Learning for Autonomous Vehicles – Part II

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

Lecture 10-11: Björn Olofsson <bjorn.olofsson@liu.se>

LINKÖPING UNIVERSITY

Purpose of this Lecture

- Provide an **introduction** (focus on usage of methods) to some **core methods in the field of learning for autonomous vehicles**:
  - **Neural networks**,
  - **Gaussian processes**,
  - **reinforcement learning**.

Expected Take-Aways from this Lecture

- Be familiar with how **neural networks** and **Gaussian processes** 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 and 2.1-2.3 in Rasmussen, C. E., & C. K. I. Williams: *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- 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.

Outline of the Lecture

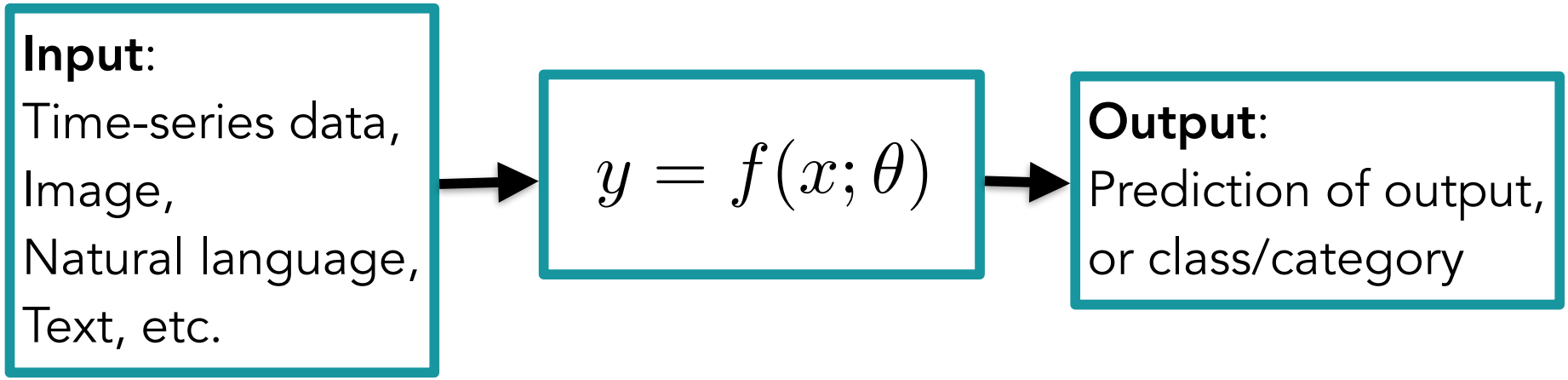
- Learning using **neural networks**.
- Learning using **Gaussian processes**.
- Introduction to **reinforcement learning**.
- **Software libraries** for machine learning.

Learning Using Neural Networks

Introduction and Background (1/2)

- Historically: **Inspiration from the human brain**, with its neurons and synapses (connections); activation of a neuron by a signal that reaches a certain threshold.
- A neural network is a **nonlinear function approximator** (often with multiple inputs and outputs), curve-fitting in **high-dimensional spaces**.
- Concept has been described **several decades ago**, and now a very **active research area again** because of much **available data**, **efficient algorithms**, and **computational hardware platforms**.

Introduction and Background (2/2)



- Function **parameterized** using parameters in θ (often high-dimensional, with thousand or millions of elements).
- Often **large amount of training data** $X = (x_1, \ldots, x_N)$ with large N.

A Nonlinear Function Approximator

- A **complex nonlinear relation** between inputs and outputs can be approximated using a **parametric function**.
- The parameters of the model are determined by **fitting the parameters** to the **training data**.
- **Separate validation data** are then used to evaluate the fit of the model.

Structure of a Single Hidden-Layer Neural Network

- A neural network consists of **hidden layers** and an **output layer**.
- The basic version contains a single hidden layer.
- A (nonlinear) **activation function** acts on an affine combination of the inputs.

$$Z_m = \sigma(\alpha_{0,m} + \alpha_m^{\mathrm{T}} X), \quad m = 1, \ldots, M,$$
$$T_k = \beta_{0,k} + \beta_k^{\mathrm{T}} Z, \quad k = 1, \ldots, K,$$
$$f_k(X) = g_k(T), \quad k = 1, \ldots, K$$

X – input data
$f_k(X)$ – model output for input X
$\sigma(\cdot)$ – activation function
α, β – parameters
$Z = (Z_1, Z_2, \ldots, Z_M)$
$T = (T_1, T_2, \ldots, T_K)$
$g_k(\cdot)$ – output function

Hastie, T., R. Tibshirani, J. Friedman, & J. Franklin: The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd Edition, Springer, 2005.

Some Example Activation Functions

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

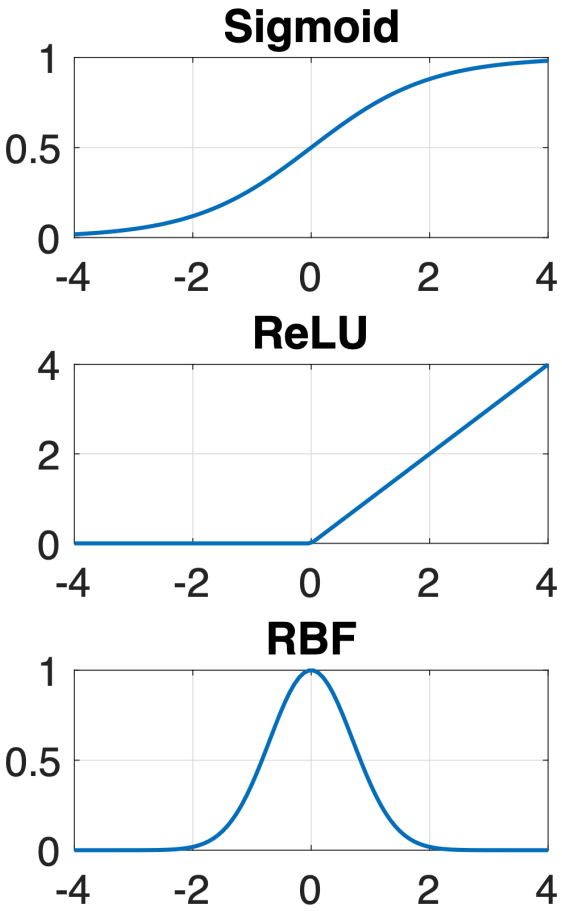
$$\sigma(v) = \frac{1}{1 + \exp(-v)}$$

- Rectifier (Rectified Linear Unit – **ReLU**):

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

- Gaussian radial basis function (**RBF**):

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



Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.

Choices of Output Function

- For **regression problems** (i.e., finding relations between dependent and independent variables), the output function can be linear

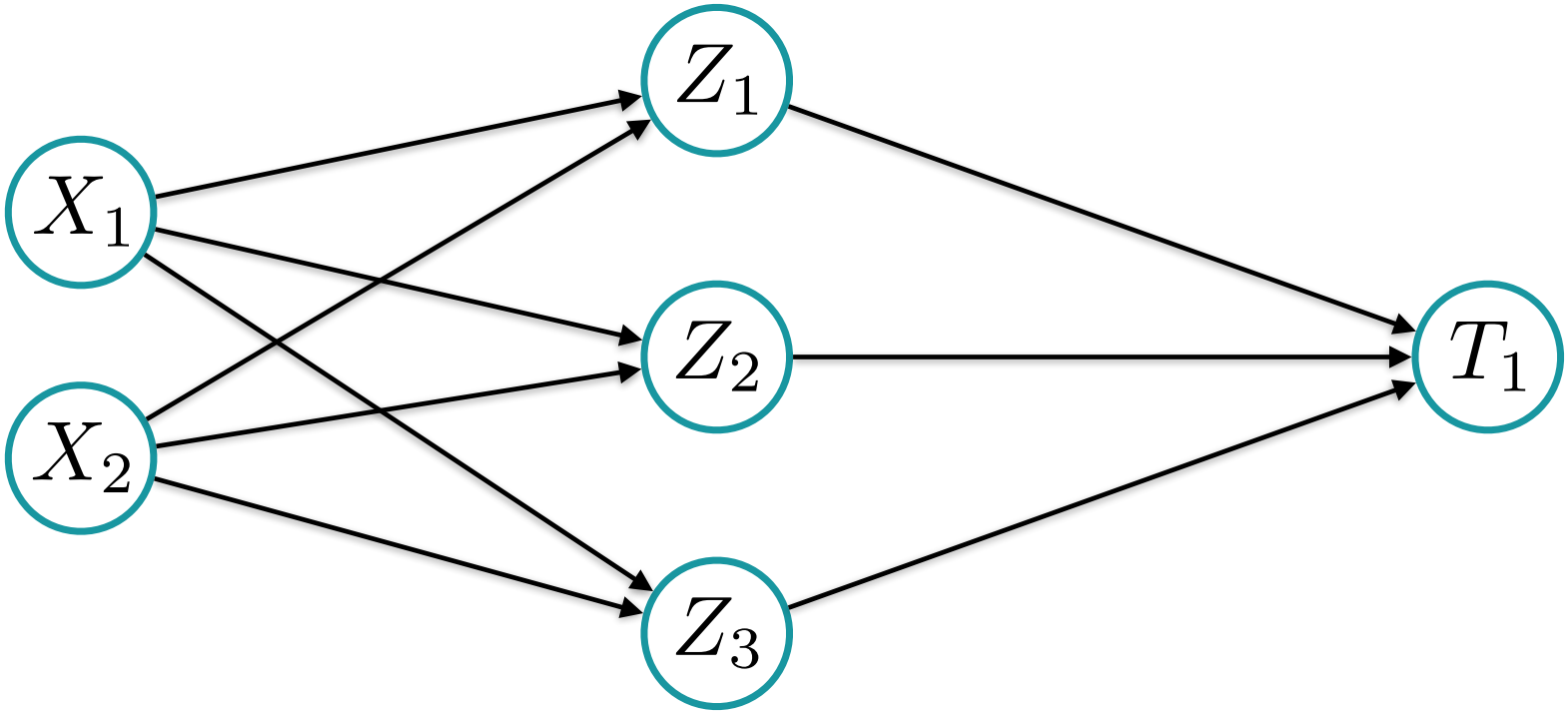
$$g_k(T) = T_k$$

- For **classification problems**, the output function can be

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

Hastie, T., R. Tibshirani, J. Friedman, & J. Franklin: The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd Edition, Springer, 2005.

Example of a Neural Network



$$\sigma(\alpha_{0,m} + \alpha_m^{\mathrm{T}} X) \qquad \beta_{0,k} + \beta_k^{\mathrm{T}} Z$$

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.
- Examples of cost functions.

Squared-error cost function (regression):

$$J(\theta) = \sum_{i,k} \left(y_{i,k} - f_k(x_i)\right)^2$$

Cross-entropy cost function (classification):

$$J(\theta) = -\sum_{i,k} y_{i,k} \log\left(f_k(x_i)\right)$$

Hastie, T., R. Tibshirani, J. Friedman, & J. Franklin: The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd Edition, Springer, 2005.

Gradient Descent (1/2)

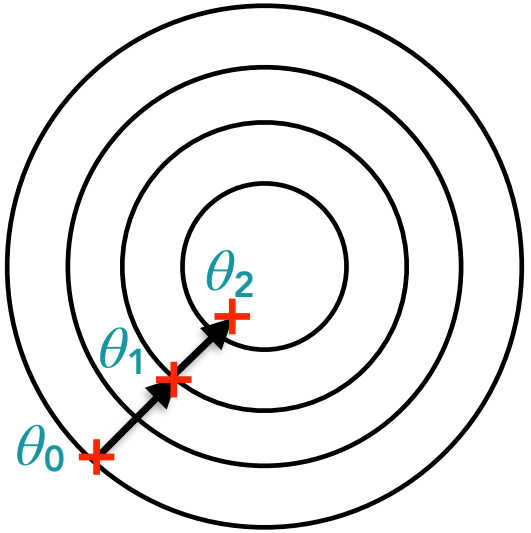
- **Gradient descent** is a method to find a **local minimum** to a (differentiable) cost function using only first-order derivatives.
- Recall from the courses in calculus that a function decreases most rapidly when **moving in the negative direction of the gradient**.

Gradient Descent (2/2)

- **Iterative method** for moving towards a local minimum:

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

γ – learning rate (pre-determined parameter)
θ_0 – initial guess for parameters



Stochastic Gradient Descent (SGD)

- Computation of the gradient becomes a challenge when the **number of data points increases to large numbers** or **network output time-consuming to compute**.
- **Stochastic gradient descent** has been suggested to mitigate this:
  - Randomly **select a subset of the data points** (in the limit only one) and perform gradient descent.
  - Repeat until a local minimum has been reached (**termination condition**).

Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.

Backpropagation

- Recall the **automatic differentiation** method for computing derivatives from Lecture 5.
- Neural networks are often trained using **backpropagation**, which also utilizes the **chain rule** to **compute the desired derivatives** efficiently and accurately (reverse-mode automatic differentiation, see Lecture 5).

Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.

Deep Neural Networks

- The neural networks considered so far only consist of one hidden layer.
- **Deep neural networks** comprise multiple hidden layers, where each layer can be considered as an abstraction.
  - Idea: **Features** in the **training data** captured by the neural network.
- **Large variety of network structures** (number of layers, how many neurons in each, types of neurons, connectivity, etc.).
- Typically a **composition of layers** of different network primitives.

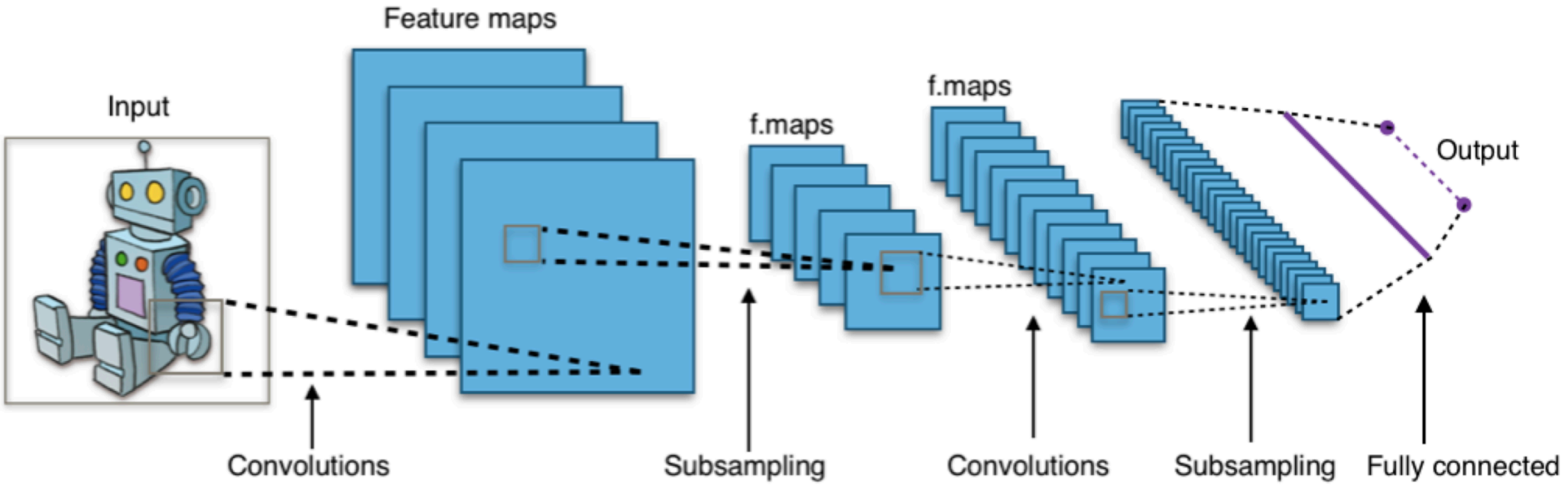
Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.

Recurrent and Convolutional Neural Networks

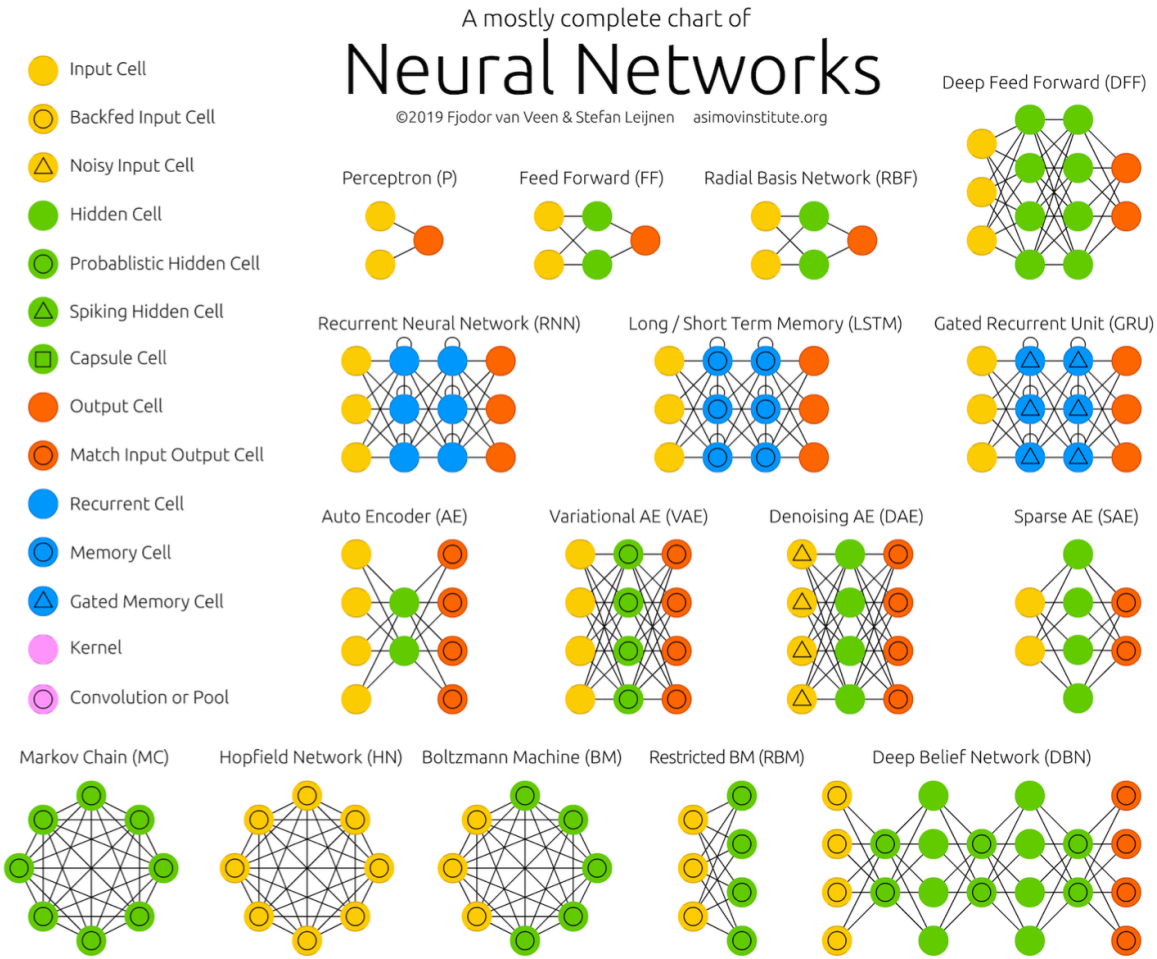
- A **recurrent neural network** also has internal feedback loops (modeling a local memory).
- A **convolutional neural network** comprises layers where convolution operators act.
  - Instead of fully connected layers, **local spatial information** is modeled through the convolutions.
  - Often used for neural networks involving **inputs** with a **grid structure**, like a camera image or written text.

Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.

Example Structure of Convolutional Neural Networks

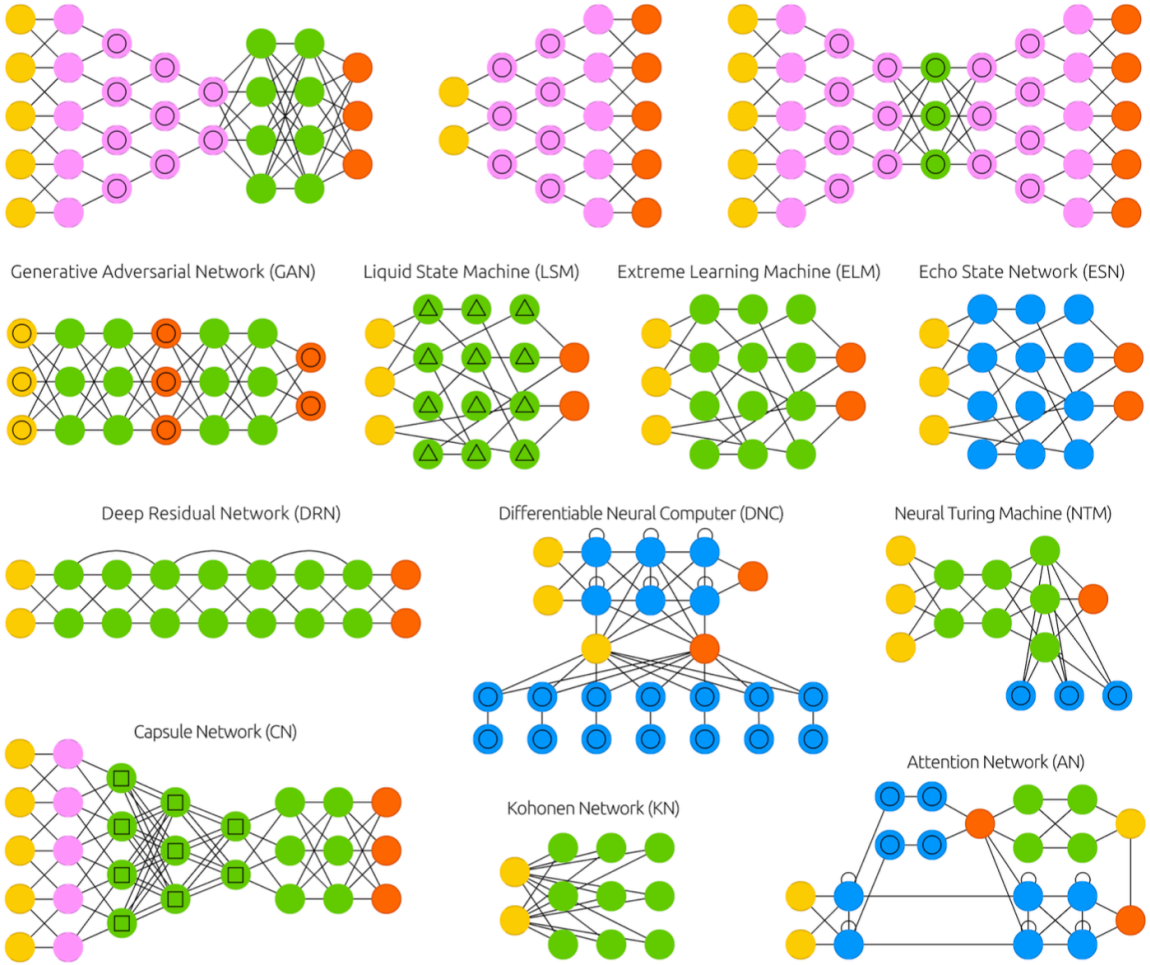


Architectures for Neural Networks (1/2)



Picture from https://www.asimovinstitute.org/neural-network-zoo/

Architectures for Neural Networks (2/2)



Picture from https://www.asimovinstitute.org/neural-network-zoo/

Implementation Aspects in Neural Network Training (1/4)

- Several **extensions and variants of stochastic gradient descent** exist for improved performance (e.g., momentum acceleration, Nesterov accelerated gradient, AdaGrad, RMSProp, and Adam).
- Choice of **initial values** of **model parameters** in the gradient descent.

Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.

Implementation Aspects in Neural Network Training (2/4)

- **Overfitting** of model parameters to training data is common.
  - With **high-dimensional parameter vectors**, it is easy to obtain overfitting to the particular data used for training.
  - Methods used for mitigating overfitting:
    - **Early termination** in the SGD.
    - **Regularization** terms in cost function, e.g.,

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

Hastie, T., R. Tibshirani, J. Friedman, & J. Franklin: The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd Edition, Springer, 2005.
Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.

Implementation Aspects in Neural Network Training (3/4)

- 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).
- To avoid getting stuck in **local minima** in the optimization, training the network with many different initial guesses of the parameters in SGD is beneficial.

Hastie, T., R. Tibshirani, J. Friedman, & J. Franklin: The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd Edition, Springer, 2005.
Goodfellow, I., Y. Bengio, & A. Courville: Deep Learning. MIT Press, 2016.

Implementation Aspects in Neural Network Training (4/4)

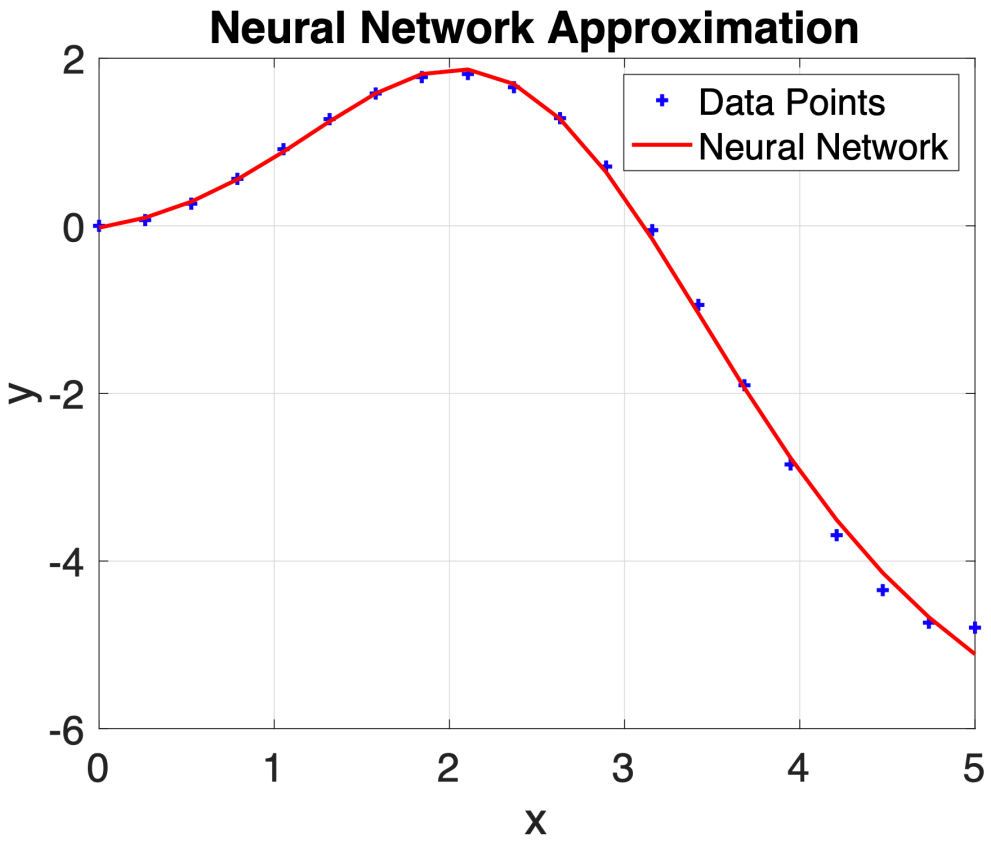
- **Imbalance in the training data** for a neural network aimed for classification can result in a **biased classifier**.
  - Example is when data from one or more classes are overrepresented compared to other classes.
- One approach to remedy this is to **weight underrepresented classes** when creating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

Example: Simple Neural Network for Regression

- Given 20 samples from the **nonlinear function**

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

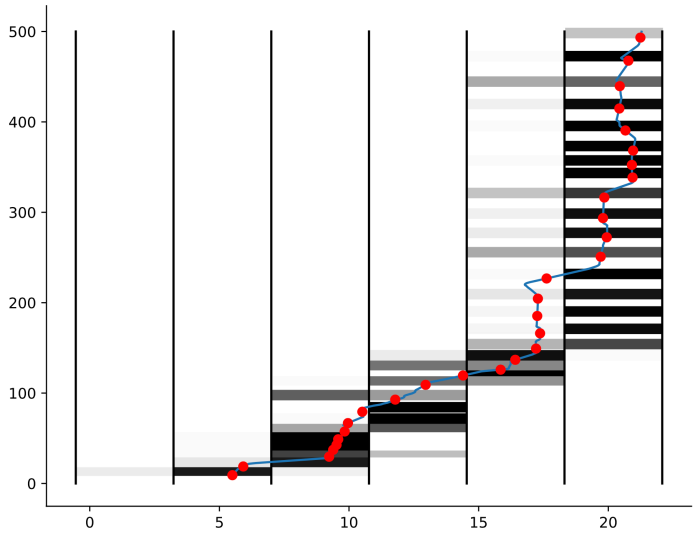
- A **neural network approximation** with **one hidden layer** with 30 neurons shown in red.



Hand-in Exercise 5: Neural Network for Driving Prediction

- In the extra 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.

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



I-80 data set citation: U.S. Department of Transportation Federal Highway Administration. (2016). Next Generation Simulation (NGSIM) Vehicle Trajectories and Supporting Data. [Dataset]. Provided by ITS DataHub through Data.transportation.gov. Accessed 2020-09-29 from http://doi.org/10.21949/1504477.

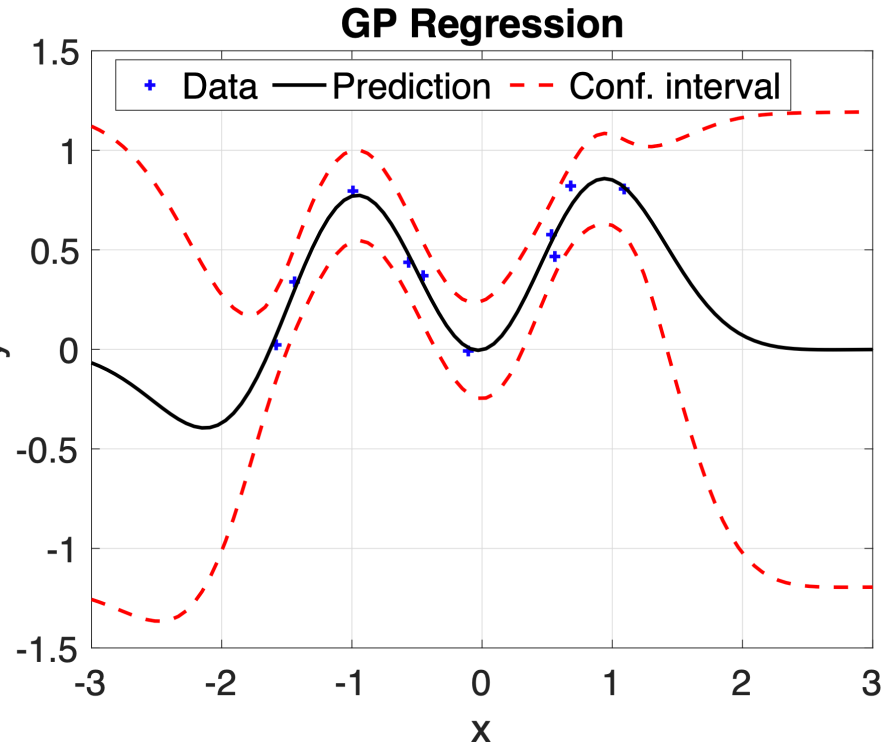
Learning Using Gaussian Processes

Basic Idea of Gaussian Processes (1/2)

- **Neural networks** use a **set of parameters** to fit a model to the training data (parametric method).
- **Inherent uncertainty in the data**, how to quantify?
- **Gaussian processes** can be used as a **non-parametric** approach to model data, which inherently capture the variations of the training data.

Basic Idea of Gaussian Processes (2/2)

- Example of **Gaussian process regression** based on 10 training data points.
- How to **make predictions for new input values**, and how about their **uncertainty**?



Background: Gaussian Probability Distribution (1/2)

- The **multivariate Gaussian probability density** for the stochastic variables $(x_1, x_2, \ldots, x_N)$ is given by

$$p(x) = \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^{\mathrm{T}} \Sigma^{-1} (x-\mu)\right)$$

$$x \sim \mathcal{N}(\mu, \Sigma), \quad x = (x_1, x_2, \ldots, x_N)$$

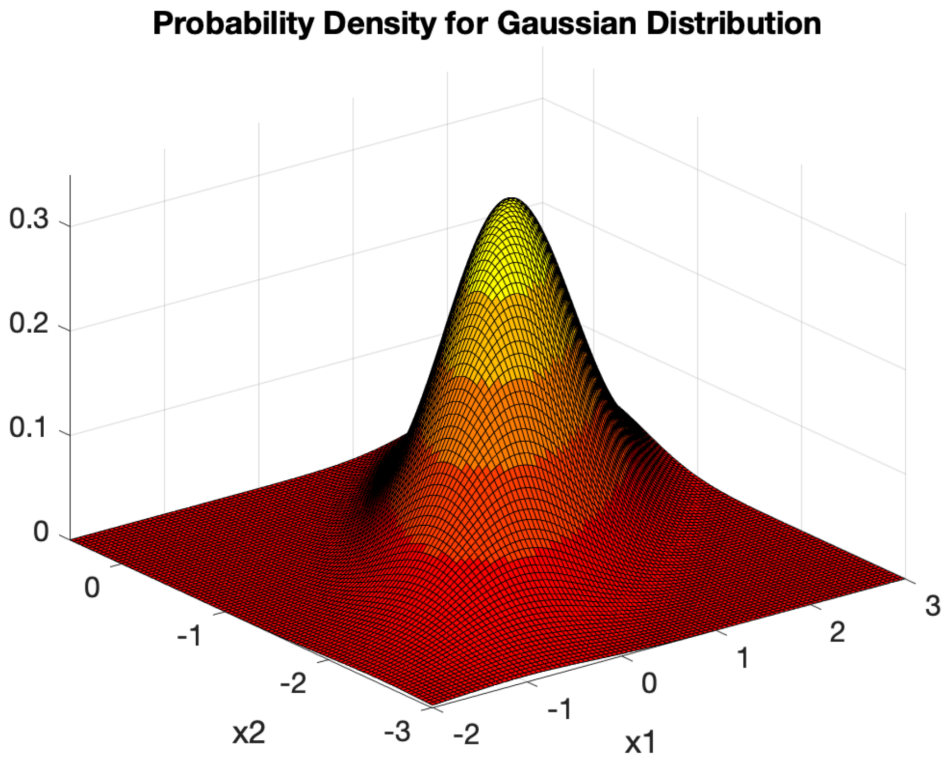
μ – mean

Σ – covariance matrix

Background: Gaussian Probability Distribution (2/2)

- Illustration of a **Gaussian probability density** in the 2D case.
- **Mean** and **covariance matrix** given by

$$\mu = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$



Definition of a Gaussian Process

- A **stochastic process** is a family of random variables, with some specified properties.
- A **Gaussian process** is defined as[1]

"A collection of random variables, any finite number of which have a joint Gaussian distribution."

[1]Rasmussen, C. E., & C. K. I. Williams: Gaussian Processes for Machine Learning. MIT Press, 2006.

Background: Conditional Gaussian Distribution

- Assume that we have two vectors that are **jointly Gaussian**

$$\begin{pmatrix} x \\ y \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \begin{pmatrix} \Sigma_x & \Sigma_{x,y} \\ \Sigma_{x,y}^{\mathrm{T}} & \Sigma_y \end{pmatrix} \right)$$

- The **conditional distribution** is then given by

$$x|y \sim \mathcal{N}\left(\mu_x + \Sigma_{x,y}\Sigma_y^{-1}(y - \mu_y), \Sigma_x - \Sigma_{x,y}\Sigma_y^{-1}\Sigma_{x,y}^{\mathrm{T}}\right)$$

Gaussian Process Regression (1/3)

- Assume that we have data from a process of the form

$$y = f(x) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

- The **covariance matrix** of the (vector) of scalar output data is assumed to be (with covariance function to be chosen)

$$\mathrm{Cov}(y) = K(X, X) + \sigma_\varepsilon^2 I \qquad K_{i,j} = k(x_i, x_j)$$

where $X = (x_1, x_2, \ldots, x_N)$

Rasmussen, C. E., & C. K. I. Williams: Gaussian Processes for Machine Learning. MIT Press, 2006.

Gaussian Process Regression (2/3)

- How to use the **available training data** and the **assumptions on the mean and covariance function** to make **predictions** about function values for new input data?

x_i – vector input data
X – matrix of stacked input data
y – vector of scalar output data
x_* – input data for prediction
X_* – matrix of stacked input prediction data

Gaussian Process Regression (3/3)

- By establishing the **joint distribution** between the **output training data** and the **output values at the desired prediction inputs** using the **conditional distribution**, it can be shown that (recall relationship for the conditional distribution)

$$y_* | X, y, X_* \sim \mathcal{N}(\bar{y}_*, \mathrm{Cov}(y_*))$$

where

$$\bar{y}_* = K(X_*, X)(K(X, X) + \sigma_\varepsilon^2 I)^{-1} y,$$

$$\mathrm{Cov}(y_*) = K(X_*, X_*) - K(X_*, X)(K(X, X) + \sigma_\varepsilon^2 I)^{-1} K(X, X_*) + \sigma_\varepsilon^2 I$$

Rasmussen, C. E., & C. K. I. Williams: Gaussian Processes for Machine Learning. MIT Press, 2006.

Choice of Covariance Function

- The **covariance function** should provide information about to what extent data samples close to each other are co-varying.
- A **common choice** is the squared exponential function (here 1D)

$$k(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{1}{2l^2}(x_i - x_j)^2\right)$$

- The **hyperparameters** of this function are $\sigma_f,\ l$.
- Several other choices of the covariance function exist (see Table 4.1 in the book by Rasmussen and Williams).

Rasmussen, C. E., & C. K. I. Williams: Gaussian Processes for Machine Learning. MIT Press, 2006.

Identification of Hyperparameters from Data

- The **hyperparameters** θ for the GP model can be tuned manually.
- They can also be learned from the training data by maximizing the **marginal likelihood** of the training output data, given the **inputs** and the **hyperparameters**

$$p(y|X,\theta)$$

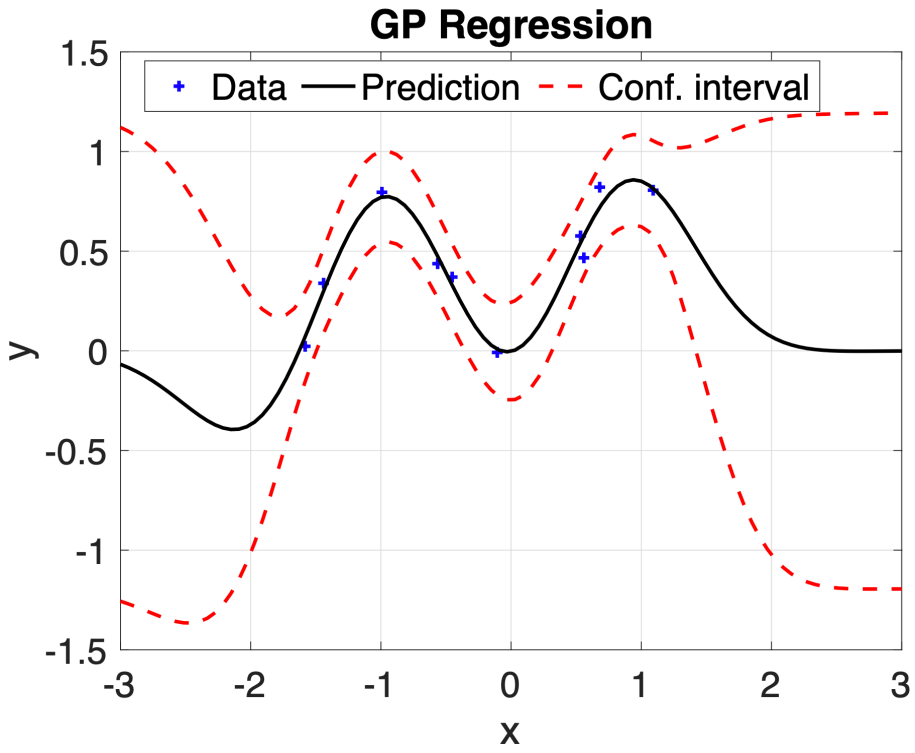
Rasmussen, C. E., & C. K. I. Williams: Gaussian Processes for Machine Learning. MIT Press, 2006.

Example: Gaussian Process Regression (1/2)

- Given 10 samples from the **process**

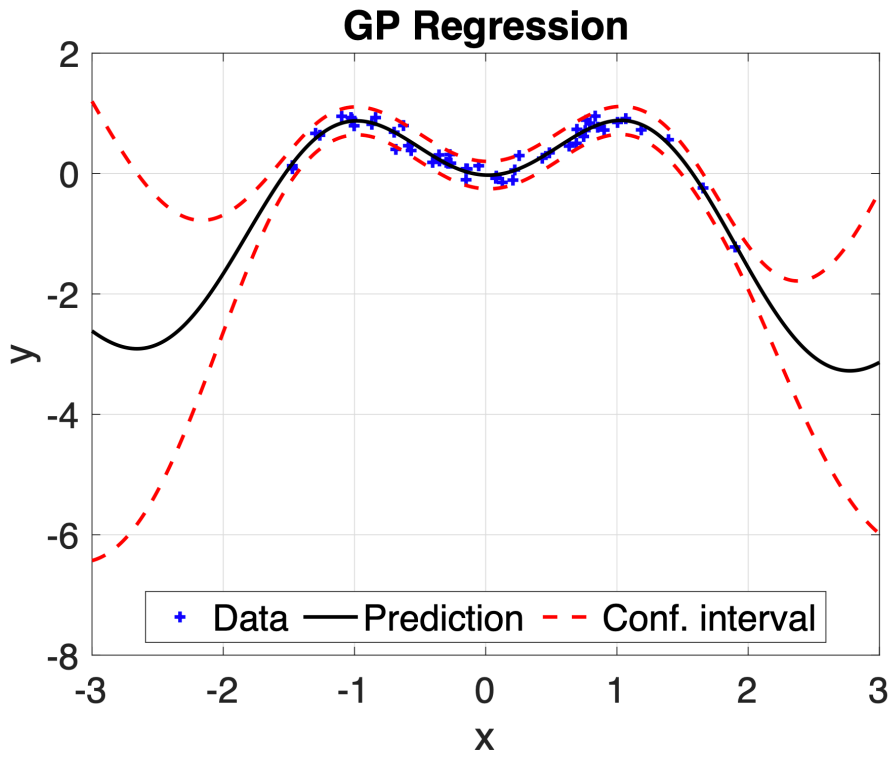
$$y = x\sin(2x) + \varepsilon,\ \ \varepsilon \sim \mathcal{N}(0, 0.1^2)$$

- **GP model hyperparameters** in squared exp. function optimized based on training data.
- **Prediction** for input values in the interval [-3,3].



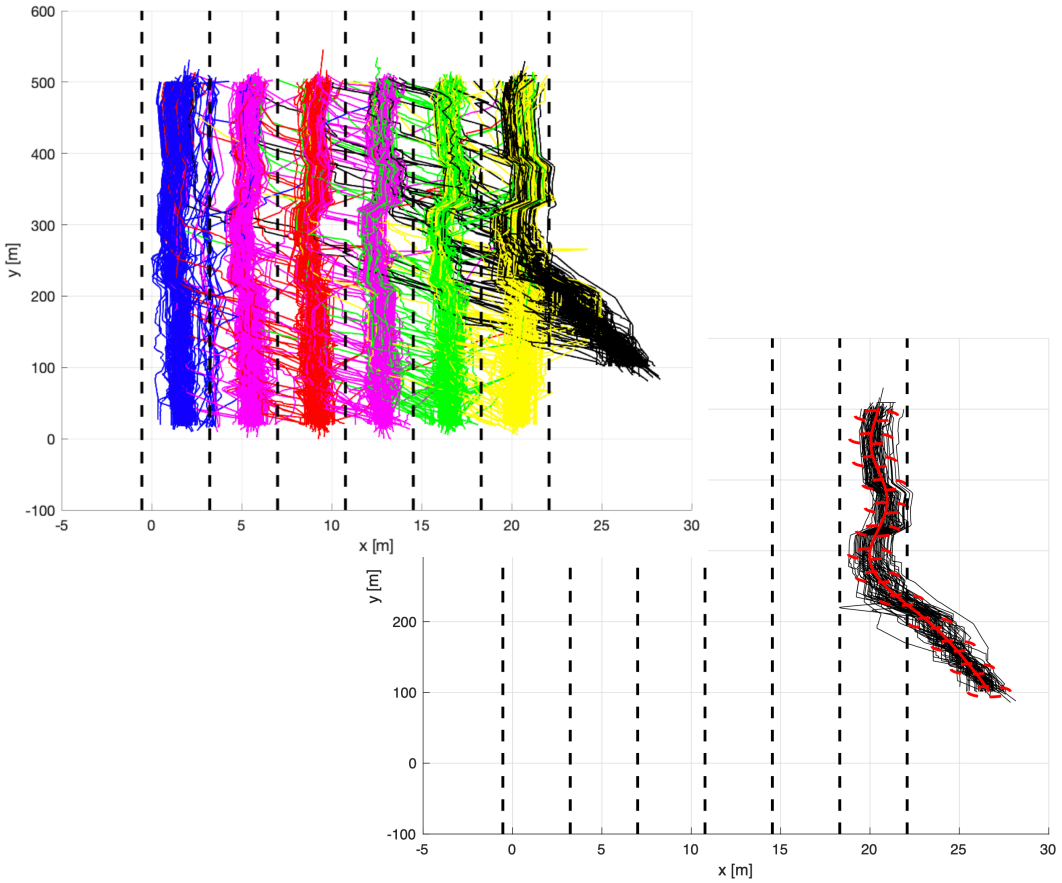
Example: Gaussian Process Regression (2/2)

- Same process as previous slide, but with **50 training data points instead**.
- **Prediction** for input values in the interval [-3,3].



Gaussian Processes in Hand-in Exercise 5

- **Regression** of **driver data** from the I-80 highway section data set using **Gaussian processes**.
- Models computed for **different lane-change combinations**.



I-80 data set citation: U.S. Department of Transportation Federal Highway Administration. (2016). Next Generation Simulation (NGSIM) Vehicle Trajectories and Supporting Data. [Dataset]. Provided by ITS DataHub through Data.transportation.gov. Accessed 2020-09-29 from http://doi.org/10.21949/1504477.

Introduction to Reinforcement Learning

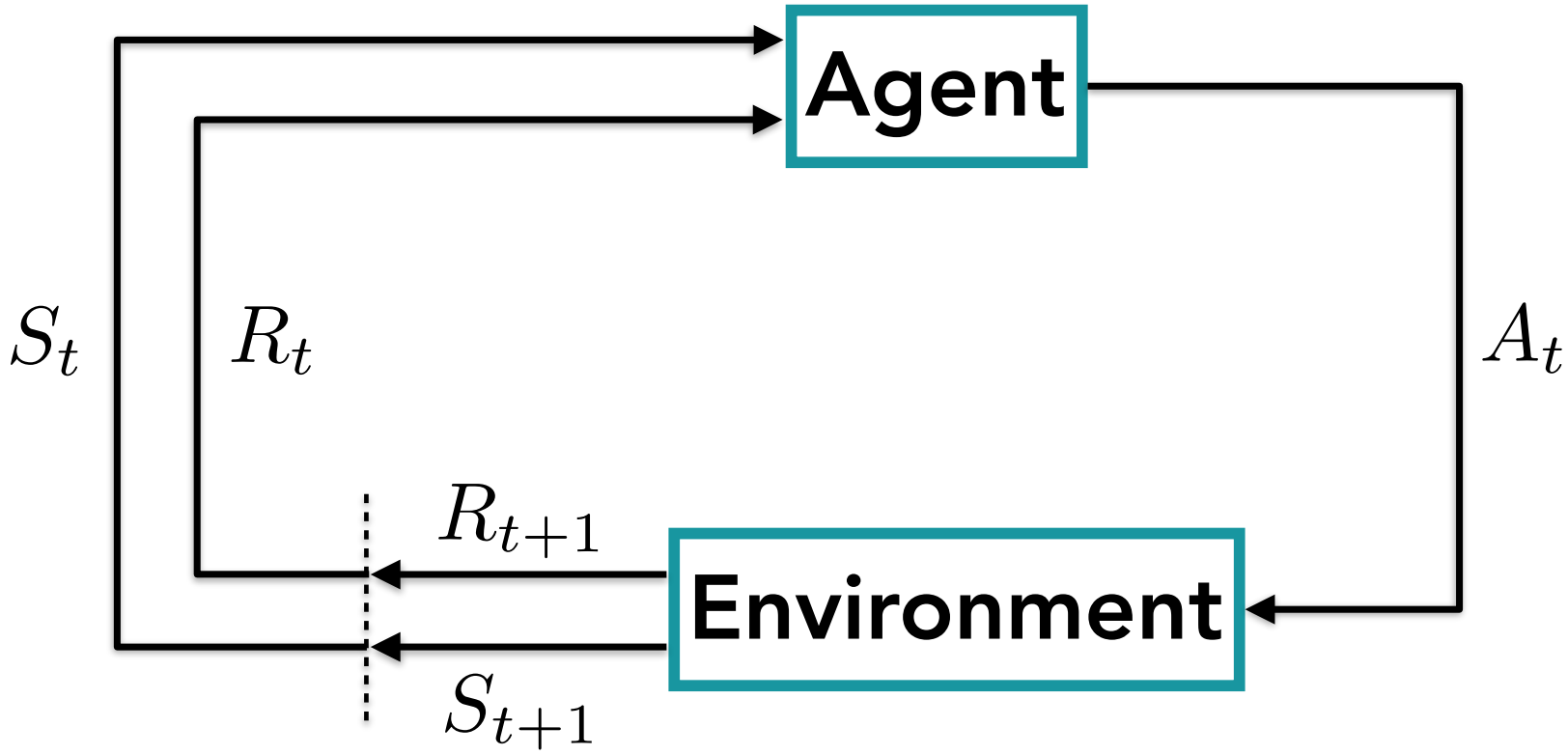
Introduction to Reinforcement Learning (1/2)

- In some environments or scenarios, it is **difficult to explicitly model the dynamics** (e.g., an autonomous vehicle in an unstructured environment).
- How to find **control laws** under **uncertainty** and (partially) **unknown dynamics,** and possibly **uncertain state information**?
- In **control engineering**: system identification, control (e.g., adaptive and stochastic) and observer design.
- **Reinforcement learning** has successfully been used for various challenging scenarios (computer games, chess, Go, robot control, etc.)

Introduction to Reinforcement Learning (2/2)

- **Learn** how to act optimally in each state in the world model (a policy), by interacting with the environment and **maximize an accumulated received reward signal**.
- Learn which actions to take in different situations by **sequentially taking actions and exploring the environment**.
- Trade-off between **exploration** and **exploitation**.
- **Uncertain environments**, typically modeled using probability distributions.

The Agent-Environment Interaction Model



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

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

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** $S_t \in \mathcal{S}$ – 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.

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Markov Decision Process (MDP) (1/3)

- **Markov decision processes** (MDPs) are a mathematical framework for sequential decision-making problems.
- Basis for formulation of many reinforcement learning problems, here we consider **finite MDPs**.
- Maximize the (discounted) **accumulated return**

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Markov Decision Process (MDP) (2/3)

- **Policy** defines **action** to be taken (updated sequentially)

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

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

$$v_\pi(s) = \mathrm{E}_\pi(G_t|S_t = s) = \mathrm{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right)$$

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

$$q_\pi(s, a) = \mathrm{E}_\pi(G_t|S_t = s, A_t = a) = \mathrm{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right)$$

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Markov Decision Process (MDP) (3/3)

- The **state transition** resulting from an action described by

$$p(s', r|s, a) = \mathrm{P}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$$

- The MDP consists of the following (finite) **sets**, **reward signal**, and **state-transition and reward probabilities**

$$\mathcal{S},\ \mathcal{A},\ \mathcal{R},\ p(s', r|s, a)$$

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

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

$$\mathrm{P}(S_t, R_t | A_{t-1}, S_{t-1}, \ldots, A_0, S_0) = \mathrm{P}(S_t, R_t | A_{t-1}, S_{t-1})$$

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

Partially Observable Markov Decision Process (POMDP)

- In the MDP formulation, it has so far been assumed that the current state is **fully observable**.
- What if all states are not known, or cannot be measured?
- **Partially observable MDP** (**POMDP**) takes the uncertainty in the state into account, by introduction of a **belief state** that is updated based on observations.

Åström, K. J.: "Optimal control of Markov processes with incomplete state information", Journal of Mathematical Analysis and Applications, 10(1), 174-205, 1965.

Optimality

- In reinforcement learning, **policies maximizing the total reward** are searched for

$$\pi_* = \mathrm{argmax}_\pi v_\pi(s)$$

- The **optimal value function** and **optimal action-value function** are given by

$$v_*(s) = \max_\pi v_\pi(s), \quad q_*(s,a) = \max_\pi q_\pi(s,a)$$

- Must hold all for **all states and all allowed state-action pairs**.

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

The Bellman Optimality Equations

- The **Bellman optimality equations** define recursive relationships for value function and action-value functions.
- Optimality equation for the **value function** ($s \to s'$)

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

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

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

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Dynamic Programming in Reinforcement Learning

- **Dynamic programming** (DP) can be used to solve a problem formulated as a finite MDP, given **exact model knowledge**.
- Basic idea: **Search for optimal policies** using the **Bellman optimality equations** for the value or action-value functions.
- Two main variants: **policy iteration** (initialize, policy evaluation, policy improvement, repeat) and **value iteration** (next slide).

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Value Iteration towards an Optimal Policy

Algorithm 1: Value Iteration

Initialize $V(s)$ arbitrarily $\forall s \in \mathcal{S}$
repeat
  $\Delta = 0$
  foreach $s \in \mathcal{S}$:
    $v = V(s)$
    $V(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma V(s'))$
  $\Delta = \max(\Delta, |v - V(s)|)$
until $\Delta < \varepsilon$
return policy π as:
  $\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)(r + \gamma V(s'))$

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

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Unknown Environment and Exploration vs. Exploitation (1/2)

- Policy and value iterations rely on **known state-transition and reward probabilities**.
- What if these are not known *a priori* and the **environment is unknown**?
- **Learn** the characteristics of the **environment** by **interacting** with it.

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Unknown Environment and Exploration vs. Exploitation (2/2)

- Leads to the question of **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:

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

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Temporal Difference (TD) Learning

- **Temporal-difference methods** try to learn optimal policies **without explicit knowledge** of the environment and its dynamics.
- Concepts from **dynamic programming** and **Monte Carlo methods** used and combined, by **bootstrapping** (update estimates based on other learned estimates).
- Basic idea: **successively update** the **value function** as

$$V(S_t) = V(S_t) + \alpha \underbrace{(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))}_{\text{TD Error}}$$

- Common methods of this type are **SARSA** and **Q-learning** (next slide).

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Q-Learning

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

Algorithm 2: Q-Learning

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

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

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Value-Function and Action-Value Approximations

- When the state and action spaces **increase in dimension**, **parameterized functions** can be used to **approximate** the value function or action-value function

$$v_*(s) \approx \hat{v}(s;\theta), \quad q_*(s,a) \approx \hat{q}(s,a;\theta), \quad \theta \text{ parameters}$$

- **Common choices** of functions are linear, polynomials, neural networks, etc.
- Reformulate previous methods to **update parameters** instead of value or action-value functions directly (e.g., using gradient descent).

Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

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

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

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

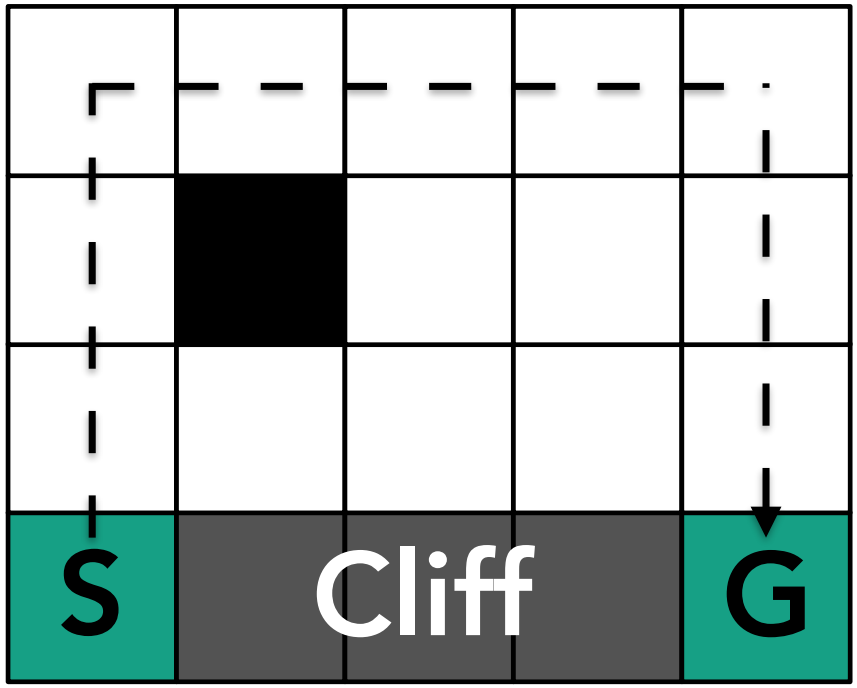
Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Simulation as a Tool for Reinforcement Learning

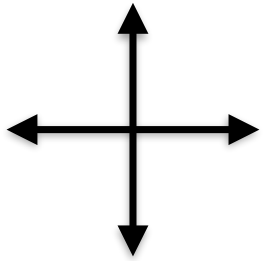
- In some environments, it could be **challenging to iteratively interact** with the environment by **actual experiments**.
- A **simulated environment** can therefore be used to train the algorithm (possibly also **reducing the time** to perform the required experiments).

Reinforcement Learning in Hand-in Exercise 5

- Autonomous vehicle moves in a **grid world** from start to goal.
- **Stochastic winds** might lead to detour from intended action.
- **Cliffs** should be avoided (negative reward).
- **Small negative reward** for states other than those in the lower row.



High negative reward



Possible actions

Expanded version of Example 6.6 in Sutton, R. S., & A. G. Barto: Reinforcement learning: An introduction. MIT Press, 2018.

Software Libraries for Machine Learning

Some Tools for Neural Networks

- **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.
- **PyTorch** is an open-source library for machine learning with support for **deep neural networks**.
  - http://pytorch.org/

Some Tools for Gaussian Processes

- **GPML Toolbox** for Matlab (regression, classification, and optimization of hyperparameters).
  - http://www.gaussianprocess.org/gpml/code/matlab/doc/
- **scikit-learn** in Python has a module for Gaussian processes.
  - https://scikit-learn.org/stable/modules/gaussian_process.html

Toolkit for Reinforcement Learning

- **OpenAI Gym toolkit** for **reinforcement learning** (collection of **example problems and environments**, and **interfaces** to other libraries such as TensorFlow).
  - https://gym.openai.com

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.
- Rasmussen, C. E., & C. K. I. Williams: *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- Sutton, R. S., & A. G. Barto: *Reinforcement learning: An introduction*. MIT Press, 2018.
- 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

LINKÖPING UNIVERSITY