Autonomous Learning

Exercise 10 - Assignment 3

Simon Reichhuber

June 29, 2020

University of Kiel, Summer Term 2020

Table of Contents

Convolutional Neural Network

What We See

08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

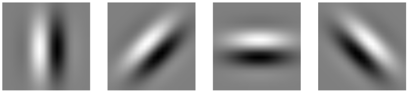
What Computers See

- input: $32 \times 32 \times 3$ (x-axis $\times$ y-axis $\times$ RGB)
- values from 0 to 255
- output: 80% dog and 20% cat

FEATURES

1. fur
2. ears
3. four paws

1. corners
2. edges
3. curves



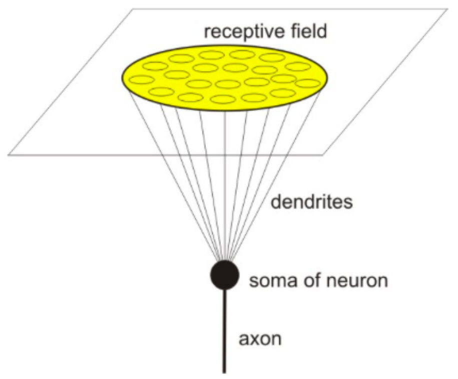
INSPIRATON FROM BIOLOGY

- research by *Hubel and Wiesel* on the brain of mammals
- *simple cells (S cells)*:

  basic shapes within certain within a certain range and angle
- *complex cells (C cells)*:

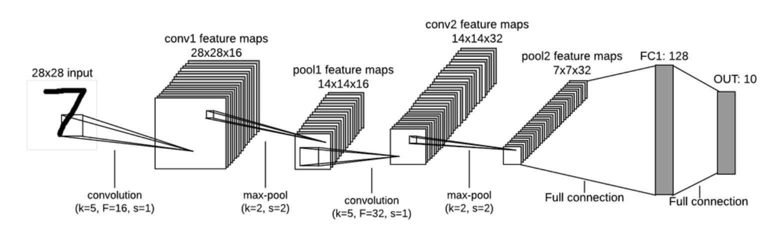
  larger receptive fields $\rightarrow$ no limitation to a specific position

RECEPTIVE FIELD



- region of the retina
- influences triggering of the associated neuron
- Every sensor neuron has the same receptive field.
- Receptive fields can overlap.

ARCHITECTURE



Two components:

1. Hidden layers / feature extraction:

   Multiple convolutions and pooling operations → features are recognised

2. Classification:

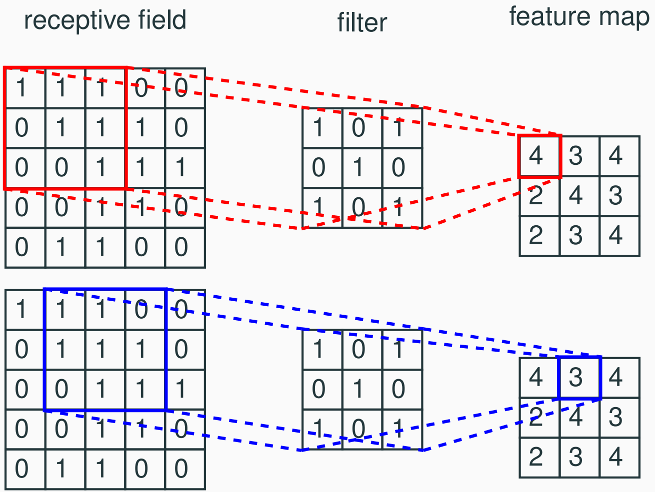
   fully connected layers → classifier working on the extracted features

Convolution

... is the mathematical combination of two functions to create a third one: Two information sets are combined.

- convolution on input data using kernel/filter $\rightarrow$ feature map
- The filter moves across the input and performs a matrix multiplication in every step.

CONVOLUTION



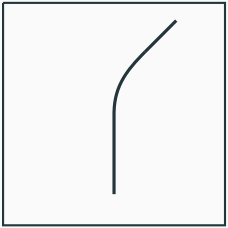
Filter

- array of numbers
- parameters or weights (numbers)
- element-wise multiplication
- slides/convolves across the input
- The depth of the filter must match the input.

High Level Perspective

- feature identifier
- edges, basic colours, and curves

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



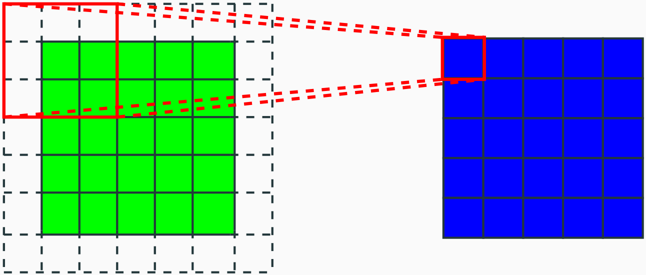
Stride

... is the step size of the filter moving across the input, normally 1 (pixel per step). Higher stride leads to less overlapping.

Padding

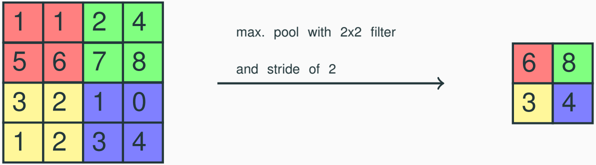
... is wrapping the input with a layer of zeros so the feature map is guaranteed to be no smaller than the input. This also ensures that filter and stride fit within the input.

Padding



POOLING

- between convolution layers
- reduces dimensionality → less parameters
- reduces training time
- counteracts overfitting

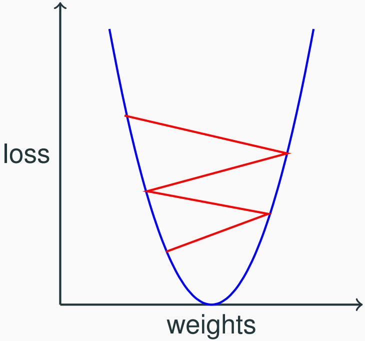


- The convolution is followed by *fully connected layers*.
- Convolution generates 3-dimensional data (several layers).
- Fully connected layers accept only 1-dimensional data.

$\Rightarrow$ Flatten

Learning Rate

- determines the step size while updating the weights
- higher learning rate $\rightarrow$ model converges faster
- learning too high $\rightarrow$ overshooting the optimal point



KERAS


```python
from keras.layers import Dense, Conv2D, Flatten
from keras.models import Sequnetial
from keras.optimizers import RMSprop

model = Sequential()

# 1st Convolutional Layer
model.add(Conv2D(16, 5, strides=1, activation='relu', input_shape=(32, 32, 2),
      kernel_initializer='he_normal', padding='same'))
# 2nd Convolutional Layer
model.add(Conv2D(32, 1, strides=1, activation='relu', kernel_initializer='he_normal',padding
      ='same'))

# Flatten Function
model.add(Flatten())

# Fully Connected Layer
model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(8, activation='linear', kernel_initializer='he_normal'))

model.compile(loss='mse',optimizer=RMSprop(lr=0,00025))
```

Improvements for DQN

Double Learning

- Agent tends to over-approximate the Q values.

$$Q(s, a) \to r + \gamma \max_{a} Q(s', a)$$

- Example:
  - All agents have the same true Q-value for s.
  - The estimation is inherently noisy $\to$ different from Q_{true}
  - $\max \to$ The action with the highest positive error is chosen
  - Error is propagated to future states.
    $\Rightarrow$ positive bias / value overestimation

Double Learning

- Solution: Double Learning
- Standard Q-Learning:
  - Two Q-functions Q_1 and Q_2 are learned independently.
  - One chooses the action which is to maximised.
  - The other determines the value of the chosen action.
  - One of the functions is randomly chosen to be updated:

$$Q_1(s,a) \to r + \gamma Q_2(s', \operatorname*{argmax}_a Q_1(s',a))$$

or

$$Q_2(s,a) \to r + \gamma Q_1(s', \operatorname*{argmax}_a Q_2(s',a))$$

Double Deep Q-Learning

- already two different Q-functions
- $Q(s, a; \theta)$
- $Q(s, a; \theta^-)$

$$Q(s, a; \theta) \rightarrow r + \gamma Q(s', \underset{a}{\operatorname{argmax}}\, Q(s', a; \theta); \theta^-)$$

- increased stability $\rightarrow$ more complex problems

- So far: All experience is treated equally
- But: From some you can learn more than from others.
- Idea: Prefer those transitions that *fit* the current estimation of the Q-function *the least*.

  $\rightarrow$ greates potential

Prioritised Replay Memory

- The error of sample $S = (s, a, r, s')$ is the distance between $Q(s, a)$ and its target $T(S)$.

$$error = |Q(S, a) - T(S)|$$

- The error is saved together with S and updated during each learning step.

  $\rightarrow$ Greedy Prioritisation

- Samples with a small TD-error during their first occurrence have a very small probability of being sampled ever again.

Stochastic Prioritisation

- Priority:

$$p_i = \frac{1}{rank(i)}$$

- *rank*: position in the buffer (sorted by TD-error)
- Priority is converted to a probability:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- α determines how much prioritisation is used $\alpha = 0 \rightarrow$ standard case ($0 \leq \alpha \leq 1$)

Proportional Prioritisation

- error $\rightarrow$ priority:

$$p_i = (error_i + \epsilon)^{\alpha}$$

- ϵ: small positive constant $\rightarrow$ no priority of 0
- α determines how much prioritisation is used $\alpha = 0 \rightarrow$ standard case ($0 \leq \alpha \leq 1$)
- Priority is converted to a probability:

$$P(i) = \frac{p_i}{\sum_k p_k}$$

Annealing Bias

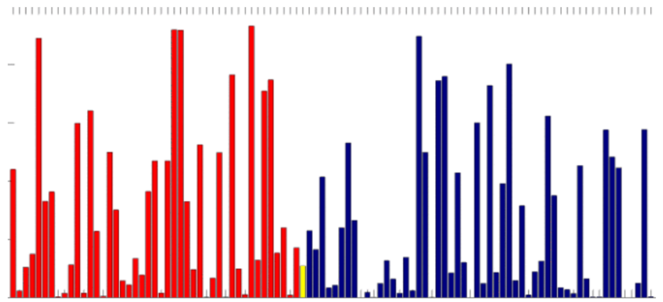
- uncontrolled distribution change $\rightarrow$ induced bias
- Solution: Weighted Importance Sampling

$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)}\right)^{\beta}$$

- β: linearly towards 1
- When $\beta = 1$ the weights completely compensate the uneven distribution $P(i)$.

Sampling

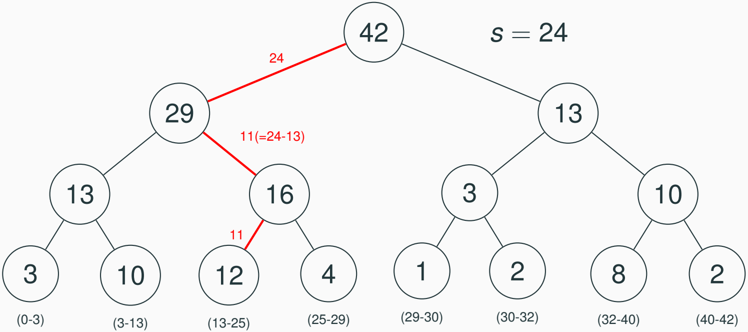
- draw a random number s with $0 \leq s \leq \sum_k p_k$
- traverse the memory from left to right
- sum up the priorities
- stop when s is reached



runtime: $O(n)$

Sum Tree

- Unsorted Sum Tree $\rightarrow$ a binary tree
- value of the parent $=$ the sum of its children
- samples in the leaves



runtime: $O(\log n)$

Advantage

- Q-value: quality of action a in state s

$$Q(s, a) = V(s) + A(s, a)$$

- $V(s)$: How good is it to be in state s?
- $A(s, a)$: How much better is it to chose action a over all others?
  $\rightarrow$ Advantage

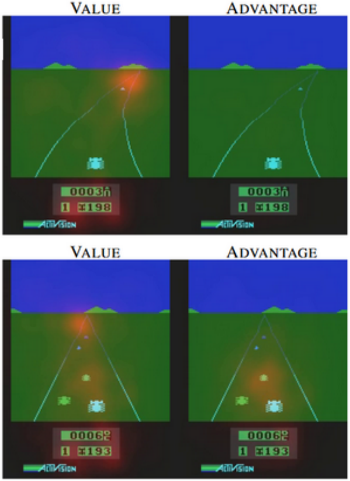
Duelling DQN

- The Network calculates $V(s)$ and $A(s, a)$ seperately and combines them to $Q(s, a)$.
- The agent is not necessarily interessted in both values.
- Why calculating the values for all actions, if they all lead to death?

  $\Rightarrow$ more robust estimation of $V(s)$

EXAMPLE

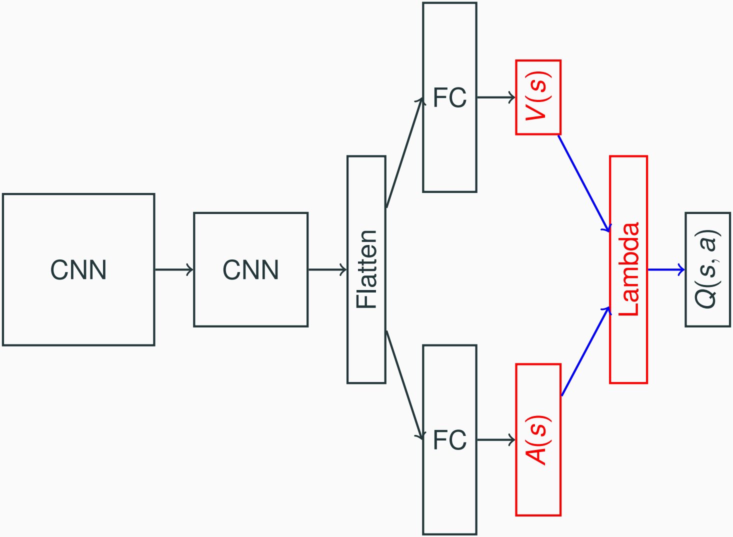
<u>Focus on 2 things:</u>
- The horizon where new cars appear
- On the score



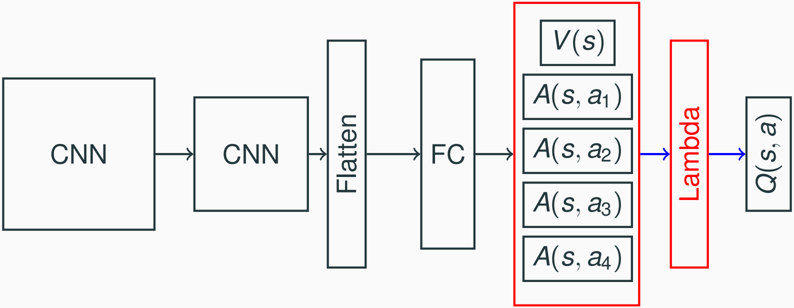
No car in front, **does not pay much attention because action choice making is not relevant**

Pays attention to the front car, in this case **choice making is crucial to survive**

ARCHITECTURE



Architecture



Naïve approach:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

BUT: problem of identifiability

$\Rightarrow$ Given $Q(s, a)$ it is not possible to determine $A(s, a)$ and $V(s)$.

$\Rightarrow$ Problem for Backpropagation

AGGREGATION

Solution: Forcing the advantage function to 0 for the chosen action through subtraction of the mean advantage of all actions:

$$Q(s, a; \alpha, \beta) = V(s, \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{A}\sum_{a'} A(s, a'; \theta, \alpha)\right)$$

- α: parameter of the advantage stream
- β: parameter of the value stream

Lamba-Layer

- Keras layer
- To define custom layers
- Example:

```python
from keras.layers import Lambda
from keras.models import Sequential
from keras import backend as K

model = Sequential()

model.add(Dense(16, activation='relu', input_dim=2))

# Calculates the square
model.add(Lambda(lambda x: x ** 2))

# Determines the mean
model.add(Lambda(lambda sq: K.mean(sq))

model.add(Dense(8)
```

Bibliography

Pictures

- https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
- https://view.stern.de/de/rubriken/tiere/hund-hunde-ohren-basset-hound-dumbo-basset-original-2012281.html
- https://www.quora.com/What-do-channels-refer-to-in-a-convolutional-neural-network
- https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050
- https://jaromiru.com/2016/11/07/lets-make-a-dqn-double-learning-and-prioritized-experience-replay/
- https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682