Deep reinforcement learning and an integral

“Deep learning provide models that automatically learns representations of data with multiple layers of abstraction.”

Thomas Schön
Division of Systems and Control
Department of Information Technology
Uppsala University.

Email: thomas.schon@it.uu.se,
www: user.it.uu.se/~thosc112

Joint work with: John-Alexander M. Assael (University of Oxford), Marc Deisenroth (Imperial College), Joel Kronander (Linköping University), Jonas Unger (Linköping University), and Niklas Wahlström (Uppsala University).

School of ICT, KTH, Stockholm, Sweden, June 1, 2016.
Background – what we do in the team


Both basic research and applied research (with companies).
Background – what we do in the team

Scientific field: Intersection of **Machine Learning**, **Automatic Control** and **Signal Processing**.

Both basic research **and** applied research (with companies).

Create **new probabilistic models** for dynamical systems and develop methods to **automatically learn** these models from measured data.
Background – what we do in the team


Both basic research and applied research (with companies).

Create new probabilistic models for dynamical systems and develop methods to automatically learn these models from measured data.

These models can be used by machines (computers) and/or humans to automatically understand and/or make decisions about what will happen next.
What is machine learning all about?

Machine learning is about learning, reasoning and acting based on data.
What is machine learning all about?

*Machine learning is about learning, reasoning and acting based on data.*

Machine learning develop methods allowing computers to improve their performance at certain tasks based on observed data.

Find and understand *hidden structures* and regularities in data.
What is machine learning all about?

Machine learning is about learning, reasoning and acting based on data.

Machine learning develop methods allowing computers to improve their performance at certain tasks based on observed data.

Find and understand hidden structures and regularities in data.

“It is one of today’s most rapidly growing technical fields, lying at the intersection of computer science and statistics, and at the core of artificial intelligence and data science.”


A recent example

First steps towards an autonomous system that learns by itself from raw pixel data.

Trial: 1 Frame: 1

- Deep autoencoder network + nonlinear dynamical model
- Learn not only a model, but also how to collect new data to achieve a goal.
- The model is automatically improved (in an iterative manner)
- DeepMind has cool work in this direction.

Another recent example

Automatically learn how to describe the contents of images.

Illustrates the modularity of the autoencoder, consisting of an encoder (vision deep CNN) and a decoder (language generating RNN).

A few examples where it failed
A few examples where it failed

Again, a very nice achievement and the fact that they show where it fails is honest.
Deep learning: One more recent example

An AI defeated a human professional for the first time in the ancient game of Go

Outline

1. Introduction via three recent DL applications
2. What is a neural network (NN)?
   a) Concrete example for regression
   b) Learning and regularization
3. What is a deep neural network?
4. Learning deep neural networks
   a) Pre-training
   b) Defining and learning the autoencoder
5. Deep reinforcement learning via an example
   a) Pixels-to-torques problem
   b) Deep autoencoder
   c) Developing a deep dynamical model
6. Solving an integral – a rendering application
A neural network (NN) is a nonlinear function $y = g_{\theta}(u)$ from an input variable $u$ to an output variable $y$ parameterized by $\theta$. 

Linear regression models the relationship between a continuous target variable $y$ and an input variable $u$, 

$$y = \sum_{i=1}^{D} w_i u_i + b + \epsilon = \theta^T u + \epsilon,$$

where $\epsilon$ is noise and $\theta$ is the parameters composed by the "weights" $w_i$ and the offset ("bias") term $b$, $\theta = (b, w_1, w_2, \cdots, w_D)^T$, $u = (1, u_1, u_2, \cdots, u_D)^T$. 

8/36 thomas.schon@it.uu.se

School of ICT, KTH, Stockholm, Sweden, June 1, 2016.
Constructing an NN for regression

A neural network (NN) is a nonlinear function $y = g_\theta(u)$ from an input variable $u$ to an output variable $y$ parameterized by $\theta$.

Linear regression models the relationship between a continuous target variable $y$ and an input variable $u$,

$$y = \sum_{i=1}^{D} w_i u_i + b + \epsilon = \theta^T u + \epsilon,$$

where $\epsilon$ is noise and $\theta$ is the parameters composed by the “weights” $w_i$ and the offset (“bias”) term $b$,

$$\theta = (b \ w_1 \ w_2 \ \cdots \ w_D)^T,$$

$$u = (1 \ u_1 \ u_2 \ \cdots \ u_D)^T.$$
Generalized linear regression

We can generalize this by introducing nonlinear transformations of the predictor $\theta^T u$,

$$y = f(\theta^T u).$$
Generalized linear regression

We can generalize this by introducing nonlinear transformations of the predictor $\theta^T u$,

$$y = f(\theta^T u).$$

Let us consider an example of a feed-forward NN, indicating that the information flows from the input to the output layer.
NN for regression – an example

1. Form $M$ linear combinations of the input $\mathbf{u} \in \mathbb{R}^D$

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} u_i + b_j^{(1)}, \quad j = 1, \ldots, M.$$
1. Form $M$ linear combinations of the input $\mathbf{u} \in \mathbb{R}^D$

\[
a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} u_i + b_j^{(1)}, \quad j = 1, \ldots, M.
\]

2. Apply a nonlinear transformation

\[
z_j = f \left( a_j^{(1)} \right), \quad j = 1, \ldots, M.
\]
NN for regression – an example

1. Form $M$ linear combinations of the input $\mathbf{u} \in \mathbb{R}^D$

$$a^{(1)}_j = \sum_{i=1}^{D} w^{(1)}_{ji} u_i + b^{(1)}_j, \quad j = 1, \ldots, M.$$  

2. Apply a nonlinear transformation

$$z_j = f \left( a^{(1)}_j \right), \quad j = 1, \ldots, M.$$  

3. Form $M_y$ linear combinations of $\mathbf{z} \in \mathbb{R}^M$

$$y_k = \sum_{j=1}^{M} w^{(2)}_{kj} z_j + b^{(2)}_k, \quad k = 1, \ldots, M_y.$$
NN for regression – an example

\[ \hat{y}_k(\theta) = \sum_{j=1}^{M} w_{kj}^{(2)} f \left( \sum_{i=1}^{D} w_{ji}^{(1)} u_i + b_j^{(1)} \right) + b_k^{(2)} \]

**Inputs**  **Hidden layer**  **Output layer**

\[ u_1 \rightarrow w_{11}^{(1)} \rightarrow z_1 \rightarrow \hat{y}_1 \]
\[ u_2 \rightarrow \cdots \rightarrow \hat{y}_M \]
\[ uD \rightarrow \cdots \rightarrow \hat{y}_M \]
Multi-layer neural networks

We can think of the neural network as a sequential/recursive construction of several generalized linear regressions.

Each layer in a multi-layer NN is modelled as 

\[ z^{(l+1)} = f(W^{(l+1)}z^{(l)}) + b^{(l+1)} \],

starting with the input 

\[ z^{(0)} = u \]. (The nonlinearity operates element-wise.)

The scalar nonlinear function \( f(\cdot) \) is what makes the neural network nonlinear. Common functions are 

\[ f(z) = \frac{1}{1 + e^{-z}} \],

\[ f(z) = \tanh(z) \] and 

\[ f(z) = \max(0, z) \].

The so-called rectified linear unit (ReLU) \( f(z) = \max(0, z) \) is heavily used for deep architectures.
Multi-layer neural networks

We can think of the neural network as a sequential/recursive construction of several generalized linear regressions.

Each layer in a multi-layer NN is modelled as

\[ z^{(l+1)} = f \left( W^{(l+1)} z^{(l)} + b^{(l+1)} \right), \]

starting with the input \( z^{(0)} = u \). (The nonlinearity operates element-wise.)
Multi-layer neural networks

We can think of the neural network as a sequential/recursive construction of several generalized linear regressions.

Each layer in a multi-layer NN is modelled as

\[ z^{(l+1)} = f \left( W^{(l+1)} z^{(l)} + b^{(l+1)} \right), \]

starting with the input \( z^{(0)} = u \). (The nonlinearity operates element-wise.)

The scalar nonlinear function \( f(\cdot) \) is what makes the neural network nonlinear. Common functions are \( f(z) = 1/(1 + e^{-z}) \), \( f(z) = \tanh(z) \) and \( f(z) = \max(0, z) \).

The so-called **rectified linear unit (ReLU)** \( f(z) = \max(0, z) \) is heavily used for deep architectures.
Training a NN

The final layer $z^{(L)}$ of the network is used for making a prediction $\hat{y}(\theta) = z^{(L)}$ and we train the network by employing:

1. A set of training data.
2. A cost function $\mathcal{L}(\hat{y}(\theta), y)$.
3. An iterative scheme to optimize the cost function

$$J(\theta) = \sum_{n=1}^{N} \mathcal{L}(\hat{y}_n(\theta), y_n).$$
The final layer $z^{(L)}$ of the network is used for making a prediction $\hat{y}(\theta) = z^{(L)}$ and we train the network by employing:

1. A set of training data.
2. A cost function $L(\hat{y}(\theta), y)$.
3. An iterative scheme to optimize the cost function

$$J(\theta) = \sum_{n=1}^{N} L(\hat{y}_n(\theta), y_n).$$

Training a NN does involve a lot of engineering skill and is more of an art than a mathematically rigorous exercise.
Backpropagation

Recall our example network again:

\[
\hat{y}_k(\theta) = \sum_{j=1}^{M} w_{kj}^{(2)} f \left( \sum_{i=1}^{D} w_{ji}^{(1)} u_i + b_j^{(1)} \right) + b_k^{(2)}
\]
Backpropagation

Recall our example network again:

\[ \hat{y}_k(\theta) = \sum_{j=1}^{M} w_{kj}^{(2)} f \left( \sum_{i=1}^{D} w_{ji}^{(1)} u_i + b_j^{(1)} \right) + b_k^{(2)} \]

In solving the optimization problem

\[ \hat{\theta} = \arg \min_{\theta} J(\theta) \]

we typically employ gradient methods using \( \nabla J(\theta) \).
Backpropagation

Recall our example network again:

\[ \hat{y}_k(\theta) = \sum_{j=1}^{M} w_{kj}^{(2)} f \left( \sum_{i=1}^{D} w_{ji}^{(1)} u_i + b_j^{(1)} \right) + b_k^{(2)} \]

In solving the optimization problem

\[ \hat{\theta} = \arg \min_{\theta} J(\theta) \]

we typically employ gradient methods using \( \nabla J(\theta) \).

Backpropagation amounts to computing the gradients via (recursive) use of the chain rule, combined with reuse of information that is needed for more than one gradient.
Tuning the model complexity

A neural network is a nonlinear parametric model that is built by recursively applying generalized linear regression,

\[ \hat{y} = f^{(L)} \circ \cdots \circ f^{(1)} \circ f^{(0)}(u). \]
Tuning the model complexity

A neural network is a nonlinear parametric model that is built by recursively applying generalized linear regression,

\[ \hat{y} = f^{(L)} \circ \cdots \circ f^{(1)} \circ f^{(0)}(u). \]

**Problem:** As with any parametric method **overfitting** will occur if the number of free parameters is too large w.r.t. the training data. The model complexity typically needs to be **tuned**.
Tuning the model complexity

A neural network is a nonlinear parametric model that is built by recursively applying generalized linear regression,

\[ \hat{y} = f^{(L)} \circ \cdots \circ f^{(1)} \circ f^{(0)}(u). \]

**Problem:** As with any parametric method overfitting will occur if the number of free parameters is too large w.r.t. the training data. The model complexity typically needs to be tuned.

**Weight decay:** Regularize using an Euclidean norm

\[ \tilde{J}(\theta) = J(\theta) + \lambda \|\theta\|^2. \]
Tuning the model complexity

A neural network is a nonlinear parametric model that is built by recursively applying generalized linear regression,

\[ \hat{y} = f^{(L)} \circ \cdots \circ f^{(1)} \circ f^{(0)}(u). \]

**Problem:** As with any parametric method overfitting will occur if the number of free parameters is too large w.r.t. the training data. The model complexity typically needs to be tuned.

**Weight decay:** Regularize using an Euclidean norm

\[ \tilde{J}(\theta) = J(\theta) + \lambda \| \theta \|^2. \]

**Weight elimination:** Regularize using a zero-forcing term \( h(\cdot) \)

\[ \tilde{J}(\theta) = J(\theta) + \lambda h(\theta). \]
Networks with built-in constraints

**Weight sharing** is a constraint that forces certain connections in the network to have the same weights.
Networks with built-in constraints

**Weight sharing** is a constraint that forces certain connections in the network to have the same weights.

**Convolutional networks (ConvNets)** Makes use of the weight sharing idea. Nodes form groups of 2D arrays.

Particularly successful in machine vision.

The convNet is a notable early successful deep architecture.
Deep neural networks

A neural network (NN) is a nonlinear function $y = g_\theta(u)$ from an input variable $u$ to an output variable $y$ parameterized by $\theta$.

A sequential construction of several generalized linear regressions,

$$z^{(l+1)} = f^{(l)} \left( W^{(l+1)} z^{(l)} + b^{(l+1)} \right), \quad z^{(0)} = u.$$
A neural network (NN) is a nonlinear function $y = g_\theta(u)$ from an input variable $u$ to an output variable $y$ parameterized by $\theta$.

A sequential construction of several generalized linear regressions,

$$z^{(l+1)} = f^{(l)} \left( W^{(l+1)} z^{(l)} + b^{(l+1)} \right), \quad z^{(0)} = u.$$ 

Think of this as multiple levels of representation (features).
A neural network (NN) is a nonlinear function $y = g_\theta(u)$ from an input variable $u$ to an output variable $y$ parameterized by $\theta$.

A sequential construction of several generalized linear regressions,

$$z^{(l+1)} = f^{(l)} \left( W^{(l+1)} z^{(l)} + b^{(l+1)} \right), \quad z^{(0)} = u.$$

Think of this as multiple levels of representation (features).

**Key aspect:** The layers are not designed by humans, they are learned from (typically lots of) data.

**Key idea:** (10 years old) Initialization by training each layer individually using an unsupervised algorithm.
Hierarchy of features

Example: Image classification

The input layer represents an image and the output layer an object identity. Each hidden layer extracts increasingly abstract features.

Training deep neural networks

The main problem with a deep architecture is the training. The strategy sketched above will not work.

The breakthrough came 10 years ago:


The main problem with a deep architecture is the training. The strategy sketched above will not work.

The breakthrough came 10 years ago:


**Key idea:** Careful initialization by training each layer individually using an unsupervised algorithm. Referred to as *pre-training*.

Finally, a supervised algorithm (e.g. backpropagation) is used to fine-tune the parameters $\theta$ using the result from the pre-training as initial values.
Outline

1. Introduction via three recent DL applications
2. What is a neural network (NN)?
   a) Concrete example for regression
   b) Learning and regularization
3. What is a deep neural network?
4. Learning deep neural networks
   a) Pre-training
   b) Defining and learning the autoencoder
5. Deep reinforcement learning via an example
   a) Pixels-to-torques problem
   b) Deep autoencoder
   c) Developing a deep dynamical model
6. Solving an integral – a rendering application
Problem formulation

**Vision:** Systems learning by themselves from raw pixel data.

**Problem formulation:** Modeling of high-dim. pixel data

**Strategy:** Construct a **deep dynamical model** that contains a low-dimensional dynamical model.

**Example:** Video stream of a pendulum

- **Input:** Torque of a pendulum
- **Output:** Pixel values of an $11 \times 11$ image
Model component – deep autoencoder

Unsupervised learning procedure for dimensionality reduction.

Notation:

- \( y_k \) - High-dim. observations
- \( z_k \) - Low-dim. features
- \( \dim(y_k) \gg \dim(z_k) \)
Notation:
- $y_k$ - High-dim. observations
- $z_k$ - Low-dim. features
- $\text{dim}(y_k) \gg \text{dim}(z_k)$

Model components:
1. Encoder: $z_k = g^{-1}(y_k; \theta_E)$
Model component – deep autoencoder

Unsupervised learning procedure for **dimensionality reduction**.

**Notation:**
- \( y_k \) - High-dim. observations
- \( z_k \) - Low-dim. features
- \( \dim(y_k) \gg \dim(z_k) \)

**Model components:**
1. **Encoder:** \( z_k = g^{-1}(y_k; \theta_E) \)
2. **Decoder:** \( \hat{y}_k^R = g(z_k; \theta_D) \)
Model component – deep autoencoder

Unsupervised learning procedure for **dimensionality reduction**.

**Notation:**
- $y_k$ - High-dim. observations
- $z_k$ - Low-dim. features
- $\dim(y_k) \gg \dim(z_k)$

Model components:
1. **Encoder:** $z_k = g^{-1}(y_k; \theta_E)$
2. **Decoder:** $\hat{y}_k^R = g(z_k; \theta_D)$

**Reconstruction error:**

$V_R(\theta_E, \theta_D) = \sum_{k=1}^{N} \| y_k - \hat{y}_k^R(\theta_E, \theta_D) \|^2$
Deep dynamical model

Notation:

• $y_k$ - High-dim. observations
• $z_k$ - Low-dim. features
• $u_k$ - Inputs
Deep dynamical model

Notation:
- \( y_k \) - High-dim. observations
- \( z_k \) - Low-dim. features
- \( u_k \) - Inputs

Model components:
1. Encoder: \( z_k = g^{-1}(y_k; \theta_E) \)
Deep dynamical model

Notation:
- \( y_k \) - High-dim. observations
- \( z_k \) - Low-dim. features
- \( u_k \) - Inputs

Model components:
1. Encoder: \( z_k = g^{-1}(y_k; \theta_E) \)
2. Prediction model: \( \hat{z}_{k+1|k} = f(z_k, u_k, \ldots, z_{k-n+1}, u_{k-n+1}; \theta_P) \)
Deep dynamical model

Notation:
- $y_k$ - High-dim. observations
- $z_k$ - Low-dim. features
- $u_k$ - Inputs

Model components:
1. Encoder: $z_k = g^{-1}(y_k; \theta_E)$
2. Prediction model: $\hat{z}_{k+1|k} = f(z_k, u_k, \ldots, z_{k-n+1}, u_{k-n+1}; \theta_P)$
3. Decoder: $\hat{y}_{k+1|k}^P = g(\hat{z}_{k+1|k}; \theta_D)$
Deep dynamical model

Notation:
- $y_k$ - High-dim. observations
- $z_k$ - Low-dim. features
- $u_k$ - Inputs

Model components:
1. Encoder: $z_k = g^{-1}(y_k; \theta_E)$
2. Prediction model: $\hat{z}_{k+1|k} = f(z_k, u_k, \ldots, z_{k-n+1}, u_{k-n+1}; \theta_P)$
3. Decoder: $\hat{y}_{k+1|k}^P = g(\hat{z}_{k+1|k}; \theta_D)$

Prediction error:
$V_P(\theta_E, \theta_D, \theta_P) = \sum_{k=n}^{N-1} \| y_{k+1} - \hat{y}_{k+1|k}^P(\theta_E, \theta_D, \theta_P) \|^2$
Training

Key ingredient: The reconstruction error and the prediction error are minimized simultaneously!

\[
(\hat{\theta}_E, \hat{\theta}_D, \hat{\theta}_P) = \arg \min_{\theta_E, \theta_D, \theta_P} V_R(\theta_E, \theta_D) + V_P(\theta_E, \theta_D, \theta_P)
\]

\[
V_R(\theta_E, \theta_D) = \sum_{k=1}^{N} \| y_k - \hat{y}_k^R(\theta_E, \theta_D) \|^2,
\]

\[
V_P(\theta_E, \theta_D, \theta_P) = \sum_{k=n}^{N-1} \| y_{k+1} - \hat{y}_{k+1|k}^P(\theta_E, \theta_D, \theta_P) \|^2.
\]
Experiment: agent in a planar system

- **Input:** Offset in $x$–dir. ($u_1$) and $y$–dir. ($u_2$)
- **Output:** Pixel values of a $51 \times 51$ image
- **Latent dim.:** $\dim(z) = 2$
Experiment: agent in a planar system

- **Input:** Offset in \(x\)-dir. \((u_1)\) and \(y\)-dir. \((u_2)\)
- **Output:** Pixel values of a 51 \(\times\) 51 image
- **Latent dim.:** \(\dim(z) = 2\)
Experiment: agent in a planar system

Simultaneous training means that we are training the network (features) jointly with the dynamics.
Experiment: agent in a planar system

Simultaneous training means that we are training the network (features) jointly with the dynamics.
Simultaneous training means that we are training the network (features) jointly with the dynamics.
Experiment: agent in a planar system

Simultaneous training means that we are training the network (features) jointly with the dynamics.
Control of two-link arm from pixels only

- Deep autoencoder network + nonlinear SSM
- Ref. image: Arm pointing upwards
- 1000 images in each trial
- After 8 – 9 trials a fairly good controller was learned.

Outline

1. Introduction via three recent DL applications
2. What is a neural network (NN)?
   a) Concrete example for regression
   b) Learning and regularization
3. What is a deep neural network?
4. Learning deep neural networks
   a) Pre-training
   b) Defining and learning the autoencoder
5. Deep reinforcement learning via an example
   a) Pixels-to-torques problem
   b) Deep autoencoder
   c) Developing a deep dynamical model
6. Solving an integral – a rendering application
Rendering in heterogeneous media

A Monte Carlo method that makes use of stochastically sampled light paths connecting the sensor with light sources in the scene.
Rendering in heterogeneous media

A Monte Carlo method that makes use of stochastically sampled light paths connecting the sensor with light sources in the scene.

Results using equal time rendering

Our method that builds on MLT

Metropolis light transport (MLT)

It comes down to computing integrals

The value $I_j$ of pixel $j$ is expressed as an integral over all possible light paths $\bar{x}$ in the scene (path space $\mathcal{P}$),

$$I_j = \int_{\mathcal{P}} f(\bar{x})\mu(x)dx.$$
It comes down to computing integrals

The value $I_j$ of pixel $j$ is expressed as an integral over all possible light paths $\bar{x}$ in the scene (path space $\mathcal{P}$),

$$I_j = \int_{\mathcal{P}} f(\bar{x}) \mu(x) dx.$$ 

How do we solve this integral?
It comes down to computing integrals

The value $I_j$ of pixel $j$ is expressed as an integral over all possible light paths $\bar{x}$ in the scene (path space $\mathcal{P}$),

$$I_j = \int_{\mathcal{P}} f(\bar{x}) \mu(x) dx.$$ 

How do we solve this integral?

Generate a Markov chain on path space to sample light paths proportional to their contribution $L(\bar{x}) \propto f(\bar{x})$. 

Computing integrals using Monte Carlo

**Idea underlying Monte Carlo:** Find a set of $N$ weighted samples (particles) $\{w^i, x^i\}_{i=1}^N$, approximating the target distribution

$$
\hat{\mu}(x) = \sum_{i=1}^{N} w^i \delta_{x^i}(x).
$$

This empirical distribution converge asymptotically ($N \to \infty$) to $\mu$ for any function $f$,

$$
\sum_{i=1}^{N} w^i f(x^i) \to \underbrace{\int f(x) \mu(x) dx}_{E_{\mu(x)}[f(x)]}.
$$


The pseudo-marginal construction

Recall our integral

\[ I_j = \int_{\mathcal{P}} f(\bar{x}) \mu(x) dx. \]

and \( L(\bar{x}) \propto f(\bar{x}) \).
The pseudo-marginal construction

Recall our integral

\[ I_j = \int_{\mathcal{P}} f(\bar{x}) \mu(x) dx. \]

and \( L(\bar{x}) \propto f(\bar{x}). \)

Pseudo-marginal Metropolis Hastings allows for **exact** computation of the target distribution, despite the use of an **unbiased estimator** \( \hat{L}(\bar{x}) \) for \( L(\bar{x}) \).

One more example

Results using equal time rendering

Our pseudo-marginal ERPT

The ASSEMBLE project

**Aim:** Creating a market place for inference/learning algorithms and probabilistic model libraries for dynamical systems.

- **Probabilistic Modeling Research**
  - Application Model
  - Inference Methods

- **Inference Methods Research**

- **Probabilistic Model Compiler**

- **Modeling Language Research**

- **Demonstrators**
  - Smart Meters (Greenely)
  - Cell Tracking (Karolinska Institute)
  - Energy-Aware Computing
  - Container Crane Automation (ABB)
  - Smart Automotive Safety (Autoliv)

- **Application Specific Machine Learning Solution**

Feedback from demonstrators enables:
- Improved modeling techniques
- Improved inference methods
- Enhanced modeling language
The ASSEMBLE project – research goals

1. Develop a formally defined probabilistic modeling language tailored for dynamical systems.

2. Construct probabilistic models representing complex dynamical systems that gain situational awareness in their environments using high-dimensional sensor data to automatically compute system controllers.

3. Automate complexity reducing techniques for inference in high-dimensional models.
The ASSEMBLE project – research goals

1. Develop a formally defined probabilistic modeling language tailored for dynamical systems.
2. Construct probabilistic models representing complex dynamical systems that gain situational awareness in their environments using high-dimensional sensor data to automatically compute system controllers.
3. Automate complexity reducing techniques for inference in high-dimensional models.

Time frame: 5 years, starting 1 July 2016.

Partners (academic): PI: Thomas Schön (Machine learning, UU), co-PIs: Black-Schaffer (Computer architecture, UU), David Broman (Modeling languages, KTH) and Joakim Jaldén (Signal processing, KTH).
Conclusions

Introduced a deep dynamical model
Conclusions

Introduced a **deep dynamical model**

Use the model also for “attention” and control
  Reinforcement learning to decide **where** to look for new data
  (resulting in new knowledge).
Conclusions

Introduced a **deep dynamical model**

Use the model also for “attention” and control
Reinforcement learning to decide **where** to look for new data
(resulting in new knowledge).

Built a Markov chain to sample light paths
Used the chain to **render images** in heterogeneous media.
Conclusions

Introduced a **deep dynamical model**

Use the model also for “attention” and control

  Reinforcement learning to decide **where** to look for new data
  (resulting in new knowledge).

Built a Markov chain to sample light paths

  Used the chain to **render images** in heterogeneous media.
Conclusions

Introduced a deep dynamical model

Use the model also for “attention” and control
   Reinforcement learning to decide where to look for new data
   (resulting in new knowledge).

Built a Markov chain to sample light paths
   Used the chain to render images in heterogeneous media.

Two additional trends that we can talk about some other time:
   1. Bayesian nonparametric models, like (GP, DP, etc.)
   2. Bayesian optimization

Remember to talk to people who work on different problems with different tools!!