

PSY9511: Seminar 7

Deep learning for image processing tasks

Esten H. Leonardsen

07.11.24



UNIVERSITY
OF OSLO

Outline

1. Reflections on exercise 5
2. Deep learning with artificial neural networks
3. Convolutional neural networks for image processing

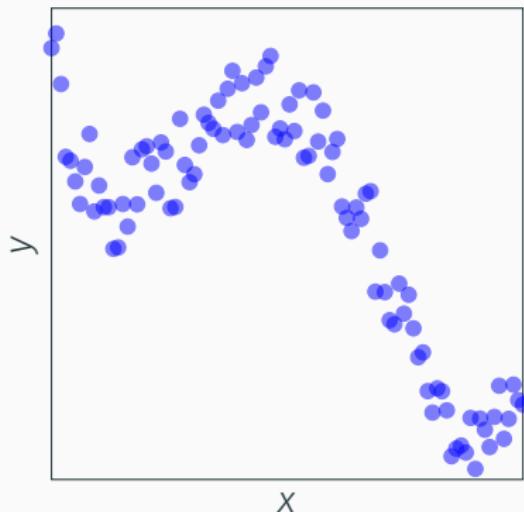


Deep learning

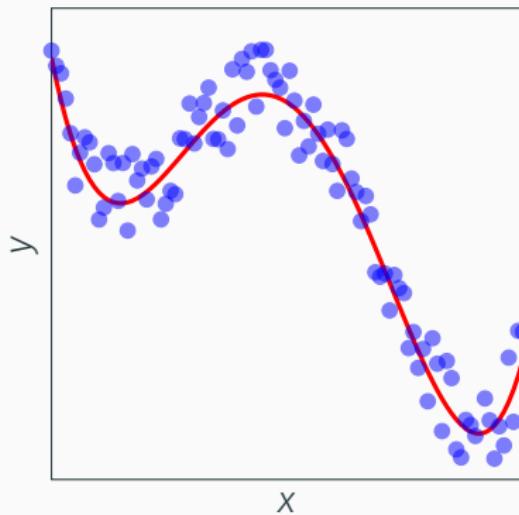


UNIVERSITY
OF OSLO

Deep learning: Motivation



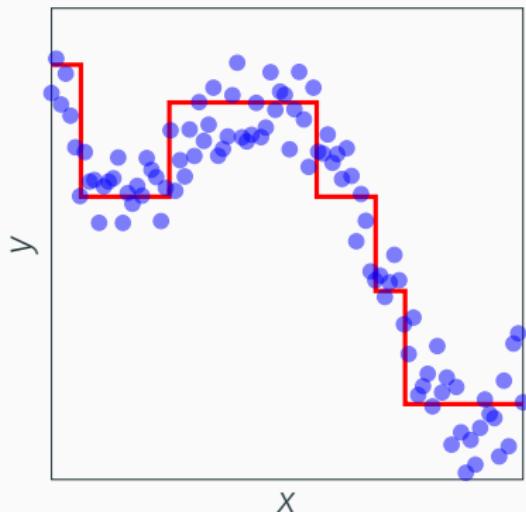
Deep learning: Motivation



$$\hat{y} = s(x)$$



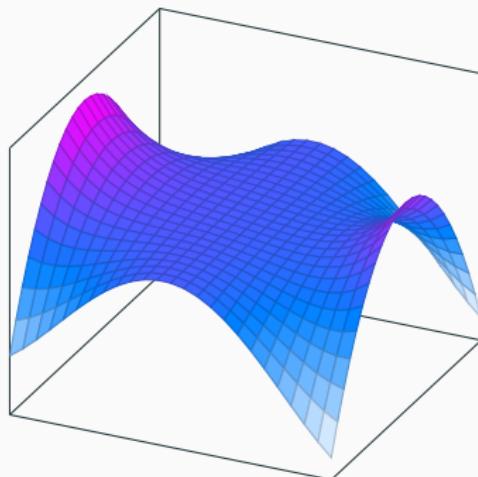
Deep learning: Motivation



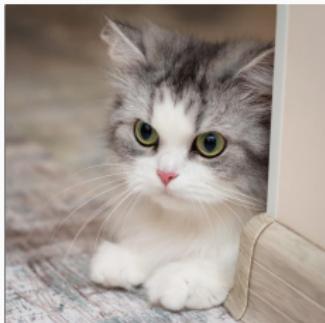
$$\hat{y} = \begin{cases} 4 & \dots \\ 3 & 0.2 \leq x < 0.6 \\ 1.5 & \dots \end{cases}$$



Deep learning: Motivation

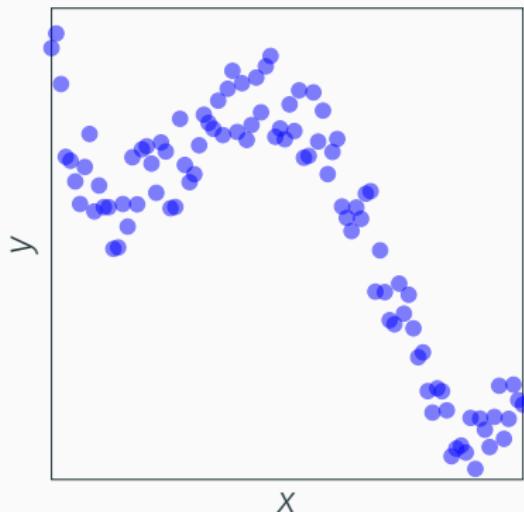


Deep learning: Motivation

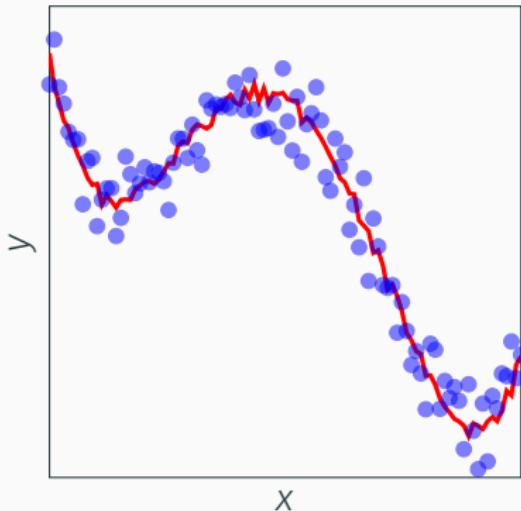


The cat wagged
its tail

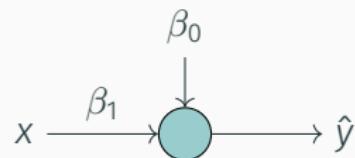
Deep learning: Motivation



Deep learning: Motivation



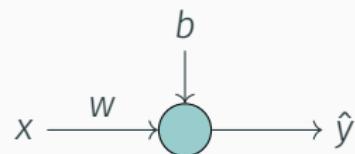
Deep learning: Artificial neural networks



$$\hat{y} = \beta_0 + \beta_1 x$$



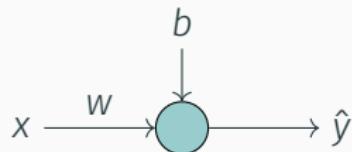
Deep learning: Artificial neural networks



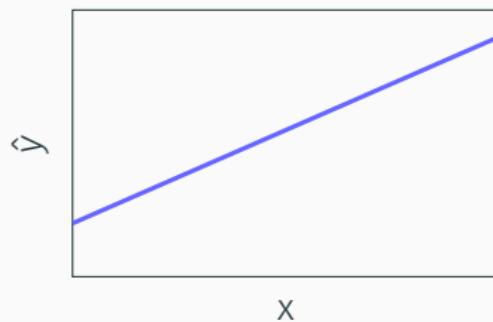
$$\hat{y} = wx + b$$



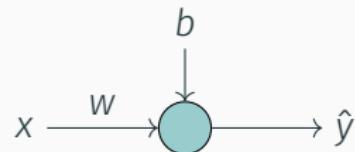
Deep learning: Artificial neural networks



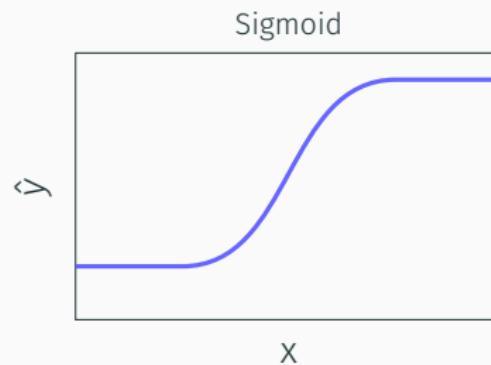
$$\hat{y} = wx + b$$



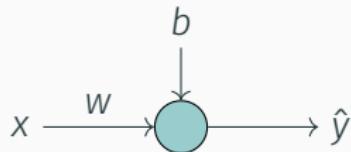
Deep learning: Artificial neural networks



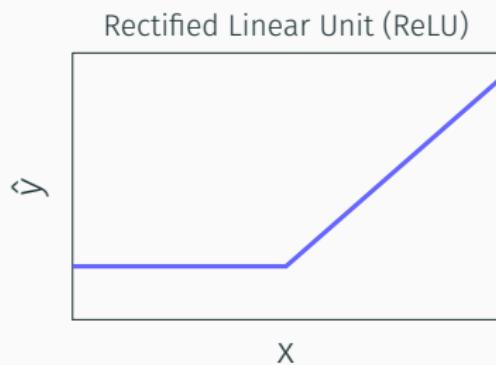
$$\hat{y} = \frac{e^{wx+b}}{1 + e^{wx+b}}$$



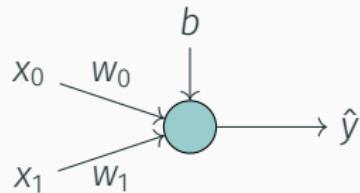
Deep learning: Artificial neural networks



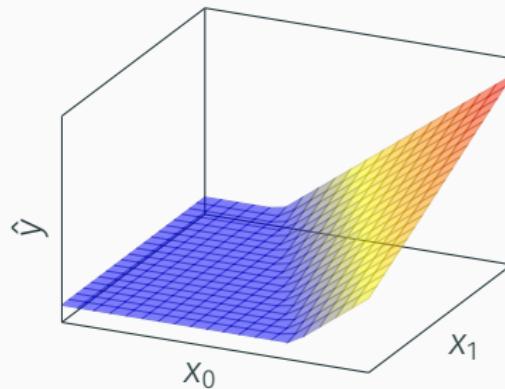
$$\hat{y} = \max(0, wx + b)$$



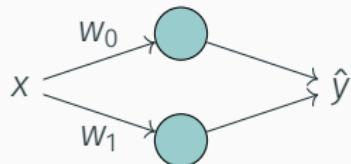
Deep learning: Artificial neural networks



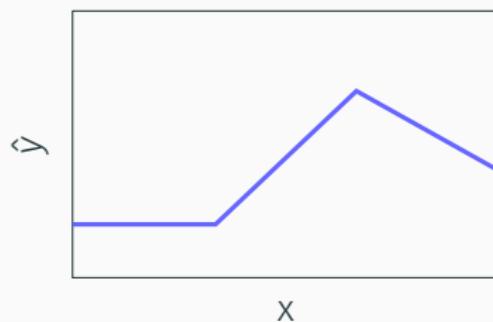
$$\hat{y} = \max(0, w_0x_0 + w_1x_1 + b)$$



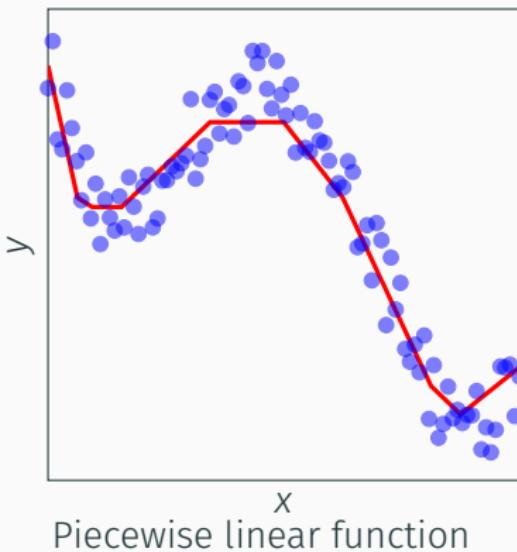
Deep learning: Artificial neural networks



$$\hat{y} = \max(0, w_0x + b_0) + \max(0, w_1x + b_1)$$



Deep learning: Artificial neural networks



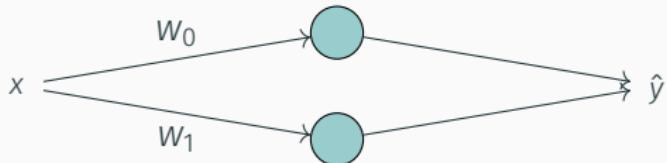
Deep learning: Artificial neural networks

Universal approximation theorem:

"Any relationship that can be described with a polynomial function can be approximated by a neural network with a single hidden layer."



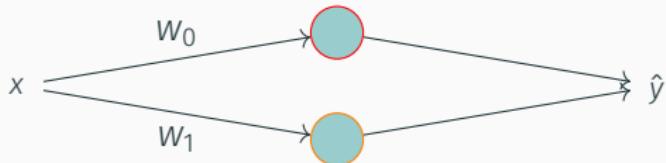
Deep learning: Artificial neural networks



$$\hat{y} = \max(0, w_0x + b_0) + \max(0, w_1x + b_1)$$



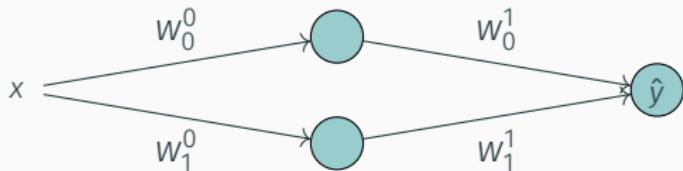
Deep learning: Artificial neural networks



$$\hat{y} = \max(0, w_0x + b_0) + \max(0, w_1x + b_1)$$



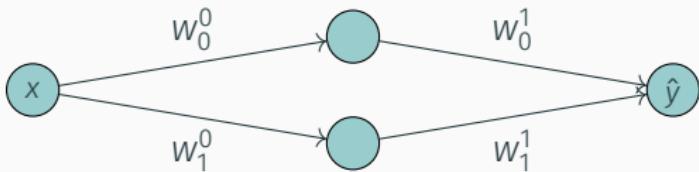
Deep learning: Artificial neural networks



$$\hat{y} = \max(0, w_0^1 * \max(0, w_0^0 * x + b_0^0) + w_1^1 * \max(0, w_1^0 * x + b_1^0) + b_0^1)$$



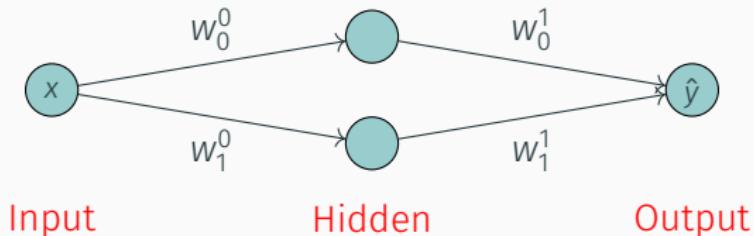
Deep learning: Artificial neural networks



$$\hat{y} = \max(0, w_0^1 * \max(0, w_0^0 * x + b_0^0) + w_1^1 * \max(0, w_1^0 * x + b_1^0) + b_0^1)$$



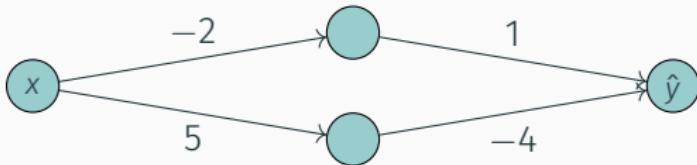
Deep learning: Artificial neural networks



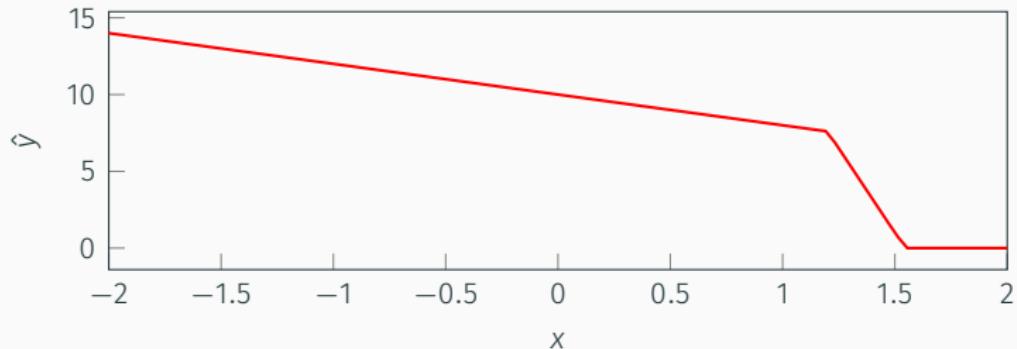
$$\hat{y} = \max(0, w_0^1 * \max(0, w_0^0 * x + b_0^0) + w_1^1 * \max(0, w_1^0 * x + b_1^0) + b_0^1)$$



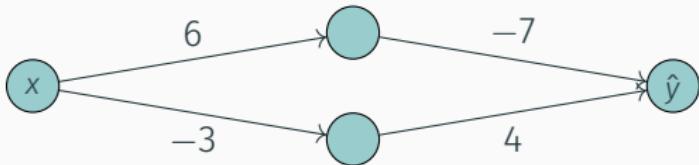
Deep learning: Artificial neural networks



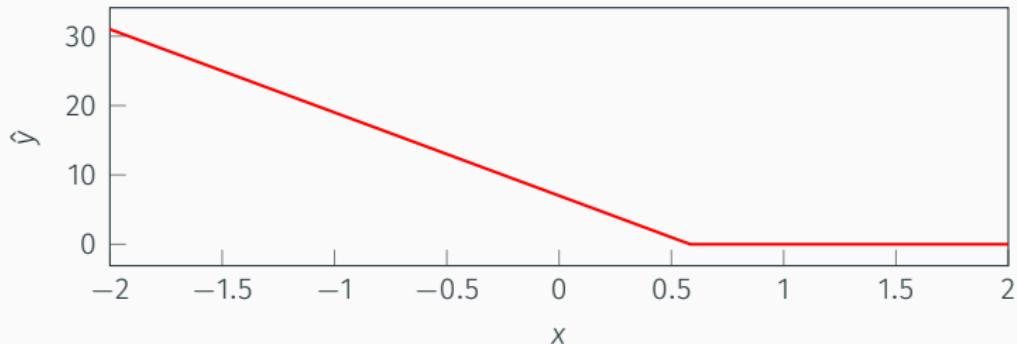
$$\hat{y} = \max(0, 1 * \max(0, (-2) * x + 3) + (-4) * \max(0, 5 * x + (-6)) + 7)$$



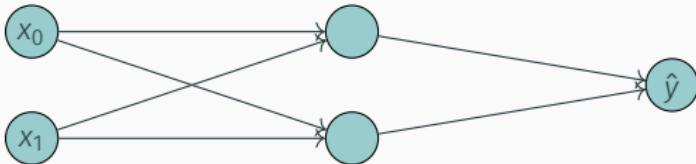
Deep learning: Artificial neural networks



$$\hat{y} = \max(0, -7 * \max(0, 6 * x + (-5)) + 4 * \max(0, (-3) * x + 2) - 1)$$



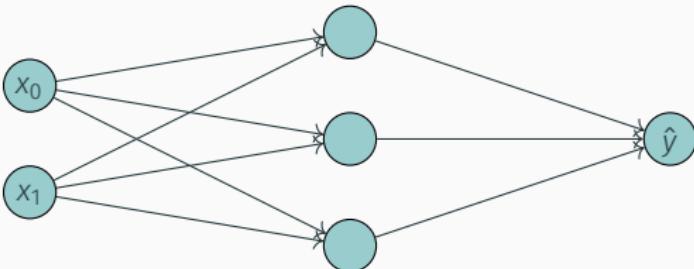
Deep learning: Artificial neural networks



$$\begin{aligned}\hat{y} = & \max(0, w_{0,0}^1 * \max(0, w_{0,0}^0 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\ & w_{1,0}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\ & b_1)\end{aligned}$$



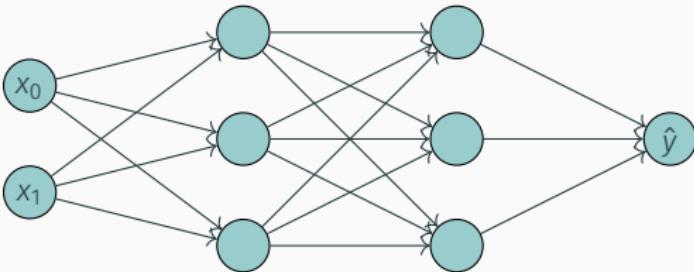
Deep learning: Artificial neural networks



$$\begin{aligned}\hat{y} = & \max(0, w_{0,0}^1 * \max(0, w_{0,0}^0 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\ & w_{1,0}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\ & w_{2,0}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\ & b_1)\end{aligned}$$



Deep learning: Artificial neural networks



$$\begin{aligned}\hat{y} = & \max(0, w_{0,0}^2 * \max(0, w_{0,0}^1 * \max(0, w_{0,0}^0 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\& w_{1,0}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\& w_{2,0}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\& b_{1,0}) + \\& w_{1,0}^2 * \max(0, w_{0,1}^1 * \max(0, w_{0,0}^0 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\& w_{1,1}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\& w_{2,1}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\& b_{1,1}) + \\& w_{2,0}^2 * \max(0, w_{0,2}^1 * \max(0, w_{0,0}^0 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\& w_{1,2}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\& w_{2,2}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\& b_{1,2}) + \\& b_2)\end{aligned}$$



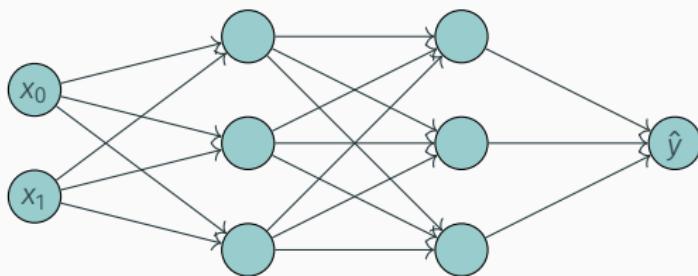
Deep learning: Artificial neural networks

Artificial neural networks: Combines artificial neurons, simple computational units that compute a non-linear function of their inputs, in a computational graph

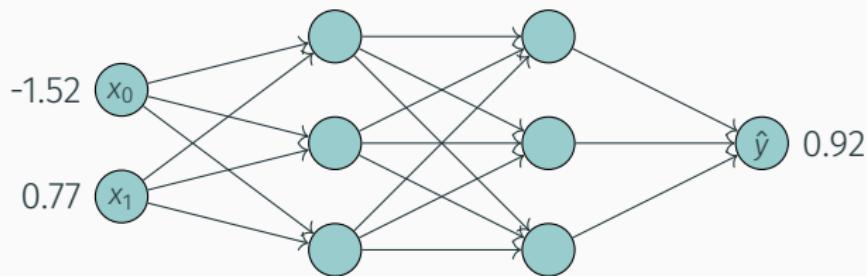
- Can approximate arbitrarily complex polynomial functions (given enough neurons)
- Organized in layers. We can expand a model in width (e.g. more neurons per layer) or depth (e.g. more layers)



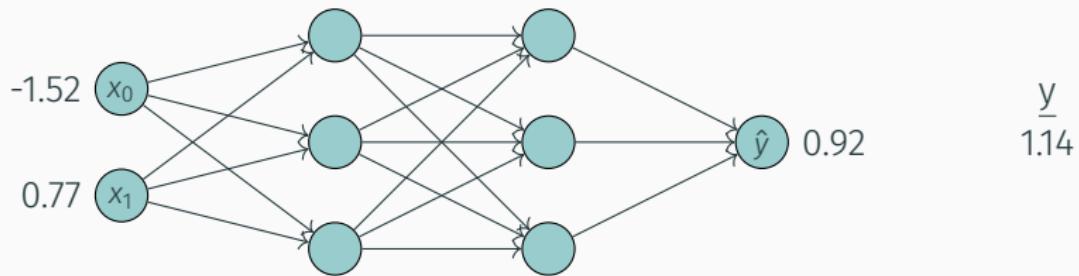
Deep learning: Loss functions



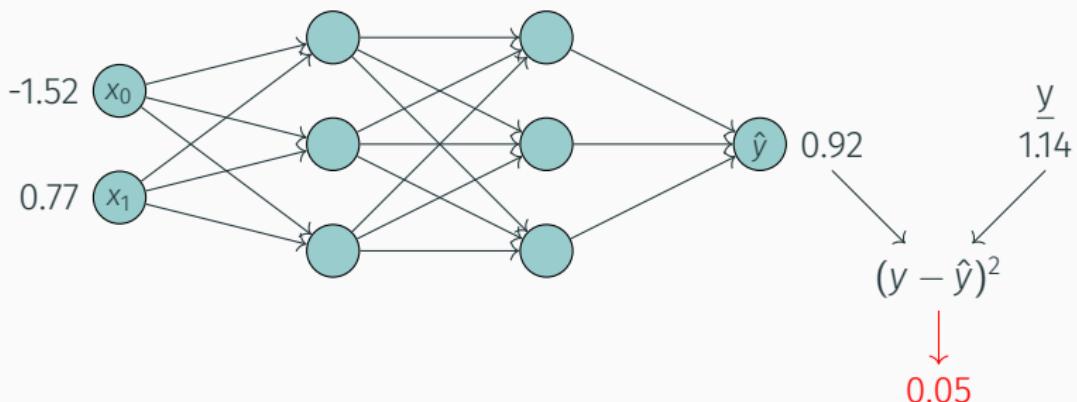
Deep learning: Loss functions



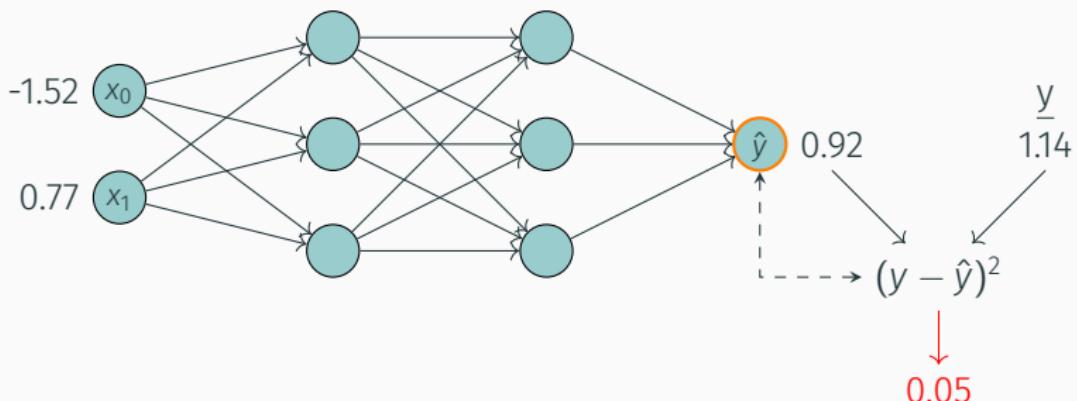
Deep learning: Loss functions



Deep learning: Loss functions



Deep learning: Loss functions



Deep learning: Loss functions

```
In[1]: from tensorflow.keras.layers import Input, Dense  
from tensorflow.keras import Model  
  
inputs = Input(shape=(2,))  
hidden1 = Dense(units=3, activation='relu')(inputs)  
hidden2 = Dense(units=3, activation='relu')(hidden1)  
outputs = Dense(units=1, activation=?)(hidden2)  
  
model = Model(inputs, outputs)  
model.compile(loss=?)
```



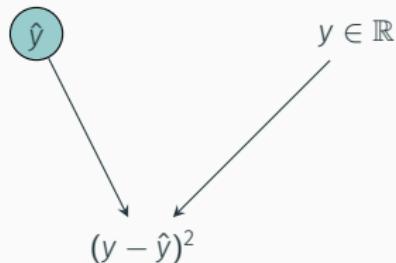
Deep learning: Loss functions

```
In[1]: from tensorflow.keras.layers import Input, Dense  
from tensorflow.keras import Model  
  
inputs = Input(shape=(2,))  
hidden1 = Dense(units=3, activation='relu')(inputs)  
hidden2 = Dense(units=3, activation='relu')(hidden1)  
outputs = Dense(units=1, activation='?')(hidden2)  
  
model = Model(inputs, outputs)  
model.compile(loss=?)
```



Deep learning: Loss functions

Regression



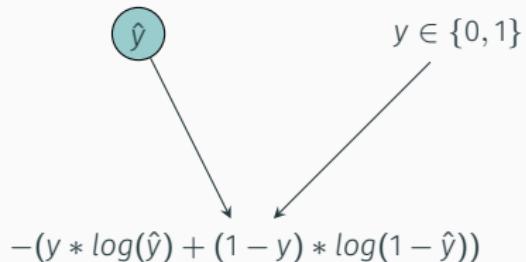
In[2]:

```
...
outputs = Dense(units=1, activation=None)(...)
...
model.compile(loss='mean_squared_error')
```



Deep learning: Loss functions

Binary classification



In[3]:

```
...  
outputs = Dense(units=1, activation='sigmoid')(...)  
...  
model.compile(loss='binary_crossentropy')
```



Deep learning: Loss functions

Multiclass classification

\hat{y}

$y \in \{cat, dog, bat\}$



Deep learning: Loss functions

Multiclass classification

\hat{y}

x_0	y
	cat
	dog
	bat
	dog



Deep learning: Loss functions

Multiclass classification

\hat{y}

x_0	cat	dog	bat
	1	0	1
	0	1	0
	0	0	1
	0	1	0



Deep learning: Loss functions

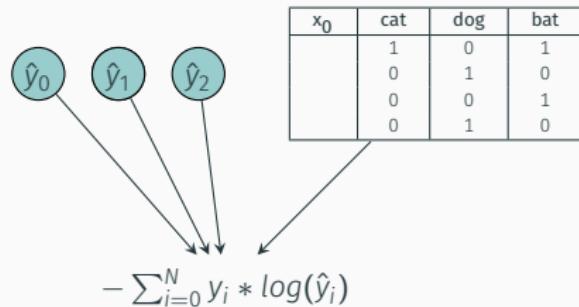
Multiclass classification

\hat{y}_0 \hat{y}_1 \hat{y}_2

x_0	cat	dog	bat
	1	0	1
	0	1	0
	0	0	1
	0	1	0



Deep learning: Loss functions



In[3]:

```
...
outputs = Dense(units=N_CLASSES, activation='softmax')(...)
...
model.compile(loss='categorical_crossentropy')
```



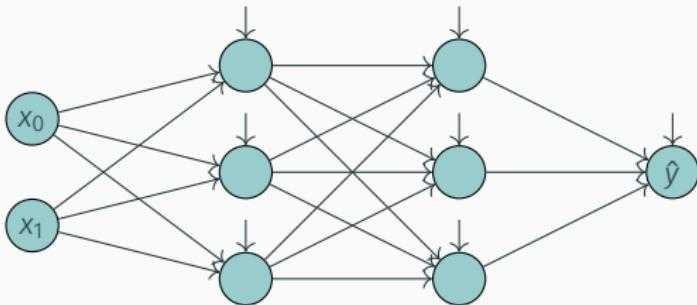
Deep learning: Loss functions

Loss functions: Behaves for neural networks as any other statistical learning model. However, important to **configure the final layer correctly**

- Regression: Mean squared error
 - No activation
- Binary classification: Binary cross-entropy
 - Sigmoid activation
- Multiclass classification: Categorical cross-entropy
 - Softmax activation



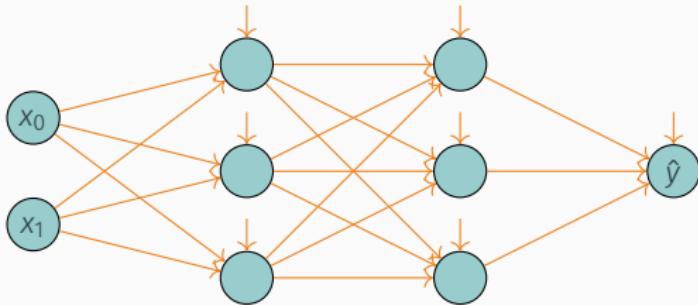
Deep learning: Training



$$\begin{aligned}\hat{y} = & \max(0, w_{0,0}^2 * \max(0, w_{0,0}^1 * \max(0, w_{0,0}^0 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\ & w_{1,0}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\ & w_{2,0}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\ & b_{1,0}) + \\ & w_{1,0}^2 * \max(0, w_{0,1}^1 * \max(0, w_{0,0}^0 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\ & w_{1,1}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\ & w_{2,1}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\ & b_{1,1}) + \\ & w_{2,0}^2 * \max(0, w_{0,2}^1 * \max(0, w_{0,0}^0 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\ & w_{1,2}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\ & w_{2,2}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\ & b_{1,2}) + \\ & b_2)\end{aligned}$$



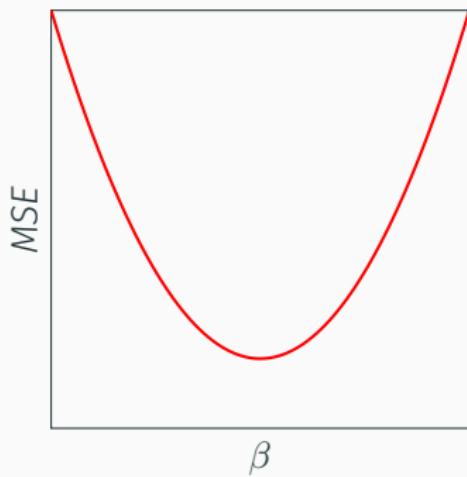
Deep learning: Training



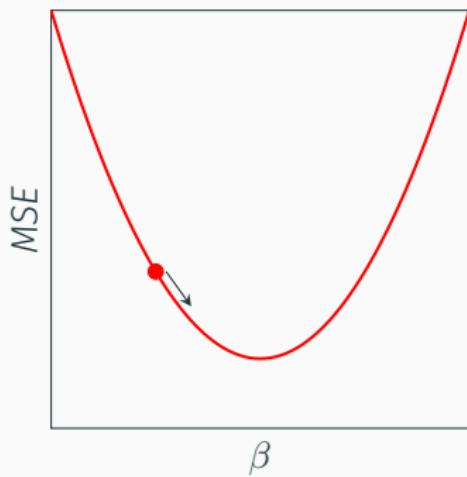
$$\begin{aligned}\hat{y} = & \max(0, w_{0,0}^2 * \max(0, w_{0,0}^1 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\& w_{1,0}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\& w_{2,0}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\& b_{1,0}) + \\& w_{1,0}^2 * \max(0, w_{0,1}^1 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\& w_{1,1}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\& w_{2,1}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\& b_{1,1}) + \\& w_{2,0}^2 * \max(0, w_{0,2}^1 * x_0 + w_{1,0}^0 * x_1 + b_{0,0}) + \\& w_{1,2}^1 * \max(0, w_{0,1}^0 * x_0 + w_{1,1}^0 * x_1 + b_{0,1}) + \\& w_{2,2}^1 * \max(0, w_{0,2}^0 * x_0 + w_{1,2}^0 * x_1 + b_{0,2}) + \\& b_{1,2}) + \\& b_2)\end{aligned}$$



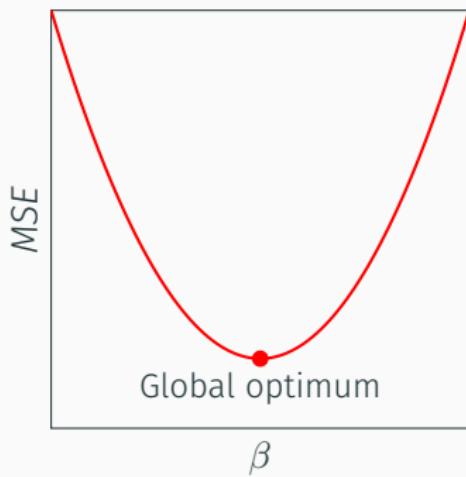
Deep learning: Training



Deep learning: Training



Deep learning: Training



Deep learning: Training

- The shape of the loss-function can be **very** non-convex
- We have a lot of parameters, meaning we have to optimize in a very high-dimensional space (typically $10^6 \leq |\beta| \leq 10^{12}$)
- Due to the nested nature of the model, the effect of the parameters rely on each other



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California,
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure¹.



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California,
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure¹.

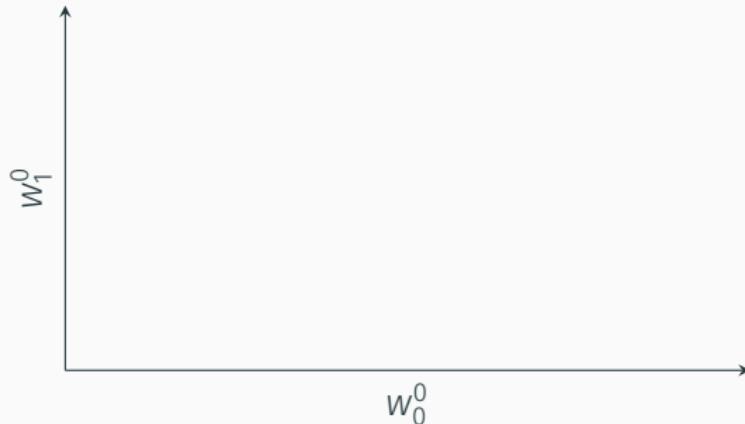


Deep learning: Training

$$\Delta R(\theta^m) = \frac{\partial R(\theta)}{\partial \theta} \Big|_{\theta=\theta^m}$$



Deep learning: Training

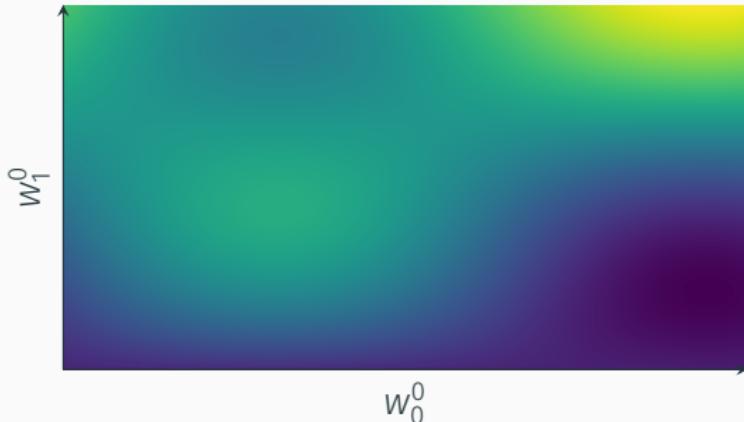


$$\Delta R(\theta^m) = \frac{\partial R(\theta)}{\partial \theta} \Big|_{\theta=\theta^m}$$

- θ are the parameters of the model



Deep learning: Training

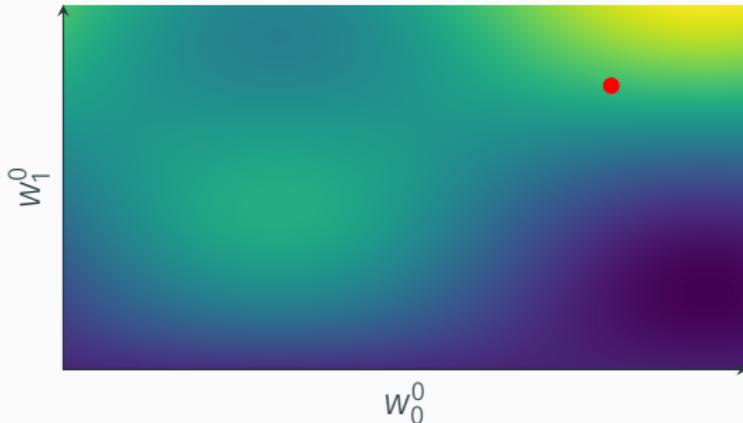


$$\Delta R(\theta^m) = \left. \frac{\partial R(\theta)}{\partial \theta} \right|_{\theta=\theta^m}$$

- θ are the parameters of the model
- $R(\theta)$ is the loss as a function of the parameters



Deep learning: Training

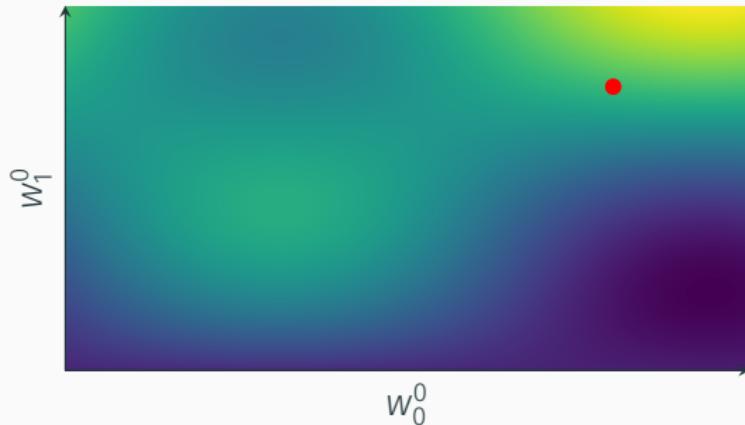


$$\Delta R(\theta^m) = \left. \frac{\partial R(\theta)}{\partial \theta} \right|_{\theta=\theta^m}$$

- θ are the parameters of the model
- $R(\theta)$ is the loss as a function of the parameters
- θ^m is a specific configuration of parameters



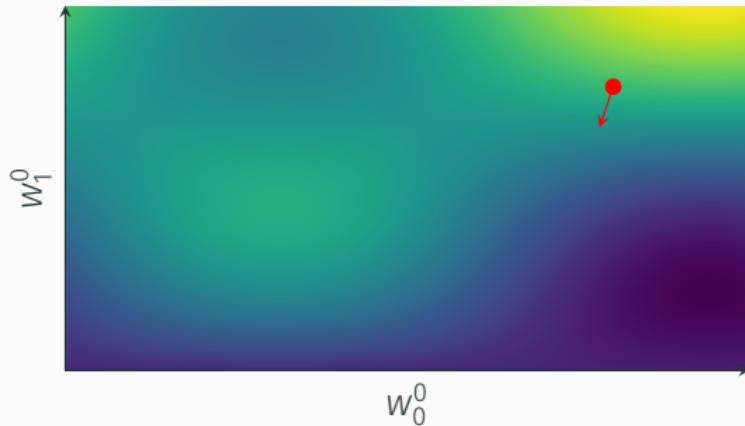
Deep learning: Training



$$\frac{\partial R_i(\theta)}{\partial w_{jk}} = \frac{\partial R(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{z_{ik}}{w_{kj}}$$



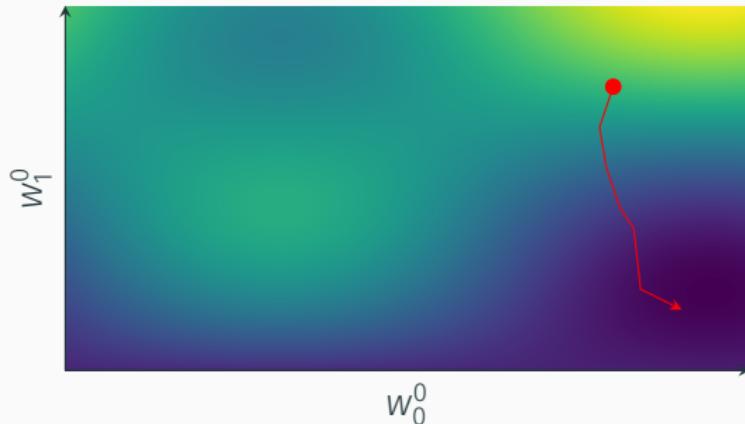
Deep learning: Training



$$\frac{\partial R_i(\theta)}{\partial w_{jk}} = \frac{\partial R(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{z_{ik}}{w_{kj}}$$



Deep learning: Training



$$\frac{\partial R_i(\theta)}{\partial w_{jk}} = \frac{\partial R(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{z_{ik}}{w_{kj}}$$



Deep learning: Training

Backpropagation: Uses gradient descent to iteratively determine how the model weights should be updated to minimize the loss function

- **Gradient descent**: Calculate gradient based on all data points



Deep learning: Training

Backpropagation: Uses gradient descent to iteratively determine how the model weights should be updated to minimize the loss function

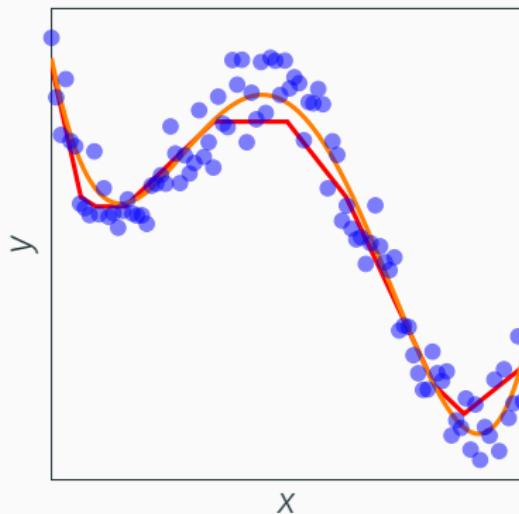
- **Gradient descent**: Calculate gradient based on all data points
- **Stochastic gradient descent**: Calculate gradient based on a batch of data points



Deep learning: The conundrum

Splines: A smooth curve implemented via piecewise polynomial functions

Neural networks: A piecewise linear function implemented as a hierarchy of artificial neurons



Deep learning: The conundrum

Splines: A smooth curve implemented via piecewise polynomial functions

- Requires us to carefully balance the complexity of the function

Neural networks: A piecewise linear function implemented as a hierarchy of artificial neurons



Deep learning: The conundrum

Splines: A smooth curve implemented via piecewise polynomial functions

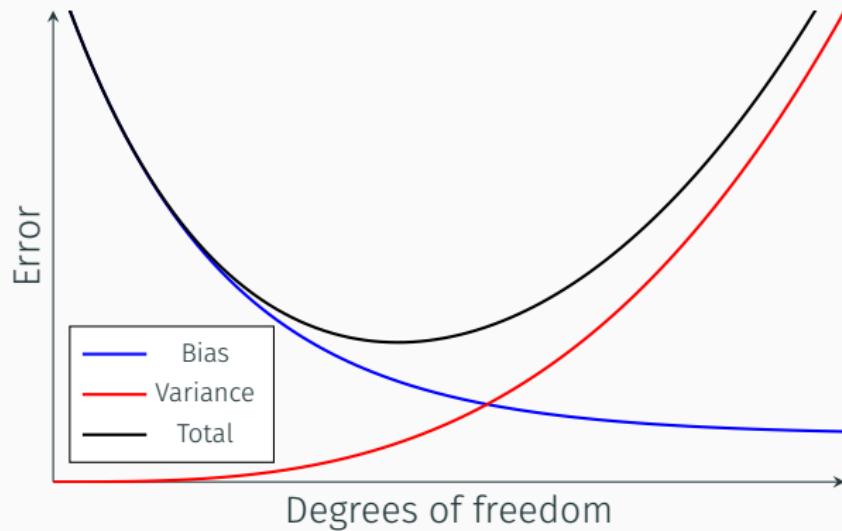
- Requires us to carefully balance the complexity of the function

Neural networks: A piecewise linear function implemented as a hierarchy of artificial neurons

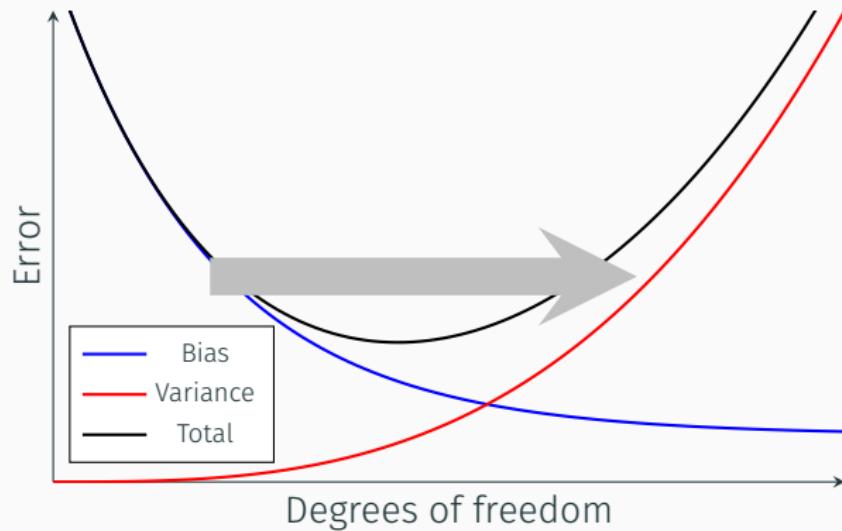
- Overparameterization 😵



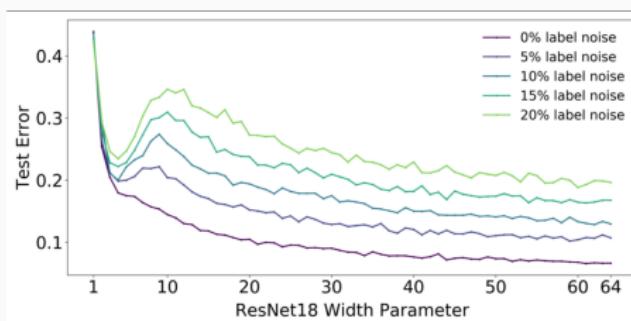
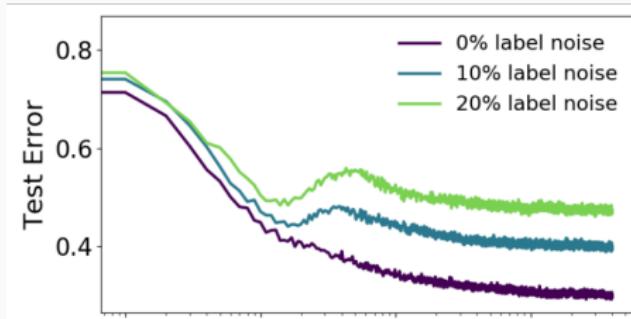
Deep learning: The conundrum



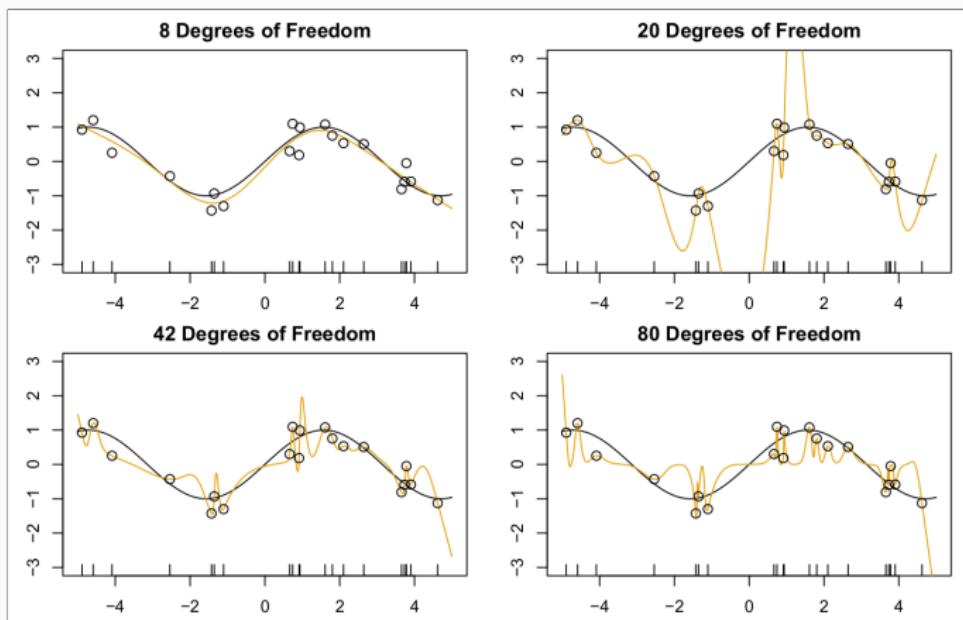
Deep learning: The conundrum



Deep learning: The conundrum



Deep learning: The conundrum



Deep learning: The conundrum

Overparameterization: Deep artificial neural networks generally have far more parameters than necessary (and often more than the number of data points)

- At face value, it is surprising that this does not yield severe overfitting
- However, it can be shown that neural networks, after perfectly fitting their training data, generally become more well-behaved and less wild



Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression

$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$



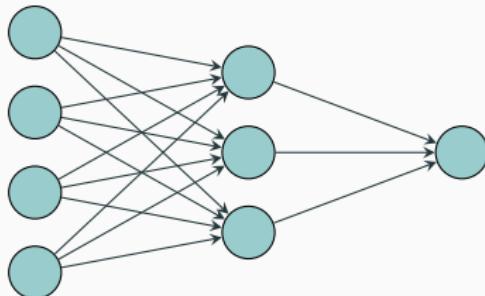
Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training



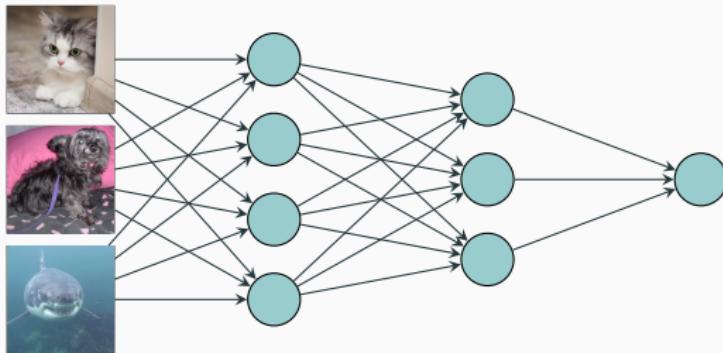
Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training



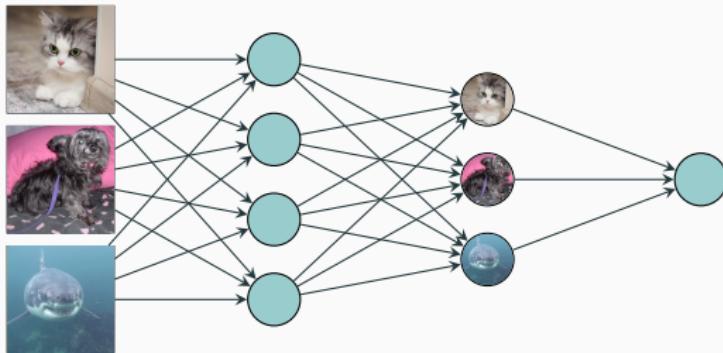
Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training



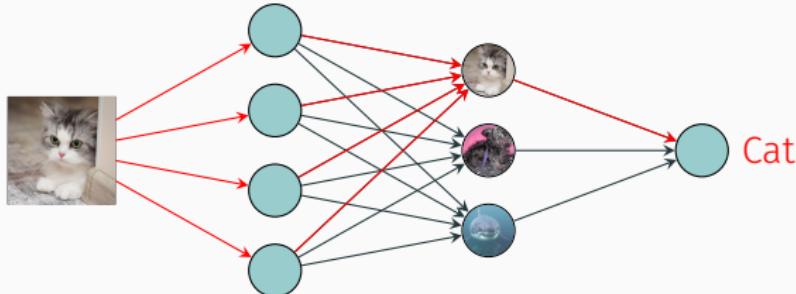
Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training



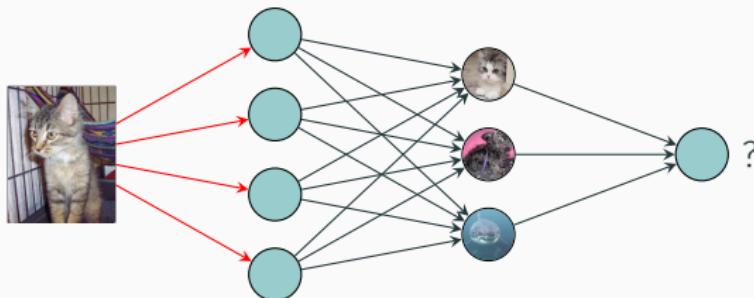
Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training



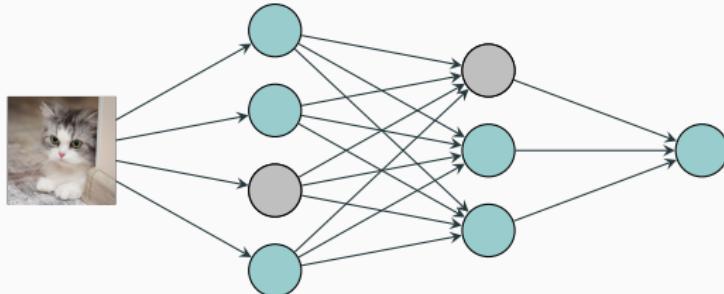
Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training



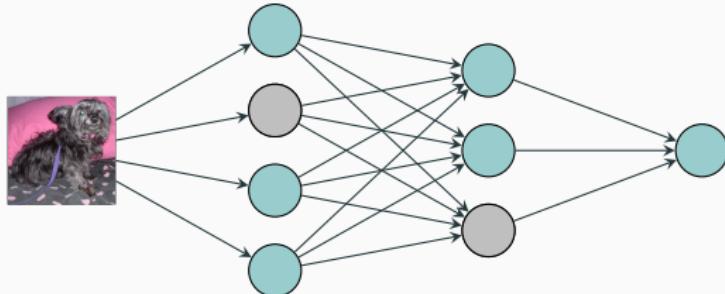
Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training



Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training



Deep learning: Regularization

- Weight decay: Applies an ℓ_2 -penalty to the weights, similarly to ridge regression
$$R(\theta; \lambda) = R(\theta) + \lambda \sum \theta^2$$
- Dropout: Randomly deactivates a fraction of the neurons during training
- Data augmentation: Attempts to generate more data



Computer vision



UNIVERSITY
OF OSLO

Image processing: Background



Cat



Image processing: Background



Sunflower



Ladybug



Cat



Airplane



Shark

ImageNet: ~14m images, ~22k categories

Image processing: Background



Image processing: Background



Image processing: Background



Image processing: Background



Image processing: Image data



Image processing: Image data



Image processing: Image data



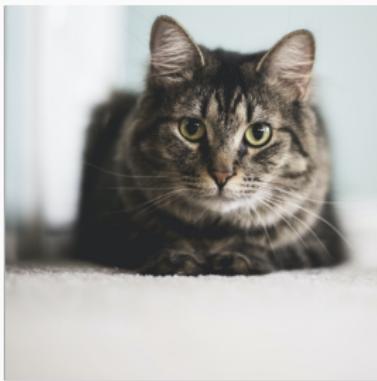
Image processing: Image data



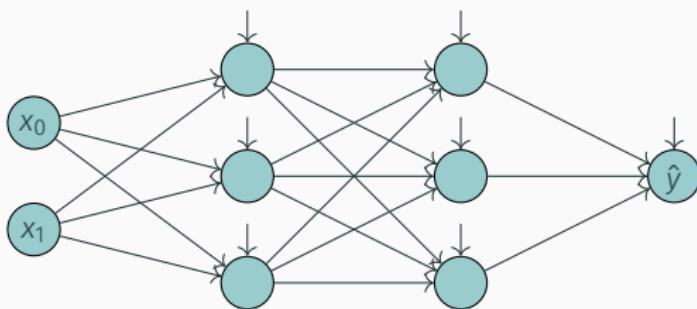
Image processing: Image data



Image processing: Image data



Convolutional neural networks: Inputs



Convolutional neural networks: Inputs

x_0

x_1

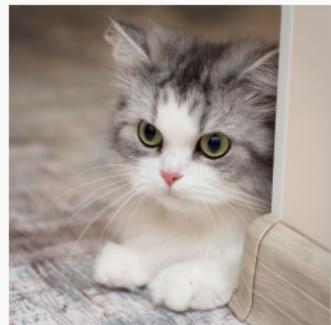


Convolutional neural networks: Inputs

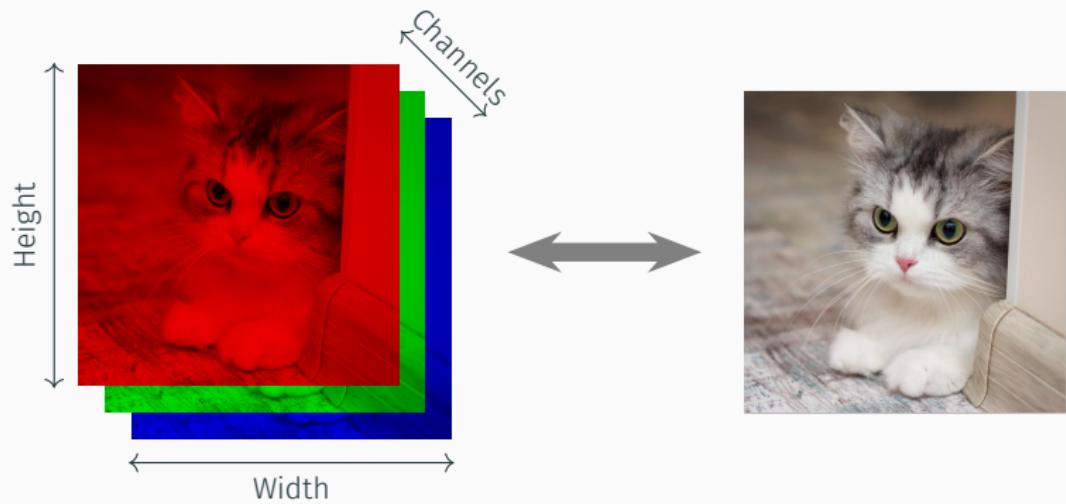
x_0

x_1

?

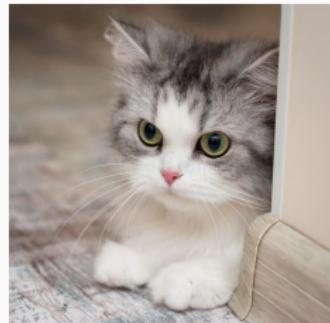


Convolutional neural networks: Inputs

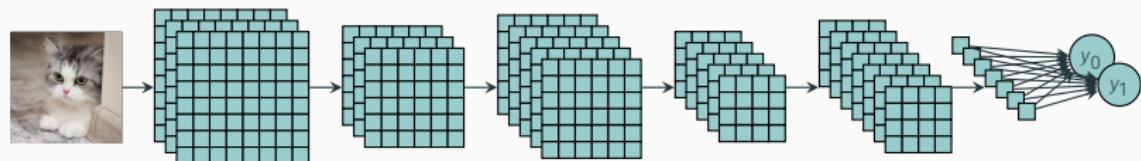


Convolutional neural networks: Inputs

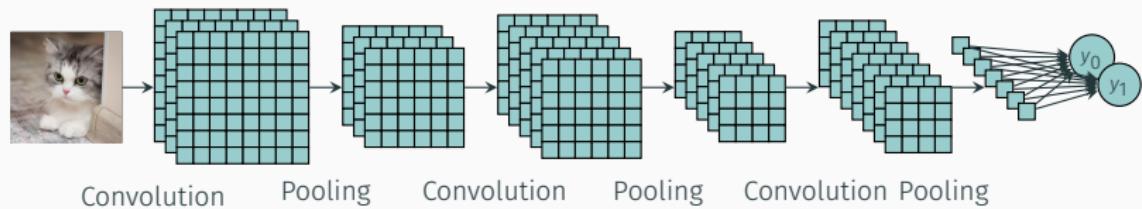
x_{000}	x_{010}	x_{020}	x_{030}	x_{040}		
x_{100}	x_{110}	x_{120}	x_{130}	x_{140}		
x_{200}	x_{210}	x_{220}	x_{230}	x_{240}		
x_{300}	x_{310}	x_{320}	x_{330}	x_{340}		
x_{400}	x_{410}	x_{420}	x_{430}	x_{440}		
					1	2
					1	2
					1	2
					1	2



Convolutional neural networks: Architecture

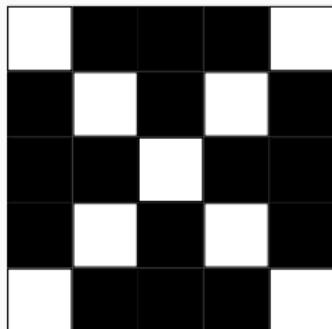


Convolutional neural networks: Architecture



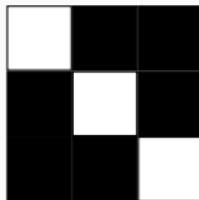
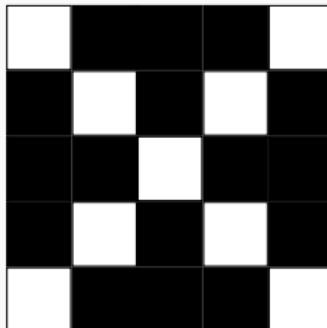
Convolutional neural networks: Convolution

Image



Convolutional neural networks: Convolution

Image

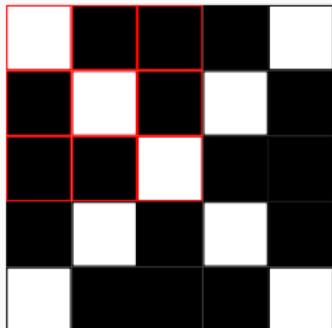


Pattern 1

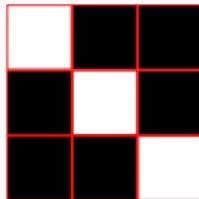


Convolutional neural networks: Convolution

Image



3

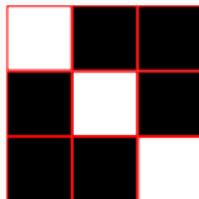
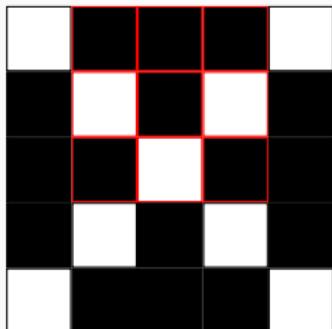


Pattern 1



Convolutional neural networks: Convolution

Image

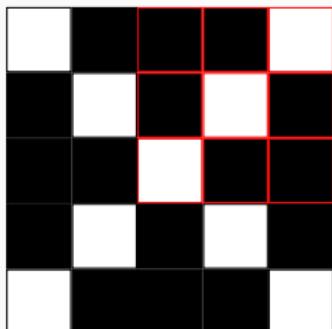


Pattern 1

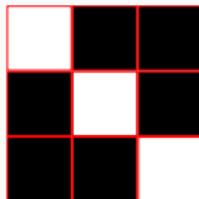


Convolutional neural networks: Convolution

Image



3	0	1
---	---	---

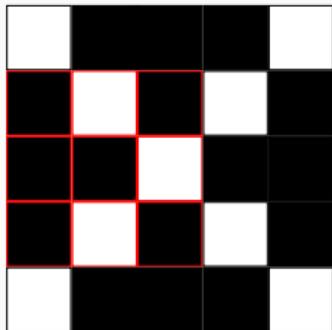


Pattern 1

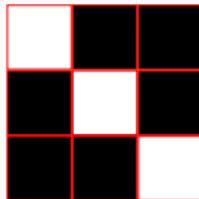


Convolutional neural networks: Convolution

Image



3	0	1
0		

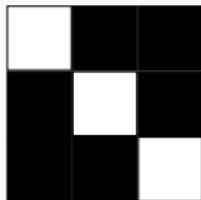
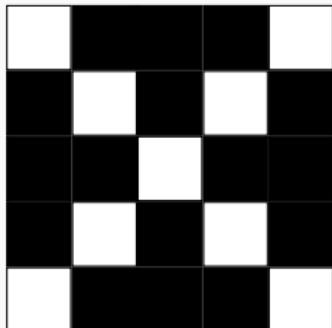


Pattern 1



Convolutional neural networks: Convolution

Image



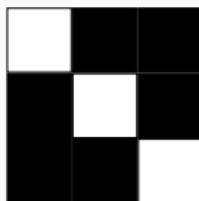
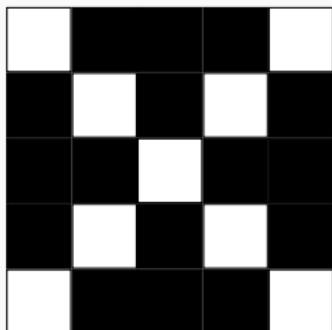
Pattern 1

3	0	1
0	3	0
1	0	3

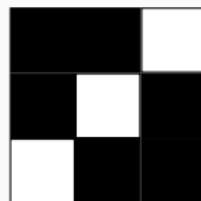


Convolutional neural networks: Convolution

Image



Pattern 1



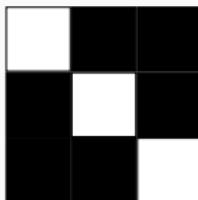
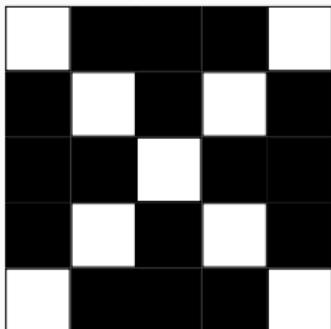
Pattern 2

1	0	3
0	3	0
3	0	1

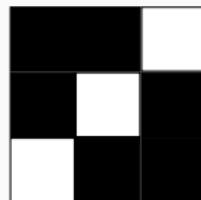


Convolutional neural networks: Convolution

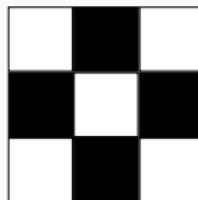
Image



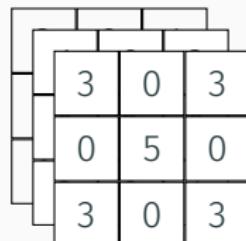
Pattern 1



Pattern 2

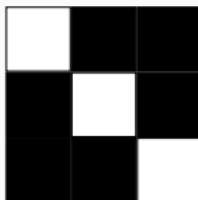
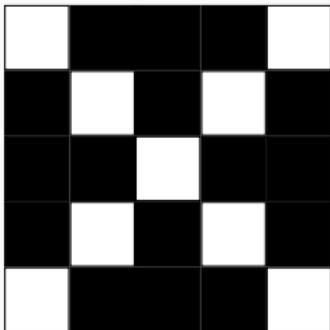


Pattern 3

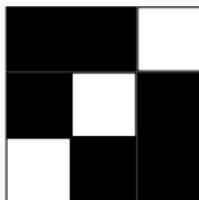


Convolutional neural networks: Convolution

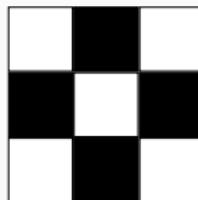
Image



Pattern 1

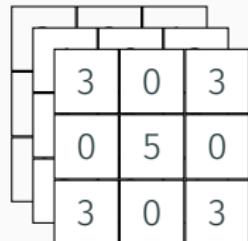


Pattern 2



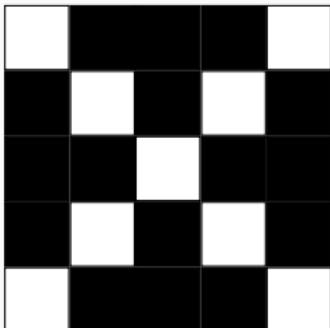
Pattern 3

Feature maps

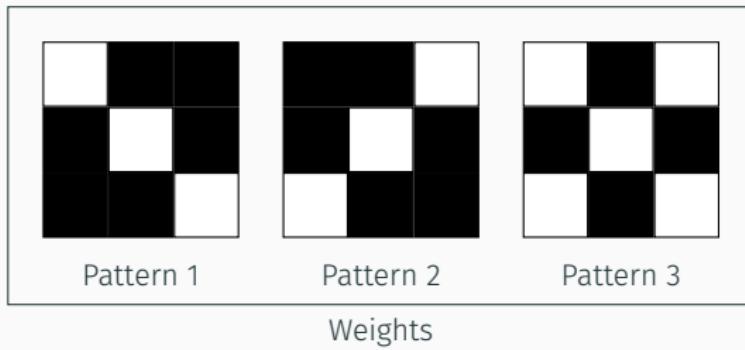
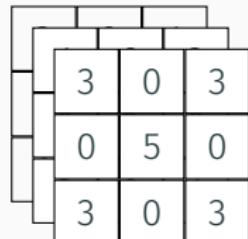


Convolutional neural networks: Convolution

Image

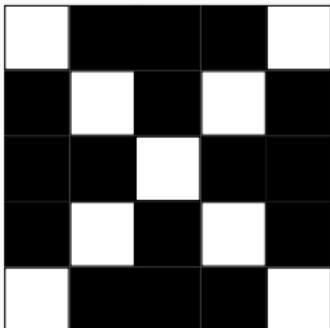


Feature maps

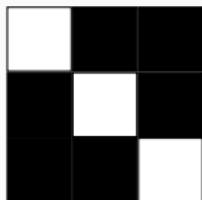
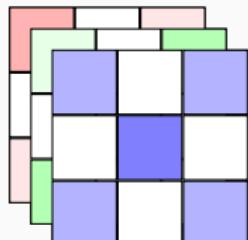


Convolutional neural networks: Convolution

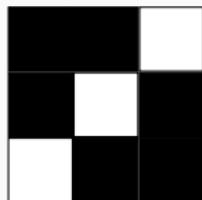
Image



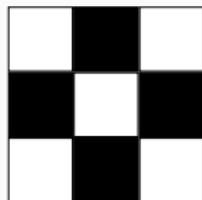
Feature maps



Pattern 1



Pattern 2



Pattern 3

Weights



Convolutional neural networks: (Max-)Pooling

Feature map

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



Convolutional neural networks: (Max-)Pooling

Feature map

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



Convolutional neural networks: (Max-)Pooling

Feature map

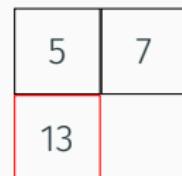
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



Convolutional neural networks: (Max-)Pooling

Feature map

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



Convolutional neural networks: (Max-)Pooling

Feature map

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

5	7
13	15



Convolutional neural networks: (Max-)Pooling

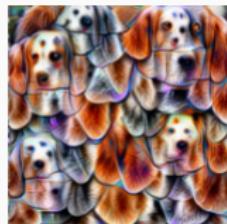
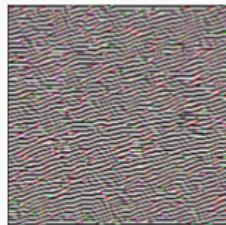
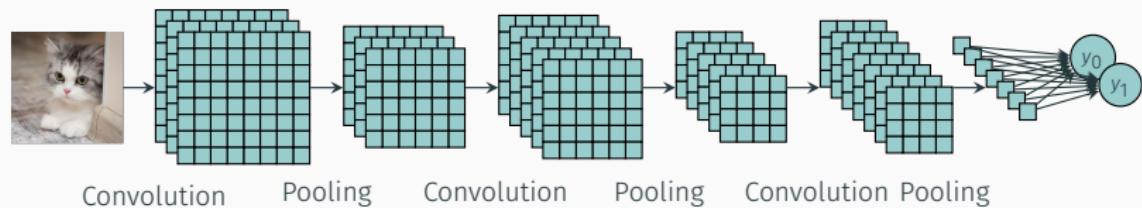
Feature map

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

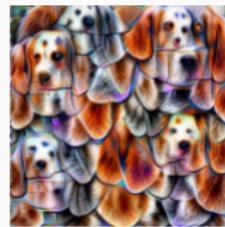
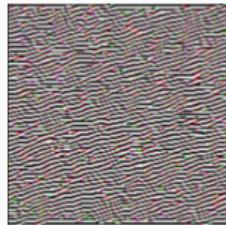
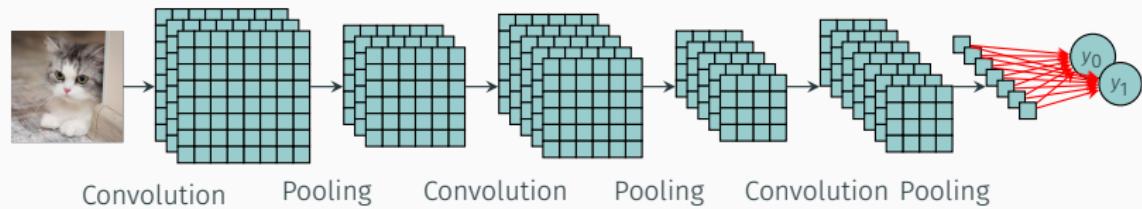
5	7
13	15



Convolutional neural networks: Overview



Convolutional neural networks: Overview

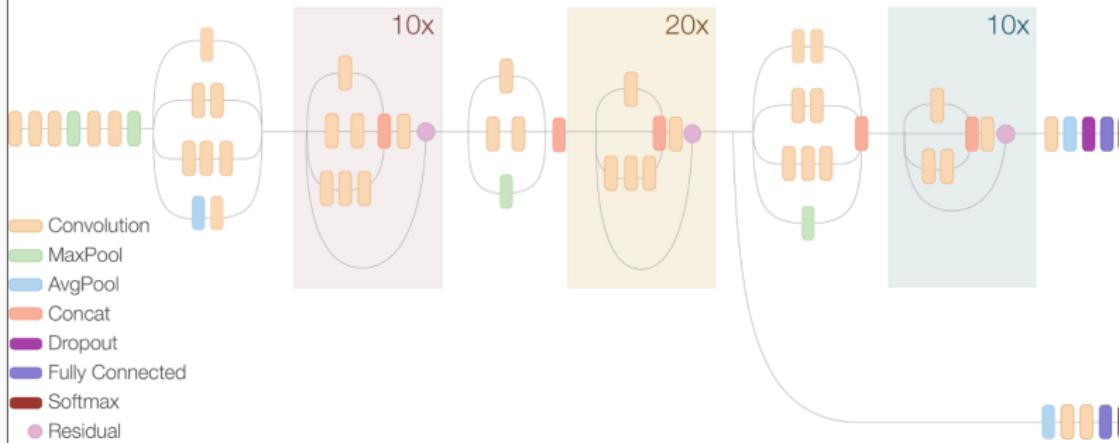


Convolutional neural networks: Overview

Inception Resnet V2 Network



Compressed View



Convolutional neural networks: Overview

Convolutional neural networks (CNNs): Artificial neural networks tailored specifically for image data.

- Takes raw pixel data as input, e.g. arrays of size $H \times W \times C$
- Mainly consists of convolutions and pooling operations, which lets us recognize larger and more abstract patterns the deeper we get in the model
- The trainable parameters are the weights of the convolutional kernels, e.g. the patterns the model is looking for
- New architectures extend beyond this basic formula, by employing residual connections, attention mechanisms etc.



Image processing: Transfer learning

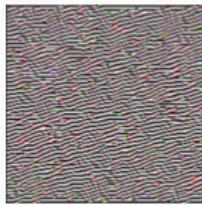
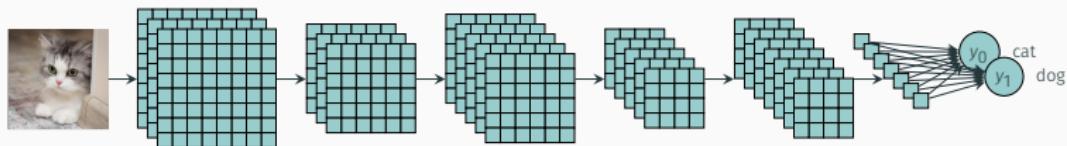


Image processing: Transfer learning

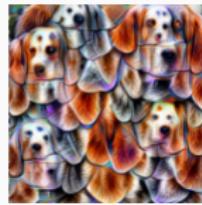
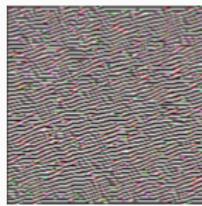
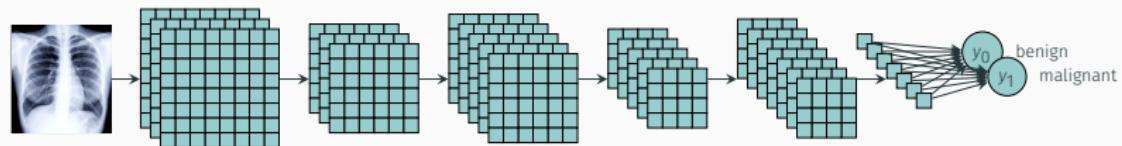


Image processing: Transfer learning

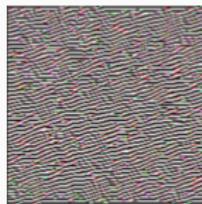
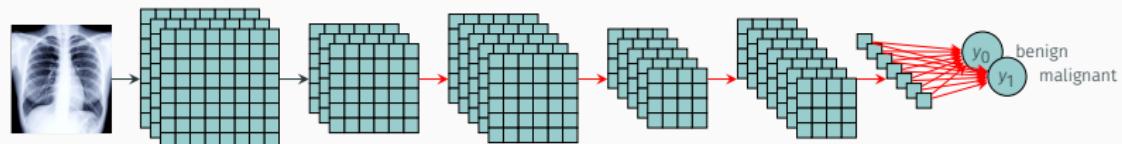


Image processing: Transfer learning

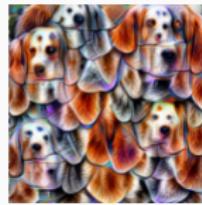
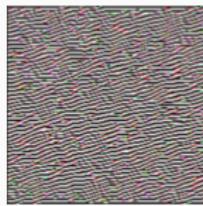
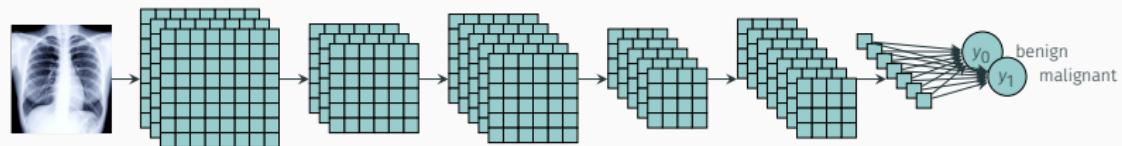


Image processing: Transfer learning

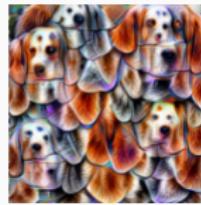
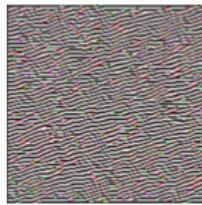
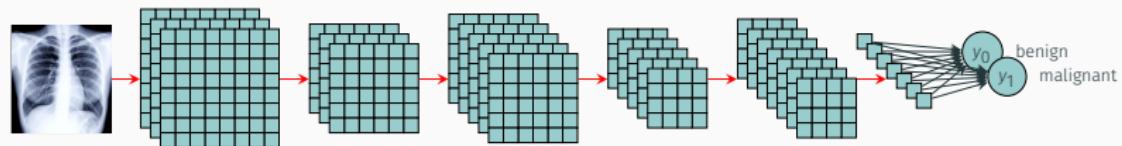


Image processing: Transfer learning

Transfer learning: Utilizing pretrained models to solve new tasks.

- Allows us to solve tasks where we don't have enough data to train a model from scratch
- Common to either freeze the weights of the convolutional part of the pre-trained model and train a new classifier on top of them, or to finetune the entire model



Image processing: Transfer learning

<https://keras.io/api/applications/>



Image processing: Data augmentation

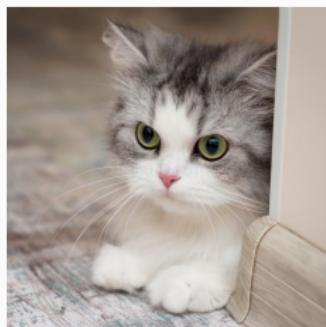


Image processing: Data augmentation

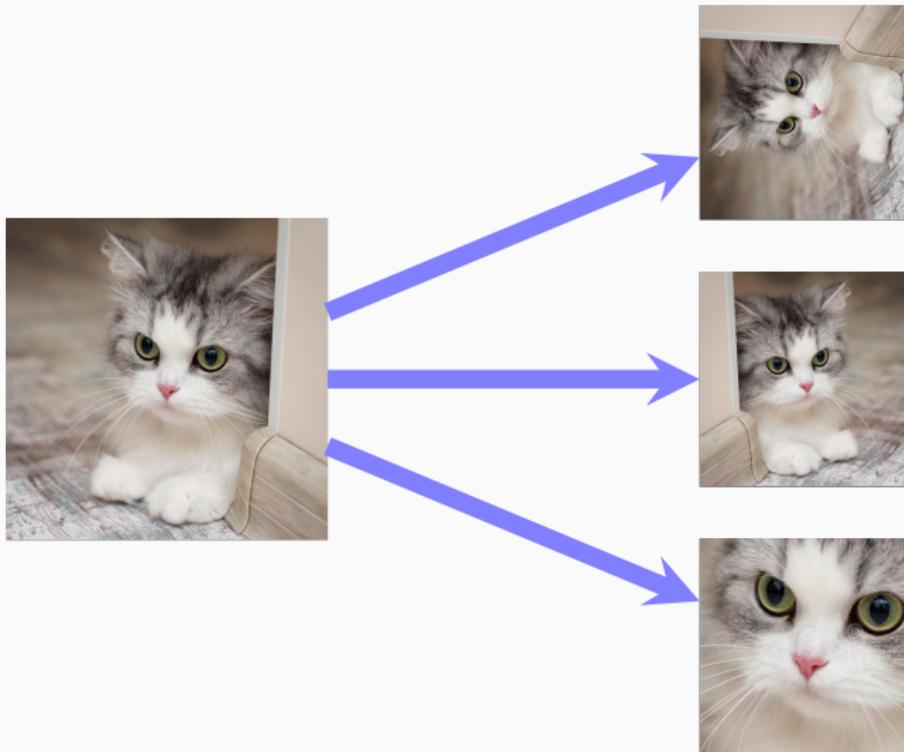


Image processing: Data augmentation

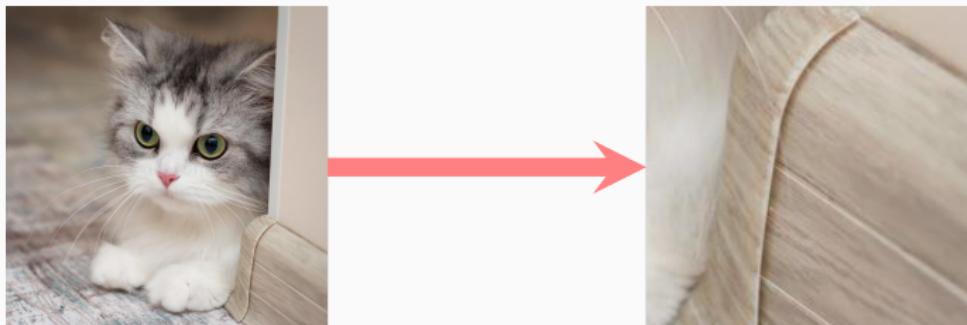


Image processing: Data augmentation

<https://albumentations.ai/>



Image processing: Tutorial

<https://github.com/estenhl/flowers>

