



## Deep Learning.

Petr Pošík

Czech Technical University in Prague  
Faculty of Electrical Engineering  
Dept. of Cybernetics



# Deep Learning



# A brief history of Neural Networks

---

- **1940s:** Model of neuron (McCulloch, Pitts)
- **1950-60s:** Modeling brain using neural networks (Rosenblatt, Hebb, etc.)
- **1969:** Research stagnated after Minsky and Papert's book *Perceptrons*
- **1970s:** Backpropagation
- **1986:** Backpropagation popularized by Rumelhardt, Hinton, Williams
- **1990s:** Convolutional neural networks (LeCun)
- **1990s:** Recurrent neural networks (Schmidhuber)
- **2006:** Revival of deep networks, unsupervised pre-training (Hinton et al.)
- **2013-:** Huge industrial interest

## Deep Learning

---

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

---

## Successes

---

## Recurrent Nets

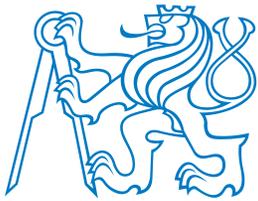
---

## Other remarks

---

## Summary

---



# What is Deep learning?

---

Conventional ML techniques:

- Limited in their ability to process natural data in their raw form.
- Successful applications required careful engineering and human expertise to extract suitable features.

## Deep Learning

- History
- **Definition**
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

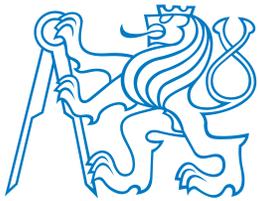
## ConvNets

## Successes

## Recurrent Nets

## Other remarks

## Summary



# What is Deep learning?

---

Conventional ML techniques:

- Limited in their ability to process natural data in their raw form.
- Successful applications required careful engineering and human expertise to extract suitable features.

**Representation learning:**

- Set of methods allowing a machine to be fed with raw data and to *automatically discover the representations* suitable for correct classification/regression/modeling.

---

## Deep Learning

- History
- **Definition**
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

## ConvNets

---

## Successes

---

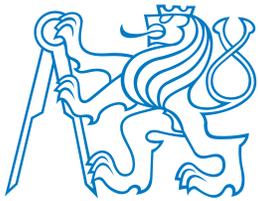
## Recurrent Nets

---

## Other remarks

---

## Summary



# What is Deep learning?

---

Conventional ML techniques:

- Limited in their ability to process natural data in their raw form.
- Successful applications required careful engineering and human expertise to extract suitable features.

## Representation learning:

- Set of methods allowing a machine to be fed with raw data and to *automatically discover the representations* suitable for correct classification/regression/modeling.

## Deep learning:

- Representation-learning methods with multiple levels of representation, with *increasing level of abstraction*.
- Compose simple, but often non-linear modules transforming the representation at one level into a representation at a higher, more abstract level.
- The layers learn to represent the inputs in a way that makes it easy to predict the target outputs.

---

### Deep Learning

- History
- **Definition**
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

### ConvNets

---

### Successes

---

### Recurrent Nets

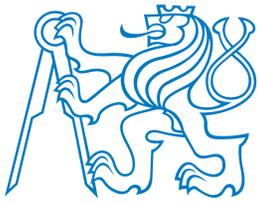
---

### Other remarks

---

### Summary

---



# Terminology

---

- **Narrow** vs **wide**: Refers to the *number of units in a layer*.
- **Shallow** vs **deep**: Refers to the *number of layers*.

## Deep Learning

- History
- Definition
- **Terminology**
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

## Successes

## Recurrent Nets

## Other remarks

## Summary



# Terminology

- **Narrow** vs **wide**: Refers to the *number of units in a layer*.
- **Shallow** vs **deep**: Refers to the *number of layers*.

## Deep Learning

- History
- Definition
- **Terminology**
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

## Successes

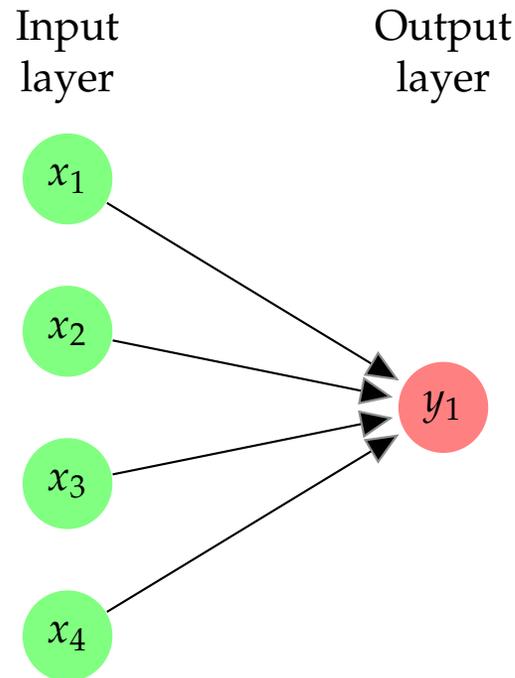
## Recurrent Nets

## Other remarks

## Summary

Making a deep architecture:

- A classifier uses the original representation:



- A classifier uses features which are derived from the original representation:
- A classifier uses features which are derived from the feature derived from the original representation:



# Terminology

- **Narrow** vs **wide**: Refers to the *number of units in a layer*.
- **Shallow** vs **deep**: Refers to the *number of layers*.

## Deep Learning

- History
- Definition
- **Terminology**
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

## Successes

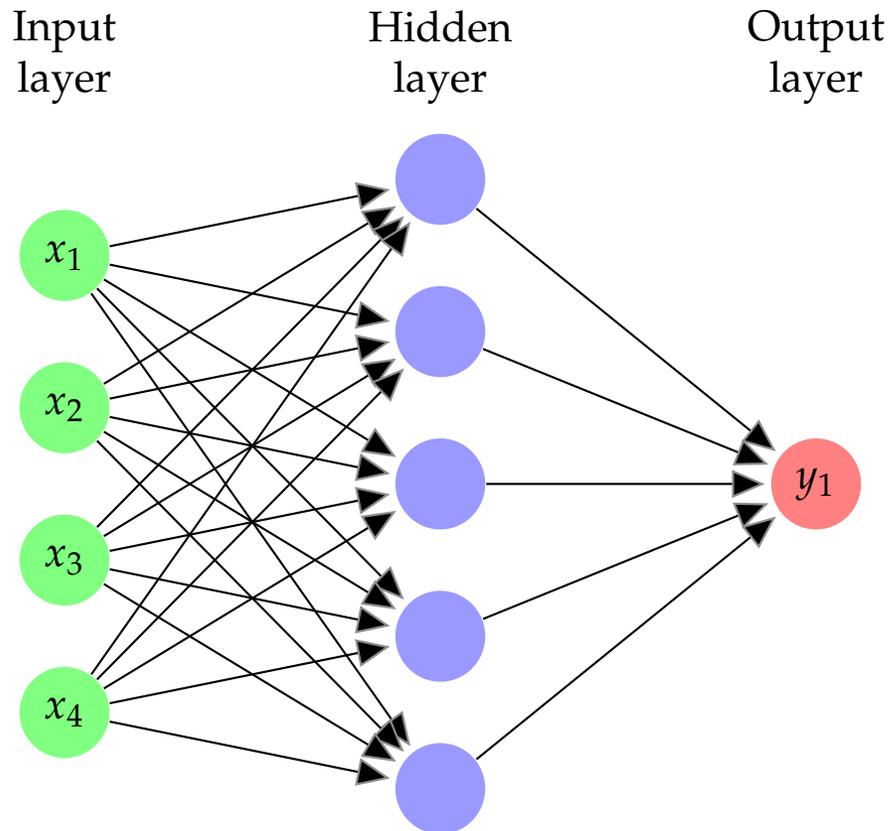
## Recurrent Nets

## Other remarks

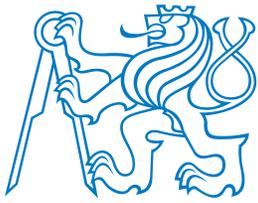
## Summary

Making a deep architecture:

- A classifier uses the original representation:
- A classifier uses features which are derived from the original representation:



- A classifier uses features which are derived from the feature derived from the original representation:



# Terminology

- **Narrow** vs **wide**: Refers to the *number of units in a layer*.
- **Shallow** vs **deep**: Refers to the *number of layers*.

## Deep Learning

- History
- Definition
- **Terminology**
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

## Successes

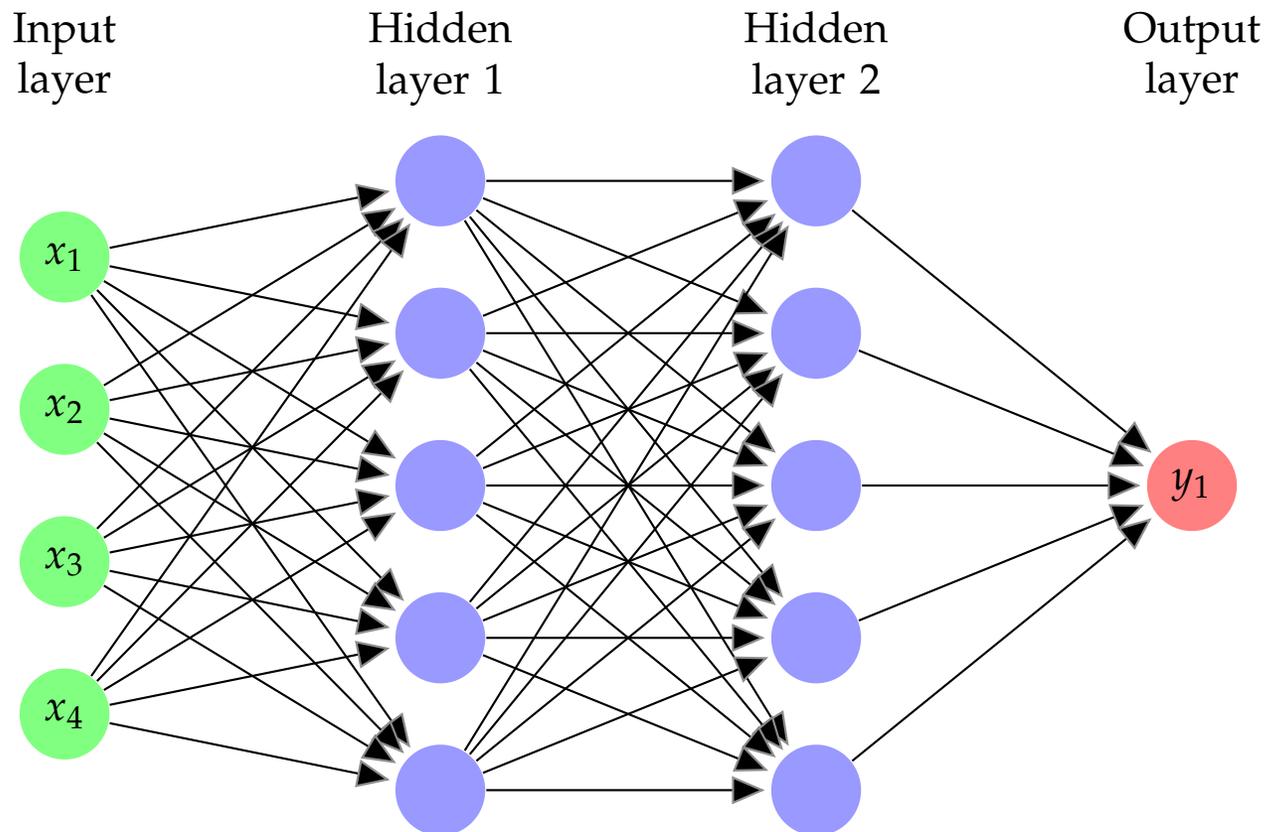
## Recurrent Nets

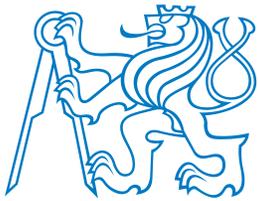
## Other remarks

## Summary

Making a deep architecture:

- A classifier uses the original representation:
- A classifier uses features which are derived from the original representation:
- A classifier uses features which are derived from the feature derived from the original representation:





# Example: Word embeddings

---

Sometimes, even shallow architectures can do surprisingly well!

## Deep Learning

- History
- Definition
- Terminology
- **Ex: Word embed.**
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

## Successes

## Recurrent Nets

## Other remarks

## Summary



## Example: Word embeddings

---

Sometimes, even shallow architectures can do surprisingly well!

Representation of text (words, sentences):

- Important for many real-world apps: search, ads recommendation, ranking, spam filtering, ...
- Local representations:
  - N-grams, 1-of-N coding, Bag of words
  - Easy to construct.
  - Large and sparse.
  - No notion of similarity (synonyms, words with similar *meaning*).

---

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

### ConvNets

---

### Successes

---

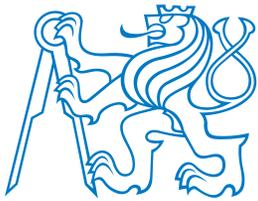
### Recurrent Nets

---

### Other remarks

---

### Summary



## Example: Word embeddings

---

Sometimes, even shallow architectures can do surprisingly well!

Representation of text (words, sentences):

- Important for many real-world apps: search, ads recommendation, ranking, spam filtering, ...
- Local representations:
  - N-grams, 1-of-N coding, Bag of words
  - Easy to construct.
  - Large and sparse.
  - No notion of similarity (synonyms, words with similar *meaning*).
- Distributed representations:
  - Vectors of real numbers in a high-dimensional continuous space (but much less dimensional than 1-of-N encoding).
  - Not clear how to meaningfully construct such a representation.
  - The size is tunable, but much smaller than that of local representations; dense.
  - Similarity well defined: synonyms should be in the same area of the space.

---

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

### ConvNets

---

### Successes

---

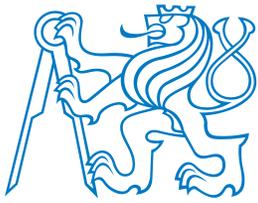
### Recurrent Nets

---

### Other remarks

---

### Summary



## Example: Word embeddings

---

Sometimes, even shallow architectures can do surprisingly well!

Representation of text (words, sentences):

- Important for many real-world apps: search, ads recommendation, ranking, spam filtering, ...
- Local representations:
  - N-grams, 1-of-N coding, Bag of words
  - Easy to construct.
  - Large and sparse.
  - No notion of similarity (synonyms, words with similar *meaning*).
- Distributed representations:
  - Vectors of real numbers in a high-dimensional continuous space (but much less dimensional than 1-of-N encoding).
  - Not clear how to meaningfully construct such a representation.
  - The size is tunable, but much smaller than that of local representations; dense.
  - Similarity well defined: synonyms should be in the same area of the space.

Assumption: *meaning* can be defined by the word context, i.e. words that surround it.

---

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

### ConvNets

---

### Successes

---

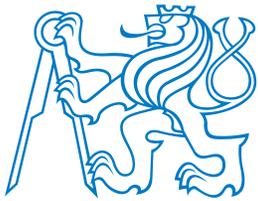
### Recurrent Nets

---

### Other remarks

---

### Summary



# Example: word2vec Architectures

## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- **Ex: w2v arch.**
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

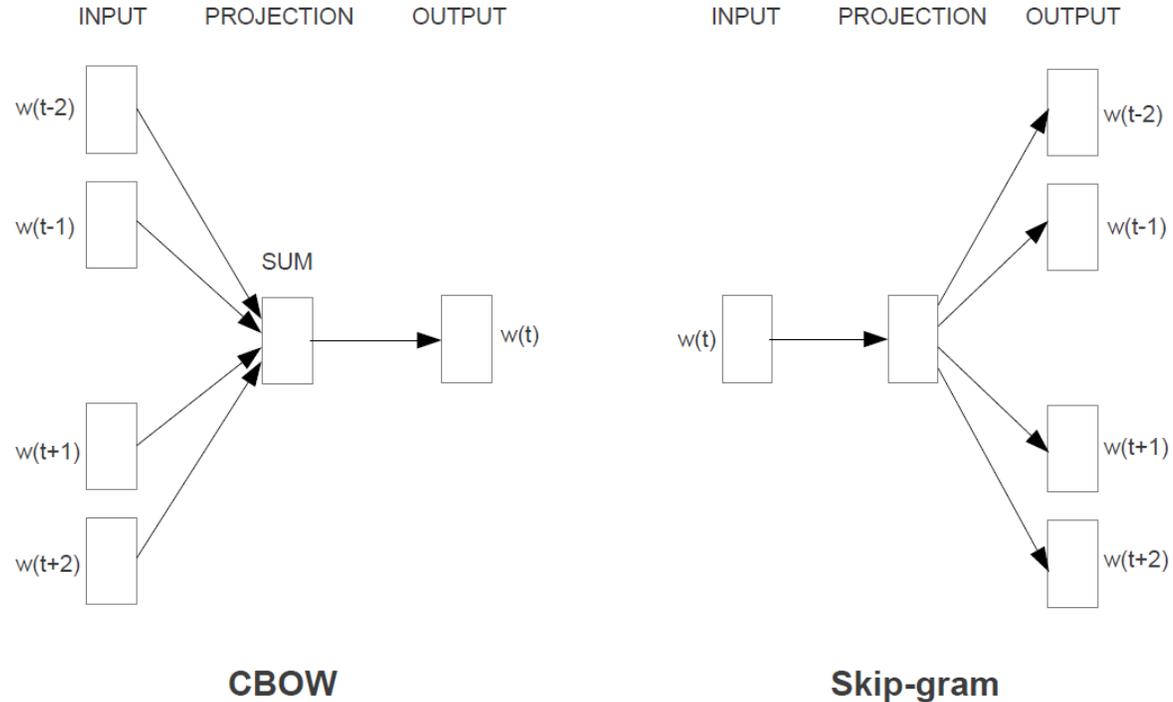
## ConvNets

## Successes

## Recurrent Nets

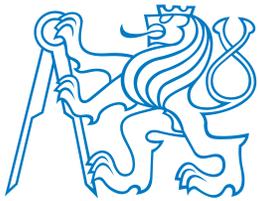
## Other remarks

## Summary



Tomas Mikolov et al.: Efficient Estimation of Word Representations in Vector Space. 2013

- **Continuous bag of words (CBOW):** Predict the current word based on the context (preceding and following words).
- **Skip-gram:** Predict the context (surrounding words) given the current word.
- The transformation of local to distributed representation is *shared among all words!*
- Trained using SGD with BP on huge data sets with billions of word n-grams, and with millions of words in the vocabulary!



## Example: Results of word2vec

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

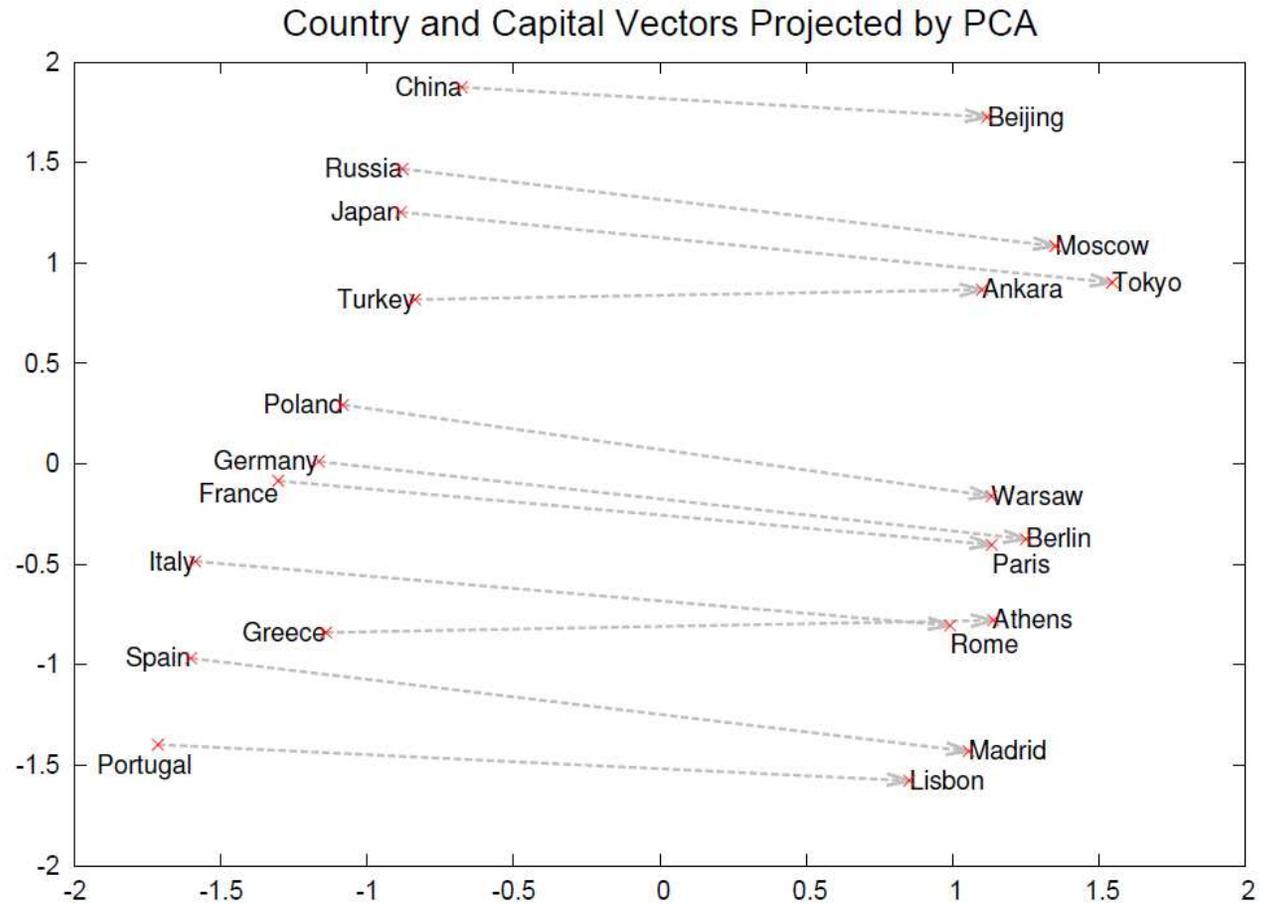
### ConvNets

### Successes

### Recurrent Nets

### Other remarks

### Summary



Tomas Mikolov et al.: Distributed Representations of Words and Phrases and their Compositionality. 2013

- Countries are found in one area, capitals in another.
- The difference vectors of countries to capitals are almost the same!
- The places roughly progress from Asia, through middle east, eastern Europe, to western Europe!



## Example: Results of word2vec

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- **Ex: w2v results**
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

### ConvNets

### Successes

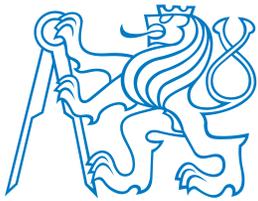
### Recurrent Nets

### Other remarks

### Summary

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Tomas Mikolov et al.: Efficient Estimation of Word Representations in Vector Space. 2013



## Example: Results of word2vec

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- **Ex: w2v results**
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

### ConvNets

### Successes

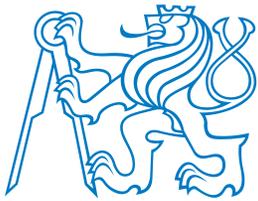
### Recurrent Nets

### Other remarks

### Summary

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Tomas Mikolov et al.: Efficient Estimation of Word Representations in Vector Space. 2013



## Example: Results of word2vec

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- **Ex: w2v results**
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

### ConvNets

### Successes

### Recurrent Nets

### Other remarks

### Summary

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Tomas Mikolov et al.: Efficient Estimation of Word Representations in Vector Space. 2013

- Trained on a lot of data!
- Turns out that similar results can be obtained by SVD factorization of (the log of) the matrix of counts of co-occurrence of words and phrases.
- Statistical learning can be much better than people expect! Sometimes even with a shallow architecture!
- Features derived by word2vec used across many big IT companies in plenty of apps (adds, search, ...)



# Why deep?

---

## Universal approximation theorem:

- A multilayer perceptron with a single hidden layer with a sufficient (but finite) number of hidden units can approximate any continuous function with arbitrary precision (under some mild assumptions on the activation functions).
- Why bother with deeper networks?

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- **Why deep?**
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

### ConvNets

### Successes

### Recurrent Nets

### Other remarks

### Summary



# Why deep?

---

## Universal approximation theorem:

- A multilayer perceptron with a single hidden layer with a sufficient (but finite) number of hidden units can approximate any continuous function with arbitrary precision (under some mild assumptions on the activation functions).
- Why bother with deeper networks?

The theorem says nothing about

- the efficiency of such a representation (maybe a NN with multiple hidden layers would represent the function with much smaller number of units), and
- the efficiency of learning such a wide and shallow network from data.
- (Although in case of word2vec, so far no deep architecture was shown to provide better results.)

---

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- **Why deep?**
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

### ConvNets

---

### Successes

---

### Recurrent Nets

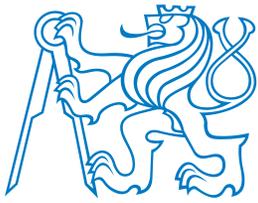
---

### Other remarks

---

### Summary

---



# Why deep?

---

## Universal approximation theorem:

- A multilayer perceptron with a single hidden layer with a sufficient (but finite) number of hidden units can approximate any continuous function with arbitrary precision (under some mild assumptions on the activation functions).
- Why bother with deeper networks?

The theorem says nothing about

- the efficiency of such a representation (maybe a NN with multiple hidden layers would represent the function with much smaller number of units), and
- the efficiency of learning such a wide and shallow network from data.
- (Although in case of word2vec, so far no deep architecture was shown to provide better results.)

When solving an algorithmic problem, we usually

- start by solving sub-problems, and then
- gradually integrate the solutions.

We build the solution through multiple layers of abstraction.

---

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- **Why deep?**
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

### ConvNets

---

### Successes

---

### Recurrent Nets

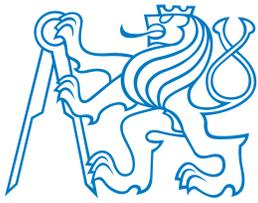
---

### Other remarks

---

### Summary

---



# A new idea?

---

Deep architectures:

- Not a new idea. Maybe 50 years old.
- Hard/impossible to train until recently.

## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- **A new idea?**
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

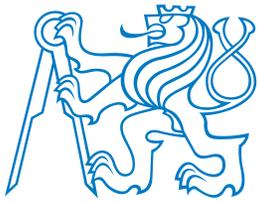
## ConvNets

## Successes

## Recurrent Nets

## Other remarks

## Summary



# A new idea?

---

Deep architectures:

- Not a new idea. Maybe 50 years old.
- Hard/impossible to train until recently.

## What makes deep networks hard to train?

- Not an easy answer, subject of ongoing research.
- Instabilities associated to gradient-based learning (vanishing/exploding gradients).
- The choice of activation function.
- Weights initialization.
- Details of implementation of gradient descent (momentum schedule).
- The choice of network architecture and hyper-parameters.

---

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- **A new idea?**
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

### ConvNets

---

### Successes

---

### Recurrent Nets

---

### Other remarks

---

### Summary

---



# A new idea?

---

Deep architectures:

- Not a new idea. Maybe 50 years old.
- Hard/impossible to train until recently.

## What makes deep networks hard to train?

- Not an easy answer, subject of ongoing research.
- Instabilities associated to gradient-based learning (vanishing/exploding gradients).
- The choice of activation function.
- Weights initialization.
- Details of implementation of gradient descent (momentum schedule).
- The choice of network architecture and hyper-parameters.

## Vanishing gradient

- Backpropagation is just a clever use of chain rule.
- During error backpropagation, the multiplication by the derivative of the activation function is used many times.
- The derivative of the “standard” sigmoidal function is from  $\langle 0, 0.25 \rangle$ .
- The size of error diminishes when propagating towards the input layer, and quickly becomes very small.
- The learning in the initial layers of deep neural network is very slow; these layers learn virtually nothing (unless trained for a very loooooong time).
- **Exploding gradient** is exactly the opposite problem.

---

### Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- **A new idea?**
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

---

### ConvNets

---

### Successes

---

### Recurrent Nets

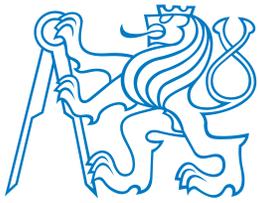
---

### Other remarks

---

### Summary

---



# Boom of Deep Nets

---

Factors of the deep nets boom in the last decade:

- **Big data era.** Building models with millions of parameters started to make sense.
- **Fast GPUs.** Training such large models started to be feasible.
- **Weight sharing.** Not that many parameters are actually needed.
- **Unsupervised pre-training.** For areas where not enough data is available.
- **Data augmentation.** Artificially created training examples by deforming/moving/rotating the available ones.
- **Regularization.** Especially using drop-out.
- **ReLU.** Usually makes backpropagation faster.

## Deep Learning

---

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- **Boom of Deep Nets**
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

---

## Successes

---

## Recurrent Nets

---

## Other remarks

---

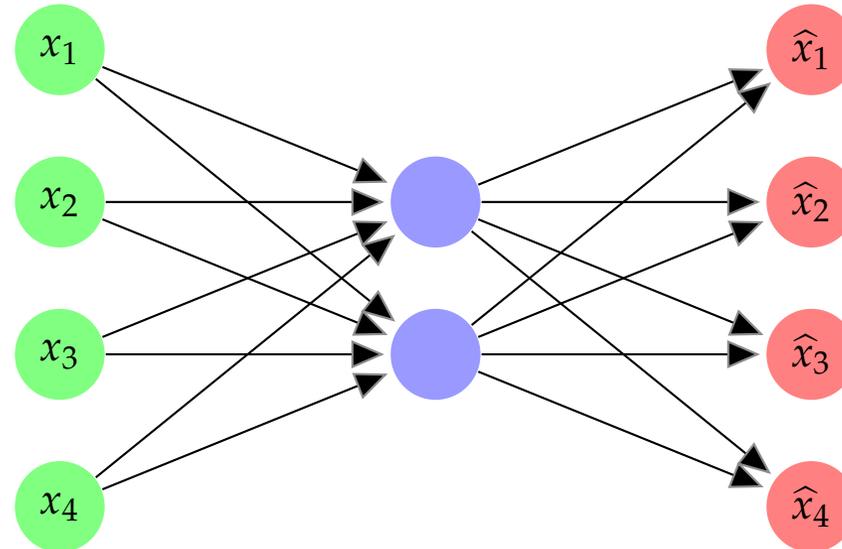
## Summary

---



# Autoencoders

- Unsupervised learning. (Or, rather “self-supervised”?)
- Networks trained to map input vector on the same output vector.
- Reconstruction error is minimized.
- Performs dimensionality reduction.
- Hidden layer contains less neurons than input and output layers.
- Hidden neurons contain compressed representation of the input examples.



## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

## Successes

## Recurrent Nets

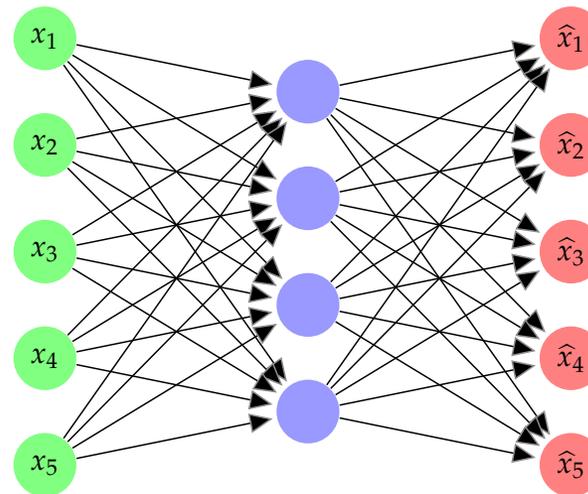
## Other remarks

## Summary



# Stacked autoencoders

1. Train the first hidden layer of an autoencoder.
2. Split the first hidden layer to encoder and decoder, keep them fixed and train the second hidden layer.
3. Split the second hidden layer to encoder and decoder, keep them fixed and train the third hidden layer.
4. ...



## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- **Stacked autoenc.**
- Pre-training

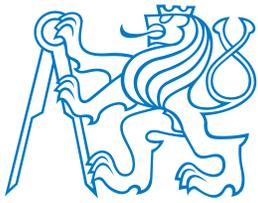
## ConvNets

## Successes

## Recurrent Nets

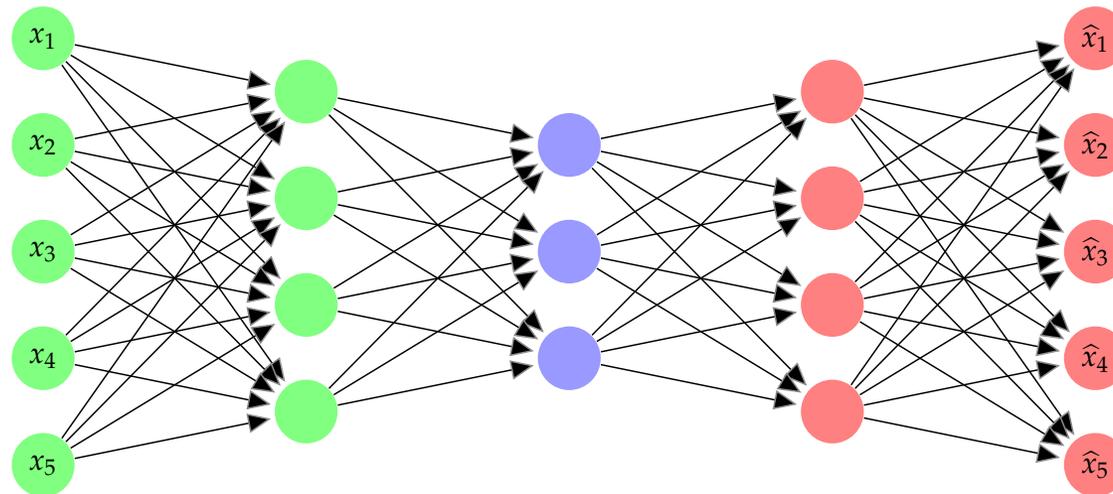
## Other remarks

## Summary



# Stacked autoencoders

1. Train the first hidden layer of an autoencoder.
2. Split the first hidden layer to encoder and decoder, keep them fixed and train the second hidden layer.
3. Split the second hidden layer to encoder and decoder, keep them fixed and train the third hidden layer.
4. ...



## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- **Stacked autoenc.**
- Pre-training

## ConvNets

## Successes

## Recurrent Nets

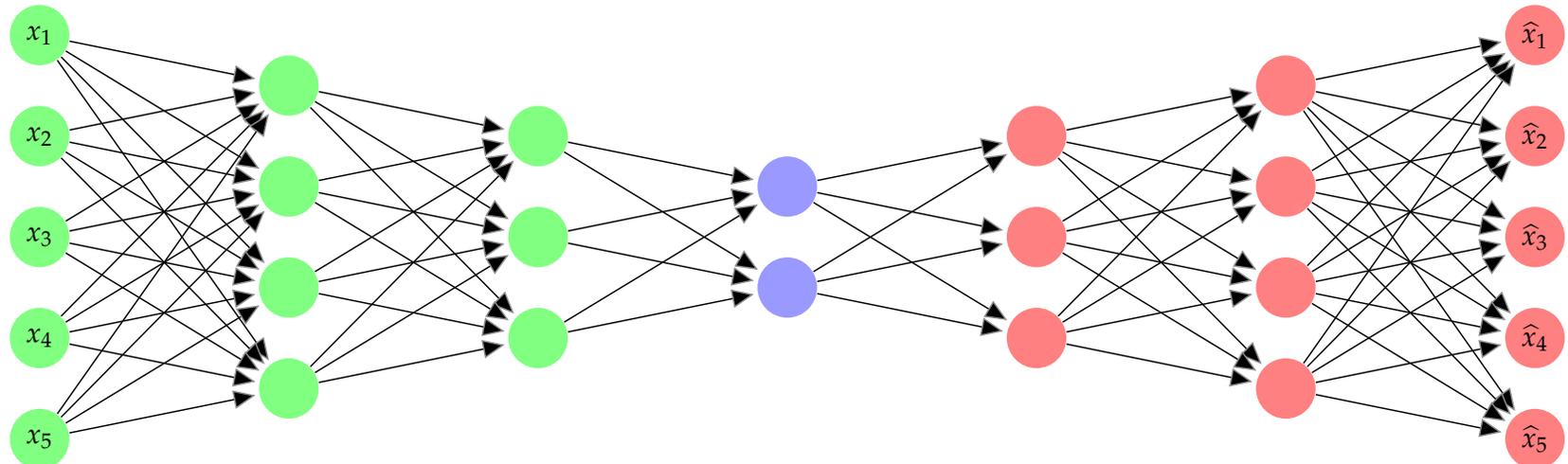
## Other remarks

## Summary



# Stacked autoencoders

1. Train the first hidden layer of an autoencoder.
2. Split the first hidden layer to encoder and decoder, keep them fixed and train the second hidden layer.
3. Split the second hidden layer to encoder and decoder, keep them fixed and train the third hidden layer.
4. ...



## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- **Stacked autoenc.**
- Pre-training

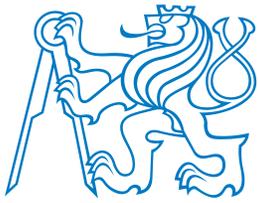
## ConvNets

## Successes

## Recurrent Nets

## Other remarks

## Summary



# Stacked autoencoders

## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- **Stacked autoenc.**
- Pre-training

## ConvNets

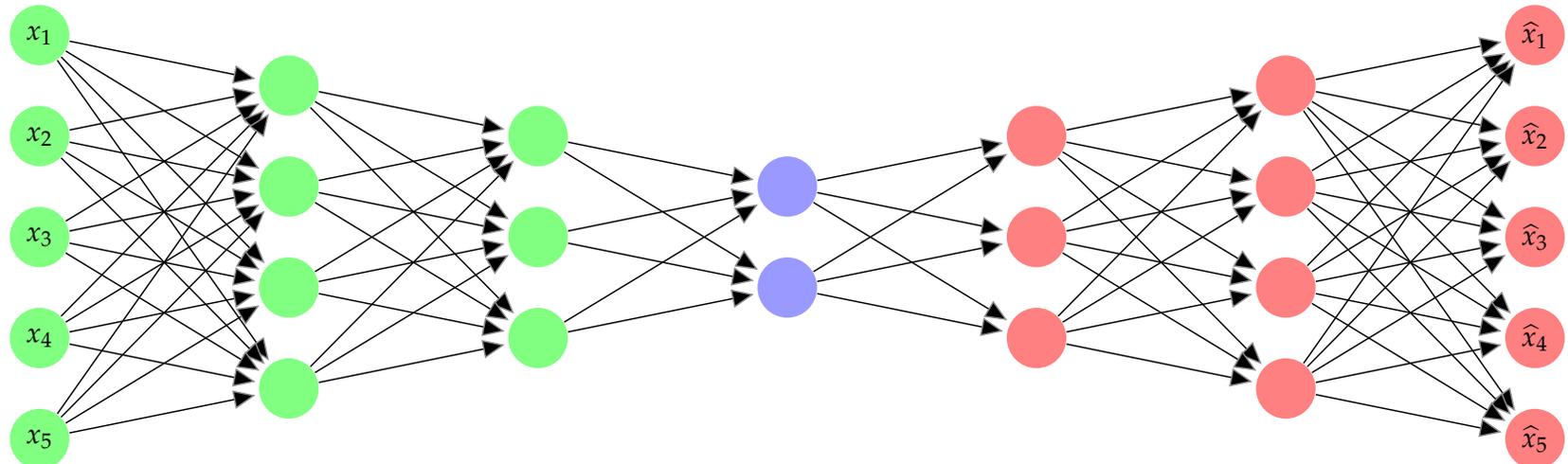
## Successes

## Recurrent Nets

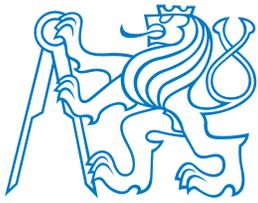
## Other remarks

## Summary

1. Train the first hidden layer of an autoencoder.
2. Split the first hidden layer to encoder and decoder, keep them fixed and train the second hidden layer.
3. Split the second hidden layer to encoder and decoder, keep them fixed and train the third hidden layer.
4. ...

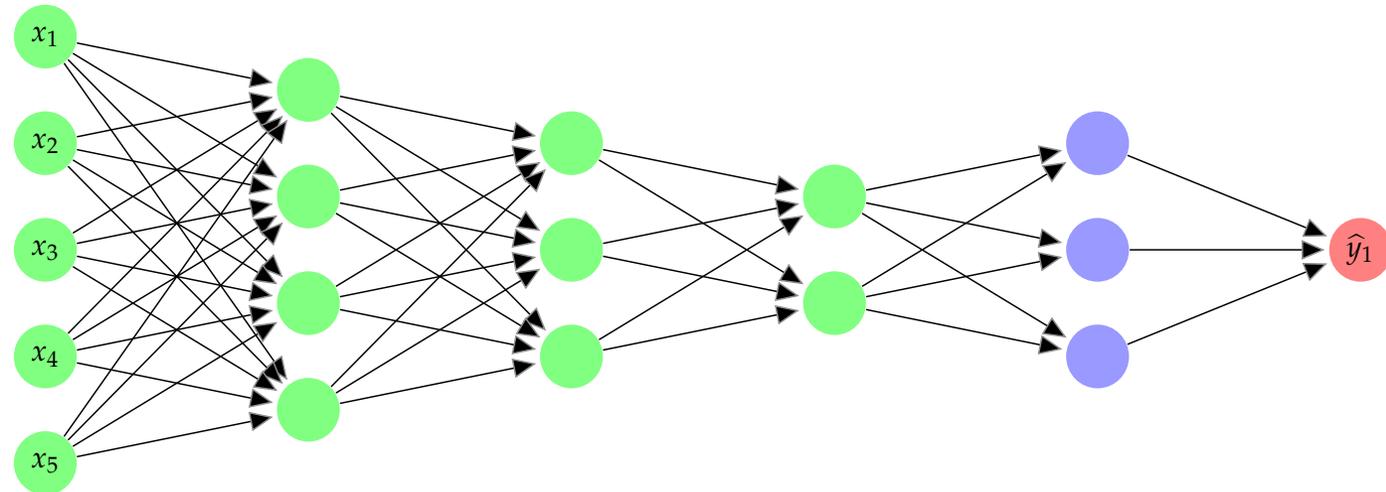


- Always training only a single layer - building a deep architecture one step at a time.
- Vanishing gradients are not an issue.



# Deep NN pre-training

- Use stacked autoencoders to pre-train the first layers of the network.
- Use the encoding part of the network, and attach a classifier for your particular task.
- Train the attached part of the network (and fine tune the encoder weights) by backpropagation.



## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- Pre-training

## ConvNets

## Successes

## Recurrent Nets

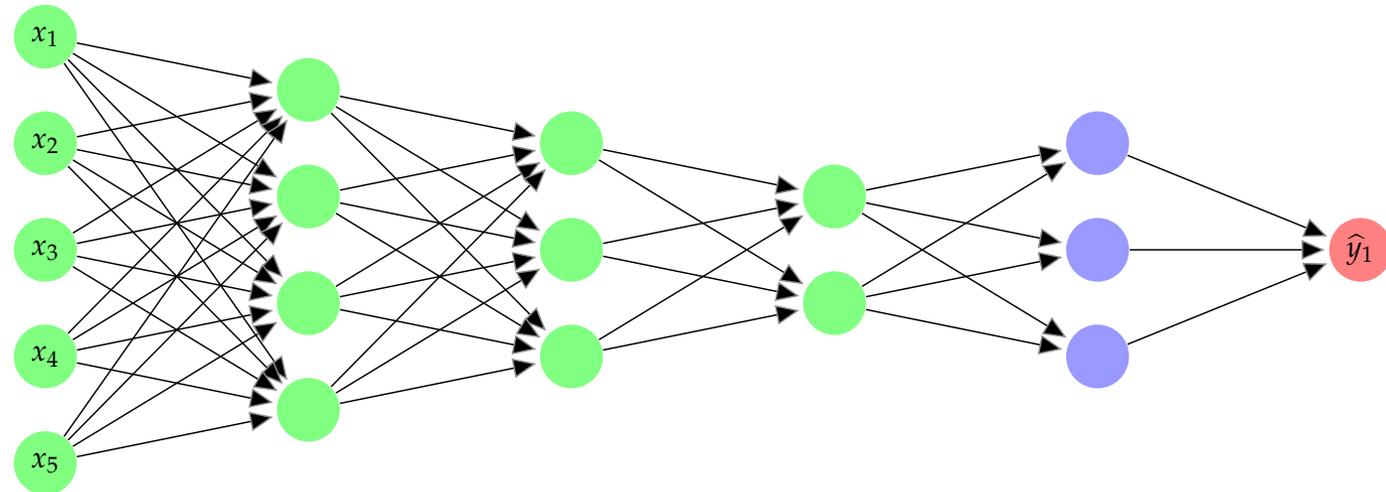
## Other remarks

## Summary



# Deep NN pre-training

- Use stacked autoencoders to pre-train the first layers of the network.
- Use the encoding part of the network, and attach a classifier for your particular task.
- Train the attached part of the network (and fine tune the encoder weights) by backpropagation.



An example of **Transfer learning**:

- Using certain part of a model trained for one task as the basis of a model performing another, but related task.

## Deep Learning

- History
- Definition
- Terminology
- Ex: Word embed.
- Ex: w2v arch.
- Ex: w2v results
- Why deep?
- A new idea?
- Boom of Deep Nets
- Autoencoders
- Stacked autoenc.
- **Pre-training**

## ConvNets

## Successes

## Recurrent Nets

## Other remarks

## Summary



# Convolutional Networks



# Image processing by NN

---

Fully-connected architecture:

- Input layer neurons are directly connected to the image pixels.
- Let's have a hidden layer with approx. the same size as the input layer *fully connected* to the input layer:
  - Small image size:  $28 \times 28$  pixels.
  - Number of input neurons:

Deep Learning

ConvNets

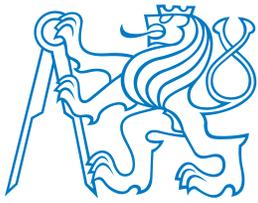
- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

Other remarks

Summary



# Image processing by NN

---

Fully-connected architecture:

- Input layer neurons are directly connected to the image pixels.
- Let's have a hidden layer with approx. the same size as the input layer *fully connected* to the input layer:
  - Small image size:  $28 \times 28$  pixels.
  - Number of input neurons:  $28^2$
  - Number of hidden layer neurons:

Deep Learning

ConvNets

- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

Other remarks

Summary



# Image processing by NN

---

Fully-connected architecture:

- Input layer neurons are directly connected to the image pixels.
- Let's have a hidden layer with approx. the same size as the input layer *fully connected* to the input layer:
  - Small image size:  $28 \times 28$  pixels.
  - Number of input neurons:  $28^2$
  - Number of hidden layer neurons:  $28^2$
  - Number of weights:

Deep Learning

ConvNets

- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

Other remarks

Summary



# Image processing by NN

---

Fully-connected architecture:

- Input layer neurons are directly connected to the image pixels.
- Let's have a hidden layer with approx. the same size as the input layer *fully connected* to the input layer:
  - Small image size:  $28 \times 28$  pixels.
  - Number of input neurons:  $28^2$
  - Number of hidden layer neurons:  $28^2$
  - Number of weights:  $\underbrace{28^2}_{\text{Outputs}} (\underbrace{28^2}_{\text{Inputs}} + \underbrace{1}_{\text{Biases}}) \approx 600 \text{ thousands}$
  - Repeat several times, if you want a deep architecture.
- *Too many parameters to learn!*
- Ignores spatial structure of the images:
  - Treats the input pixels far/close to each other in exactly the same way:
  - Sensitive to movements of the object in the image.

---

Deep Learning

---

ConvNets

- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

---

Successes

---

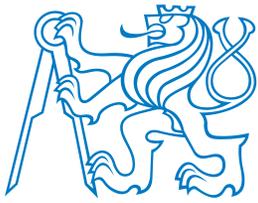
Recurrent Nets

---

Other remarks

---

Summary



# Image processing by NN

---

Fully-connected architecture:

- Input layer neurons are directly connected to the image pixels.
- Let's have a hidden layer with approx. the same size as the input layer *fully connected* to the input layer:
  - Small image size:  $28 \times 28$  pixels.
  - Number of input neurons:  $28^2$
  - Number of hidden layer neurons:  $28^2$
  - Number of weights:  $\underbrace{28^2}_{\text{Outputs}} (\underbrace{28^2}_{\text{Inputs}} + \underbrace{1}_{\text{Biases}}) \approx 600 \text{ thousands}$
  - Repeat several times, if you want a deep architecture.
- *Too many parameters to learn!*
- Ignores spatial structure of the images:
  - Treats the input pixels far/close to each other in exactly the same way:
  - Sensitive to movements of the object in the image.

**Convolutional networks** solve these issues by

- local receptive fields,
- shared weights, and
- pooling.

Deep Learning

---

ConvNets

---

- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

---

Recurrent Nets

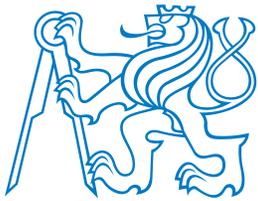
---

Other remarks

---

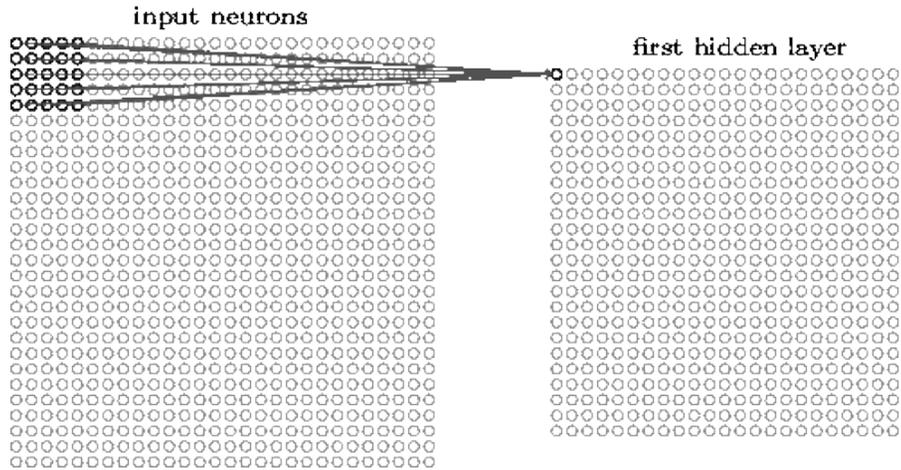
Summary

---



# Local receptive fields

- Each hidden unit is connected to only a few inputs localized in a small window.



Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

- This window is the **local receptive field**.
- The window then “slides” across the entire image.
  - Each of its position corresponds to a hidden neuron.
  - **Stride length:** The number of pixels by which the window moves in each step.
- Number of weights:

Deep Learning

ConvNets

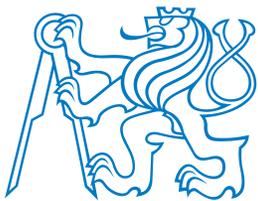
- Fully connected
- **Local reception**
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

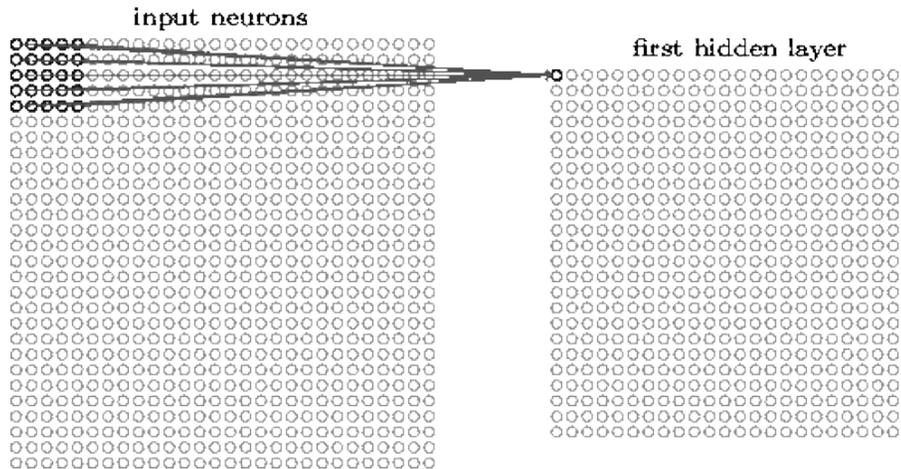
Other remarks

Summary



# Local receptive fields

- Each hidden unit is connected to only a few inputs localized in a small window.



Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

- This window is the **local receptive field**.
- The window then “slides” across the entire image.
  - Each of its position corresponds to a hidden neuron.
  - **Stride length:** The number of pixels by which the window moves in each step.
- Number of weights:  $\underbrace{24^2}_{\text{Outputs}} \left( \underbrace{5^2}_{\text{Inputs}} + \underbrace{1}_{\text{Biases}} \right) \approx 15 \text{ thousands}$

Deep Learning

ConvNets

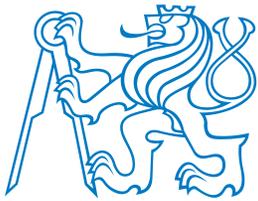
- Fully connected
- **Local reception**
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

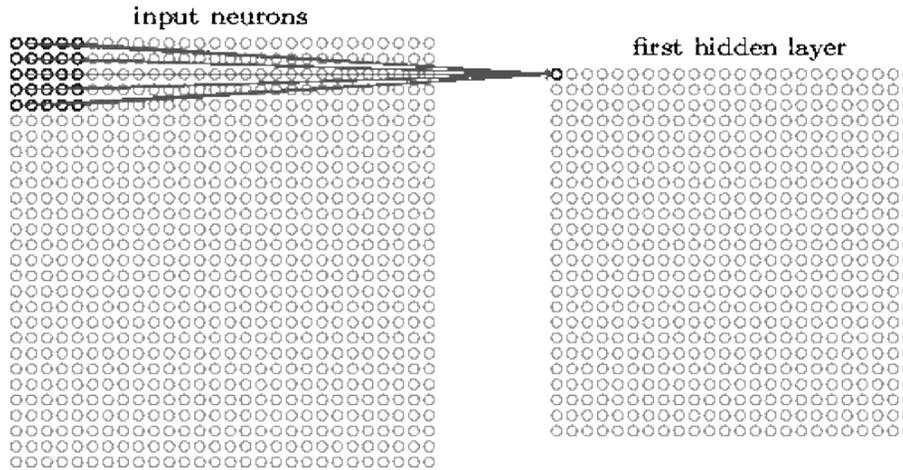
Other remarks

Summary



# Local receptive fields

- Each hidden unit is connected to only a few inputs localized in a small window.



Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

- This window is the **local receptive field**.
- The window then “slides” across the entire image.
  - Each of its position corresponds to a hidden neuron.
  - **Stride length:** The number of pixels by which the window moves in each step.
- Number of weights:  $\underbrace{24^2}_{\text{Outputs}} \left( \underbrace{5^2}_{\text{Inputs}} + \underbrace{1}_{\text{Biases}} \right) \approx 15 \text{ thousands}$
- **Multiple input channels:** e.g., in case of *color image*, we have 3 intensity images (for colors R, G, B).  
Number of weights:

Deep Learning

ConvNets

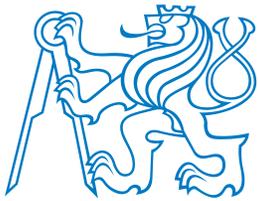
- Fully connected
- **Local reception**
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

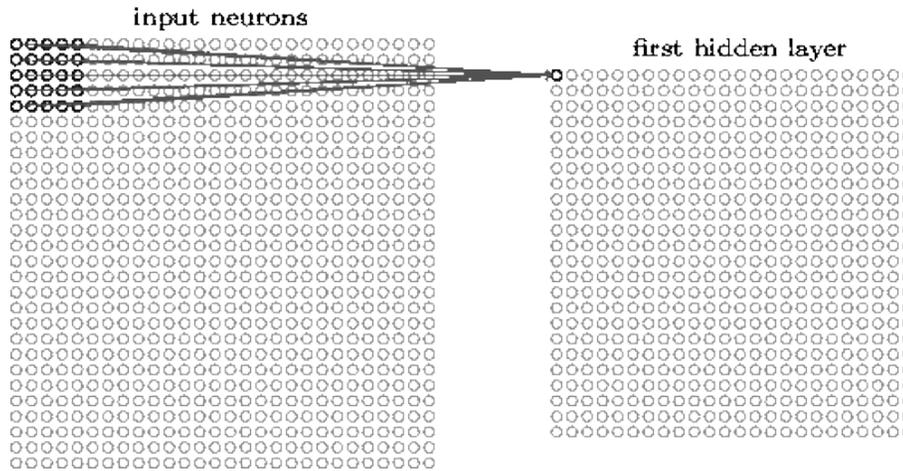
Other remarks

Summary



# Local receptive fields

- Each hidden unit is connected to only a few inputs localized in a small window.



Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

- This window is the **local receptive field**.
- The window then “slides” across the entire image.
  - Each of its position corresponds to a hidden neuron.
  - **Stride length:** The number of pixels by which the window moves in each step.

- Number of weights:  $\underbrace{24^2}_{\text{Outputs}} \left( \underbrace{5^2}_{\text{Inputs}} + \underbrace{1}_{\text{Biases}} \right) \approx 15 \text{ thousands}$

- **Multiple input channels:** e.g., in case of *color image*, we have 3 intensity images (for colors R, G, B).

Number of weights:  $\underbrace{24^2}_{\text{Outputs}} \left( \underbrace{3 \cdot 5^2}_{\text{Inputs}} + \underbrace{1}_{\text{Biases}} \right) \approx 45 \text{ thousands}$

Deep Learning

ConvNets

- Fully connected
- **Local reception**
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

Other remarks

Summary



# Weight sharing

---

- Each hidden neuron has bias and  $5 \times 5$  weights.
- All hidden neurons use *the same weights and bias*, they define a *filter*!
- The output of the hidden neuron at  $(r, c)$  is

$$z_{r,c} = g \left( b + \sum_{j=0}^4 \sum_{k=0}^4 w_{j,k} x_{r+j,c+k} \right),$$

where  $w_{j,k}$  are the shared weights,  $b$  is the shared bias, and  $g$  is the activation function (sigmoid, ReLU, ...).

- The sum is closely related to the operation of **convolution**, hence the name.

Deep Learning

---

ConvNets

---

- Fully connected
- Local reception
- **Weight sharing**
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

---

Recurrent Nets

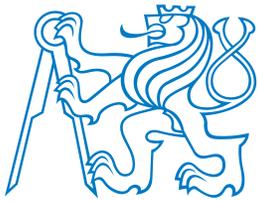
---

Other remarks

---

Summary

---



# Weight sharing

Deep Learning

ConvNets

- Fully connected
- Local reception
- **Weight sharing**
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

Other remarks

Summary

- Each hidden neuron has bias and  $5 \times 5$  weights.
- All hidden neurons use *the same weights and bias*, they define a *filter*!
- The output of the hidden neuron at  $(r, c)$  is

$$z_{r,c} = g \left( b + \sum_{j=0}^4 \sum_{k=0}^4 w_{j,k} x_{r+j,c+k} \right),$$

where  $w_{j,k}$  are the shared weights,  $b$  is the shared bias, and  $g$  is the activation function (sigmoid, ReLU, ...).

- The sum is closely related to the operation of **convolution**, hence the name.
- In case of *multiple input channels*:

$$z_{r,c} = g \left( b + \sum_{i=0}^2 \sum_{j=0}^4 \sum_{k=0}^4 w_{i,j,k} x_{i,r+j,c+k} \right).$$

- Number of parameters:



# Weight sharing

Deep Learning

ConvNets

- Fully connected
- Local reception
- **Weight sharing**
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

Other remarks

Summary

- Each hidden neuron has bias and  $5 \times 5$  weights.
- All hidden neurons use *the same weights and bias*, they define a *filter*!
- The output of the hidden neuron at  $(r, c)$  is

$$z_{r,c} = g \left( b + \sum_{j=0}^4 \sum_{k=0}^4 w_{j,k} x_{r+j,c+k} \right),$$

where  $w_{j,k}$  are the shared weights,  $b$  is the shared bias, and  $g$  is the activation function (sigmoid, ReLU, ...).

- The sum is closely related to the operation of **convolution**, hence the name.
- In case of *multiple input channels*:

$$z_{r,c} = g \left( b + \sum_{i=0}^2 \sum_{j=0}^4 \sum_{k=0}^4 w_{i,j,k} x_{i,r+j,c+k} \right).$$

- Number of parameters:  $(\underbrace{3 \cdot 5^2}_{\text{Inputs}} + \underbrace{1}_{\text{Bias}}) = 76$
- All the neurons in the hidden layer detect exactly *the same feature*, just *at different locations* in the input image.



# Convolutional layer

---

We know how to

- turn the input image represented as a volume ( $3 \text{ channels} \times \text{width} \times \text{height}$ ) into
- a single feature map (how much is a feature expressed on all places of the image)
- using a single filter.

Deep Learning

ConvNets

- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

Other remarks

Summary



# Convolutional layer

We know how to

- turn the input image represented as a volume (3 channels  $\times$  width  $\times$  height) into
- a single feature map (how much is a feature expressed on all places of the image)
- using a single filter.

## Convolutional layer:

- For a reliable image recognition we need more than one feature maps using *many different filters* (tens, hundreds, ...).
- “It processes volume into volume”.

Deep Learning

ConvNets

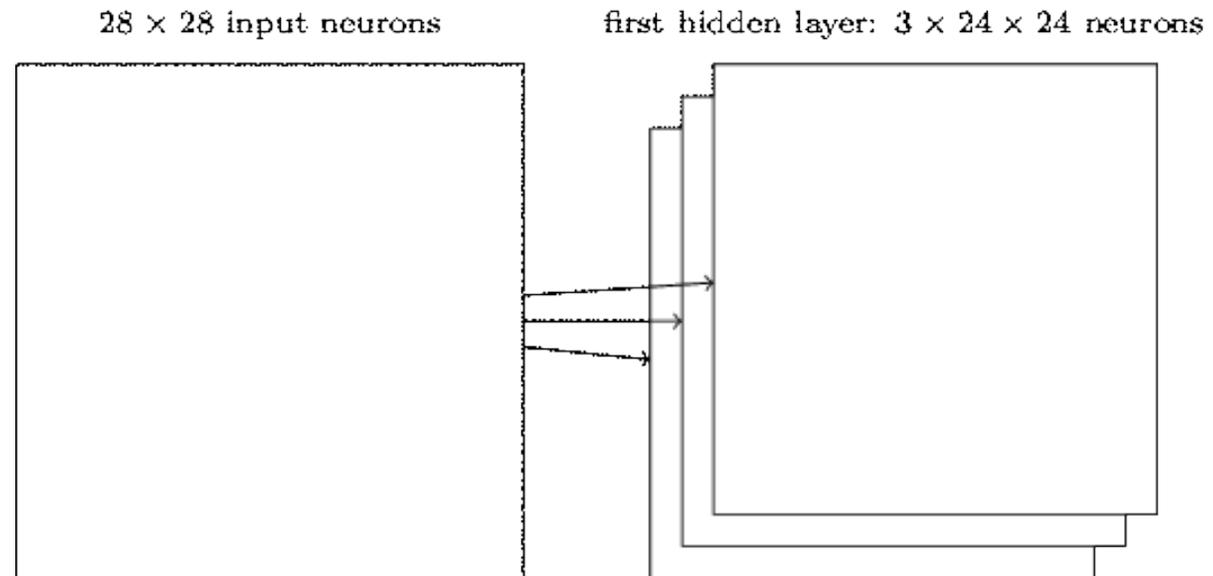
- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

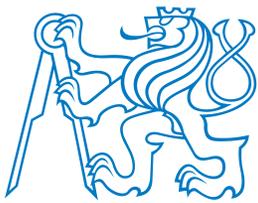
Recurrent Nets

Other remarks

Summary



Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015



# Filters

An example of 20 filters for the MNIST number database:

Deep Learning

ConvNets

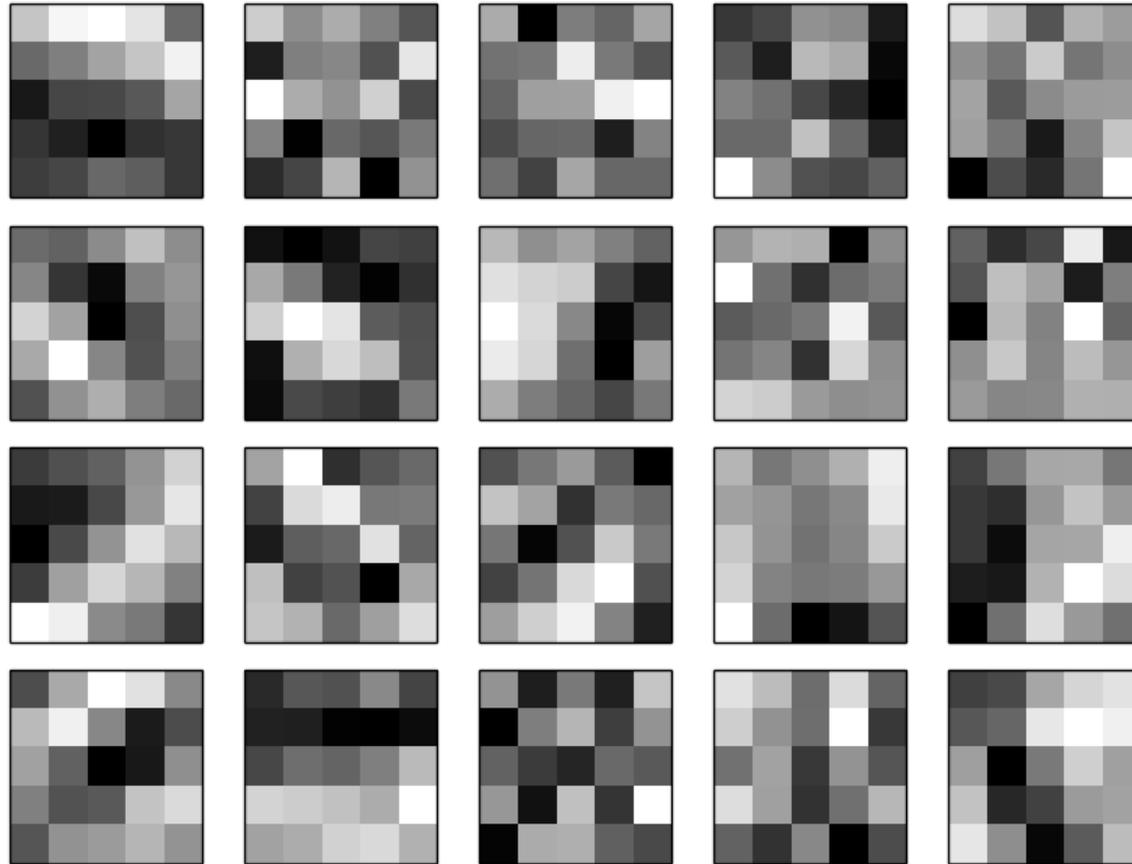
- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- **Filters**
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

Other remarks

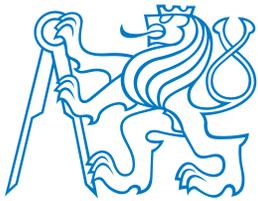
Summary



Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

These filters were

- not hand-crafted,
- but automatically trained!



# Pooling

Pooling layers:

- Usually used immediately after convolutional layers.
- They simplify the information in the output of the convolutional layer by creating a *condensed feature map*.
- **Max-pooling:**
  - Each unit in pooling layer summarizes a region of, say,  $2 \times 2$  neurons in the previous layer by taking the maximum of the entries.
  - It reduces the size of the feature map by a factor of  $2 \times 2 = 4$ .

Deep Learning

ConvNets

- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- **Pooling**
- Complete ConvNet

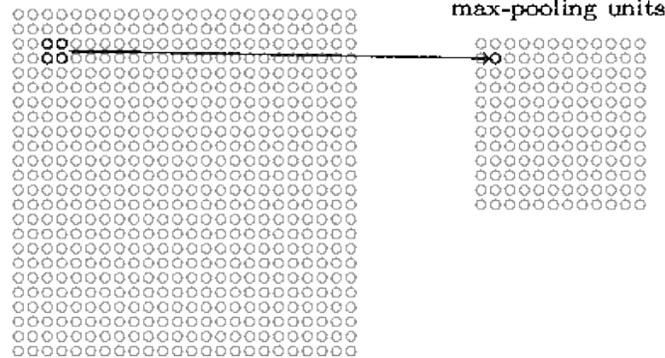
Successes

Recurrent Nets

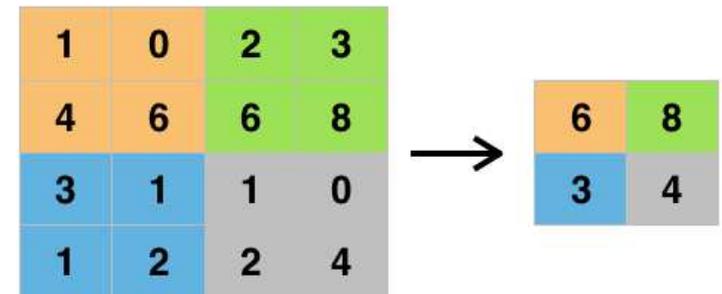
Other remarks

Summary

hidden neurons (output from feature map)

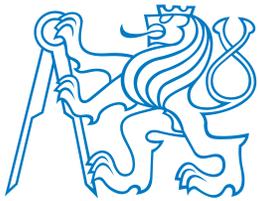


Single depth slice



Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

- Pooling is usually applied to each of the channels separately.
- Other types of pooling exist: L2 pooling, average pooling, ...



# A complete ConvNet

An example of a complete ConvNet applied to MNIST classification:

- A classification of  $28 \times 28$  grey-scale bitmap images into 10 classes (numbers 0 - 9).

Deep Learning

ConvNets

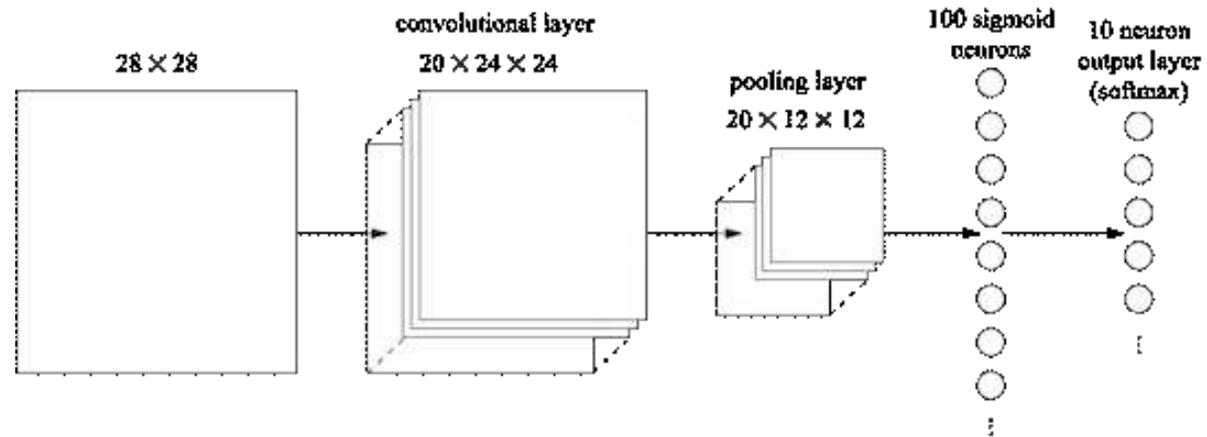
- Fully connected
- Local reception
- Weight sharing
- Convolutional layer
- Filters
- Pooling
- Complete ConvNet

Successes

Recurrent Nets

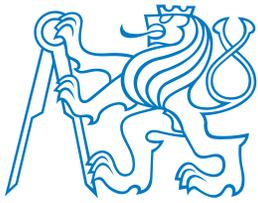
Other remarks

Summary



Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

- The input  $28 \times 28$  layer encodes pixel intensities for MNIST images.
- The first hidden layer is convolutional with  $5 \times 5$  receptive field and 10 filters resulting in  $20 \times 24 \times 24$  hidden feature neurons.
- The second hidden layer is max-pooling with  $2 \times 2$  regions; the result is  $20 \times 12 \times 12$  hidden feature neurons.
- These are *fully connected* to the final "classifier" with 2 layers of 100 and 10 output neurons (because the MNIST dataset contains 10 classes).
- You can stack more convolutional and pooling layers one after another!
- The whole network is trained by *backpropagation*.



## Recent successes of Deep ConvNets



# ImageNet Dataset

Deep Learning

ConvNets

Successes

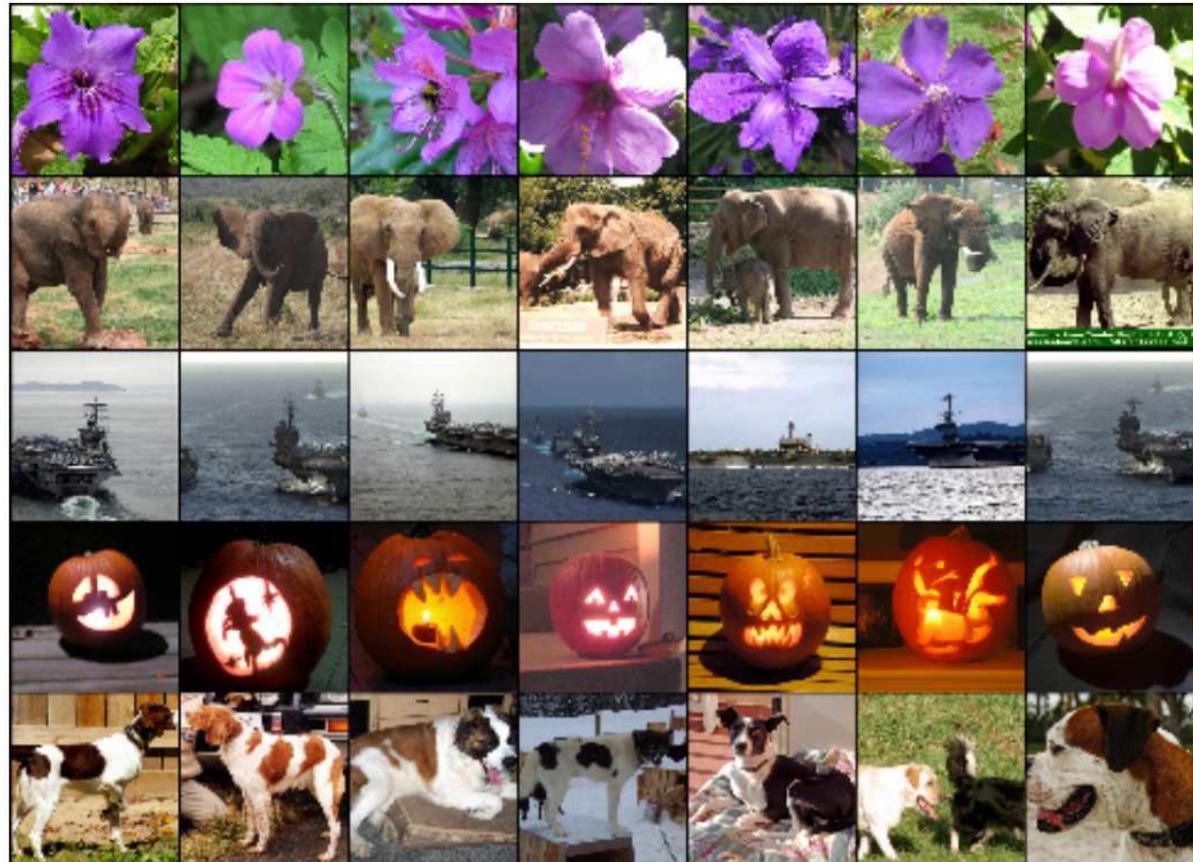
- ImageNet Dataset
- AlexNet results
- AlexNet
- AlexNet Successors

Recurrent Nets

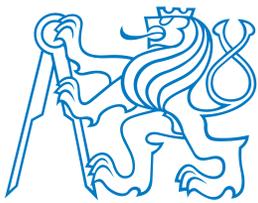
Other remarks

Summary

- High-resolution color images: 15M images, 22k classes
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) uses subset of ImageNet: 1.3M training, 50k validation, 100k testing samples, 1000 classes



- Some images contain more than 1 object.
- **Top 5:** an algorithm is considered correct if the actual ImageNet classification was among the 5 classifications the algorithm considered most likely.

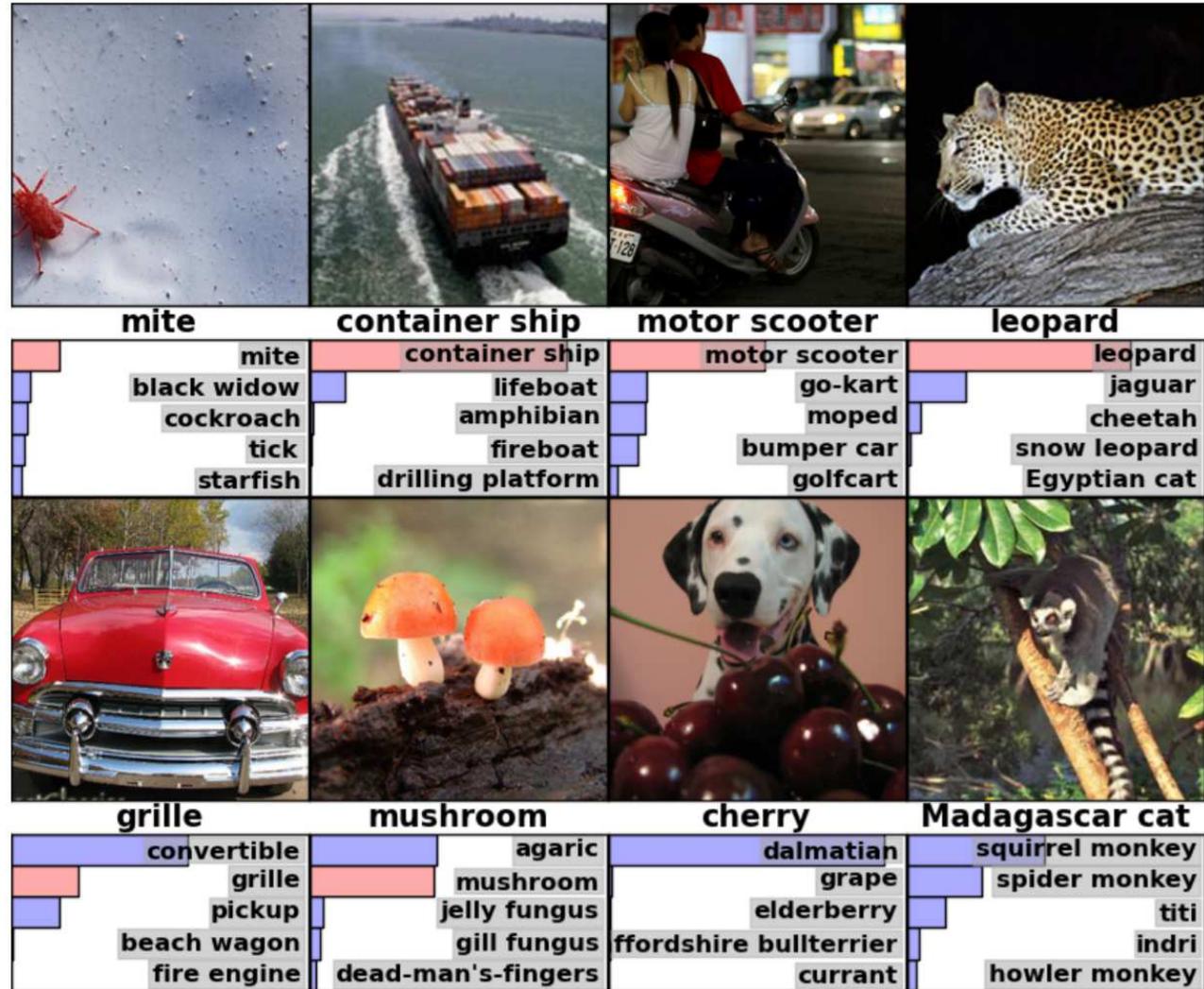


# AlexNet results

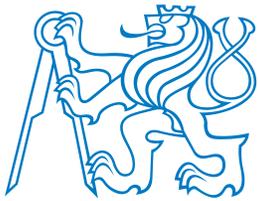
AlexNet: Breakthrough of ConvNets in computer vision!

- Top-5 error rate of AlexNet: 15.3 %, the second best entry: 26.2 %

- Deep Learning
- ConvNets
- Successes
  - ImageNet Dataset
  - AlexNet results
  - AlexNet
  - AlexNet Successors
- Recurrent Nets
- Other remarks
- Summary

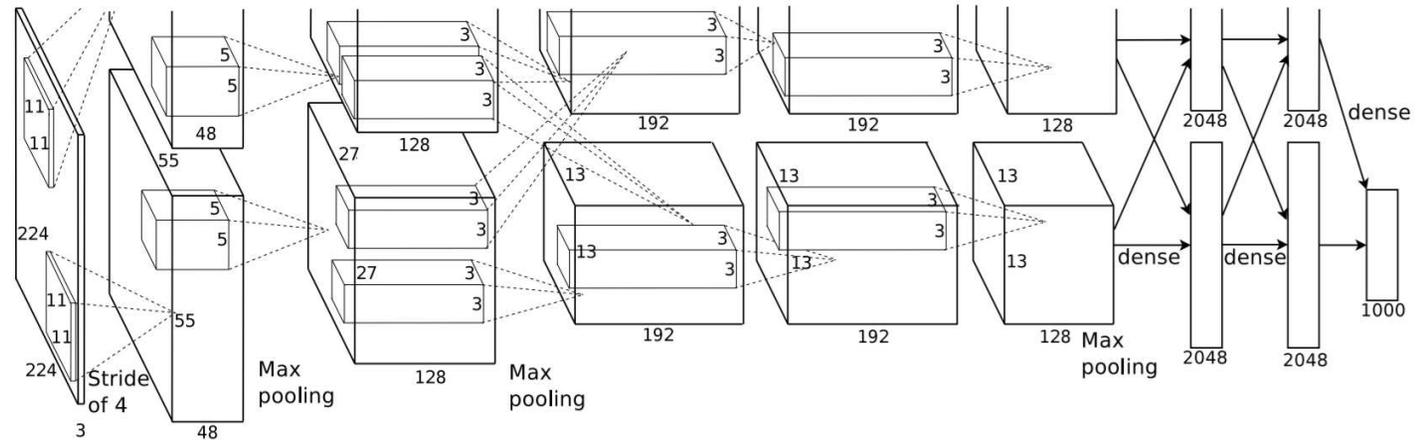


Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012



# AlexNet

Structure:



Deep Learning

ConvNets

Successes

- ImageNet Dataset
- AlexNet results
- **AlexNet**
- AlexNet Successors

Recurrent Nets

Other remarks

Summary



# AlexNet

Structure:

Deep Learning

ConvNets

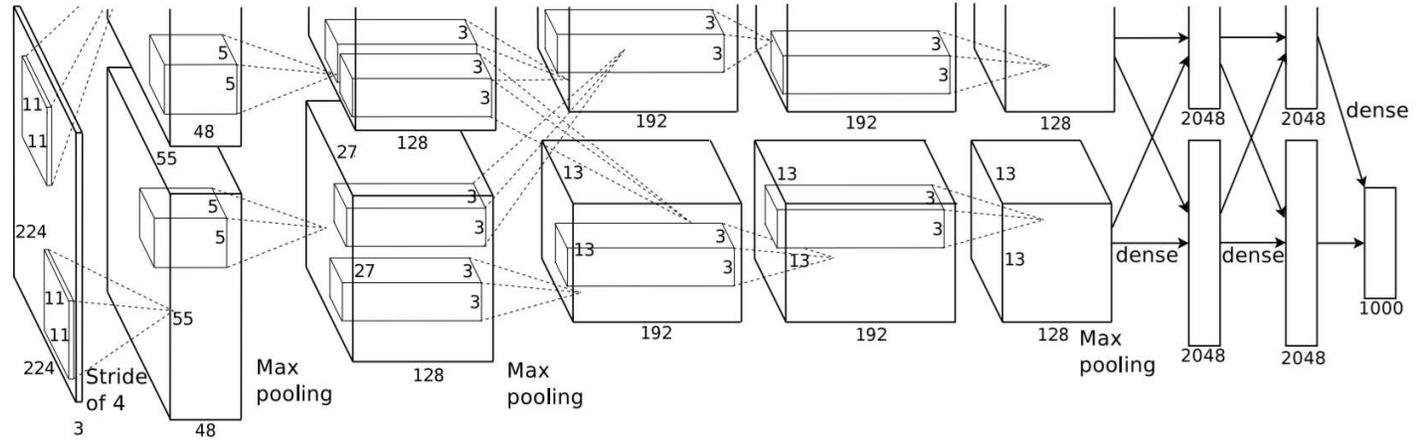
Successes

- ImageNet Dataset
- AlexNet results
- AlexNet
- AlexNet Successors

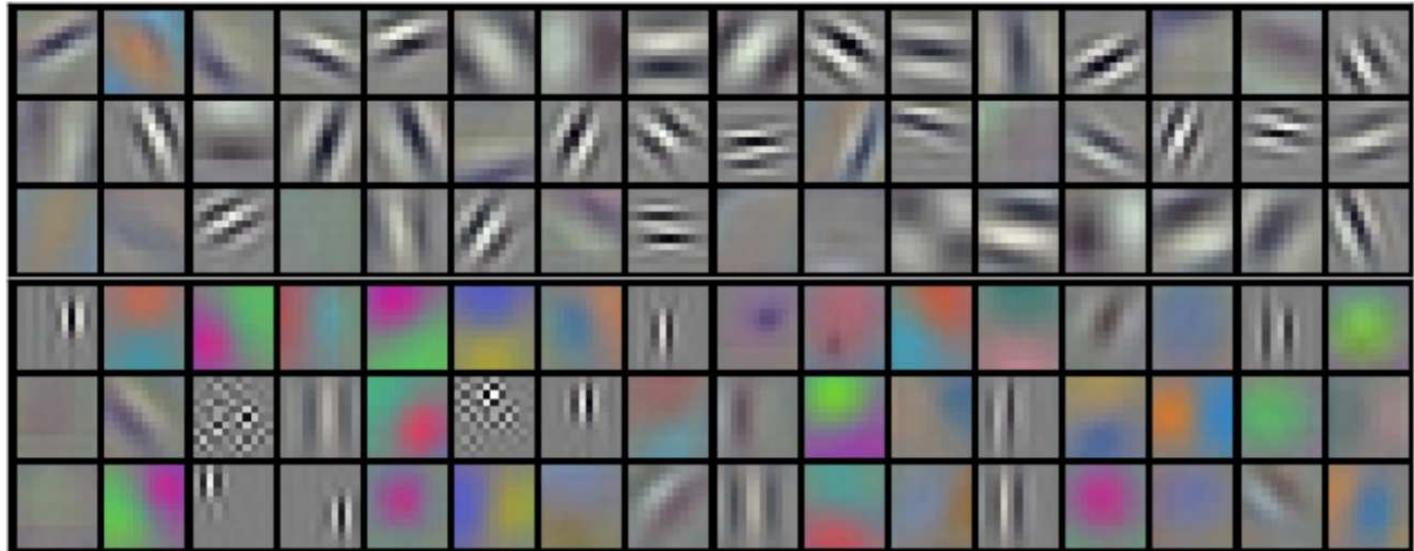
Recurrent Nets

Other remarks

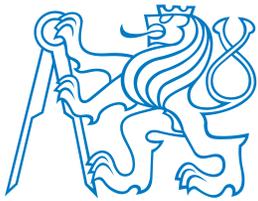
Summary



Filters learned at the first hidden layer:



Both pictures from Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012



# AlexNet Successors

---

ILSVRC 2012: Alexnet's Top-5 error rate 15.3 %

- Two separate data flows for 2 GPUs, 60M parameters.
- ReLUs, dropout.

Deep Learning

ConvNets

Successes

- ImageNet Dataset
- AlexNet results
- AlexNet
- AlexNet Successors

Recurrent Nets

Other remarks

Summary



# AlexNet Successors

---

ILSVRC 2012: Alexnet's Top-5 error rate 15.3 %

- Two separate data flows for 2 GPUs, 60M parameters.
- ReLUs, dropout.

ILSVRC 2013: ZFNet's Top-5 error rate 11.2 %

- Larger number of smaller filters, deeper.

[Deep Learning](#)

[ConvNets](#)

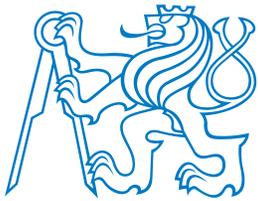
[Successes](#)

- ImageNet Dataset
- AlexNet results
- AlexNet
- [AlexNet Successors](#)

[Recurrent Nets](#)

[Other remarks](#)

[Summary](#)



# AlexNet Successors

---

Deep Learning

ConvNets

Successes

- ImageNet Dataset
- AlexNet results
- AlexNet
- AlexNet Successors

Recurrent Nets

Other remarks

Summary

ILSVRC 2012: Alexnet's Top-5 error rate 15.3 %

- Two separate data flows for 2 GPUs, 60M parameters.
- ReLUs, dropout.

ILSVRC 2013: ZFNet's Top-5 error rate 11.2 %

- Larger number of smaller filters, deeper.

ILSVRC 2014: VGG Net's Top 5 error rate 6.8 %

- 22 layers of neurons

ILSVRC 2014: GoogLeNet's Top 5 error rate 6.66 %

- More than 30 layers of neurons.
- Inception modules instead of convolutional (a module containing several convolutional layers with small filters, and pooling).
- Only 5M parameters (compared to 60M of AlexNet)
- Human error rate (not easily obtainable) is 5.1 %



# AlexNet Successors

---

Deep Learning

ConvNets

Successes

- ImageNet Dataset
- AlexNet results
- AlexNet
- AlexNet Successors

Recurrent Nets

Other remarks

Summary

ILSVRC 2012: Alexnet's Top-5 error rate 15.3 %

- Two separate data flows for 2 GPUs, 60M parameters.
- ReLUs, dropout.

ILSVRC 2013: ZFNet's Top-5 error rate 11.2 %

- Larger number of smaller filters, deeper.

ILSVRC 2014: VGG Net's Top 5 error rate 6.8 %

- 22 layers of neurons

ILSVRC 2014: GoogLeNet's Top 5 error rate 6.66 %

- More than 30 layers of neurons.
- Inception modules instead of convolutional (a module containing several convolutional layers with small filters, and pooling).
- Only 5M parameters (compared to 60M of AlexNet)
- Human error rate (not easily obtainable) is 5.1 %

ILSVRC 2015: ResNet's Top 5 error 3.6 %

- 152 layers (2-3 weeks on 8 GPUs)
- Skip connections: each layer is trained to residuals of the previous layer.



# Recurrent Neural Networks



# Recurrent NNs

---

## Memory:

- Feedforward NNs (including ConvNets) do not have a memory:
  - Fixed-sized vector as input (e.g. an image), fixed-sized vector as output (e.g. probabilities of different classes).
  - Fixed amount of computational steps (e.g. the number of layers in the model).
- In real world, reasoning is a lot about memory, context, thought persistence.
- Even if your inputs/outputs are fixed vectors, it is still possible to process them in a sequential manner.

Deep Learning

ConvNets

Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary



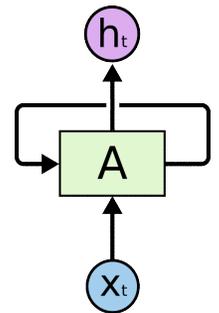
# Recurrent NNs

## Memory:

- Feedforward NNs (including ConvNets) do not have a memory:
  - Fixed-sized vector as input (e.g. an image), fixed-sized vector as output (e.g. probabilities of different classes).
  - Fixed amount of computational steps (e.g. the number of layers in the model).
- In real world, reasoning is a lot about memory, context, thought persistence.
- Even if your inputs/outputs are fixed vectors, it is still possible to process them in a sequential manner.

## Recurrent Neural Networks (RNNs):

- Allow us to operate over *sequences of vectors*.
- Feedback loops in network implement the concept of memory.
- A part of network  $A$  transforms the current input  $x_t$  and the previous network state into the network output.



Deep Learning

ConvNets

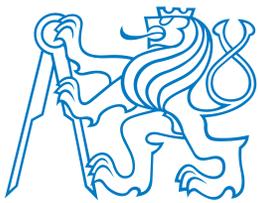
Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary



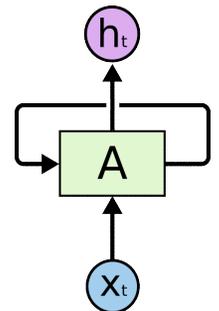
# Recurrent NNs

## Memory:

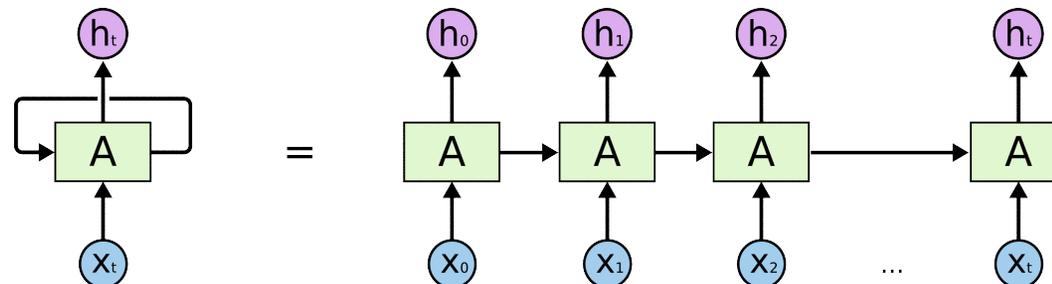
- Feedforward NNs (including ConvNets) do not have a memory:
  - Fixed-sized vector as input (e.g. an image), fixed-sized vector as output (e.g. probabilities of different classes).
  - Fixed amount of computational steps (e.g. the number of layers in the model).
- In real world, reasoning is a lot about memory, context, thought persistence.
- Even if your inputs/outputs are fixed vectors, it is still possible to process them in a sequential manner.

## Recurrent Neural Networks (RNNs):

- Allow us to operate over *sequences of vectors*.
- Feedback loops in network implement the concept of memory.
- A part of network  $A$  transforms the current input  $x_t$  and the previous network state into the network output.



**RNN unrolled in time:** multiple copies of the same network, each passing a message to a successor:



Both pictures from Christopher Olah: Understanding LSTMs

Deep Learning

ConvNets

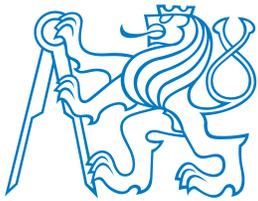
Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary



# RNN Applications

Deep Learning

ConvNets

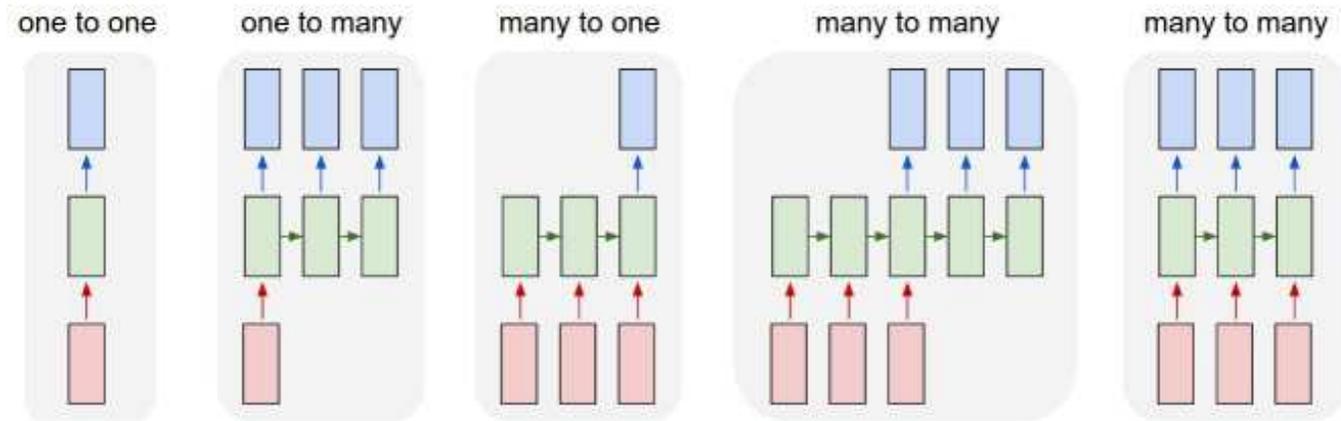
Successes

Recurrent Nets

- Recurrent NNs
- **RNN Applications**
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary



Andrej Karpathy: The Unreasonable Effectiveness of Recurrent Neural Networks

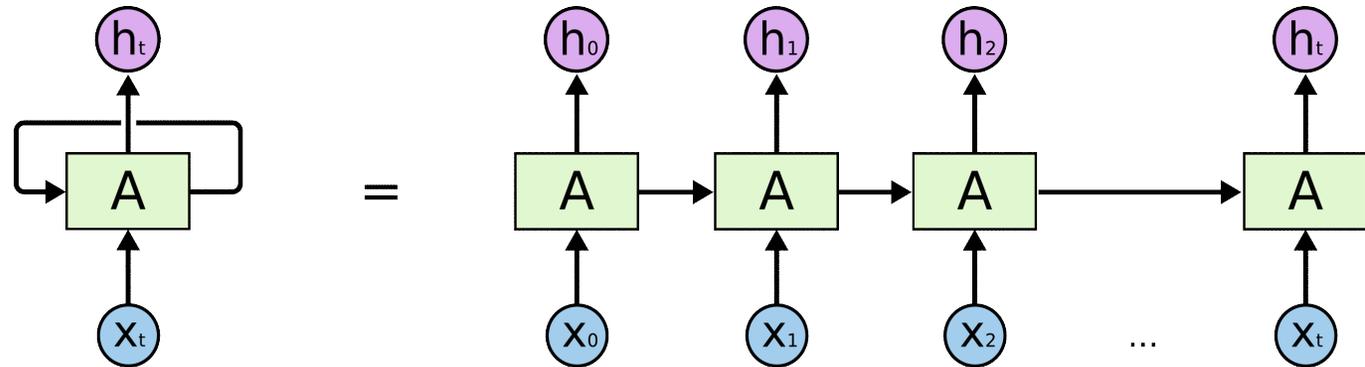
1. Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).
  2. Sequence output (e.g. image captioning takes an image and outputs a sentence of words).
  3. Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).
  4. Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).
  5. Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).
- ...speech recognition, language modeling, machine translation, attention modeling, ...



# Backpropagation through time

## Backprop through time (BPTT):

- A method to compute the gradients for weights of RNNs.
- Unroll the network several steps in time:



Christopher Olah: Understanding LSTMs

- This can be viewed as a “normal” feedforward network with the same weights in all the layers.
- BPTT is equivalent to normal BP on unfolded networks, with the exception that *the gradients for a particular weight are summed over the layers.*

Deep Learning

ConvNets

Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary



# Short- and long-term dependencies

---

**Dependencies:** RNNs connect previous information to the present task, e.g.

- previous video frames help us understand the present frame,
- a language model predicts the next word based on previous ones, ...

Deep Learning

ConvNets

Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- **Dependencies**
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary



# Short- and long-term dependencies

Deep Learning

ConvNets

Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary

**Dependencies:** RNNs connect previous information to the present task, e.g.

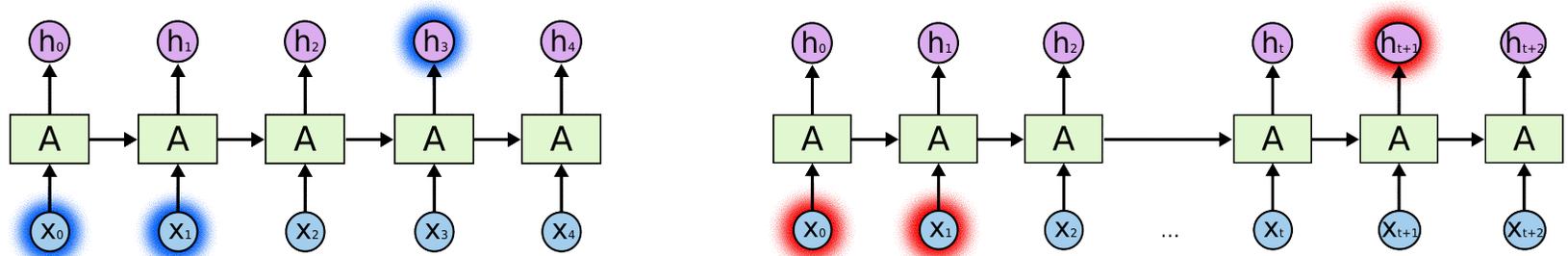
- previous video frames help us understand the present frame,
- a language model predicts the next word based on previous ones, ...

**Short-term dependencies:** easy for vanilla RNNs.

- Predict the last word in the sequence “the clouds are in the [sky]”.
- The gap between relevant information and the place it is needed is small.

**Long-term dependencies:** hard for vanilla RNNs, special units required.

- Predict the last word in the sequence “I was born in France. Blah blah blah ... blah. I speak fluently [French].”
- In theory, vanilla RNNs are absolutely capable of handling such long-term dependencies.
- In practice, RNNs don't seem to be able to learn them.



Christopher Olah: Understanding LSTMs



# Long Short-Term Memory

---

## LSTM Networks:

- Special kind of RNN explicitly designed to be capable of learning long-term dependencies.
- Introduced by Hochreiter and Schmidhuber in 1997.
- In practice, all the successful applications of RNNs were achieved by LSTM Networks.

Deep Learning

ConvNets

Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary

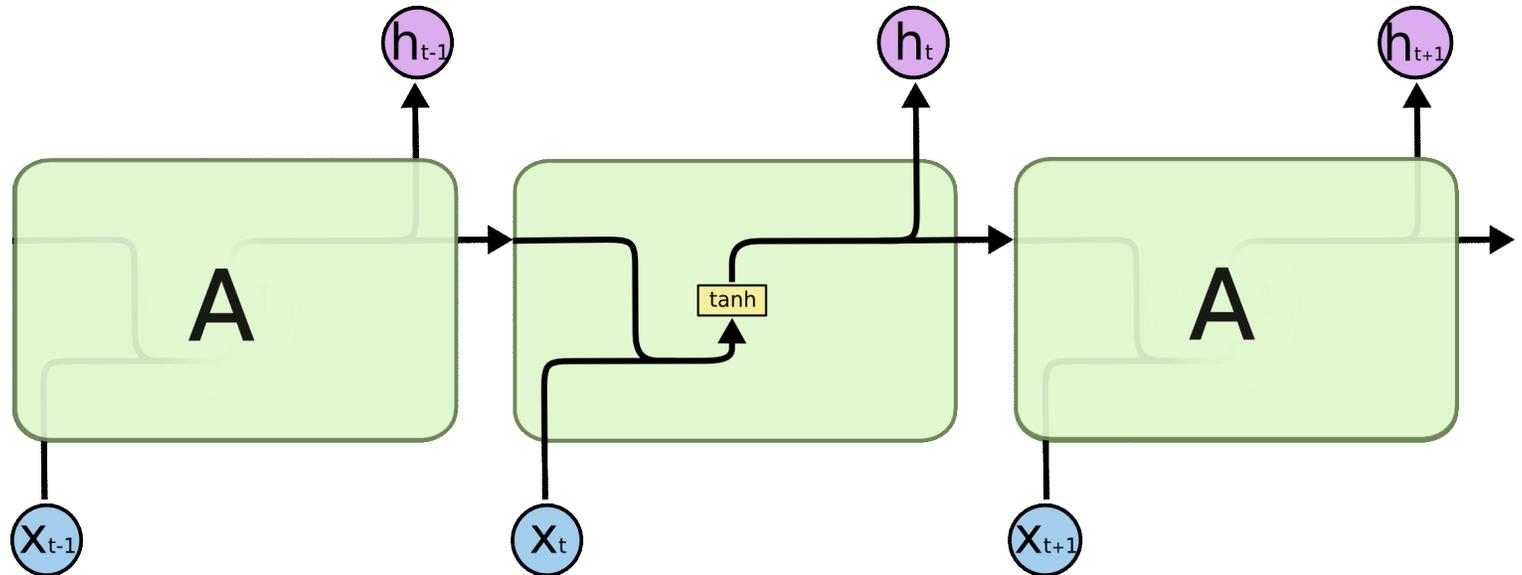


# Long Short-Term Memory

LSTM Networks:

- Special kind of RNN explicitly designed to be capable of learning long-term dependencies.
- Introduced by Hochreiter and Schmidhuber in 1997.
- In practice, all the successful applications of RNNs were achieved by LSTM Networks.

A vanilla RNN unrolled in time:



Christopher Olah: Understanding LSTMs

Deep Learning

ConvNets

Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

Summary

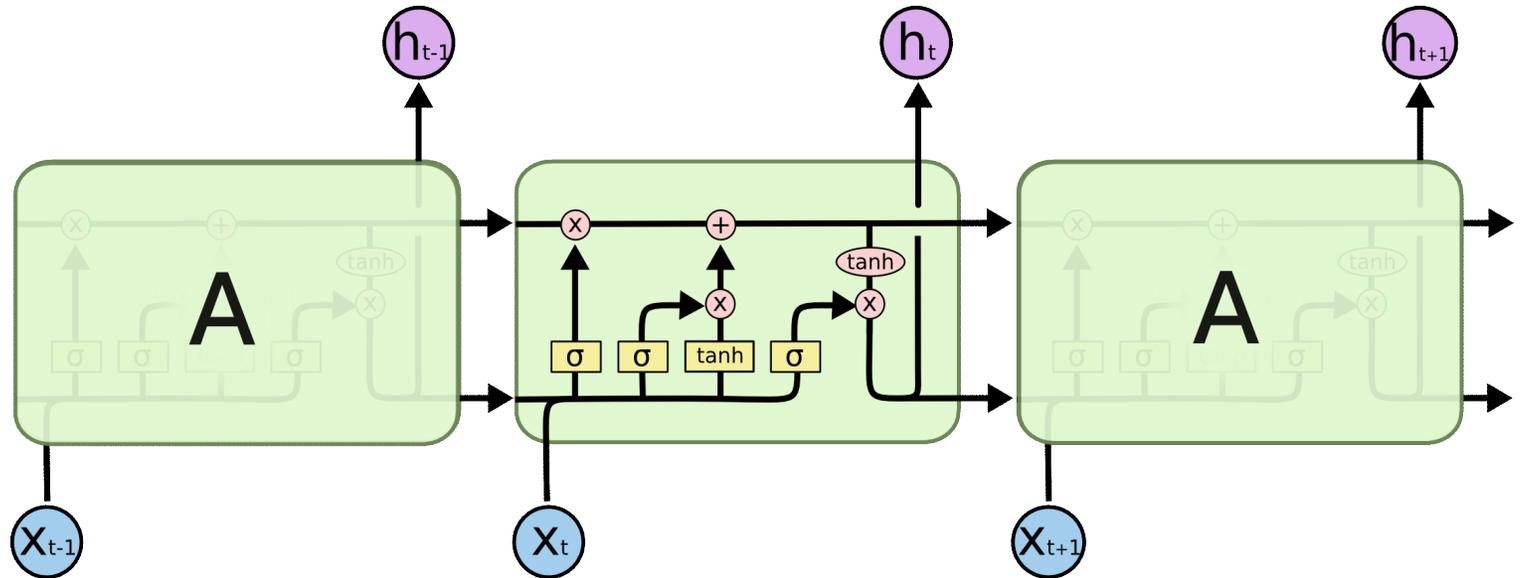


# Long Short-Term Memory

LSTM Networks:

- Special kind of RNN explicitly designed to be capable of learning long-term dependencies.
- Introduced by Hochreiter and Schmidhuber in 1997.
- In practice, all the successful applications of RNNs were achieved by LSTM Networks.

An LSTM unrolled in time:



Christopher Olah: Understanding LSTMs

Deep Learning

ConvNets

Successes

Recurrent Nets

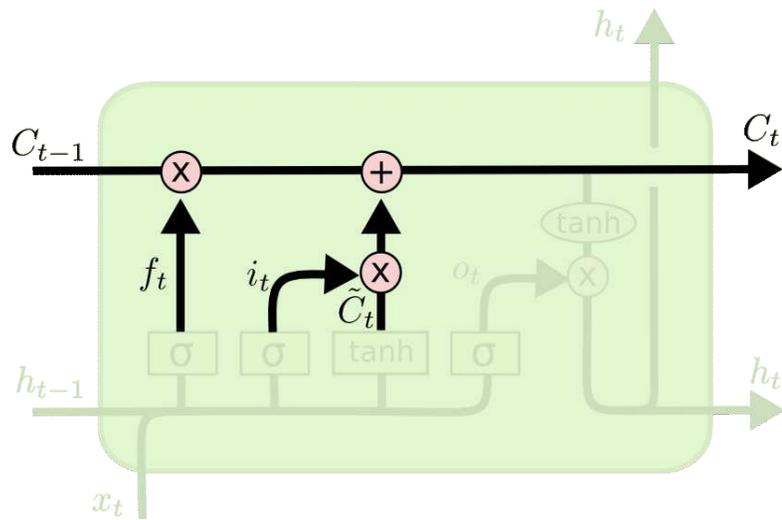
- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- Image captioning
- Captioning results

Other remarks

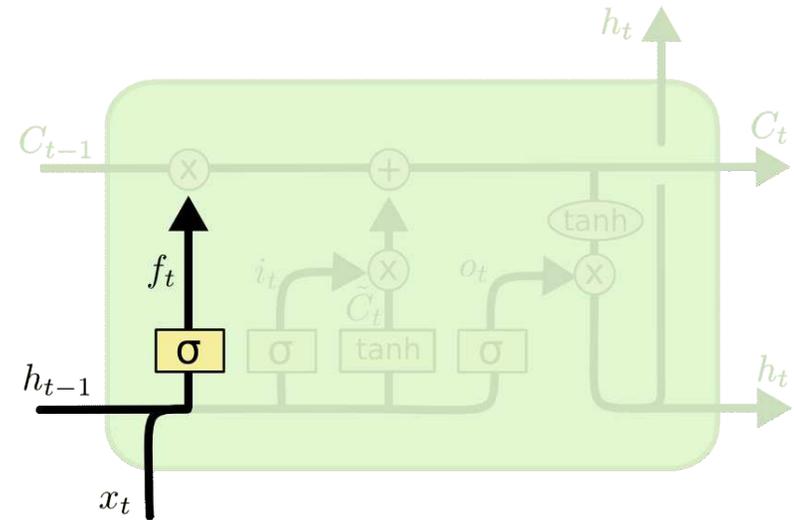
Summary

# LSTM gates

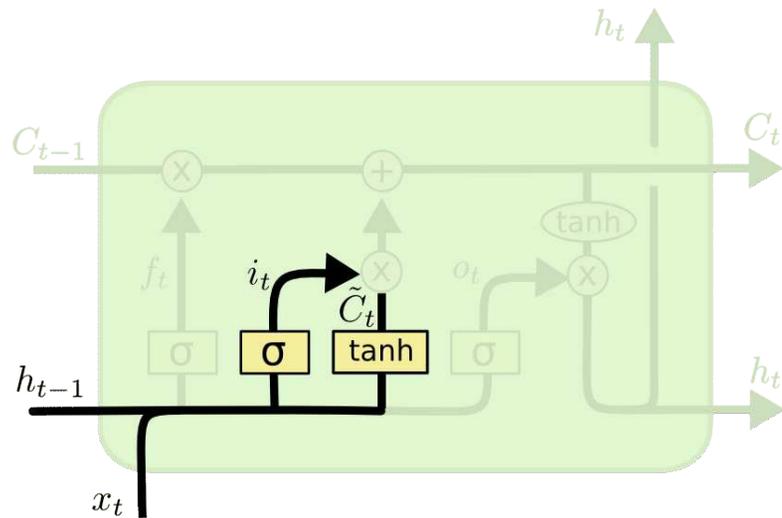
LSTM state (memory):



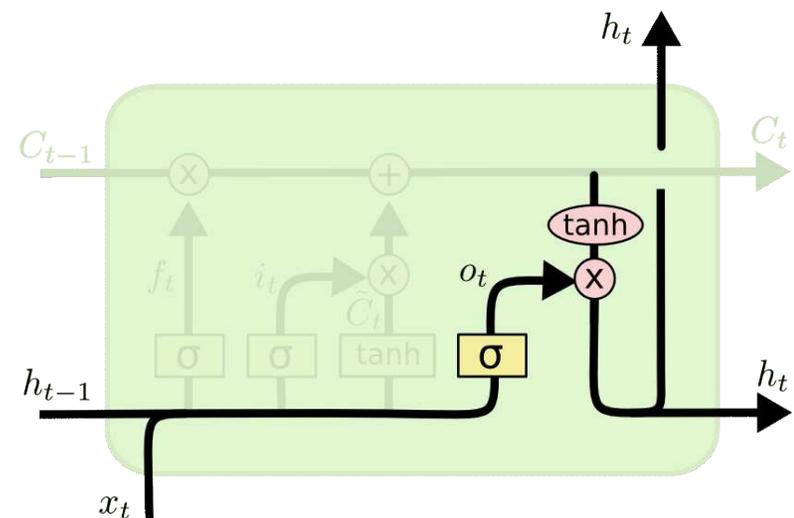
LSTM forget gate:



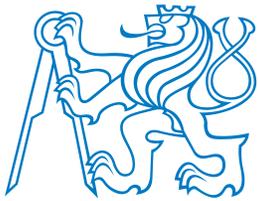
LSTM input gate:



LSTM output gate:



Christopher Olah: Understanding LSTMs



# Image captioning

- Automated creation of image descriptions in natural language.
- A combination of
  - word embeddings,
  - ConvNets generating a high-level representation of the image, and
  - RNNs generating the textual description.
- The model is trained to maximize the likelihood of the target description sentence given the training image.

Deep Learning

ConvNets

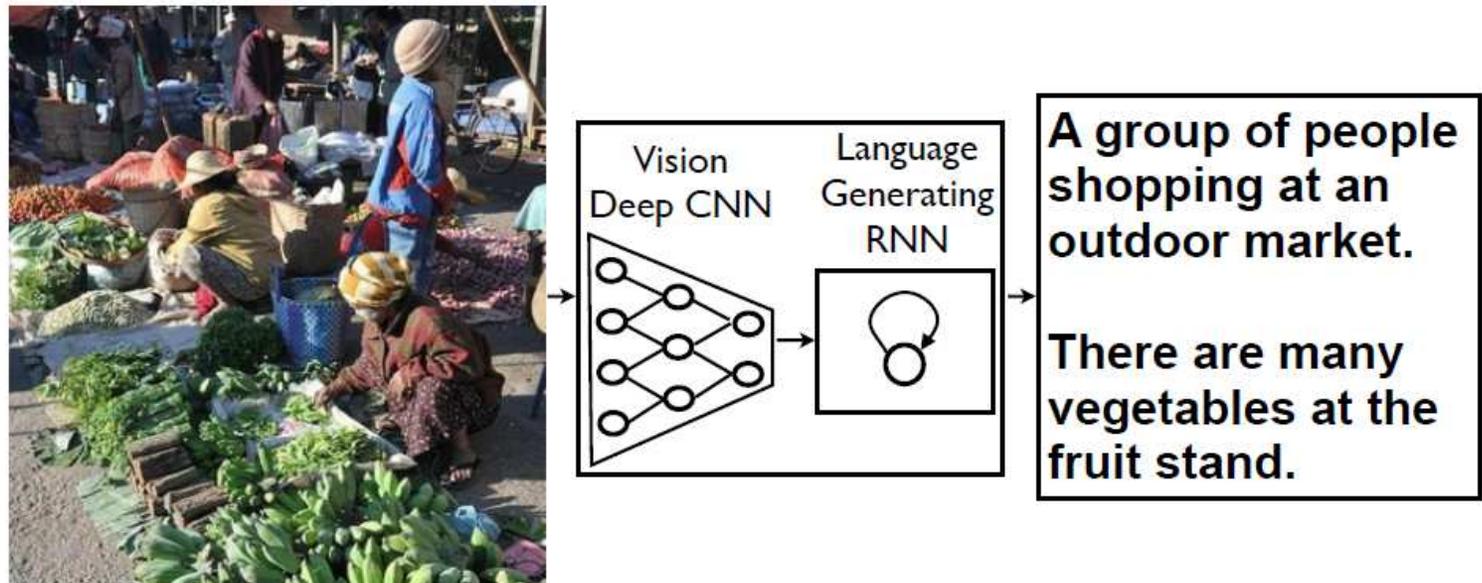
Successes

Recurrent Nets

- Recurrent NNs
- RNN Applications
- BP in time
- Dependencies
- LSTM
- LSTM gates
- **Image captioning**
- Captioning results

Other remarks

Summary



Vinyals et al., Show and Tell: A Neural Image Caption Generator. 2015

# Image captioning results

A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



A skateboarder does a trick on a ramp.



A dog is jumping to catch a frisbee.



A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.



A little girl in a pink hat is blowing bubbles.



A refrigerator filled with lots of food and drinks.



A herd of elephants walking across a dry grass field.



A close up of a cat laying on a couch.



A red motorcycle parked on the side of the road.



A yellow school bus parked in a parking lot.



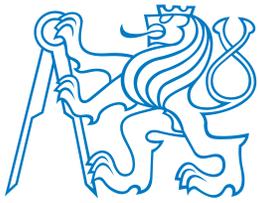
Describes without errors

Describes with minor errors

Somewhat related to the image

Unrelated to the image

Vinyals et al., Show and Tell: A Neural Image Caption Generator. 2015



## Other remarks



# Software

---

- Caffe
- Theano
- TensorFlow
- Torch
- Keras

Deep Learning

ConvNets

Successes

Recurrent Nets

Other remarks

- [Software](#)
- [Resources](#)

Summary



## Resources

---

- Yann LeCun, Yoshua Bengio, Geoffrey Hinton: Deep Learning. Nature, 2015. doi:10.1038/nature14539
- Michael A. Nielsen: Neural Networks and Deep Learning, Determination Press, 2015
- Ian Goodfellow, Yoshua Bengio and Aaron Courville: Deep Learning.
- Christopher Olah: Deep Learning, NLP, and Representations
- Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition
- Adit Deshpande: Understanding CNNs (Part 1, Part 2, Part 3)
- YN<sup>2</sup>: A Guide to Deep Learning
- Andrej Karpathy: The Unreasonable Effectiveness of Recurrent Neural Networks
- Christopher Olah: Understanding LSTM Networks
- Denny Britz: Recurrent neural networks tutorial

Deep Learning

ConvNets

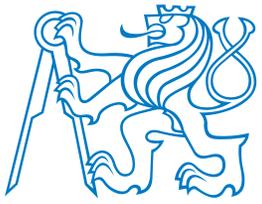
Successes

Recurrent Nets

Other remarks

- Software
- Resources

Summary



# Summary

# Competencies

---

After this lecture, a student shall be able to ...

- define what a deep learning is and how it is related to representation learning;
- describe word embeddings (word2vec) and exemplify its features;
- explain why deep networks are hard to train, especially the vanishing gradient effect;
- describe the factors that facilitated the practical use of deep learning in the last decade;
- explain what an autoencoder is and how it is related to pre-training of deep nets;
- explain the principle of convolutional layers, the importance of weight sharing and pooling, and describe their main differences from fully connected layers;
- give examples of applications of convolutional networks to image processing;
- describe the difference of recurrent neural networks from feedforward networks;
- describe the unrolling of RNN in time and give an example;
- explain the difference between short- and long-term dependencies when processing sequences;
- describe LSTM and its main differences from a regular RNN unit;
- explain what the backpropagation in time is and what it is used for;
- give examples of applications of recurrent neural networks to language processing.