Digitale Bildverarbeitung und Mustererkennung

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.

Rough Outline

Deep Learning Foundations

Transfer Learning and Object Detection

Segmentation Networks

Deep Reinforcement Learning

Generative Adversarial Networks

Course overview

Approaching deep learning concepts from a practitioners perspective.

Providing a **theoretical background**

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

This lecture in one slide

Introduction and motivation for deep learning

History

General concepts of data science

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

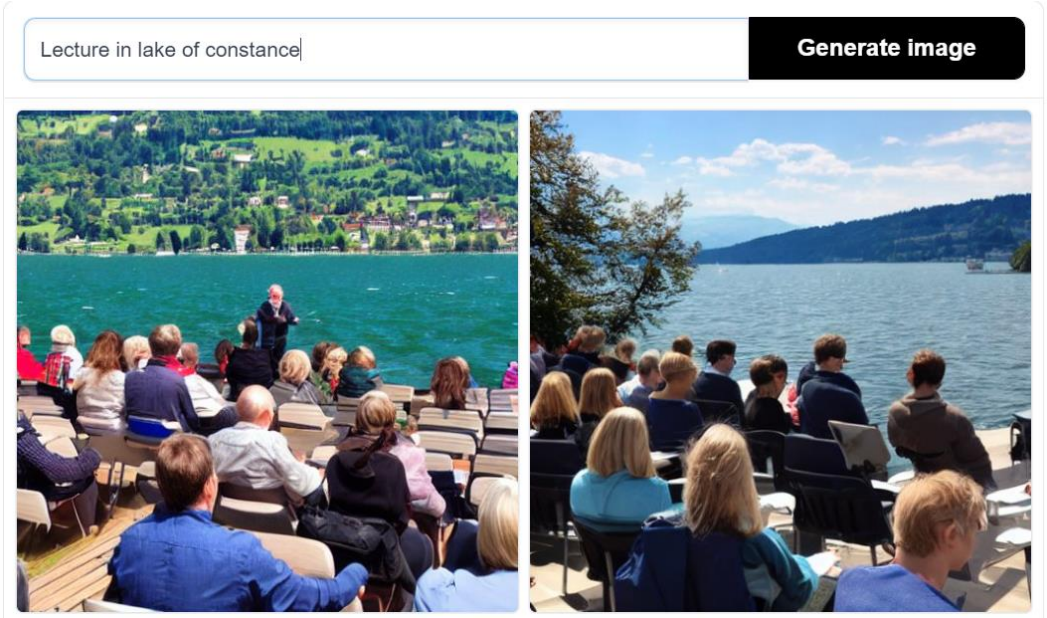
Historical development

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
2022	Stable Diffusion

Historical development

2022 Stable Diffusion

https://huggingface.co/spaces/stabilityai/stable-diffusion

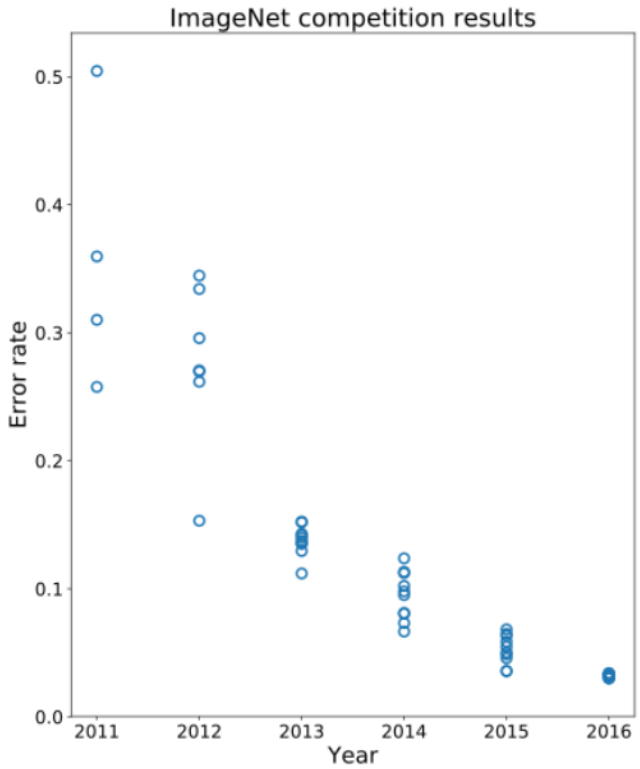


Deep learning – A machine learning revolution?

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

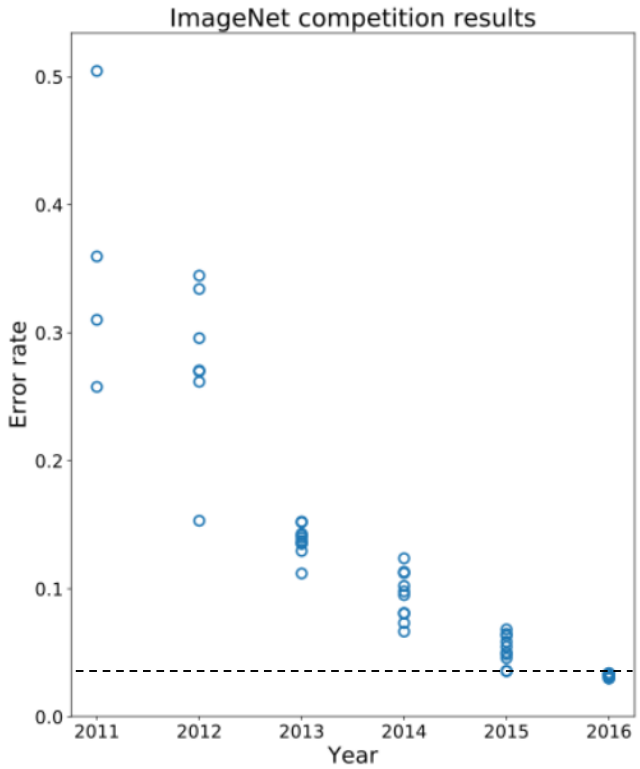
Deep learning – A machine learning revolution?

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?

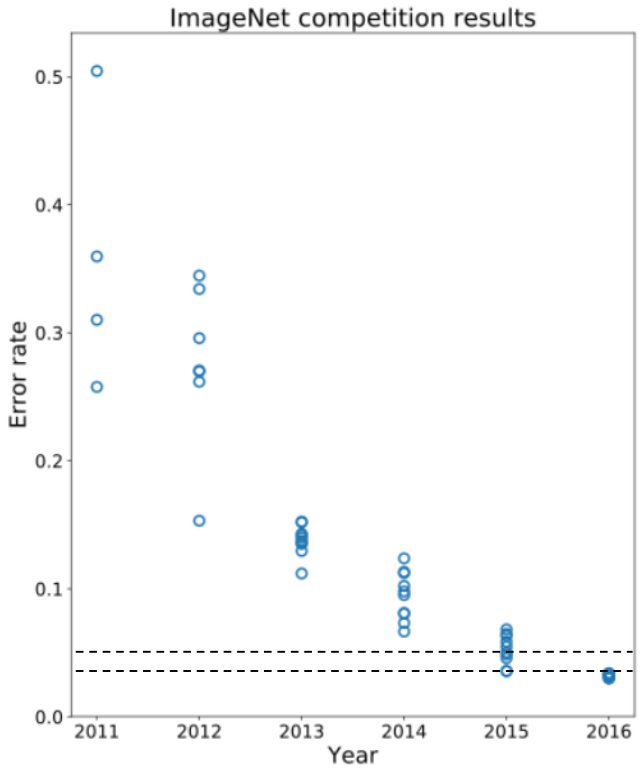
Deep Learning does not come from the void.

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

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

CityScapes

Wikipedia dataset

Software and Infrastructure

Backing by large companies

Leaps in Research

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

http://cocodataset.org/#home

Why now? – Deep Learning is software driven.

High Performance Parallel Computing

Available large Datasets

Software and Infrastructure

TensorFlow

Caffe

Pytorch

Backing by large companies

Leaps in Research

https://www.tensorflow.org/

Why now? – Deep Learning is investment driven.

High Performance Parallel Computing

Available large Datasets

Software and Infrastructure

Backing by large companies and industries

Google	ZF
Meta	Daimler
Amazon	VW
Uber	Bosch
NVIDIA	Tesla

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

Transformers

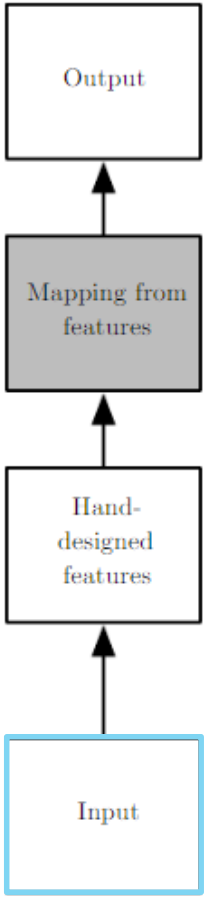
General concepts of pattern recognition

Classic machine learning

Input
Data acquisition
Data annotation
Data preprocessing
Data augmentation

Hand-designed features

Mapping from features
Output



Apple: Label 0


Pear: Label 1

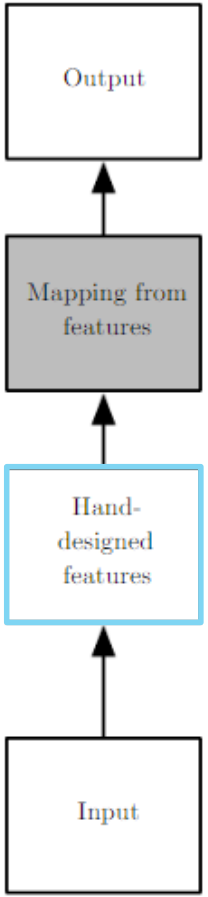

General concepts of pattern recognition

Classic machine learning

Input

Hand-designed features
Domain knowledge
Trial and error
Feature selection methods
Feature set selection methods

Mapping from features
Output



Color:
Red vs. Green

Shape:
Round vs. Obconic

General concepts of pattern recognition

Classic machine learning

Input

Hand-designed features

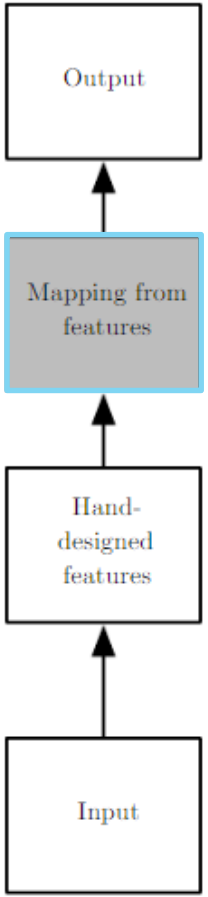
Mapping from features

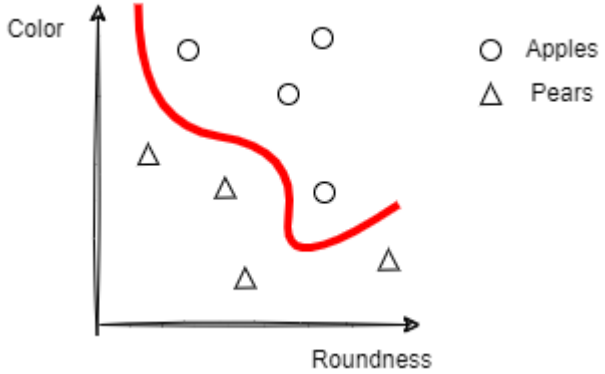
Learn pattern from data:

- Support vector machines
- Random forests

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

Output





General concept of data science

Classic machine learning

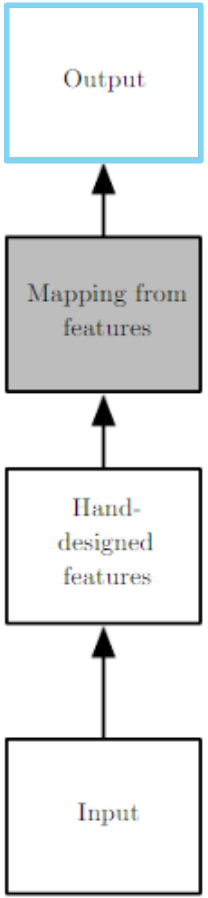
Input

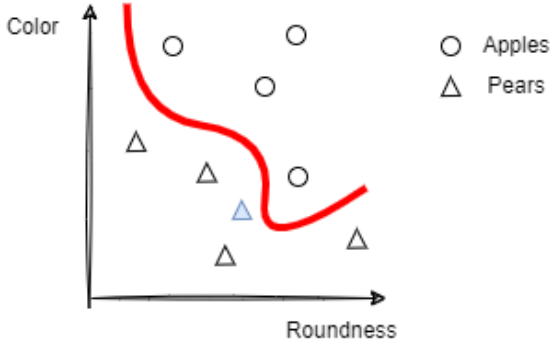
Hand-designed features

Mapping from features

Output

Inference step on data not seen during training





Learning pattern through optimization

Parameters of a model are iteratively updated with respect to loss L 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

training set
validation set
test set

Verifying generalization

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

Dataset split

Less training data

…higher variance in parameter estimates

Less validation and test data

…higher variance in performance estimate

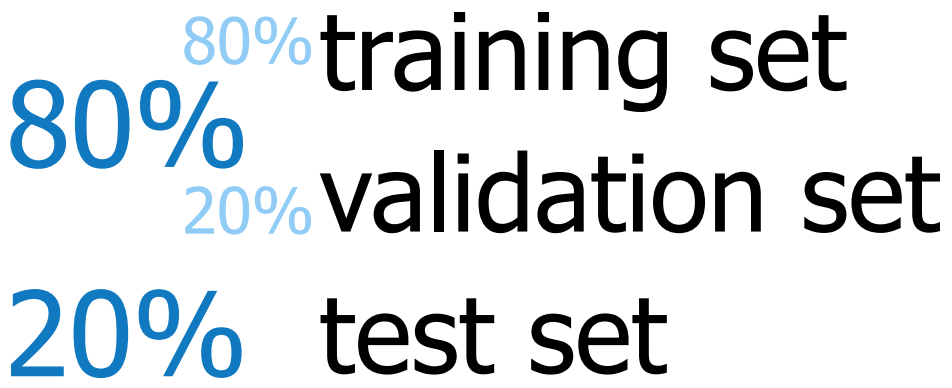
Highly dependent on dataset and task

Dataset split

Less training data
.. higher variance in parameter estimates

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

Cross validate if possible.



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/

https://www.tensorflow.org/

TensorFlow

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

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
TensorRT	Optimization module for inference

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.

This lecture in one slide

Introduction and motivation for deep learning

Neural network conception

Architecture

Back propagation

Activation functions

Objective functions

Metrics

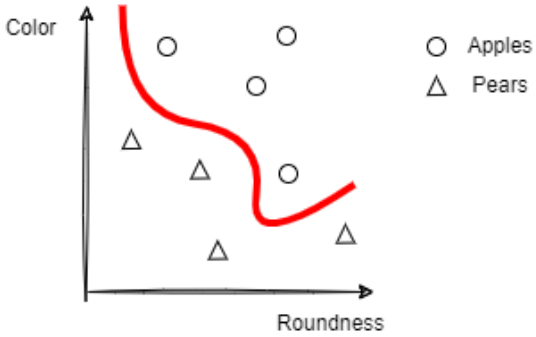
Optimization

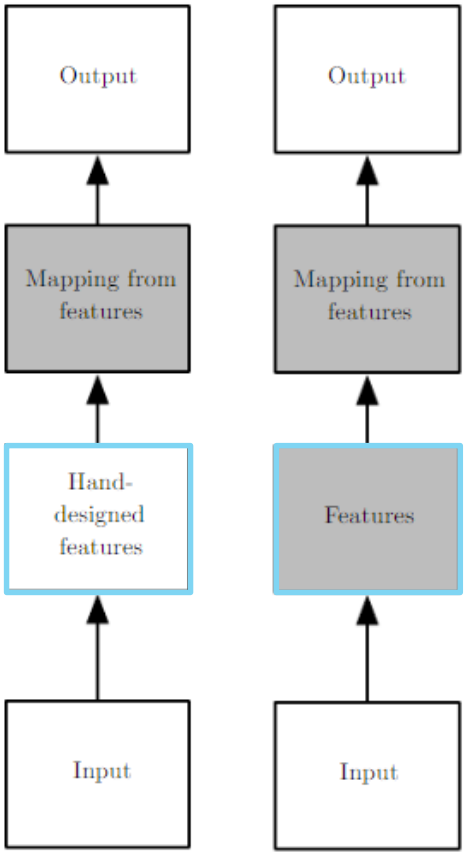
Regularization

Classic machine learning to deep learning

Classic machine learning

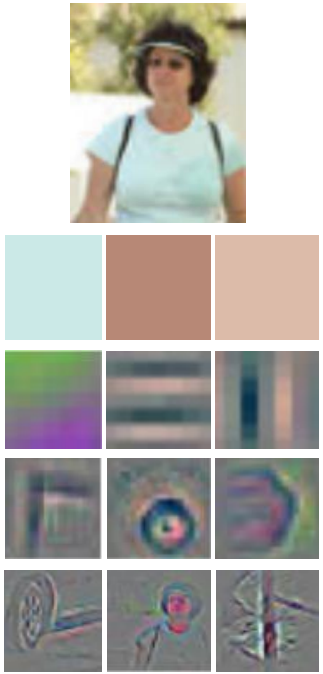
Hand-designed features





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 Networks Introduction

In the next slides we will learn how a neural network works and is trained.

There will be math, but…
you should be able to follow with highschool maths.

Neural Network Unit

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

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

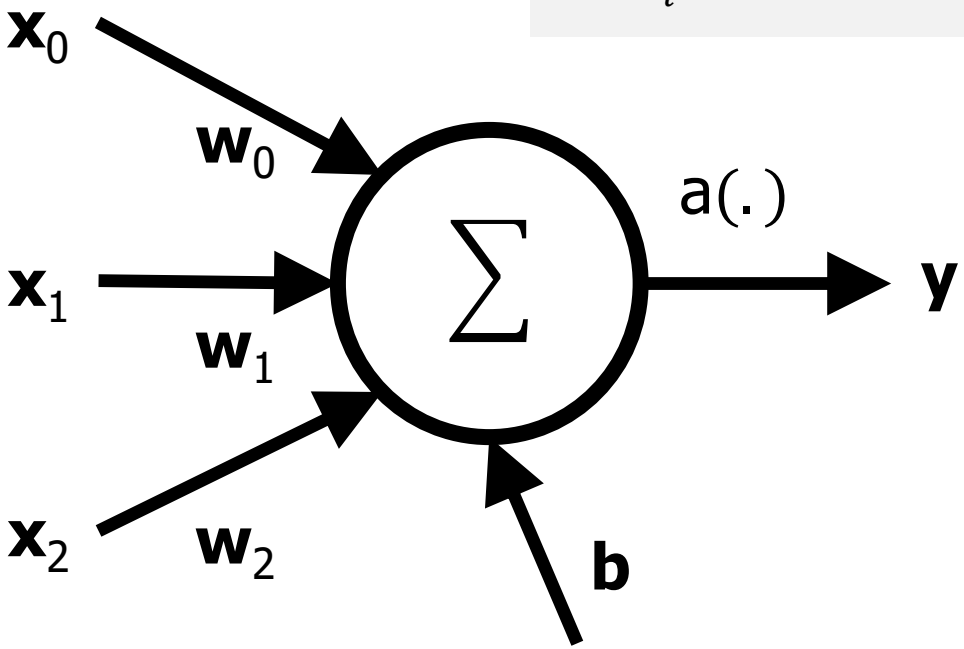
$\mathbf{x}_i$ Inputs

$\mathbf{w}_i$ Weights

$\mathbf{b}$ Biases

$a(.)$ Activation function

$\mathbf{y}$ Output

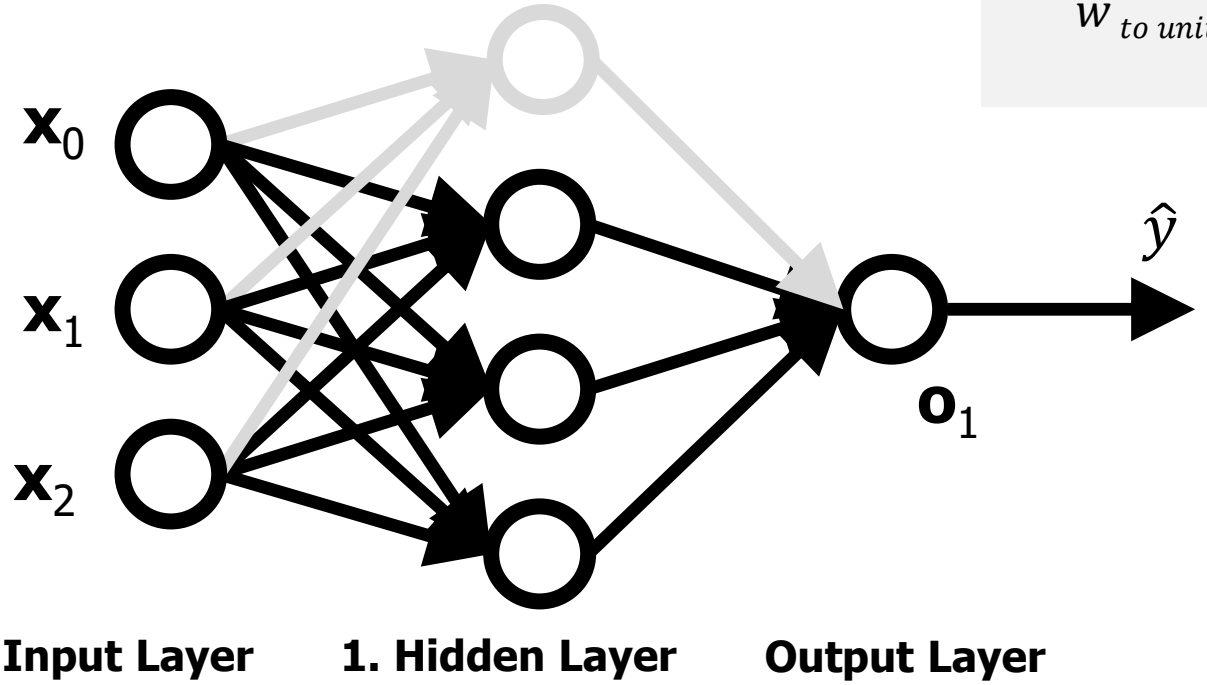


Neural Network Layers

Neural networks are **layer-wise organized**

wji

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



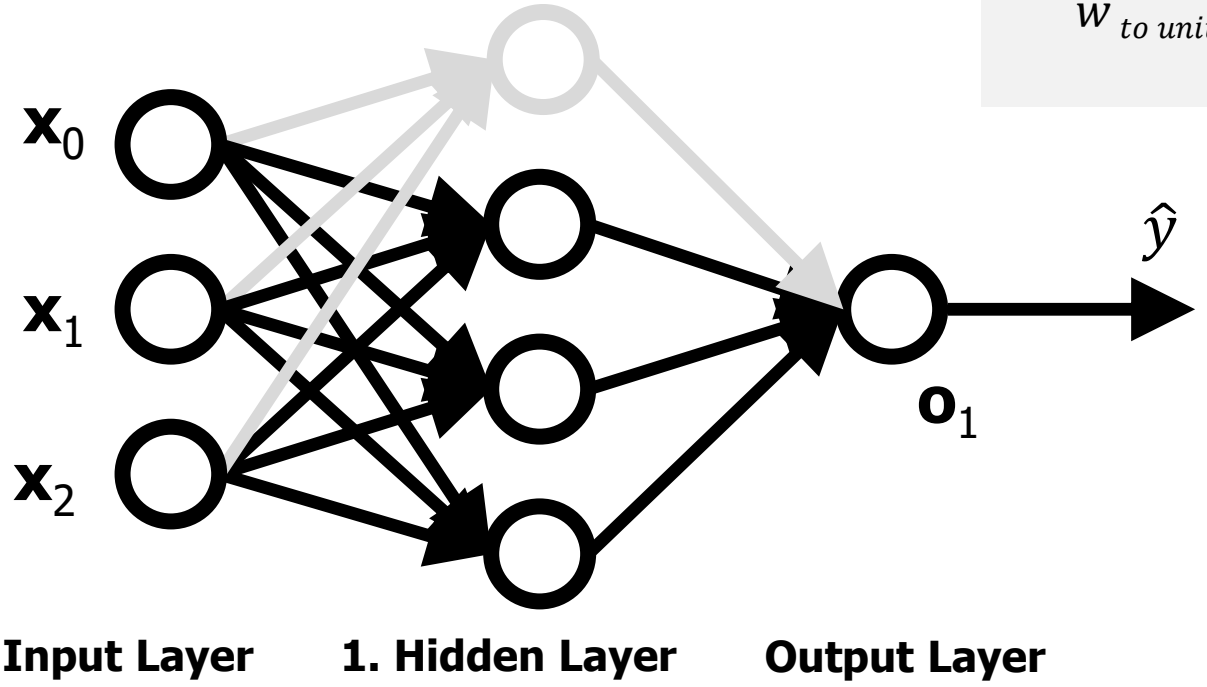
Neural Network Layers

Neural networks are layer-wise organized

$$wji$$

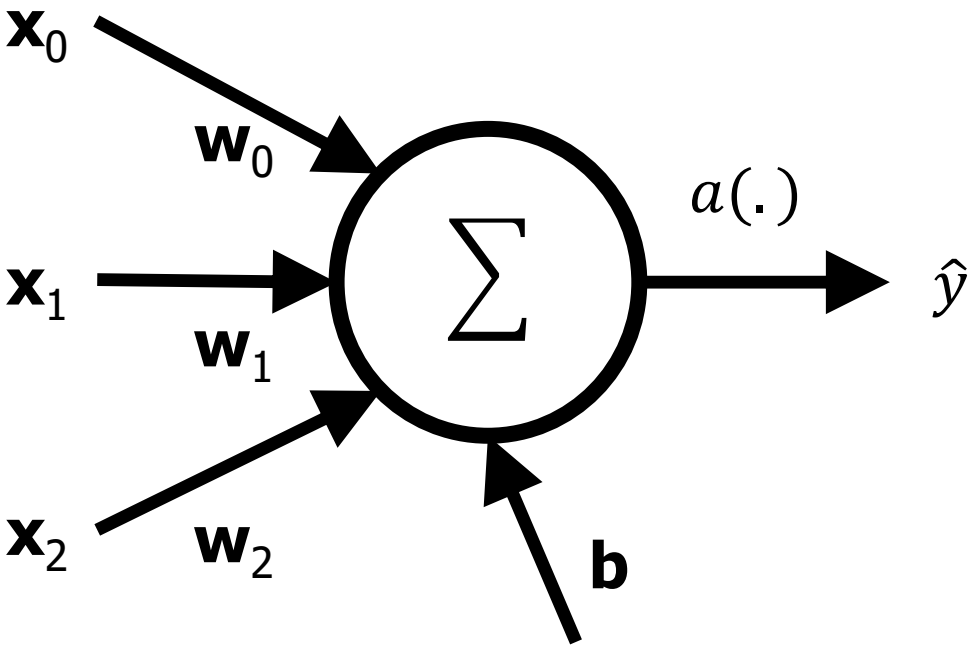
$$w_{\,to\ unit\ from\ unit}$$

Fully connected single layer neural network



Neural Network Forwardpass

Repeated matrix multiplication within an activation function



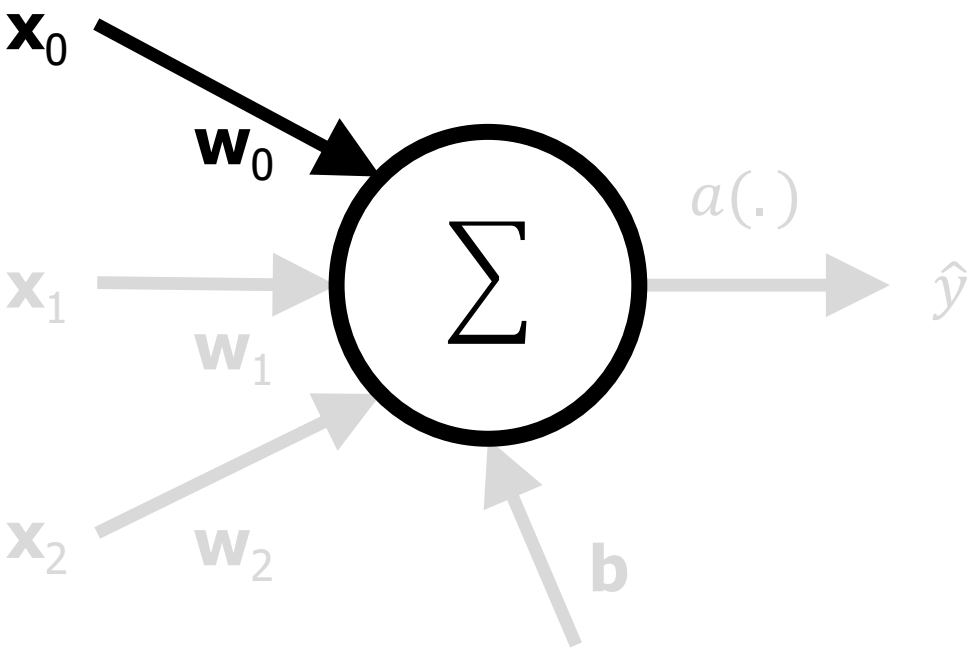
$$\hat{y} = a(\sum_i w_i x_i + b)$$

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

$$= a(\underbrace{w_0 x_0 + w_1 x_1 + w_2 x_2 + b}_{s(\boldsymbol{x})})$$

Neural Network Forwardpass

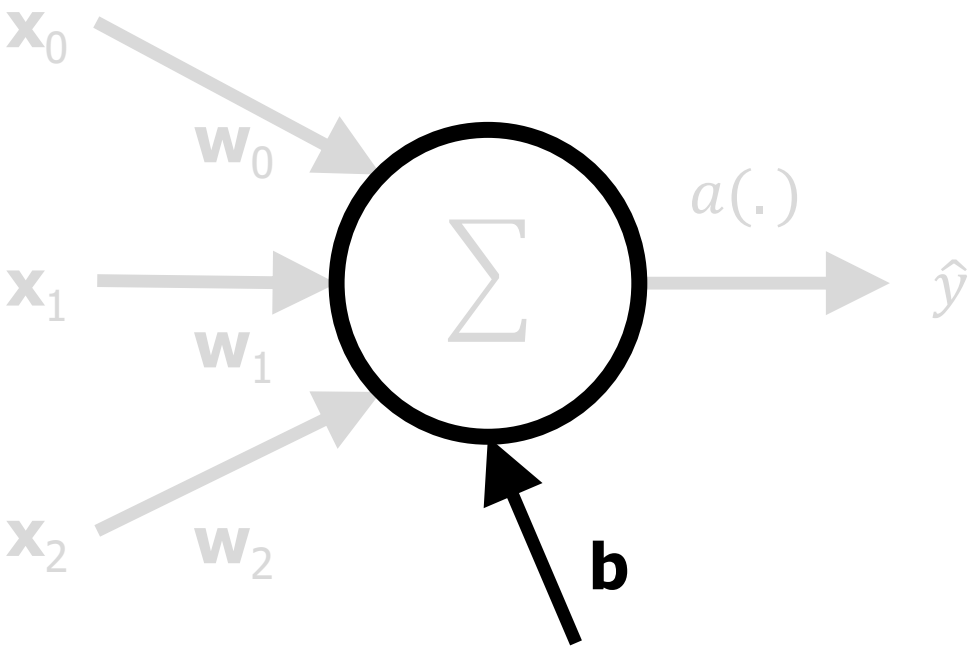
Repeated matrix multiplication within an activation function



$$\hat{y} = a(w_0 x_0 + w_1 x_1 + w_2 x_2 + b)$$

Neural Network Forwardpass

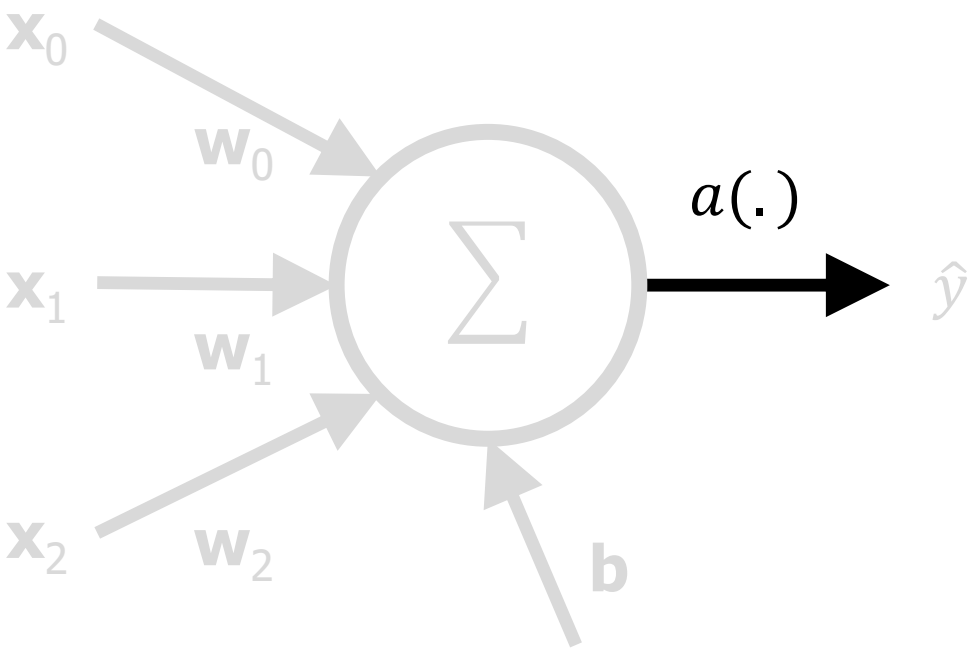
Repeated matrix multiplication within an activation function



$$\hat{y} = a(w_0 x_0 + w_1 x_1 + w_2 x_2 + b)$$

Neural Network Forwardpass

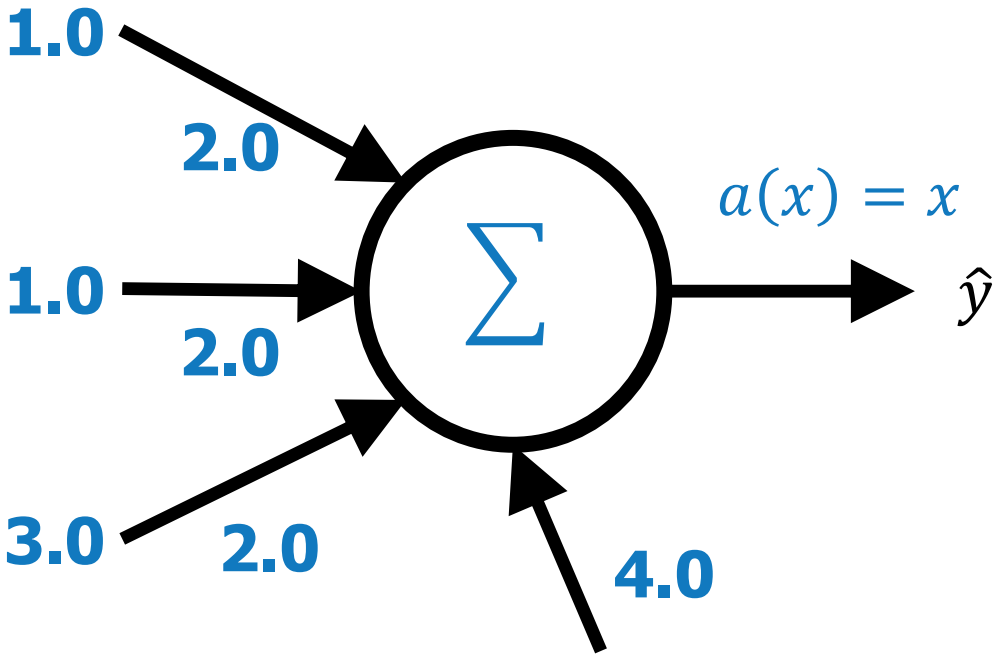
Repeated matrix multiplication within an activation function



$$\hat{y} = a(w_0x_0 + w_1x_1 + w_2x_2 + b)$$

Neural Network Forwardpass

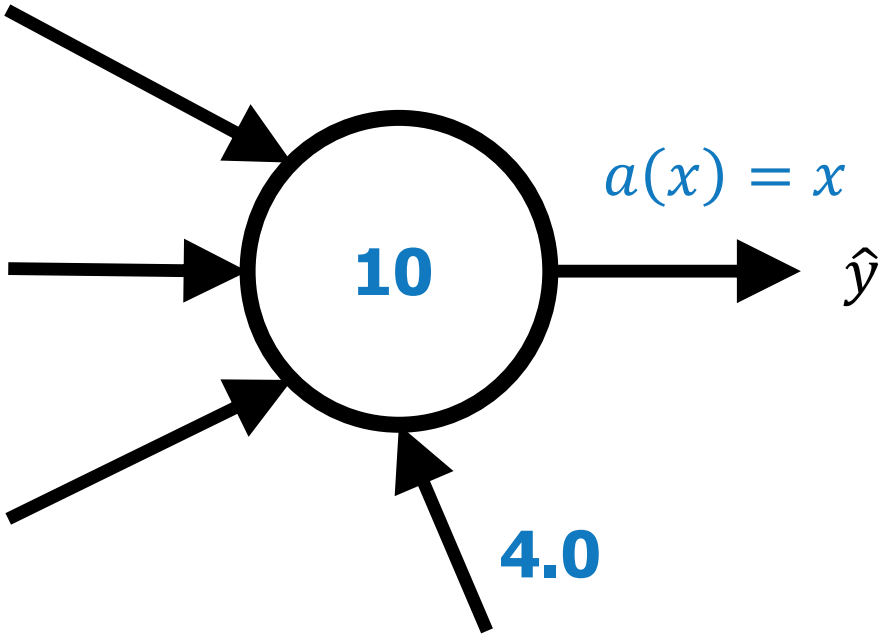
Example with input [1.0, 1.0, 3.0] and weights [2.0, 2.0, 2.0] and bias [4.0] and linear function as activation function



$$\hat{y} = a(2.0 + 2.0 + 6.0 + 4.0)$$

Neural Network Forwardpass

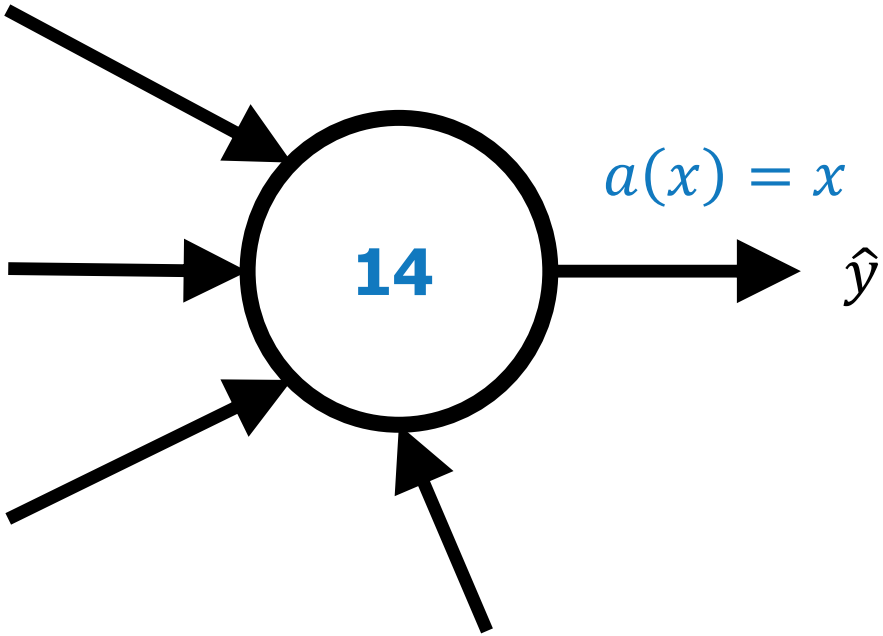
Example with input [1.0, 1.0, 3.0] and weights [2.0, 2.0, 2.0] and bias [4.0] and linear function as activation function



$$\hat{y} = a(10.0 + 4.0)$$

Neural Network Forwardpass

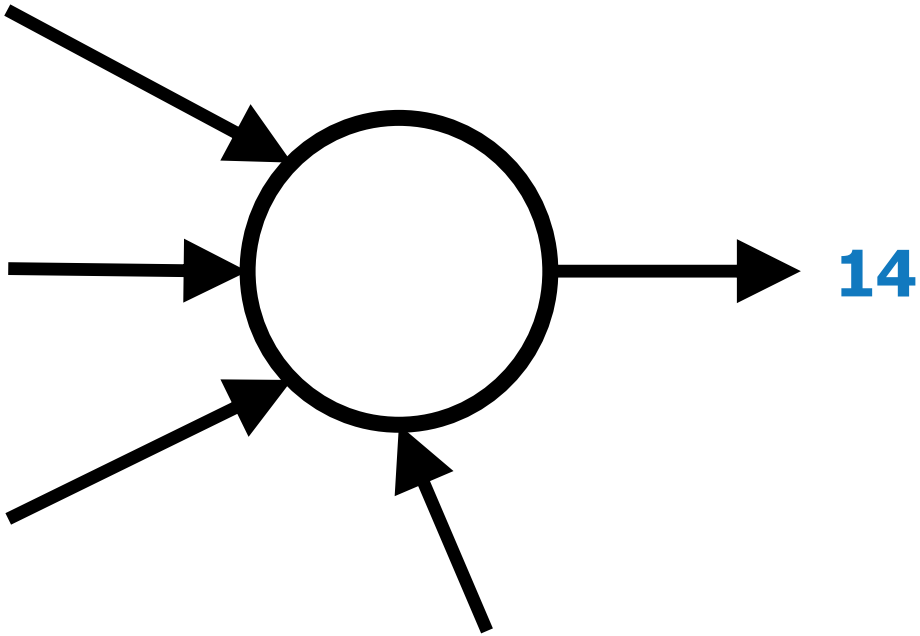
Example with input [1.0, 1.0, 3.0] and weights [2.0, 2.0, 2.0] and bias [4.0] and linear function as activation function



$$\hat{y} = f(14.0)$$

Neural Network Forwardpass

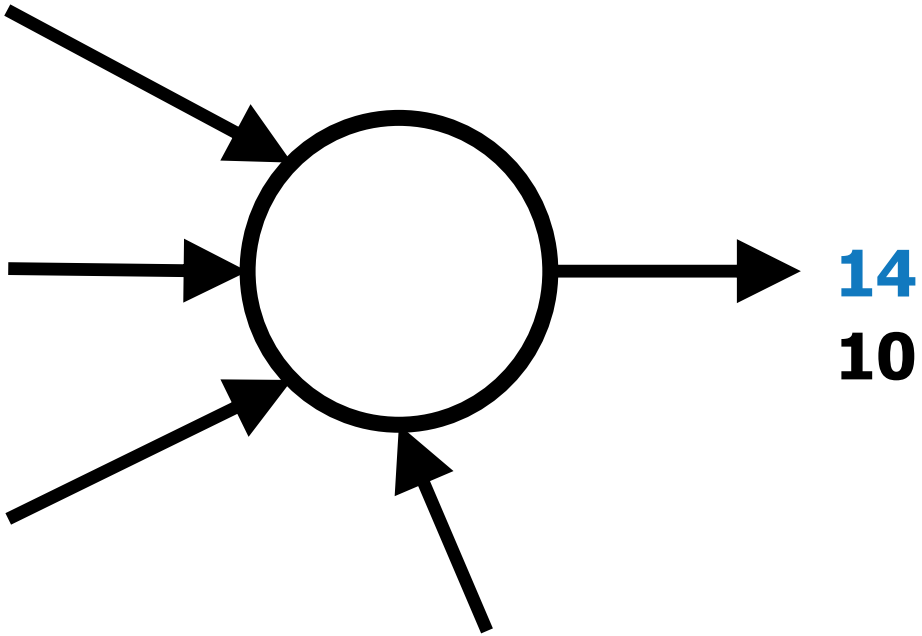
Example with input [1.0, 1.0, 3.0] and weights [2.0, 2.0, 2.0] and bias [4.0] and linear function as activation function



$$\hat{y} = 14.0$$

Neural Network Supervised Learning

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



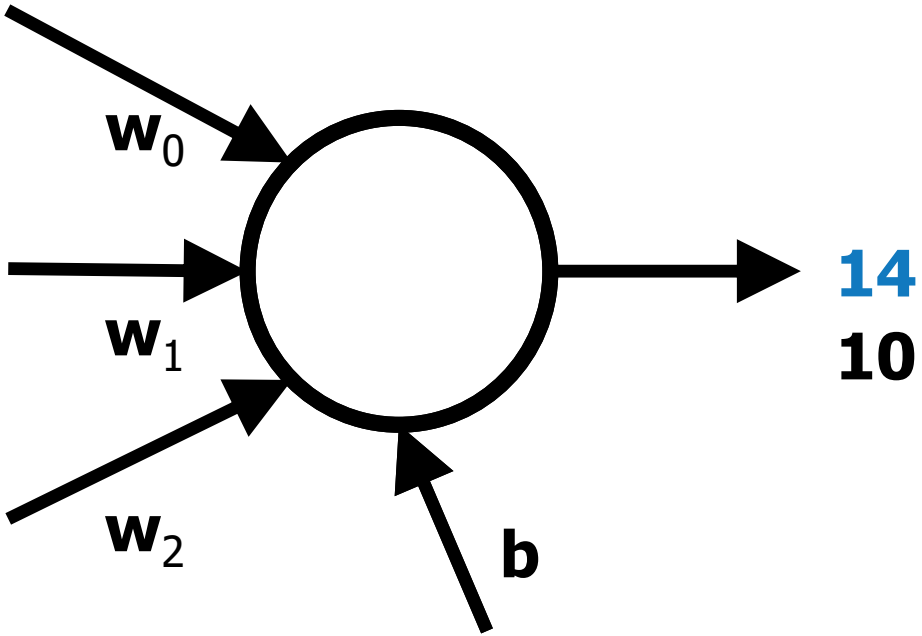
$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

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**.

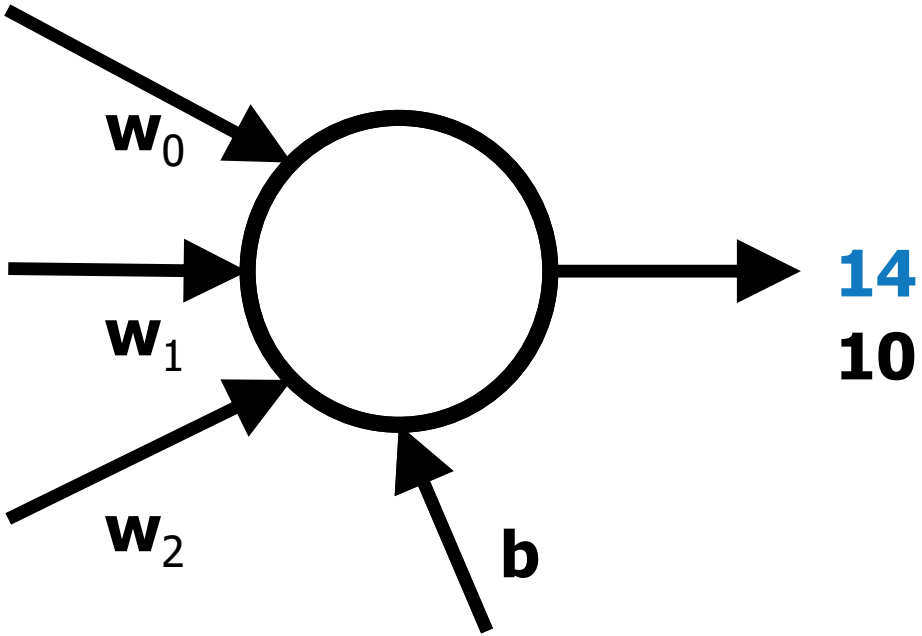


$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

Neural Network Back Propagation

Let's plug the error into a loss.



Loss Function:

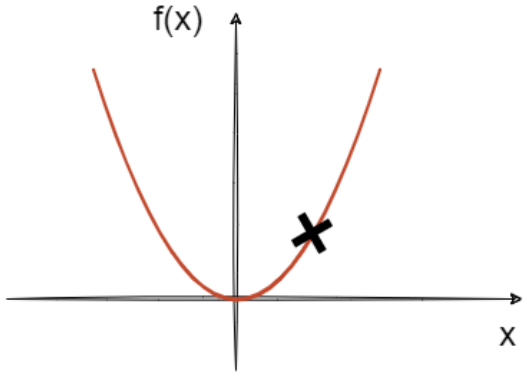
$$L = \frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - \hat{y}_i)^2$$

here $n = 1$

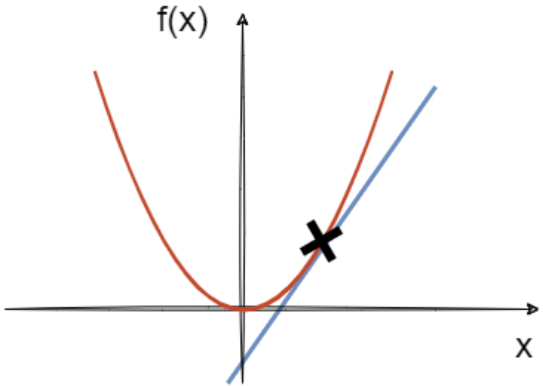
$$L = (10 - 14)^2 = 16$$

Recap: Derivatives

Derivatives tell us the effect of an infinitesimal change of an input x to the function output f(x).



$$f(x) = x^2$$
$$f'(x) = 2x$$



$$x = 1$$
$$f(1) = 1$$
$$f'(1) = 2$$

If we want to minimize that function we want to go into the direction of the negative gradient!
The gradient itself gives us the "steepness" and, hence, the magnitude of a sensible parameter update.

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.

First possibility:
Combine all functions and calculate partial derivatives.

$$L(w_0, w_1, w_2) = (10 - (1w_0 + 1w_1 + 3w_2 + b))^2$$

$$\frac{\partial L}{\partial w_0} \qquad \frac{\partial L}{\partial w_1} \qquad \frac{\partial L}{\partial w_2} \qquad \frac{\partial L}{\partial b}$$

Drawbacks:
- Gets complex quite quickly.
- Only seems reasonable in our easy case. A neural network consists of many layers!

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.

Second possibility:
Make use of the chain rule and calculate partial derivatives.

$$s = w_0 x_0 + w_1 x_1 + w_2 x_2 + b \qquad \hat{y}_i = a(s) = s \qquad \mathrm{L} = \frac{1}{n} \sum_{i=1}^{n} (\tilde{y}_i - \hat{y}_i)^2$$

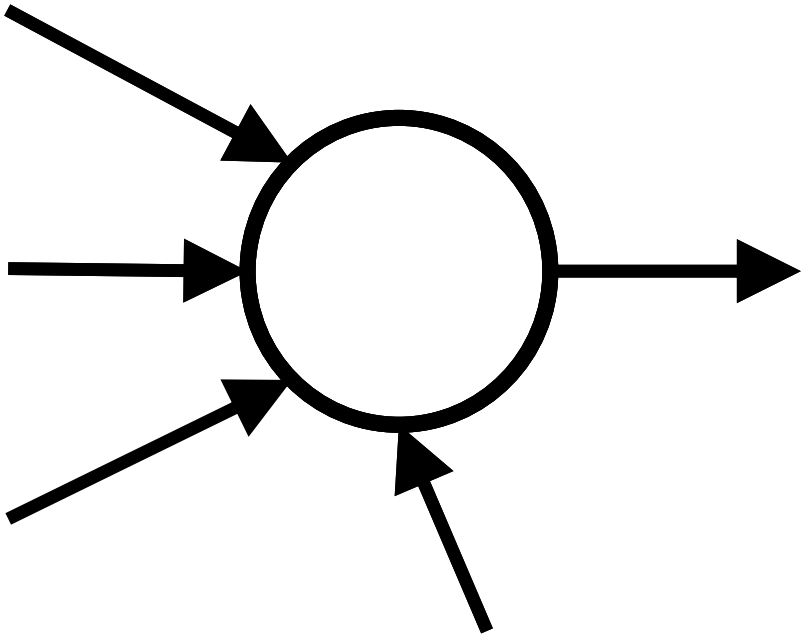
$$\frac{\partial L}{\partial w_0} = \frac{\partial s}{\partial w_0} \cdot \frac{\partial \hat{y}_i}{\partial s} \cdot \frac{\partial L}{\partial \hat{y}_i}$$

$$\frac{\partial L}{\partial w_0} = x_0 \cdot 1 \cdot 2 \cdot \sum_{i=1}^{n} (\tilde{y}_i - \hat{y}_i)^2$$

similar for $\frac{\partial L}{\partial w_1}$, $\frac{\partial L}{\partial w_2}$, $\frac{\partial L}{\partial b}$

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.



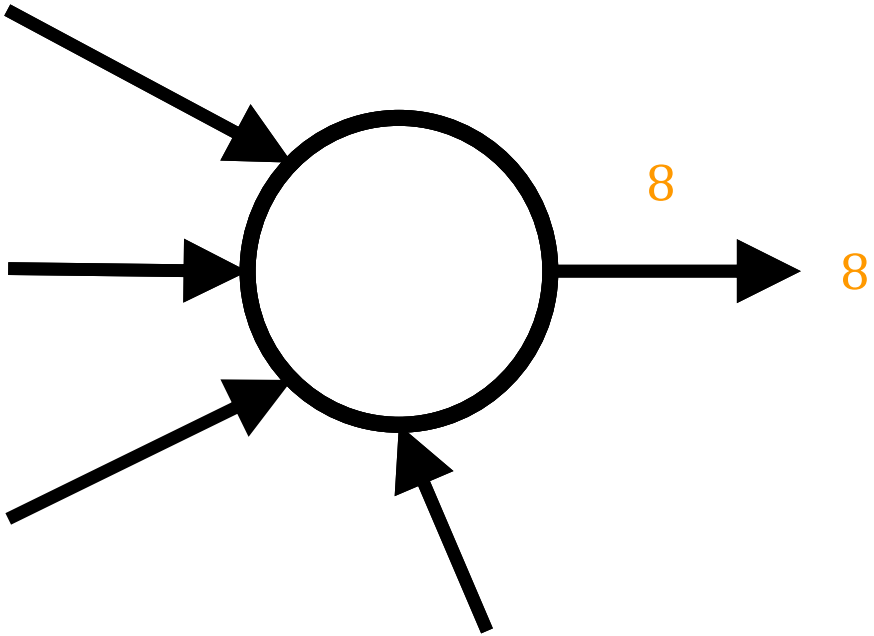
$$\frac{\partial L}{\partial \hat{y}} = -2 * (\tilde{y} - \hat{y})$$

$$= -2 * (10 - 14)$$

$$= 8$$

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.

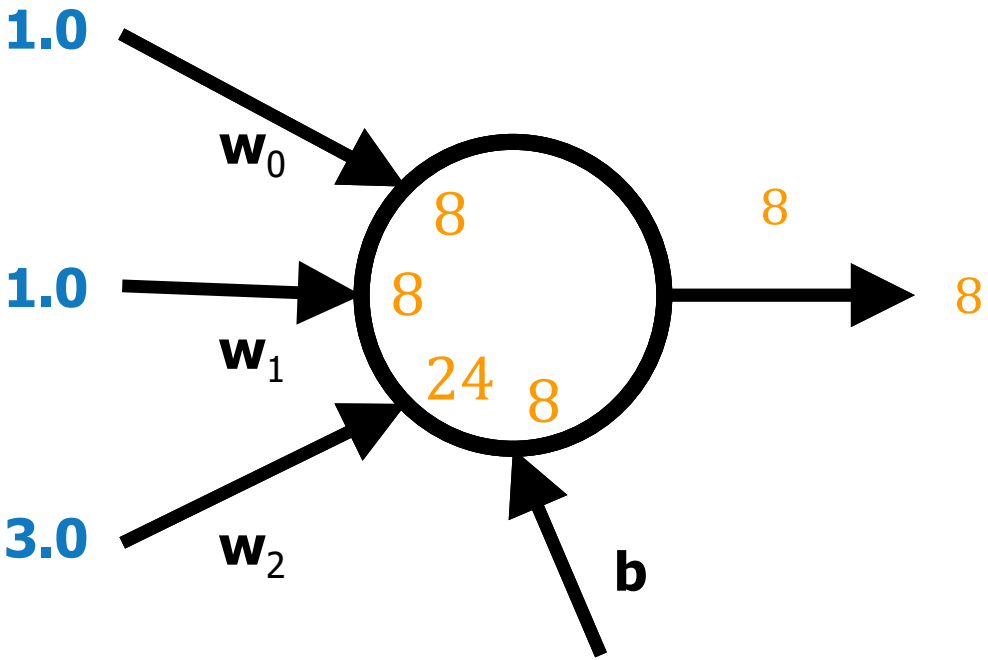


$$\frac{\partial a(s)}{\partial s} = 1$$

$$\frac{\partial L}{\partial s} = 1 * 8 = 8$$

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.



$$\frac{\partial s}{\partial w_0} = \frac{\partial(w_0 x_0 + w_1 x_1 + w_2 x_2 + b)}{\partial(w_0)} = x_0$$

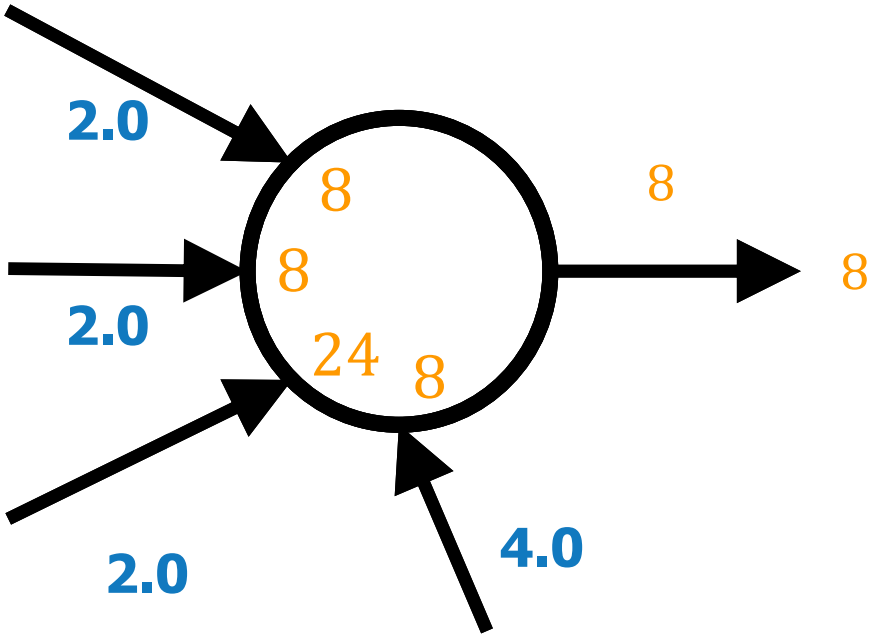
$$\frac{\partial L}{\partial w_0} = 1 * 1 * 8 = 8 \qquad \frac{\partial L}{\partial w_1} = 8$$

$$\frac{\partial L}{\partial w_2} = 24 \qquad \frac{\partial L}{\partial b} = 8$$

The gradients tell you about the parameters' influence

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.



$$\frac{\partial s}{\partial w_0} = \frac{\partial(w_0 x_0 + w_1 x_1 + w_2 x_2 + b)}{\partial(w_0)} = x_0$$

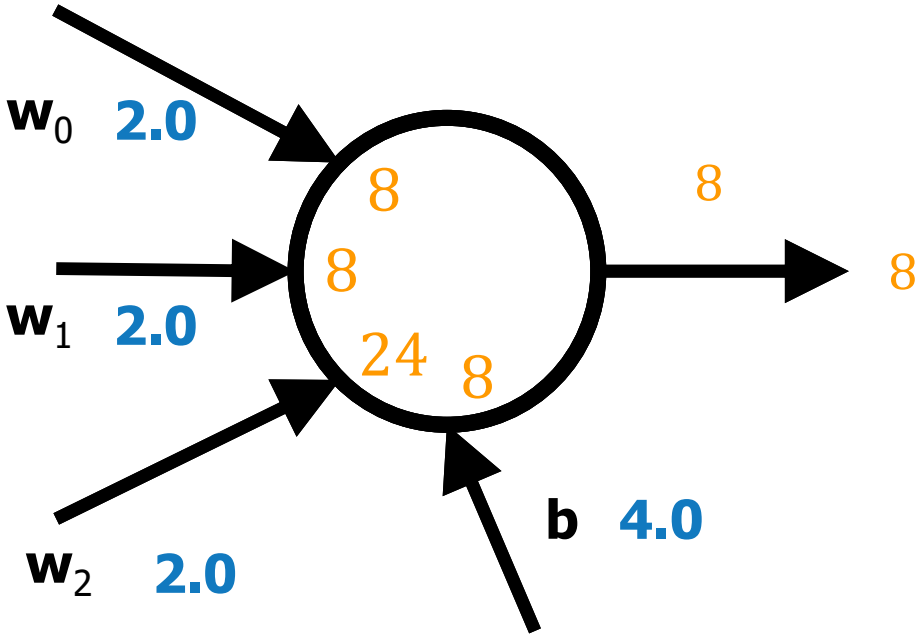
$$\frac{\partial L}{\partial w_0} = 1 * 1 * 8 = 8 \qquad \frac{\partial L}{\partial w_1} = 8$$

$$\frac{\partial L}{\partial w_2} = 24 \qquad \frac{\partial L}{\partial b} = 8$$

The gradients tell you about the parameters' influence

Neural Network Back Propagation

Now we can do a feature update. A very important parameter for a neural network is the learning rate l. It defines the magnitude of the update.



$$lr = 0.04$$

$$w_{0,upd} = \mathrm{w}_0 - \left(\mathrm{lr} \cdot \frac{\partial L}{\partial w_0}\right) = 1.68$$

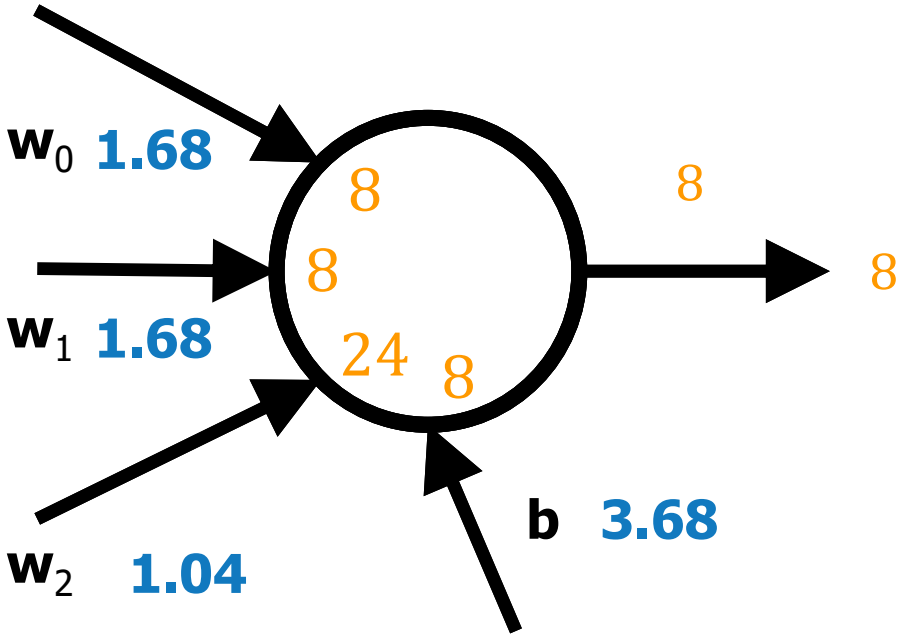
$$w_{1,upd} = 1.68$$

$$w_{2,upd} = 1.04$$

$$b_{upd} = 3.68$$

Neural Network Back Propagation

Now we can do a feature update. A very important parameter for a neural network is the learning rate l. It defines the magnitude of the update.



$$lr = 0.04$$

$$w_{0,upd} = \mathrm{w}_0 - \left(\mathrm{lr} \cdot \frac{\partial L}{\partial w_0}\right) = 1.68$$

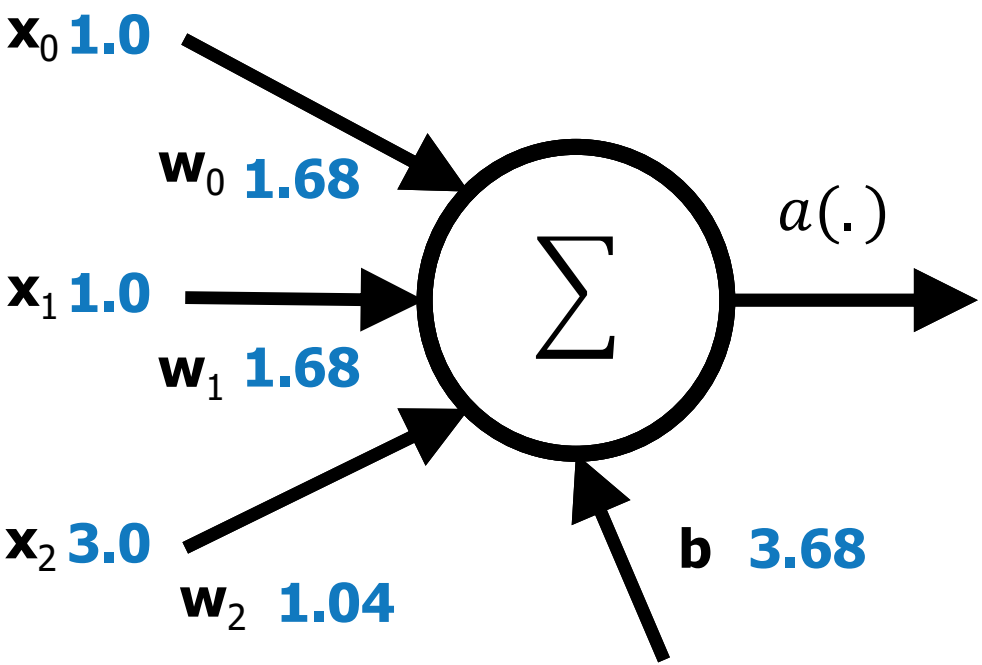
$$w_{1,upd} = 1.68$$

$$w_{2,upd} = 1.04$$

$$b_{upd} = 3.68$$

Neural Network Back Propagation

Another forward pass shows that we did a good job.



$$\hat{y}_i = a(w_0 x_0 + w_1 x_1 + w_2 x_2 + b)$$

$$\hat{y} = 10.16$$

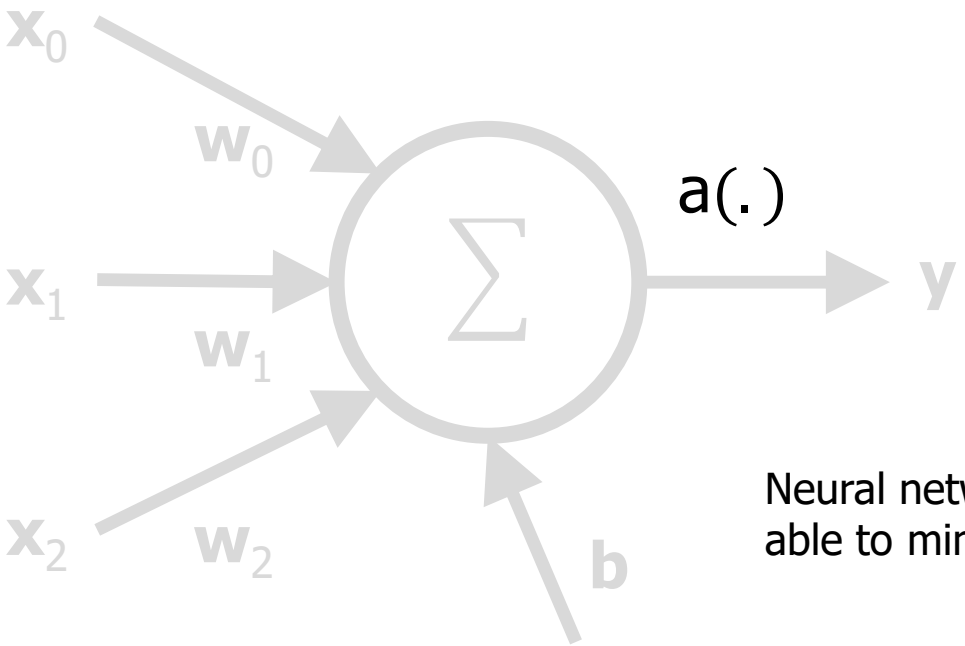
$$\tilde{y} = 10$$

$$\mathrm{L} = (10 - 10.16)^2 = 0.0256$$

The loss is significantly reduced after one optimization step.

Neural Network Activation Functions

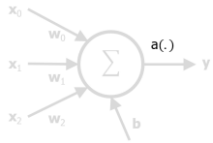
Every **activation function** $a(s)$ takes a single number s and performs a certain mathematical operation on it. s is also known as the cell state.



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

Neural networks need non-linearity to be able to mimic any function which exists.

Neural Network Activation Functions



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

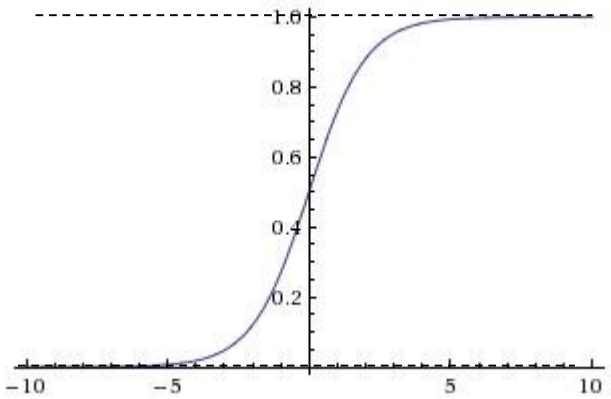
Sigmoid

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

Saturates and vanishes gradients

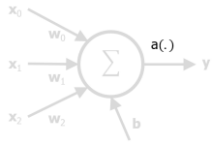
Outputs are not zero-centered

Complying derivative characteristics



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

Neural Network Activation Functions



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

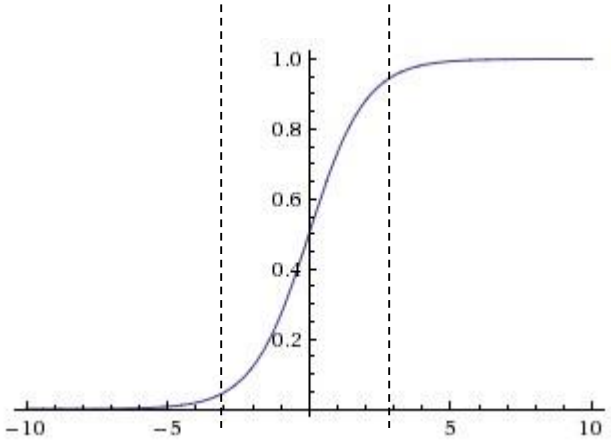
Sigmoid

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

Saturates and vanishes gradients

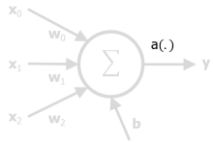
Outputs are not zero-centered

Complying derivative characteristics



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

Neural Network Activation Functions



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

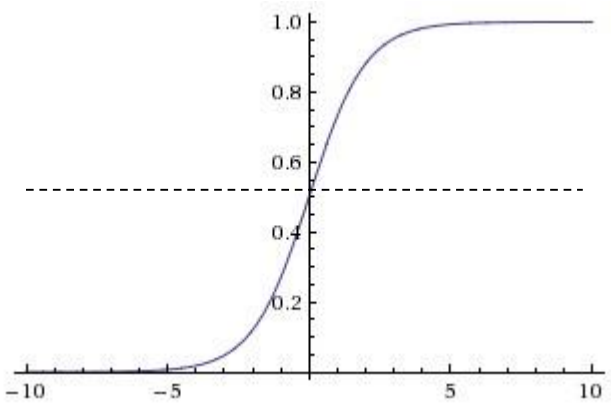
Sigmoid

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

Saturates and vanishes gradients

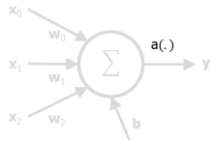
Outputs are not zero-centered

Complying derivative characteristics



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

Neural Network Activation Functions



Sigmoid

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

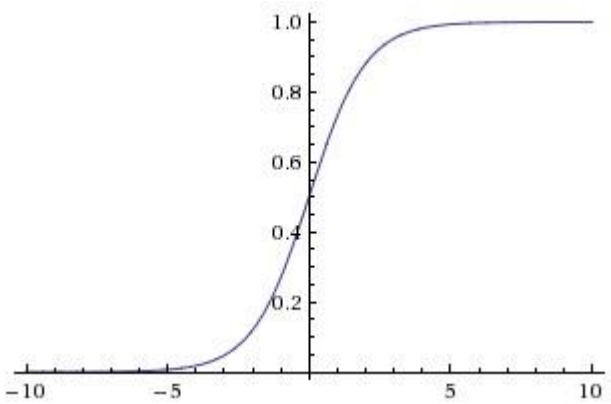
$$a'(x) = \big(1 - a(x)\big)a(x)$$

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

Saturates and vanishes gradients

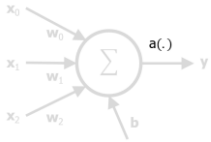
Outputs are not zero-centered

Complying derivative characteristics



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

Neural Network Activation Functions



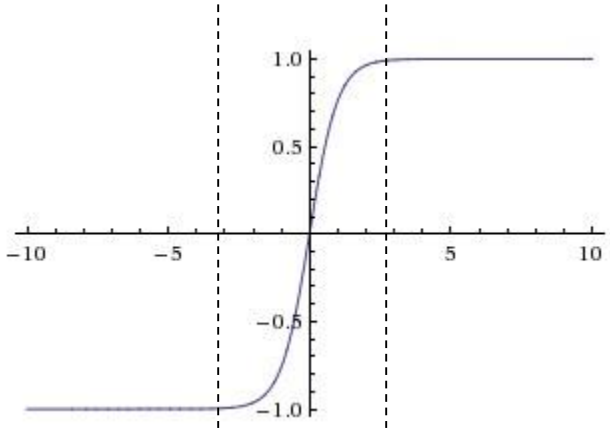
$$a(x) = 1 - \frac{1}{e^{2x} + 1}$$

Tanh

Saturates and vanishes gradients

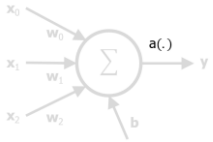
Outputs are zero-centered in a range between minus one and one

Complying derivative characteristics



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

Neural Network Activation Functions



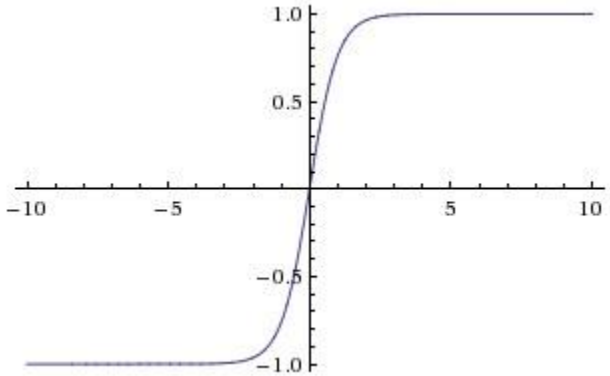
$$a(x) = 1 - \frac{1}{e^{2x} + 1}$$

Tanh

Saturates and vanishes gradients

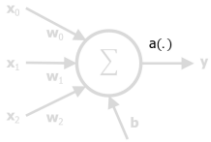
Outputs are zero-centered in a range between minus one and one

Complying derivative characteristics



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

Neural Network Activation Functions



$$a(x) = 1 - \frac{1}{e^{2x} + 1}$$

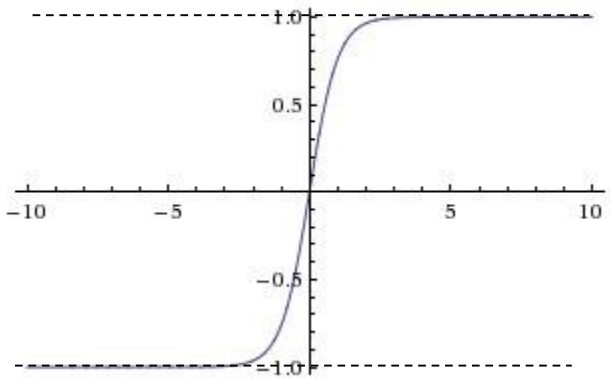
Tanh

$$a'(x) = 1 - a(x)^2$$

Saturates and vanishes gradients

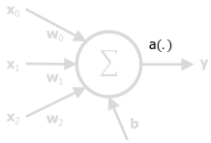
Outputs are zero-centered in a range between minus one and one

Complying derivative characteristics



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

Neural Network Activation Functions



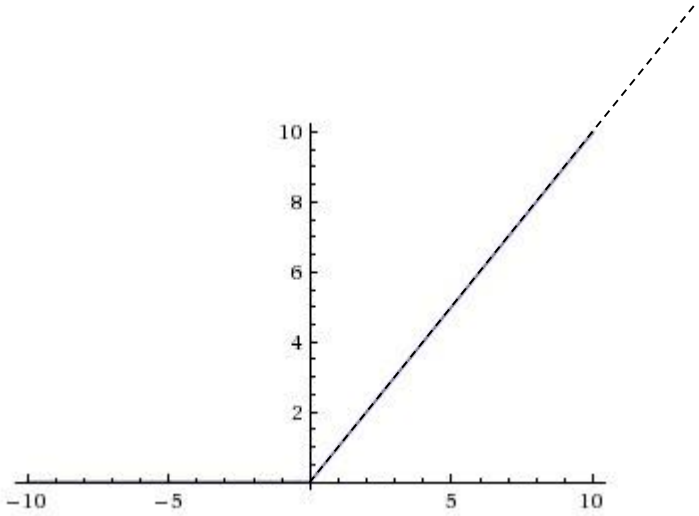
$$a(x) = \max\{0, x\}$$

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 facilitated

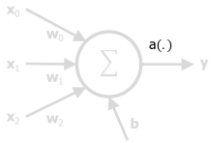
Cheap operation of thresholding at zero

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



$$a(x) = \max\{0, x\}$$

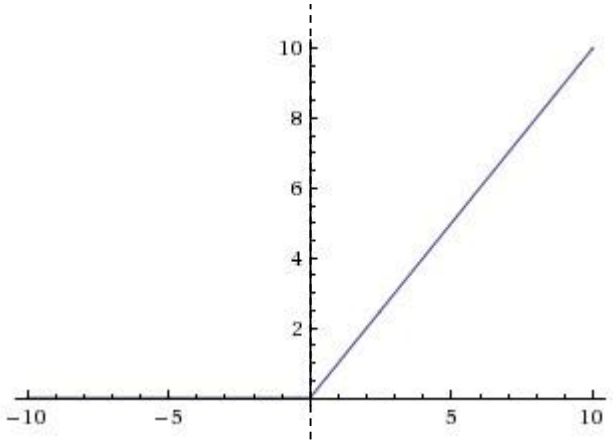
ReLU (Rectified Linear Unit)

$$a'(x) = \begin{cases} 0, & if\ x < 0 \\ 1, & if\ x > 0 \end{cases}$$

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

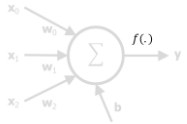
Cheap operation of thresholding at zero

ReLUs can "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



$$a(x) = \max\{0, x\}$$

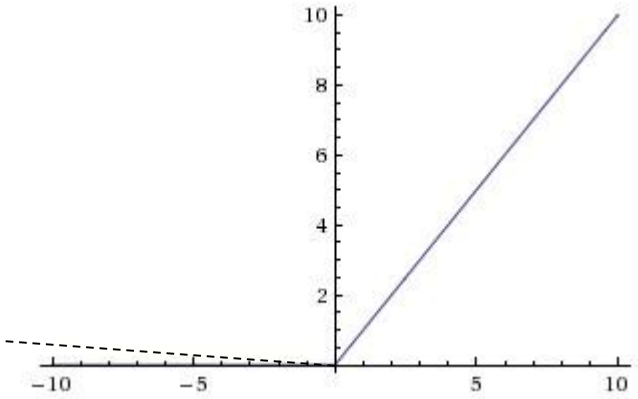
ReLU (Rectified Linear Unit)

$$a'(x) = \begin{cases} 0, & if\ x < 0 \\ 1, & if\ x > 0 \end{cases}$$

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

Cheap operation of thresholding at zero

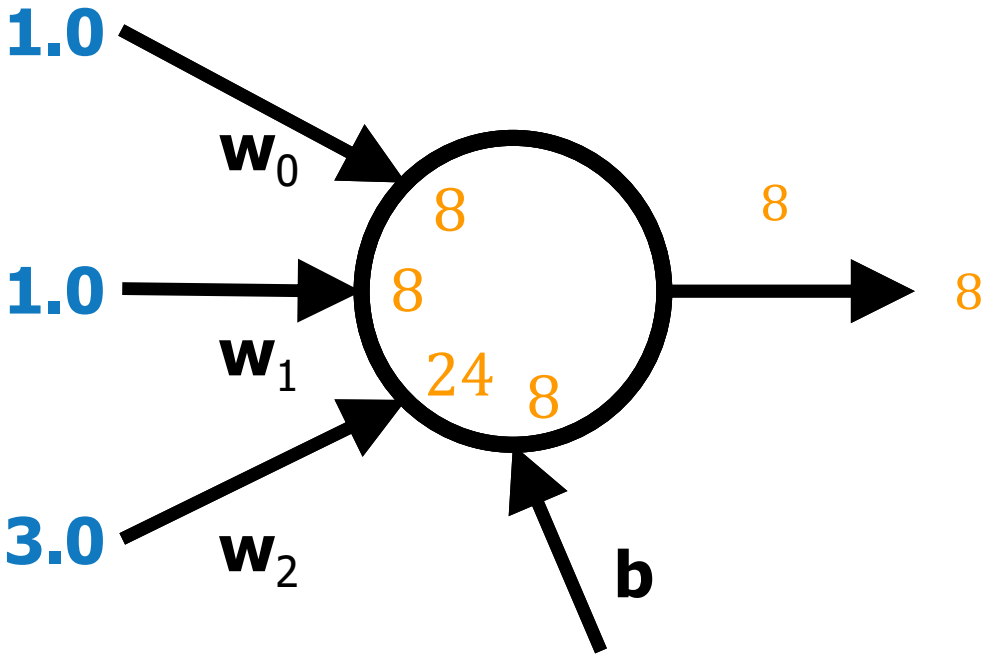
ReLUs can "die" during training when the weights are updated too far into the negative domain. Fixed by leaky ReLUs.



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



$$\frac{\partial s}{\partial w_0} = \frac{\partial(w_0 x_0 + w_1 x_1 + w_2 x_2 + b)}{\partial(w_0)} = x_0$$

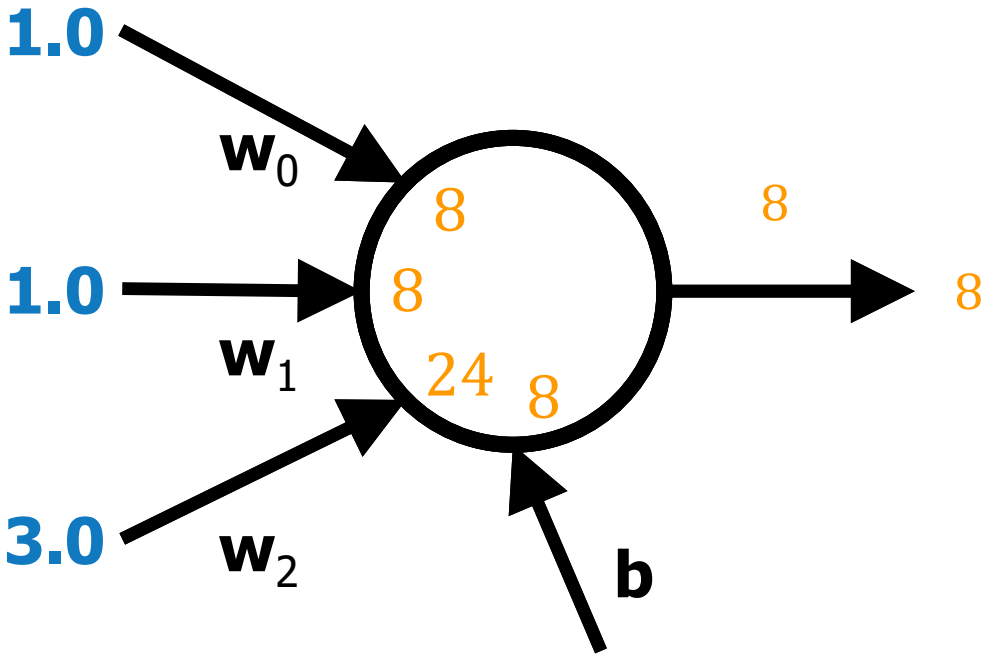
$$\frac{\partial L}{\partial w_0} = 1 \cdot 1 \cdot 8 = 8 \qquad \frac{\partial L}{\partial w_1} = 8$$

$$\frac{\partial L}{\partial w_2} = 24 \qquad \frac{\partial L}{\partial b} = 8$$

The gradients tell you about the parameters' influence

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.



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

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.

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

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 *L*.

$$\mathrm{L} = \frac{1}{\mathrm{n}}\left(\sum_{i=1}^{n} L_i\right)$$

$$\mathrm{L}(\hat{y},\ \tilde{y})$$

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 (Least Absolute Deviations)

L2 squared norm (Least Square Errors)

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

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 (Least Absolute Deviations)

$$\mathrm{L} = \frac{1}{\mathrm{n}}\left(\sum_{i=1}^{n} |\tilde{y}_i - \hat{y}_i|\right)$$

L2 squared norm (Least Square Errors)

$$\hat{y} = 14.0$$

$$\tilde{y} = 10.0$$

$$4$$

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 (Least Absolute Deviations)

L2 squared norm (Least Square Errors)

$$\mathrm{L} = \frac{1}{n}\left(\sum_{i=1}^{n}(\tilde{y}_i - \hat{y}_i)^2\right)$$

$\hat{y} = 14.0$

$\tilde{y} = 10.0$

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.

Softmax function

$$\hat{y}_i = \frac{e^{y_i}}{\sum_{i=1}^{n} e^{y_i}}$$

Cross-entropy

$$\mathrm{L} = -\frac{1}{n}\left(\sum_{i=1}^{n} \tilde{y}_i \cdot \log(\hat{y}_i)\right)$$

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

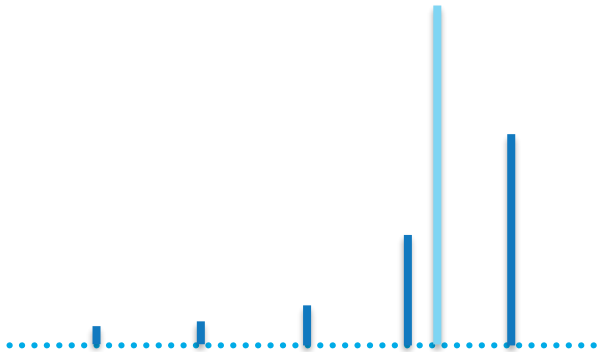
https://9gag.com

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.

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*