# Deep Learning for NLP

TECHNISCHE UNIVERSITÄT DARMSTADT

**Nils Reimers**

## Course-Website: www.deeplearning4nlp.com

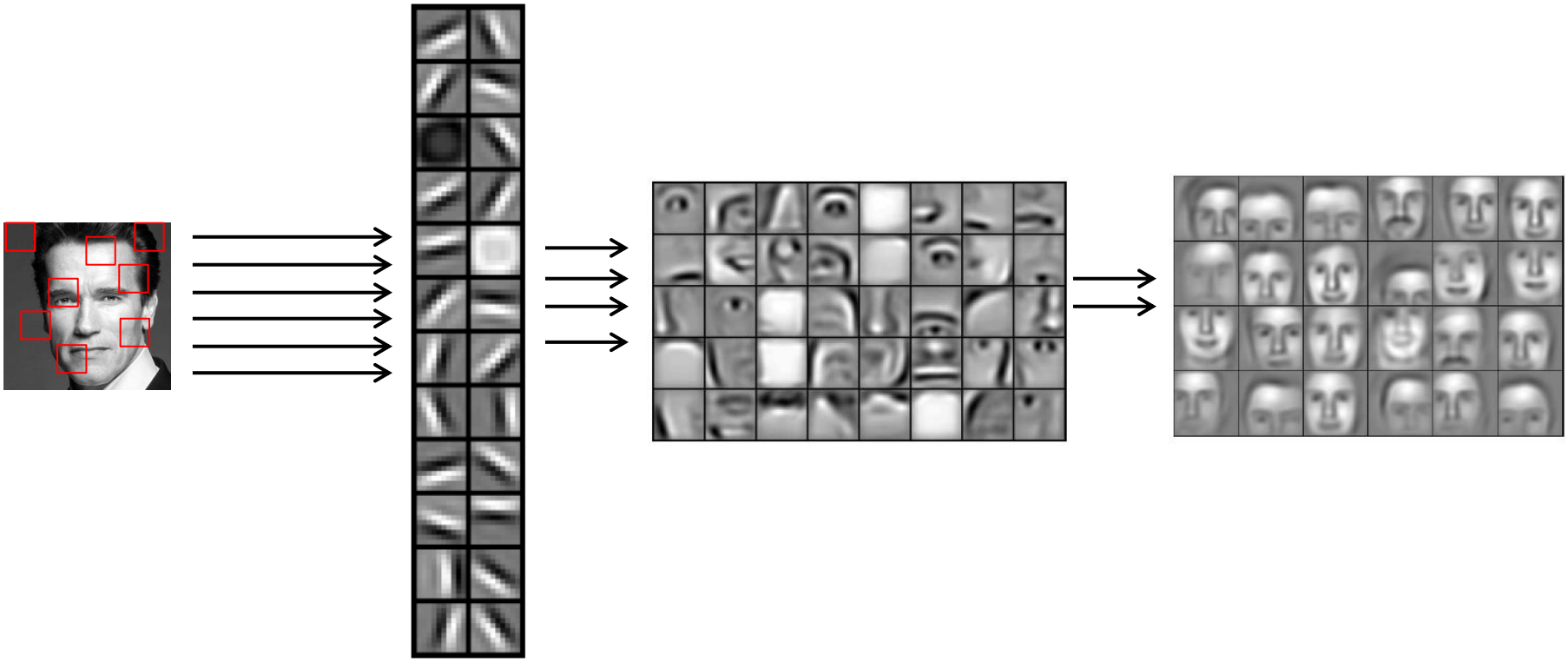
# Last Week Exercise

TECHNISCHE UNIVERSITÄT DARMSTADT

- Adapt the Keras NER implementation and include capitalization information

# What does the intermediate Layers learn?

- Lee et al. (ICML 2009) learned automatically different feature detectors



Input
e.g. 40 000 dim

Layer 1
9 600 dim

Layer 2
3800 dim

Layer 3
900 dim

# Warm-Up Question

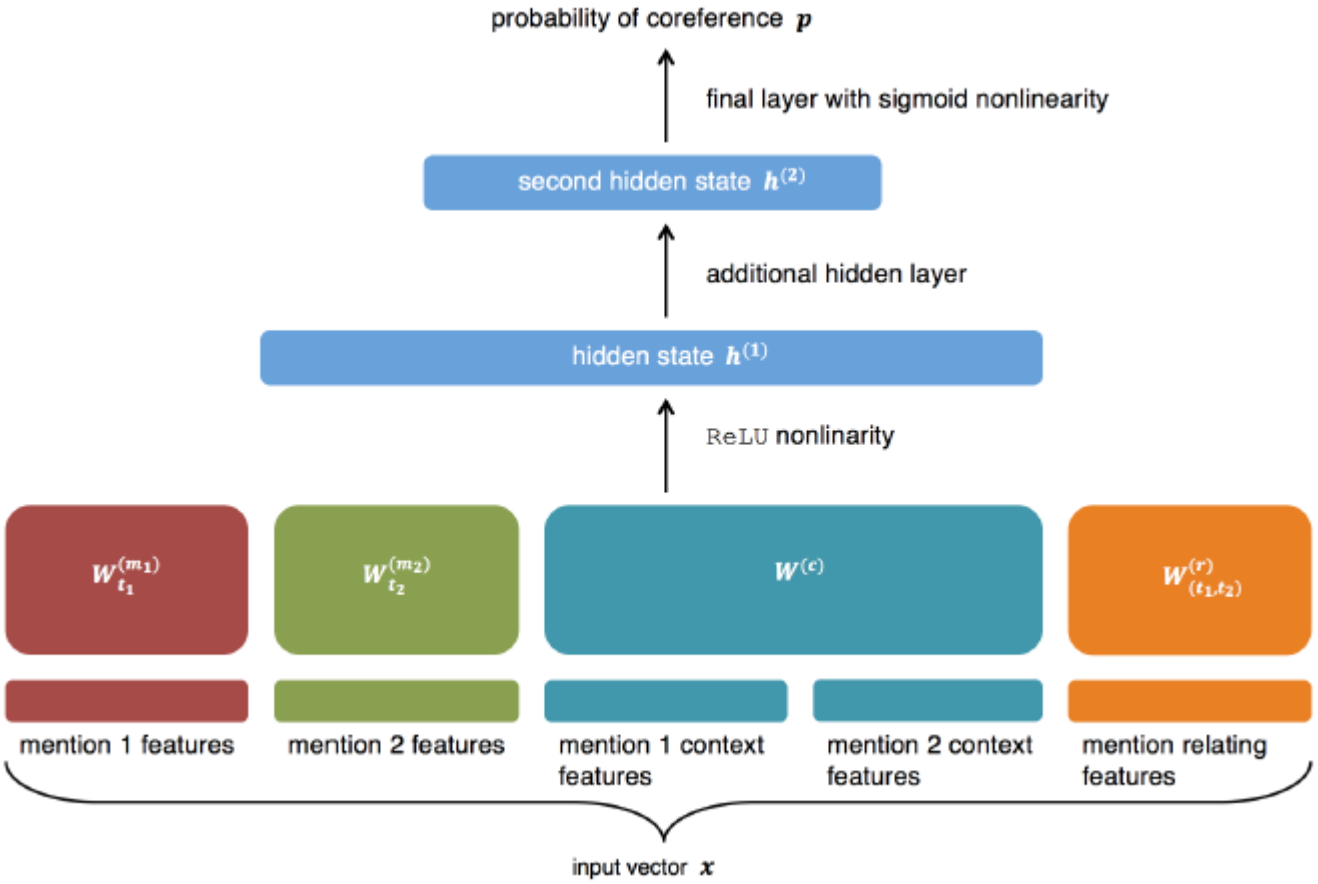
TECHNISCHE UNIVERSITÄT DARMSTADT

- How to use a feed forward network for coreference resolution?
- Hint: We can model this as a binary classification task: does two mention refer to the same entity?

# Solution

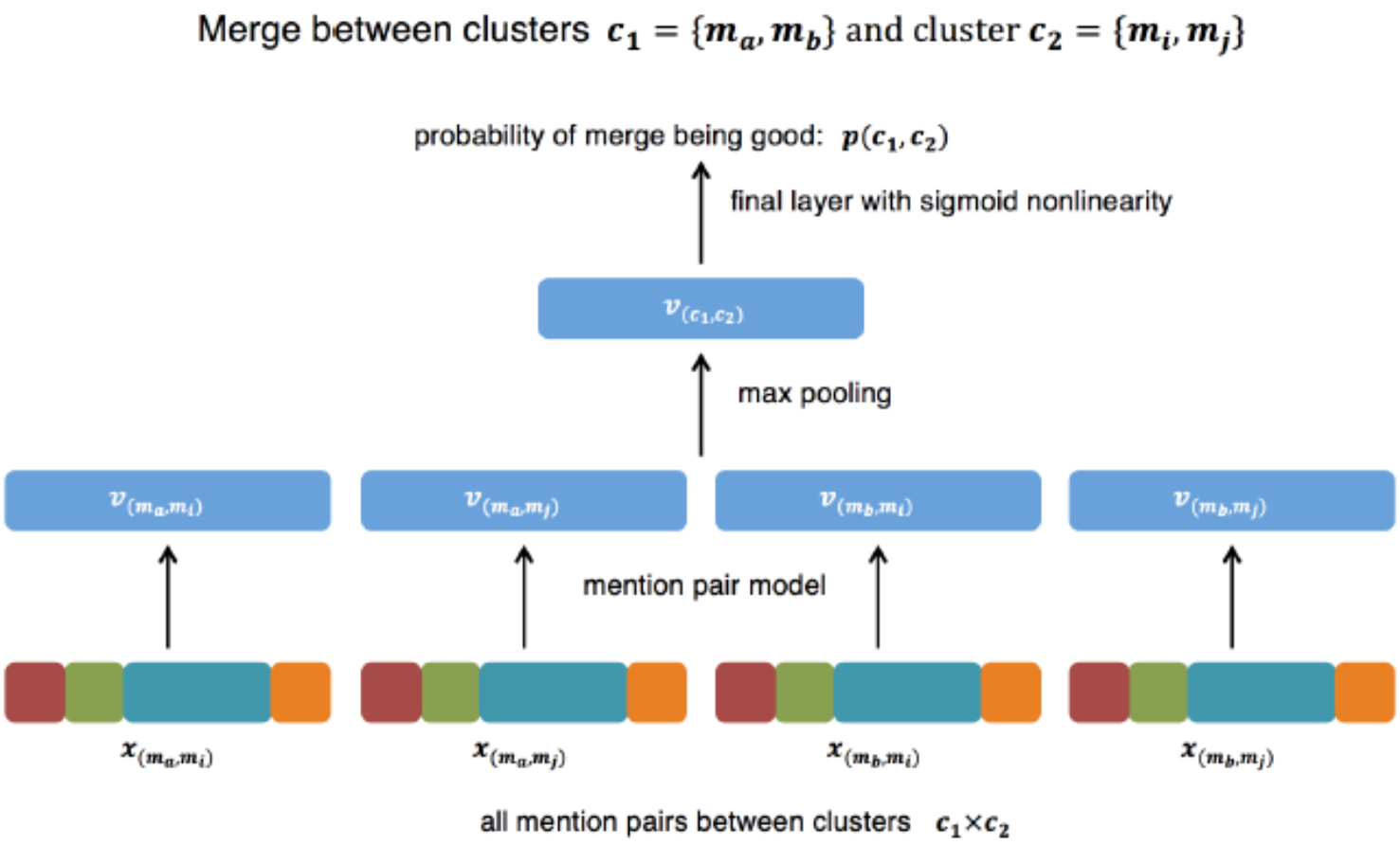
TECHNISCHE UNIVERSITÄT DARMSTADT

- Kevin Clark, 2014, *Neural Coreference Resolution*



# Solution II

- Kevin Clark, 2014, *Neural Coreference Resolution*



# Recommended Readings

TECHNISCHE UNIVERSITÄT DARMSTADT

- Autoencoders:
  - http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/
  - http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders
  - http://ufldl.stanford.edu/wiki/index.php/Fine-tuning_Stacked_AEs
- Recursive Neural Networks:
  - Socher et al., 2011, Semi-supervised recurisve autoencoders for predicting sentiment distributions http://dl.acm.org/citation.cfm?id=2145450
  - Socher et al., 2013, Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank http://nlp.stanford.edu/%7Esocherr/EMNLP2013_RNTN.pdf
- Good video from Hinton on autoencoders and pretraining
  - https://www.youtube.com/watch?v=vShMxxqtDDs

# Autoencoders

TECHNISCHE UNIVERSITÄT DARMSTADT

## Autoencoders

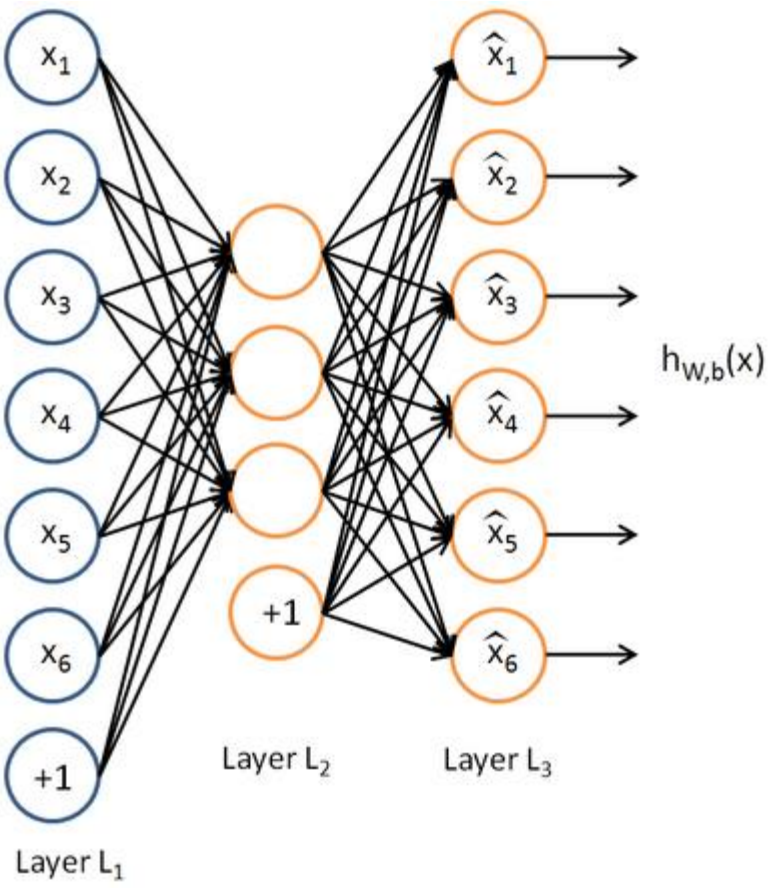
- Unsupervised Learning Algorithm
- Given an input *x*, we learn a *compressed* representation of the input, which we then try to reconstruct
- In the simpliest form: Feed forward network with hidden size < input size.
- We then search for parameters such that:

$$\hat{x} \approx x$$

  for all training examples
- The error function is:

$$E(x, W, b) = ||\hat{x} - x||_2$$

- Once we finished training, we are interested in the compressed representation, i.e. the values of the hidden units




Source:
http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity

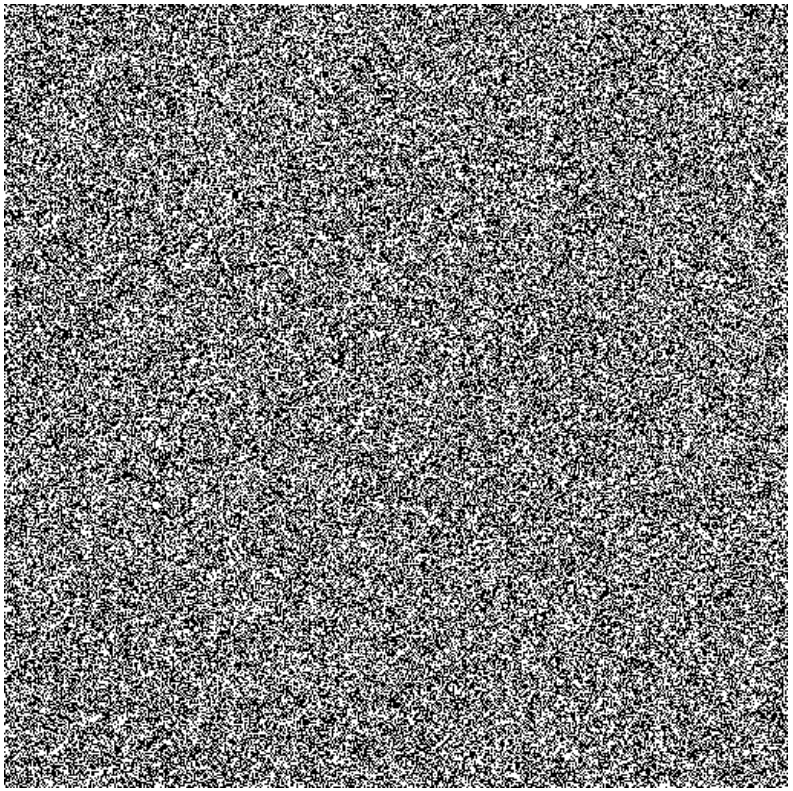


UKP

# Why would we use autoencoders?

TECHNISCHE UNIVERSITÄT DARMSTADT

- How does a randomly generated image look like?



# Why would we use autoencoders?

TECHNISCHE UNIVERSITÄT DARMSTADT

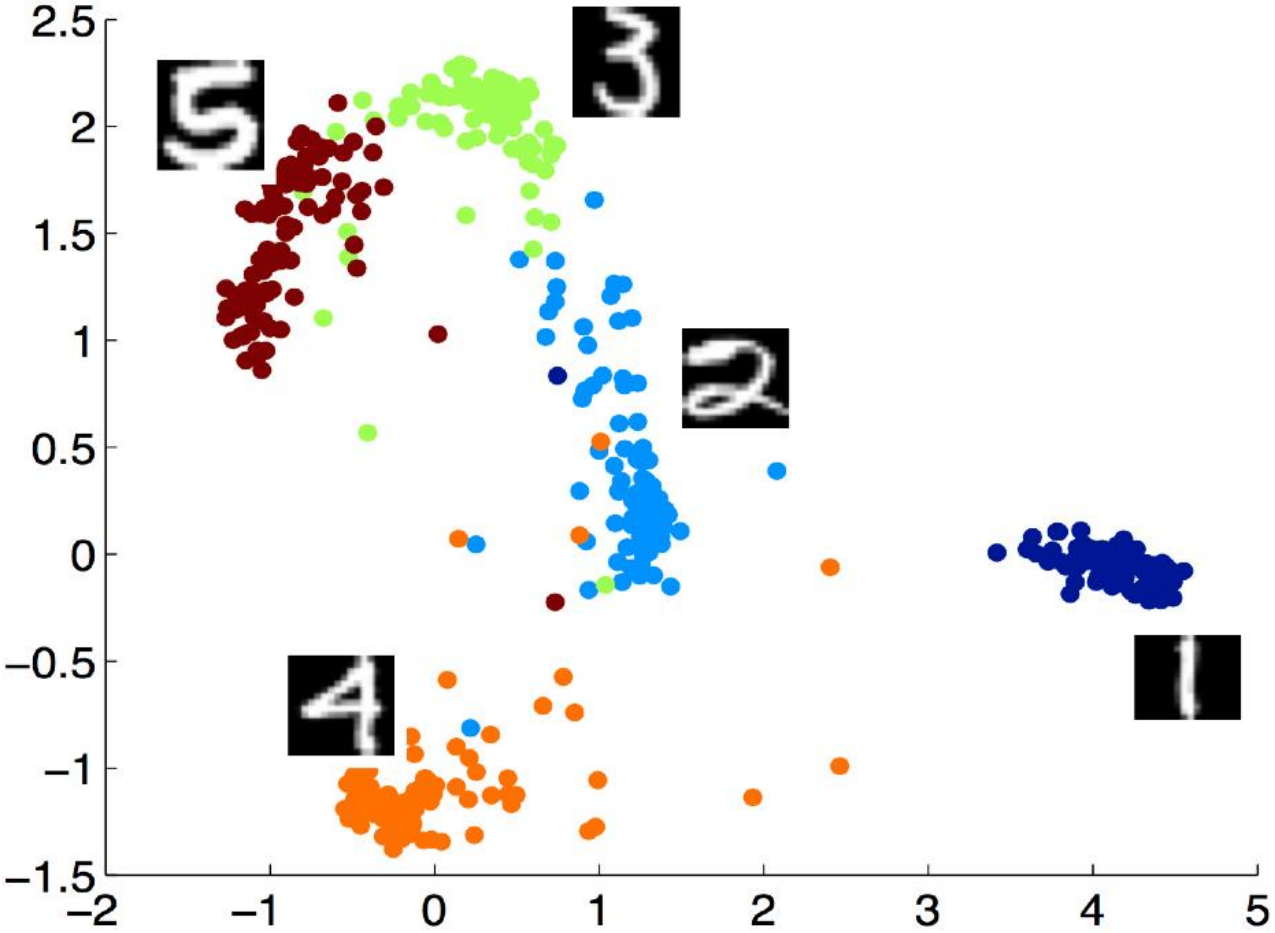
- What would be the probability to get an image like this from random sampling?

# Why would we use autoencoders?

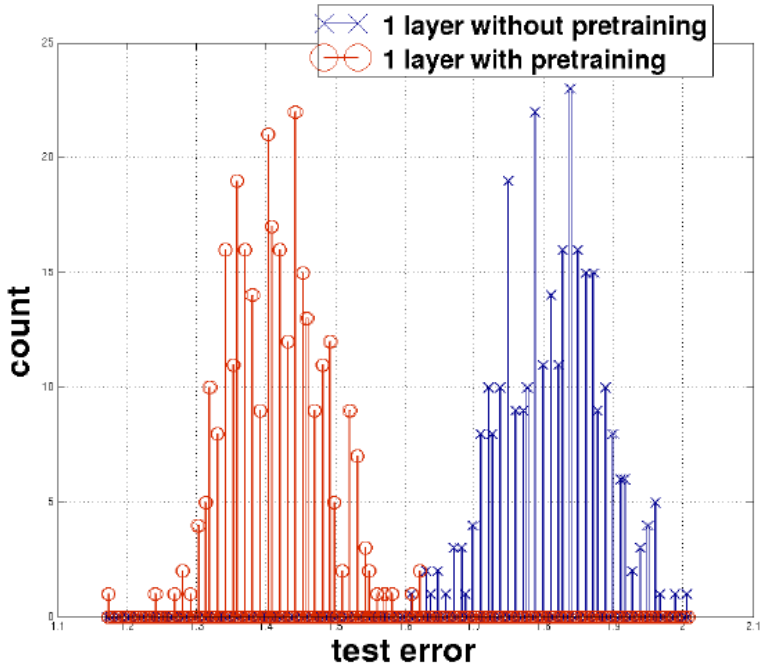
TECHNISCHE UNIVERSITÄT DARMSTADT

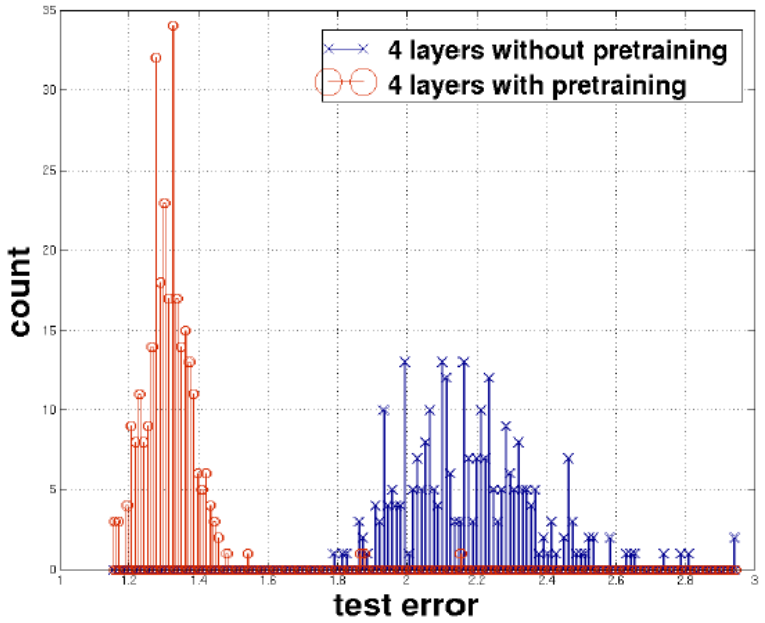
- Produce a compressed representation of a high-dimensional input (for example images)
- The compression is lossy. Learning drives the encoder to be a good compression in particular for training examples
- For random input, the reconstruction error will be high
- The autoencoder learns to abstract properties from the input. What defines a natural image? Color gradients, straight lines, edges etc.
- The abstract representation of the input can make a further classification task much easier

# Dimension-Reduction can simplify classifcation tasks – MNIST Task



# Dimension-Reduction can simplify classifcation tasks – MNIST Task




- Histogram-plot of test error on the MNIST hand written digit recognition.
- Comparison of neural network with and without pretraining

Source: Erhan et al, 2010, Why Does Unsupervised Pre-training Help Deep Learning?

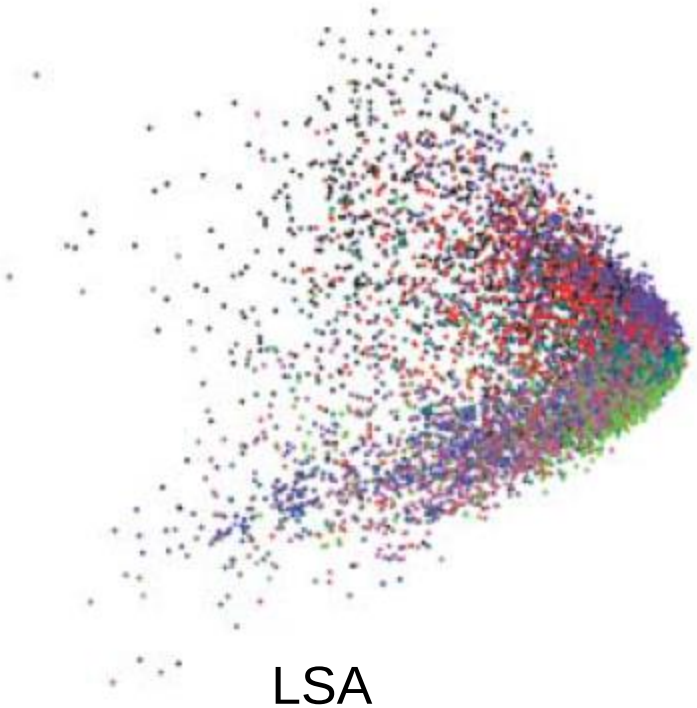
# Autoencoders vs. PCA

- Principle component analysis (PCA) converts a set of correlated variables to a set of linearly uncorrelated variables called *principle components*
- PCA is a standard method to break down high-dimensional vector spaces, e.g. for information extraction or visualization
- However, PCA can only capture linear correlations

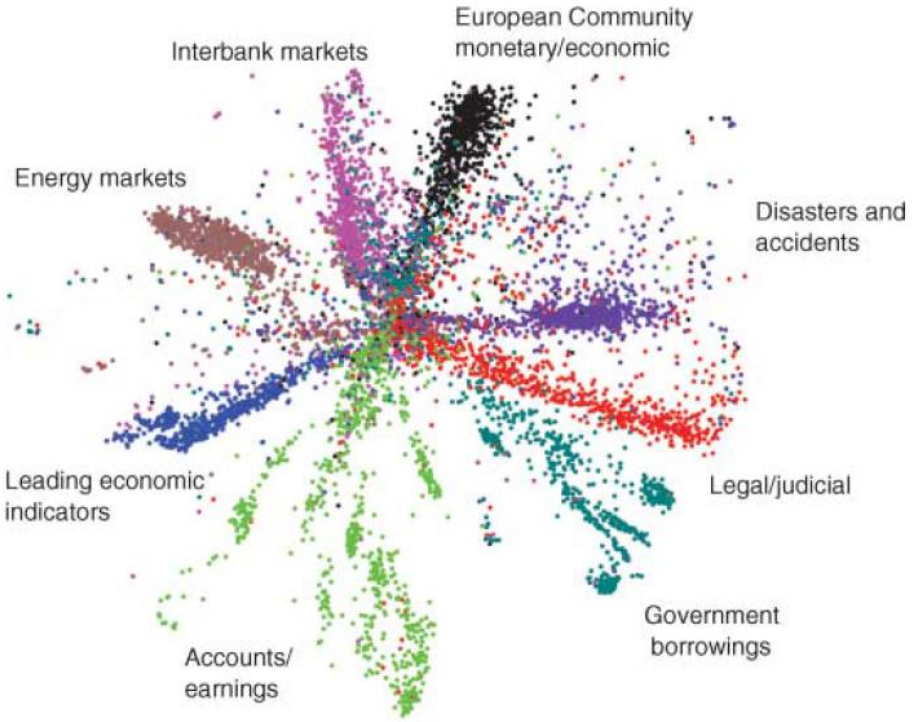
# Autoencoders vs. PCA - Example

TECHNISCHE UNIVERSITÄT DARMSTADT

- Articles from Reuter corpus were mapped to a 2000 dimensional vector, using the 2000 most common word stems



LSA




Deep Autoencoder

Source: Hinton et al., Reducing the Dimensionality of Data with Neural Networks



UKP

# How to ensure the encodes does *not* learn the identity function?

TECHNISCHE UNIVERSITÄT DARMSTADT

## Identify Function

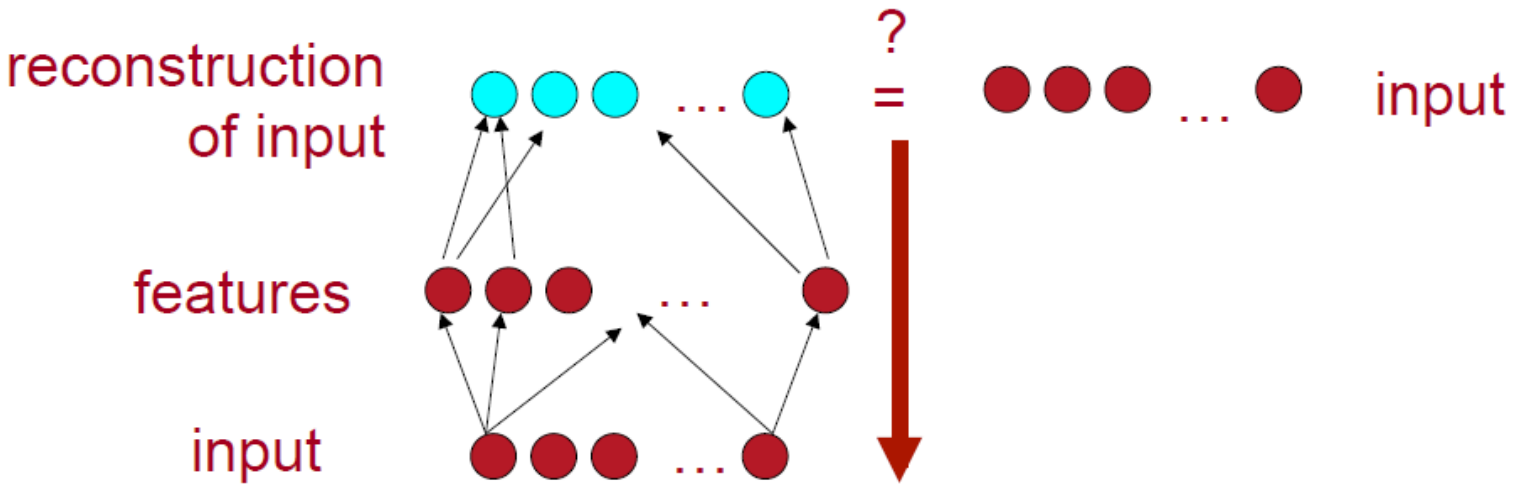
- Learning the identity function would not be helpful
- Different approaches to ensure this:
  - Bottleneck constraint: The hidden layer is (much) smaller than the input layer
  - Sparse coding: Forcing many hidden units to be zero or near zero
  - Denoising encoder: Add randomness to the input and/or the hidden values

## Denoising Encoder

- Create some random noise $\varepsilon$
- Compute $\hat{x} = f(x + \varepsilon)$
- Reconstruction Error: $\hat{x} \approx x?$
- Alternatively: Set some of the neurons (e.g. 50%) to zero
- The noise forces the hidden layer to learn more robust features

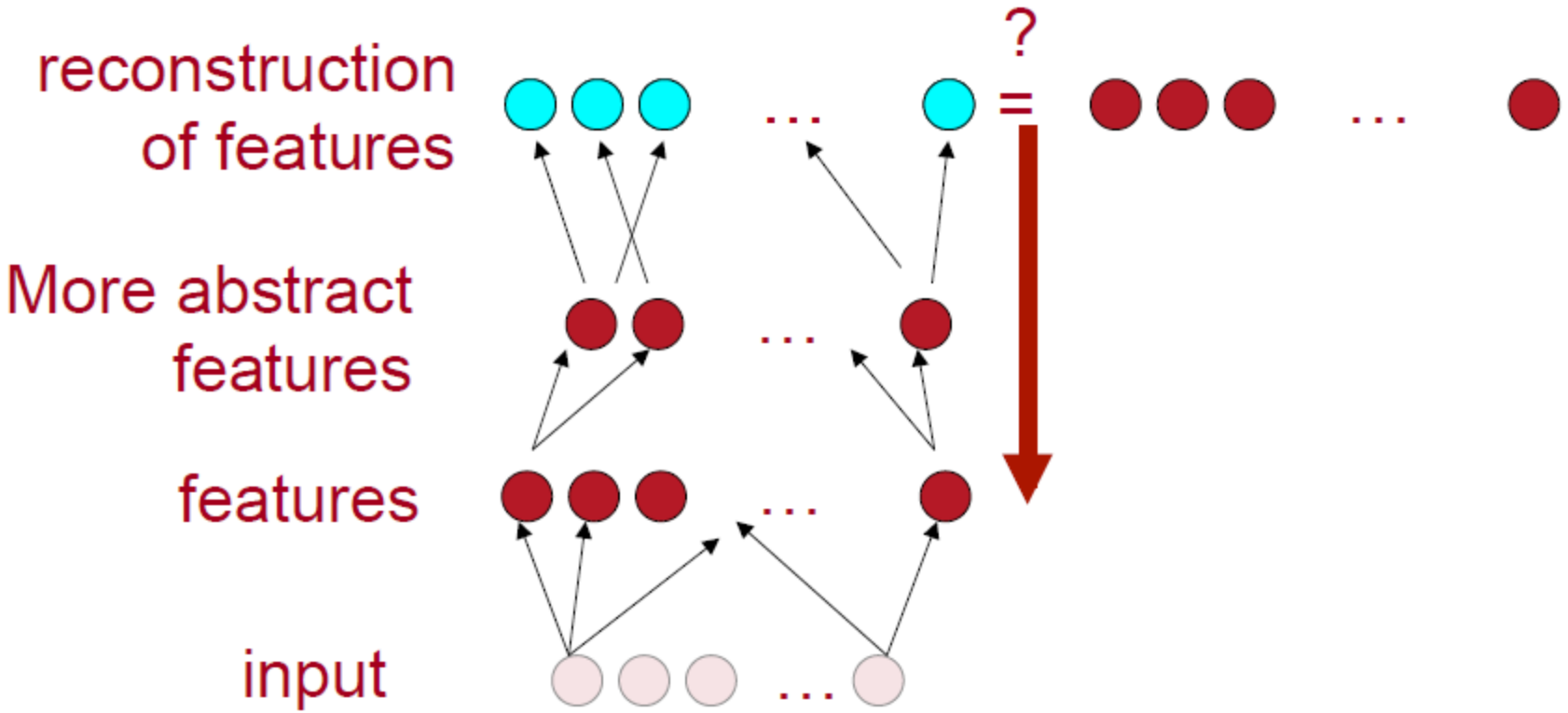
# Stacking Autoencoders

- We can stack multiple hidden layers to create a deep autoencoder
- These are especially suitable for highly non-linear tasks
- The layers are trained layer-wise – one at a time




Step 1: Train single layer autoencoder until convergence

# Stacking Autoencoders




Step 2: Add additional hidden layer and train this layer by trying to reconstruct the output of the previous hidden layer. Previous layers are will not be changed. Error function: $||\hat{h}_1 - h_1||_2$ .

# Stacking Autoencoders – Fine-tuning

TECHNISCHE UNIVERSITÄT DARMSTADT

- After pretraining all hidden layers, the deep autoencoder is fine-tuned

**Unsupervised Fine-Tuning:**

- Apply back propagation to the complete deep autoencoder
- Error-Function:

$$E(x, W^{(1)}, W^{(2)}, ...) = ||\hat{x} - x||_2$$

- Further details, see Hinton et al.
- *(It appears that supervised fine-tuning is more common nowadays)*

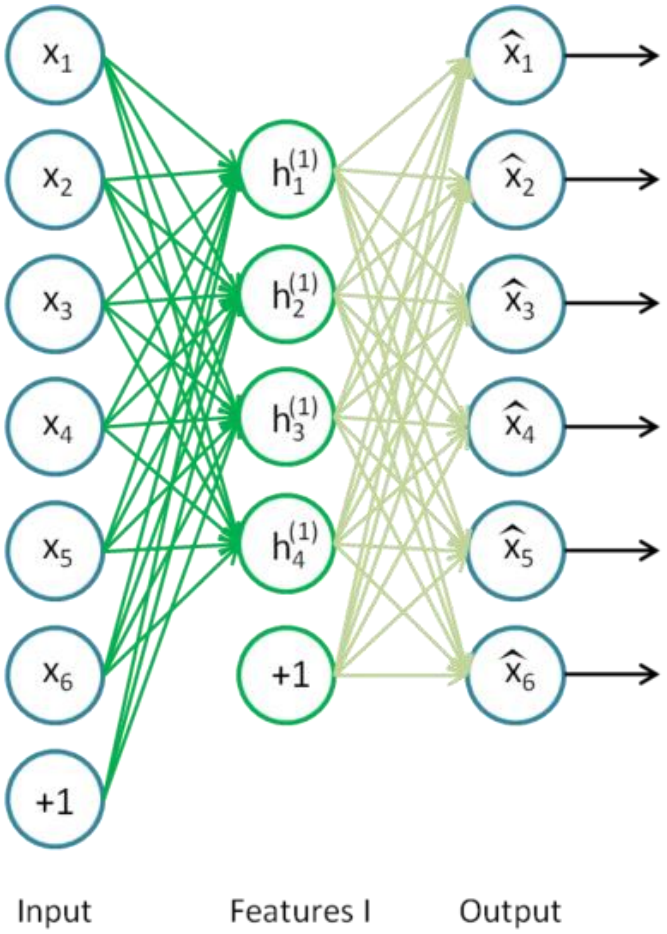
**Supervised Fine-Tuning:**

- Use your classification task to fine-tune your autoencoders
- A softmax-layer is added after the last hidden layer
- Weights are tuned by using back prograpagtion.
- See next slides for an example or http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

# Stacking Autoencoders - Example

**Pretrain first autoencoder**

- Train an autoencoder to get the first weight matrix $W^{(1)}$ and first bias vector $b^{(1)}$
- The second weight matrix, connecting the hidden and the output units, will be disregarded after the first pretraining step
- Stop after a certain number of iterations

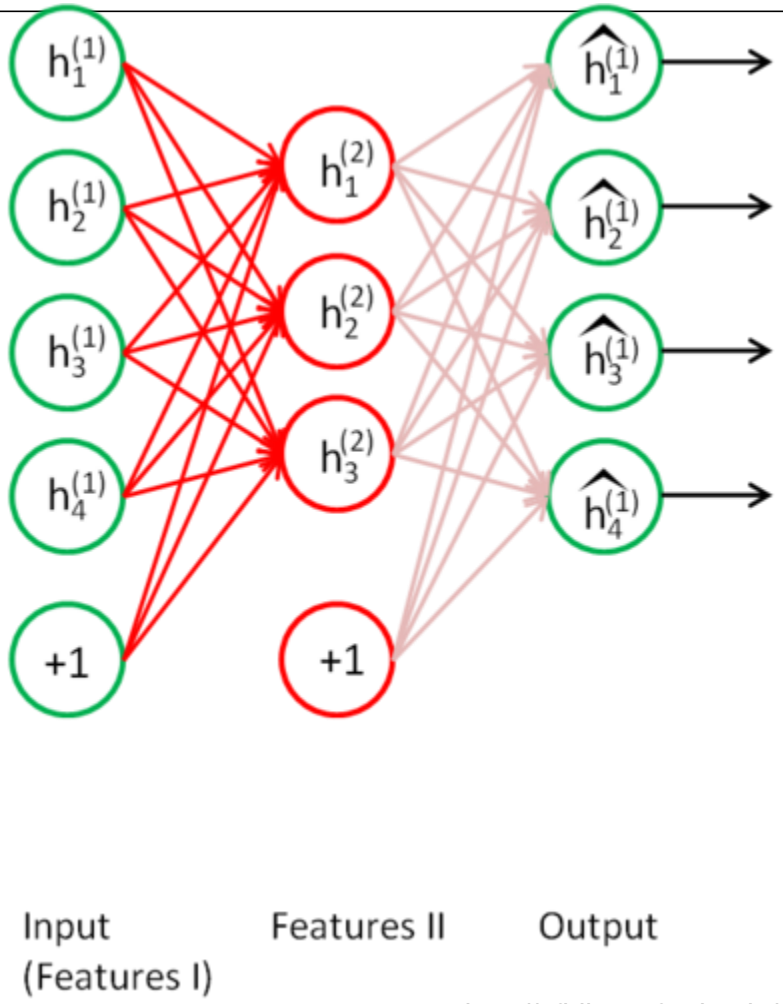



Source: http://ufldl.stanford. edu/wiki/

# Stacking Autoencoders - Example

**Pretrain second autoencoder**

- Use the values of the previous hidden units as input for the next autoencoder.
- Train as before

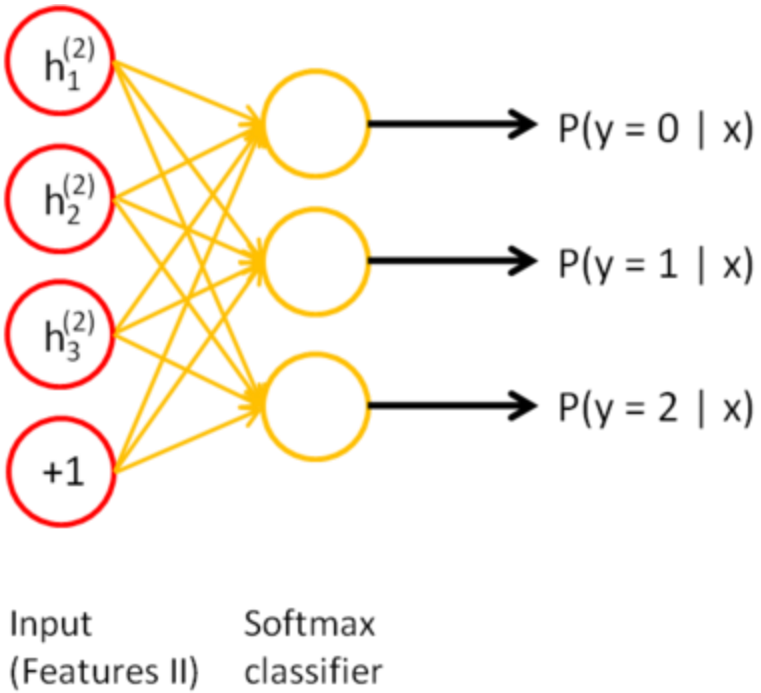



Source: http://ufldl.stanford. edu/wiki/

# Stacking Autoencoders - Example

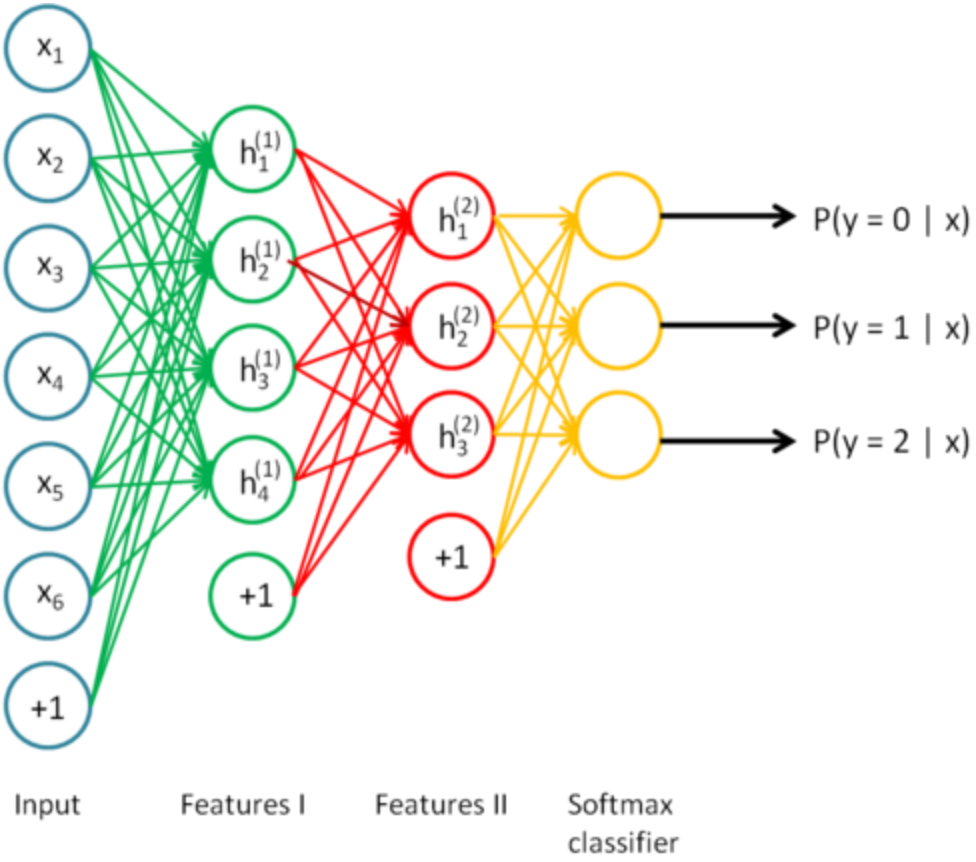
**Pretrain softmax layer**

- After second pretraining finishes, add a softmax layer for your classification task
- Pretrain this layer using back propagation




Source: http://ufldl.stanford. edu/wiki/

# Stacking Autoencoders - Example



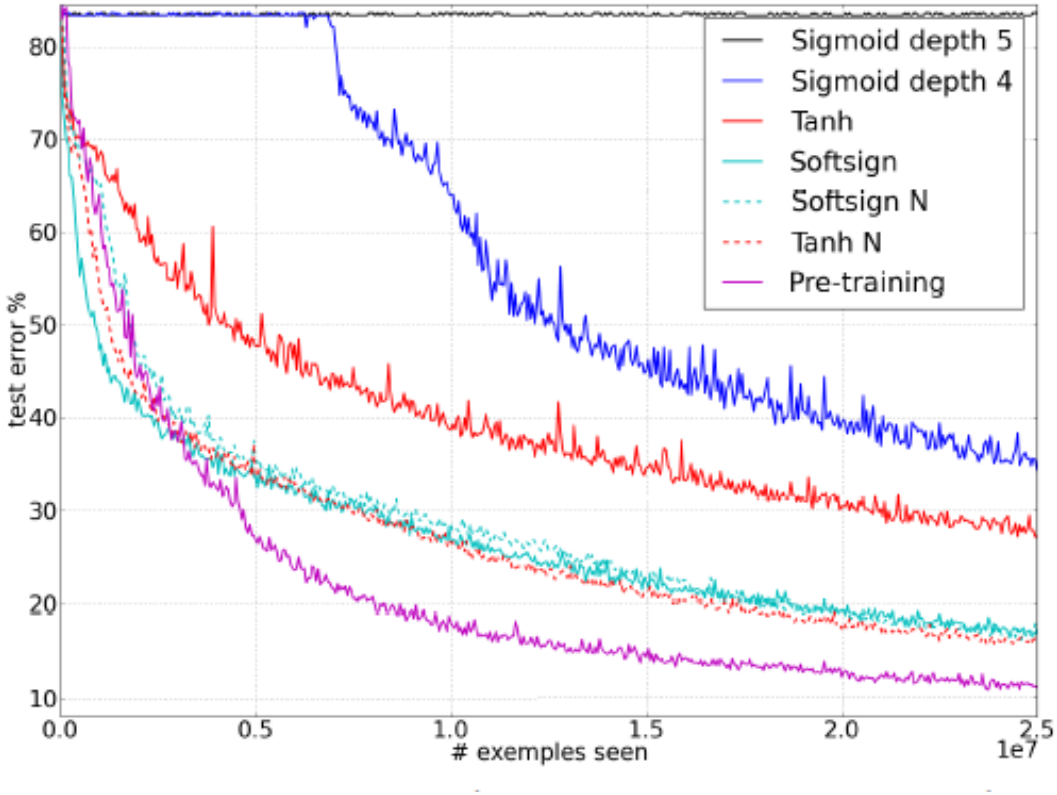

**Fine-tuning**

- Plug all layers together
- Compute the costs based on the actual input $x$
- Update **all** weights using backpropagation

Source: http://ufldl.stanford. edu/wiki/

# Is pre-training really necessary?

- Xavier Glorot and Yoshua Bengio, 2010, *Understanding the difficulty of training deep feedforward neural networks*
- With the right activation function and initialization, the importance of pre-training decreases



# Is pre-training really necessary?

TECHNISCHE UNIVERSITÄT DARMSTADT

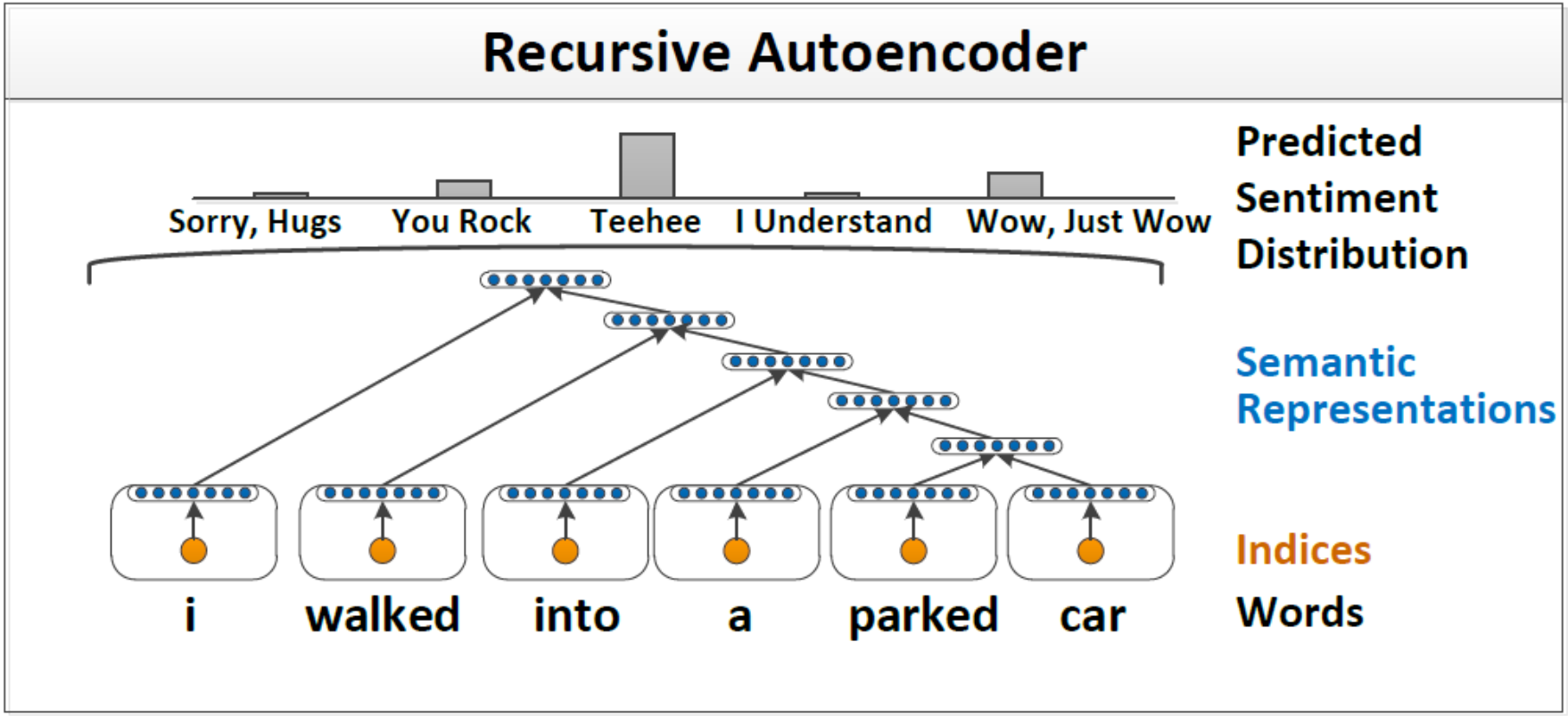
- Pre-training achieves two things:
  - It makes optimization easier
  - It reduces overfitting
- Pre-training is not required to make optimization work, if you have enough data
  - Mainly due to a better understanding how initialization works
- Pre-training is still very effective on small datasets
- More information: https://www.youtube.com/watch?v=vShMxxqtDDs

# Recursive Neural Networks

TECHNISCHE UNIVERSITÄT DARMSTADT

- Socher et al., 2011, *Semi-supervised recursive autoencoders for predicting sentiment distributions*
- Socher et al., 2013, *Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank*

# Recursive Autoencoders



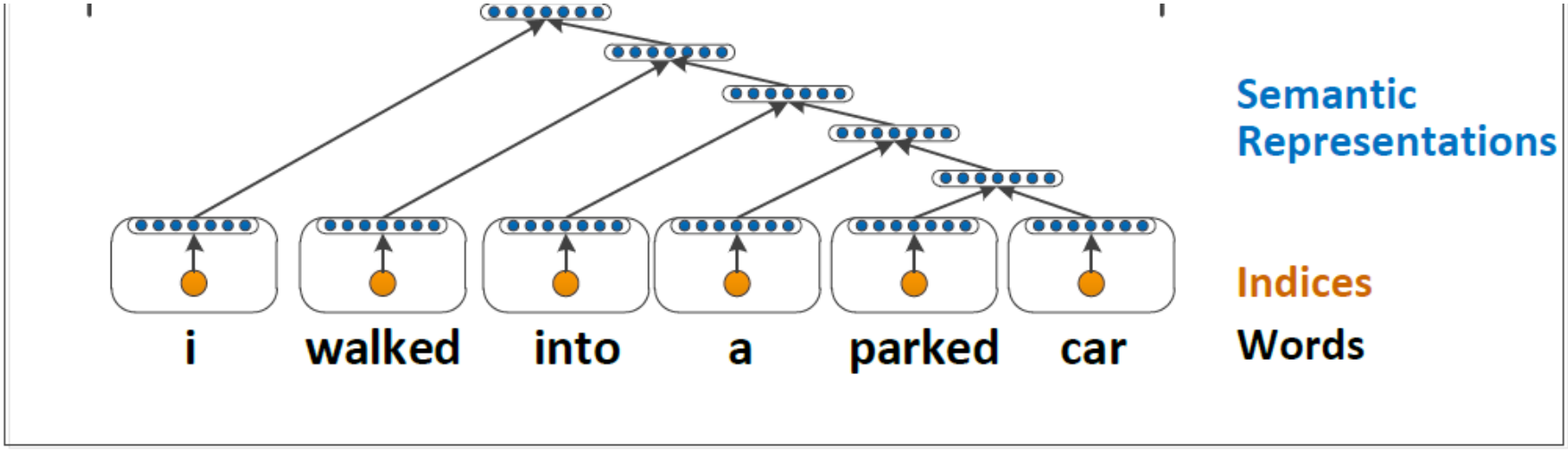

- In a first step, words are mapped to dense vectors (word embeddings)
- Iteratively they are combined and reduced to form a single compact representation of the sentence

# Recursive Autoencoders (RAE)

- Given two embeddings $x_1, x_2$ each with length *n*
- The autoencoder takes $[x_1; x_2]$ as input and maps it to a hidden layer of size *n:*

$$y_1 = f(W[x_1; x_2] + b)$$

- The function is repeatedly applied for the whole sentence until we receive a single vector of size *n*, representing the semantic of this sentence



# Selecting the nodes that should be combined

TECHNISCHE UNIVERSITÄT DARMSTADT

- The previous slides showed a joining of the vectors from right to left
- However, we can define any tree structure for the combination of two vectors, for example a parse tree
- Socher et al. present a greedy approach for the combination of vectors
  - Compute the reconstruction error for all neighboring vectors.
  - The two neighbors with the lowest error are selected and their nodes are replaced by the compressed representation.
  - Repeat the previous two steps until we end up with a single vector representing the semantics of the sentence
- A different, more recent approach, is to use parse trees

- The output of the recursive autoencoder can be used for a classification task by adding a final softmax layer:

$$o = softmax(W^{label} y_m)$$

# Dropout in Neural Networks

TECHNISCHE UNIVERSITÄT DARMSTADT

Inspired by Hinton
https://www.youtube.com/watch?v=vShMxxqtDDs

For details:
Srivastava, Hinton et al., 2014, *Dropout: A Simple Way to Prevent Neural Networks from Overtting*

# Ensemble Learning

TECHNISCHE UNIVERSITÄT DARMSTADT

- Create many different models and combine them at test time to make prediction
- Averaging over different models is very effective against overfitting
- Random Forest
  - A single decision trees is not very powerful
  - Creating hundreds of different trees and combine them
- Random forests works really well
  - Several Kaggle competitions, e.g. Netflix, were won by random forests

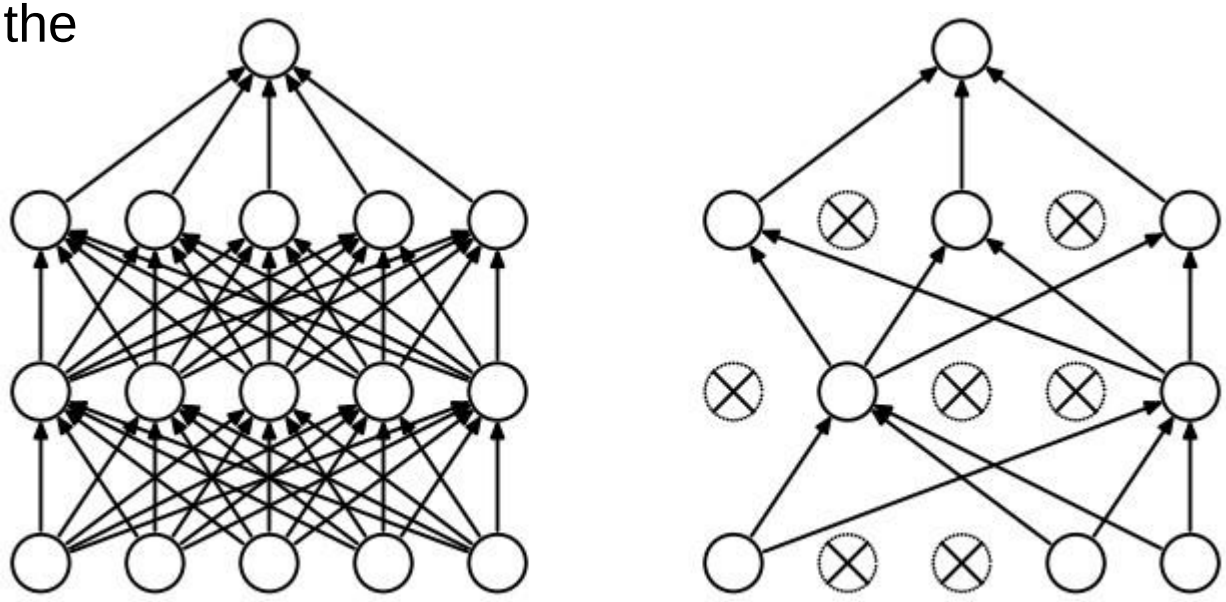
# Model Averaging with Neural Nets

TECHNISCHE UNIVERSITÄT DARMSTADT

- We would like to do massive model averaging
  - Average over 100, 1.000, 10.000 or 100.000 models
- Each net takes a long time to train
  - We don’t have enough time to learn so many models
- At test time, we don’t want to run lots of large neural nets
- We need something that is more efficient
  - Use dropouts!

# Dropout

- Each time present a training example, we dropout 50% of the hidden units
- With this, we randomly sample over $2^H$ different architectures
  - H: Number of hidden units
- All architectures share the same weights



(a) Standard Neural Net (b) After applying dropout.

Img source: http://cs231n.github.io/

# Dropout

TECHNISCHE UNIVERSITÄT DARMSTADT

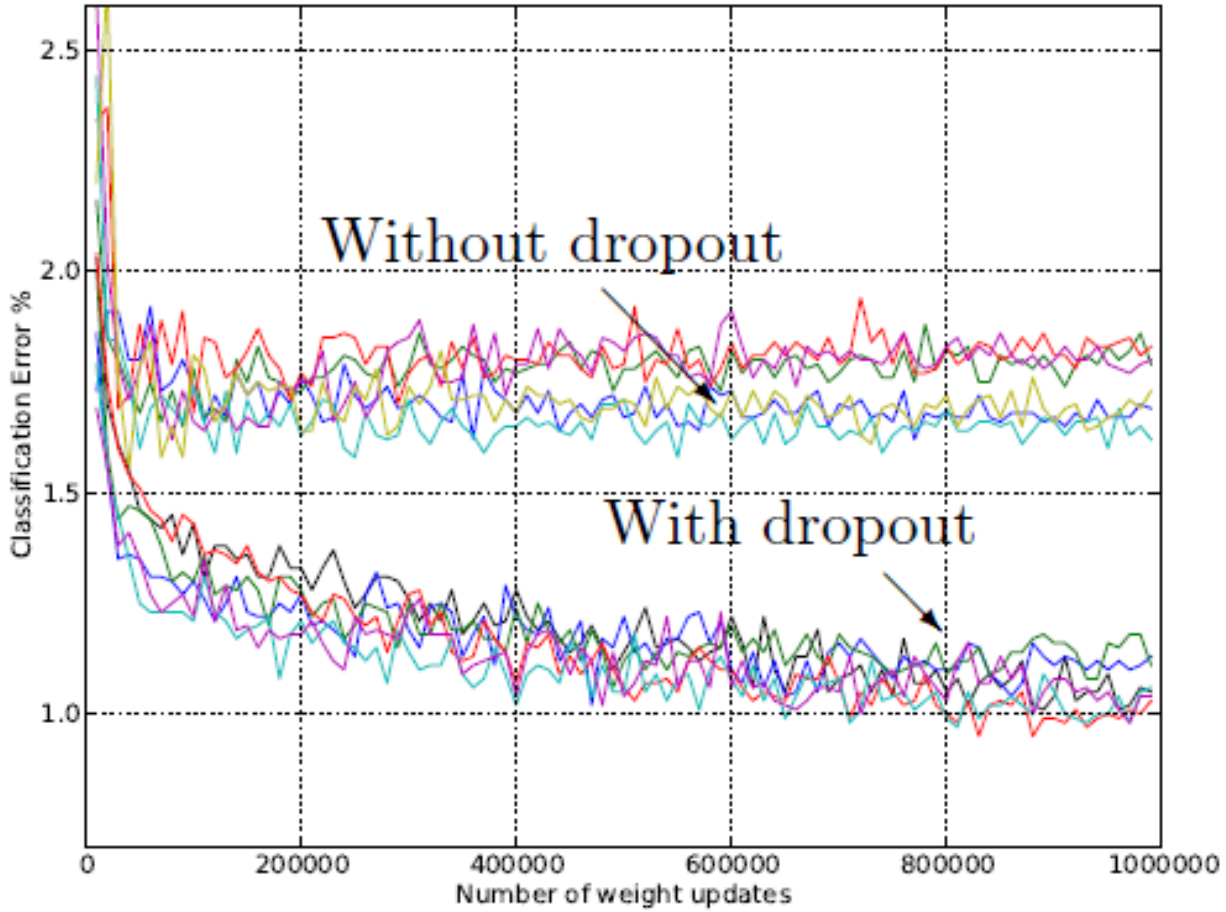
- With H hidden units, we sample from $2^H$ different models
  - Only few of the models get ever trained and they only get 1 training example
- Sharing of weights means that every model is strongly regularized
  - Much better than L1 and L2 regularization, which pulls weights towards zero
  - It pulls weights towards what other models need
  - Weights are pulled towards sensible values
- This works in experiments extremely well

# Dropout – at test time

TECHNISCHE UNIVERSITÄT DARMSTADT

- We could sample many different architectures and average the output
  - This would be way too slow
- Instead: Use all hidden units and half their outgoing weights
  - Computes the geometric mean of the prediction of all $2^H$ models
  - We can use other dropout rates than p=0.5. At test time, multiply weights by 1-p
- Using this trick, we train and use trillions of “different” models
- For the input layer:
  - We could apply dropout also to the input layer
  - The probability should be then smaller than 0.5
  - This is known as denoising autoencoder
  - Currently this cannot be implemented in out-of-the-box Keras

# How well does dropout work?




Classification error on MNIST dataset

Source: Srivastava et al, 2014, Drouput A Simple Way to Prevent Neural Networks from Overtting

# How well does dropout work?

TECHNISCHE UNIVERSITÄT DARMSTADT

- If your deep neural network is significantly overfitting, dropout will reduce the number of errors a lot

- If your deep neural network is not overfitting, you should be using a bigger one
  - Our brain: #parameters >> #experiences
  - Synapses are much cheaper then experiences

# Another way to think about Dropout

TECHNISCHE UNIVERSITÄT DARMSTADT

- In a fully connected neural network, a hidden unit knows which other hidden units are present
  - The hidden unit co-adapt with them for the training data
  - But big, complex conspiracies are not robust -> they fail at test time

- In the dropout scenario, each unit has to work with different sets of co-workers
  - It is likely that the hidden unit does something individually useful
  - It still tries to be different from its co-workers