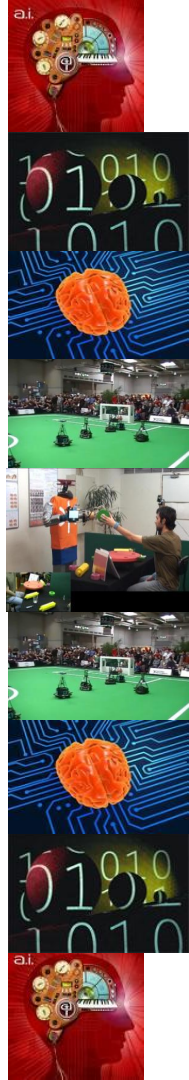
Hopfield Neural Network

Introduction- Recurrent NN

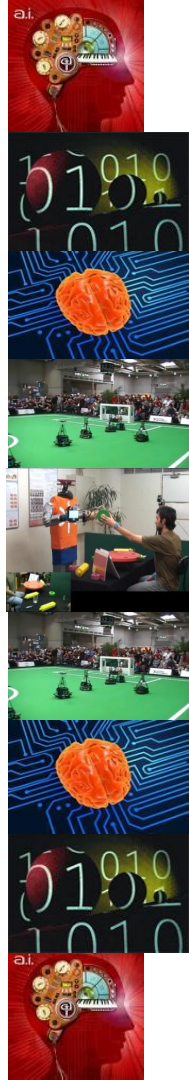
- A **RNN** is a type of artificial neural network where the connections between its nodes create a directed graph that follows a sequence over time.
- This enables the network to display dynamic changes in behavior as time progresses.
- RNNs evolved from feedforward neural networks and have the ability to use their internal memory to handle inputs of varying lengths.



Introduction- Recurrent NN

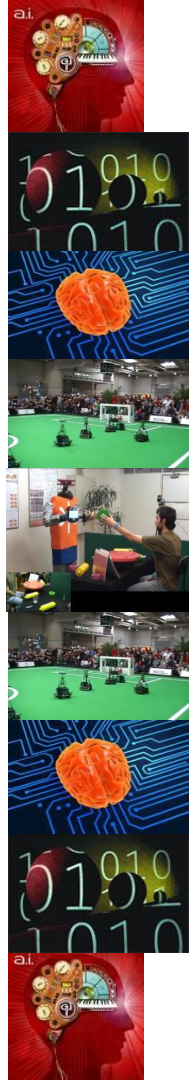
- RNNs are distinguished by their “memory” as they take information from prior inputs to influence the current input and output.

- Unlike traditional deep neural networks that treat inputs and outputs as separate entities, the output of an RNN is influenced by previous elements in the sequence.



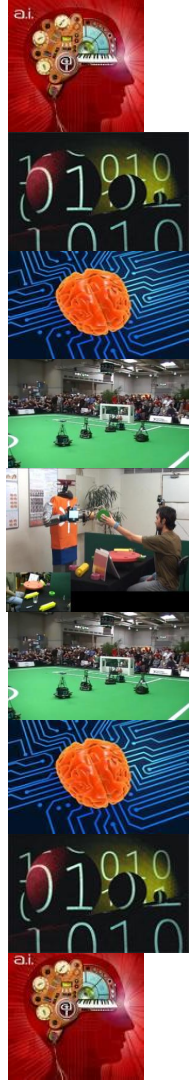
Introduction

- Performs pattern association
- Associative memory neural networks
- These are single layer neural networks
- Training data: input-output vector pair (Binary or Bipolar)
- Autoassociative memory
- Heteroassociative memory

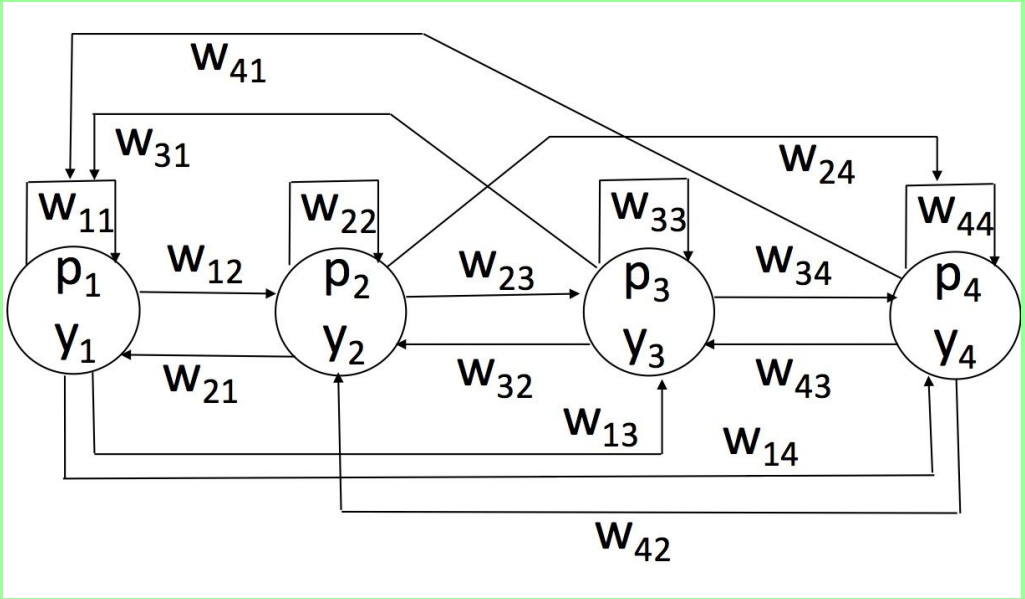


Introduction

- Can be feedforward or recurrent
- Examples:
  - Heteroassociative – feedforward
  - Autoassociative – feedforward
  - Bidirectional associative memory – recurrent
- Hopfield neural network is an example of a recurrent neural network.

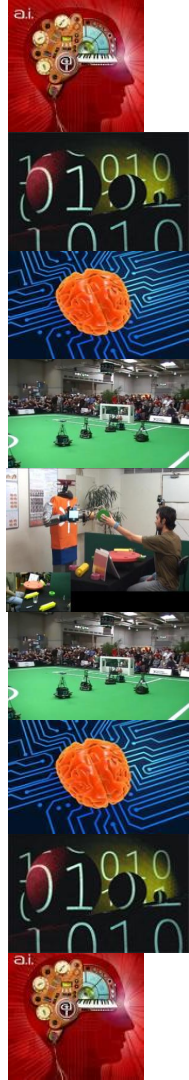


Hopfield Neural Network - Example



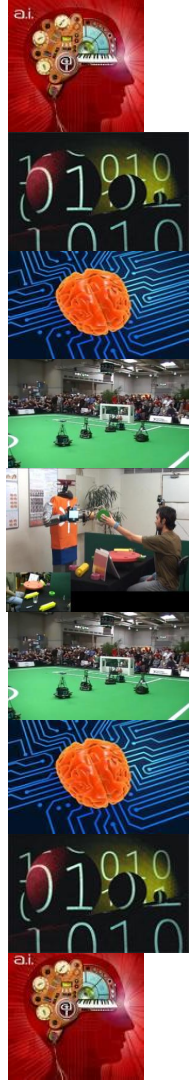
Hopfield Neural Network

- Is trained by determining a weight matrix
- An outer product of the input and output vectors for training
- Unsupervised and recurrent
- Trained neural network is used to repair input vectors with
  - Missing components
  - Incorrect components



Hopfield Neural Network

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{pmatrix}$$



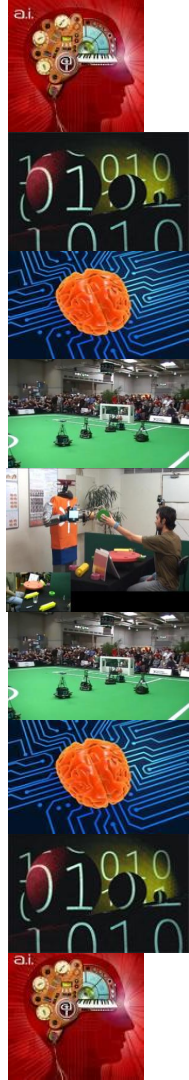
Hopfield Neural Network

If the training data consists of binary values each weight is calculated using

$$w_{ij} = \sum_e [2p_i - 1][2p_j - 1] \quad i = 1, ..., n \quad j = 1, ..., n$$

If the training data consists of bipolar values each weight is calculated using

$$w_{ij} = \sum_e p_i p_j \quad i = 1, ..., n \quad j = 1, ..., n$$



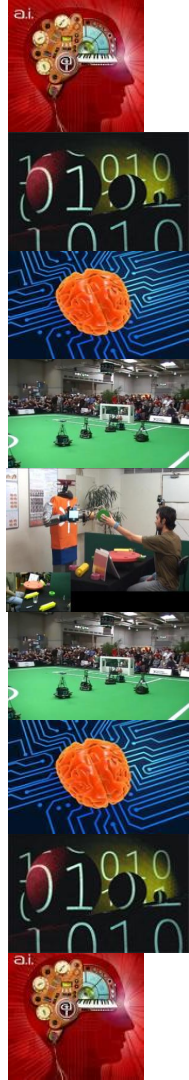
Hopfield Neural Network- Algorithm

Algorithm 6 Hopfield Application Algorithm

1: Set the output vector $\mathbf{y}$ to be equivalent to the input vector $\mathbf{p}$:
2: **for** $i \leftarrow 1, n$ **do**
3: $y_i = p_i$
4: **end for**
5: **while** the algorithm has not converged **do**
6: **while** all components y_i are not updated.
7: Randomly select a component y_i **do**

$$s_i = p_i + \sum_{j=1}^{n} y_j w_{ji}$$

$$y_i = 1 \text{ if } s_i > \theta_i$$
8: $$y_i = y_i \text{ if } s_i = \theta_i$$
9: $$y_i = 0 \text{ if } s_i < \theta_i$$
10:
11: Update the input vector $\mathbf{p}$ according to the changes to $\mathbf{y}$
12: **end while**
13: **end while**

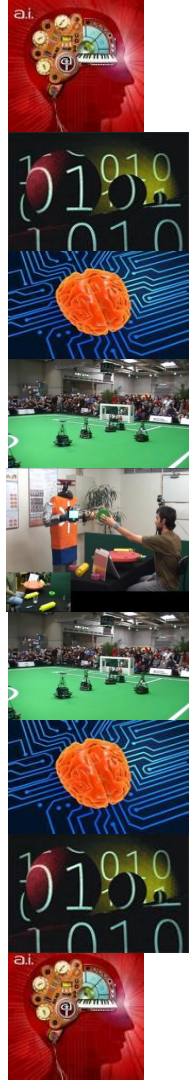


Hopfield Neural Network-Example

Suppose that you are required to train a Hopfield neural network to store the pattern [1 1 1 0]

$$W = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{pmatrix}$$

$$w_{ij} = \sum_{e} [2p_i - 1][2p_j - 1] \quad i = 1, ..., n \quad j = 1, ..., n$$



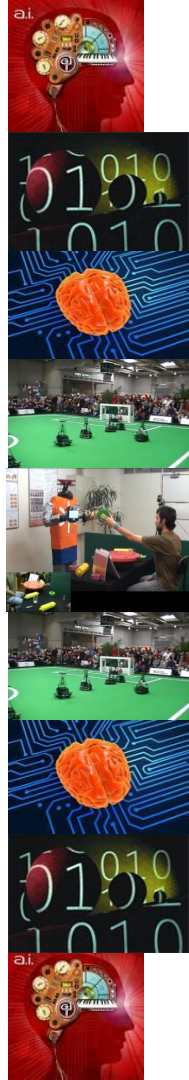
Hopfield Neural Network -Example

Apply the application algorithm to the input [0 0 1 0]

p = [0 0 1 0], y = [0 0 1 0] randomly choose y_1

$$s_1 = 0 + \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ -1 \end{pmatrix} = 1$$

$s_1 > \theta_1$, therefore $y_1 = 1$ and y = [1 0 1 0], p=[1 0 1 0]

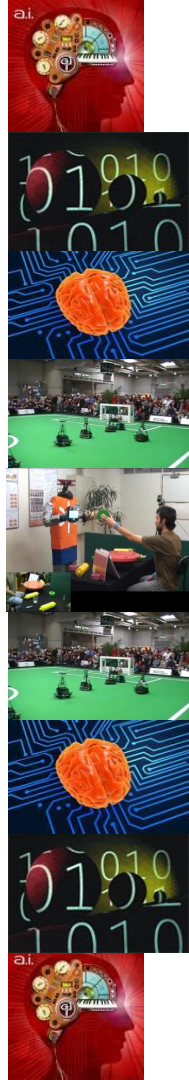


Hopfield Neural Network

Randomly choose y_4 as the component to update:

$$s_4 = 0 + \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \\ -1 \\ 0 \end{pmatrix} = -2$$

$s_4 < \theta_4$, therefore $y_4 = 0$ and y = [1 0 1 0], p=[1 0 1 0]

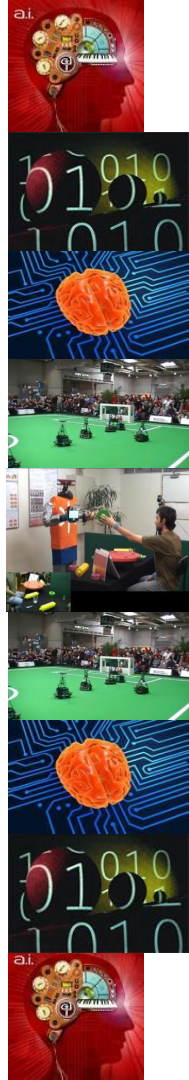


Hopfield Neural Network

Randomly choose y_2 as the component to update:

$$s_2 = 0 + \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ -1 \end{pmatrix} = 2$$

$s_2 > \theta_2$, therefore $y_2 = 1$ and y = [1 1 1 0], p=[1 1 1 0]

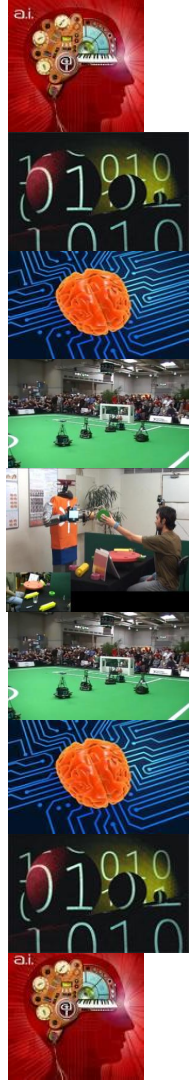


Hopfield Neural Network

Randomly choose y_3 as the component to update:

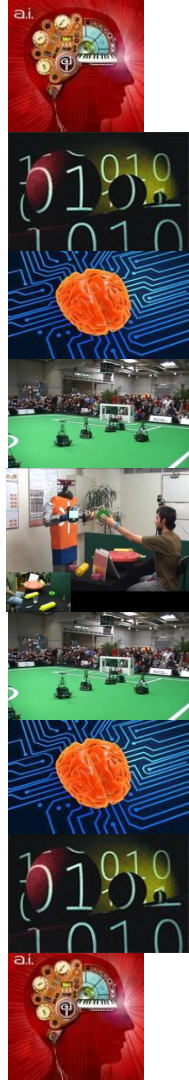
$$s_3 = 1 + \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ -1 \end{pmatrix} = 3$$

$s_3 > \theta_3$, therefore $y_3 = 1$ and y = [1 1 1 0], p=[1 1 1 0]



Hopfield Neural Network Applications

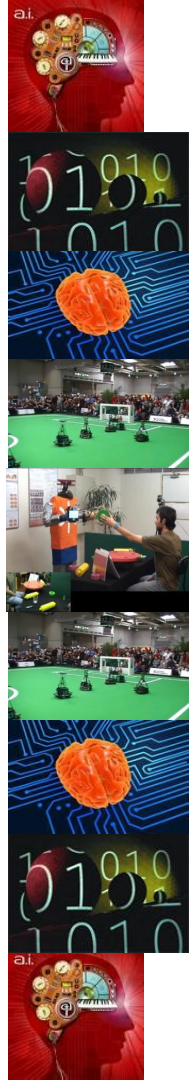
- Medical Imaging.
- Image restoration (continuous mode).
- Image segmentation(binary mode).
- Boundary detection.



Deep Learning NN

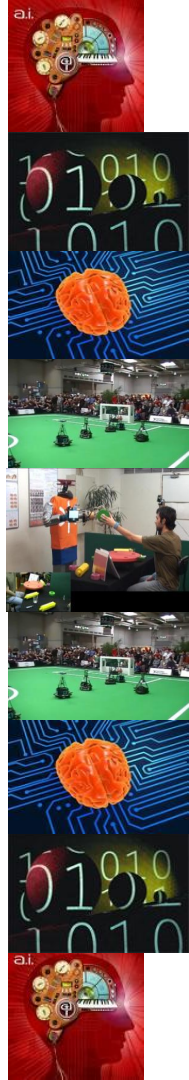
Deep Learning

- Multiple layers between the input and output layers.
- Multilayer vs. deep neural networks
- Can be feedforward or recurrent
- Learning process - incremental.
- Libraries.



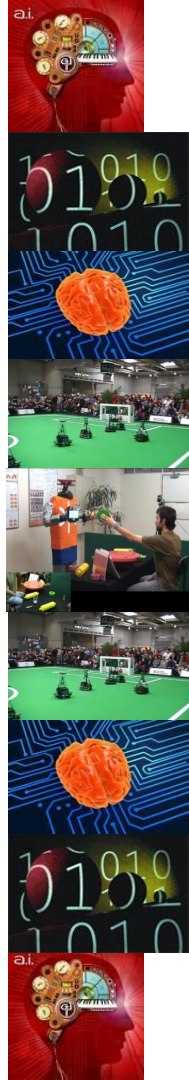
Deep Learning

- Assessing deep learning performance
  - Accuracy
  - Loss
- Applications of deep neural networks
  - Image processing
  - Speech recognition



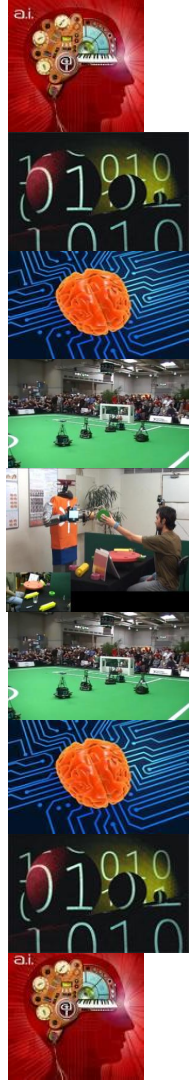
Learning Process

- Incremental learning
- Weights are calculated in a feedforward manner to the output layer
- A loss function is used to calculate the error in the activation
- An optimizer is used to update the weights to reduce the loss



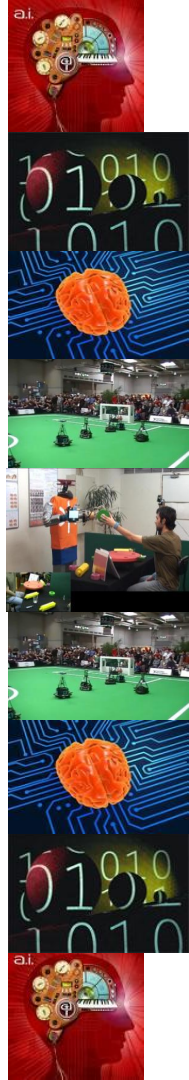
Example Loss Function

- Regression
  - Mean squared error
  - Mean squared logarithmic error
  - Mean absolute error
- Binary classification
  - Binary cross-entropy
  - Hinge loss
  - Squared hinge loss



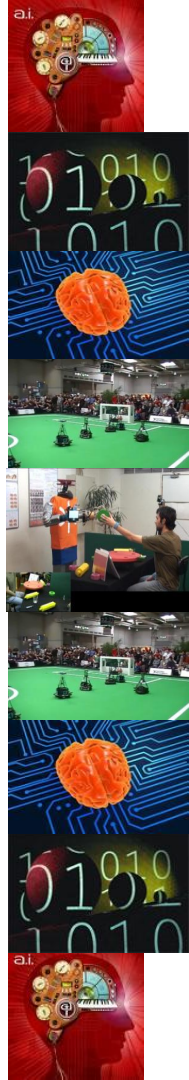
Example Loss Function

- Multiclass classification
  - Multiclass cross-entropy
  - Sparse multiclass cross-entropy
  - Kullback Leibler divergence



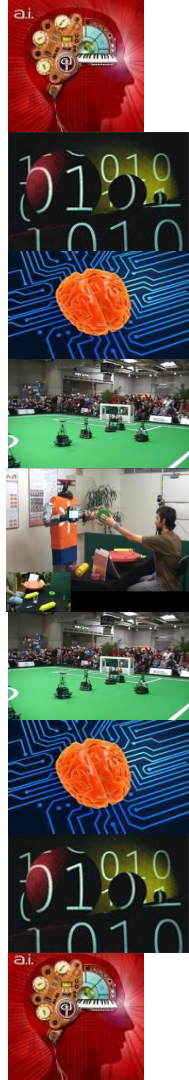
Optimizers

- Backpropagation
- Gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent
- Momentum



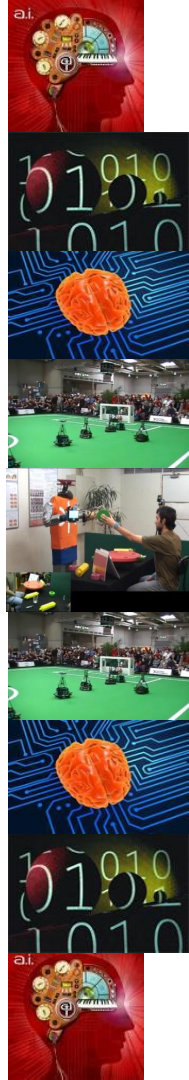
Optimizer

- Adagrad optimizer
- AdaDelta optimizer
- Adam
- RMSProp
- Nesterov accelerated gradient
- Learning rate



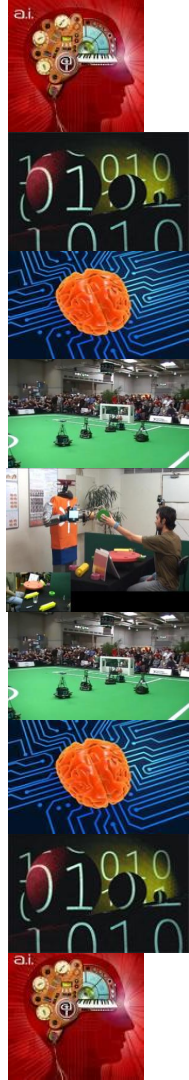
Activation Function

- Sigmoid
- Softmax
- Tanh - Hyperbolic Tangent
- ReLU - Rectified Linear Unit
- Leaky ReLU
- Swish



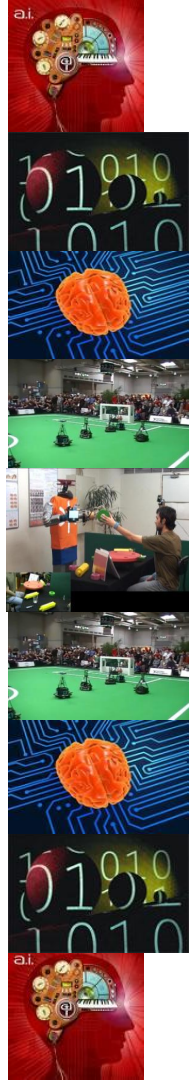
Types of Deep Learning

- Convolutional Neural Networks
- Autoencoders
- Restricted Boltzman Machines
- Deep Belief Networks
- Long Short Term Memory (LSTM)



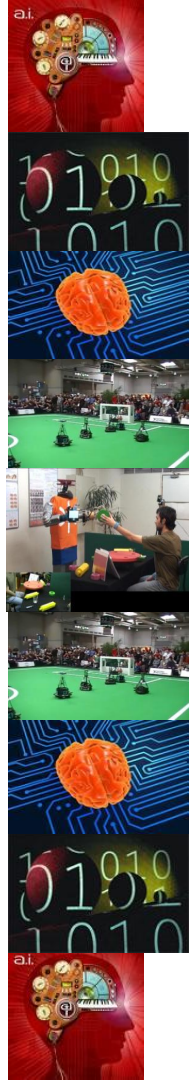
Overfitting

- Deep neural networks are susceptible to overfitting
- Performs well on the training set
- Does not perform well on the test set



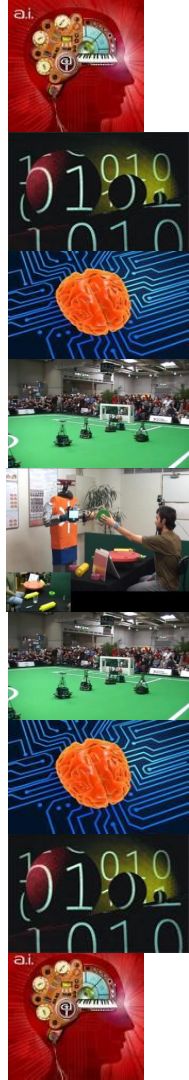
Overcoming overfitting

- Reducing the size of the network
- Regularization
- Early stopping
- Adding dropout



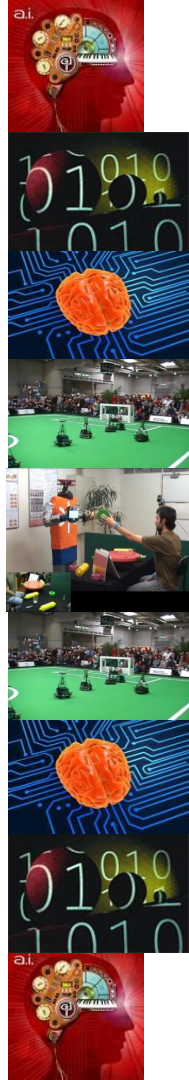
Transfer Learning

- Applied in similar domains
- Applied where there is limited data
- What to transfer ?
- When to transfer?
- How to transfer ?



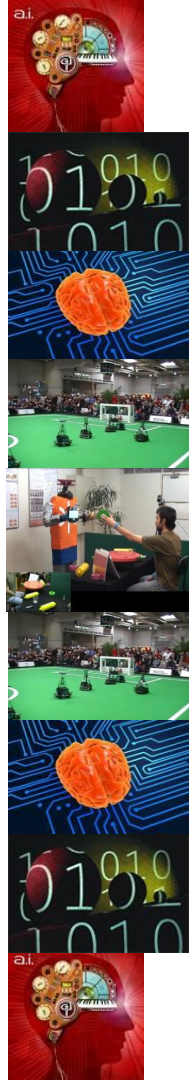
Transfer Learning and Pre-trained Model

- Data needed to generalize well
- Insufficient data
- Transfer learning – features and weights
- Retraining the last layer
- Selecting layers to retrain
- Pre-trained models and ImageNet



Pre-trained Models for Computer Vision

- VGG 16
- VGG 19
- Inception V3
- Xception
- ResNet 50



Pre-trained Models for NLP

- Word2Vec
- GloVe
- FastText