COMP3308/3608, Lecture 9b
ARTIFICIAL INTELLIGENCE

Deep Learning

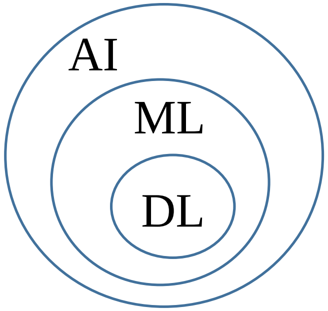
Tutorials on Deep Learning:

1) **http://cs.stanford.edu/~quocle/tutorial1.pdf**
2) **http://cs.stanford.edu/~quocle/tutorial2.pdf**
3) **http://deeplearning.stanford.edu/tutorial/**

Outline

- **What is deep learning?**
- **Autoencoder neural networks**
- **Convolutional neural networks**
- **Applications**

What is Deep Learning?
(high-level definition)



John Kelleher (Deep Learning, MIT Press, 2019)

- **Part of AI that focuses on creating *large NNs* that are capable of making accurate *data-driven decisions***
- **Particularly suited for applications where the *data is complex* and where *large datasets are available***

- **Who uses it?**
  - **Facebook to analyse text in online conversations**
  - **Google, Baidu and Microsoft for image search and machine translation**
  - **Almost all smart phones for speech recognition and face detection**
  - **Self-driving cars –for localization, motion planning and steering, as well as tracking driver state**
  - **Healthcare – for processing medical images (X-ray, CT, MRI)**

Deep Learning an AlphaGo

- **AlphaGo – defeated the world Go champions in 2016 and 2017** **https://theconversation.com/ai-has-beaten-us-at-go-so-what-next-for-humanity-55945 (Lee Sedol in 2016 and Ke Jie in 2017)**
  - **AlphaGo's success was surprising!**
  - **Most people expected that it would take much longer before a computer can compete with top human Go players**
  - **Go is much more difficult for computers than chess – massive search space:**
- **1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,00 0,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,00 0,000,000,000,000,000,000,000,000,000,000,000**
  - **More states (board configurations) than the number of atoms in the universe!**
- **Compare progress in chess and Go:**
  - **Chess: It took 30 years for chess programs to progress from human to world champion level (from (1967 to 1997)**
  - **Go: Using deep learning it took only 7 years to progress from advanced amateur to world champion (from 2009 to 2016)**
  - **=> revolutionary impact of deep learning; big acceleration of performance, also applicable to other fields, not only games**

Deep Learning in the News

- **GoogleTranslate**

http://www.nature.com/news/deep-learning-boosts-google-translate-tool-1.20696

https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html?_r=0

- **Self-Driving cars**

http://spectrum.ieee.org/cars-that-think/transportation/advanced-cars/deep-learning-makes-driverless-cars-better-at-spotting-pedestrians

Deep Learning in the News (2)

- http://www.timesnow.tv/technology-science/article/deep-learning-google-maps-to-become-more-accurate-through-artificial-intelligence/60610
- https://venturebeat.com/2017/04/07/how-olay-skin-advisor-built-their-deep-learning-algorithms/
- http://www.newyorker.com/magazine/2017/04/03/ai-versus-md
- https://www.techemergence.com/deep-learning-applications-in-medical-imaging/
- https://www.wired.com/2014/02/netflix-deep-learning/

What is Deep Learning?
(more specific definitions)

- ***Deep Learning*** **means different things to different people in AI:**

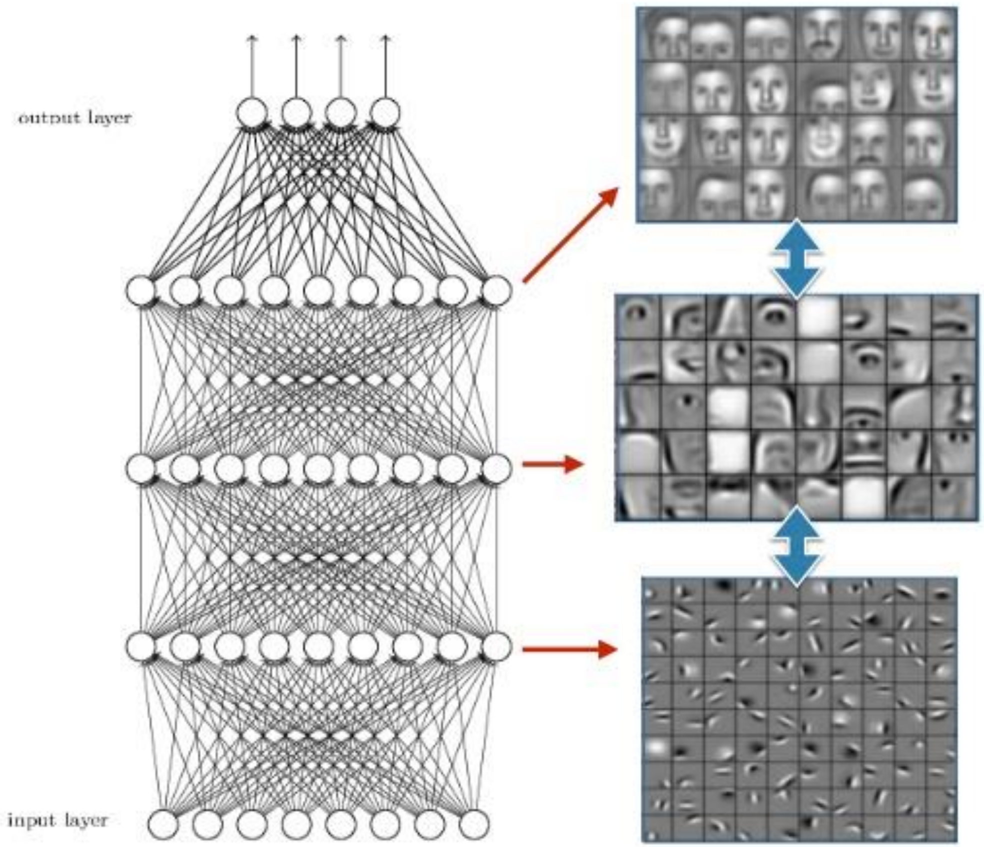
  1. **The NN has *more than 1 hidden layer***

  2. **No need for human-invented and pre-selected features – the NN is *able to learn the important features automatically***

  3. **Some deep learning architectures use *unlabeled data for pre-training* of the NN layers, which is followed by supervised learning**

What is Deep Learning? (2)

- ***Deep Learning:*** **NNs that learn hierarchical feature representations**
- **Novel techniques developed in the last 10 years**



Backpropagation NNs - Issues

- **Training is slow – requires many epochs**
- **The NN is typically fully connected – too many parameters to adjust**
- **The weights are initialized randomly and then adjusted by the gradient descent – is there a better way to do this?**
- **With many hidden layers, the learning becomes less effective**
  - **The *vanishing gradient problem* – the weight changes for the lower levels are very small; these layers learn slower than the higher hidden layers**
- **Require a large dataset of labeled data – this may not be available or difficult to obtain**
- **May get stuck in a local minimum and not find a good solution**
- **Require feature-engineering to select useful features and represent them appropriately (most ML algorithms require this); can we learn the important features automatically?**

Why do we Need More than One Hidden Layer?

- **Cybenko's Theorem:** **Backpropagation NNs with 1 hidden layer are universal approximators – can learn any function with arbitrary low error. Then why do we need more than 1 hidden layer?**

- **1) This is an existence theorem, i.e. it says that there is a NN with 1 hidden layer that can do this but doesn't tell us how to find this NN**

- **2) This doesn't mean that 1 hidden layer is the most effective representation that will result in the fastest learning, easiest implementation or best solution (ability to classify correctly new examples)**

Deep Learning Architectures

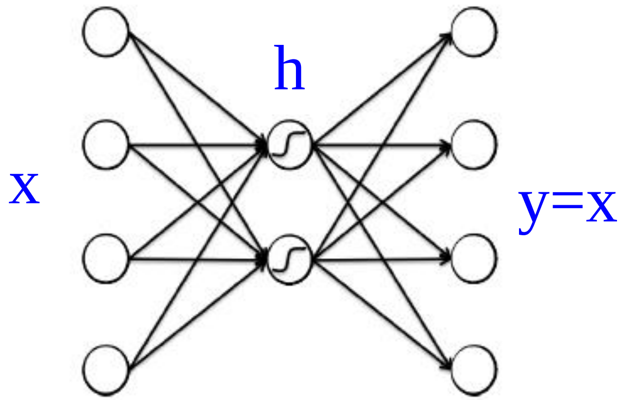
1. **Stacked autoencoder networks**
2. **Restricted Bolzmann machines**
3. **Convolutional networks**

- **We will study 1 and 3**
- **2 are similar to 1 - use unsupervised learning for pre-training of the layers**

Autoencoder Neural Networks

Autoencoder NN

- **We have a set of input vectors without their class (unlabelled data): x ={x1,x2, x3…}**
- **Each xi is a n-dim vector representing 1 input vector**
- **An autoencoder NN:**
  - **Sets the target values to be the same as the input values (yi=xi) and uses the backpropagation algorithm to learn this mapping**
    - **=> the number of input and output neurons is the same**
  - **Has 1 hidden layer with a smaller number of neurons than the input neurons**

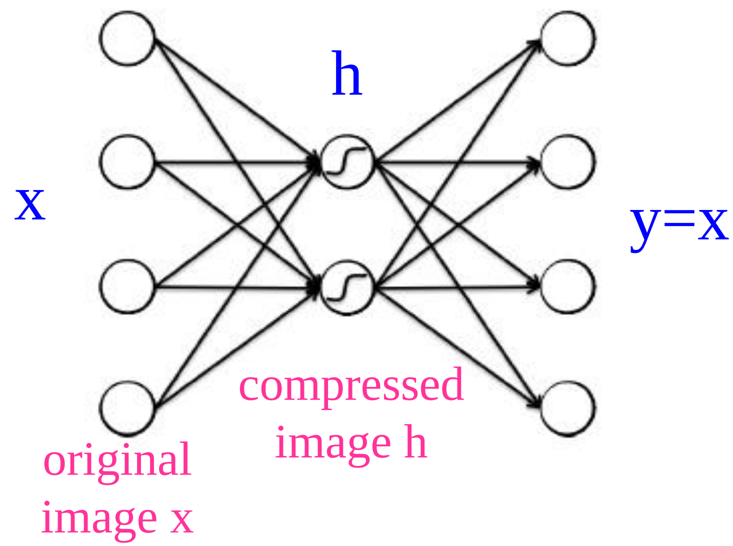


Autoencoders - History

- **Autoencoders were first mentioned by Rumelhart, Hinton and Williams in 1986 in the paper which introduced the backpropagation algorithm: http://www.cs.toronto.edu/~fritz/absps/pdp8.pdf**
- **They are typically used for dimensionality reduction, image and data compression**
- **More recently - in deep NN for pre-training of the network (weight initialization)**

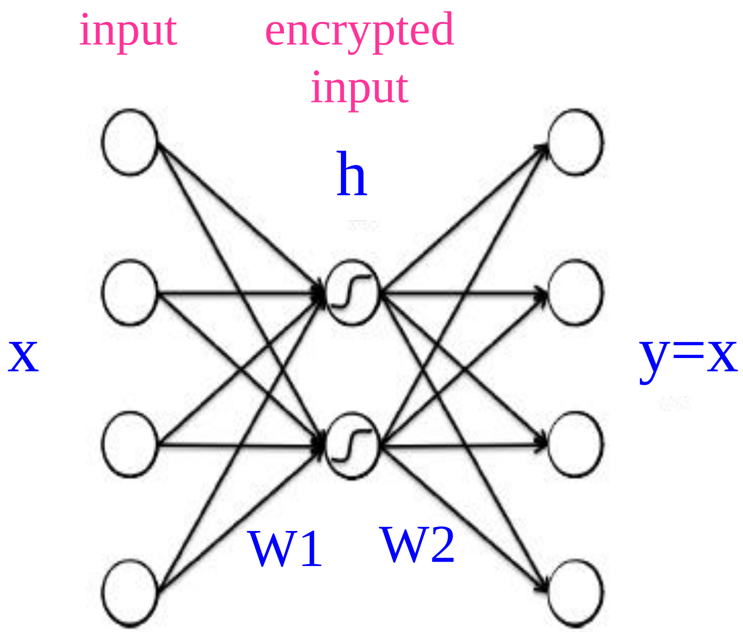
Autoencoder NN – Main Idea

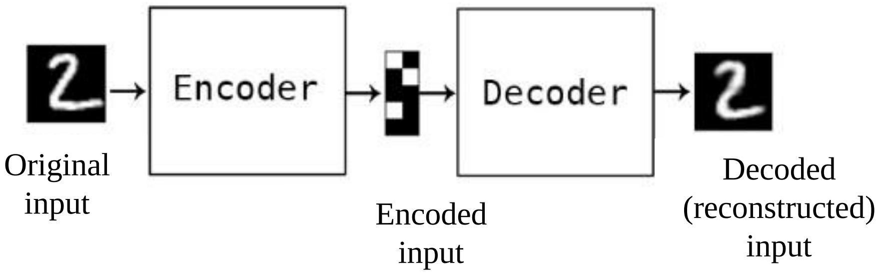
- We are interested in the hidden layer, in particular the outputs of the hidden neurons
  - **hi** – the vector at the hidden layer for input vector **xi**
- The hidden layer can be seen as trying to learn a compressed version of the input vector
  - Compressed because the number of hidden neurons is smaller than the number of input neurons
- Example - we can use the autoencoder for image compression:
  - **x** are the pixel values of a 10x10 image => **xi** is a 100-dim vector
  - We have 50 hidden neurons – **hi** is 50-dim vector
  - The network learns a compressed representation of the image – **hi** is a compressed version of **xi**



Autoencoders – Traditional Applications

- In addition to image and data compression, autoencoders can be used for encryption
  - The weights W1 perform encoding
  - The weights W2 perform decoding
- The receiver needs W2 to decode the encrypted input



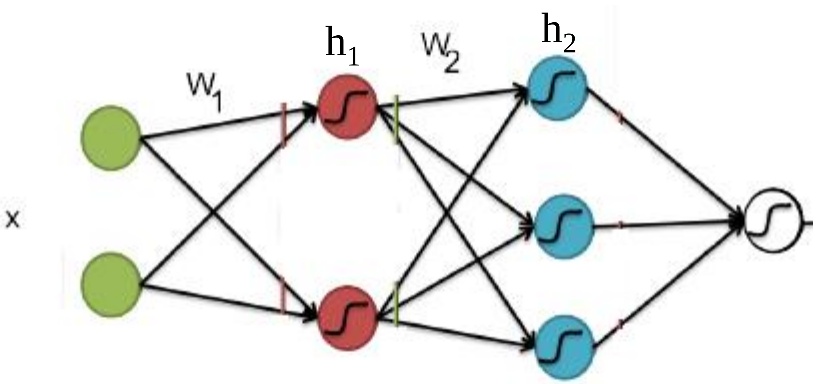


Autoencoders as Initialization Method for Deep NN

- **Can be used to pre-train the layers of a deep NN in advance**
  - **1 layer at a time, 1 autoencoder for each layer**

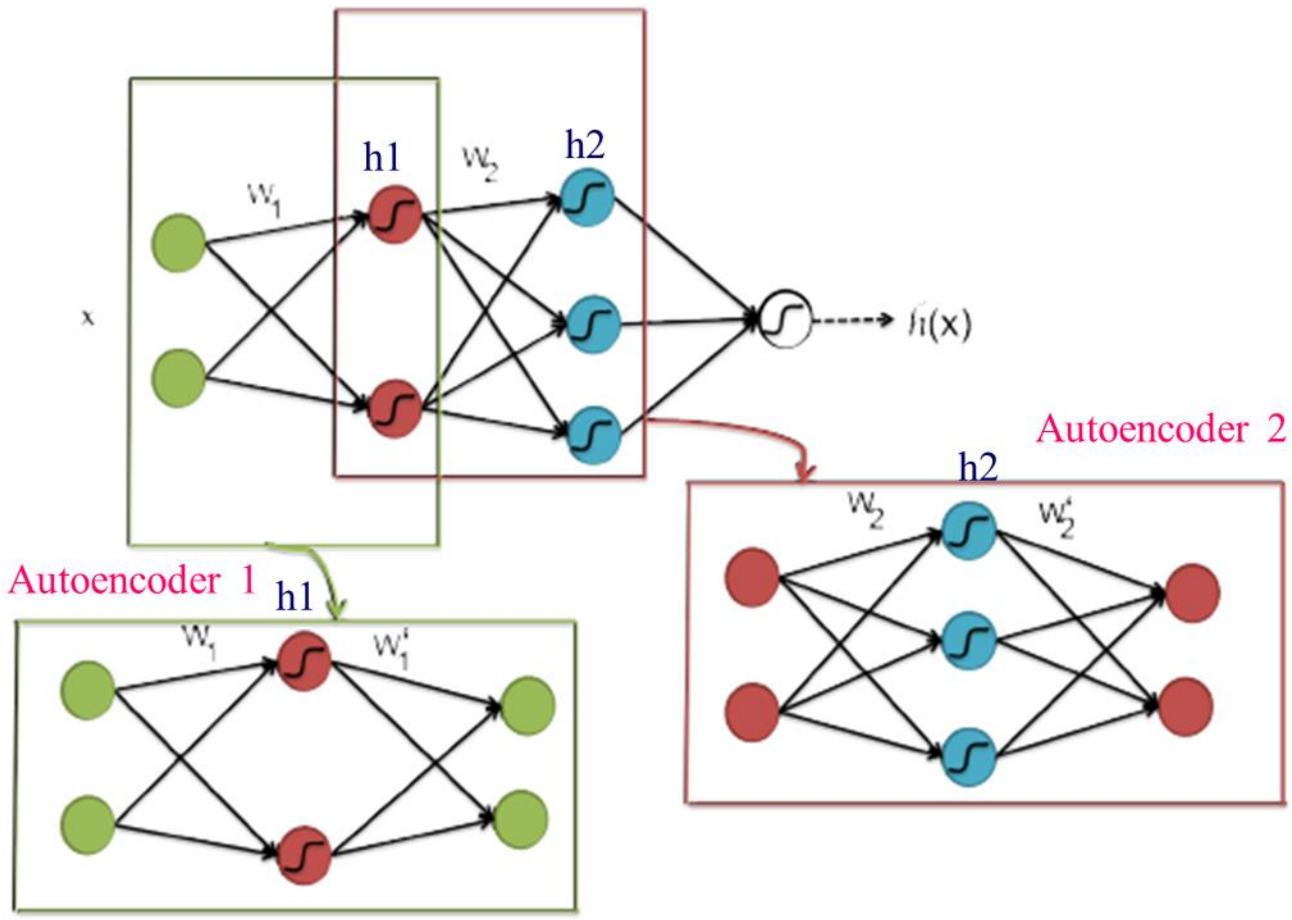
- **The training of a deep NN will include 3 steps:**
  1. **Pre-training step: Train a sequence of autoencoders, 1 for each layer (unsupervised)**
  2. **Fine-tuning step 1: Train the last layer using backpropagation (supervised)**
  3. **Fine-tuning step 2: train the whole network using backpropagation (supervised)**

Example: Let's use this method to pre-train a deep NN with 2 hidden layers, h1 and h2



Autoencoders as Initialization Method - Example

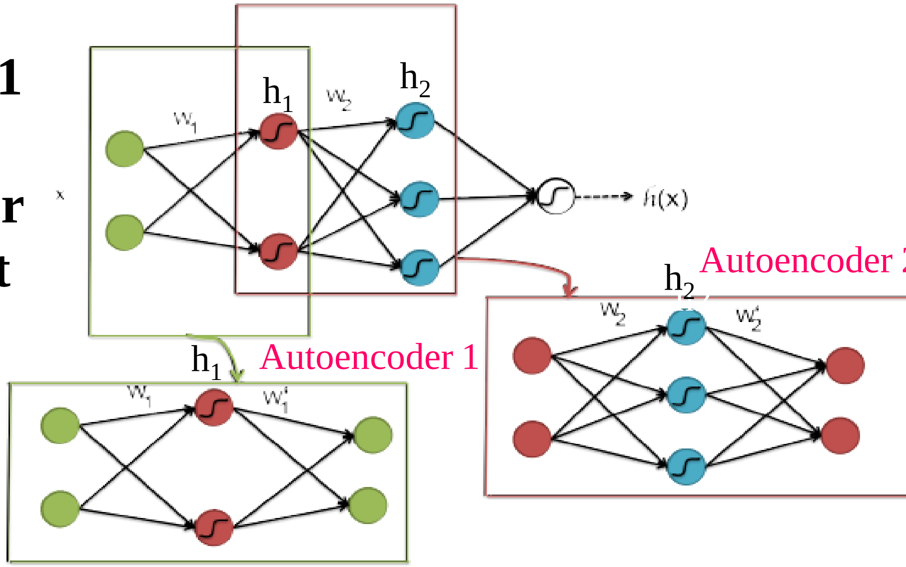
- **Pre-training step: Train a sequence of autoencoders, 1 for each layer (unsupervised) = 2 autoencoders for our example**



How is the Pre-training Done? (1)

- **Pre-training means finding W1 and W2 for our deep NN**
- **To find W1, we train Autoencoder 1 with weights W1 and W1' and h1 num. of hidden neurons (unsupervised using the input vectors x only)**
- **After the training is completed:**
  - **The learned W1 is set in the deep NN as values for the weights between the input and first hidden layer**
  - **W1' is not needed; it is discarded**

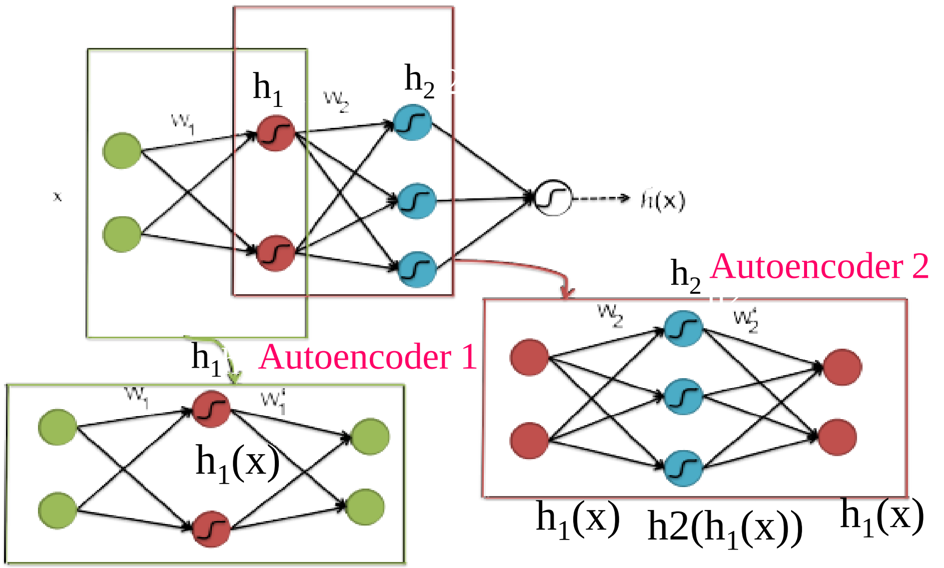
- **But we also need to find W2 – the weights between the hidden layer h1 and the hidden layer h2**
- **This will be done using Autoencoder 2, but we need to compute the input for Autoencoder 2**



How is the Pre-training Done? (2)

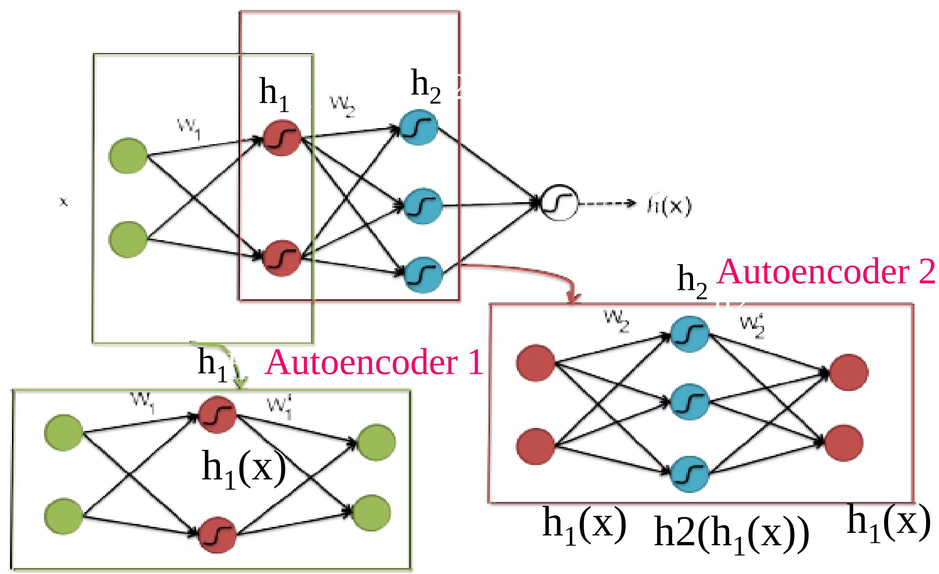
- **Computing the input for Autoencoder 2:**
  - **h1 has formed h1(x), a compressed representation of the input data x, i.e. has discovered and extracted useful structure/pattern (we hope)**
  - **we use the learned W1 to compute the values of the neurons in h1 in Autoencoder 1 for all the data (all training examples), i.e. we compute h1(x)**
  - **These values will be used as an input to Autoencoder 2**

  - **h1(x) can be seen as a different representation of the training data – a transformation applied to the training data**



How is the pre-training done? (3)

- **To find W2**, we train Autoencoder 2 with weights W2 and W2’ and h2 num. of hidden neurons (unsupervised using the output produced by Autoencoder 1, i.e. using h1(x))
- **After the training is completed:**
  - The learned W2 is set in the deep NN as values for the weights between the first hidden and second hidden layer
  - W2’ is discarded

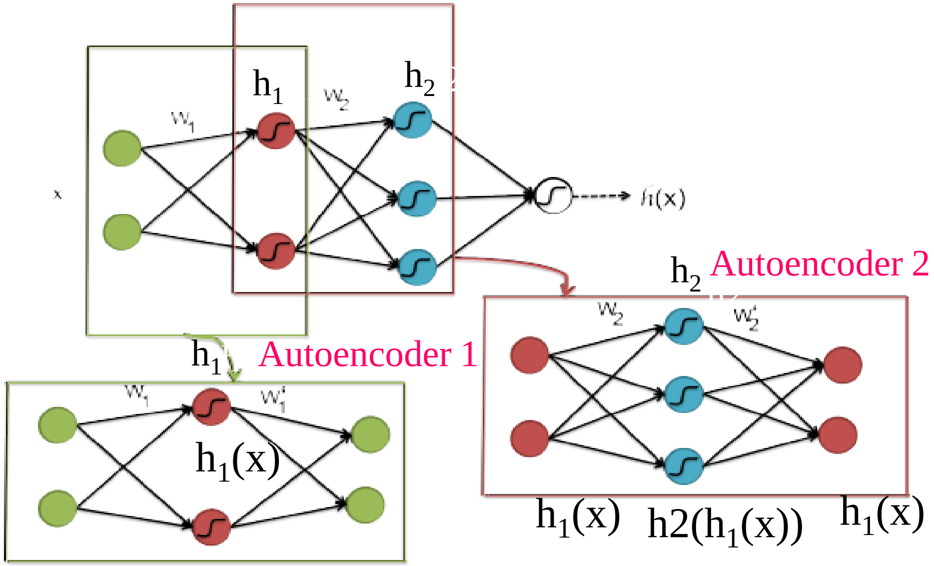


How is fine tuning step 1 done?

- **The next step is:**

Fine-tuning step 1: Train the last layer using backpropagation (supervised)

- **We need to compute the input for this training, which is the output of h2**
- **h2 has formed h2(h1(x)), a compressed representation of h1(x)**
- **We use the learned W2 to compute the values of the neurons in h2 in Autoencoder 2 for all our data**



Stacked Autoencoders

- Using several autoencoders for pre-training in this way is called *stacking autoencoders*
  - Each layer of the network learns an encoding of the layer below
  - The network can learn hierarchical features in an unsupervised way
- The network is called a *stacked autoencoder*

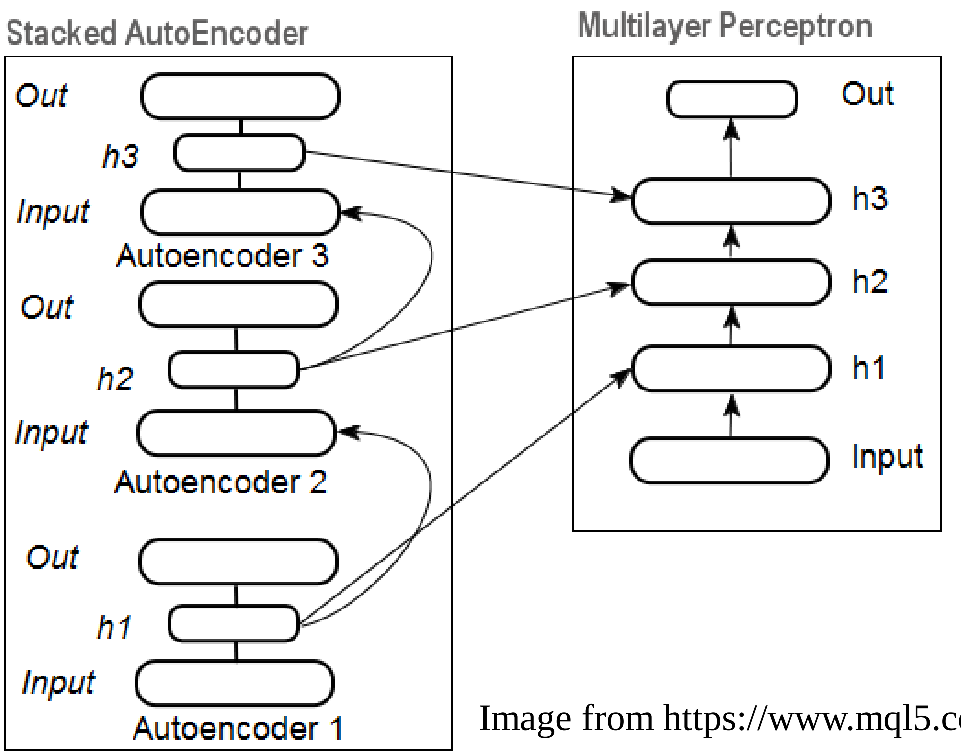
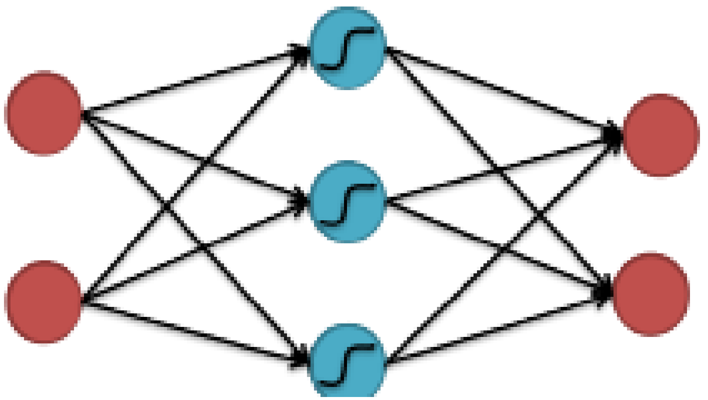


Image from https://www.mql5.com/en/articles/1103

Other Types of Autoencoders

- ***Sparse autoencoder*** **- an autoencoder with more hidden neurons than inputs**
  - **It doesn't compress the input but may still discover interesting structure in data, a different representation that may be useful**
- ***Denoising autoencoders***
  - **A percentage of data is randomly removed**
  - **This forces the autoencoder to learn robust features that generalize better**
  - **Similar to another idea - dropout – see next slides**



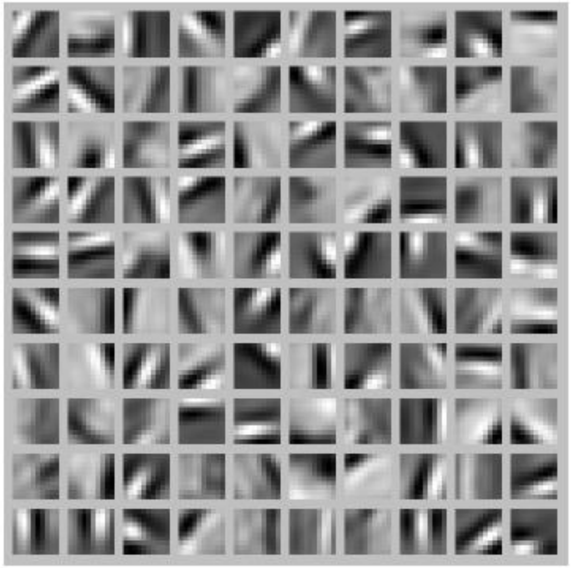
Visualizing a Trained Autoencoder

- Consider image processing
- We have trained the autoencoder on 20x20 images and have 100 hidden neurons
- After the training has completed, we would like to visualize what the autoencoder has learnt (i.e. the function computed by each hidden neuron hi)
- We will do this as an image – for each hidden neuron we will visualize the input that maximizes the neuron's activation
- It can be shown that this image is formed by pixels computed as:

$$x_j = \frac{w_{ij}}{\sqrt{\sum_{j=1}^{400} w_{ij}^2}}$$

Visualizing a Trained Autoencoder (2)

- Each square shows the image that maximally activated each of the 100 hidden neurons
- Some of the hidden neurons have learned to detect edges at different positions and with different orientations
  - These are useful features for object recognition



Visualizing a Trained Stacked Autoencoder

- A stacked autoencoder can learn a hierarchy of features
- Example: handwritten digit recognition (MNIST dataset, 60 000 training and 10 000 testing examples of 28x28 handwritten digits)
- 3 stacked autoencoders were used to pre-train a NN
    - 1st hidden layer has learned stroke-like features
    - 2nd hidden layer – digit parts
    - 3rd layer – entire digits

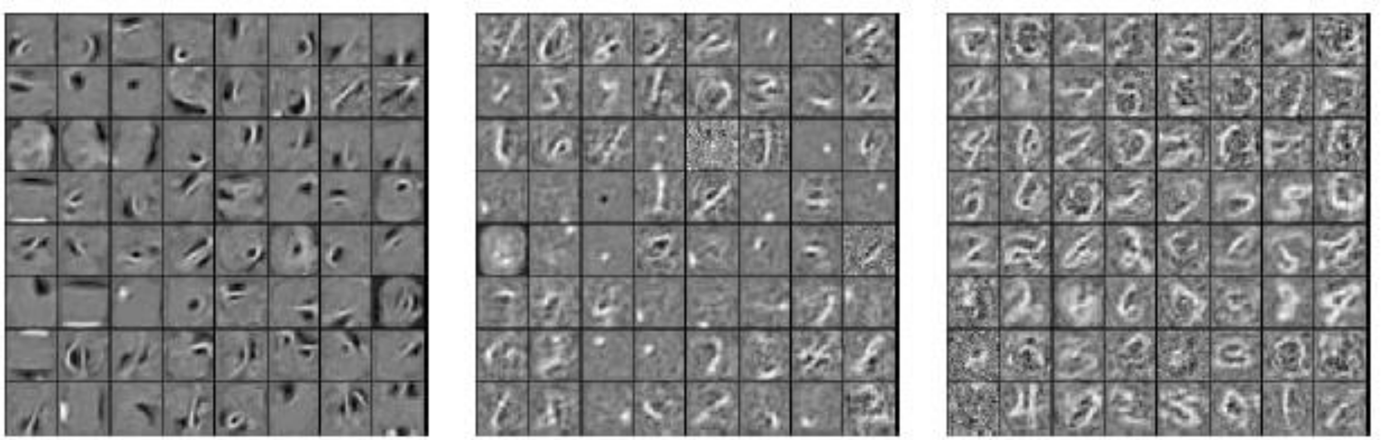


Image from Erhan et. al (2010) – Why does unsupervised pre-training help deep learning? JMLR 2010, http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf

Autoencoders - Advantages

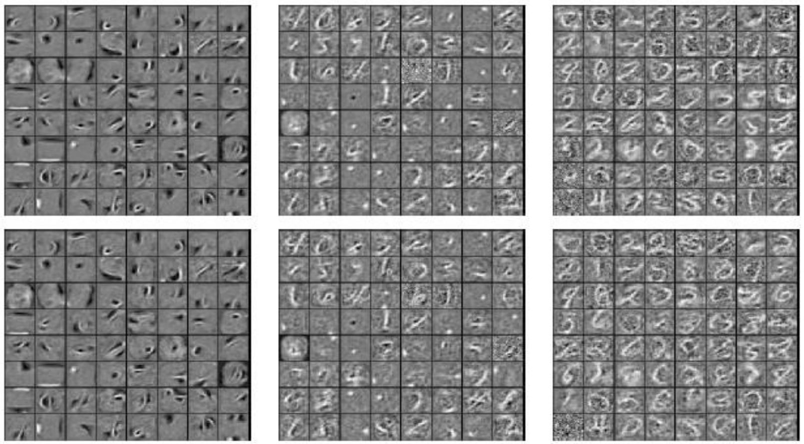
- **Able to automatically learn features from unlabeled data**
  - **Especially important for sensory data applications - computer vision, audio processing and natural language processing - where researchers have spent many years manually devising good features (vision, audio and text)**
  - **Note: in many domains the features learnt by autoencoders are still not superior than the best hand-engineered features but there are some emerging cases where they are (with more sophisticated autoencoders)**
- **Useful for pre-training layers of deep NNs – Erhan et al. (2010)**
  - **Shown experimentally that NNs pre-trained with autoencoders converge faster and have better generalization ability (i.e. finds a better solution)**
  - **In contrast, the standard randomly initialized deep NN is slower to train, and easily gets stuck in a poor local minima**

Why Does Unsupervised Pre-training Help Deep Learning?

- **Erhan et al (2010), http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf**
- **Compared deep NNs with and without pre-training experimentally on several big dataset – results:**
  - **NNs with pre-training have better accuracy on test data than NNs without pre-training**
  - **In NNs without pre-training, the probability to find a poor local minimum increases as the number of hidden layers increases. NNs with pre-training are robust to this.**
  - **NN with pre-training provide a better starting position for the NN – in a "basin" with a better local minimum**

Results With and Without Pre-training

Pre-trained NN:



after pre-training

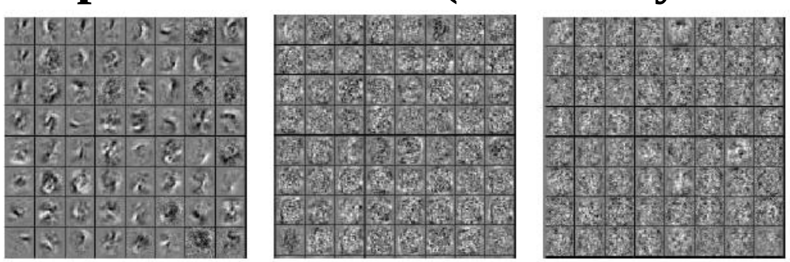
after fine-tuning with backpropagation

hidden layer first second third

1. Not a big difference – pre-training already provided a good starting position, the fine-tuning doesn't see to change the weights significantly

2. The fine-tuning changes least the first layer

Not pre-trained NN (randomly initialized):



after training with back-propagation

Layers 2 and 3 doesn't seem to learn structured features (at least not visually interpretable features)

Convolutional Neural Networks

Convolutional NNs

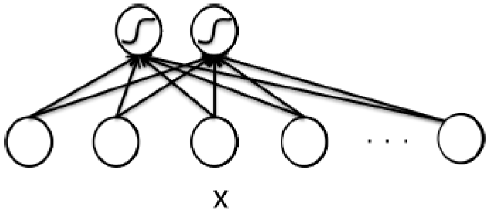
- **Introduced by LeCun et al. in 1989**

  http://yann.lecun.org/exdb/publis/pdf/lecun-89e.pdf
- **A special type of multilayer NNs**
  - **Trained with the backpropagation algorithm as most of the other multilayer NNs but have a different architecture**
- **Designed to recognize visual patterns directly from pixel images with minimal pre-processing**
- **Can recognize patterns with high variability, e.g. handwritten characters, and are robust to distortions and geometric transformations such as shifting**
- **Used in speech and image recognition; have shown excellent performance in hand-written digit classification, face detection, image classification (e.g. the ImageNet dataset)**

http://yann.lecun.com/exdb/lenet/

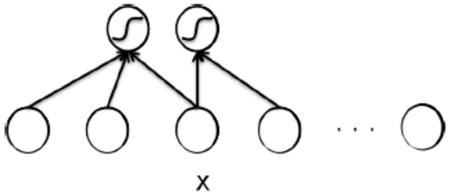
Main Idea 1 – Local Connectivity

- **Fully connected network – each neuron from a given layer is connected with each neuron in the next layer – too many connections per neuron**
- **Ex.: The input is a 100x100 pixels image => input vector is 10^4 dimensional; each hidden neuron in the first hidden layer will have 10^4 connections = 10^4 weights (+ 1 bias weight) to learn = too computationally expensive**



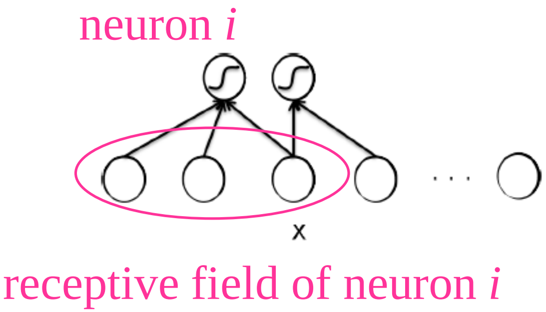
Fully connected neuron

- **Instead, we can restrict the connections - each hidden neuron is connected only to a small subset of inputs, corresponding to adjacent pixels (a patch, continuous region in the image)**
- **Inspired by biological neural systems, e.g. neurons in the visual cortex have localized receptive fields (i.e. respond only to stimuli in a certain location)**



Locally connected neuron

Local Connectivity (2)



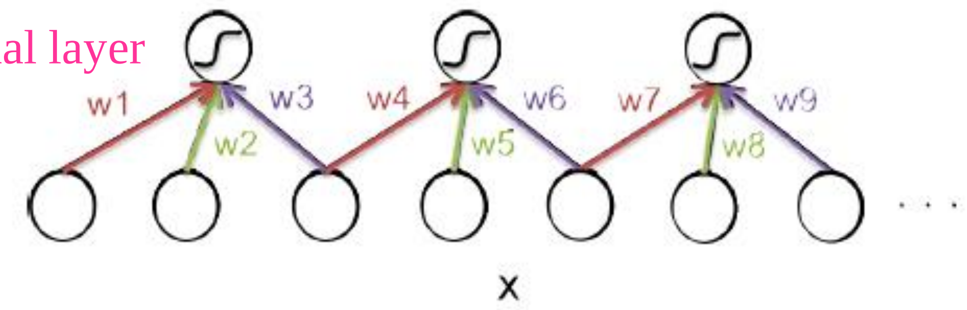
- **With local connectivity, each neuron is responsive to changes in its inputs only (i.e. in its *receptive field*)**

- **We can extend this idea to all layers**
- **We can easily modify the backpropagation algorithm to work with local connectivity:**
  - **Forward pass – assume that the missing connections have weights 0**
  - **Backward pass: no need to compute the gradient for the missing connections**

Main Idea 2 – Sharing Weights

- **The number of connections can be further reduced by *weight sharing* - some of the weights are constrained to be *equal* to each other – example:**

convolutional layer



w1=w4=w7
w2=w5=w8
w3=w6=w9

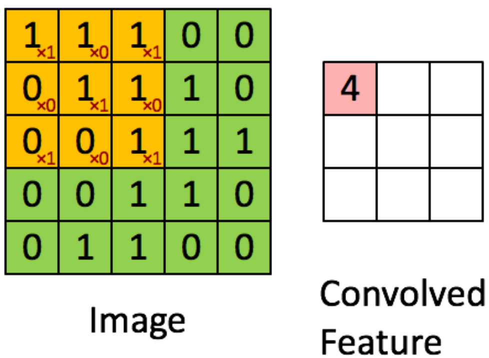
- **=> we need to store a smaller number of weights – instead of storing weights from w1 to w9, we will store w1, w2 and w3 only**

- **Weight sharing means using the same weights to different parts of the image**
- **This is similar to the *convolution* operation in signal processing where a filter (a set of weights) is applied to different positions in the input signal => this layer is called *convolutional layer***

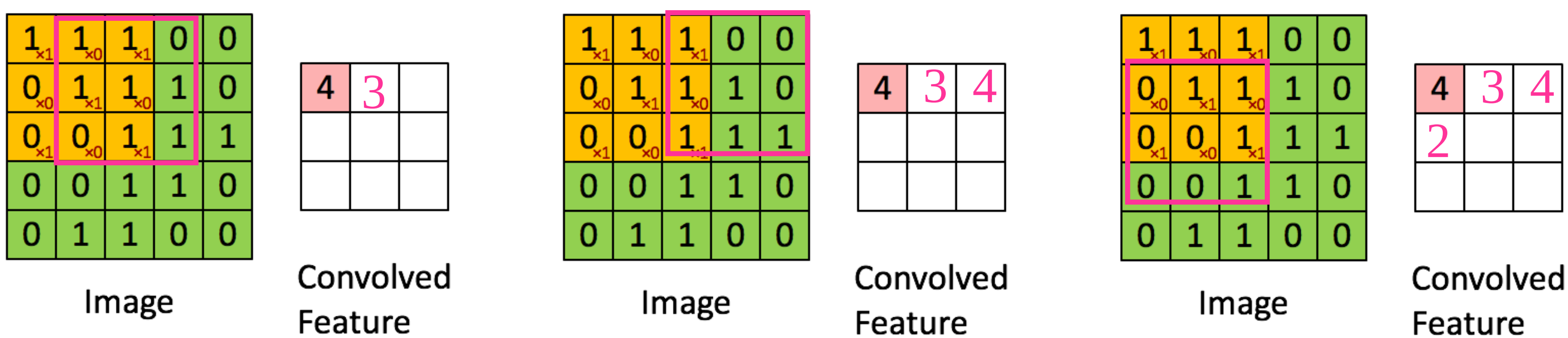
Convolution - Example

- Convolution is like applying a sliding window to a matrix
- The corresponding elements are multiplied and summed

Demo at http://deeplearning.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

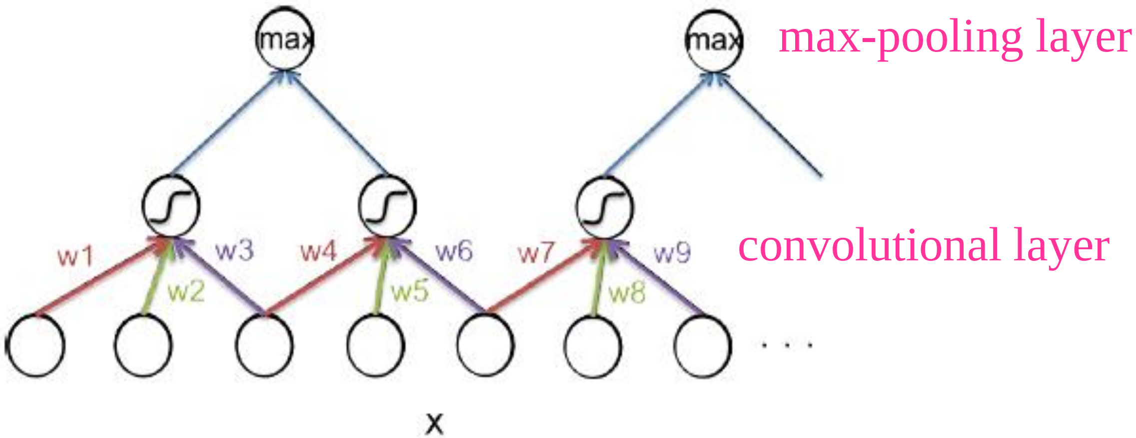


- image of black and white values: 0 is black 1 is white
- 3x3 sliding window (filter, kernel) with values shown in red
- 4 = 1*1+1*0+1*1+0*0+1*1+1*0+0*1+0*0+1*1
- Then the window is shifted as shown



Main Idea 3 – Pooling

- **The convolutional layers is used together with a *max-pooling* layer**
  - **It takes the maximum value of a selected set of neurons from the convolutional layer**



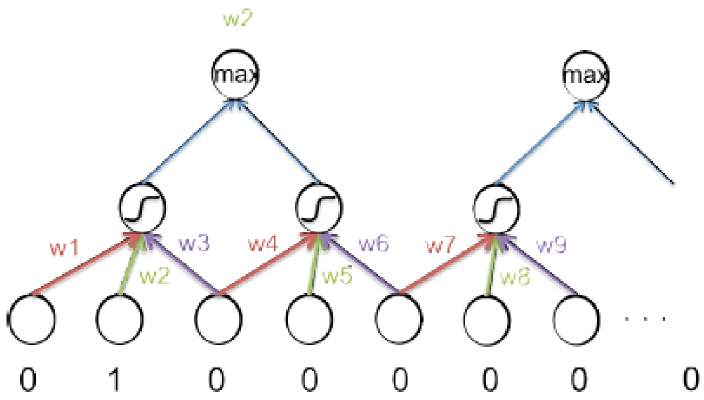
- **The pooling layer is also called a *subsampling layer* because it reduces the size of the input data**

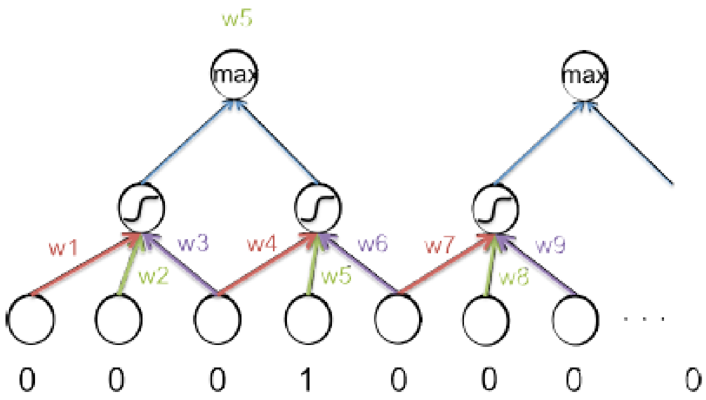
- **Important property: the output of a max-pooling neuron is invariant to shifts in the inputs**

Pooling (2)

- Example: 2 input images (1-dim), each with a white dot which got shifter 2 pixels to the right:

  x1=[0,1,0,0,0,0,0…] x2=[0,0,0,1,0,0,0…]





- output first max-pooling neuron: w2 for x1 and w5 for x2
- But w2=w5, so the value of the neuron is the same

- => the outputs are invariant to translation
- Translational invariance is important for natural data such as images and sounds as translation is one of the major sources of distortion

Main Idea 4 – Local Contrast Normalization

- **Sometimes the max-pooling layer is followed by another layer, called Local Contrast Normalization (LCN) layer**
- **It normalizes the output of each max-pooling neuron by subtracting the mean of their incoming neurons and dividing by the standard deviation of these neurons**
- **LCN allows for brightness invariance => useful for image recognition**

Convolutional NNs

- **Contain a convolutional layer followed by a max-pooling layer and sometimes an LCN layer**
- **This can be repeated several times – the output of the max-pooling layer is an input to a convolutional layer, followed by a max-pooling layer and a LCN layer, etc.**
- **Finally, there is 1 or 2 fully connected hidden layers**
- **The backpropagation algorithm can be easily modified to train convolutional NNs**

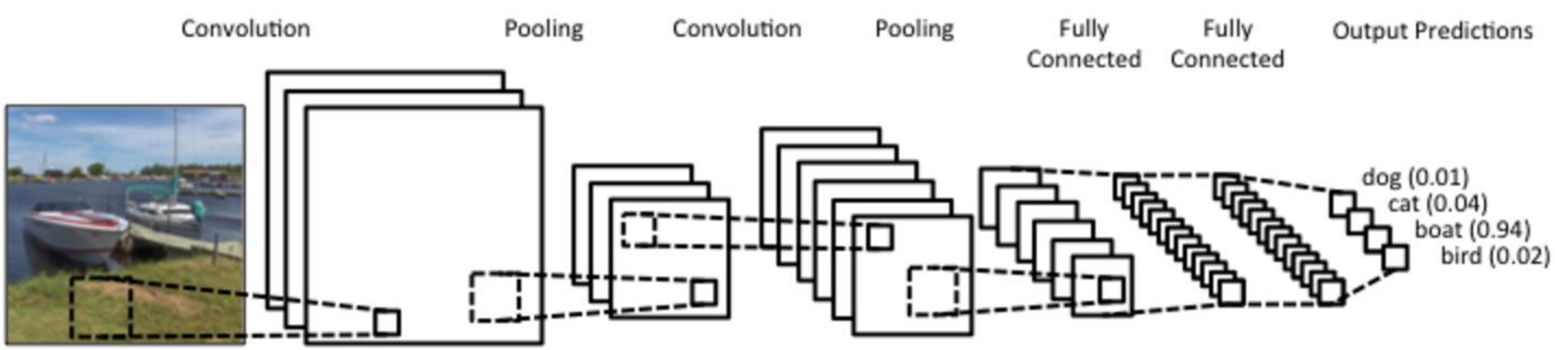
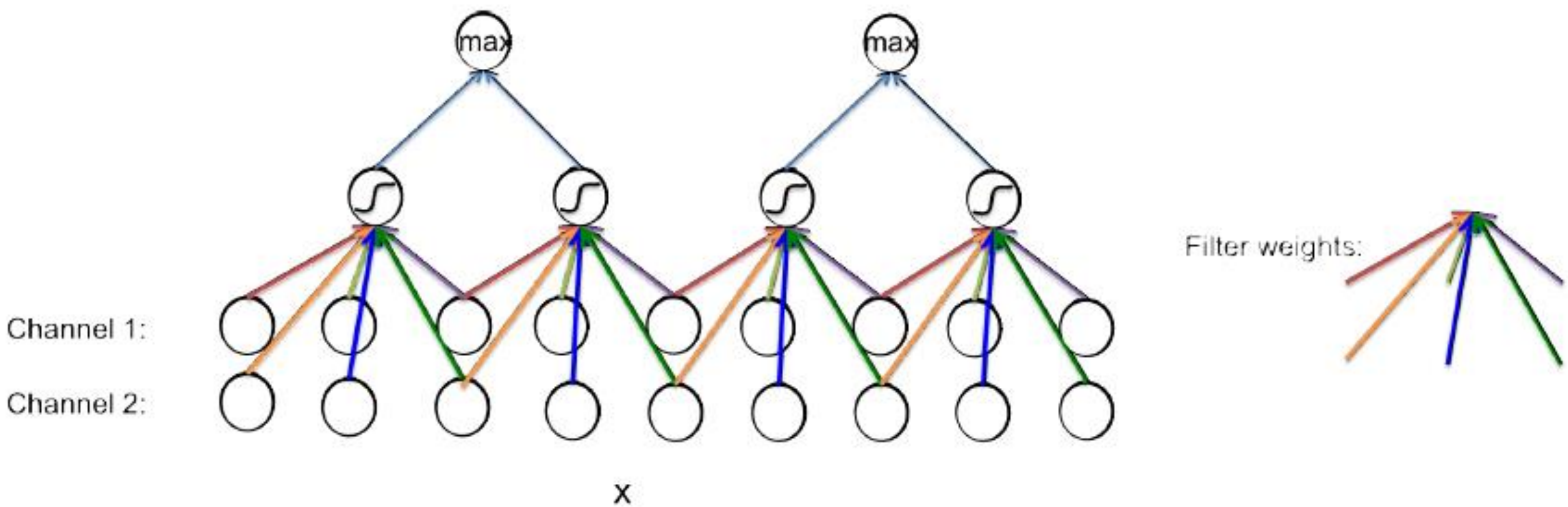


Image from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Convolutional NNs with Multi-channel Inputs

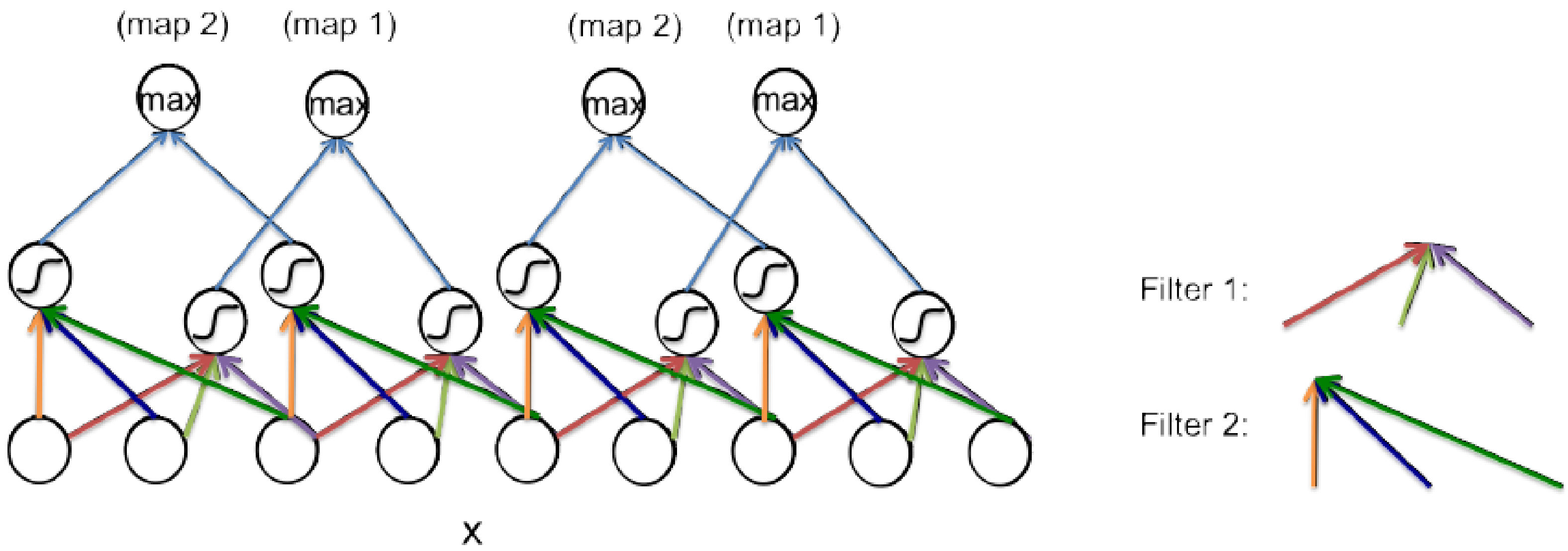
- Images have multiple channels, e.g. Red, Green and Blue
- We can modify the convolutional NN architecture to work with multiple channels
  - A filer that looks at multiple channels
  - Weights are not shared across a channel, only within a channel



Convolutional NNs with Multiple Maps

- **So far we had 1 filter for an input; we can have more than 1**
- **E.g. we can have 2 filers looking at 1 pixel**
- **The output produced by each filter is called a *map***

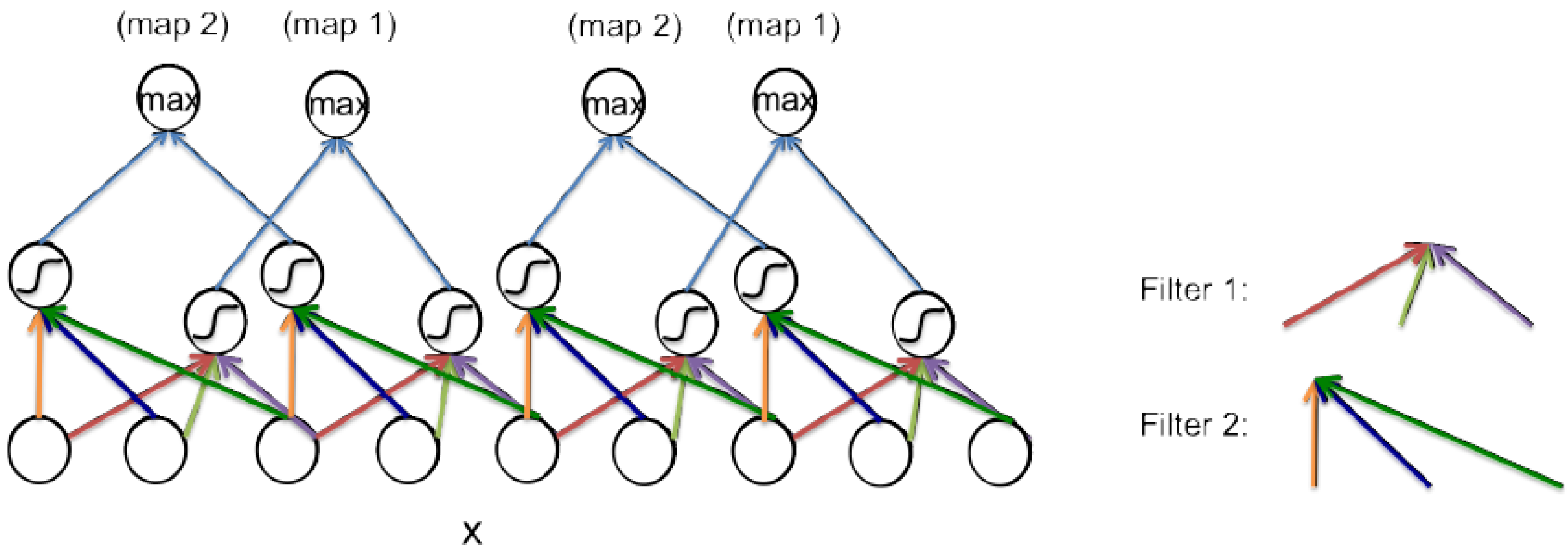
- **Map 1 is created by Filter 1, Map 2 is created by Filter 2**



Convolutional NNs with Multiple Maps

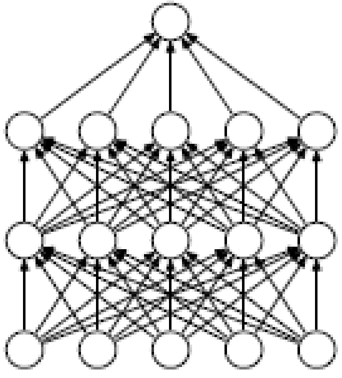
- So far we had 1 filter for an input; we can have more than 1
- E.g. we can have 2 filers looking at 1 pixel
- The output produced by each filter is called a ***map***

- Map 1 is created by Filter 1, Map 2 is created by Filter 2

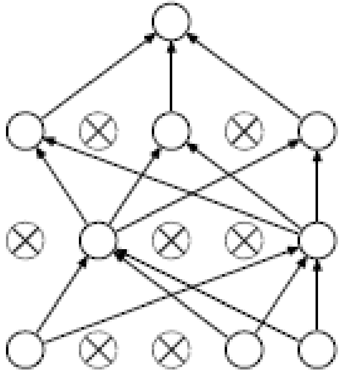


Main Idea 5 – Dropout

- **Used in the fully connected layers to prevent overfitting**
  - **During training, at each iteration of the backpropagation, we select randomly neurons in each layer and set their values to 0 (i.e. we drop them out from the weight adjustment = we temporarily disable them)**
  - **During testing, we do not drop out any neurons but scale their weights**
  - **Example: neurons at layer *l* have a probability *p* to be dropped out; p=0.5 means that 50% of the neurons will be dropped out. During testing the incoming weights to layer *l* are multiplied by *p***
- **Dropout forces the NN to be less dependent on certain neurons, to collect more evidence from other neurons => to be more robust to noise**



(a) Standard Neural Net



(b) After applying dropout.

Image from Shrivastava et al. (2014). Dropout: A simple way to prevent neural networks from overfitting, https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Applications of Deep NNs

LeNet-5 for Handwritten Digit Recognition

- **LeCun, L. Bottou, Y. Bengio and P. Haffner (1998), Gradient-based learning applied to document recognition, Proceedings of the IEEE**

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

- **Architecture: 2 convolution, 2 max-pooling, 3 fully connected**
- **Local connectivity not full connectivity**

PROC. OF THE IEEE, NOVEMBER 1998 7

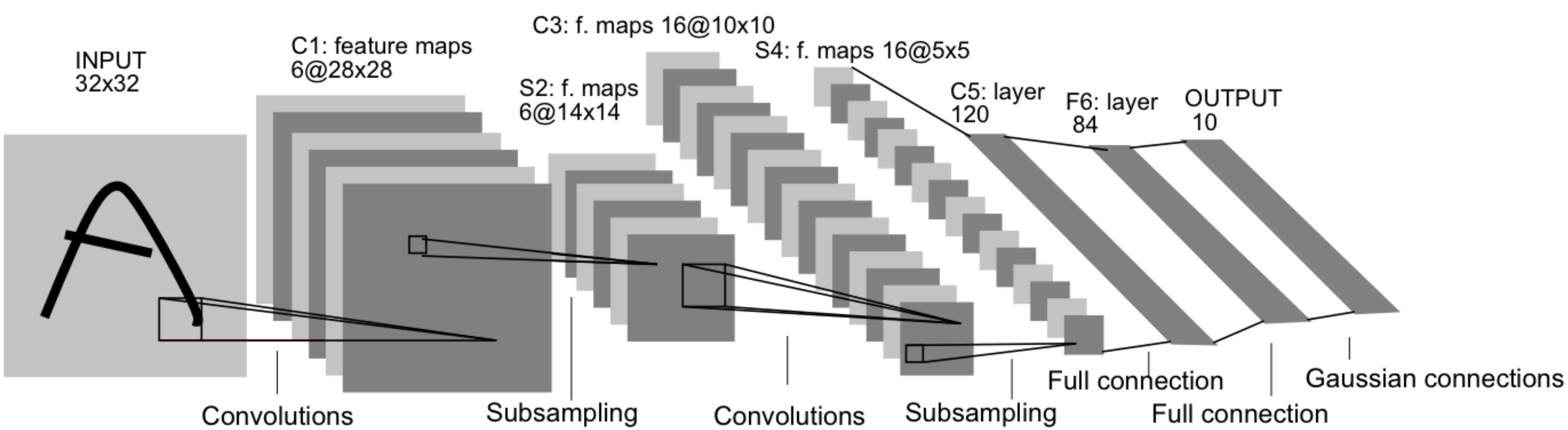


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet-5 for Handwritten Digit Recognition (2)

Image from http://yann.lecun.com/exdb/lenet/multiples.html

Applications of Deep Learning Applications

- **Classification of images into different categories**
- **Krizhevsky, I. Sutskever, G. Hinton (2012), ImageNet classification with deep convolutional neural networks, Proceedings of NIPS**

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

- **Training data: 1.2 million images from ImageNet dataset labelled in 1000 classes**

Image from http://vision.stanford.edu/resources_links.html

ImageNet Classification with Convolutional NN

- **Trained a deep convolutional NN:**
  - **5 convolutional layers**
  - **3 max-pooling layers**
  - **3 fully connected**
- **60 million weights and 650 000 neurons**
- **Used dropout, different activation function (rectified linear) and very efficient GPU implementation of the convolution operation**

- **Achieved a very impressive performance: 16.4% error rate (83.6% accuracy)**

Dramatic Improvement in Deep Learning – Why Now?

The main ideas and algorithms have been around for a long time, why do we see this dramatic improvement only now?

Reasons:

1. **Computational power - fast and powerful computers; powerful GPUs (Graphics Processing Units)**
2. **Availability of much larger datasets, especially labelled datasets - millions of examples**
3. **Some new ideas, e.g. dropout, using autoencoders for pre-training of the NN, ability to visualize what the hidden layers have learnt**

Caution

- **Just because deep NNs are very popular now, it doesn't mean that they are a panacea that will solve all ML problems!**
- **Depending on the problem, classical shallow NNs and other ML algorithms may do even better**

"A bulldozer is more powerful than a spade, and yet the gardener prefers the spade most of the time."

Based on M. Kubat, Introduction to ML, Springer, 2017

Interpretability

- **Often we need not only a decision (e.g. predicted class) but also a reasoning behind it**
- **Especially important when the decision concerns a person, e.g.**
  - **medical diagnosis**
  - **credit assessment**
- **Privacy and ethics regulations – individuals affected by decisions made by automated systems have the right for an explanation how the decision was made**
- **Different algorithms provide different level of interpretability but deep learning models are probably the least interpretable – disadvantage**
- **Research on interpreting deep learning networks is becoming more important**

Based on John Kelleher (Deep Learning, MIT Press, 2019)

Software

Matlab, https://au.mathworks.com/discovery/deep-learning.html

Keras, https://keras.io/

TensorFlow - Google Brain, https://www.tensorflow.org/

Theano – Uni Montreal, http://deeplearning.net/software/theano/

Caffe - Berkeley AI Research, http://caffe.berkeleyvision.org/

Torch, https://github.com/torch/torch7

https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

References

- Quoc V. Le (2015), A tutorial on Deep Learning – part 1 and part 2

  http://cs.stanford.edu/~quocle/tutorial1.pdf

  http://cs.stanford.edu/~quocle/tutorial2.pdf
- Stanford Deep Learning tutorial

  http://ufldl.stanford.edu/tutorial/

  http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

  http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/
- Andrew Ng (2011), Sparse autoencoder

  https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf
- Michael Nielsen (2017), Neural Networks and Deep Learning,

  http://neuralnetworksanddeeplearning.com/
- Yann LeCun, Yoshua Bengio and Geoffrey Hinton (2015), Deep Learning, Nature, vo. 521, pp.436-444

  https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

References (2)

- D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio (2010), Why does unsupervised pre-training help Deep Learning?

  http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf

- Y. LeCun, L. Bottou, Y. Bengio and P. Haffner (1998), Gradient-based learning applied to document recognition, Proceedings of the IEEE

  http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

- G. Hinton and R. Salakhutdinov (2006), Reducing the dimensionality of data with neural networks, Science vol. 313

  https://www.cs.toronto.edu/~hinton/science.pdf

- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov (2014), Dropout: a simple way to prevent neural networks from overfitting

  https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

References (3)

- **A. Krizhevsky, I. Sutskever, G. Hinton (2012), ImageNet classification with deep convolutional neural networks, Proceedings of NIPS**

  https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

- **Yann LeCun and Marc Ranzato, Deep Learning Tutorial**
  http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf

- **M. D. Zeller and R. Fergus (2014), Visualizing and Understanding Convolutional Networks, Proceedings of ECCV**

  http://link.springer.com/chapter/10.1007/978-3-319-10590-1_53