3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
AdaGAN - AdaGAN: Boosting Generative Models
AffGAN - Amortised MAP Inference for Image Super-resolution
AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
ALI - Adversarially Learned Inference
AMGAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
Bayesian GAN - Deep and Hierarchical Implicit Models
BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
BiGAN - Adversarial Feature Learning
BS-GAN - Boundary-Seeking Generative Adversarial Networks
CGAN - Conditional Generative Adversarial Nets
CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
CoGAN - Coupled Generative Adversarial Networks
Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
DTN—Unsupervised Cross-Domain Image Generation
DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
EBGAN - Energy-based Generative Adversarial Network
f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
FF-GAN - Towards Large-Pose Face Frontalization in the Wild
GAWWN - Learning What and Where to Draw
GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
IAN - Neural Photo Editing with Introspective Adversarial Networks
iGAN - Generative Visual Manipulation on the Natural Image Manifold
IcGAN - Invertible Conditional GANs for image editing
ID-CGAN- Image De-raining Using a Conditional Generative Adversarial Network
Improved GAN - Improved Techniques for Training GANs
InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks
LR-GAN - LR-GAN: Layered Recursive Generative Adversarial Networks for Image Generation

LSGAN - Least Squares Generative Adversarial Networks
LS-GAN - Loss-Sensitive Generative Adversarial Networks on Lipschitz Densities
MGAN - Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks
MAGAN - MAGAN: Margin Adaptation for Generative Adversarial Networks
MAD-GAN - Multi-Agent Diverse Generative Adversarial Networks
MalGAN - Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN
MARTA-GAN - Deep Unsupervised Representation Learning for Remote Sensing Images
McGAN - McGan: Mean and Covariance Feature Matching GAN
MedGAN - Generating Multi-label Discrete Electronic Health Records using Generative Adversarial Networks
MIX+GAN - Generalization and Equilibrium in Generative Adversarial Nets (GANs)
MPM-GAN - Message Passing Multi-Agent GANs
MV-BiGAN - Multi-view Generative Adversarial Networks
pix2pix - Image-to-Image Translation with Conditional Adversarial Networks
PPGN - Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space
PrGAN - 3D Shape Induction from 2D Views of Multiple Objects
RenderGAN - RenderGAN: Generating Realistic Labeled Data
RTT-GAN - Recurrent Topic-Transition GAN for Visual Paragraph Generation
SGAN - Stacked Generative Adversarial Networks
SGAN - Texture Synthesis with Spatial Generative Adversarial Networks
SAD-GAN - SAD-GAN: Synthetic Autonomous Driving using Generative Adversarial Networks
SalGAN - SalGAN: Visual Saliency Prediction with Generative Adversarial Networks
SEGAN - SEGAN: Speech Enhancement Generative Adversarial Network
SeGAN - SeGAN: Segmenting and Generating the Invisible
SeqGAN - SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient
SketchGAN - Adversarial Training For Sketch Retrieval
SL-GAN - Semi-Latent GAN: Learning to generate and modify facial images from attributes
Softmax-GAN - Softmax GAN
SRGAN - Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network
S^2GAN - Generative Image Modeling using Style and Structure Adversarial Networks
SSL-GAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
StackGAN - StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks
TGAN - Temporal Generative Adversarial Nets
TAC-GAN - TAC-GAN - Text Conditioned Auxiliary Classifier Generative Adversarial Network
TP-GAN - Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis
Triple-GAN - Triple Generative Adversarial Nets
Unrolled GAN - Unrolled Generative Adversarial Networks
VGAN - Generating Videos with Scene Dynamics
VGAN - Generative Adversarial Networks as Variational Training of Energy Based Models
VAE-GAN - Autoencoding beyond pixels using a learned similarity metric
VariGAN - Multi-View Image Generation from a Single-View
ViGAN - Image Generation and Editing with Variational Info Generative Adversarial Networks
WGAN - Wasserstein GAN
WGAN-GP - Improved Training of Wasserstein GANs
WaterGAN - WaterGAN: Unsupervised Generative Network to Enable Real-time Color Correction of Monocular Underwater Images