Applied Deep Learning
Foundation

Mark Schutera

Karlsruhe Institute of Technology | ZF Friedrichshafen AG | Hochschule Ravensburg-Weingarten

Course overview

Approaching **deep learning** concepts from a **practitioners** perspective.

Providing a **theoretical background**.

Course overview

Approaching deep learning concepts from a practitioners perspective.

Providing a theoretical background.

1. Deep Learning Foundations

3. Transfer Learning and Object Detection

5. Segmentation Networks

8. Deep Reinforcement Learning

10. Generative Adversarial Networks

12. Recurrent Neural Networks

Course overview

Approaching deep learning concepts from a practitioners perspective.

Providing a **theoretical background**

supported by **guided exercises** and **tutorials**.

Course overview

Approaching deep learning concepts from a practitioners perspective.

Providing a theoretical background supported by guided exercises and tutorials.

2. Tensorflow and Neural Networks

4. Object Detection Network

6. Image Segmentation

9. Artificial Intelligence Gym

11. Image Generation

13. Natural Language Processing

Course overview

Approaching deep learning concepts from a practitioners perspective.

Providing a theoretical background supported by guided exercises and tutorials.

Rounded off with a **practical course project** on- and off-class.

Course overview

Approaching deep learning concepts from a practitioners perspective.

Providing a theoretical background supported by guided exercises and tutorials.

Rounded off with practical course work.

Project proposal

One page on introduction, methods, dataset

Deadline 3. Lecture

Intermediate presentation

Ten minutes on achievements, problems, next steps

Due 7. Lecture

Final presentation

Code and results

Due 14. Lecture

Final documentation

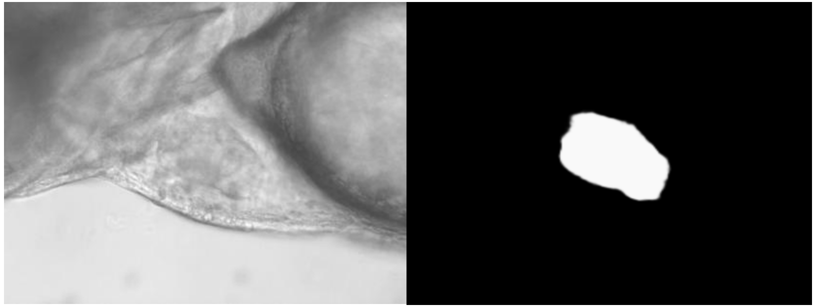
Paper and code on github or jupyter notebook

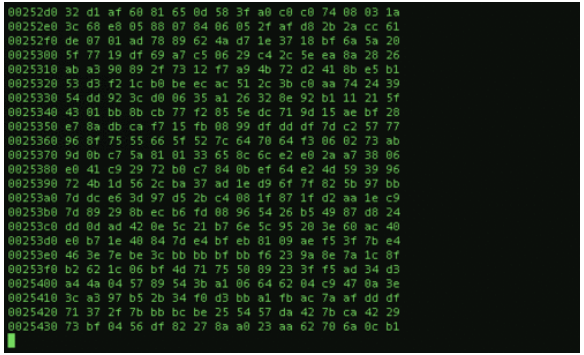
Deadline 14. Lecture

Project inspiration needed?

Deep Learning Application

Segmentation, Classification, etc. on medical data, geo data, autonomous driving data





https://pixabay.com/

Deep Learning Dataset

Define a new and interesting problem statement where annotated data lacks. Dive into data collection and data annotation strategies

Deep Learning Leaderboards

Compete on leaderboards such as Kaggle, Starcraft gym, AI gym, conference competition tracks and other benchmarks

https://github.com/deepmind/pysc2

Course features

Sli.do

Every question matters.

Get the app.

Ask questions (with slide number) or vote on other students' questions during the lecture.

And give direct feedback.

#TOBEDETERMINED

Questions will be covered immediately or in the next lecture in more depth.

Github

Find slides, tutorials, flashcards and references on Github.

https://github.com/schutera/DeepLearningLecture_Schutera

You found typos, additional material such as links, algorithms, papers, literature or want to contribute to the slides and lecture notes..

..Feel free to contribute, e-mail me.

Course features

Sli.do

Every question matters.

Get the app.

Ask questions (with slide number) or vote on other students' questions during the lecture.

#TOBEDETERMINED

Questions will be covered directly or in the next lecture in more depth.

Github

Typos, additional material such as links, algorithms, paper, literature, lecture notes..

..Feel free to contribute.

Grade Bonus .3

Prepare flashcards based on Ian Goodfellow's Deep Learning Book

- Commit to flashcard set by emailing me, first come first serve
- Must be comprehensive

This lecture in one slide

Introduction and motivation for deep learning

History

General concepts of data science

TensorFlow

Neural network conception

Optimization

Regularization

Deep learning is **representation learning** or **feature learning**.

Neural networks turn complex **information** into compact **knowledge**.

Artificial Intelligence

Machine Learning

Representation Learning

Deep Learning

Historical development

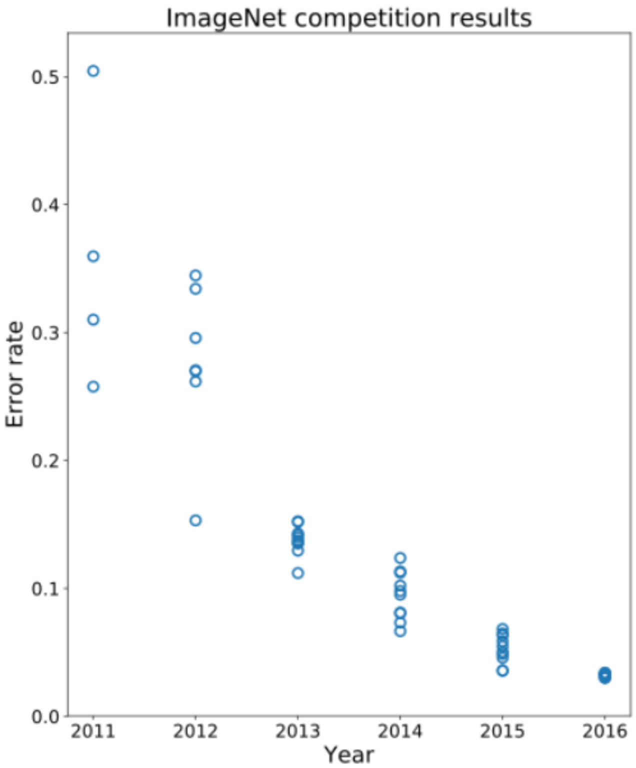
1943	Neural Networks
1957	Perceptron
1974	Backpropagation
1986	Recurrent Neural Networks
2006	“Deep Learning”
2007	CUDA
2009	ImageNet
2014	Generative Adversarial Networks
2015	Tensorflow 0.1
2016	AlphaGo
2017	Pytorch 0.1
2019	AlphaStar

Deep learning – A machine learning revolution?

Deep Learning does not come from the void.

ImageNet competition as example for Image Classification

14+ million images within ~22k categories



https://en.wikipedia.org/wiki/File:ImageNet_error_rate_history_(just_systems).svg

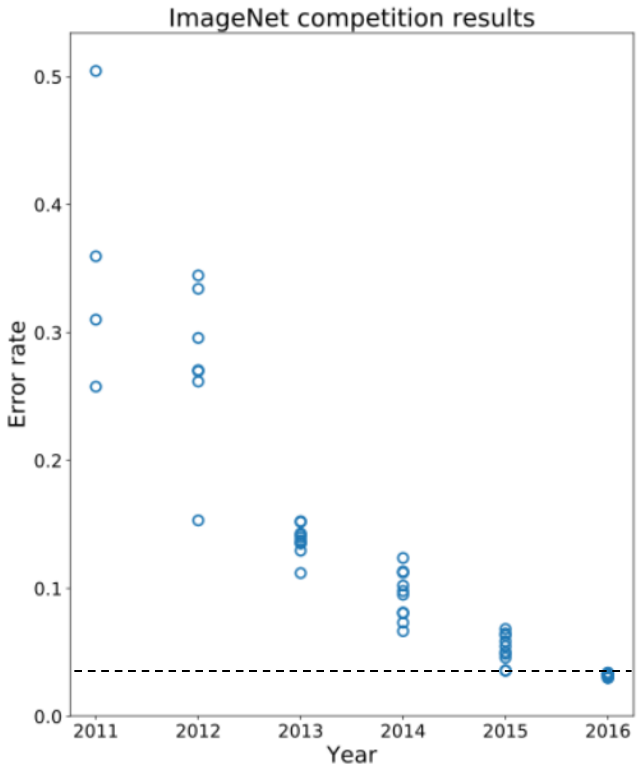
Deep learning – A machine learning revolution?

Deep Learning does not come from the void.

ImageNet competition as example for Image Classification

14+ million images within ~22k categories

ResNet (2015) reaches an error of **3.57%** surpassing all other conventional approaches.



https://en.wikipedia.org/wiki/File:ImageNet_error_rate_history_(just_systems).svg

Deep learning – A machine learning revolution?

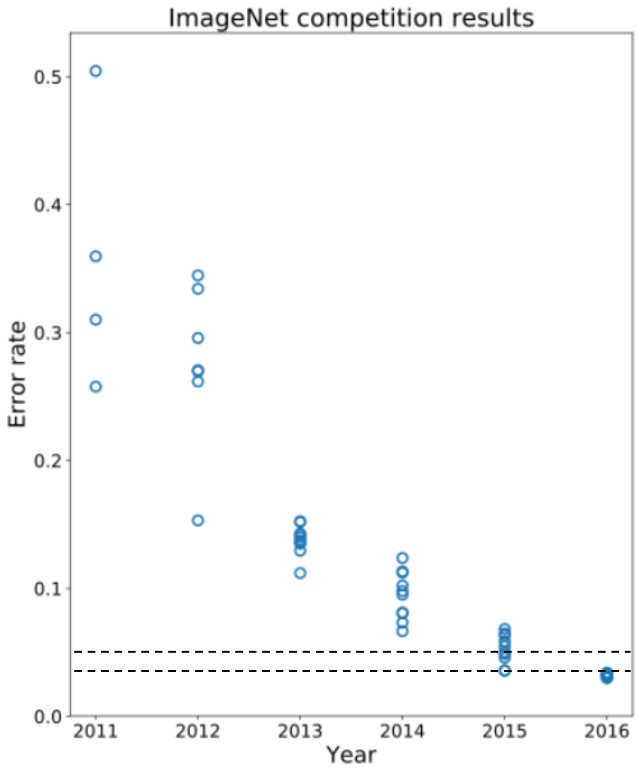
Deep Learning does not come from the void.

ImageNet competition as example for Image Classification

14+ million images within ~22k categories

ResNet (2015) reaches an error of 3.57% surpassing all other conventional approaches.

Surpassing (Karpathy) **Human Error** of **5.1%**



https://en.wikipedia.org/wiki/File:ImageNet_error_rate_history_(just_systems).svg

Why now? – Deep Learning is SIMD driven.

High Performance Parallel Computing

CPUs

GPUs

ASICs

FPGAs

ZF Pro AI

Available large Datasets

Software and Infrastructure

Backing by large companies

Leaps in Research

Why now? – Deep Learning is data driven.

High Performance Parallel Computing

Available large Datasets

ImageNet

COCO

KITTI

CityScapes

The KITTI Vision Benchmark Suite

A project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago

http://www.cvlibs.net/datasets/kitti/

COCO

Common Objects in Context

http://cocodataset.org/#home

Software and Infrastructure

Backing by large companies

Leaps in Research

Why now? – Deep Learning is software driven.

High Performance Parallel Computing

Available large Datasets

Software and Infrastructure

TensorFlow

Caffe

Pytorch

ROS

Git

https://www.tensorflow.org/

Backing by large companies

Leaps in Research

Why now? – Deep Learning is investment driven.

High Performance Parallel Computing

Available large Datasets

Software and Infrastructure

Backing by large companies and industries

Google
Facebook
Amazon
Uber
ZF

NVIDIA
Daimler
VW
Bosch

Leaps in Research

https://www.nvidia.com/en-us/

https://www.zf.com

Why now? – Deep Learning is investment driven.

High Performance Parallel Computing

Available large Datasets

Software and Infrastructure

Backing by large companies

Leaps in Research

Backpropagation

CNN

LSTM

GAN

General concept of data science

Classic machine learning

Input

Data acquisition

Data annotation

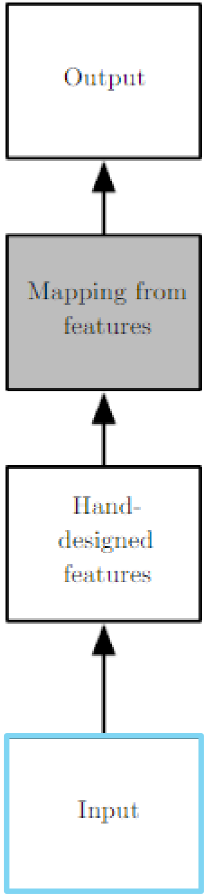
Data preprocessing

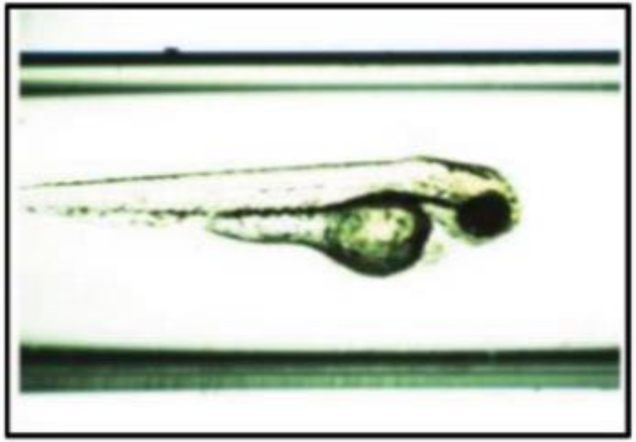
Data augmentation

Hand-designed features

Mapping from features

Output





General concept of data science

Classic machine learning

Input

Hand-designed features

Domain knowledge

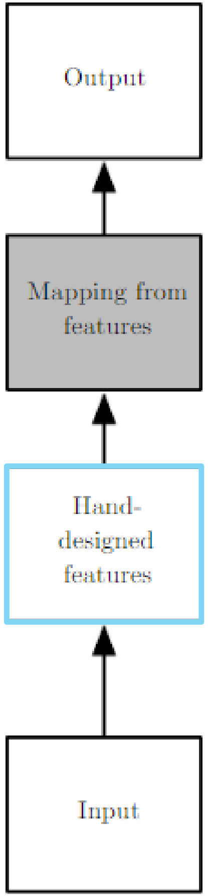
Trial and error

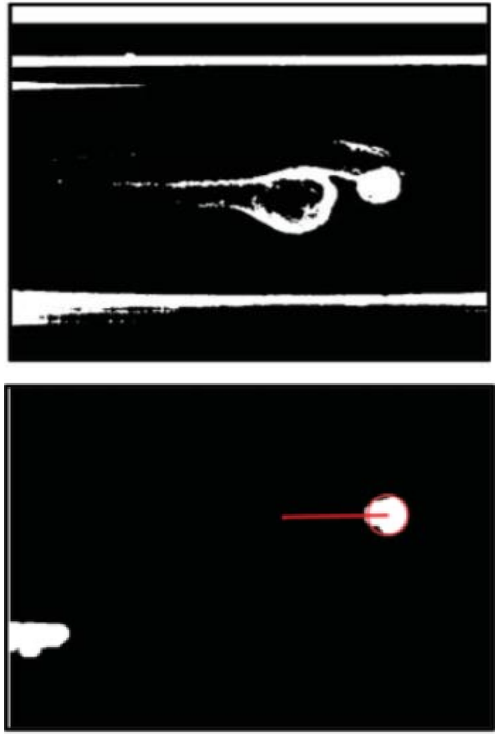
Feature selection methods

Feature set selection methods

Mapping from features

Output





General concept of data science

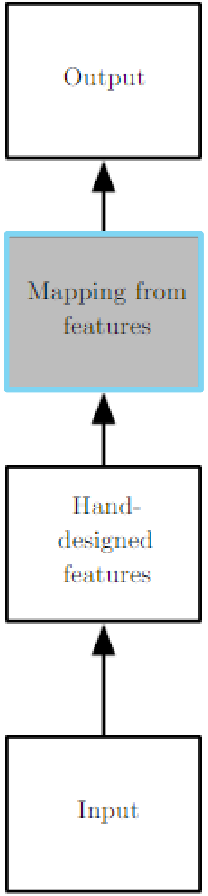
Classic machine learning

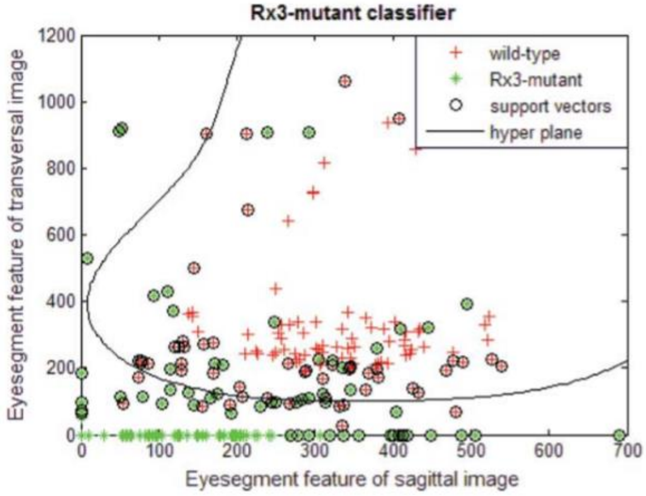
Input
Hand-designed features

Mapping from features
Learn pattern from data..
..Support vector machine
..Random forest

..Unsupervised – Class label is not known
..Supervised – Class label is known

Output



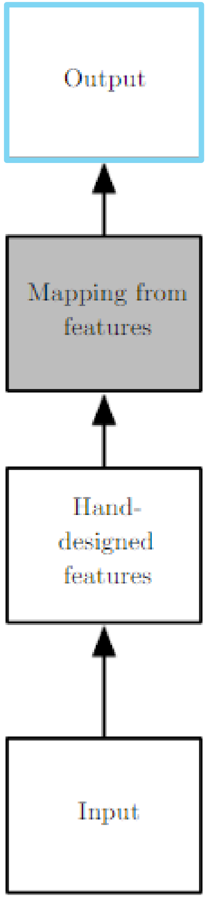


General concept of data science

Classic machine learning

Input
Hand-designed features
Mapping from features

Output
Inference step on data not seen during training



0 wildtype

1 mutant

Evaluation and Dataset split

Learning pattern through optimization

Parameters of a model are iteratively updated with respect to loss J of the training set

Hyperparameter configuration

Regularization

training set

validation set

test set

Evaluation and Dataset split

Learning pattern through optimization

Hyperparameter configuration

Model is tested on validation data in order to be able to configure hyperparameters and see the generalization performance of the model

Regularization

training set

validation set

test set

Evaluation and Dataset split

Learning pattern through optimization
Hyperparameter configuration

Verifying generalization

Through adjusting the hyperparameters based on the validation loss, this data is learned implicitly. To verify your model performance it is finally run on the test set

training set

validation set

test set

Dataset split

Less training data
.. higher variance in parameter estimates

Less validation and test data
..higher variance in performance estimate

Highly dependent on dataset and task

training set
validation set
test set

Dataset split

Less training data
.. higher variance in parameter estimates

Less validation and test data
..higher variance in performance estimate

Cross validate if possible.

80% — 80% training set, 20% validation set

20% test set

Random sampling

When splitting your dataset, do random sampling to break collection biases.

E.g. Time dependencies, sensor dependencies

Dataset with high variance

Sampling order

Random order

https://pixabay.com/

Evaluation and Dataset split

Random sampling

Dataset with high variance

Your model can only depict the data you collected.

A high data variance enables generalization and makes your model less prone to data biases.

https://pixabay.com/

tank

no tank

https://pixabay.com/

Evaluation and Dataset split

Random sampling

Dataset with high variance

Your model can only depict the data you collected.

A high data variance enables generalization and makes your model less prone to data biases.

https://pixabay.com/

tank
sunny

overcast
no tank

https://pixabay.com/

TensorFlow

TensorFlow is an open source, Python, **deep learning library** from google.

https://www.tensorflow.org/

pip install tensorflow

https://www.tensorflow.org/

TensorFlow

TensorFlow is an open source, Python, deep learning library from google. https://www.tensorflow.org/

Keras	High level API
TensorFlow Lite	Embedded systems
Colaboratory	Free GPUs in the cloud
TPU	Optimized tensor processing units
TensorBoard	Visualization
TensorFlow	Hub Ready to use graph modules
TensorRT	Optimization modules

https://www.tensorflow.org/

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[3] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition. `http://cs231n.github.io/neural-networks-3/`, 2018. Zugriff: 20.01.2018.

[4] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[5] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] Mark Schutera, Thomas Dickmeis, Marina Mione, Ravindra Peravali, Daniel Marcato, Markus Reischl, Ralf Mikut, and Christian Pylatiuk. Automated phenotype pattern recognition of zebrafish for high-throughput screening. *Bioengineered*, 7(4):261–265, 2016.

This lecture in one slide

Introduction and motivation for deep learning

Neural network conception

Architecture

Back propagation

Activation functions

Objective functions

Metrics

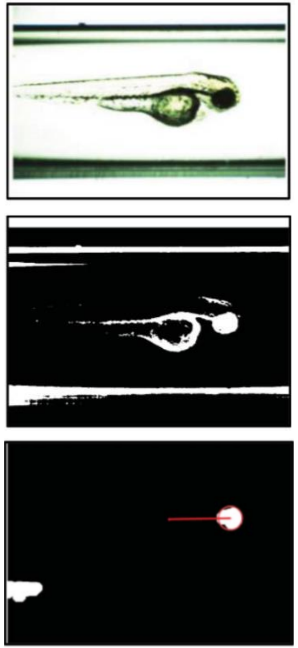
Optimization

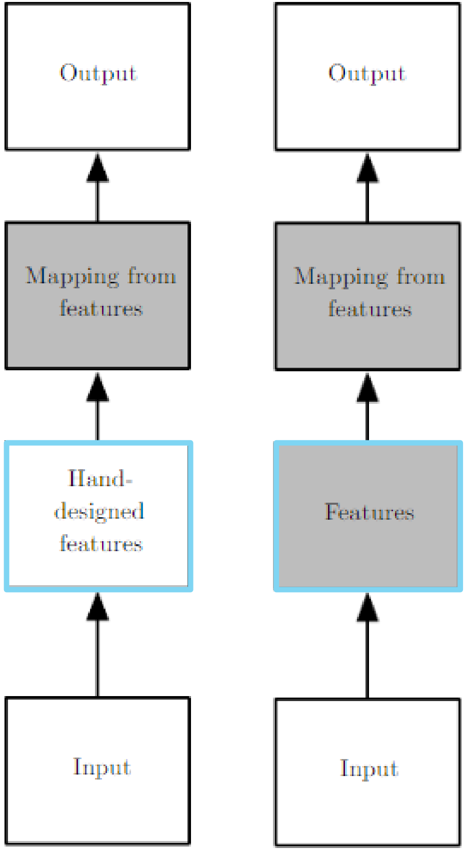
Regularization

Classic machine learning to deep learning

Classic machine learning

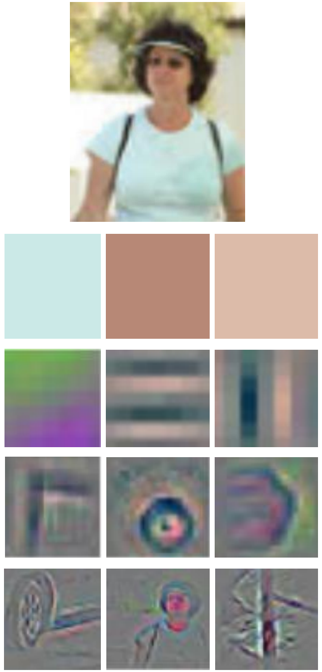
Hand-designed features





Representation learning

Data-driven feature selection



Neural Networks Introduction

Neural networks are **biologically inspired**

Neuron
Learning
Activation

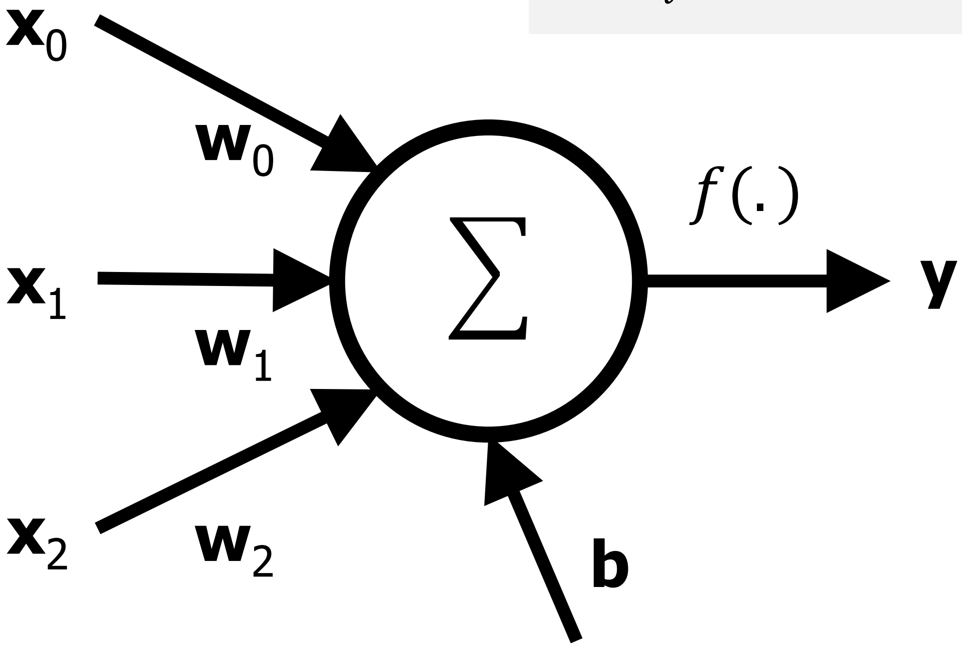
But this is only a **coarse analogy**
Synapses are complex non-linear dynamical systems.

Neural Network Unit

$$f(\sum_i w_i x_i + b)$$

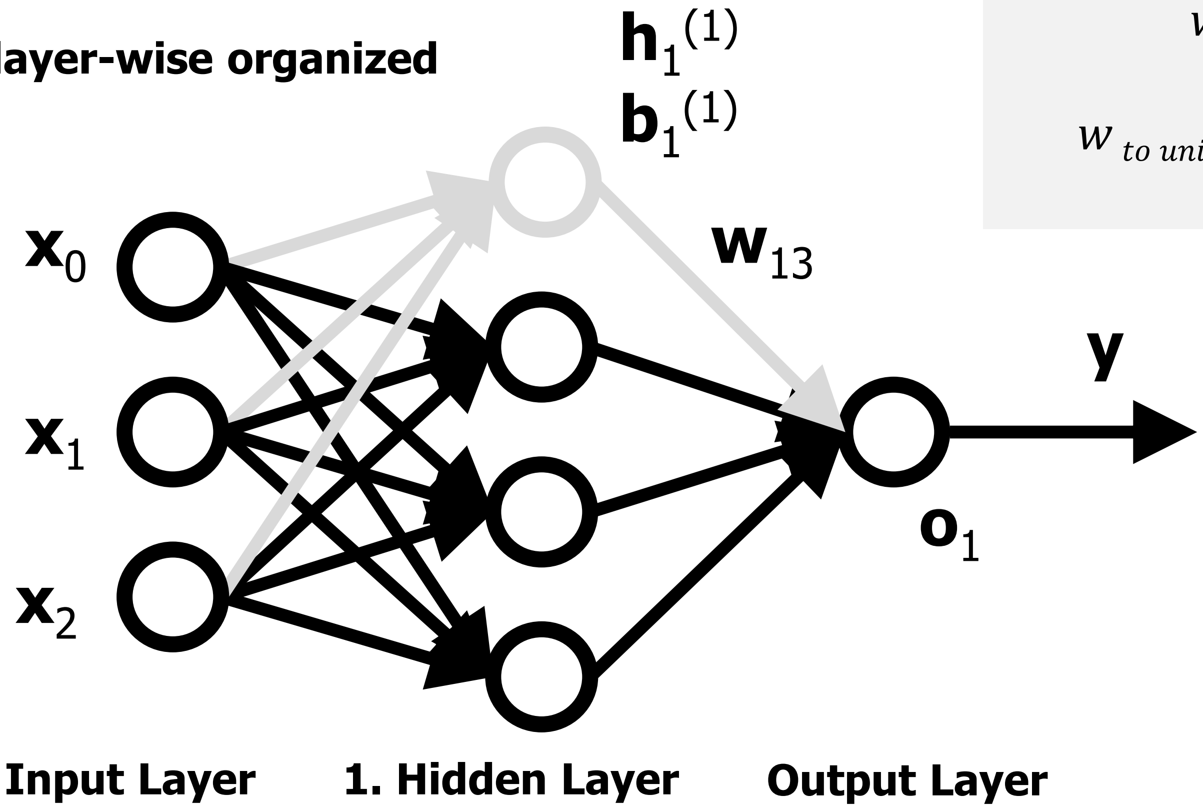
Neural networks are **mathematical models** that map an input to an output

$\mathbf{x}_i$	Inputs
$\mathbf{w}_i$	Weights
$\mathbf{b}$	Biases
$f(.)$	Activation function
$\mathbf{y}$	Output



Neural Network Layers

Neural networks are **layer-wise organized**



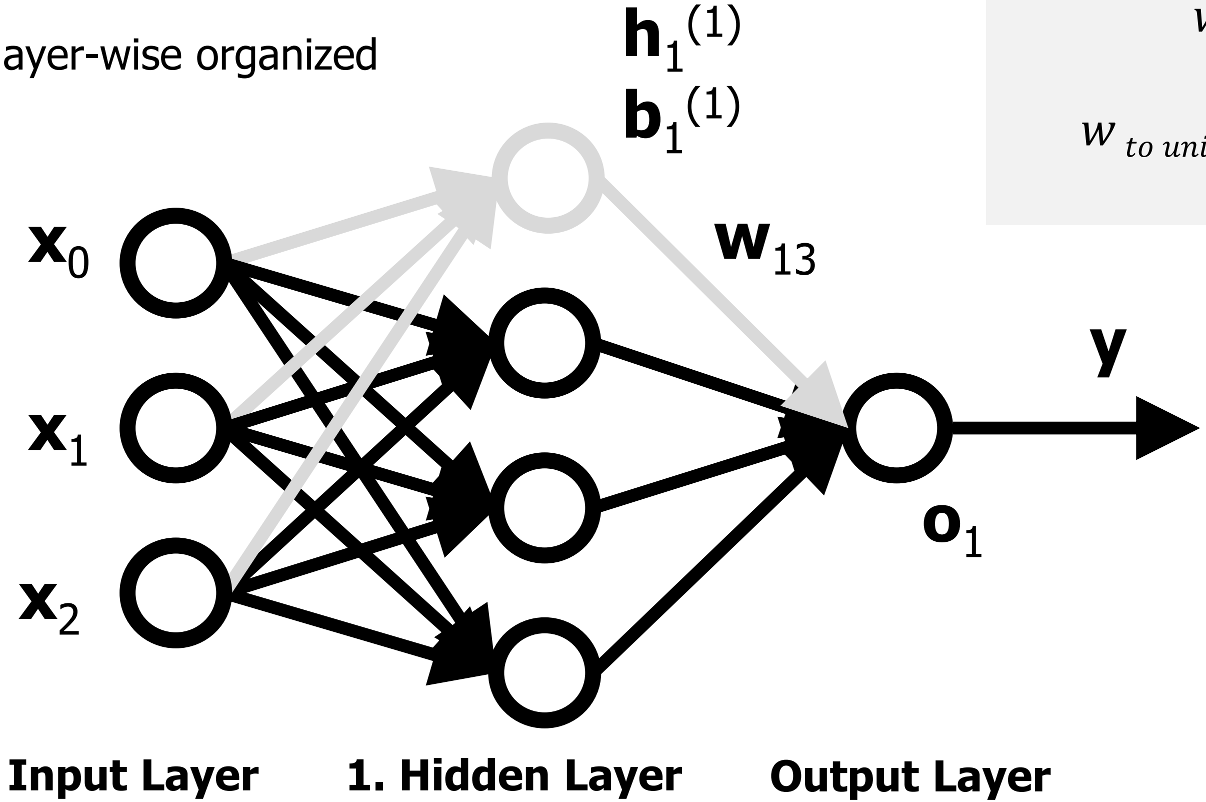
wji

$w_{to\ unit\ from\ unit}$

Neural Network Layers

Neural networks are layer-wise organized

Fully connected single layer neural network

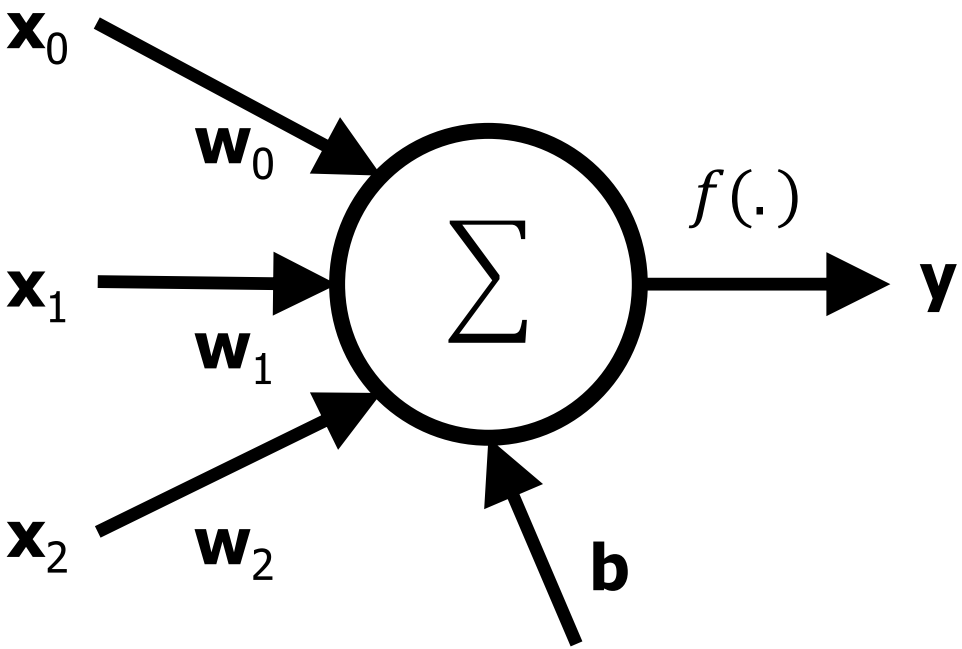


wji

$w_{to\,unit\,from\,unit}$

Neural Network Forwardpass

Repeated matrix multiplication within an activation function



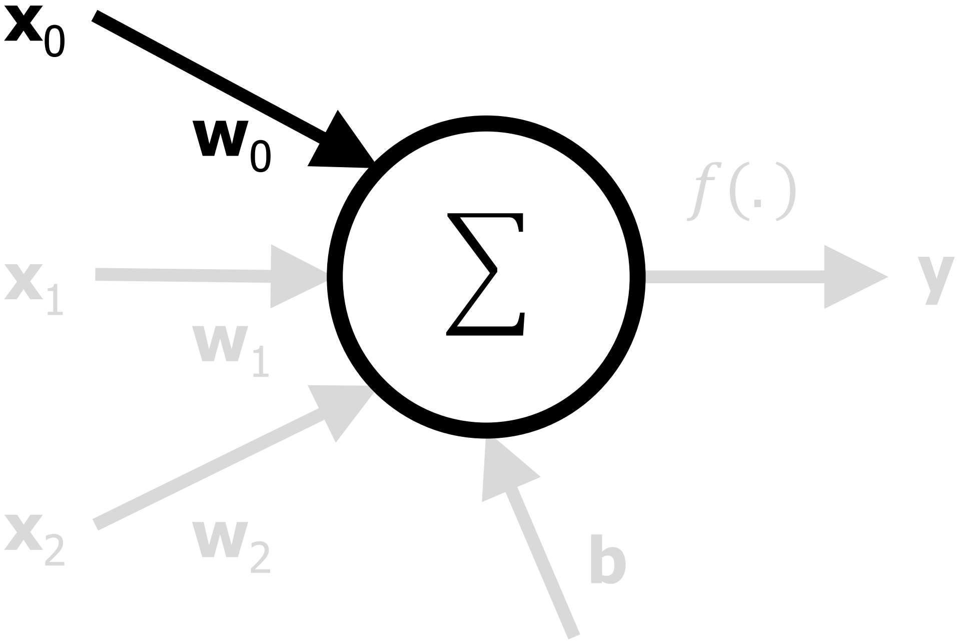
$$y = f(\sum_i w_i x_i + b)$$

$$= f(\boldsymbol{W}^T \boldsymbol{x} + b)$$

$$= f(w_0 x_0 + w_1 x_1 + w_2 x_2 + b)$$

Neural Network Forwardpass

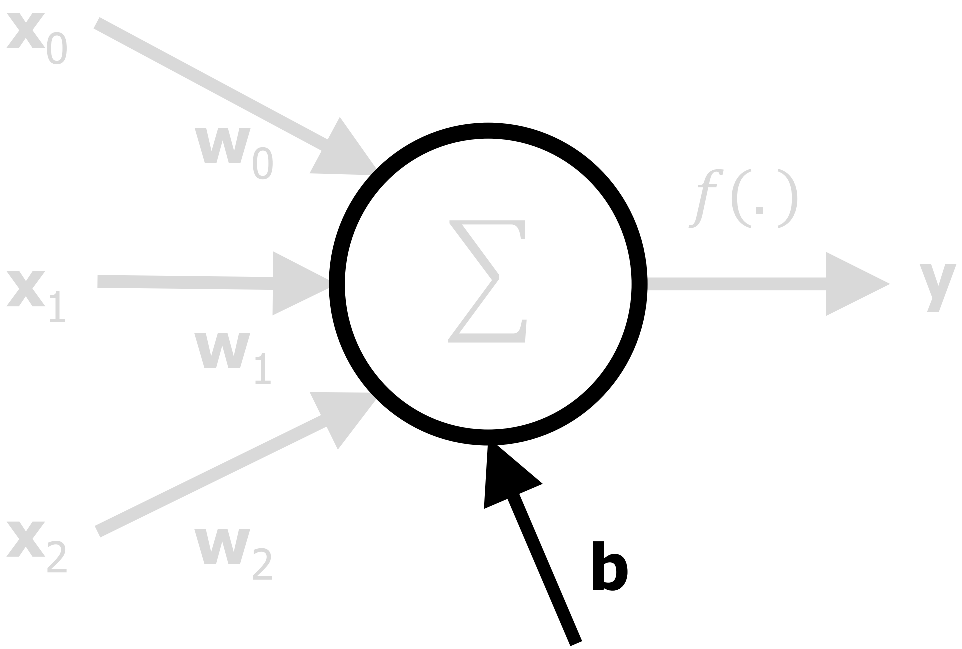
Repeated matrix multiplication within an activation function



$$y = f(w_0 x_0 + w_1 x_1 + w_2 x_2 + b)$$

Neural Network Forwardpass

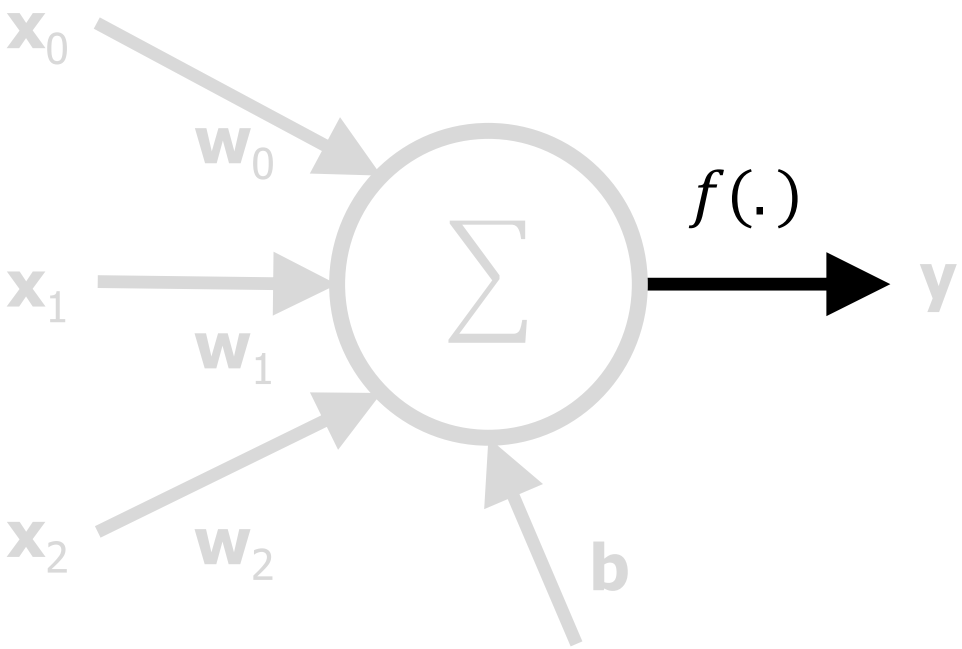
Repeated matrix multiplication within an activation function



$$y = f(w_0x_0 + w_1x_1 + w_2x_2 + b)$$

Neural Network Forwardpass

Repeated matrix multiplication within an activation function

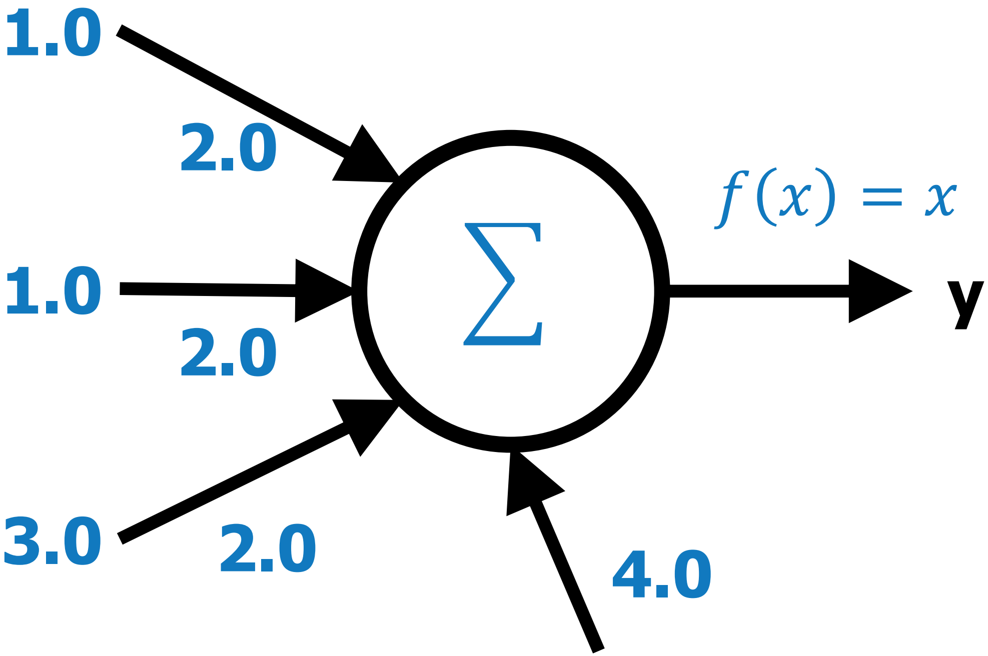


$$y = f(w_0x_0 + w_1x_1 + w_2x_2 + b)$$

```python
# Define graph / network
weights = {
    'h1': tf.Variable(np.reshape([np.float32(2.0), np.float32(2.0), np.float32(2.0)], (3, 1))),
    # 'h1': tf.Variable(tf.random_normal([n_input, n_units])),
}

biases = {
    'b1': tf.Variable(np.reshape([np.float32(4.0)], (1, 1))),
    # 'b1': tf.Variable(tf.random_normal([n_units])),
}


def unit(x0, weights, biases):
    # unit / neuron structure
    layer_1 = tf.add(tf.matmul(tf.cast(x0, tf.float32), weights['h1']), biases['b1'])
    # activation function
    y_pred = tf.nn.relu(layer_1)
    return y_pred

```

Neural Network Forwardpass

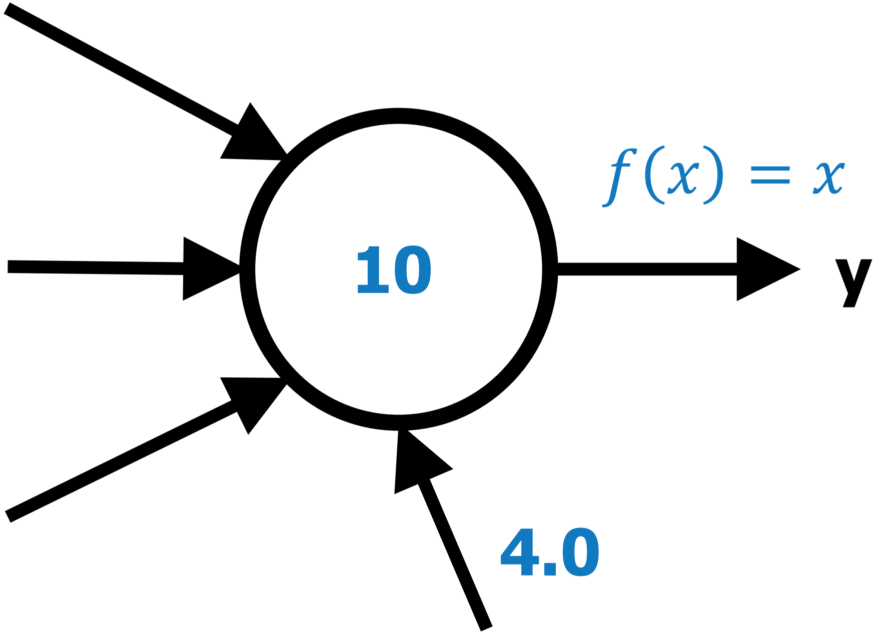
Example with input [1.0, 1.0, 3.0] and weights [2.0, 2.0, 2.0] and bias [4.0] and linear function as activation function



$$y = f(2.0 + 2.0 + 6.0 + 4.0)$$

Neural Network Forwardpass

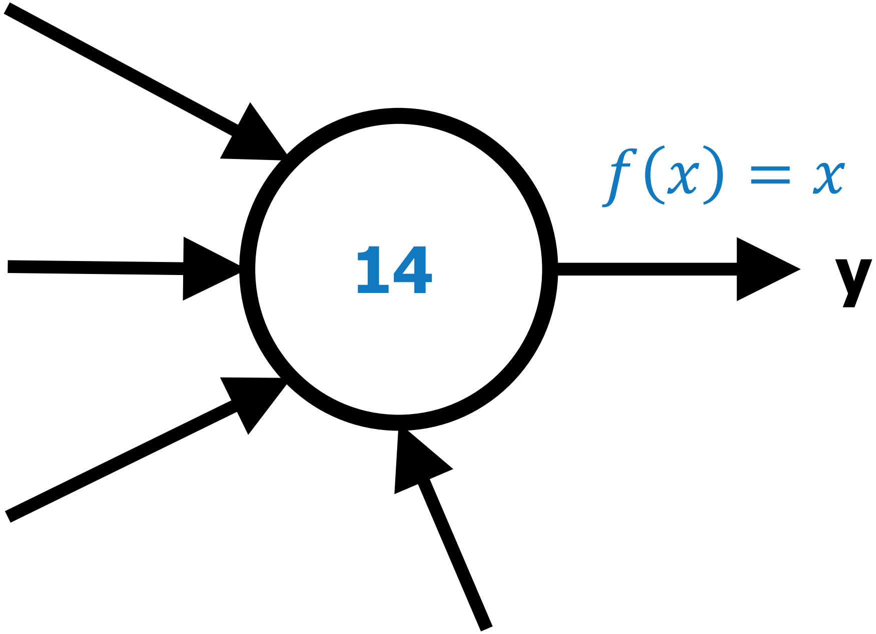
Example with input [1.0, 1.0, 3.0] and weights [2.0, 2.0, 2.0] and bias [4.0] and linear function as activation function



$$y = f(10.0 + 4.0)$$

Neural Network Forwardpass

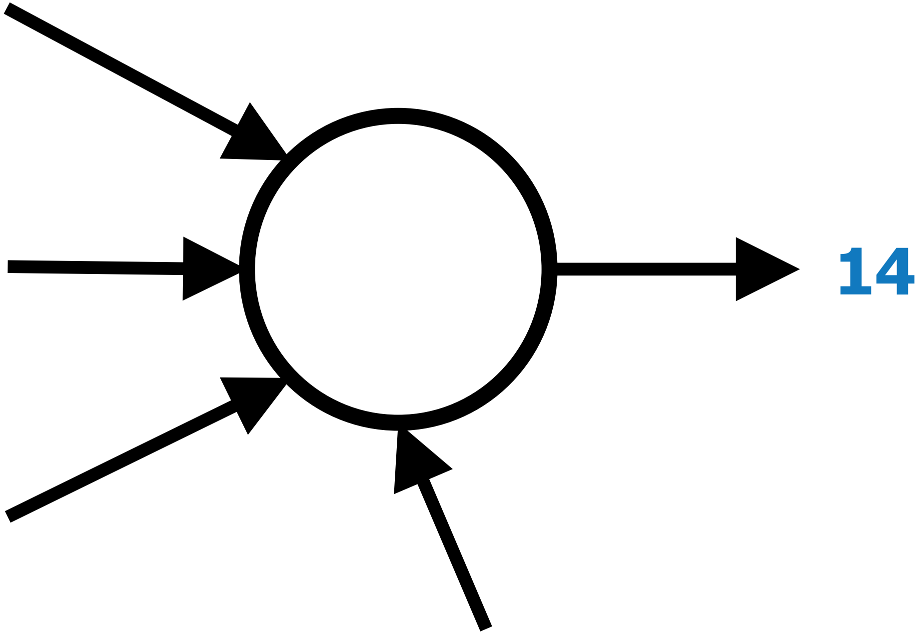
Example with input [1.0, 1.0, 3.0] and weights [2.0, 2.0, 2.0] and bias [4.0] and linear function as activation function



$$y = f(14.0)$$

Neural Network Forwardpass

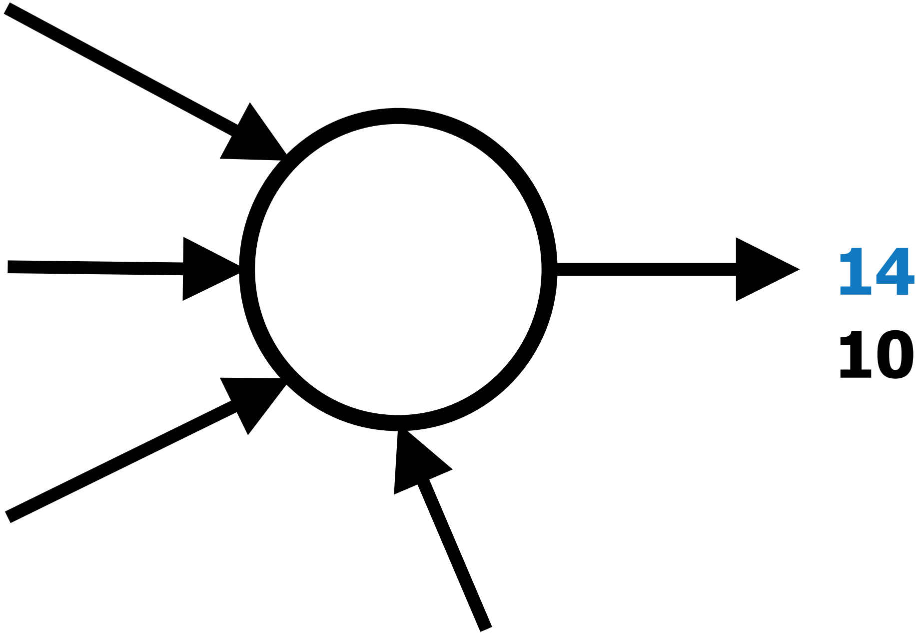
Example with input [1.0, 1.0, 3.0] and weights [2.0, 2.0, 2.0] and bias [4.0] and linear function as activation function



$$y = 14.0$$

Neural Network Supervised Learning

In supervised learning **we know the expected output** $\tilde{y}$ of our model, given a certain input x.



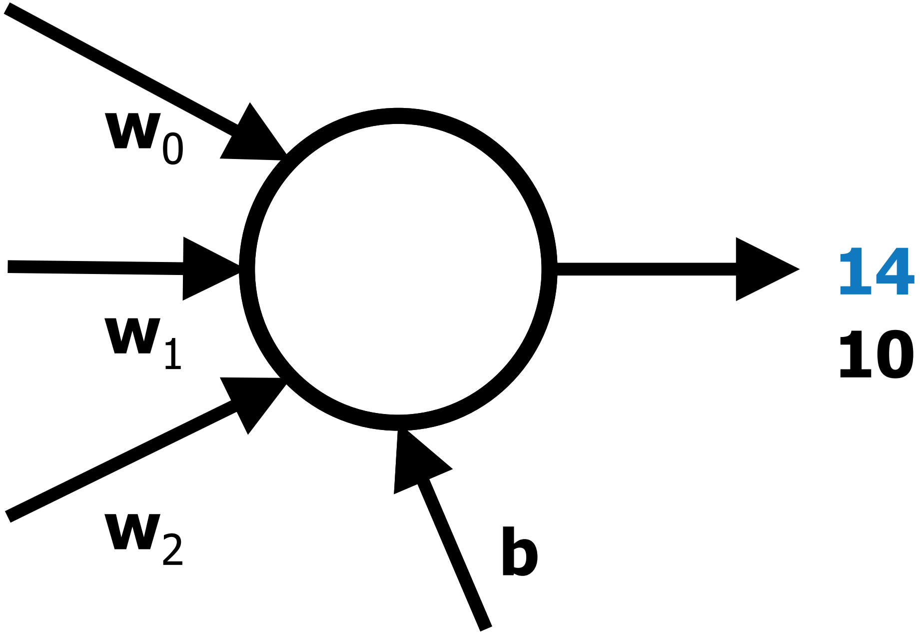
$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

Neural Network Supervised Learning

By **iteratively updating** the model's parameters θ namely $\boldsymbol{W}, \boldsymbol{b}$ the model learns to depict the knowledge inherent to the data.

At some point the model will be able to **generate the expected output**.



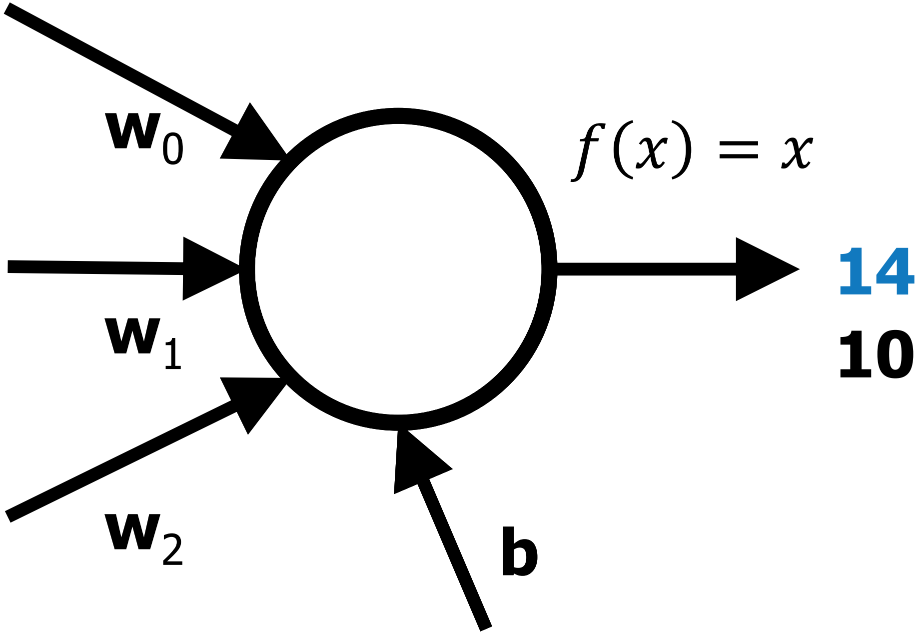
$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

```python
# Input
x = tf.Variable(np.reshape([1.0, 1.0, 3.0], (1, 3)))
# predicted output
y_pred = unit(x, weights, biases)
# Expected output
y_gt = tf.Variable(np.reshape([10.0], (1, 1)))

```

Neural Network Back Propagation

The model's parameters θ are updated by **backpropagation** of the error through the model by computing the gradients in our model.

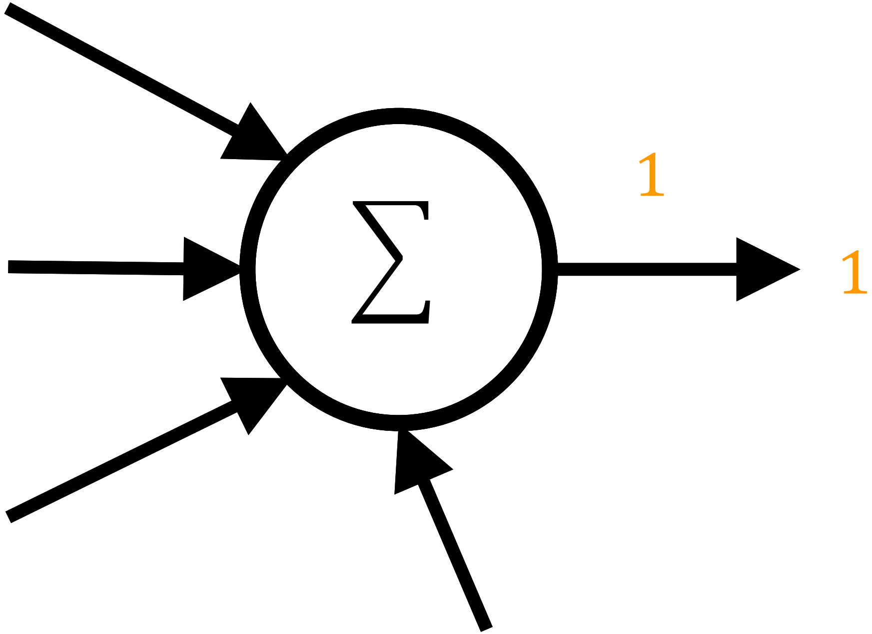


$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

Neural Network Back Propagation

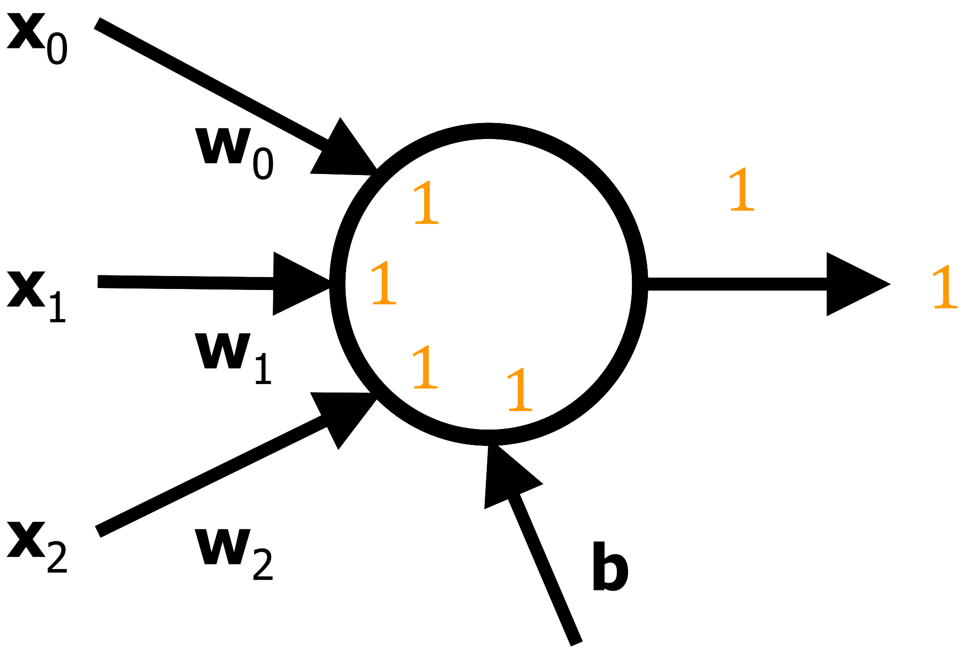
The model's parameters θ are updated by **backpropagation** of the error through the model by computing the gradients in our model.



$$\frac{\partial f(x)}{\partial x} = \frac{\partial x}{\partial x} = 1.0$$

Neural Network Back Propagation

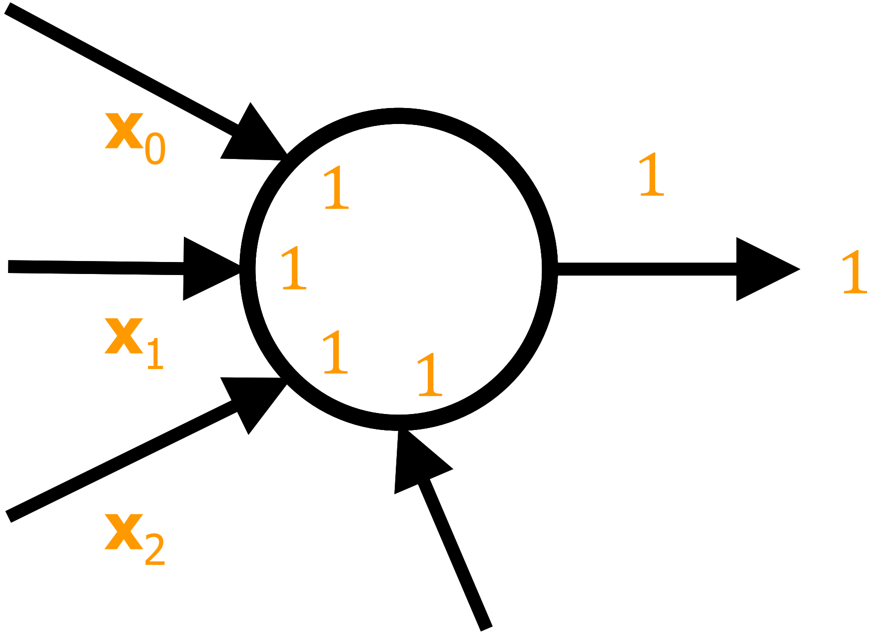
The model's parameters θ are updated by **backpropagation** of the error through the model by computing the gradients in our model.



$$\frac{\partial(w_0x_0 + w_1x_1 + w_2x_2 + b)}{\partial(w_0x_0)} = 1.0$$

Neural Network Back Propagation

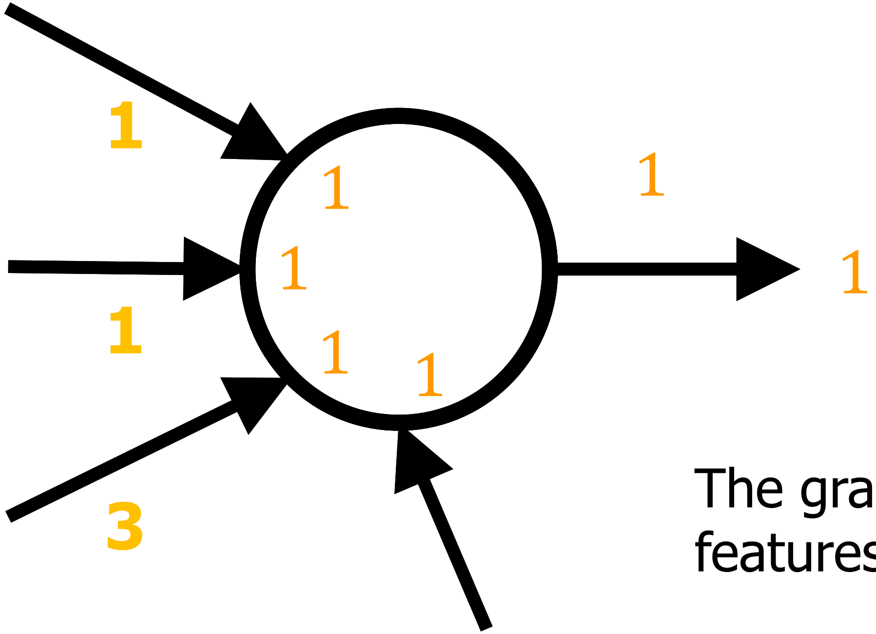
The model's parameters θ are updated by **backpropagation** of the error through the model by computing the gradients in our model.



$$\frac{\partial(w_0 x_0)}{\partial(w_0)} = x_0$$

Neural Network Back Propagation

The model's parameters θ are updated by **backpropagation** of the error through the model by computing the gradients in our model.

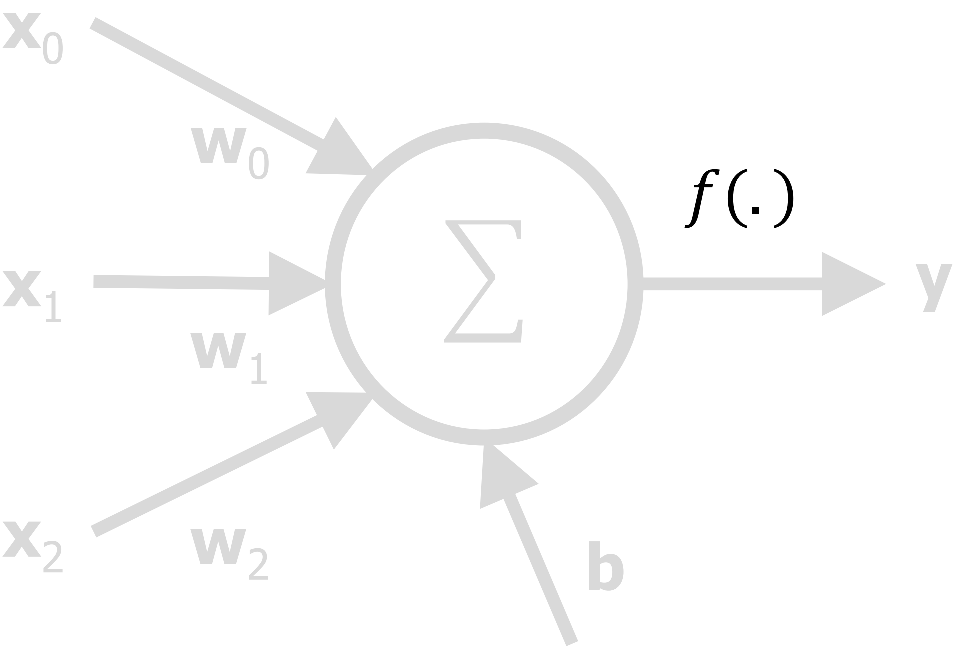


$$\frac{\partial(w_0 x_0)}{\partial(w_0)} = x_0$$

The gradients tell you about the features' influence

Neural Network Activation Functions

Every **activation function** $f(z)$ takes a single number z and performs a certain mathematical operation on it. z is also known as the cell state.



$$f(\sum_i w_i x_i + b)$$

Neural Network Activation Functions

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

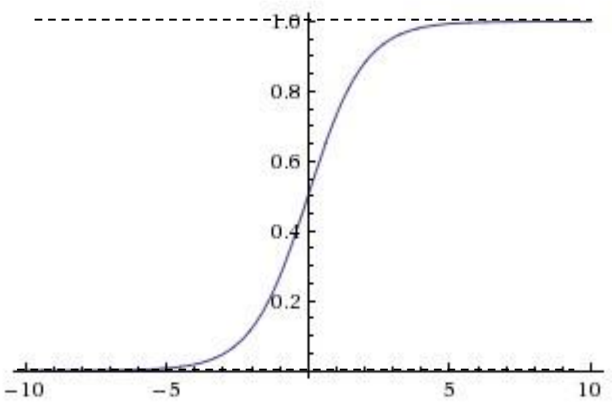
Sigmoid

Complies with the interpretation of a firing neuron, between zero and one

Saturates and vanishes gradients

Outputs are not zero-centered

Complying derivative characteristics



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

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

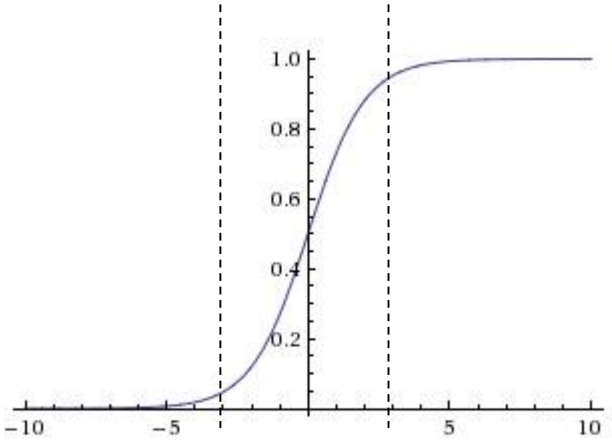
Sigmoid

Complies with the interpretation of a firing neuron, between zero and one

Saturates and vanishes gradients

Outputs are not zero-centered

Complying derivative characteristics



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

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

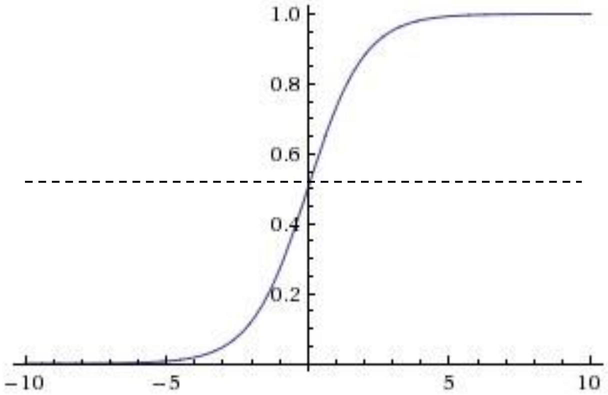
Sigmoid

Complies with the interpretation of a firing neuron, between zero and one

Saturates and vanishes gradients

Outputs are not zero-centered

Complying derivative characteristics



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

$$f(z) = 1 - \frac{1}{1+\mathrm{e}^{-x}}$$

$$f'(z) = (1 - f(z))f(z)$$

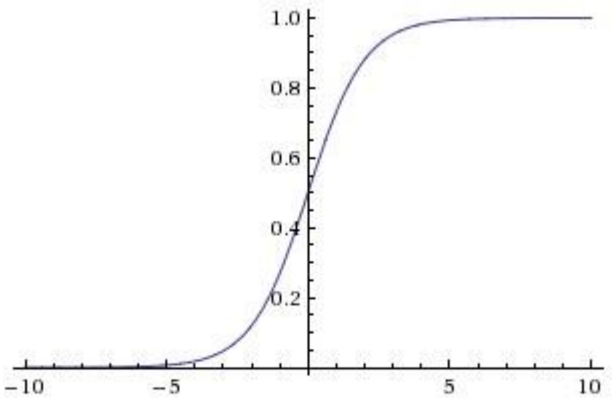
Sigmoid

Complies with the interpretation of a firing neuron, between zero and one

Saturates and vanishes gradients

Outputs are not zero-centered

Complying derivative characteristics



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

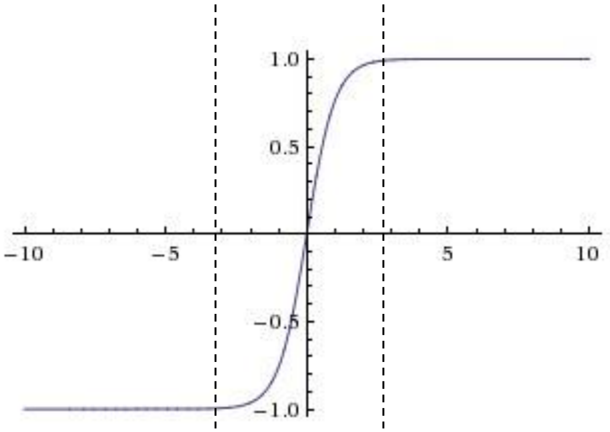
$$f(z) = 1 - \frac{2}{e^{2z} + 1}$$

Tanh

Saturates and vanishes gradients

Outputs are zero-centered in a range between minus one and one

Complying derivative characteristics



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

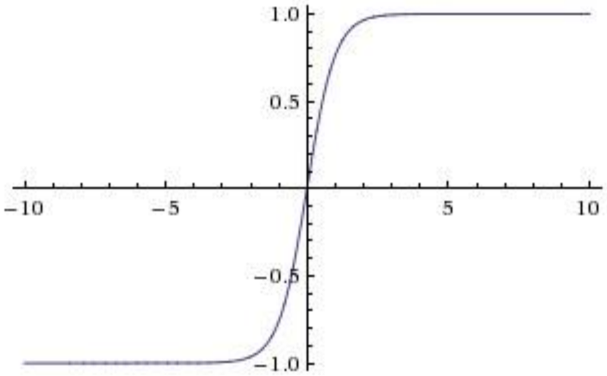
$$f(z) = 1 - \frac{2}{e^{2z} + 1}$$

Tanh

Saturates and vanishes gradients

Outputs are zero-centered in a range between minus one and one

Complying derivative characteristics



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

$$f(z) = 1 - \frac{2}{e^{2z} + 1}$$

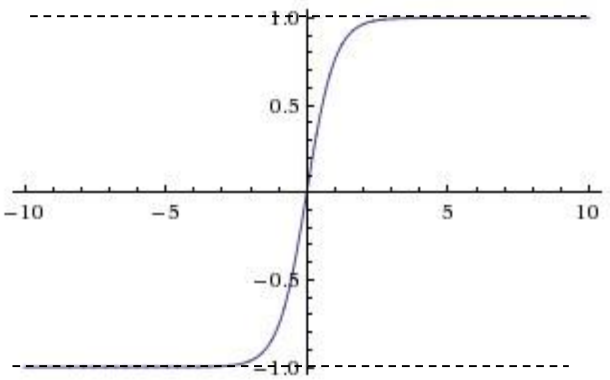
Tanh

$$f'(z) = 1 - f(z)^2$$

Saturates and vanishes gradients

Outputs are zero-centered in a range between minus one and one

Complying derivative characteristics



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

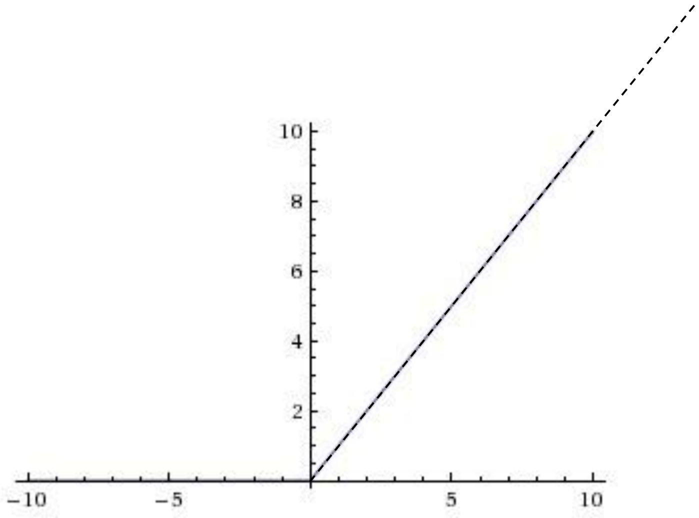
$$f(z) = \max\{0, z\}$$

ReLU (Rectified Linear Unit)

Does not saturate in the positive domain and thus the gradients do not vanish in the positive direction and learning is accelerated

Cheap operation of thresholding at zero

ReLUs can be fragile and "die" during training when the weights are updated too far into the negative domain. Fixed by leaky ReLU



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

$$f(z) = \max\{0, z\}$$

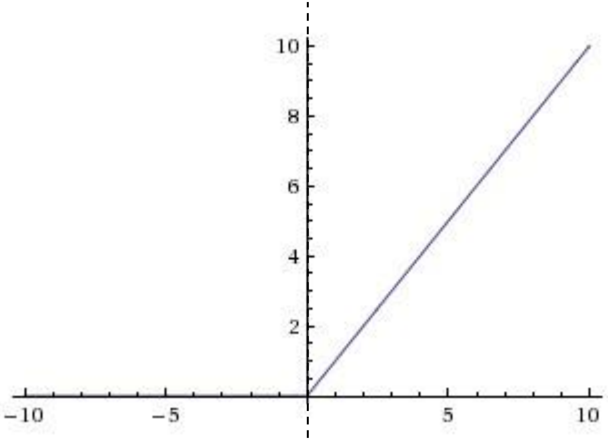
$$f'(z) = \begin{cases} 0, & \text{if } z < 0. \\ 1, & \text{if } z > 0. \end{cases}$$

ReLU (Rectified Linear Unit)

Does not saturate and thus the gradients do not vanish and learning is accelerated

Cheap operation of thresholding at zero

ReLUs can be fragile and "die" during training when the weights are updated too far into the negative domain. Fixed by leaky ReLU



http://cs231n.github.io/neural-networks-1/

Neural Network Activation Functions

$$f(z) = \max\{0, z\}$$

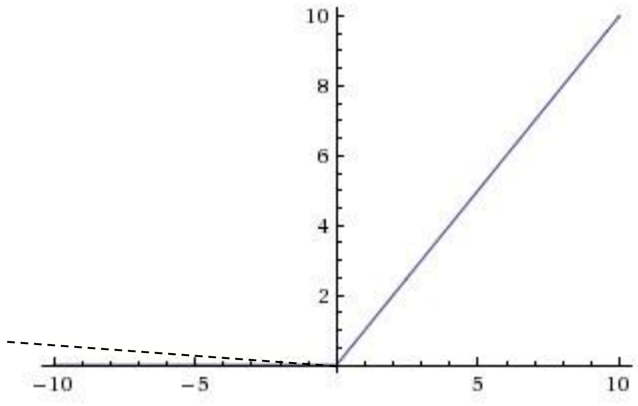
$$f'(z) = \begin{cases} 0, & \text{if } z < 0. \\ 1, & \text{if } z > 0. \end{cases}$$

ReLU (Rectified Linear Unit)

Does not saturate and thus the gradients do not vanish and learning is accelerated

Cheap operation of thresholding at zero

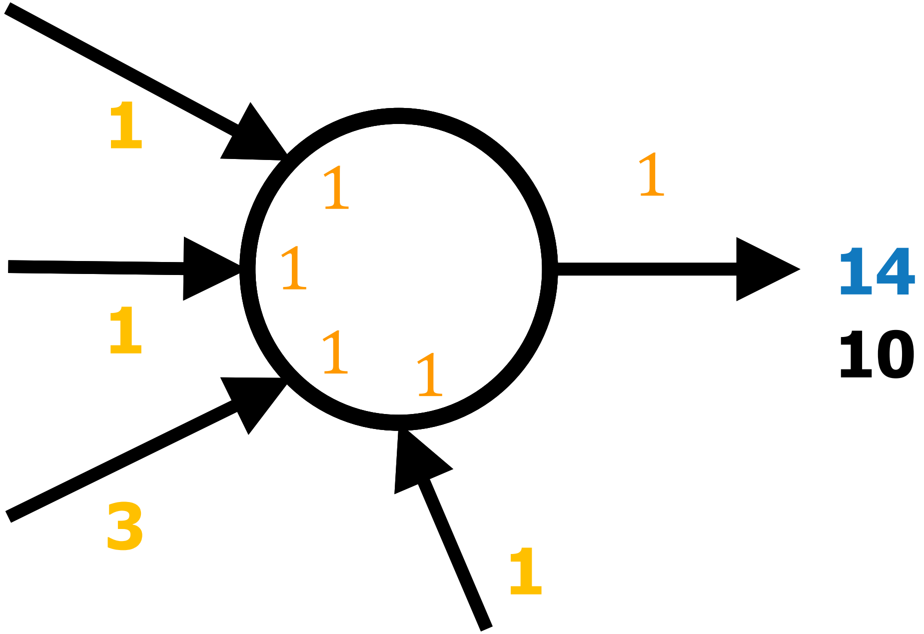
ReLUs can be fragile and "die" during training when the weights are updated too far into the negative domain. Fixed by leaky ReLUs and an adjusted learning rate.



http://cs231n.github.io/neural-networks-1/

Neural Network Objective Function

The model's parameters θ are updated by **backpropagation** of the error or loss through the model by computing the gradients in our model.



$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

$$J(\hat{y}, \tilde{y})$$

Neural Network Objective Function

The error or loss of the model measures the compatibility between a prediction $\hat{y}$ and the ground truth label $\tilde{y}$.

There are multiple ways to model the loss, these functions are called objective functions or loss functions.

$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

$$J(\hat{y}, \tilde{y})$$

Neural Network Objective Function

The error or loss of the model measures the compatibility between a prediction $\hat{y}$ and the ground truth label $\tilde{y}$.

There are multiple ways to model this compatibility, these functions are called **objective functions** or loss functions J.

$$L = \frac{1}{N} \sum_i J_i$$

$\hat{y} = 14.0$

$\tilde{y} = 10.0$

$J(\hat{y}, \tilde{y})$

Neural Network Objective Function – Regression Objective Function

Regression is the task of predicting real-valued quantities. For this task, it is common to compute the loss between the predicted quantity and the true answer.

L1 norm

L2 squared norm

$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

$$J(\hat{y}, \tilde{y})$$

Neural Network Objective Function – Regression Objective Function

Regression is the task of predicting real-valued quantities. For this task, it is common to compute the loss between the predicted quantity and the true answer.

L1 norm

$$J_i = \|f - y_i\|_1 = \sum_j |\, f_j - (y_i)_j|$$

$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

4

L2 squared norm

Neural Network Objective Function – Regression Objective Function

Regression is the task of predicting real-valued quantities. For this task, it is common to compute the loss between the predicted quantity and the true answer.

L1 norm

L2 squared norm

$$J_i = \|f - y_i\|_2^2$$

$\hat{y} = 14.0$

$\tilde{y} = 10.0$

16

Neural Network Objective Function – Classification Objective Function

Classification here, we assume a dataset of samples and a single correct label (out of a fixed set) for each sample.

Cross-entropy

$$J = -\frac{1}{N}\left(\sum_{i=1}^{N} \mathbf{y_i} \cdot \log(\hat{\mathbf{y}}_\mathbf{i})\right)$$

Note: Will be discussed in detail later in the lecture

Neural Network Objective Function – Closer Look

Regression losses (e.g. L2) are more fragile and harder to optimize, output exactly one correct value.

Classification losses (e.g. Softmax), output a distribution where only the magnitudes matter.

When faced with a regression task, consider discretizing your outputs to bins and perform a classification

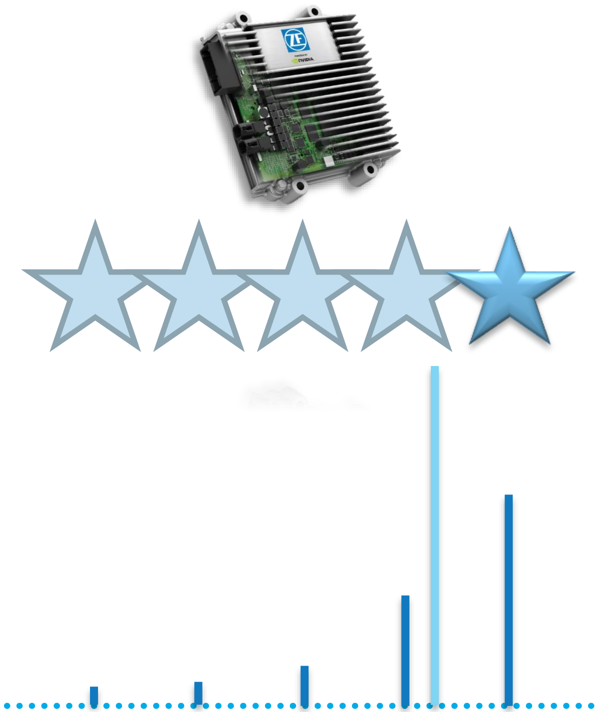
https://9gag.com

Neural Network Objective Function – Closer Look

Regression losses (e.g. L2) are more fragile and harder to optimize, to output exactly one correct value than

classification losses (e.g. Softmax), to output a distribution where only the magnitudes matter.

When faced with a regression task, consider discretizing your outputs to bins and perform a classification



Neural Network Objective Function – Metrics

Objective functions are optimized during training.

Metrics are a standard of measurement, especially one that evaluates a system.

Example Object Detection

Neural Network Objective Function – Metrics

Objective function

$$J_i = \|f - y_i\|_2^2$$

Metrics

- Intersection over union
- Precision & Recall
- Anything to evaluate and analyse the performance of your model and to gain insights …

Note: Metrics do not need to contribute a meaningful loss for the gradient

bbox 2D (0-based) bounding box of the object: Left, top, right, bottom image coordinates

References

[9] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[10] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[11] Medium article on metrics for object detection (April 2019) *https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173*

This lecture in one slide

Introduction and motivation for deep learning

Neural network conception

Optimization

Stochastic Gradient Descent

Momentum methods

Adaptive methods

Vanishing and Exploding Gradients

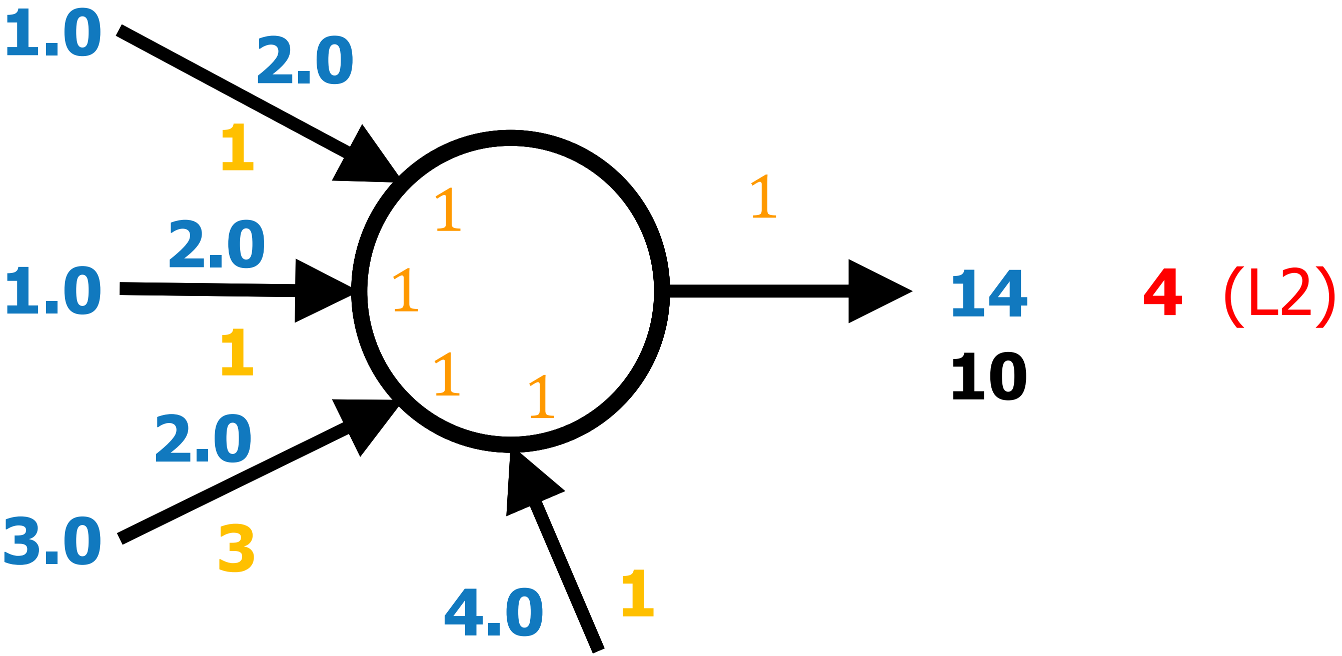
Weight Initialization

Regularization

Neural Network Optimization

The error or loss is calculated by a forward pass and the **objective function**.

The model's parameters θ are updated by **backpropagation** of the error or loss through the model by computing the gradients in our model.



Neural Network Optimization – Stochastic gradient descent

Algorithm 1 Stochastic gradient descent - $O(n)$

1: **procedure** WEIGHT UPDATE (θ initial weights, ϵ_i learning rate in iteration k, m batch size)
2: $\quad k \leftarrow 1$
3: $\quad$ **while** stopping criterion not met **do**
4: $\quad\quad$ Estimate average batch gradient: $\hat{\mathbf{g}} = \frac{1}{m} \nabla_\theta \sum_i^m L(f(\mathbf{x}_i; \theta); \mathbf{y}_i)$
5: $\quad\quad$ Update the weights: $\theta' = \theta - \epsilon_k \, \hat{\mathbf{g}}(\theta)$
6: $\quad\quad k \leftarrow k + 1$

Neural Network Optimization – Stochastic gradient descent

Stochastic (gradient of a batch) as opposed to deterministic (gradient of the whole dataset)

Standard error of the mean.
Unbiased estimate of the gradient.
Computational effort

Neural Network Optimization – Stochastic gradient descent

Stochastic

Standard error of the mean $\frac{\sigma}{\sqrt{m}}$. Decreases only by $\sqrt{m}$.
With *m* samples in a batch.

Unbiased estimate of the gradient.
Computational effort

Neural Network Optimization – Stochastic gradient descent

Stochastic

Standard error of the mean.

Randomly selected set of m training samples for a batch achieves an **unbiased estimate of the gradient**.

Computational effort

Neural Network Optimization – Stochastic gradient descent

Stochastic

Standard error of the mean.

Unbiased estimate of the gradient.

Limiting number of m samples per batch, sets and upper bound to the **computational effort** during the update (growing datasets, growing sample size)

```python

cost = tf.losses.mean_squared_error(labels=y_gt, predictions=y_pred)
opt = tf.train.GradientDescentOptimizer(0.0001)
train = opt.minimize(cost)


# Run graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for _ in range(n_updates):
        sess.run(train)
```

SGD

```
Biases: [[ 3.99840403]]
Prediction [[ 13.96173477]]

Gradient [  7.84316492   7.84316492  23.60477257]
Weights: [ 1.99761569  1.99761569  1.99283969]
Biases: [[ 3.997612]]
Prediction [[ 13.9427824]]

Gradient [  7.7917676    7.7917676   23.47492409]
Weights: [ 1.99683142  1.99683142  1.99047923]
Biases: [[ 3.99682403]]
Prediction [[ 13.9239502]]
```

What do we observe, concerning:
- Gradients
- Weights and biases
- Prediction

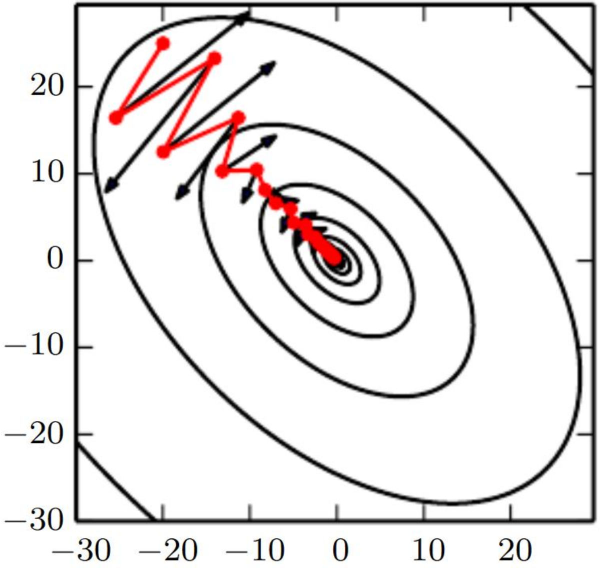
Neural Network Optimization – Momentum methods

Average gradients of past iterations as velocity **v**

Consider recent gradients stronger by accounting for friction α in $[0,1)$

$$\mathbf{v} = \alpha \mathbf{v} - \epsilon \mathbf{g},$$

$$\theta' = \theta + \mathbf{v}.$$



Red velocity, black current gradient [1]

SGD

```
Biases: [[ 3.99840403]]
Prediction [[ 13.96173477]]

Gradient [  7.84316492   7.84316492  23.60477257]
Weights: [ 1.99761569  1.99761569  1.99283969]
Biases: [[ 3.997612]]
Prediction [[ 13.9427824]]

Gradient [  7.7917676    7.7917676   23.47492409]
Weights: [ 1.99683142  1.99683142  1.99047923]
Biases: [[ 3.99682403]]
Prediction [[ 13.9239502]]
```

Momentum

```
Weights: [ 1.99780524  1.99780524  1.99341321]
Biases: [[ 3.99780416]]
Prediction [[ 13.94735718]]

Gradient [  7.73596764   7.73596764  23.33368492]
Weights: [ 1.99597359  1.99597359  1.98790956]
Biases: [[ 3.9959681]]
Prediction [[ 13.90345001]]

Gradient [  7.59647274   7.59647274  22.9798317 ]
Weights: [ 1.99382627  1.99382627  1.98144841]
Biases: [[ 3.99381113]]
Prediction [[ 13.85199928]]
```

What do we observe, concerning:

- Gradients
- Weights and biases
- Prediction

Neural Network Optimization – Adaptive methods

Adapting the learning rate throughout the optimization process

AdaGrad

Individually adapts the learning rates of all model parameters, inversely proportional to the historical values of the gradient.

RMSProp

Adam

Adapting the learning rate throughout the optimization process

AdaGrad

RMSProp

Modifies AdaGrad by approaching the accumulation of historical gradient values as a exponentially weighted moving average. Influence of very old historical values is reduced.

Adam

Adapting the learning rate throughout the optimization process

AdaGrad
RMSProp

Adam
(Adaptive moments) combination of exponential weight decay together with first- and second-order moments.

Note:
There is no single best optimization algorithm. Adam is generally robust to the choice of hyperparameters, besides the learning rate. Adam is a reasonable choice for a start.

Adam

```
Gradient [ 0.01294271  0.01294271  0.0534188 ]
Weights: [ 1.84825361  1.84825361  1.48213327]
Biases: [[ 3.81778502]]
Prediction [[ 10.19217873]]

Gradient [ 0.01273675  0.01273675  0.05256975]
Weights: [ 1.84824812  1.84824812  1.48211086]
Biases: [[ 3.81777668]]
Prediction [[ 10.19202805]]


Process finished with exit code 1
```

Momentum

```
Weights: [ 1.99780524  1.99780524  1.99341321]
Biases: [[ 3.99780416]]
Prediction [[ 13.94735718]]

Gradient [  7.73596764   7.73596764  23.33368492]
Weights: [ 1.99597359  1.99597359  1.98790956]
Biases: [[ 3.9959681]]
Prediction [[ 13.90345001]]

Gradient [  7.59647274   7.59647274  22.9798317 ]
Weights: [ 1.99382627  1.99382627  1.98144841]
Biases: [[ 3.99381113]]
Prediction [[ 13.85199928]]
```

What do we observe, concerning:

- Gradients
- Weights and biases
- Prediction

Adam (lr 0.1)

```
Biases: [[ 3.9000001]]
Prediction [[ 12.82999992]]

Gradient [ 2.50330496  2.50330496  8.73594475]
Weights: [ 1.80395114  1.80395114  1.80308497]
Biases: [[ 3.80270195]]
Prediction [[ 11.75545788]]

Gradient [ 0.7409544   0.7409544   2.81789088]
Weights: [ 1.71574485  1.71574485  1.71282506]
Biases: [[ 3.71153021]]
Prediction [[ 10.80770302]]

Gradient [-0.44467068 -0.44467068 -1.84507322]
```

Momentum

```
Weights: [ 1.99780524  1.99780524  1.99341321]
Biases: [[ 3.99780416]]
Prediction [[ 13.94735718]]

Gradient [  7.73596764   7.73596764  23.33368492]
Weights: [ 1.99597359  1.99597359  1.98790956]
Biases: [[ 3.9959681]]
Prediction [[ 13.90345001]]

Gradient [  7.59647274   7.59647274  22.9798317 ]
Weights: [ 1.99382627  1.99382627  1.98144841]
Biases: [[ 3.99381113]]
Prediction [[ 13.85199928]]
```

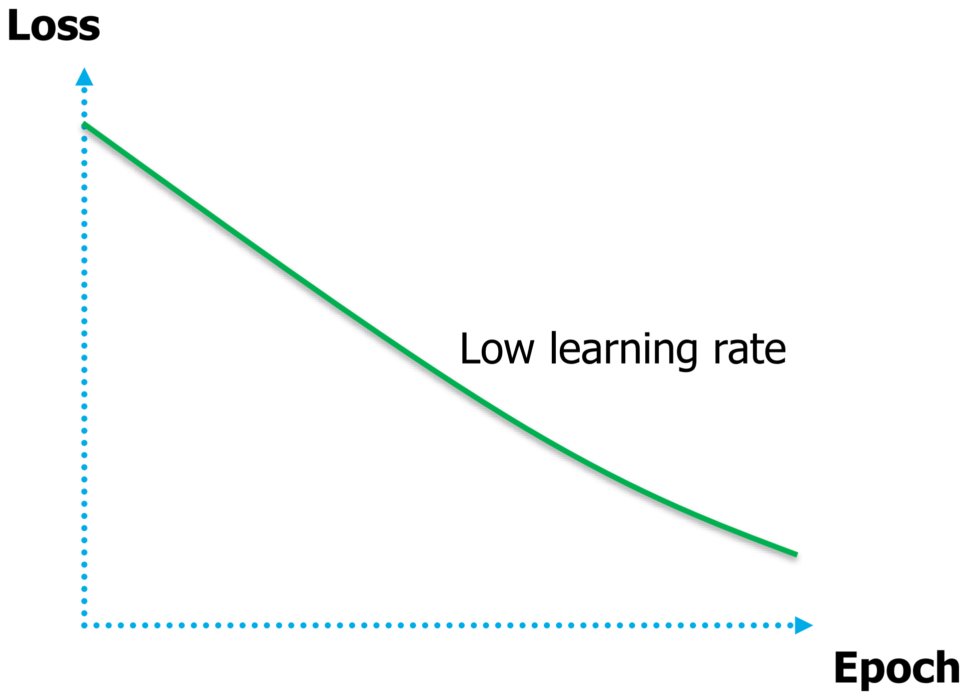
What do we observe, concerning:

- Gradients
- Weights and biases
- Prediction

Neural Network Optimization – Learning Rate

Low learning rates

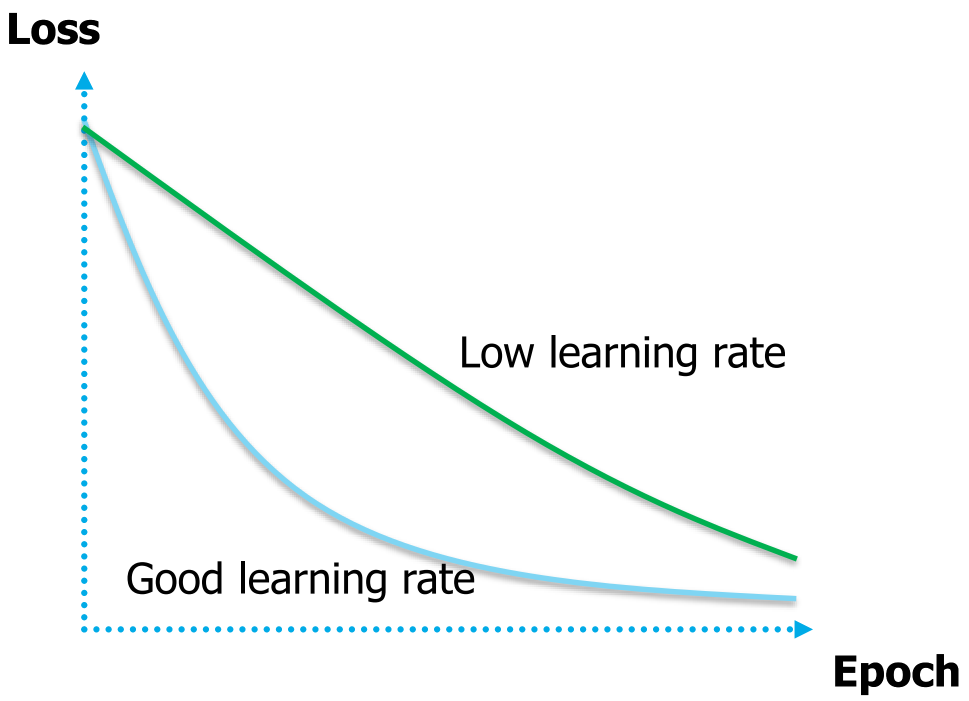
Loss decay will be linear, and result in high training times.



Neural Network Optimization – Learning Rate

Low learning rates
Loss decay will be linear, and result in high training times.

Higher learning rates
Loss decay will start to become exponential.

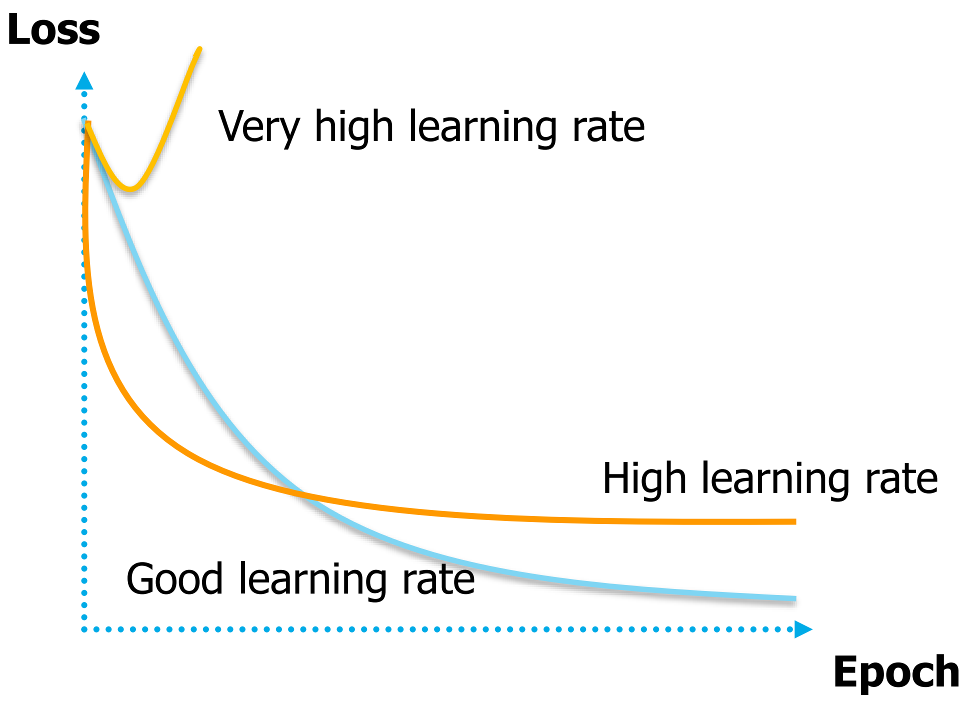


Neural Network Optimization – Learning Rate

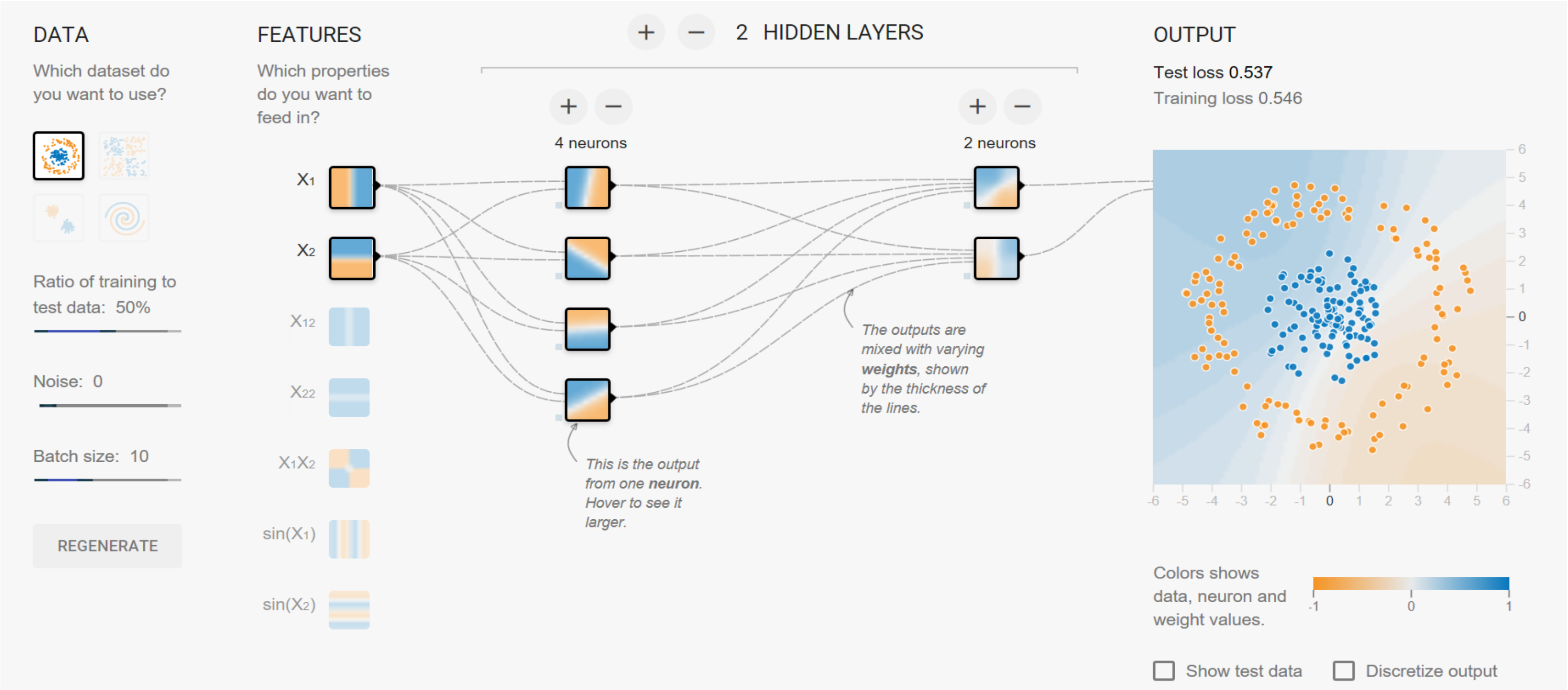
Higher learning rates

Loss decay will start to become exponential.

At some point the parameters will get stuck in worse parameter values, due to bouncing around, not being able to settle.



Neural Network Playground - Tinker with a Neural Network in your browser



http://playground.tensorflow.org

References

[12] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[13] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[14] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

[15] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[17] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, 2014.

[18] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

This lecture in one slide

Introduction and motivation for deep learning
Neural network conception
Optimization

Regularization
Parameter constraints
Batch methods
Dropout
Augmentation
Early stopping
Hyperparameter search

Optimization minimizes the error of a model on observed samples.

Machine Learning
Regularization

Neural Network Regularization

Optimization

Machine Learning prioritizes the model performance on unobserved data, assuming *i.i.d* (independent and identically distributed), called generalization.

Regularization

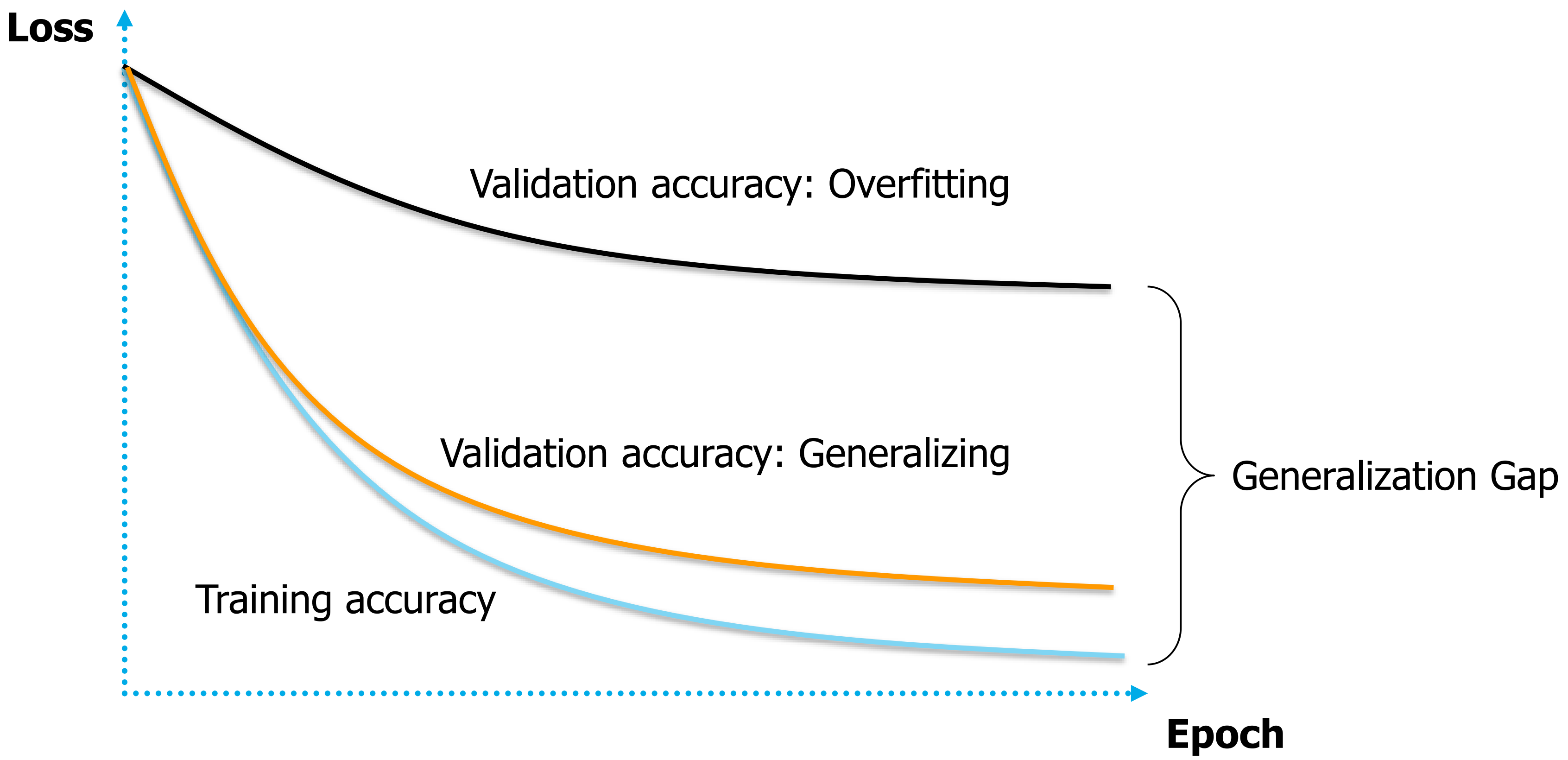
Neural Network Regularization

Optimization
Machine Learning

Regularization is the process of bridging the generalization gap between the performance on observed (training data) and unobserved samples (validation and test data).

Idea: Reducing the capacity of the model

Neural Network Regularization – Bridging the generalization gap

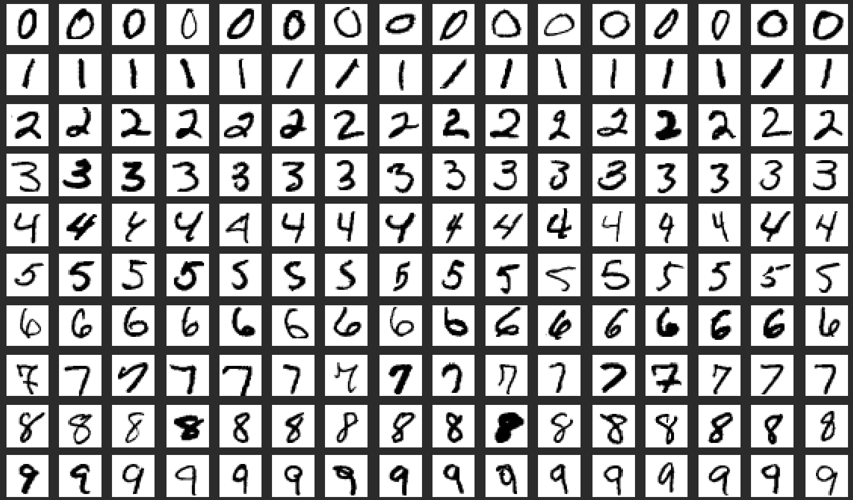


MNIST Dataset

The MNIST database, the ‘hello world!’ of machine learning.

Large database of handwritten digits. Grayscale images with dimension of 28x28 pixels.

60k training samples
10k testing images



https://en.wikipedia.org/wiki/MNIST_database#/media/File:MnistExamples.png

Parameter norm penalties

Adding a cost depending on the parameter values:

$$\tilde{J}(\theta;\mathbf{X},\mathbf{y}) = J(\theta;\mathbf{X},\mathbf{y}) + \alpha\Omega(\theta).$$

$$\Omega(\theta) = \frac{1}{2}||\mathbf{w}||_2^2,$$

The most common is the L2 norm penalty, shifting the parameter values to be small (also known as weight decay).

Idea: Small changes in the input have small influence on the predicted output.

Parameter sharing

Parameter norm penalties

Parameter sharing

Force tying parameter values, due to prior knowledge:

$$\mathbf{w}^A \text{ to equal } \mathbf{w}^B.$$

- Translation invariance in images (Convolution Filters)
- Recurring similar inputs (Recurrent Neural Networks)
- Shared feature space (Encoder-Decoder Architecture)

Parameter constraints

```
# 1. L2 Parameter norm penalty by kernel regularizer:
tf.keras.layers.Dense(512, activation=tf.nn.relu, kernel_regularizer=tf.keras.regularizers.l2(0.01)),
tf.keras.layers.Dense(512, activation=tf.nn.relu, kernel_regularizer=tf.keras.regularizers.l2(0.01)),
tf.keras.layers.Dense(512, activation=tf.nn.relu, kernel_regularizer=tf.keras.regularizers.l2(0.01)),
tf.keras.layers.Dense(512, activation=tf.nn.relu, kernel_regularizer=tf.keras.regularizers.l2(0.01)),
```

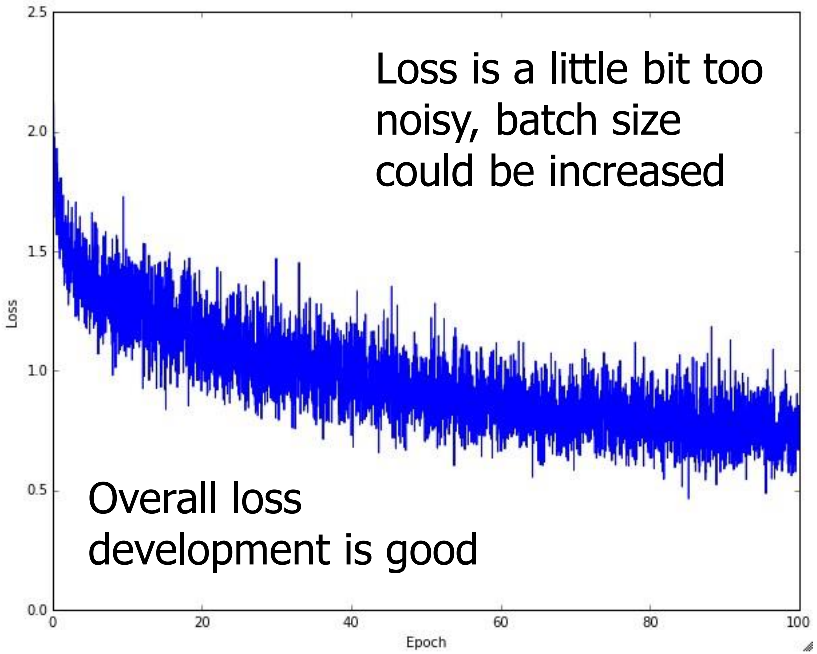
Why minibatches?

- Unbiased estimate of the gradient
- Computational effort

- Noise induced regularization for small batch sizes
  Note: This is usually not worth it

Which batch size should I go for?

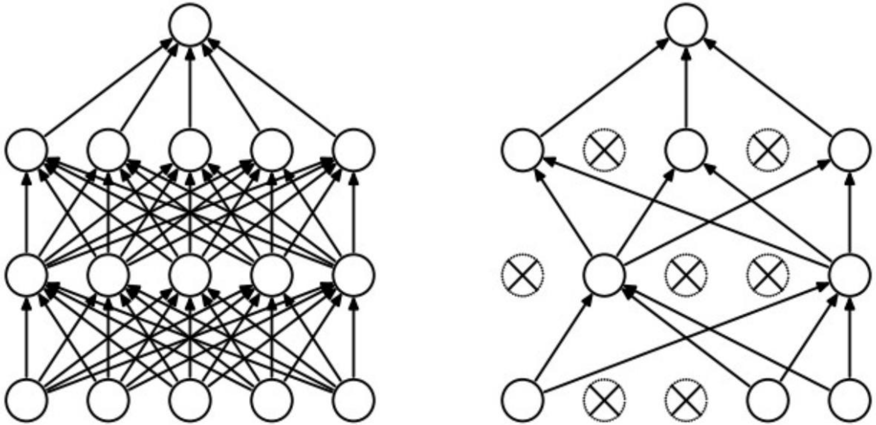
- Hardware restrictions set upper limit
- Power-of-two batch sizes match physical processor and improve runtime
- Loss band should be smooth, implying even gradient estimates.



http://cs231n.github.io/neural-networks-3/#baby

Neural Network Regularization – Dropout

Dropout keeps a neuron active with some probability (keep rate) during training, or setting it zero otherwise.



(a) Standard Neural Net (b) After applying dropout.

http://cs231n.github.io/neural-networks-2/#reg

Neural Network Regularization – Dropout

Dropout keeps a neuron active with some probability (keep rate) during training, or setting it zero otherwise.

For each weight update a different **sub neural network** is sampled from the standard neural network.

This implicitly trains an ensemble of networks, while inducing a regularization pressure, because **each parameter needs to function in all the ensembles**.

Neural Network Regularization – Dropout

Dropout keeps a neuron active with some probability (keep rate) during training, or setting it zero otherwise.

For each weight update a different sub neural network is sampled from the standard neural network.

This implicitly trains an ensemble of networks, while inducing a regularization pressure, because each parameter needs to function in all the ensembles.

During inference there is no dropout applied.

Note: The keep rate is commonly between 0 and 0.5

Dropout

```
# 2. Dropout:
tf.keras.layers.Dense(512, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(512, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(512, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(512, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.5),
```

Neural Network Regularization – Augmentation

Generalization improves with an **increased dataset size**.
The number of iterations an individual samples is used for training

Neural Network Regularization – Augmentation

Generalization improves with an increased dataset size.
The number of iterations an individual samples is used for training

Increasing number of samples demands a great effort:

- Collecting data
- Preparing data
- Annotate data

Neural Network Regularization – Augmentation

Generalization improves with an increased dataset size.
The number of iterations an individual samples is used for training

Increasing number of samples demands a great effort

Data augmentation presents a useful solution
By transforming the existing training samples, while keeping the affiliated ground truth samples.

Neural Network Regularization – Augmentation

Generalization improves with an increased dataset size.
The number of iterations an individual samples is used for training

Increasing number of samples demands a great effort

Data augmentation presents a useful solution

Examples of augmentation operations

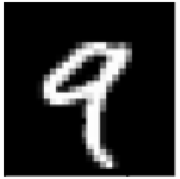
- Rotation, Zoom, Cropping, Distortion and Translation
- Brightness and Saturation

Neural Network Regularization – Augmentation

Think before you augment:

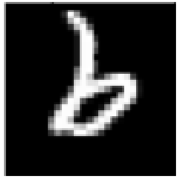
Prevent class switches and class breaks, know your data and your problem statement.

Initial sample



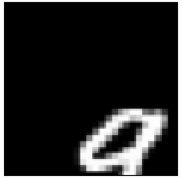
9

Rotation



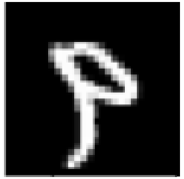
6

Shift



0

Mirror



NaN

Note: Make sure to motivate the boundary conditions of your augmentation operations.

Neural Network Regularization – Augmentation

There are a bunch of good libraries for this purpose

```python
import Augmentor

p = Augmentor.Pipeline("/home/user/augmentor_data_tests")
```

Now you can begin adding operations to the pipeline object:

```python
p.rotate90(probability=0.5)
p.rotate270(probability=0.5)
p.flip_left_right(probability=0.8)
p.flip_top_bottom(probability=0.3)
p.crop_random(probability=1, percentage_area=0.5)
p.resize(probability=1.0, width=120, height=120)
```

Once you have added the operations you require, you can sample images from this pipeline:

```python
p.sample(100)
```

Augmentor

https://github.com/mdbloice/Augmentor

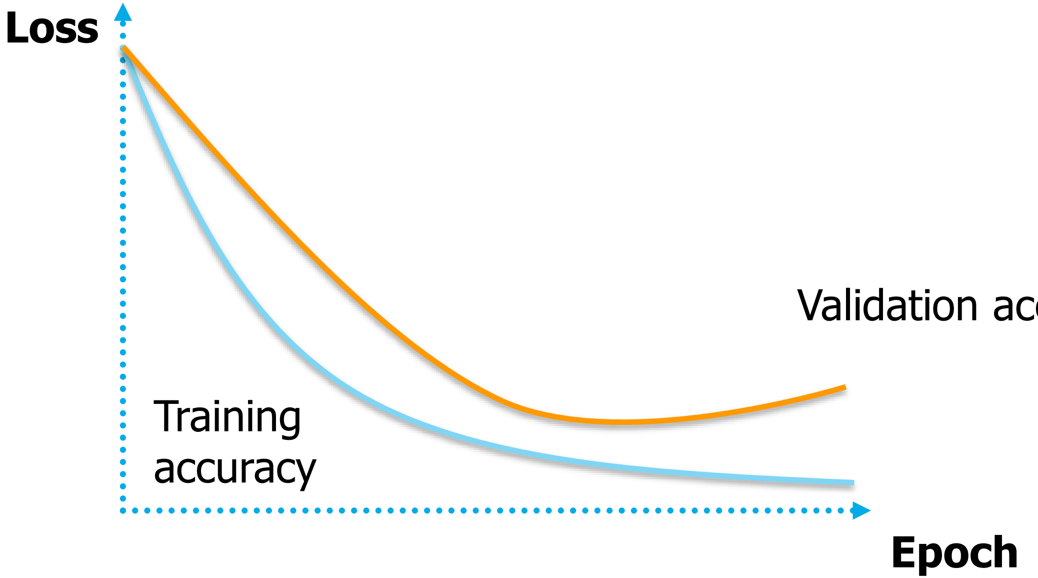
https://keras.io/preprocessing/image/

Augmentation

```python
# 3. Augmentation
x_train = x_train.reshape(x_train.shape[0], 1, 28, 28)
datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(
                                                                 featurewise_center=True,
                                                                 featurewise_std_normalization=True,
                                                                 rotation_range=20,
                                                                 width_shift_range=0.2,
                                                                 height_shift_range=0.2,
                                                                 horizontal_flip=False
                                                                 )

for e in range(10):
    print('Epoch', e)
    batches = 0
    for x_batch, y_batch in datagenerator.flow(x_train, y_train, batch_size=32):
        model.fit(np.reshape(x_batch, (-1, 28, 28)), y_batch, shuffle=True)
        batches += 1
        if batches >= len(x_train) / 32:
            # we need to break the loop by hand because
            # the generator loops indefinitely
            break
```

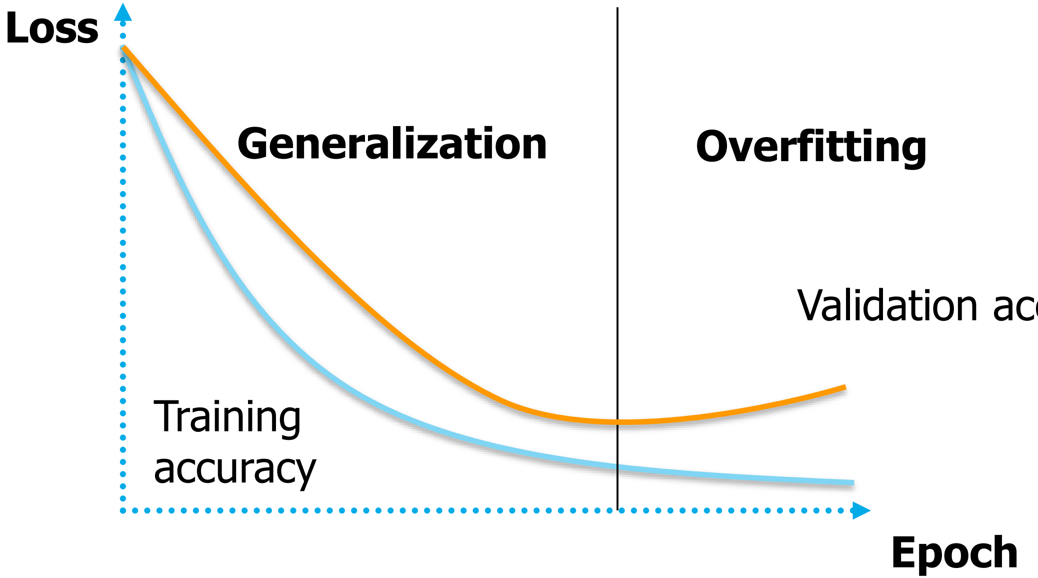
Neural Network Regularization – Early Stopping

When **training a model with large capacity** (large number of parameters), the training error steadily decreases.



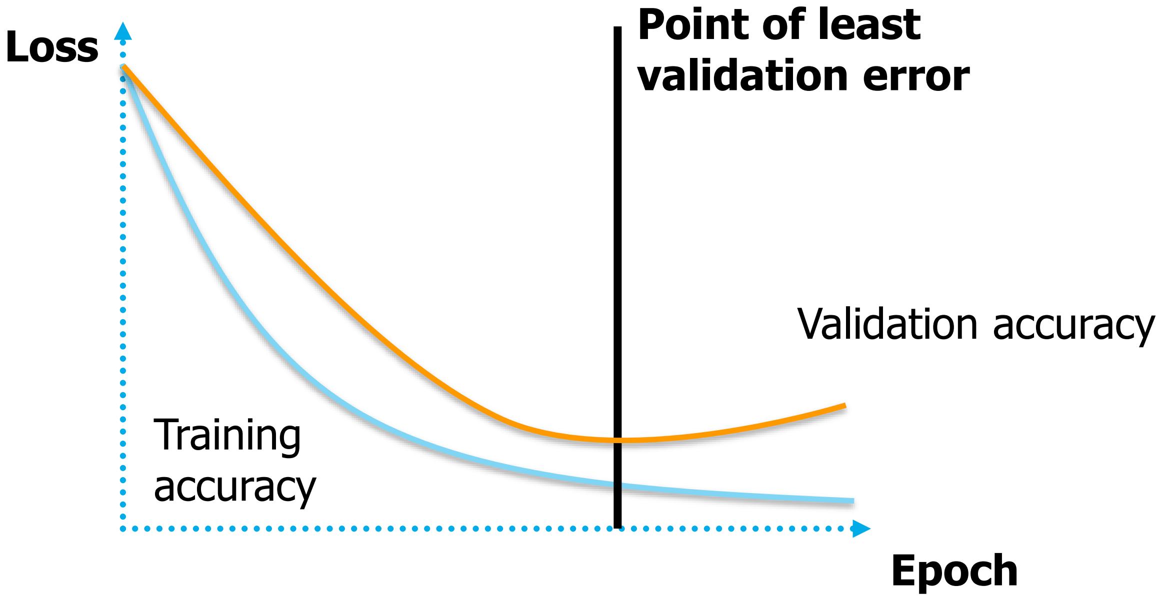
Neural Network Regularization – Early Stopping

At some point the model overfits on the training samples, leading to an **increased validation loss**.



Neural Network Regularization – Early Stopping

Early stopping is the process of finding the point of least validation error by monitoring the validation accuracy and then exiting the training process.



Early Stopping

```
# 4. Early stopping
es = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                      min_delta=0,
                                      patience=1,
                                      mode='auto'
                                      )

# Fit model on training data (with callback)
model.fit(x_train, y_train, epochs=10, shuffle=True, callbacks=[es])
```

Neural Network Regularization – Hyperparameter Search

Training Neural Networks involves **many hyperparameters** on the optimization and the regularization size.

This makes it necessary to **perform a hyperparameter search**, to find the optimal hyperparameter configuration λ^*:

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}} \mathscr{L}(\theta_\lambda).$$

One-fold validation
Hyperparameter ranges
Search structure

One-fold validation

Larger neural networks can take days / weeks to train.

Preferring one validation fold over cross-validation speeds up the search.
Note: Use a validation set with respectable size.

Hyperparameter ranges
Search structure

One-fold validation

Hyperparameter ranges

Search on log scales when range of magnitudes is large (e.g. learning rate)

*learning_rate = 10 ** uniform(-6, 1)*

Search on original scale when range is in a single magnitude (e.g. dropout)

*dropout_rate = 0.1 * uniform(0, 10)*

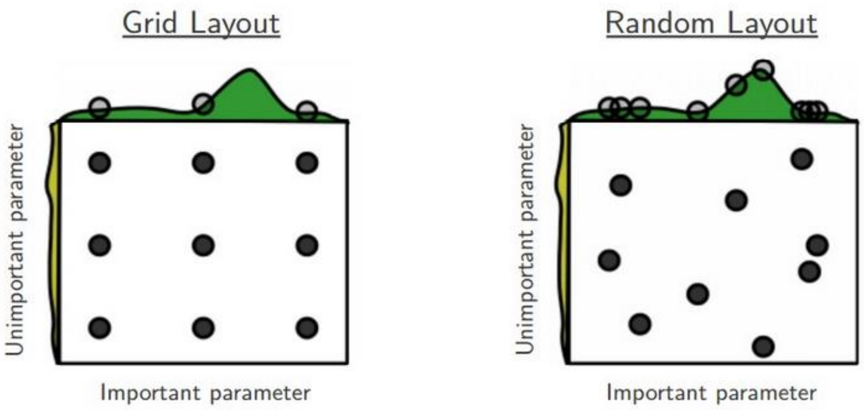
Search structure

Neural Network Regularization – Hyperparameter Search

One-fold validation
Hyperparameter ranges

Search structure

Randomly chosen trials are more efficient to cover the whole hyperparameter space.



http://cs231n.github.io/neural-networks-3/#baby

Note: Expert driven hyperparameter selection is an art and a profession.

Check initialization

Make sure the observed loss is what you expect it to be for a random prediction.

E.g. for MNIST (10 classes) we expect a probability of 0.1 for each class. With Softmax objective function the expected loss is $-\log(0.1) = 2.303$

```
32/60000 [............................] - ETA: 10:17 - loss: 2.3875 - acc: 0.0938
```

Regularization check

Overfit on a small dataset

Neural Network Training – Sanity checks

Check initialization

Regularization check

Increasing the regularization strength should increase the training loss. Such as L2 parameter constrain.

```
32/60000 [..............................] - ETA: 14:24 - loss: 8.7382 - acc: 0.1250
```

Overfit on a small dataset

Check initialization
Regularization check

Overfit on a small dataset
Train your model on a few samples (e.g. 50), you should reach a training loss of zero. Do not use regularization methods during this test.

```
model.fit(x_train[0:50], y_train[0:50], epochs=1000, shuffle=True)
```

```
Epoch 1000/1000

32/50 [==================>...........] - ETA: 0s - loss: 1.5039e-05 - acc: 1.0000
50/50 [==============================] - 0s 350us/step - loss: 1.6422e-05 - acc: 1.0000
```

References

[19] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[20] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[21] John L. Hennessy and David A. Patterson. *Computer Organization and Design (2Nd Ed.): The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[23] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[24] David Kriesel. A brief introduction on neural networks, 2015.

[25] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Thanks for your time
Questions?

Contact!

mark.schutera@kit.edu