Neural networks

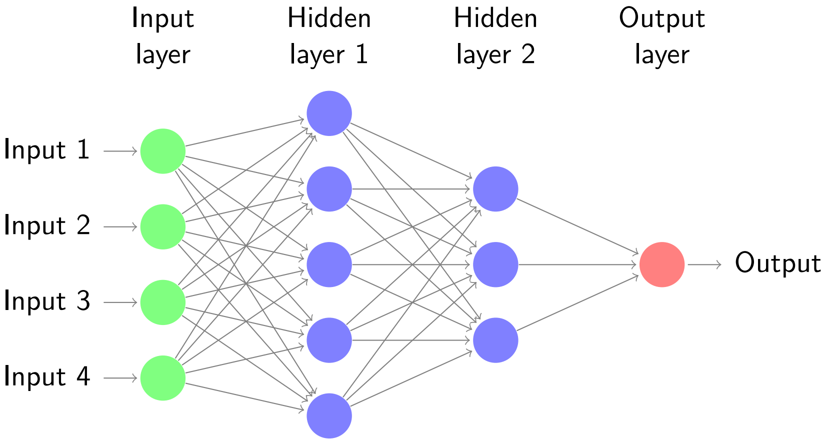
Architectures and training tips

Sebastian Björkqvist

IPRally Technologies

09.01.2019

What is a neural network?



Modified from `http://www.texample.net/tikz/examples/neural-network/`

What is a neural network?

At each hidden layer node i the output value is calculated by

$$o_i = f(\sum w_{ki} o_{ki-1} + b_i).$$

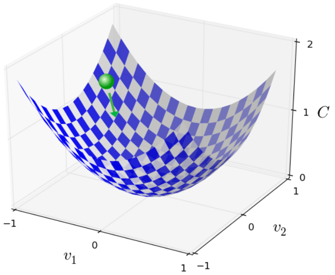
What is a neural network?

At each hidden layer node i the output value is calculated by

$$o_i = f(\sum w_{ki} o_{ki-1} + b_i).$$

The function f is called the activation function. It must be non-linear to allow the network to learn non-linear dependencies.

Training neural networks using SGD



[Nielsen, 2015], Chapter 1.

The training data is processed in small batches, and the weights of the model are iteratively updated by going in the direction of the negative gradient of the loss function.

Why neural networks?

- Can approximate any function [Hornik, 1991]

Why neural networks?

- Can approximate any function [Hornik, 1991]
- May learn to respond to unexpected patterns

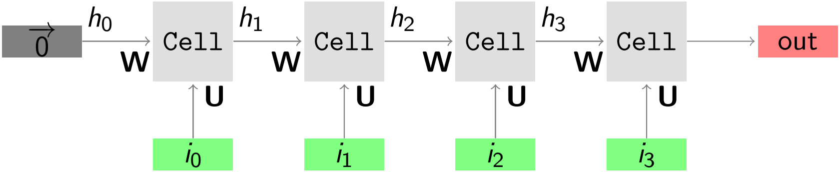
Why neural networks?

- Can approximate any function [Hornik, 1991]
- May learn to respond to unexpected patterns
- Useful especially when the amount of data is large compared to input dimensionality

Why neural networks?

- Can approximate any function [Hornik, 1991]
- May learn to respond to unexpected patterns
- Useful especially when the amount of data is large compared to input dimensionality
- Less need for feature engineering compared to traditional ML methods

Recurrent neural network (RNN)



Processes each element of the input sequence in order, and keeps information about the past elements in a hidden state vector.

Recurrent neural network (RNN)

At each timestep t the new hidden state is calculated using the new input at this timestep and the existing hidden state. The most basic version is the following:

$$h_t = \sigma(Wh_{t-1} + Ui_t + b).$$

Recurrent neural network (RNN)

At each timestep t the new hidden state is calculated using the new input at this timestep and the existing hidden state. The most basic version is the following:

$$h_t = \sigma(Wh_{t-1} + Ui_t + b).$$

Other RNN architectures (for instance LSTM or GRU) use more complicated ways of updating the hidden state to control the flow of information to and from the hidden state.

RNN pros and cons

RNN pros and cons

+ Accepts input of variable size, i.e. sequences (time series, sentences etc)

RNN pros and cons

+ Accepts input of variable size, i.e. sequences (time series, sentences etc)
    - Can even be used to process tree-structured inputs by using Tree-LSTMs [Tai et. al., 2015]

RNN pros and cons

- + Accepts input of variable size, i.e. sequences (time series, sentences etc)
  - Can even be used to process tree-structured inputs by using Tree-LSTMs [Tai et. al., 2015]
- + May learn long-term dependencies, especially when using LSTM architecture

RNN pros and cons

- + Accepts input of variable size, i.e. sequences (time series, sentences etc)
  - Can even be used to process tree-structured inputs by using Tree-LSTMs [Tai et. al., 2015]
- + May learn long-term dependencies, especially when using LSTM architecture
- − Training may be slow when sequence length is large

RNN pros and cons

- \+ Accepts input of variable size, i.e. sequences (time series, sentences etc)
  - Can even be used to process tree-structured inputs by using Tree-LSTMs [Tai et. al., 2015]
- \+ May learn long-term dependencies, especially when using LSTM architecture
- − Training may be slow when sequence length is large
  - This is because each time step depends on the earlier ones and they must thus be processed sequentially

RNN pros and cons

+ Accepts input of variable size, i.e. sequences (time series, sentences etc)
  - Can even be used to process tree-structured inputs by using Tree-LSTMs [Tai et. al., 2015]
+ May learn long-term dependencies, especially when using LSTM architecture
– Training may be slow when sequence length is large
  - This is because each time step depends on the earlier ones and they must thus be processed sequentially
– Prediction accuracy may also suffer if the sequence is long

RNN pros and cons

+ Accepts input of variable size, i.e. sequences (time series, sentences etc)
    - Can even be used to process tree-structured inputs by using Tree-LSTMs [Tai et. al., 2015]
+ May learn long-term dependencies, especially when using LSTM architecture
− Training may be slow when sequence length is large
    - This is because each time step depends on the earlier ones and they must thus be processed sequentially
− Prediction accuracy may also suffer if the sequence is long
    - The model may not remember early inputs and can be biased toward the end of the sequence

RNN real life use case: Patent search

At IPRally we work on automated patent searches. The basic idea is the following:

RNN real life use case: Patent search

At IPRally we work on automated patent searches. The basic idea is the following:

1. Patents are transformed to graphs by extracting the relevant information from the patent claims and specifications

RNN real life use case: Patent search

At IPRally we work on automated patent searches. The basic idea is the following:

1. Patents are transformed to graphs by extracting the relevant information from the patent claims and specifications
2. The graphs are then embedded to vectors by using a Tree-LSTM model

RNN real life use case: Patent search

At IPRally we work on automated patent searches. The basic idea is the following:

1. Patents are transformed to graphs by extracting the relevant information from the patent claims and specifications
2. The graphs are then embedded to vectors by using a Tree-LSTM model
   - The model is trained by using millions of real-life positive and negative novelty citations from previous patent applications

RNN real life use case: Patent search

At IPRally we work on automated patent searches. The basic idea is the following:

1. Patents are transformed to graphs by extracting the relevant information from the patent claims and specifications
2. The graphs are then embedded to vectors by using a Tree-LSTM model
   - The model is trained by using millions of real-life positive and negative novelty citations from previous patent applications
   - Patents with a positive citation get vectors that are close to each other

RNN real life use case: Patent search

At IPRally we work on automated patent searches. The basic idea is the following:

1. Patents are transformed to graphs by extracting the relevant information from the patent claims and specifications
2. The graphs are then embedded to vectors by using a Tree-LSTM model
   - The model is trained by using millions of real-life positive and negative novelty citations from previous patent applications
   - Patents with a positive citation get vectors that are close to each other
3. A prior art search for a new patent can then be done by searching for the nearest neighbors of the vector created from the new invention

RNN real life use case: Patent search

- table
  - worktop
  - legs
    - telescope pipe
    - motor

IPRally

US patent no. 7 908 981

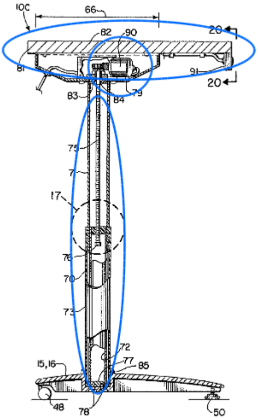


FIG. 16.

Convolutional neural network (CNN)

Feed-forward networks do not scale well to images due to the large input size. Convolutional neural networks are constrained to looking only at a small part of the image at a time, and thus the number of weights stays manageable.

A CNN uses three types of layers: convolutional, pooling and fully connected.

CNN - Convolutional layer

CNN - Convolutional layer

- Consists of a set of learnable filters

CNN - Convolutional layer

- Consists of a set of learnable filters
- Each individual filter is small (i.e. small height and width)

CNN - Convolutional layer

- Consists of a set of learnable filters
- Each individual filter is small (i.e. small height and width)
- During the forward pass we slide each filter across the entire input and compute dot products between input entries and filter weights

CNN - Convolutional layer

- Consists of a set of learnable filters
- Each individual filter is small (i.e. small height and width)
- During the forward pass we slide each filter across the entire input and compute dot products between input entries and filter weights
- The idea is that each filter learns to identify some kind of feature (for instance part of a shape)

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

1	1
1	0

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

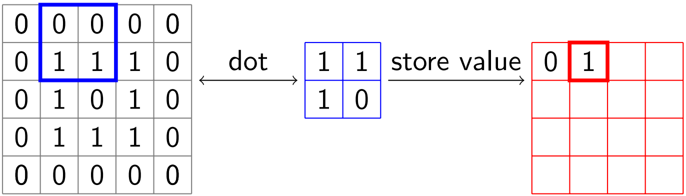
$\xleftrightarrow{\text{dot}}$

1	1
1	0

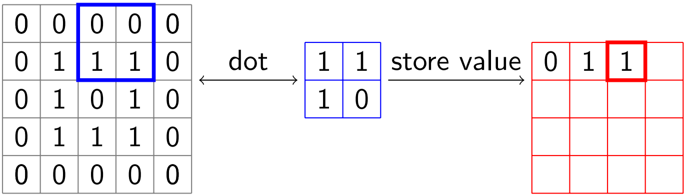
CNN - Convolutional layer

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xleftrightarrow{\text{dot}} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \xrightarrow{\text{store value}} \begin{bmatrix} \boxed{0} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

CNN - Convolutional layer



CNN - Convolutional layer



CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

$\xleftrightarrow{\text{dot}}$

1	1
1	0

$\xrightarrow{\text{store value}}$

0	1	1	1

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

dot

1	1
1	0

store value

0	1	1	1
1			

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

$\xleftrightarrow{\text{dot}}$

1	1
1	0

$\xrightarrow{\text{store value}}$

0	1	1	1
1	3		

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

$\xleftrightarrow{\text{dot}}$

1	1
1	0

$\xrightarrow{\text{store value}}$

0	1	1	1
1	3	2	

CNN - Convolutional layer

$$\begin{array}{|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \xleftrightarrow{\text{dot}} \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 0 \\ \hline \end{array} \xrightarrow{\text{store value}} \begin{array}{|c|c|c|c|}\hline 0 & 1 & 1 & 1 \\ \hline 1 & 3 & 2 & \boxed{2} \\ \hline & & & \\ \hline & & & \\ \hline \end{array}$$

CNN - Convolutional layer

$$\begin{bmatrix}0&0&0&0&0\\0&1&1&1&0\\0&1&0&1&0\\0&1&1&1&0\\0&0&0&0&0\end{bmatrix} \xleftrightarrow{\text{dot}} \begin{bmatrix}1&1\\1&0\end{bmatrix} \xrightarrow{\text{store value}} \begin{bmatrix}0&1&1&1\\1&3&2&2\\\boxed{1}&&&\\&&&\end{bmatrix}$$

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

dot

1	1
1	0

store value

0	1	1	1
1	3	2	2
1	2		

CNN - Convolutional layer

$$\begin{array}{|c|c|c|c|c|}\hline 0&0&0&0&0\\ \hline 0&1&1&1&0\\ \hline 0&1&0&1&0\\ \hline 0&1&1&1&0\\ \hline 0&0&0&0&0\\ \hline\end{array} \xleftrightarrow{\text{dot}} \begin{array}{|c|c|}\hline 1&1\\ \hline 1&0\\ \hline\end{array} \xrightarrow{\text{store value}} \begin{array}{|c|c|c|c|}\hline 0&1&1&1\\ \hline 1&3&2&2\\ \hline 1&2&\boxed{2}&\\ \hline &&&\\ \hline\end{array}$$

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

$\xleftrightarrow{\text{dot}}$

1	1
1	0

$\xrightarrow{\text{store value}}$

0	1	1	1
1	3	2	2
1	2	2	2

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

$\xleftrightarrow{\text{dot}}$

1	1
1	0

$\xrightarrow{\text{store value}}$

0	1	1	1
1	3	2	2
1	2	2	2
1			

CNN - Convolutional layer

$$
\begin{array}{|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 \\ \hline
0 & 1 & 1 & 1 & 0 \\ \hline
0 & 1 & 0 & 1 & 0 \\ \hline
0 & 1 & 1 & 1 & 0 \\ \hline
0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}
\xleftrightarrow{\text{dot}}
\begin{array}{|c|c|}
\hline
1 & 1 \\ \hline
1 & 0 \\ \hline
\end{array}
\xrightarrow{\text{store value}}
\begin{array}{|c|c|c|c|}
\hline
0 & 1 & 1 & 1 \\ \hline
1 & 3 & 2 & 2 \\ \hline
1 & 2 & 2 & 2 \\ \hline
1 & \boxed{2} & & \\ \hline
\end{array}
$$

CNN - Convolutional layer

$$
\begin{array}{|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 \\ \hline
0 & 1 & 1 & 1 & 0 \\ \hline
0 & 1 & 0 & 1 & 0 \\ \hline
0 & 1 & 1 & 1 & 0 \\ \hline
0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}
\xleftrightarrow{\text{dot}}
\begin{array}{|c|c|}
\hline
1 & 1 \\ \hline
1 & 0 \\ \hline
\end{array}
\xrightarrow{\text{store value}}
\begin{array}{|c|c|c|c|}
\hline
0 & 1 & 1 & 1 \\ \hline
1 & 3 & 2 & 2 \\ \hline
1 & 2 & 2 & 2 \\ \hline
1 & 2 & \boxed{2} & \\ \hline
\end{array}
$$

CNN - Convolutional layer

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

dot

1	1
1	0

store value

0	1	1	1
1	3	2	2
1	2	2	2
1	2	2	1

CNN - Pooling layer

CNN - Pooling layer

- Reduces the size of the representation by summarizing parts of the input

CNN - Pooling layer

- Reduces the size of the representation by summarizing parts of the input
    - This is done to decrease the amount of computation in the network

CNN - Pooling layer

- Reduces the size of the representation by summarizing parts of the input
  - This is done to decrease the amount of computation in the network
- A common way is to take the maximum value of a small grid (max pooling)

CNN - Pooling layer

- Reduces the size of the representation by summarizing parts of the input
  - This is done to decrease the amount of computation in the network
- A common way is to take the maximum value of a small grid (max pooling)
  - For instance if we use max pooling with a filter of size 2x2 we discard 75 percent of the values

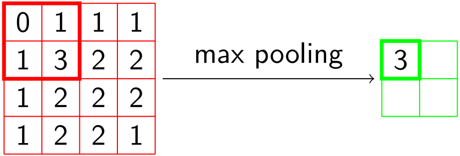
CNN - Pooling layer

0	1	1	1
1	3	2	2
1	2	2	2
1	2	2	1

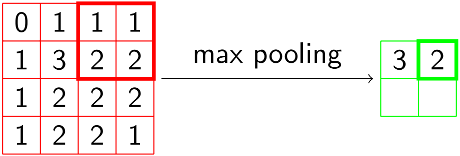
CNN - Pooling layer

0	1	1	1
1	3	2	2
1	2	2	2
1	2	2	1

CNN - Pooling layer



CNN - Pooling layer



CNN - Pooling layer

$$\begin{bmatrix}0&1&1&1\\1&3&2&2\\1&2&2&2\\1&2&2&1\end{bmatrix} \xrightarrow{\text{max pooling}} \begin{bmatrix}3&2\\2&\end{bmatrix}$$

CNN - Pooling layer

$$\begin{array}{|c|c|c|c|}\hline 0 & 1 & 1 & 1 \\ \hline 1 & 3 & 2 & 2 \\ \hline 1 & 2 & 2 & 2 \\ \hline 1 & 2 & 2 & 1 \\ \hline \end{array} \xrightarrow{\text{max pooling}} \begin{array}{|c|c|}\hline 3 & 2 \\ \hline 2 & 2 \\ \hline \end{array}$$

CNN pros and cons

CNN pros and cons

+ Works well with image data and text data

CNN pros and cons

- \+ Works well with image data and text data
- \+ Training can be effectively parallelized

CNN pros and cons

+ Works well with image data and text data
+ Training can be effectively parallelized
+ Pre-existing models can be fine-tuned for specific tasks

CNN pros and cons

- \+ Works well with image data and text data
- \+ Training can be effectively parallelized
- \+ Pre-existing models can be fine-tuned for specific tasks
- − Does not take into account orientation of the object

Challenges when training neural networks

Challenges when training neural networks

- Finding the optimal neural network layout is often time-consuming

Challenges when training neural networks

- Finding the optimal neural network layout is often time-consuming
- The model may be sensitive to changes in hyperparameters, especially learning rate and activation function

Challenges when training neural networks

- Finding the optimal neural network layout is often time-consuming
- The model may be sensitive to changes in hyperparameters, especially learning rate and activation function
- A model may take several hours or even days to train
    - This makes hyperparameter searches very expensive

Challenges when training neural networks

- Finding the optimal neural network layout is often time-consuming
- The model may be sensitive to changes in hyperparameters, especially learning rate and activation function
- A model may take several hours or even days to train
    - This makes hyperparameter searches very expensive
- It's often hard to know why the model predicts as it does because of the complexity of the model

Tips and tricks

- Write unit tests for your model [Roberts, 2017]
    - Check that each layer actually changes weights
    - Make sure that model converges on tiny data set

Tips and tricks

- Write unit tests for your model [Roberts, 2017]
  - Check that each layer actually changes weights
  - Make sure that model converges on tiny data set
- Stick to well-known architectures when starting out (e.g. LSTM/GRU for sequential data)

Tips and tricks

- Write unit tests for your model [Roberts, 2017]
  - Check that each layer actually changes weights
  - Make sure that model converges on tiny data set
- Stick to well-known architectures when starting out (e.g. LSTM/GRU for sequential data)
- Start by using small batch size
  - Usually makes model less sensitive to other hyperparameters

Tips and tricks

- Write unit tests for your model [Roberts, 2017]
  - Check that each layer actually changes weights
  - Make sure that model converges on tiny data set
- Stick to well-known architectures when starting out (e.g. LSTM/GRU for sequential data)
- Start by using small batch size
  - Usually makes model less sensitive to other hyperparameters
- Use normalization (batch, layer, group, weight...)
  - Speeds up convergence significantly
  - Start by trying batch normalization for CNN and feed-forward nets and layer normalization for RNN

The curious case of the batch size

- Training of neural nets can be sped up by increasing the batch size, since then the GPU/TPU can process more training examples in parallel
- Unfortunately increasing the batch size may result in a worse model. In extreme cases the model might not learn anything at all! [Masters et. al., 2018]
- The basic rule is to increase the learning rate linearly when increasing the batch size (e.g. double learning rate when doubling batch size)
    - Otherwise the magnitude of the weight updates decreases
- This means that increasing the batch size trades computational efficiency for stale gradients.

References I

- Nielsen, Michael A. *Neural Networks And Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/
- Hornik, Kurt. *Approximation Capabilities of Multilayer Feedforward Networks*. Neural Networks, 4(2), 251–257, 1991.
- Roberts, Chase. *How to unit test machine learning code. Medium.com 2017. https://medium.com/@keeper6928/how-to-unit-test-machine-learning-code-57cf6fd81765.*
- *Tai, Kai Sheng et al.* Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. ACL 2015. https://arxiv.org/abs/1503.00075

References II

Masters, Dominic, Luschi, Carlo. *Revisiting Small Batch Training for Deep Neural Networks. arXiv preprint 2018. https://arxiv.org/abs/1804.07612*