Pixels to Torques: Control using Deep Dynamical Models

Niklas Wahlström[1], John-Alexander M. Assael[3], Thomas B. Schön[2], Marc P. Deisenroth[3]

[1]Department of Electrical Engineering, Linköping University, Sweden

[2]Department of Information Technology, Uppsala University, Sweden

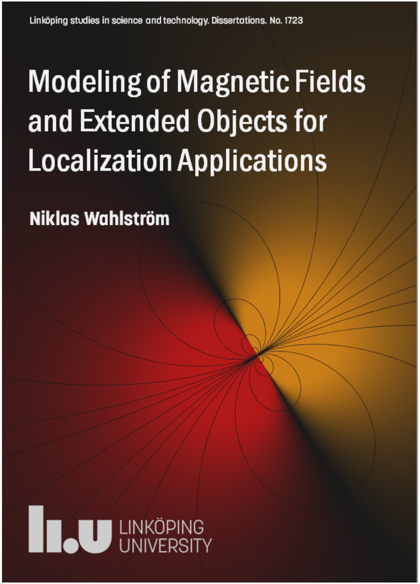
[3]Department of Computing, Imperial College London, UK

LINKÖPING UNIVERSITY

Short about me

- 2005 - 2010: Applied Physics and Electrical Engineering - International, Linköping University.
  - 2007-2008: Exchange student, ETH Zürich, Swizerland
- 2010-2015 : PhD student in Automatic Control, Linköping University
  - Spring 2014, Research visit, Imperial College, London, UK
- *2016- : Postdoc at Department of Information Technology, Uppsala University*

My thesis

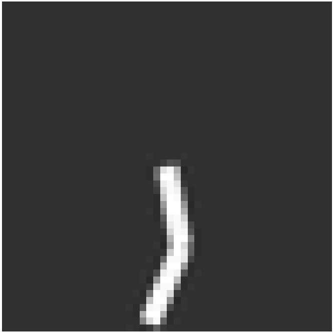


Three areas:

- Magnetic tracking
- Extended target tracking
- Deep dynamical models for control

Deep Learning: A recent example

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

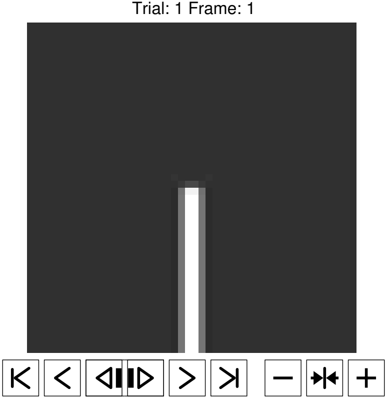


- Deep autoencoder network + nonlinear dynamical model
- Model predictive control (MPC)
- Ref. value: $\mathbf{z}_{\text{ref}} = f_{\text{d}}(\mathbf{y}_{\text{ref}})$
- The model is automatically improved (in an iterative manner)

J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth. **Data-Efficient Learning of Feedback Policies from Image Pixels using Deep Dynamical Models.** In *Deep Reinforc. Learning WS at the Conference on Neural Information Processing Systems (NIPS)*, Montréal, Canada, Dec. 2015.

Deep Learning: A recent example

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



- Deep autoencoder network + nonlinear dynamical model
- Model predictive control (MPC)
- Ref. value: $\mathbf{z}_{\text{ref}} = f_{\text{d}}(\mathbf{y}_{\text{ref}})$
- The model is automatically improved (in an iterative manner)

J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth. **Data-Efficient Learning of Feedback Policies from Image Pixels using Deep Dynamical Models.** In *Deep Reinforc. Learning WS at the Conference on Neural Information Processing Systems (NIPS)*, Montréal, Canada, Dec. 2015.

Deep Learning: 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 woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

Xu, K., Lei Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R. Richard S. Zemel, R. S., and Bengio, Y. **Show, attend and tell: neural image caption generation with visual attention**. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Lille, France, July, 2015.

A few examples where it failed

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.

A woman is sitting at a table with a large pizza.

A man is talking on his cell phone while another man watches.

Deep learning: A *very* recent example

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

Silver, D. et al. Mastering the game of Go with deep neural networks and tree search, *Nature*, Vol 529, 484–489 (2016)

Outline

1. Introduction via three recent 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. Developing and learning a deep dynamical model
   a) Problem formulation
   b) Deep dynamical model
6. Some pointers, summary and the future

Constructing an NN for regression

A **neural network (NN)** is a nonlinear function $\mathbf{y} = \mathrm{g}_{\boldsymbol{\theta}}(\mathbf{u})$ from an input variable $\mathbf{u}$ to an output variable $\mathbf{y}$ parameterized by $\boldsymbol{\theta}$.

Linear regression

Constructing an NN for regression

A **neural network (NN)** is a nonlinear function $\mathbf{y} = \mathrm{g}_{\boldsymbol{\theta}}(\mathbf{u})$ from an input variable $\mathbf{u}$ to an output variable $\mathbf{y}$ parameterized by $\boldsymbol{\theta}$.

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

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

where ϵ is noise and $\boldsymbol{\theta}$ is the parameters composed by the "weights" w_i and the offset ("bias") term b,

$$\boldsymbol{\theta} = \begin{pmatrix} b & w_1 & w_2 & \cdots & w_D \end{pmatrix}^{\mathsf{T}},$$
$$\mathbf{u} = \begin{pmatrix} 1 & u_1 & u_2 & \cdots & u_D \end{pmatrix}^{\mathsf{T}}.$$

Generalized linear regression

We can generalize this by introducing nonlinear transformations of the predictor $\boldsymbol{\theta}^\mathsf{T}\mathbf{u}$,

$$y = f(\boldsymbol{\theta}^\mathsf{T}\mathbf{u}).$$

Generalized linear regression

We can generalize this by introducing nonlinear transformations of the predictor $\boldsymbol{\theta}^\mathsf{T}\mathbf{u}$,

$$y = f(\boldsymbol{\theta}^\mathsf{T}\mathbf{u}).$$

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 $\mathbf{u} \in \mathbb{R}^D$

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} u_i + b_j^{(1)}, \qquad j = 1, \dots, M.$$

NN for regression – an example

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

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

2. Apply a nonlinear transformation

$$z_j = f\left(a_j^{(1)}\right), \qquad j = 1, \ldots, M.$$

NN for regression – an example

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

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} u_i + b_j^{(1)}, \qquad j = 1, \dots, M.$$

2. Apply a nonlinear transformation

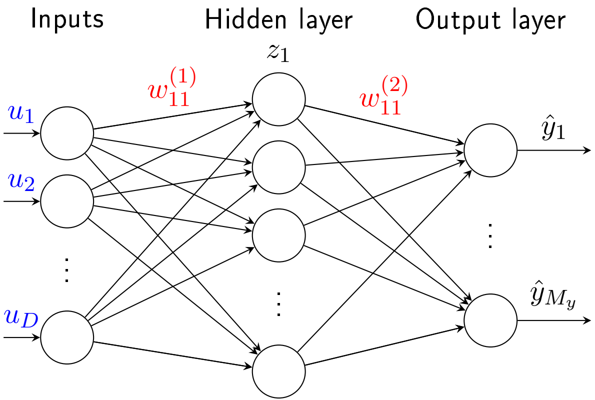
$$z_j = f\left(a_j^{(1)}\right), \qquad j = 1, \dots, M.$$

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

$$y_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)}, \qquad k = 1, \dots, M_y.$$

NN for regression – an example

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



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

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

starting with the input $\mathbf{z}^{(0)} = \mathbf{u}$. (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

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

starting with the input $\mathbf{z}^{(0)} = \mathbf{u}$. (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 $\mathbf{z}^{(L)}$ of the network is used for making a prediction $\hat{\mathbf{y}}(\boldsymbol{\theta}) = \mathbf{z}^{(L)}$ and we train the network by employing:

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

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{L}\left(\hat{\mathbf{y}}_n(\theta), \mathbf{y}_n\right).$$

Training a NN

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

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

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{L}\left(\hat{\mathbf{y}}_n(\theta), \mathbf{y}_n\right).$$

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

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \; J(\boldsymbol{\theta})$$

we typically employ gradient methods using $\nabla J(\boldsymbol{\theta})$.

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

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

we typically employ gradient methods using $\nabla J(\boldsymbol{\theta})$.

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,

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

Tuning the model complexity

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

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

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,

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

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

$$\widetilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|^2.$$

Tuning the model complexity

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

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

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

$$\widetilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|^2.$$

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

$$\widetilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda h(\boldsymbol{\theta}).$$

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 forms groups of 2D arrays.

Particularly successful in machine vision.

The convNet is a notable early successful deep architecture.

Outline

1. Introduction via three recent 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. Developing and learning a deep dynamical model
   a) Problem formulation
   b) Deep dynamical model
6. Some pointers, summary and the future

Deep neural networks

Deep learning methods allow a machine to make use of raw data to automatically discover the representations (abstractions) that are necessary to solve a particular task.

Deep neural networks

Deep learning methods allow a machine to make use of raw data to automatically discover the representations (abstractions) that are necessary to solve a particular task.

It is accomplished by using **multiple levels of representation**. Each level transforms the representation at the previous level into a new and more abstract representation,

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

starting from the input (raw data) $\mathbf{z}^{(0)} = \mathbf{u}$.

Deep neural networks

Deep learning methods allow a machine to make use of raw data to automatically discover the representations (abstractions) that are necessary to solve a particular task.

It is accomplished by using **multiple levels of representation**. Each level transforms the representation at the previous level into a new and more abstract representation,

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

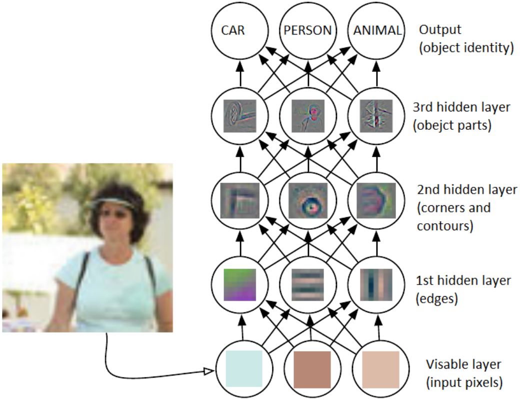
starting from the input (raw data) $\mathbf{z}^{(0)} = \mathbf{u}$.

Key aspect: The layers are **not** designed by human engineers, they are generated from (typically lots of) data using a learning procedure and lots of computations.

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.



Zeiler, M. D. and Fergus, R. **Visualizing and understanding convolutional networks**

Computer Vision - ECCV (2014).

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:

Hinton, G. E., Osindero, S. and Teh, Y-W. **A Fast Learning Algorithm for Deep Belief Nets**. *Neural Computation*, 18, 1527–1554, 2006.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. **Greedy layer-wise training of deep networks**. In *Proc. Advances in Neural Information Processing Systems (NIPS)* 19, 153–160, 2006.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. **Efficient learning of sparse representations with an energy-based model**. In *Proc. Advances in Neural Information Processing Systems (NIPS)* 19, 1137–1144, 2006.

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:

Hinton, G. E., Osindero, S. and Teh, Y-W. **A Fast Learning Algorithm for Deep Belief Nets**. *Neural Computation*, 18, 1527–1554, 2006.

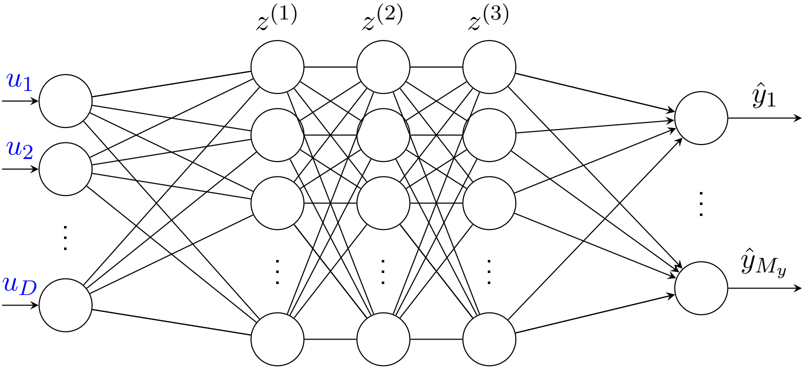
Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. **Greedy layer-wise training of deep networks**. In *Proc. Advances in Neural Information Processing Systems (NIPS)* 19, 153–160, 2006.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. **Efficient learning of sparse representations with an energy-based model**. In *Proc. Advances in Neural Information Processing Systems (NIPS)* 19, 1137–1144, 2006.

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 θ using the result from the pre-training as initial values.

Pre-training

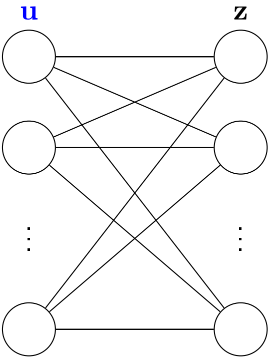


Pre-training evolves sequentially from input to output. Here:

- 3 stages of unsupervised training
- 1 stage of supervised training

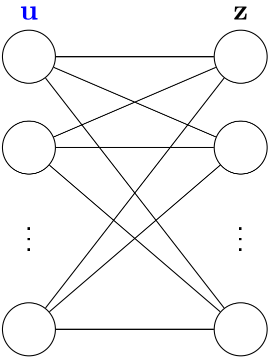
Pre-training – RBM

Restricted Boltzmann machine (RBM): an undir. graphical model with no connections among nodes of the same layer.



Pre-training – RBM

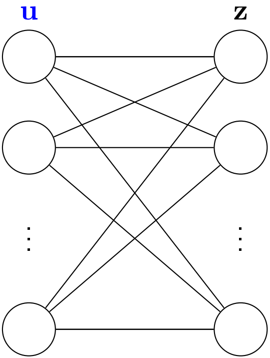
Restricted Boltzmann machine (RBM): an undir. graphical model with no connections among nodes of the same layer.



We have an observed input layer $\mathbf{u}$ and an unobserved output layer $\mathbf{z}$.

Pre-training – RBM

Restricted Boltzmann machine (RBM): an undir. graphical model with no connections among nodes of the same layer.



We have an observed input layer $\mathbf{u}$ and an unobserved output layer $\mathbf{z}$.

Training strategy: Maximum likelihood

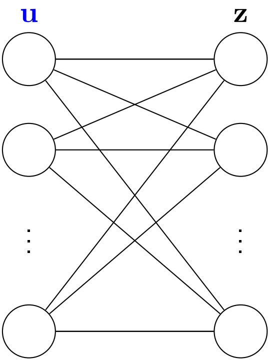
$$\widehat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{u}),$$

where $p_{\boldsymbol{\theta}}(\mathbf{u})$ is found via marginalization,

$$p_{\boldsymbol{\theta}}(\mathbf{u}) = \int p_{\boldsymbol{\theta}}(\mathbf{u}, \mathbf{z}) \mathrm{d}\mathbf{z}.$$

Pre-training – RBM

Restricted Boltzmann machine (RBM): an undir. graphical model with no connections among nodes of the same layer.



We have an observed input layer $\mathbf{u}$ and an unobserved output layer $\mathbf{z}$.

Training strategy: Maximum likelihood

$$\widehat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{u}),$$

where $p_{\boldsymbol{\theta}}(\mathbf{u})$ is found via marginalization,

$$p_{\boldsymbol{\theta}}(\mathbf{u}) = \int p_{\boldsymbol{\theta}}(\mathbf{u}, \mathbf{z}) \mathrm{d}\mathbf{z}.$$

The RBM is a **generative model**, which implies that we can simulate the output, which is then the input to the next layer.

Intuitive interpretation

Interpret the hidden layers as feature vectors and think of the deep architecture as a scheme for learning a **hierarchy of features**.

Intuitive interpretation

Interpret the hidden layers as feature vectors and think of the deep architecture as a scheme for learning a **hierarchy of features**.

We leave the feature generation, as much as possible, to the machine.

Intuitive interpretation

Interpret the hidden layers as feature vectors and think of the deep architecture as a scheme for learning a **hierarchy of features**.

We leave the feature generation, as much as possible, to the machine.

The unsupervised learning in the pre-training step is a way of discovering information hidden in the data. This is a way of learning underlying regularities in the data.

Intuitive interpretation

Interpret the hidden layers as feature vectors and think of the deep architecture as a scheme for learning a **hierarchy of features**.

We leave the feature generation, as much as possible, to the machine.

The unsupervised learning in the pre-training step is a way of discovering information hidden in the data. This is a way of learning underlying regularities in the data.

Pre-training can in this way be thought of as a **regularizer** that forces the parameters to "good" regions, by exploiting extra information from the unsupervised learning stage.

Intuitive interpretation

Interpret the hidden layers as feature vectors and think of the deep architecture as a scheme for learning a **hierarchy of features**.

We leave the feature generation, as much as possible, to the machine.

The unsupervised learning in the pre-training step is a way of discovering information hidden in the data. This is a way of learning underlying regularities in the data.

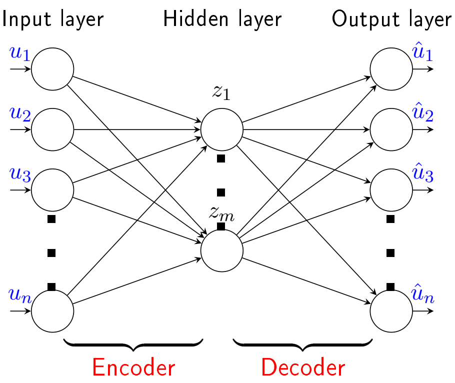
Pre-training can in this way be thought of as a **regularizer** that forces the parameters to “good” regions, by exploiting extra information from the unsupervised learning stage.

There is still **no theoretical justification** as to why these deep networks exhibit such good generalization performance.... That is a good problem to solve.

Autoencoder

The autoencoder is an unsupervised learning procedure for **dimensionality reduction**.

It is a NN that learns compressed representations $\mathbf{z}$ of high-dimensional data $\mathbf{u}$, where $\dim(\mathbf{u}) \gg \dim(\mathbf{z})$.



Encoder: $\mathbf{z} = \mathbf{f}_\mathsf{e}(\mathbf{u}) = \mathbf{f}(W^\mathsf{T}\mathbf{u} + \mathbf{b})$.

Decoder: $\hat{\mathbf{u}} = \mathbf{f}_\mathsf{d}(\mathbf{z}) = \mathbf{f}(\bar{W}^\mathsf{T}\mathbf{z} + \bar{\mathbf{b}})$.

Training the autoencoder

The unknown parameters

$$\boldsymbol{\theta} = \{W, \mathbf{b}, \bar{W}, \bar{\mathbf{b}}\}$$

are estimated by minimizing the reconstruction error

$$\mathbf{e} = \mathbf{u} - \hat{\mathbf{u}}(\boldsymbol{\theta}),$$

using some cost function $J(\boldsymbol{\theta})$, for example LS

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \|\mathbf{u}_n - \hat{\mathbf{u}}_n(\boldsymbol{\theta})\|^2.$$

Training the autoencoder

The unknown parameters

$$\boldsymbol{\theta} = \{W, \mathbf{b}, \bar{W}, \bar{\mathbf{b}}\}$$

are estimated by minimizing the reconstruction error

$$\mathbf{e} = \mathbf{u} - \hat{\mathbf{u}}(\boldsymbol{\theta}),$$

using some cost function $J(\boldsymbol{\theta})$, for example LS

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \|\mathbf{u}_n - \hat{\mathbf{u}}_n(\boldsymbol{\theta})\|^2.$$

After the training the encoder and the decoder will (by construction) be approximate inverses of each other,

$$\mathbf{f}_{\mathsf{d}}(\mathbf{f}_{\mathsf{e}}(\mathbf{u})) \approx \mathbf{u}.$$

Autoencoder

We can then easily transform either u into z or z into $\hat{u}$ using either the encoder

$$\mathbf{z} = \mathbf{f}_{\mathsf{e}}(W^{\mathsf{T}}\mathbf{u} + \mathbf{b}),$$

or the decoder,

$$\hat{\mathbf{u}} = \mathbf{f}_{\mathsf{d}}(\bar{W}^{\mathsf{T}}\mathbf{z} + \bar{\mathbf{b}}).$$

The access to both of these two mappings is important for certain applications (such as the deep dynamical model).

Autoencoder

We can then easily transform either u into z or z into $\hat{u}$ using either the encoder

$$\mathbf{z} = \mathbf{f}_{\mathsf{e}}(W^{\mathsf{T}}\mathbf{u} + \mathbf{b}),$$

or the decoder,

$$\hat{\mathbf{u}} = \mathbf{f}_{\mathsf{d}}(\bar{W}^{\mathsf{T}}\mathbf{z} + \bar{\mathbf{b}}).$$

The access to both of these two mappings is important for certain applications (such as the deep dynamical model).

If $\mathbf{f}_{\mathsf{e}}(\cdot)$ is chosen to be the identity (i.e. $\mathbf{z} = W^{\mathsf{T}}\mathbf{u} + \mathbf{b}$) and $\dim \mathbf{u} < \dim \mathbf{z}$ then the autoencoder is **equivalent to PCA.** Hence, the autoencoder is a nonlinear generalization of PCA.

Deep autoencoder

The deep autoencoder is simply an autoencoder with several hidden layers.

Again, careful initialization is important for this to work, using the same pre-training as described before.

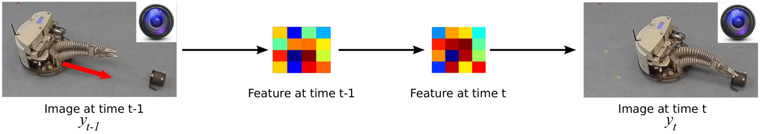
Hinton, G. E. and Salakhutdinov, R. R. **Reducing the Dimensionality of Data with Neural Networks**. Science, 313, 504–507, 2006.

Outline

1. Introduction via three recent 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. **Developing and learning a deep dynamical model**
   a) Problem formulation
   b) Deep dynamical model
6. Some pointers, summary and the future

Motivation

- **Vision:** fully autonomous systems that learn by themselves from raw pixel data.
- **This work:** Modeling of high-dimensional pixel data
- **Strategy:** A deep dynamical model is proposed that contains a low-dimensional dynamical model.



N. Wahlström, T. B. Schön, M. P. Deisenroth **Learning deep dynamical models from image pixels**

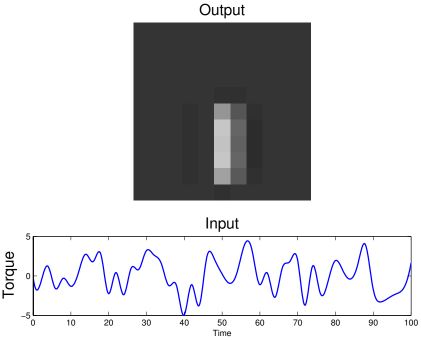
The 17th IFAC Symposium on System Identification (SYSID)

Problem Formulation

Problem formulation: Modeling of high-dimensional pixel data

Example: Video stream of a pendulum

- **Input:** Torque of a pendulum
- **Output:** Pixel values of an 11×11 image

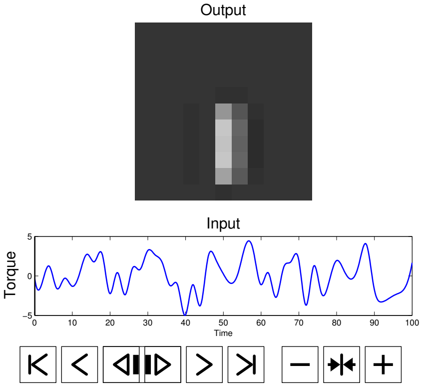


Problem Formulation

Problem formulation: Modeling of high-dimensional pixel data

Example: Video stream of a pendulum

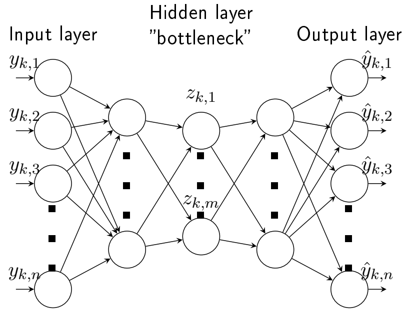
- **Input:** Torque of a pendulum
- **Output:** Pixel values of an 11×11 image



The Autoencoder

Notation:

- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features



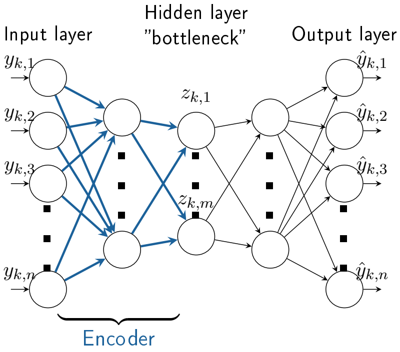
The Autoencoder

Notation:

- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features

Model components:

1. Encoder: $\mathbf{z}_k = \mathbf{f}_\mathsf{e}(\mathbf{y}_k; \boldsymbol{\theta}_\mathsf{E})$



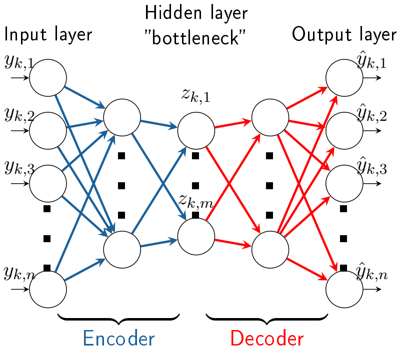
The Autoencoder

Notation:

- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features

Model components:

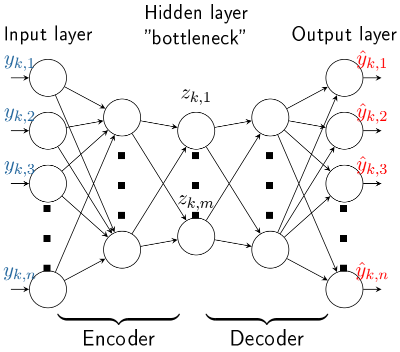
1. Encoder: $\mathbf{z}_k = \mathbf{f}_\mathsf{e}(\mathbf{y}_k; \boldsymbol{\theta}_\mathsf{E})$
2. Decoder: $\widehat{\mathbf{y}}_k^\mathsf{R} = \mathbf{f}_\mathsf{d}(\mathbf{z}_k; \boldsymbol{\theta}_\mathsf{D})$



The Autoencoder

Notation:

- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features



Model components:

1. Encoder: $\mathbf{z}_k = \mathbf{f}_\mathsf{e}(\mathbf{y}_k; \boldsymbol{\theta}_\mathsf{E})$
2. Decoder: $\widehat{\mathbf{y}}_k^\mathsf{R} = \mathbf{f}_\mathsf{d}(\mathbf{z}_k; \boldsymbol{\theta}_\mathsf{D})$

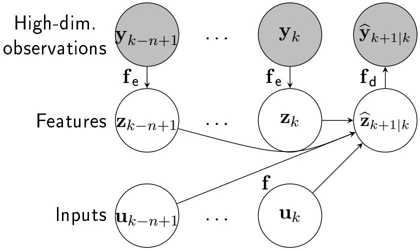
Reconstruction error:

$$V_\mathsf{R}(\boldsymbol{\theta}_\mathsf{E}, \boldsymbol{\theta}_\mathsf{D}) = \sum_{k=1}^{N} \|\mathbf{y}_k - \widehat{\mathbf{y}}_k^\mathsf{R}(\boldsymbol{\theta}_\mathsf{E}, \boldsymbol{\theta}_\mathsf{D})\|^2$$

Deep Dynamical Model

Notation:

- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features
- $\mathbf{u}_k$ - Inputs



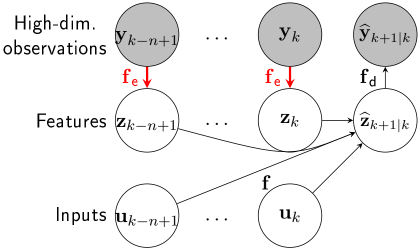
Deep Dynamical Model

Notation:

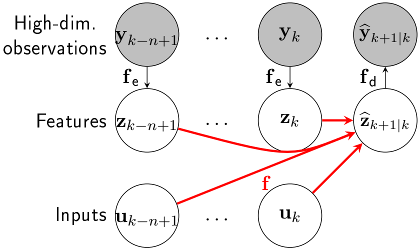
- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features
- $\mathbf{u}_k$ - Inputs

Model components:

1. Encoder: $\mathbf{z}_k = \mathbf{f}_{\mathsf{e}}(\mathbf{y}_k; \boldsymbol{\theta}_{\mathsf{E}})$



Deep Dynamical Model



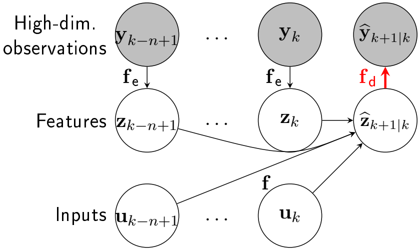
Notation:

- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features
- $\mathbf{u}_k$ - Inputs

Model components:

1. Encoder: $\mathbf{z}_k = \mathbf{f}_\mathsf{e}(\mathbf{y}_k; \boldsymbol{\theta}_\mathsf{E})$
2. Prediction model: $\widehat{\mathbf{z}}_{k+1|k} = \mathbf{f}(\mathbf{z}_k, \mathbf{u}_k, \ldots, \mathbf{z}_{k-n+1}, \mathbf{u}_{k-n+1}; \boldsymbol{\theta}_\mathsf{P})$

Deep Dynamical Model



Notation:

- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features
- $\mathbf{u}_k$ - Inputs

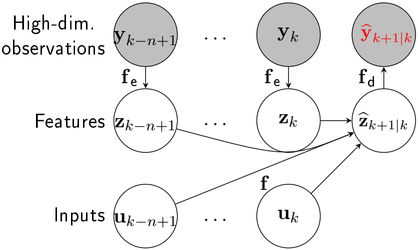
Model components:

1. Encoder: $\mathbf{z}_k = \mathbf{f}_\mathsf{e}(\mathbf{y}_k; \boldsymbol{\theta}_\mathsf{E})$
2. Prediction model: $\widehat{\mathbf{z}}_{k+1|k} = \mathbf{f}(\mathbf{z}_k, \mathbf{u}_k, \dots, \mathbf{z}_{k-n+1}, \mathbf{u}_{k-n+1}; \boldsymbol{\theta}_\mathsf{P})$
3. Decoder: $\widehat{\mathbf{y}}^\mathsf{P}_{k+1|k} = \mathbf{f}_\mathsf{d}(\widehat{\mathbf{z}}_{k+1|k}; \boldsymbol{\theta}_\mathsf{D})$

Deep Dynamical Model

Notation:

- $\mathbf{y}_k$ - High-dim. observations
- $\mathbf{z}_k$ - Low-dim. features
- $\mathbf{u}_k$ - Inputs



Model components:

1. Encoder: $\mathbf{z}_k = \mathbf{f}_\mathsf{e}(\mathbf{y}_k; \boldsymbol{\theta}_\mathsf{E})$
2. Prediction model: $\widehat{\mathbf{z}}_{k+1|k} = \mathbf{f}(\mathbf{z}_k, \mathbf{u}_k, \dots, \mathbf{z}_{k-n+1}, \mathbf{u}_{k-n+1}; \boldsymbol{\theta}_\mathsf{P})$
3. Decoder: $\widehat{\mathbf{y}}^\mathsf{P}_{k+1|k} = \mathbf{f}_\mathsf{d}(\widehat{\mathbf{z}}_{k+1|k}; \boldsymbol{\theta}_\mathsf{D})$

Prediction error:

$$V_\mathsf{P}(\boldsymbol{\theta}_\mathsf{E}, \boldsymbol{\theta}_\mathsf{D}, \boldsymbol{\theta}_\mathsf{P}) = \sum_{k=n}^{N-1} \|\mathbf{y}_{k+1} - \widehat{\mathbf{y}}^\mathsf{P}_{k+1|k}(\boldsymbol{\theta}_\mathsf{E}, \boldsymbol{\theta}_\mathsf{D}, \boldsymbol{\theta}_\mathsf{P})\|^2$$

Training

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

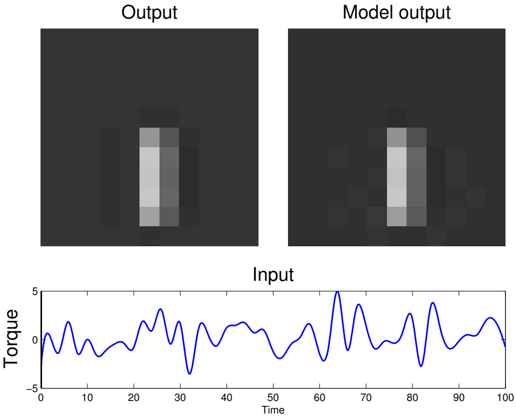
$$(\widehat{\boldsymbol{\theta}}_{\mathsf{E}}, \widehat{\boldsymbol{\theta}}_{\mathsf{D}}, \widehat{\boldsymbol{\theta}}_{\mathsf{P}}) = \arg\min_{\boldsymbol{\theta}_{\mathsf{E}},\boldsymbol{\theta}_{\mathsf{D}},\boldsymbol{\theta}_{\mathsf{P}}} V_{\mathsf{R}}(\boldsymbol{\theta}_{\mathsf{E}}, \boldsymbol{\theta}_{\mathsf{D}}) + V_{\mathsf{P}}(\boldsymbol{\theta}_{\mathsf{E}}, \boldsymbol{\theta}_{\mathsf{D}}, \boldsymbol{\theta}_{\mathsf{P}})$$

$$V_{\mathsf{R}}(\boldsymbol{\theta}_{\mathsf{E}}, \boldsymbol{\theta}_{\mathsf{D}}) = \sum_{k=1}^{N} \|\mathbf{y}_k - \widehat{\mathbf{y}}_k^{\mathsf{R}}(\boldsymbol{\theta}_{\mathsf{E}}, \boldsymbol{\theta}_{\mathsf{D}})\|^2,$$

$$V_{\mathsf{P}}(\boldsymbol{\theta}_{\mathsf{E}}, \boldsymbol{\theta}_{\mathsf{D}}, \boldsymbol{\theta}_{\mathsf{P}}) = \sum_{k=n}^{N-1} \|\mathbf{y}_{k+1} - \widehat{\mathbf{y}}_{k+1|k}^{\mathsf{P}}(\boldsymbol{\theta}_{\mathsf{E}}, \boldsymbol{\theta}_{\mathsf{D}}, \boldsymbol{\theta}_{\mathsf{P}})\|^2.$$

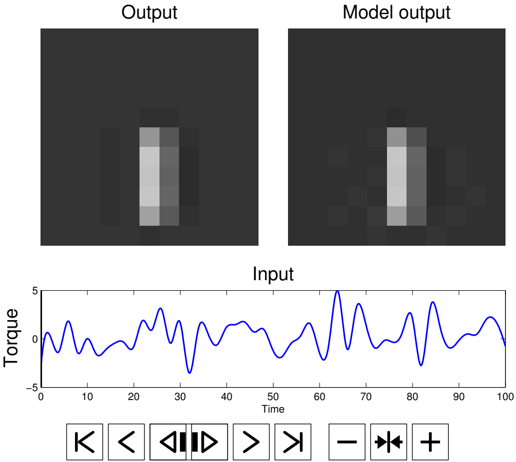
Experiment: Pendulum

- Layers in encoder/decoder: 4
- Latent dim.: $\dim(\mathbf{z}) = 1$
- Order of prediction model: $n = 4$



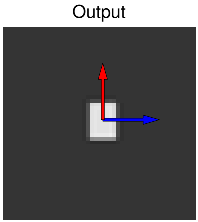
Experiment: Pendulum

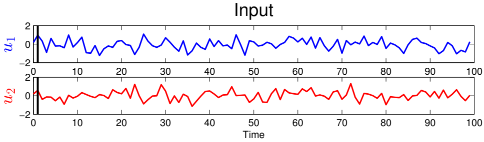
- Layers in encoder/decoder: 4
- Latent dim.: $\dim(\mathbf{z}) = 1$
- Order of prediction model: $n = 4$



Experiment: Agent in a Planar System

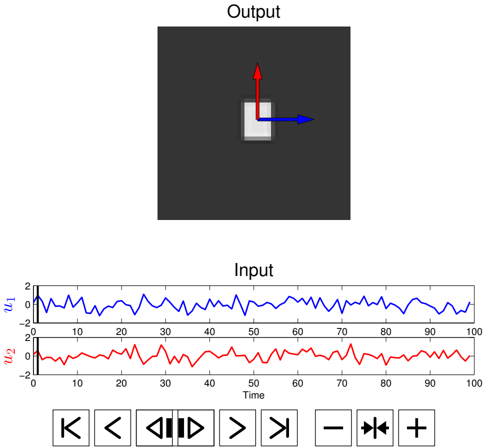
- **Input:** Offset in x−dir. (u_1) and y−dir. (u_2)
- **Output:** Pixel values of a 51×51 image
- **Latent dim.:** dim($\mathbf{z}$)=2





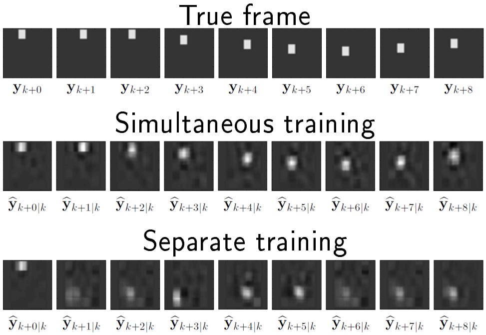
Experiment: Agent in a Planar System

- **Input:** Offset in x−dir. (u_1) and y−dir. (u_2)
- **Output:** Pixel values of a 51×51 image
- **Latent dim.:** dim($\mathbf{z}$)=2

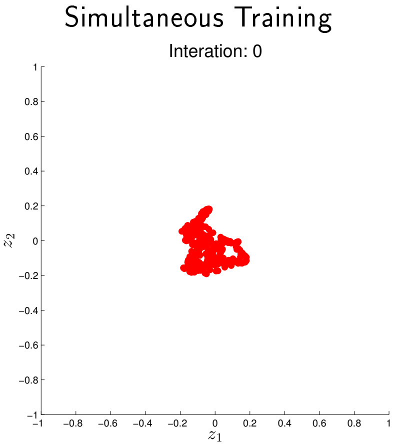


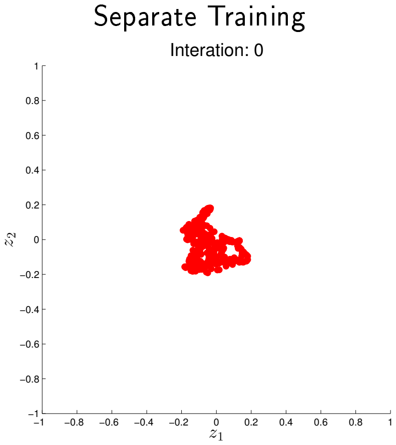
Experiment: Agent in a Planar System

Separate vs. Simultaneous Training

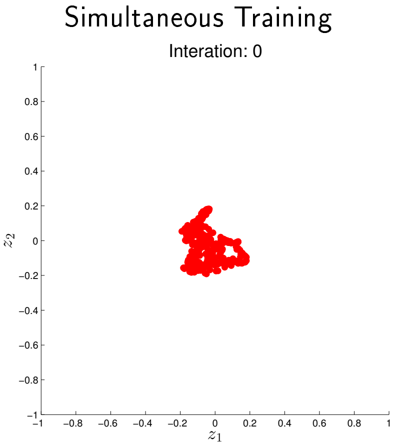


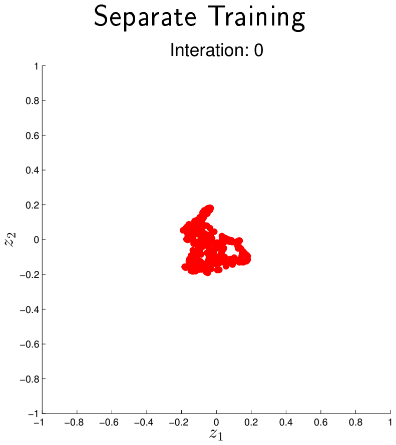
Experiment: Agent in a Planar System





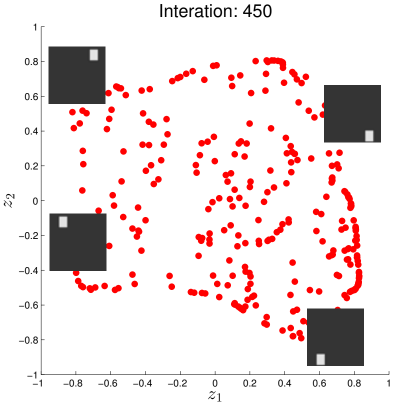
Experiment: Agent in a Planar System



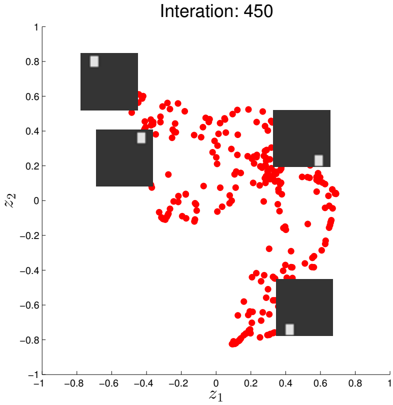


Experiment: Agent in a Planar System

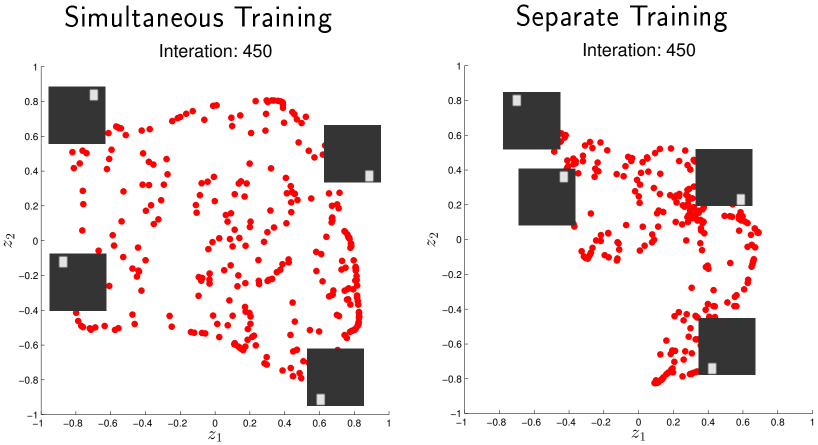
Simultaneous Training



Separate Training



Experiment: Agent in a Planar System



Deep Dynamical Models for Control

The DDM is used to learn a closed-loop policy via nonlinear **model predictive control (MPC)**. Future control signals are optimized by minimizing

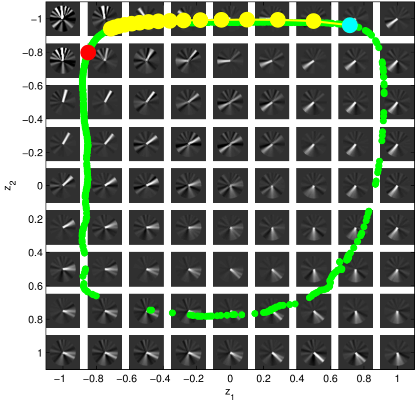
$$u_0^*, \dots, u_{K-1}^* \in \arg\min_{u_{0:K-1}} \sum_{k=0}^{K-1} \|\widehat{\mathbf{z}}_k - \mathbf{z}_{\text{ref}}\|^2 + \lambda \|u_k\|^2,$$

where $\mathbf{z}_{\text{ref}} = \mathbf{f}_e(y_{\text{ref},\boldsymbol{\theta}_e}$ is the feature of the reference image. When the control sequence $u_0^*, \dots, u_{K-1}^*$ is determined, the first control u_0^* is applied to the system.
Hence, the MPC is **only applied in the low-dimensional feature space!**

Deep Dynamical Models for Control

Proposed algorithm

Follow a random control strategy and record data
loop
  Update DDM with all data collected so far
  for $k = 0$ to $N-1$ **do**
    - Get $z_k, \ldots, z_{k-n+1}$ via encoder.
    - $u_k^* \leftarrow \epsilon$-greedy MPC policy using DDM prediction.
    - Apply u_k^* and record data.
  end for
end loop

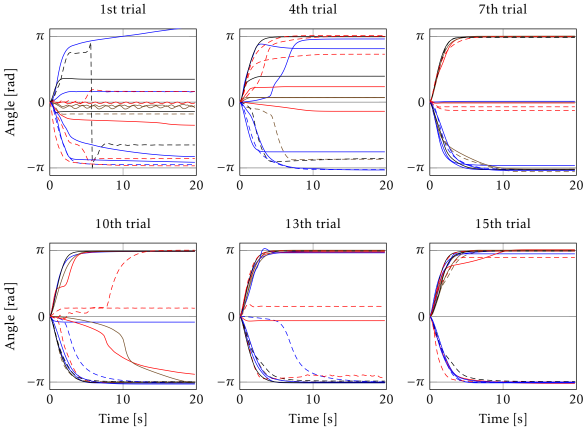


Green: Previous feature values
Cyan: Current feature
Red: Reference feature
Yellow: 15-step ahead prediction

N. Wahlström, T. B Schön, and M. P. Deisenroth
From Pixels to Torques: Policy Learning with Deep Dynamical Models. *ArXiv e-prints 1502.02251*

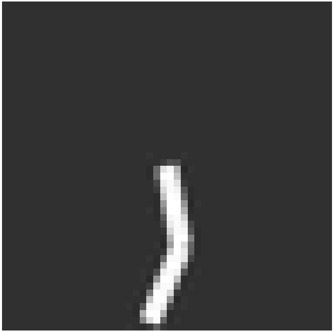
Experiment: Control of a Pendulum from Pixels Only

- Ref. image: Pendulum pointing upwards
- 100 images in each trial
- After 15 trials, a good controller was learned



Application: Control of Two-Link Arm from Pixels Only

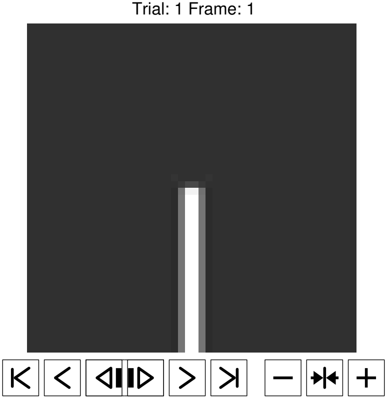
Trial: 3 Frame: 94



- Ref. image: Arm pointing upwards
- 1000 images in each trial
- After 8-9 trials a fairly good controller was learned.

J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth. **Data-Efficient Learning of Feedback Policies from Image Pixels using Deep Dynamical Models**. In *Deep Reinforc. Learning WS at the Conference on Neural Information Processing Systems (NIPS)*, Montréal, Canada, Dec. 2015.

Application: Control of Two-Link Arm from Pixels Only



- Ref. image: Arm pointing upwards
- 1000 images in each trial
- After 8-9 trials a fairly good controller was learned.

J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth. **Data-Efficient Learning of Feedback Policies from Image Pixels using Deep Dynamical Models.** In *Deep Reinforc. Learning WS at the Conference on Neural Information Processing Systems (NIPS)*, Montréal, Canada, Dec. 2015.

Outline

Some pointers

Key **publication channels** in machine learning, NIPS and ICML

From NIPS in December (`nips.cc/Conferences/2015`) three of the six tutorials deals with deep learning:

1. G. Hinton, Y. Bengio and Y. LeCun, Deep learning
   `https://nips.cc/Conferences/2015/Schedule?event=4891`
2. B. Dally, High-performance hardware for Machine Learning
   `nips.cc/Conferences/2015/Schedule?event=4894`
3. J. Dean, Large-scale distributed systems for training NN
   `nips.cc/Conferences/2015/Schedule?event=4895`

Some pointers

Key **publication channels** in machine learning, NIPS and ICML

From NIPS in December (`nips.cc/Conferences/2015`) three of the six tutorials deals with deep learning:

1. G. Hinton, Y. Bengio and Y. LeCun, Deep learning
   `https://nips.cc/Conferences/2015/Schedule?event=4891`
2. B. Dally, High-performance hardware for Machine Learning
   `nips.cc/Conferences/2015/Schedule?event=4894`
3. J. Dean, Large-scale distributed systems for training NN
   `nips.cc/Conferences/2015/Schedule?event=4895`

A well written and timely introduction:

LeCun, Y., Bengio, Y., and Hinton, G. **Deep learning**, *Nature*, Vol 521, 436–444.

Some pointers

Key **publication channels** in machine learning, NIPS and ICML

From NIPS in December (`nips.cc/Conferences/2015`) three of the six tutorials deals with deep learning:

1. G. Hinton, Y. Bengio and Y. LeCun, Deep learning
   `https://nips.cc/Conferences/2015/Schedule?event=4891`
2. B. Dally, High-performance hardware for Machine Learning
   `nips.cc/Conferences/2015/Schedule?event=4894`
3. J. Dean, Large-scale distributed systems for training NN
   `nips.cc/Conferences/2015/Schedule?event=4895`

A well written and timely introduction:

LeCun, Y., Bengio, Y., and Hinton, G. **Deep learning**, *Nature*, Vol 521, 436–444.

You will also find more material than you can possibly want here

`http://deeplearning.net/`

Summary (I/II)

A **neural network (NN)** is a nonlinear function $\mathbf{y} = \mathrm{g}_{\boldsymbol{\theta}}(\mathbf{u})$ from an input variable $\mathbf{u}$ to an output variable $\mathbf{y}$ parameterized by $\boldsymbol{\theta}$.

Summary (I/II)

A **neural network (NN)** is a nonlinear function $\mathbf{y} = \mathrm{g}_{\boldsymbol{\theta}}(\mathbf{u})$ from an input variable $\mathbf{u}$ to an output variable $\mathbf{y}$ parameterized by $\boldsymbol{\theta}$.

We can think of an NN as a sequential/recursive construction of several generalized linear regressions.

Summary (I/II)

A **neural network (NN)** is a nonlinear function $\mathbf{y} = \mathrm{g}_{\boldsymbol{\theta}}(\mathbf{u})$ from an input variable $\mathbf{u}$ to an output variable $\mathbf{y}$ parameterized by $\boldsymbol{\theta}$.

We can think of an NN as a sequential/recursive construction of several generalized linear regressions.

Deep learning refers to learning NNs with several hidden layers. Allows for data-driven models that automatically learns rep. of data (features) with multiple layers of abstraction.

Summary (I/II)

A **neural network (NN)** is a nonlinear function $\mathbf{y} = \mathrm{g}_{\boldsymbol{\theta}}(\mathbf{u})$ from an input variable $\mathbf{u}$ to an output variable $\mathbf{y}$ parameterized by $\boldsymbol{\theta}$.

We can think of an NN as a sequential/recursive construction of several generalized linear regressions.

Deep learning refers to learning NNs with several hidden layers. Allows for data-driven models that automatically learns rep. of data (features) with multiple layers of abstraction.

The deep **autoencoder** makes use of a multi-layer "encoder" network to transform high-dimensional data into a low-dimensional code/feature and a similar "decoder" network is used to recover the data from the code.

Summary (II/II)

Deep dynamical model:

- Model for high-dimensional pixel data
- Simultaneous training is crucial
- Application: Control based on pixel data only

The future

The best predictive performance is obtained from **highly flexible models** (especially when large datasets are used). There are basically two ways of achieving flexibility:

1. Using models with a large number of parameters compared to the data set (e.g. deep NN).
2. Models using non-parametric components, e.g. Gaussian processes.

The future

The best predictive performance is obtained from **highly flexible models** (especially when large datasets are used). There are basically two ways of achieving flexibility:

1. Using models with a large number of parameters compared to the data set (e.g. deep NN).
2. Models using non-parametric components, e.g. Gaussian processes.

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

Deep reinforcement learning workshop at NIPS in december.

Niklas Wahlström
niklas.wahlstrom@liu.se

www.liu.se

LINKÖPING UNIVERSITY