

Learning for Autonomous Vehicles – Neural Networks and Reinforcement Learning

TSFS12: Autonomous Vehicles – Planning, Control, and Learning Systems

Lecture 10: Erik Frisk <erik.frisk@liu.se>

Purpose and take-aways from this Lecture

- Provide an introduction (focus on usage of methods) to some core methods in the field of learning for autonomous vehicles:
 - Neural networks,
 - Reinforcement learning.
- Be familiar with how neural networks can be used for learning.
- Have basic knowledge about the formalism of Markov decision processes and basic methods for solving reinforcement-learning problems in discrete time and finite state and action spaces.

Literature Reading

The following book and article sections are the main reading material for this lecture. References to further reading are provided throughout the slides and at the end of the lecture slides.

- Sections 11.2-11.8 in Hastie, T., R. Tibshirani, J. Friedman, & J. Franklin: The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd Edition, Springer, 2005.
- Sections 1, 4.1-4.4, and 6.1-6.5 in Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.
- Scan the content of Mnih, V., Kavukcuoglu, K., Silver, D. et al: "Human-level control through deep reinforcement learning", Nature 518, 529–533, 2015.

Outline of the Lecture

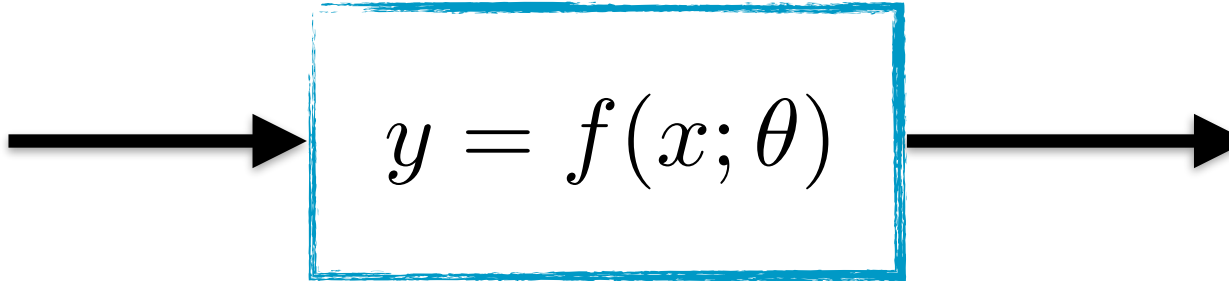
- Learning using neural networks.
- Introduction to reinforcement learning.
- Combining neural networks and reinforcement learning.
- Pointers to some useful software libraries for machine learning.

Learning Using Neural Networks

Introduction and Background

Input:

Time-series data,
All pixels in an Image,
Natural language,
Text, etc.

**Output:**

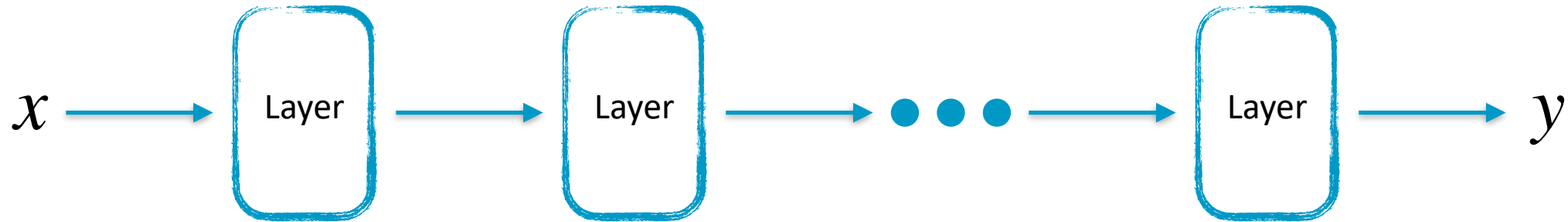
Prediction of output,
or class/category

- A neural network is a nonlinear function approximator (often with multiple inputs and outputs), curve-fitting in high-dimensional spaces.
- Function parameterized using parameters in θ (often high-dimensional vector, with thousand, millions, or billions of elements).
- Typically needs large amount of data $X = (x_1, \dots, x_N)$ to determine the parameters.
- Roughly
 - Classification — discrete output indicating class of input
 - Regression — predict a continuous output

Introduction and background

$$y = f(x; \theta)$$

$$\theta^* = \arg \min_{\theta} L(\theta; y, x)$$



- Basics
 - fairly simple model structure in basic feed-forward networks
 - massively over-parameterized model + regularization techniques
 - With depth comes automated feature discovery
 - Increasingly complex architectures in modern applications
- Non-linear but simple — efficient ways to differentiate model wrt. θ
- Then, minimize loss function $L(\theta; x, y)$ using a first order method (gradient search)

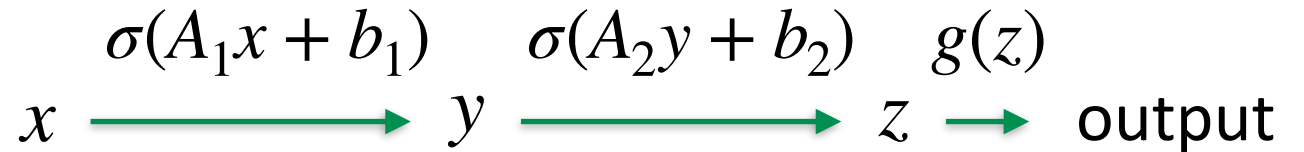
Structure of a Single Hidden-Layer Neural Network

- A neural network consists of hidden layers and an output layer.

- Basic building block

$$\sigma(Ax + b)$$

An affine transformation + a non-linearity called activation function



- Parameters A_i and b_i
- The simplest version contains a single hidden layer.
- There are of course more complex layers

Some Example Activation Functions

- Linear combination of the inputs.
- Sigmoid (approximation of a step function):

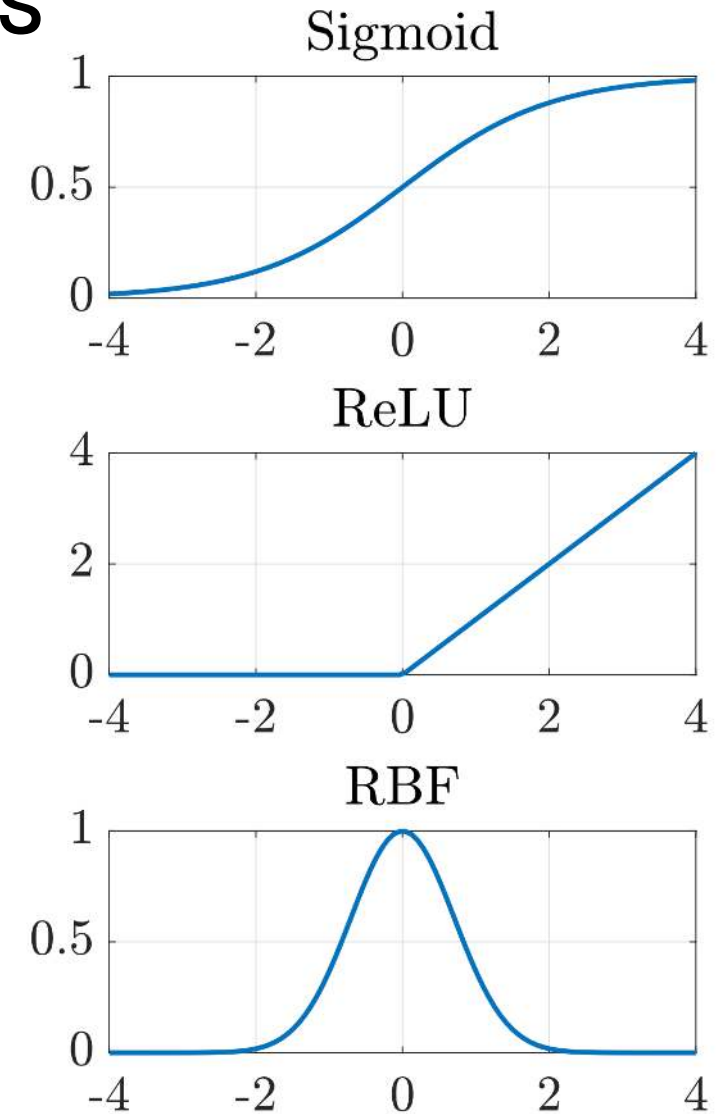
$$\sigma(v) = \frac{1}{1 + \exp(-v)}$$

- Rectifier (Rectified Linear Unit – ReLU):

$$\sigma(v) = \max(0, v)$$

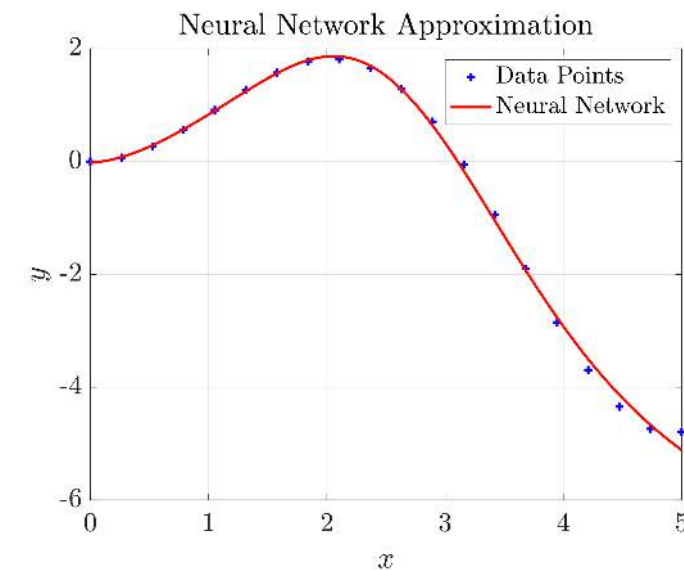
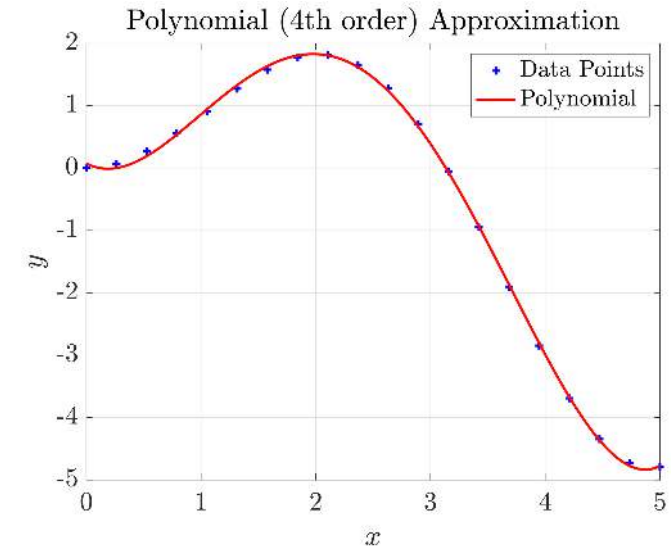
- Gaussian radial basis function (RBF):

$$\sigma(v) = \exp(-\gamma \|v - c\|^2)$$



Choices of Output Function (1/2)

- For regression problems (i.e., finding relations between dependent and independent variables), the output function can be linear
$$g_k(T) = T_k$$
- Compare with least-squares approximation of data points using polynomials from previous courses.



Choices of Output Function (2/2)

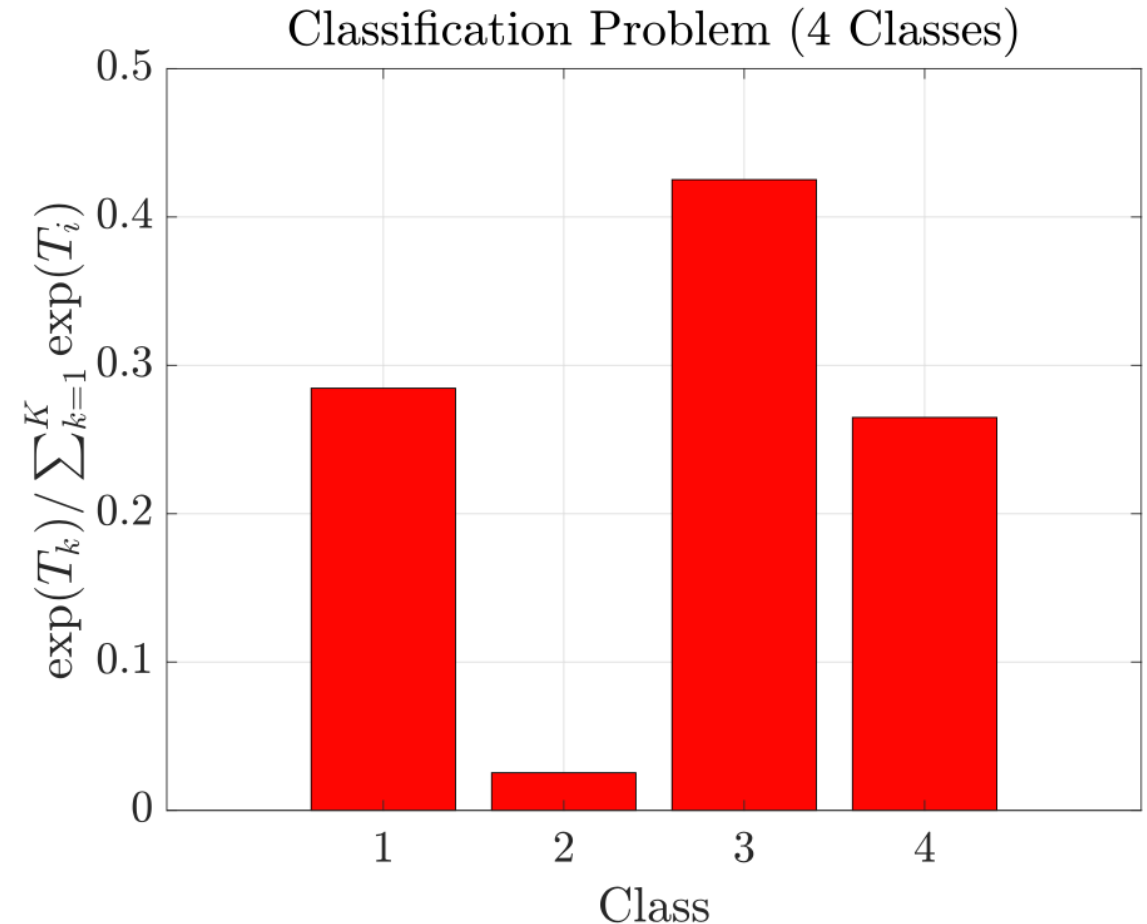
- For classification problems, a common output function is the softmax function

$$g_k(T) = \frac{\exp(T_k)}{\sum_{i=1}^K \exp(T_i)}$$

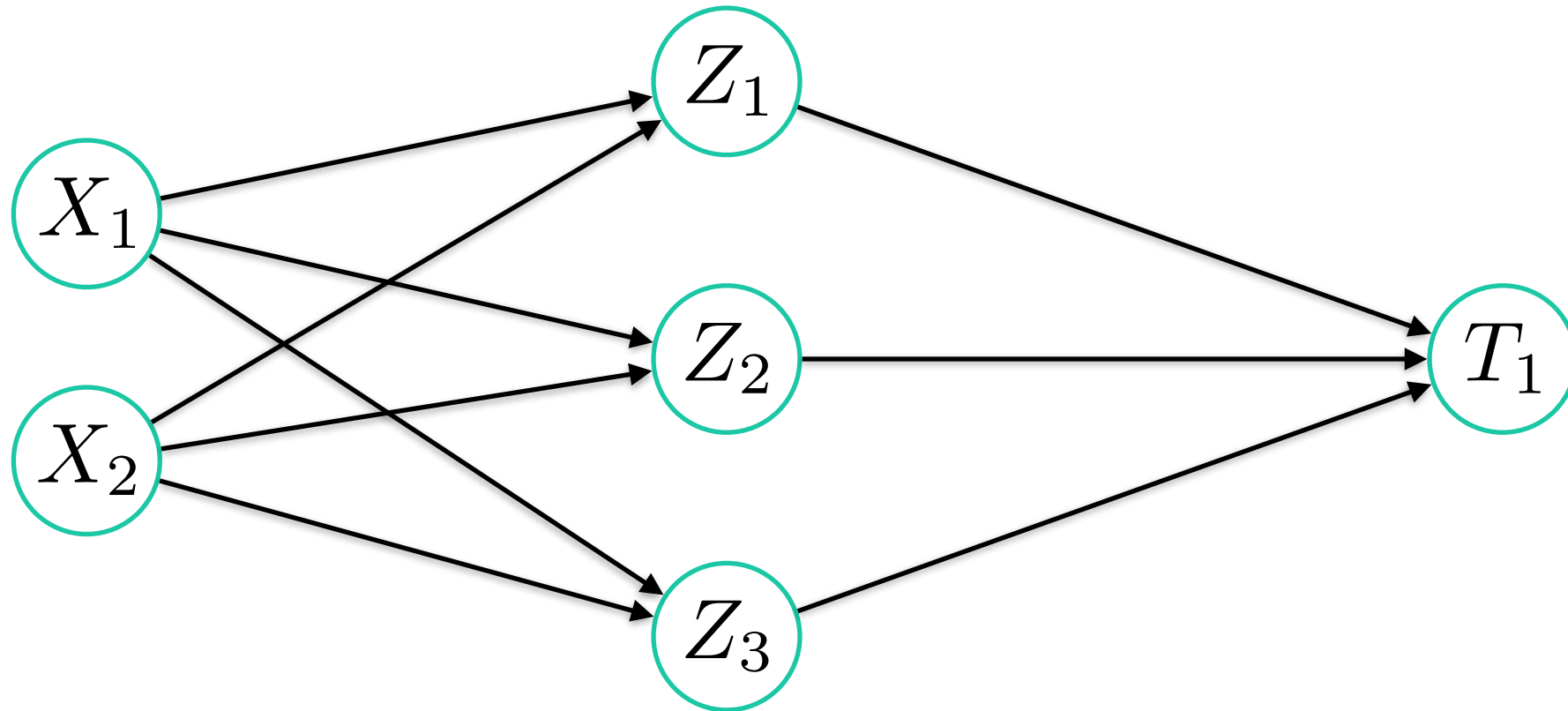
- Has the property

$$\sum_{i=1}^K g_k(T) = 1$$

\approx probability of a class



Example of a Neural Network



$$\sigma(\alpha_{0,m} + \alpha_m^T X)$$

$$\beta_{0,k} + \beta_k^T Z$$

Training of a Neural Network

- Use training data to determine the parameters θ of the model.
- The parameters can be computed by minimizing a cost function – an optimization problem.
- Number of data-points N can be a very large number
- Gradient descent
 - find a local minimum using $\nabla_{\theta} J(\theta)$
 - A function decreases most rapidly in the negative direction of the gradient

Examples of cost functions

Squared-error cost function (regression):

$$J(\theta) = \sum_{i=1}^N (y_i - f(x_i; \theta))^2$$

Cross-entropy (log-likelihood)
cost function (classification in K classes):

$$J(\theta) = - \sum_{i=1}^N \sum_{k=1}^K y_{i,k} \log(f_k(x_i; \theta))$$

Training of a Neural Network

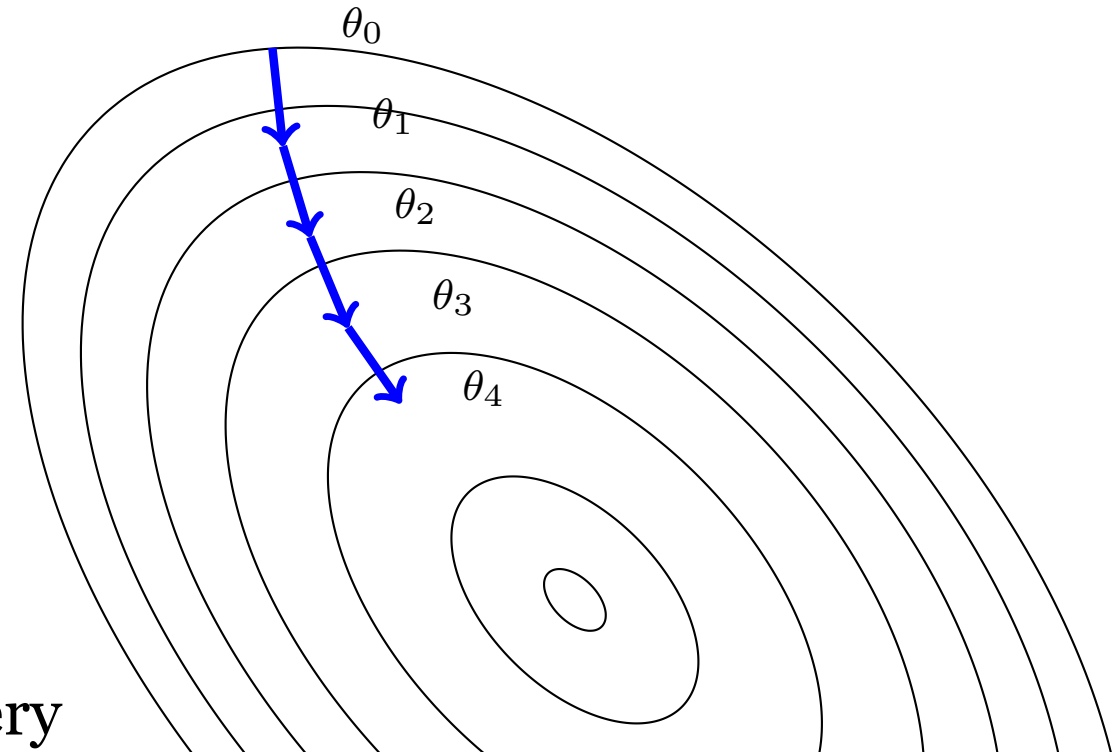
- Iterative method for moving towards a local minimum:

$$\theta_{i+1} = \theta_i - \gamma \nabla J(\theta_i)$$

- γ - learning rate
- θ_0 - initial guess
(often a random number)
- The **problem**: If N is very large

$$\nabla J(\theta) = \sum_{i=1}^N \frac{\partial}{\partial \theta} (y_i - f(x_i; \theta))^2$$

- Infeasible to compute this in every iteration



Stochastic Gradient Descent (SGD)

- Computation of the gradient becomes a challenge when the number of data points becomes large; time-consuming to compute one iteration.
- Stochastic gradient descent is one way to address this:
 - Randomly select a batch of the data points (in the limit only one) and perform gradient descent.

- Idea for reducing size of parameter-optimization problem:

$$\nabla \sum_{i=1}^N (y_i - f(x_i; \theta))^2 \approx \nabla \sum_{i=1}^n (y_i - f(x_i; \theta))^2, \quad n \ll N$$

- Trade a better step for a cheaper step and iterate many times

Backpropagation, computing derivatives

- Recall the automatic differentiation method for computing derivatives from Lecture 5.
 - Forward or backward/adjoint mode
 - Forward mode when $\#inputs \geq \#outputs$.
 - Backward/adjoint mode when $\#inputs \ll \#outputs$.
- For neural networks, the second case is typical. Input is sequence of, e.g., images and output is a classification
- Backpropagation, a reverse-mode method for efficient computation of the derivatives utilizing the chain rule

Training and optimizing

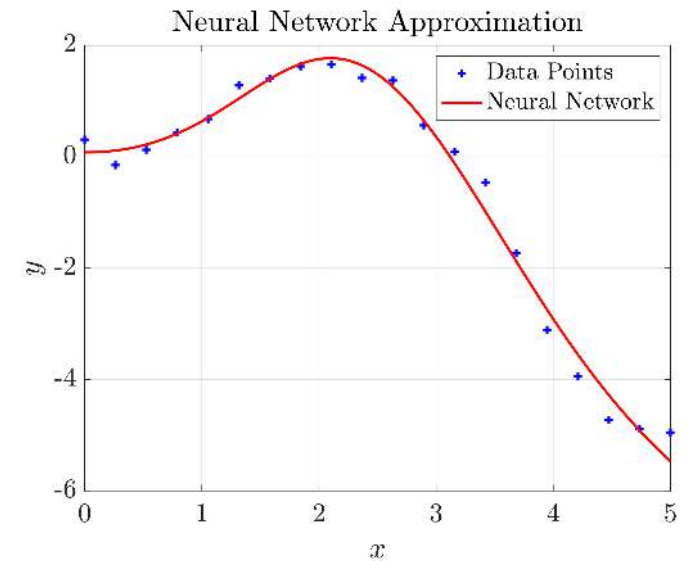
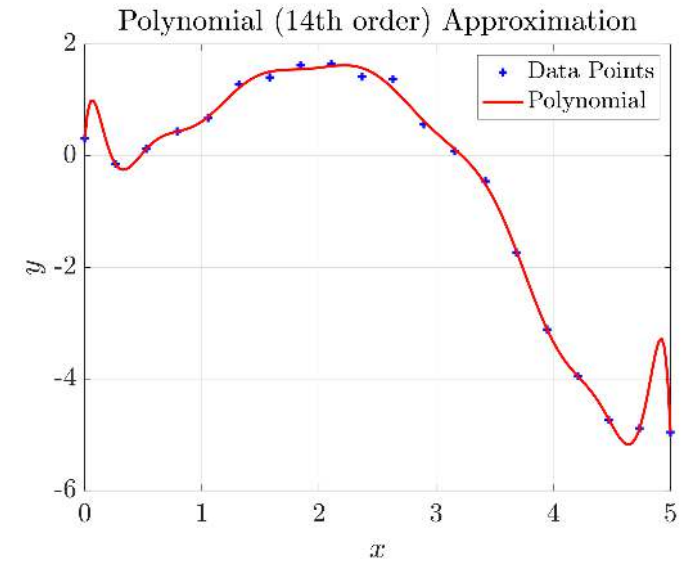
- Several extensions and variants of stochastic gradient descent exist for improved performance (e.g., momentum acceleration, Nesterov accelerated gradient, AdaGrad, RMSProp, and Adam, ...).
- Overfitting of model parameters to training data is a common challenge in any function approximation.
 - With over-parameterized models, easy to overfit to the training data
 - learning noise realizations in data and not model the true function

Overfitting of Model Parameters

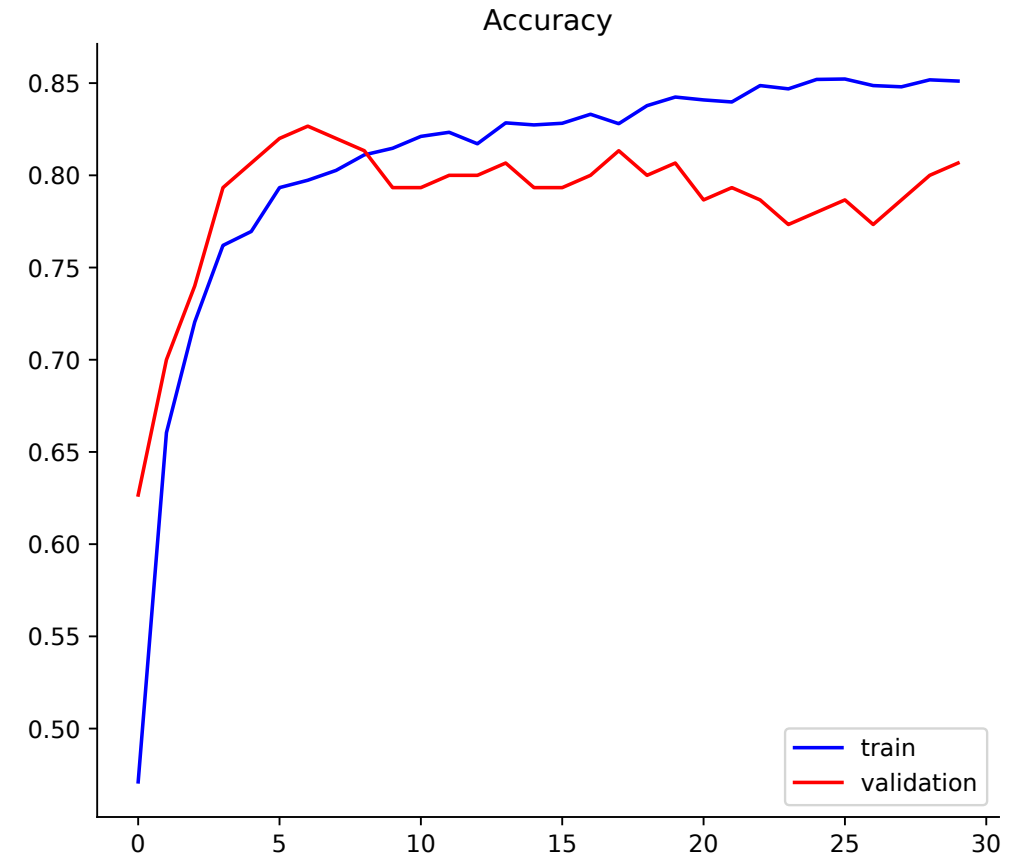
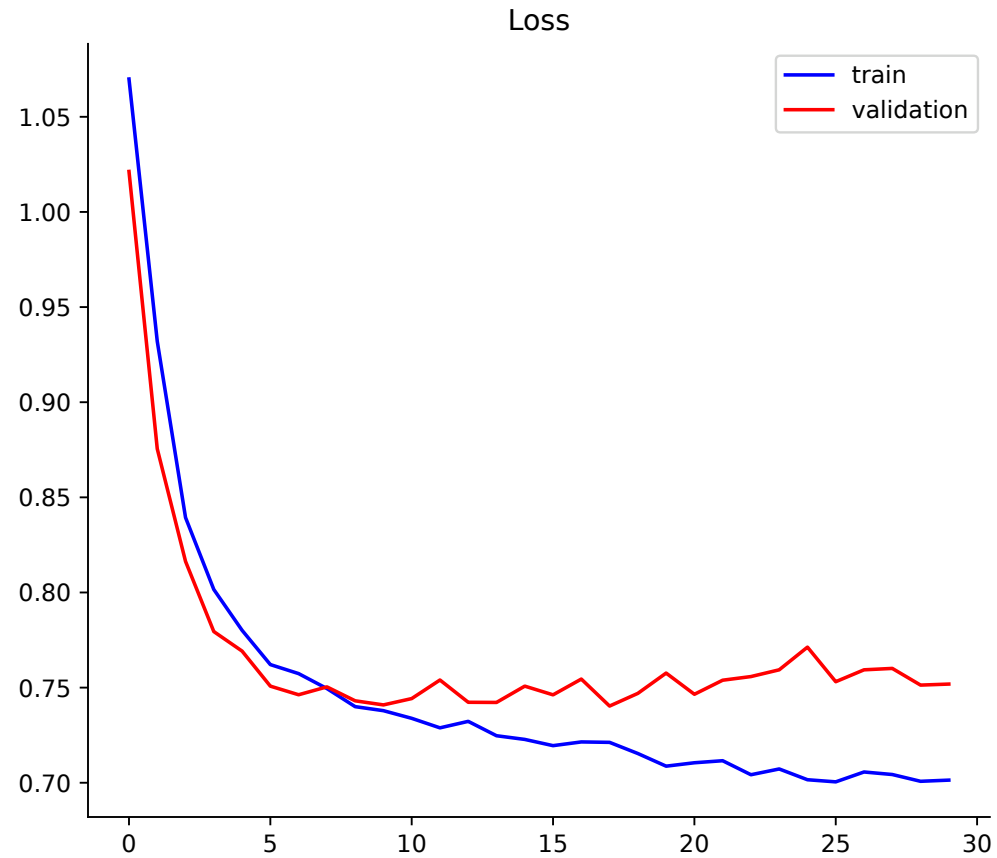
- Example: Fit a high-order polynomial model (14th order) and a neural network model (30 hidden neurons), respectively.
- 20 data points, where noise from a Normal distribution with standard deviation 0.2 has been added, from the function

$$x \sin(x), \quad x \in [0, 5]$$

- Clear overfitting in polynomial model.



Overfitting



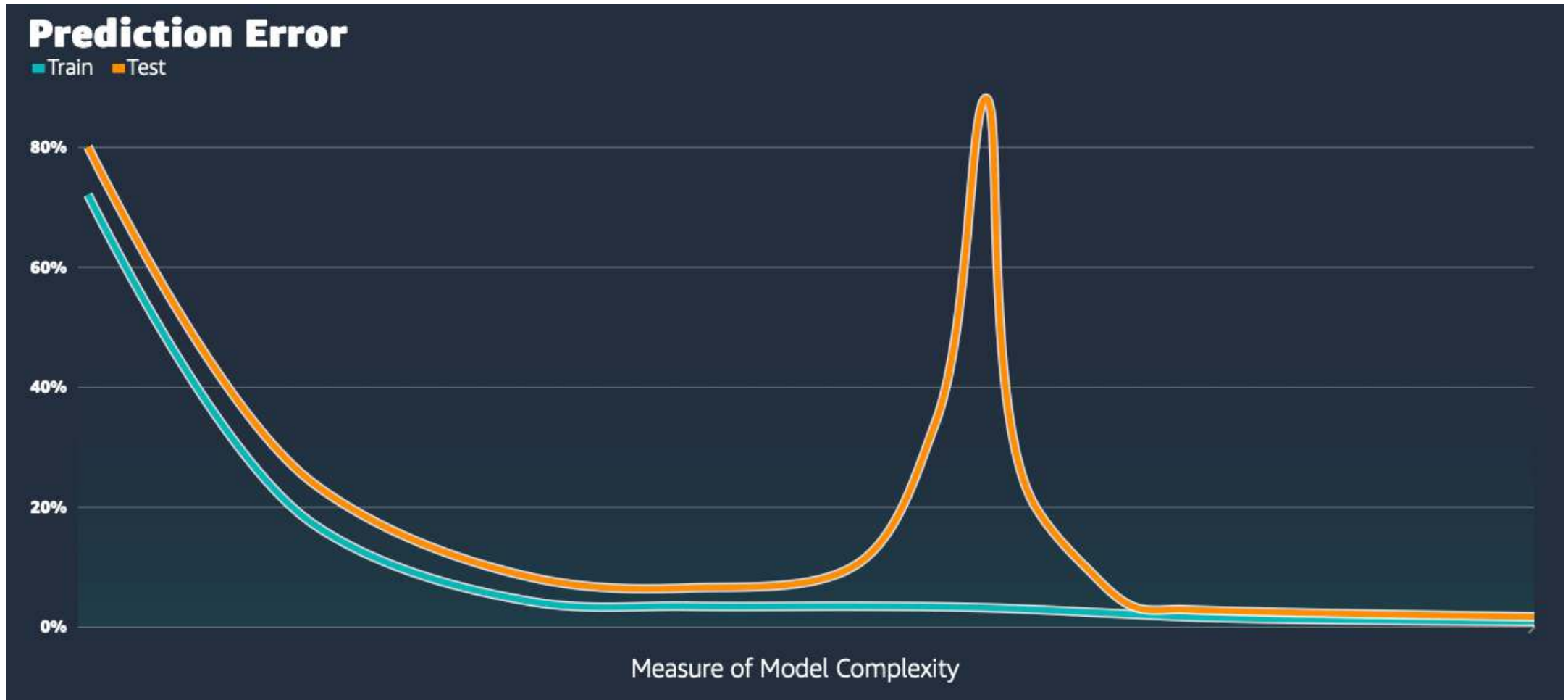
Overfitting of Model Parameters

- Methods used for mitigating overfitting:
 - Early termination in the SGD.
 - Regularization terms in cost function, e.g.,

$$\lambda \sum_i \theta_i^2, \text{ where } \lambda \text{ is a weight parameter}$$

- Empirical methods to avoid overfitting are, e.g.,
 - Dropout (randomly disconnect a subset of the nodes in each phase of the training and then re-connect them again).

Not a simple topic though ... double descent phenomena²¹

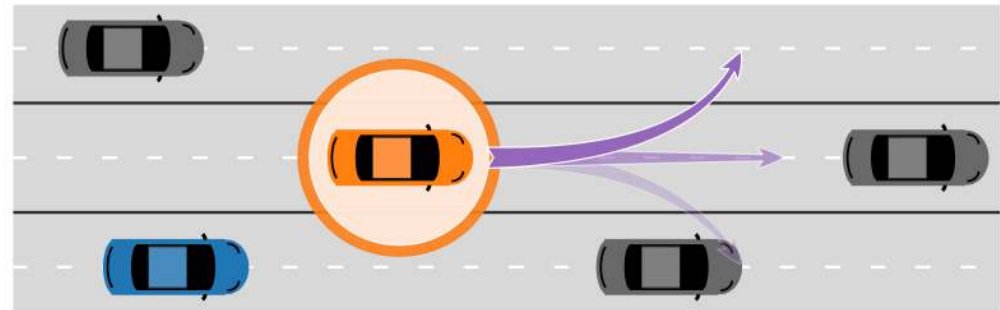


Imbalanced Learning

- Imbalance in the training data for a neural network aimed at classification can result in a biased classifier.
 - Example is when data from one or more classes are overrepresented compared to other classes.
- One direct approach to remedy this is to weight underrepresented classes when creating the training data set by sampling (with replacement) from the actual data set resulting in a balanced data-set

Imbalanced Learning - Example

- Example: Assume that the task is to predict if a certain vehicle will change lane within the next few seconds.
- If the training data consist of 99 % cases driving forward, a model always predicting moving forward would have an accuracy of 99 % for that data set.
- Clearly such a predictor is not satisfactory, since the important lane-change situations will never be predicted – imbalance in data.

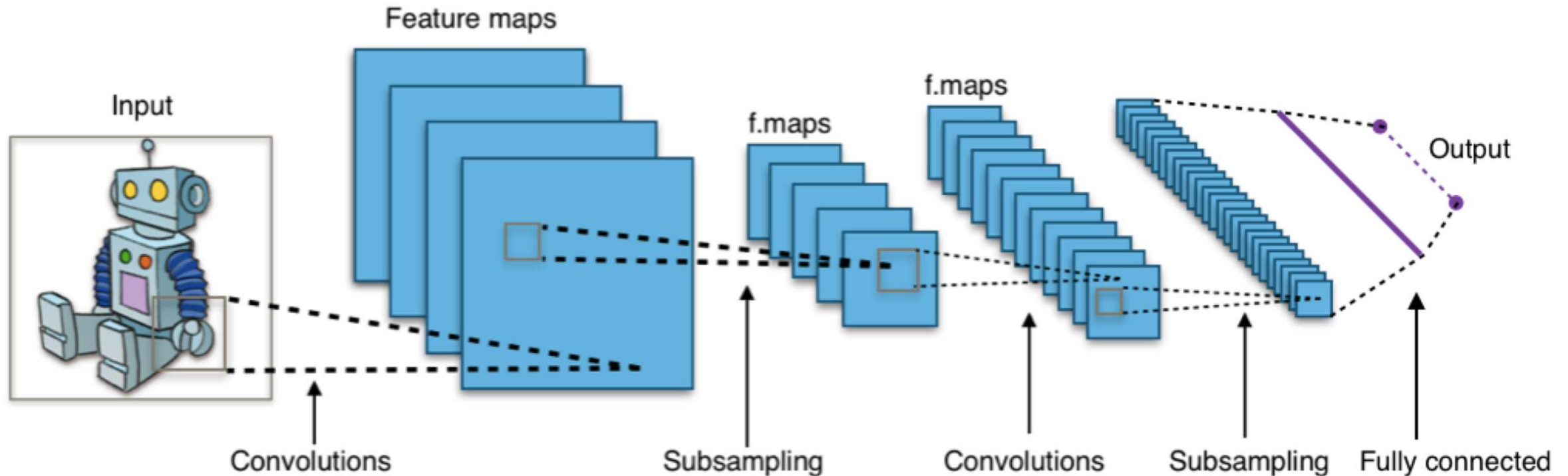


Deep nets and other architectures

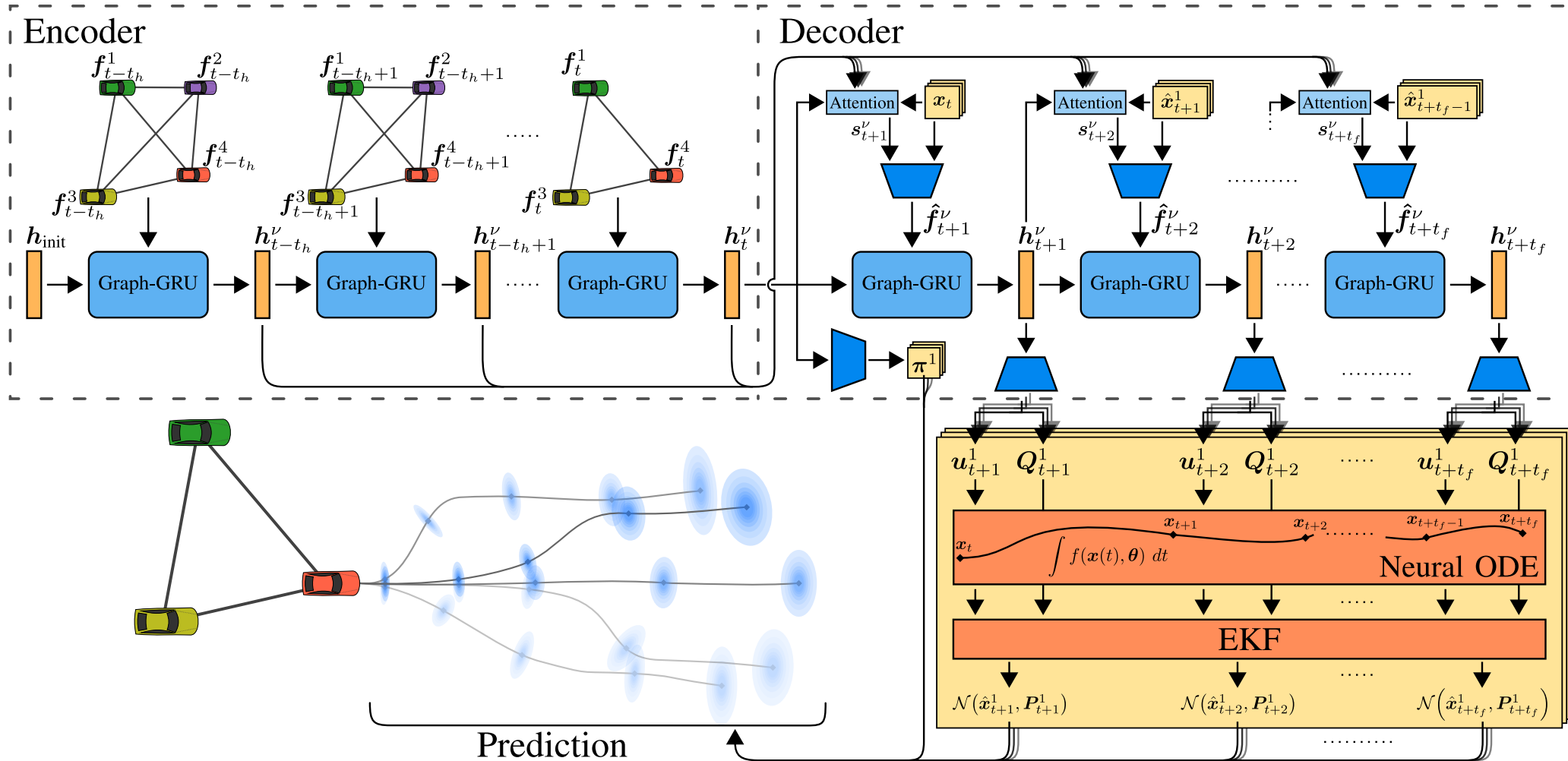
- Deep networks comprise multiple hidden layers (10/100/1000)
 - Idea: Features in the training data captured by the neural network.
- Convolutional neural network includes convolution operators
 - Local spatial information is modeled through the convolutions.
 - Spatial invariance (cat in the top left and in the bottom right)
- Recurrent neural network with feedback loops modeling local memory
 - Text and time-series analysis
 - Typically difficult to train with slow convergence
 - modern approaches include transformer architectures to address this issue

Example Structure of Convolutional Neural Networks

- A recurrent neural network does not only propagate the inputs forward, but also allows internal feedback loops (local memory).



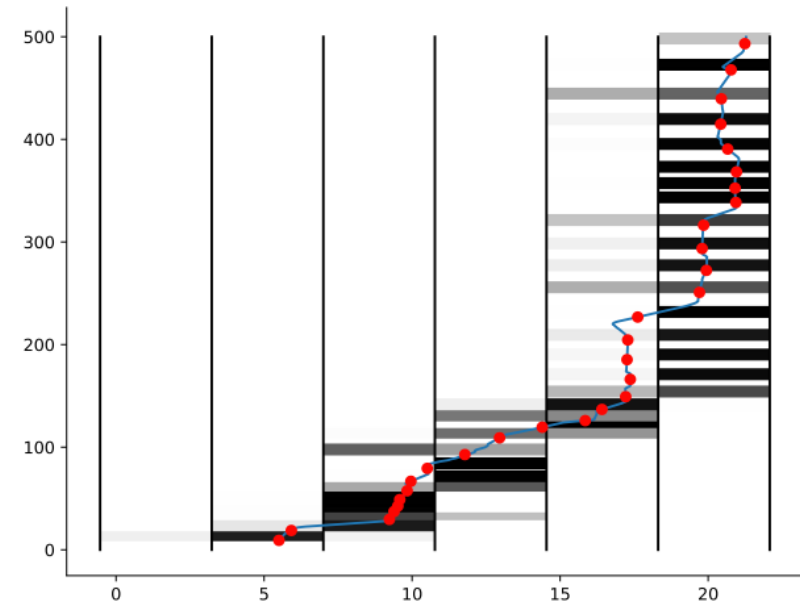
Recurrent Graph Neural Networks and Neural ODEs



Hand-in Exercise 5: Neural Network for Intent Prediction²⁷

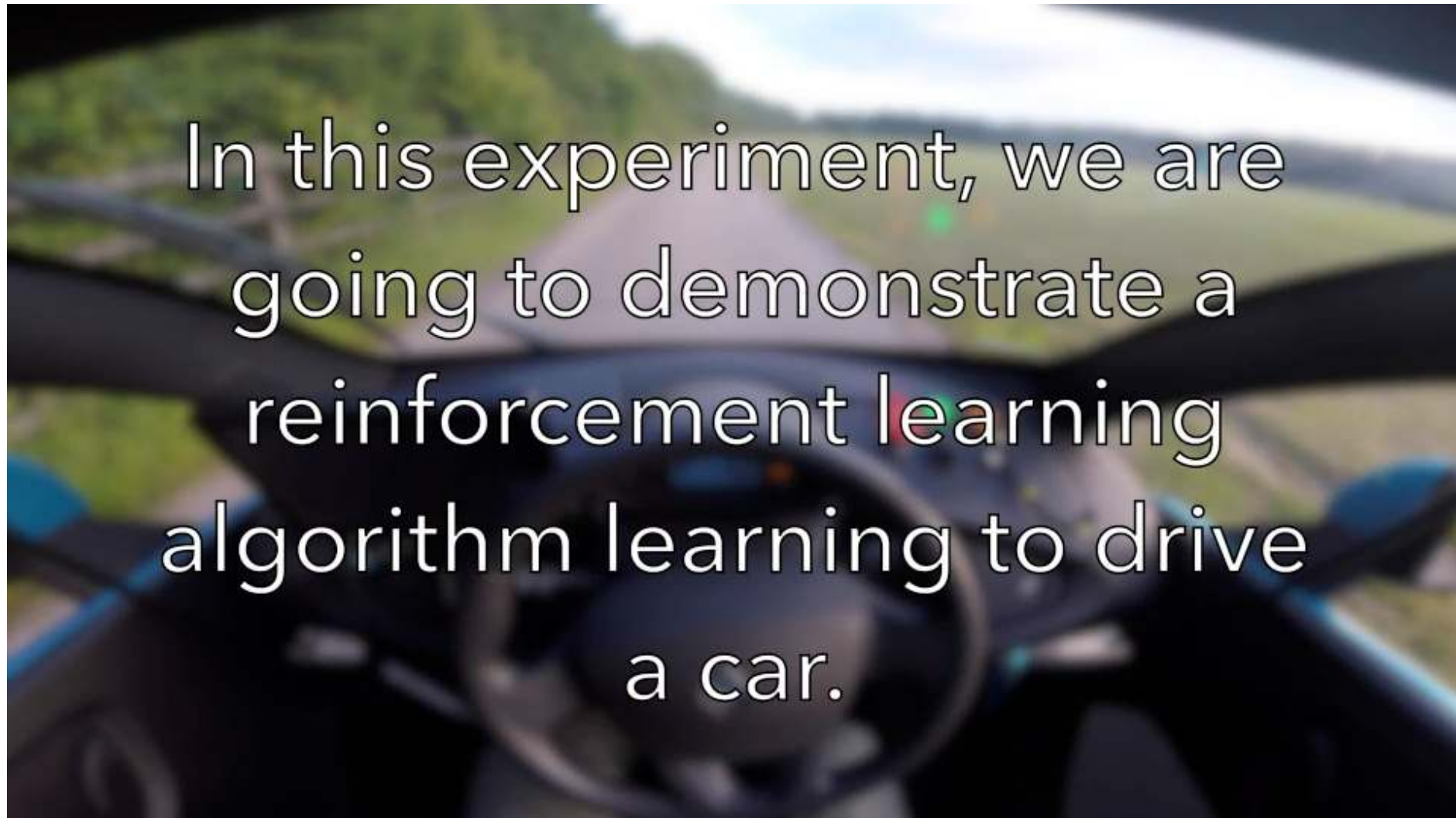
- In Hand-in Exercise 5, lane-change predictions are computed based on driver data from the I-80 highway section in the U.S.
- A neural network is trained as a classifier using 41 features in 4,383 trajectories.

$$y = \begin{cases} 0 & \text{if the vehicle will change to the left within three seconds} \\ 1 & \text{if the vehicle will stay in lane for the next three seconds} \\ 2 & \text{if the vehicle will change to the right within three seconds} \end{cases}$$

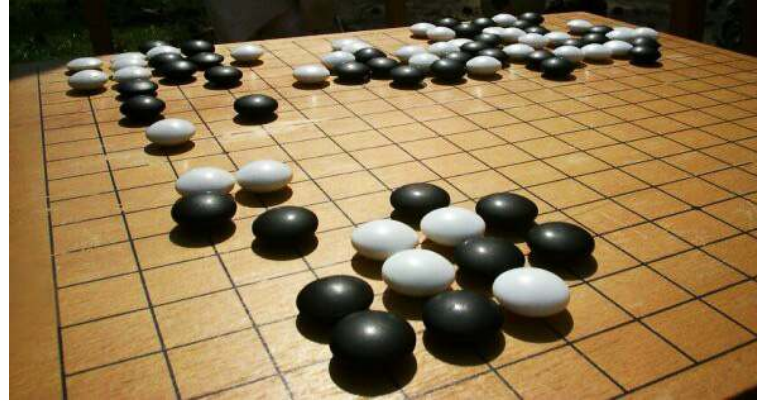


Introduction to Reinforcement Learning

End-to-end (camera-to-control) learning



These are difficult ...



- Reinforcement learning has been very successful in games with a closed-world, i.e., there are distinct rules to the game
- AlphaZero, starting from a random acting player, with a couple of hours of self-play beats any human player in chess. Does this entirely by self-play.
- There are ideas, primarily in learning/vision community, that this also extends to problems like autonomous driving
- Q: How relevant are the successes of AlphaZero for cyber-physical systems?

(Optimal-) Control

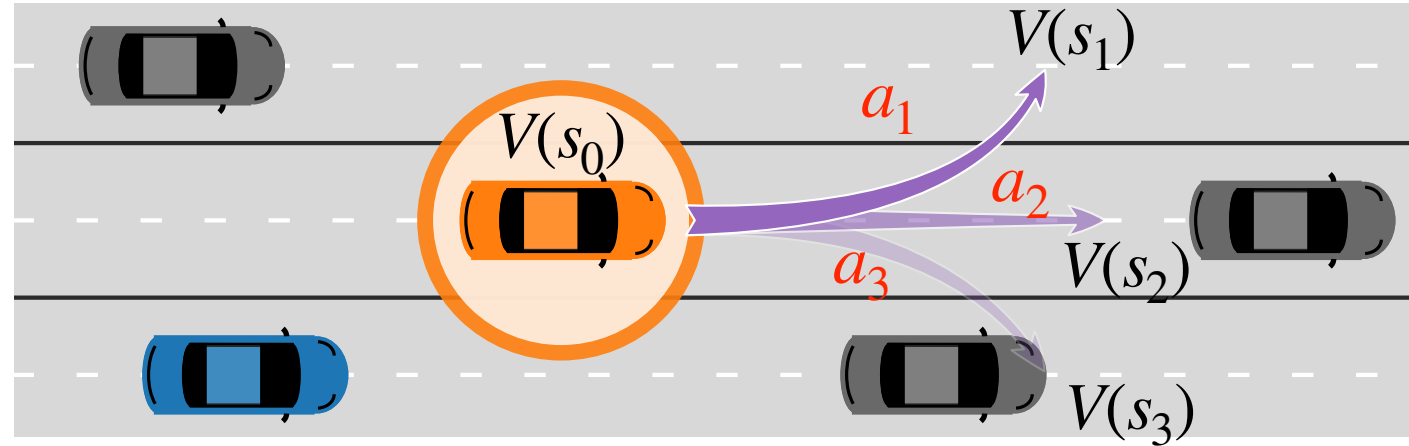
- To (optimally) control the vehicle, it would be great to know the *value* $V(s)$ of some state s , or *quality* $Q(s, a)$ of an action a in state s .
- With a model that tells us next state and the cost of that action

$$s_{next} = f(s, a)$$

a control strategy is then to choose the control action that optimizes $V(s_{next})$

$$\pi(s) = \arg \max_a Q(s, a)$$

- With a model for dynamics and cost-function, then, DP (and limited state-space) or MPC for a fixed-horizon possible solutions
- But what if valuating a state or doing predictions is really, really, hard?



$V(s)$ = value of state s

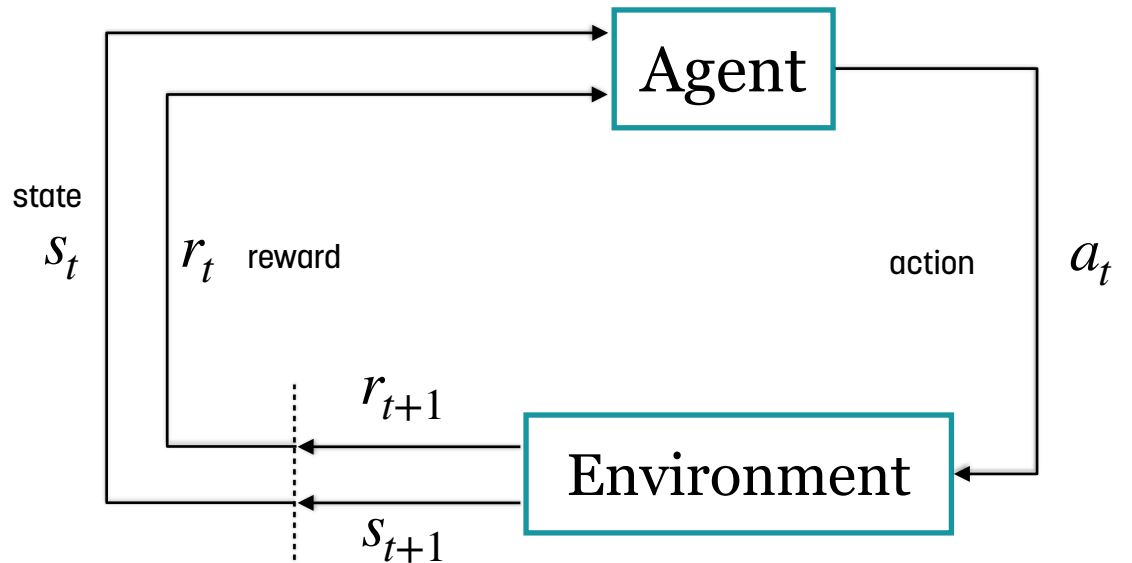
$Q(s, a)$ = value (quality?) of action a in state s

Learning by doing – reinforcement learning

- If we have no model but can probe the system and get feedback, a reward. Then we can learn something.
- Reinforcement learning is learning what to do by *maximizing the reward* without being told what to do.
- Learn the policy/controller by exploring and evaluating the outcome, iteratively refine the policy/controller

$$a_t = \pi(s_t) / u_t = g(x_t)$$

- There are many different techniques, and this glimpse I will focus on value-based approaches, based on approximations of $V(s)$ and/or $Q(s, a)$. Suitable for a discrete action space.
- Self-play



A First Example – K-Armed Bandit (1/4)

- A classical first example in reinforcement learning is the so called k-armed bandit problem.
- Consider choosing between $k = 3$ different actions at every time step corresponding to 3 different arms on a slot machine.
 - Each action results in a reward according to an a priori unknown normal distribution with constant mean and variance.
 - Arm 3 is actually a little better
- The objective is to maximize the accumulated reward over a specified number of time steps.

A First Example - K-Armed Bandit (2/4)

- Let us try to decide on the optimal action (policy) by interacting with the 3-armed bandit, i.e., sequentially choosing and applying actions and observing the rewards obtained.
- Introduce a function that measures the value of a particular action:

$$Q_t(a) = \frac{\text{sum of rewards when action } a \text{ chosen}}{\text{total number of times action } a \text{ chosen}}$$

- Actions chosen according to an epsilon-greedy policy, where a random action is chosen with probability ε (exploration) and else the currently greedy action (exploitation) is chosen as

$$a_{\text{greedy}} = \arg \max_{a'} Q_t(a')$$

A First Example - K-Armed Bandit (3/4)

Iteration	$Q_t(a_1)$	$Q_t(a_2)$	$Q_t(a_3)$	$R_t(a_1)$	$R_t(a_2)$	$R_t(a_3)$	a	Type
1	4.8510	6.1812	5.2415	4.8510	6.1812	5.2415	1,2,3	Initial
2	4.8892	<u>6.1812</u>	5.2415	4.9273			1	Explore
3	4.8892	<u>5.6798</u>	5.2415		5.1784		2	Exploit
4	4.8892	5.6487	5.2415		5.5864		2	Exploit
5	4.8892	<u>5.6487</u>	4.8661			4.490	3	Explore
6	4.8892	<u>5.4258</u>	4.8661		4.7572		2	Exploit
7	4.8892	4.9476	4.8661		3.0346		2	Exploit
8	4.8892	4.9476	<u>5.9204</u>			8.029	3	Explore
9	4.8892	4.9476	<u>6.0912</u>			6.603	3	Exploit
10	4.8892	4.9476	6.4277			7.773	3	Exploit
11	4.9536	4.9476	6.4277	<u>5.0826</u>			1	Explore
12	4.9536	4.9476	6.1285			4.632	3	Exploit

A First Example - K-Armed Bandit (4/4)

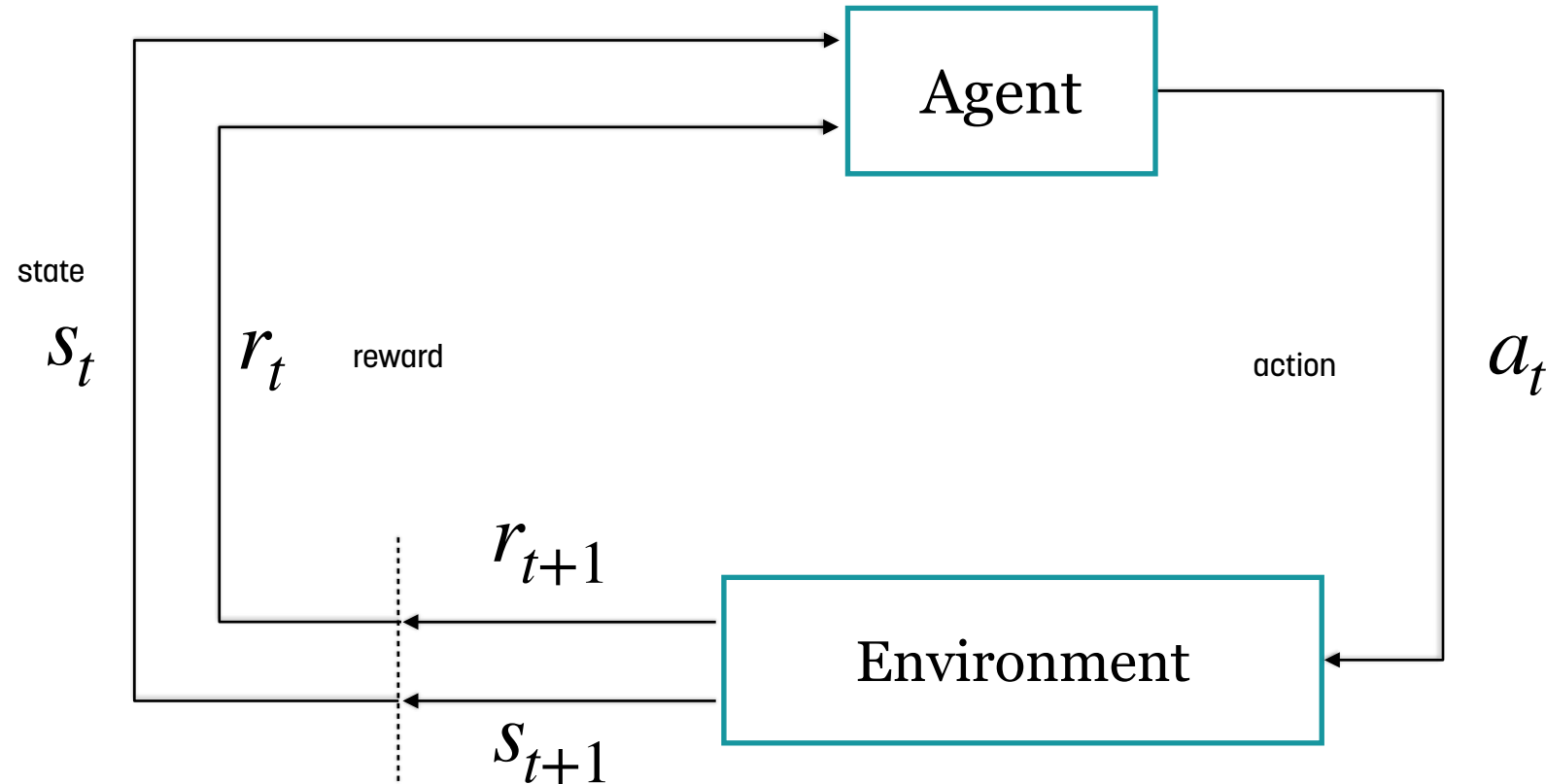
- The actual parameters of the normal distributions of the 3-armed bandit example are (mean and standard deviation):

Action 1: $\mu_1 = 5.5$, $\sigma_1 = 1$ Action 2: $\mu_2 = 5$, $\sigma_2 = 1$ Action 3: $\mu_3 = 6$, $\sigma_3 = 1$

- The example introduces many key features of reinforcement learning:
 - Exploration/exploitation to both gain and use information, estimate value of an action by repeated interactions by the agent with the environment, and stochastic uncertainty in the world.
- In this example, the reward is not dependent on a state. This will be considered in the formal definition of a Markov decision process.

The Agent-Environment Interaction Model

- Searching for a policy (control law) on how to act in each state.



Definition of Concepts

- Agent – the controller subject to learning.
- Environment – the agent interacts with the surroundings (controlled system).
- Action $a_t \in \mathcal{A}(s)$ – control signal decided by the agent.
- State $s_t \in S$ – describes all relevant aspects of the environment.
- Reward $r_t \in \mathcal{R}$ – numerical value given by the environment and received by the agent as a result of the action taken.

The problem

[significantly simplified Markov Decision Process, deterministic case]

- The objective is to find a policy/controller

$$a_t = \pi(s_t)$$

maximizing the (discounted) accumulative return, the state-value function $V_\pi(s)$

$$V_\pi(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- The discount factor $0 < \gamma \leq 1$ determines how “greedy” the policy should be.

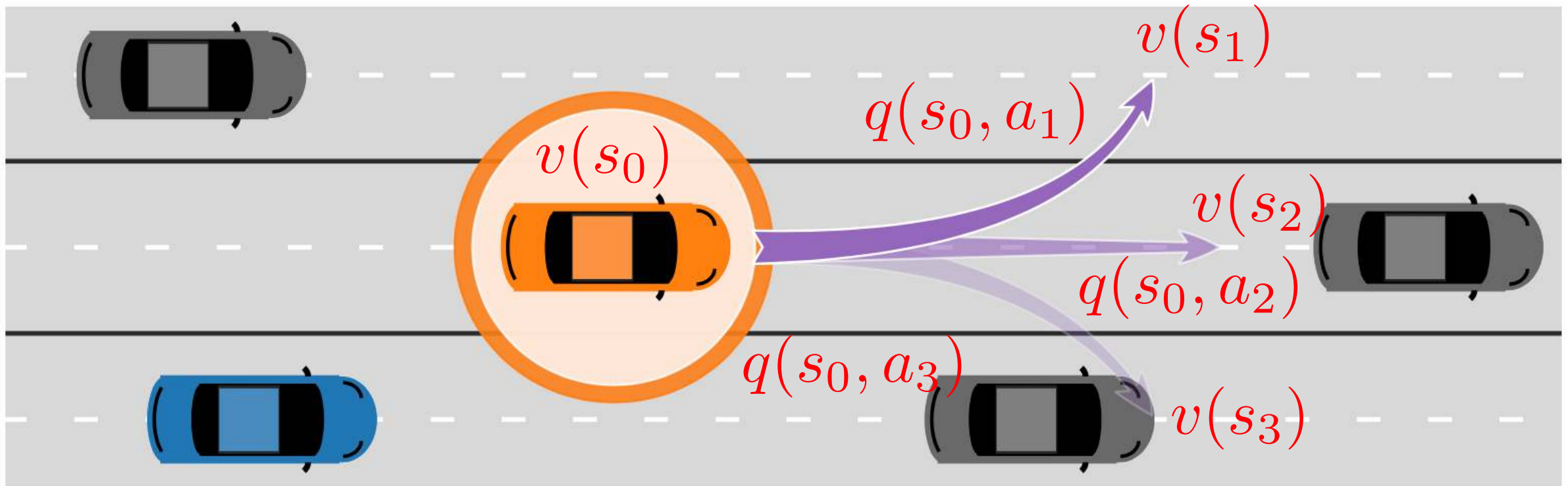
- The action-value function $Q_\pi(s, a)$ is consequently defined, for policy $\pi(s)$, as

$$Q_\pi(s, a) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \text{ when } s_t = s, a_t = a \text{ and then policy } \pi(s) \text{ is used}$$

- For a given $Q(s_t, a_t)$, the greedy policy is

$$\pi(s) = \arg \max_a Q(s, a)$$

State and Action-Value Functions - Example



Markov Decision Process (MDP) - probabilistic case

- Policy defines action to be taken

$$a = \pi(s), \quad \text{or} \quad \pi(a|s) = P(A_t = a | S_t = s)$$

- The state-value function defines expected return, given policy

$$v_{\pi}(s) = E_{\pi} \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right)$$

- The action-value function defines the expected value of an action in a certain state

$$Q_{\pi}(s, a) = E_{\pi} \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right)$$

The Markov Assumption

- The Markovian assumption implies that all information needed is contained in the current state.

- In terms of probabilities, this can be expressed as

$$P(s_t, r_t | a_{t-1}, s_{t-1}, \dots, a_0, s_0) = P(s_t, r_t | a_{t-1}, s_{t-1})$$

- This assumption is fundamental in the formulation of the MDP, since state and action-value functions depend only on the current state.
- Significantly simplifies the problem

The Bellman Optimality Equations

- The Bellman optimality equations define recursive relationships for value function and action-value functions.

- Optimality equation for the value function ($s \rightarrow s'$)

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) = \max_a \sum_{s', r} p(s', r | s, a) (r + \gamma v_*(s'))$$

- Optimality equation for the action-value function ($s \rightarrow s'$)

$$\begin{aligned} q_*(s, a) &= \mathbb{E} \left(R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a \right) \\ &= \sum_{s', r} p(s', r | s, a) (r + \gamma \max_{a'} q_*(s', a')) \end{aligned}$$

The Bellman Optimality Equations - (simplified deterministic case)

- Optimality equation for the value function (policy gives $s \rightarrow s'$, with reward r)

$$v^*(s) = r + \gamma v^*(s')$$

- Optimality equation for the action-value function (action a then greedy policy gives $s \rightarrow s'$)

$$q^*(s, a) = r + \gamma v^*(s') = r + \gamma \max_{a'} q^*(s', a')$$

- These two relations, intuitive when you see it, are the basis for
 - Value iterations
 - TD-learning/Q-learning

Value Iteration towards an Optimal Policy

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) = \max_a \sum_{s', r} p(s', r | s, a) (r + \gamma v_*(s'))$$

Algorithm 1: Value Iteration

```

1  Initialize  $V(s)$  arbitrarily  $\forall s \in \mathcal{S}$ 
2  repeat
3       $\Delta = 0$ 
4      foreach  $s \in \mathcal{S}$  :
5           $v = V(s)$ 
6           $V(s) = \max_a \sum_{s', r} p(s', r | s, a) (r + \gamma V(s'))$ 
7           $\Delta = \max(\Delta, |v - V(s)|)$ 
8  until  $\Delta < \epsilon$ 
9  return policy  $\pi$  as:
10      $\pi(s) = \operatorname{argmax}_a \sum_{s', r} p(s', r | s, a) (r + \gamma V(s'))$ 

```

Need to know state-
transition function
 $p(s', r | s, a)$

- After convergence, it holds that $V(s) = v^*(s)$.

Unknown Environment ~ Exploration and Exploitation

- Policy and value iterations rely on known state-transition and reward probabilities. What if these are not known?
- Learn the characteristics of the environment by interacting with it.
- Trade-off between exploration (test new actions to investigate the environment) and exploitation (use the information acquired so far and act according to best possible strategy).
- Epsilon-greedy exploration common choice:

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|}, & a = \operatorname{argmax}_a Q(s, a) \\ \frac{\varepsilon}{|\mathcal{A}|}, & a \neq \operatorname{argmax}_a Q(s, a) \end{cases}$$

Greedy policy

Random action

Temporal Difference (TD) and Q-Learning

- Temporal-difference methods try to learn optimal policies without explicit knowledge of the environment and its dynamics.

- Recall the basic relations for optimal $q^*(s, a)$ ($s, a' \rightarrow s'$)

$$q^*(s, a) = r + \gamma \max_{a'} q^*(s', a')$$

- Basic idea: With an approximate $Q(s, a)$, compute the temporal error

$$\Delta_{TD} = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$$

- Then, update, e.g., as in Q-learning

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Delta_{TD}$$

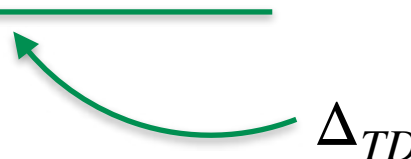
Q-Learning

$$Q(s, a) \approx q_*(s, a)$$

Algorithm 2: Q-Learning

```

1  Initialize  $Q(s, a)$  arbitrarily  $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
2      and initialize  $Q(S_T, \cdot) = 0 \forall$  terminal states
3  repeat for  $K$  episodes
4      Initialize  $S$  to start state
5      repeat
6          Choose action  $A$  from state  $S$  using policy determined from  $Q$ 
7          Take action  $A$ , receive reward  $R$ , and get next state  $S'$ 
8           $Q(S, A) = Q(S, A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S, A))$ 
9           $S = S'$ 
10     until  $S$  is a terminal state
  
```



Δ_{TD}

- Q-learning is called an off-policy TD method; SARSA a common, similar, on-policy method.

Policy-Gradient Methods

- An alternative method to approximating the value or action-value functions is to directly parameterize the policy (possibly along with value function) as

$$\pi(a|s, \theta) = P(A_t = a | S_t = s, \theta_t = \theta)$$

- Then update parameters θ based on some performance $J(\theta)$ metric (compare with cost function from Lecture 5).
- Common methods in this category are REINFORCE and Actor-Critic.

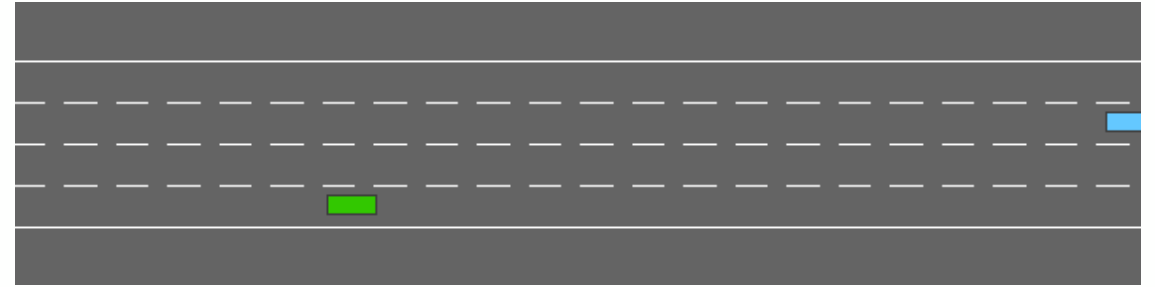
Simulation as a Tool for Reinforcement Learning

- In some environments, it could be challenging to iteratively interact with the environment by actual experiments.
- A simulated environment can therefore be used to train the algorithm (possibly also reducing the time to perform the required experiments).

Combining Neural Networks and Reinforcement Learning – Deep Q-Learning

Let's get "practical"

- Can we use tabular Q-learning to solve this simple (?) game?
- Not that easy to write a good rule-based controller for this scenario
- Rules of the game



- *Actions*: $\mathcal{A} = \{\text{lane_left}, \text{idle}, \text{lane_right}, \text{faster}, \text{slower}\}$
- *Reward*: $r \leq 1$, “drive fast, no collision, and in the rightmost lane”
- *Observation/state*: 5×5 matrix (ignore first column)
First row ego vehicle: (x, y, v_x, v_y)
Other rows: $(\Delta x, \Delta y, \Delta v_x, \Delta v_y)$

ego vehicle: $(x, y, v_x, v_y) = (1, 0.5, 0.416, 0)$

surr. vehicle 1: $(\Delta x, \Delta y, \Delta v_x, \Delta v_y) = (0.133, -0.25, -0.019, 0)$

$$s_t = \begin{pmatrix} 1 & 1 & 0.5 & 0.416 & 0 \\ 1 & 0.133 & -0.25 & -0.019 & 0 \\ 1 & 0.277 & 0.25 & -0.030 & 0 \\ 1 & 0.427 & 0.25 & -0.052 & 0 \\ 1 & 0.583 & -0.25 & -0.020 & 0 \end{pmatrix}$$

Is a tabular $Q(s, a)$ feasible?

- First problem: observations are continuous variables.
 - We could try to discretize but even for this simple problem, the table gets *huge*
 - We have 4 variables and 5 vehicles, assume discretization into {low, medium, high}, then the table for $Q(s, a)$ will have $3^{4 \cdot 5} \cdot 5 = 17.4 \cdot 10^9$ elements.
- Even with this very coarse discretization, problem is infeasible
- A natural idea: Make a functional approximation of the table as

$$Q(s, a) = f(s, a; \theta)$$

or (as here) the slight modification of the function

$$Q(s) = f(s; \theta) \in \mathbb{R}^m, \text{ where } m \text{ is the number of actions}$$

where θ are the model parameters

Q-learning in large state/action-spaces

- A table for $Q(s, a)$ works for small state and action-spaces; exponential growth of table size.

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih^{1*}, Koray Kavukcuoglu^{1*}, David Silver^{1*}, Andrei A. Rusu¹, Joel Veness¹, Marc G. Bellemare¹, Alex Graves¹, Martin Riedmiller¹, Andreas K. Fiedjeland¹, Georg Ostrovski¹, Stig Petersen¹, Charles Beattie¹, Amir Sadik¹, Ioannis Antonoglou¹, Helen King¹, Dharshan Kumaran¹, Daan Wierstra¹, Shane Legg¹ & Demis Hassabis¹

- For cases where it is infeasible to use a tabular Q-function, approximate with a parameterized function, e.g., a neural network

$$Q(s, a) = f(s, a; \theta)$$

- Deep Q-learning (DQN) published in 2015 played Atari games like no other

Deep Q-Learning

- In principle, the training then consists of collecting, by simulation, many tuples
(state, action, reward, next_state) = (s_i, a_i, r_i, s_i^+)
and update the model parameters θ by

$$\min_{\theta} \sum_i \overset{\text{Target}}{r_i + \max \gamma Q(s_i^+; \theta)} - \overset{\text{Prediction}}{Q(s_i, a_i; \theta)}$$

- One observation from the original paper: It is not good to use the same network, i.e., same set of parameters θ , for both prediction and target
- So they propose to have two networks with same structure and parameters θ and θ'

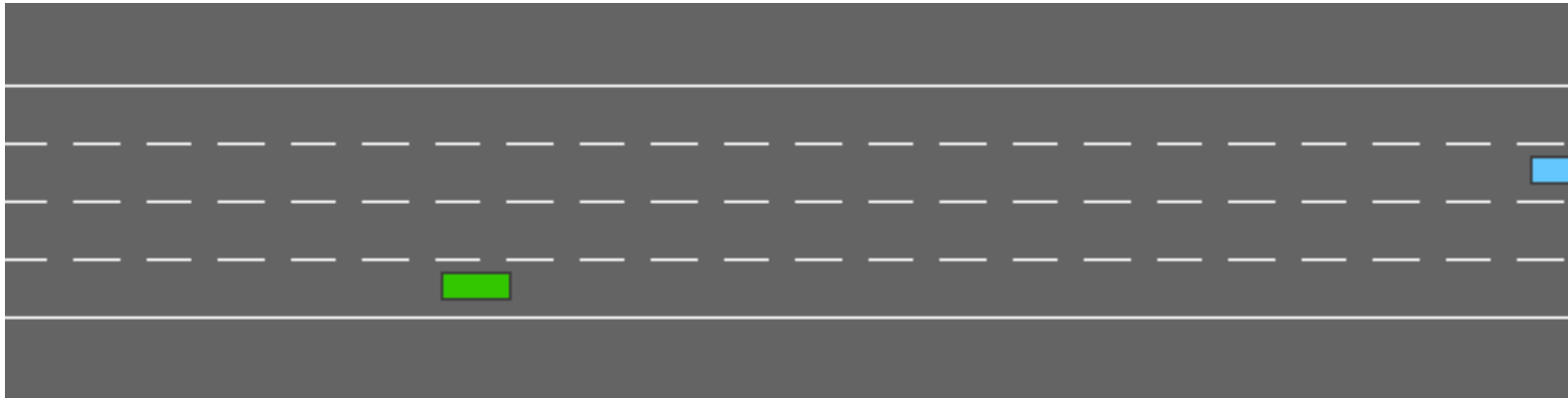
$$\min_{\theta} \sum_i r_i + \max \gamma Q(s_i^+; \theta') - Q(s_i, a_i; \theta)$$

and regularly update target network to be $\theta' := \theta$

- There is also a Double Deep Q-Learning that separates target and prediction even more.

Example: Deep Q-Learning in Highway Scenario (1/3)

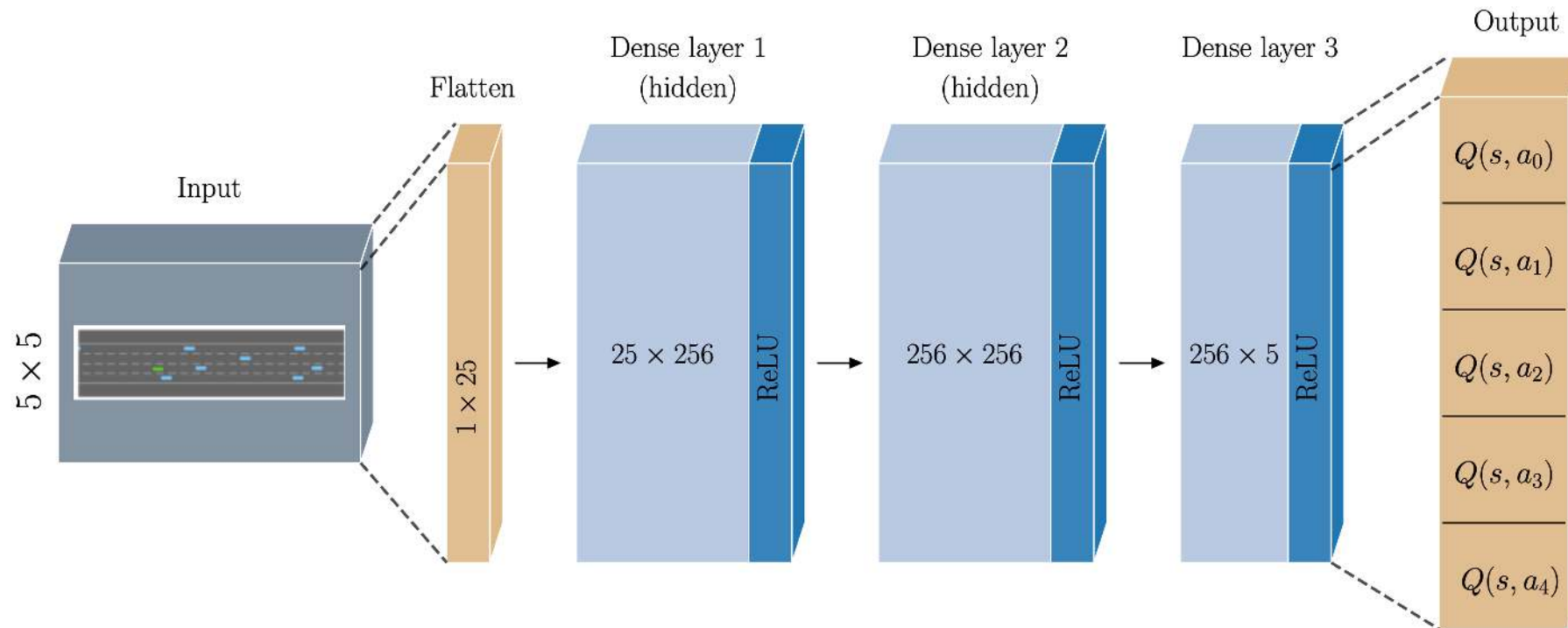
- Highway scenario used in extra assignment in Hand-in Exercise 5.



- Navigate through the lanes, with multiple vehicles driving, states and actions are continuous variables.
- Even with a coarse discretization of the continuous variables, Q-learning with a discrete, tabular Q-function would be infeasible.

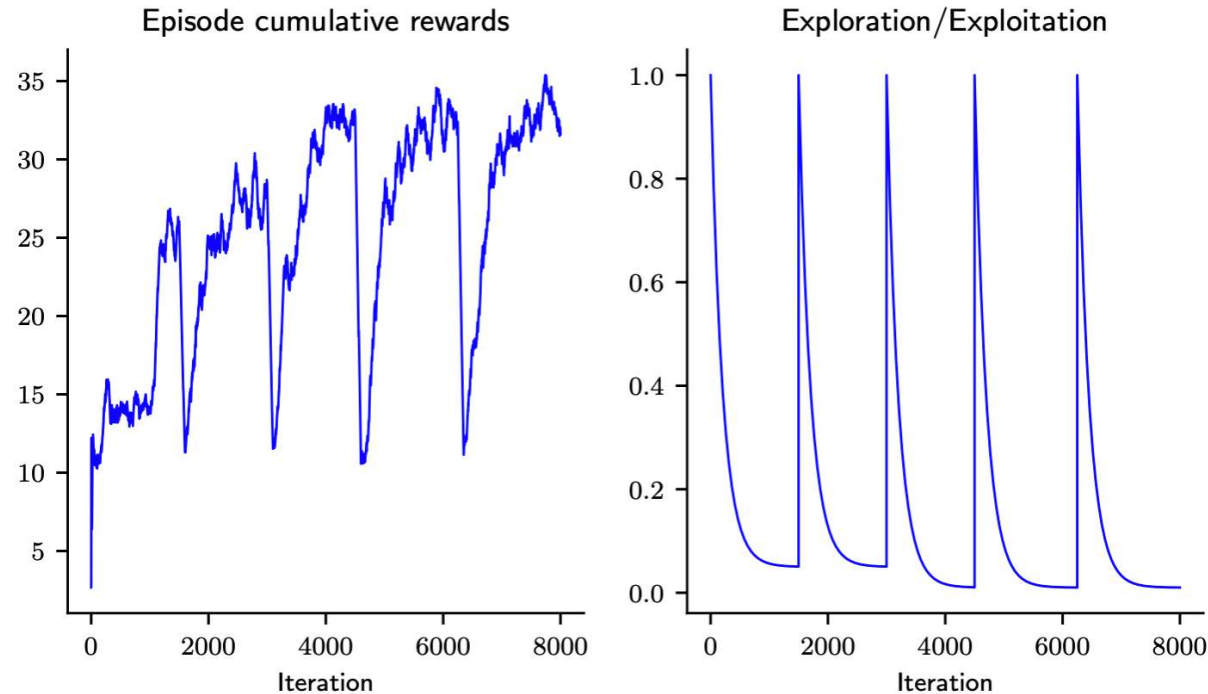
Example: Deep Q-Learning in Highway Scenario (2/3)⁵⁷

- Instead apply deep Q-learning using a neural-network approximation.
- Neural network has 73 753 parameters. Quite a lot, but still significantly less than the size of a corresponding discretized table.
- HI5-extra



Example: Deep Q-Learning in Highway Scenario (3/3)⁵⁸

- The agent chooses actions according to an epsilon-greedy strategy during training, gives a trade-off between exploration and exploitation.
- Sequences of such phases during training.



Software Libraries for Machine Learning

Some Tools for Machine Learning

- PyTorch is an open-source library for machine learning with support for deep neural networks.
 - <http://pytorch.org/>
 - Common in research, used in HI5-extra
- TensorFlow open-source library for machine learning, notably neural networks with high-dimensional parameter vectors and large amount of data.
 - <https://www.tensorflow.org>, <https://playground.tensorflow.org/>
 - Keras is a high-level interface to TensorFlow (and now they have extended it with Keras Core to multiple backends)
- Scikit-learn, many classical techniques in an easy-to-use package
<https://scikit-learn.org/>

Toolkit for Reinforcement Learning

- Gymnasium (former OpenAI Gym) toolkit for reinforcement learning
<https://gymnasium.farama.org/>
- Environments for implementation and evaluation of reinforcement-learning methods for traffic scenarios:
 - <https://github.com/eleurent/highway-env>

References and Further Reading

All the following books and articles are not part of the reading assignments for the course, but cover the topics studied during this lecture in more detail.

- Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.
- Hastie, T., R. Tibshirani, J. Friedman, & J. Franklin: The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd Edition, Springer, 2005.
- Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.
- Mnih, V., Kavukcuoglu, K., Silver, D. et al: "Human-level control through deep reinforcement learning", Nature 518, 529–533, 2015.
- Westny, T., Frisk, E., Olofsson, B: "Vehicle Behavior Prediction and Generalization Using Imbalanced Learning Techniques", IEEE International Intelligent Transportation Systems Conference, 2021.
- Åström, K. J.: "Optimal control of Markov processes with incomplete state information", Journal of Mathematical Analysis and Applications, 10(1), 174-205, 1965.

www.liu.se