NTNU | Norwegian University of Science and Technology

DEEP LEARNING APPLICATIONS AND ARCHITECTURES

Tor Andre Myrvoll

October 23, 2020

Introduction - Outline

In this lecture we will look at some useful applications of neural networks, as well as some special architectures

- Recurrent Neural Networks (RNN)
- Autoencoders
- Generative Adversarial Networks (GAN)

Recurrent Neural Networks - Basic RNNs

Many problems are not amiable to be modeled using a simple feed-forward network:

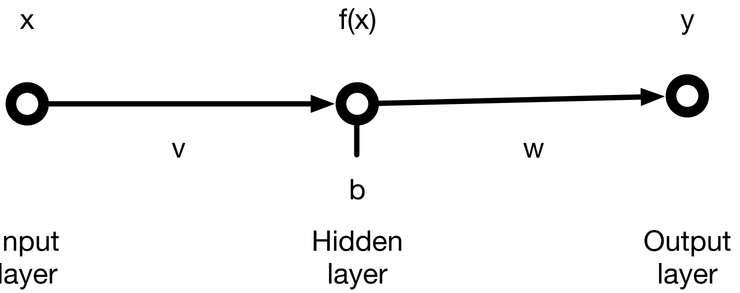
- Language modeling, text generation
- Machine translation
- Speech recognition
- Video tagging
- Prediction problems: $\hat{y}_n = f(y_{n-1}, w_n)$

Recurrent Neural Networks - Basic RNNs

A simple multilayer perceptron takes an input of fixed dimension and produces an output.
There is no memory involved – sequences of inputs can be presented in arbitrary order, giving the same results.
We need some way to keep the *state* of the system between examples.
This can be achieved by using the hidden layers as memory cells as well as processing units.

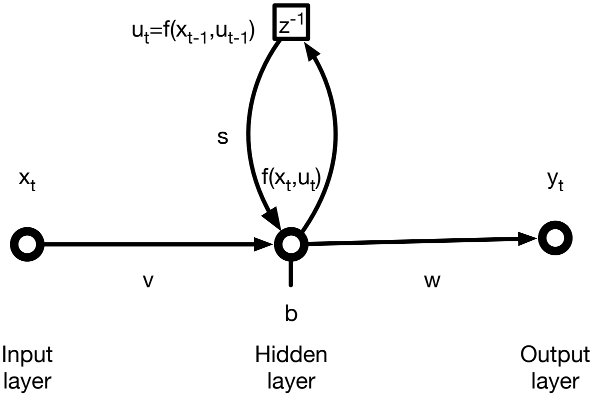
Recurrent Neural Networks - Recursive neural networks

Consider the simplest MLP possible:



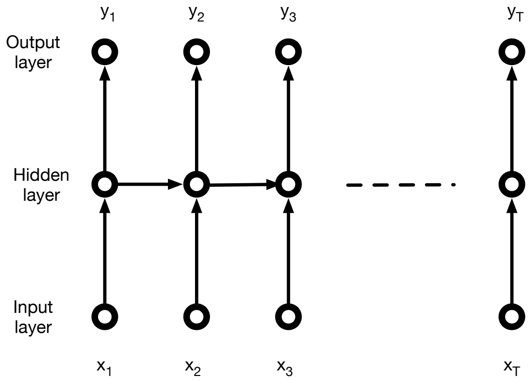
Recurrent Neural Networks - Recursive neural networks

Extending the simplest MLP possible:



Recurrent Neural Networks - Recursive neural networks

Training the basic RNN is done by unfolding the graph, then using back propagation



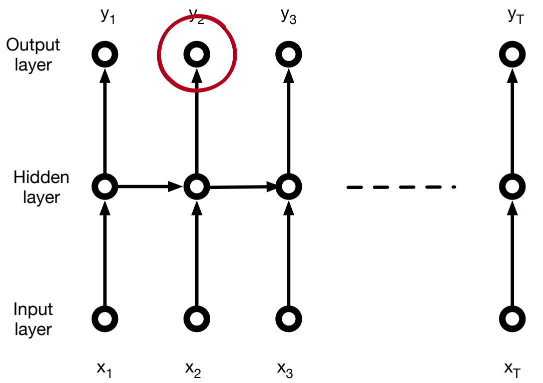
Recurrent Neural Networks - Recursive neural networks

One drawback of the simple recursive network is the inability to utilize future dependencies. This is needed for instance in machine translation, where words can be out of order

- “I like icecream” $\Rightarrow$ “Jeg liker iskrem”
- “I like icecream” $\Rightarrow$ “Watashi wa aisukurīmu ga suki”

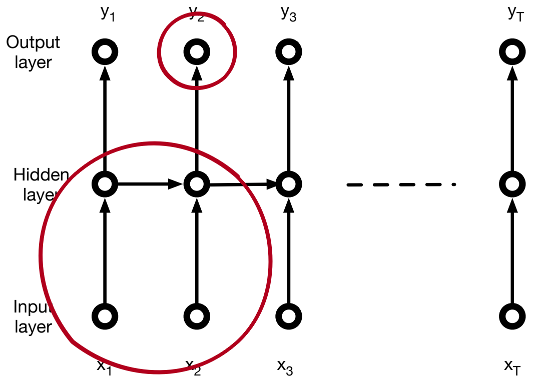
Recurrent Neural Networks - Recursive neural networks

One drawback of the simple recursive network is the inability to utilize future dependencies



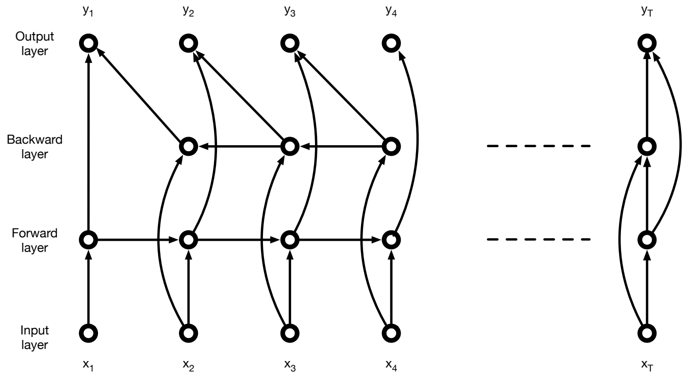
Recurrent Neural Networks - Recursive neural networks

One drawback of the simple recursive network is the inability to utilize future dependencies



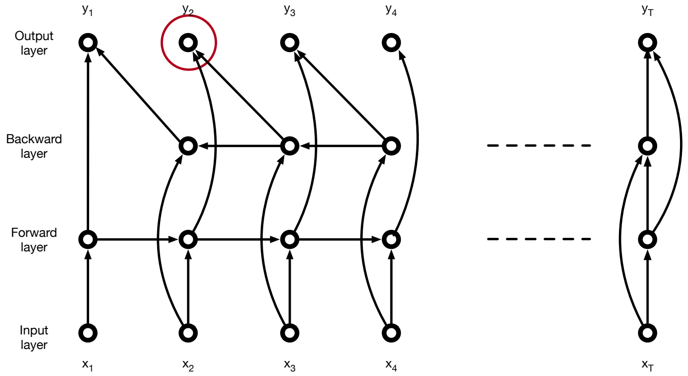
Recurrent Neural Networks - Recursive neural networks

The bi-directional RNN addresses this issue



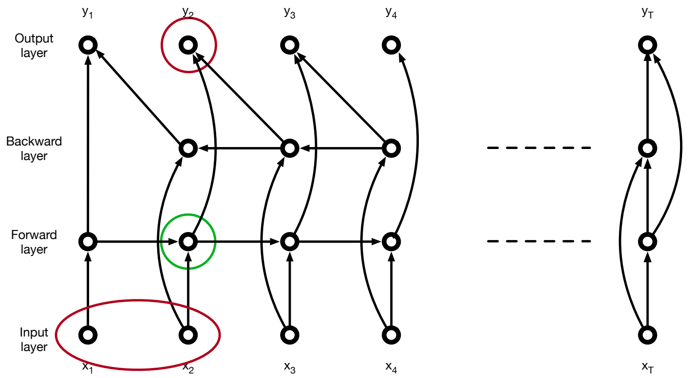
Recurrent Neural Networks - Recursive neural networks

The bi-directional RNN addresses this issue



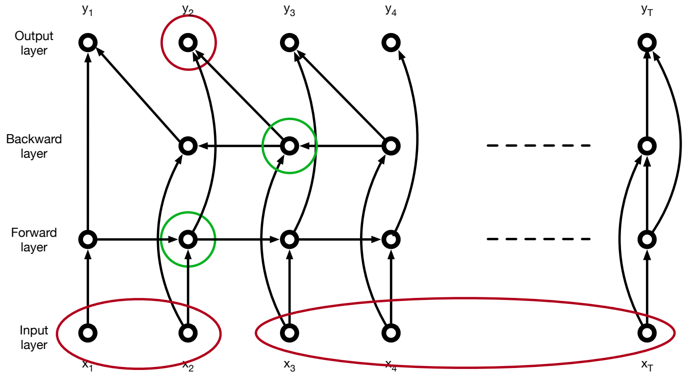
Recurrent Neural Networks - Recursive neural networks

The bi-directional RNN addresses this issue



Recurrent Neural Networks - Recursive neural networks

The bi-directional RNN addresses this issue

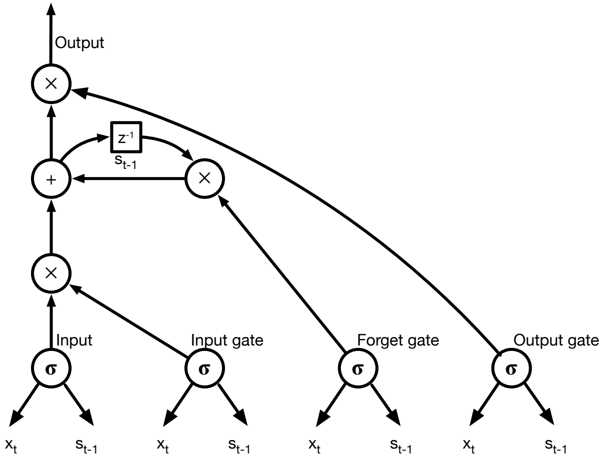


Recurrent Neural Networks - Long short-term memory

The basic recursive neural network has several issues

- Many problems will have more inputs than outputs or vice versa
  - Machine translation
  - Text generation
- Long dependencies are hard to model
  - Example – word prediction
  - “French is the language I am most fluent in as I was born and raised in . . . ”
  - Here the most imporatant information for the prediction, “French”, is 15 words ahead of the work to be predicted (most likely “France”)
- Training is hard due to the vanishing gradient problem
- The *Long short-term memory* (LSTM) network addresses and solves many of these issues

Recurrent Neural Networks - Long short-term memory



Recurrent Neural Networks - Other architectures

There are a large variety of RNN architectures found in the literature. Some examples:

- Gated recurrent unit: Simplified LSTM with no output gate. Better performance on some smaller datasets.
- Encoder-decoder network: Encodes an entire sequence into a neural network “memory”, then generates a new “translated” sequence using a second neural network.
- Transformer networks: Current state-of-the-art in machine translation loosely based on the encoder-decoder principle. Has mostly replaced LSTM in modern systems.

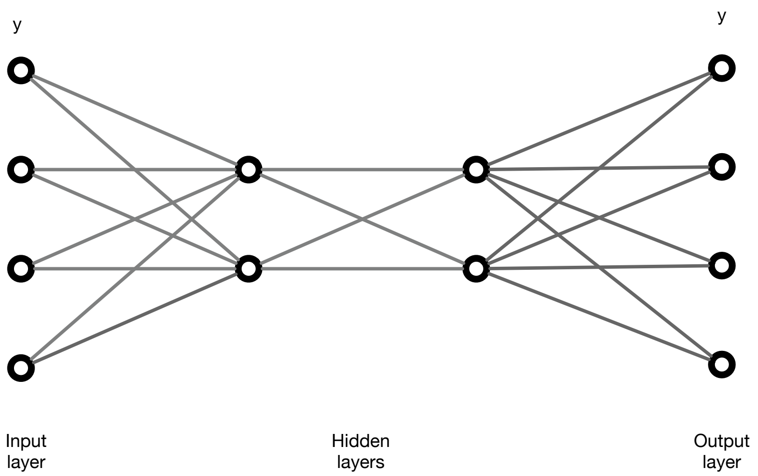
Autoencoders - Motivation

The autoencoder structure is a neural network that learns to estimate

$$\hat{y} = f(y)$$

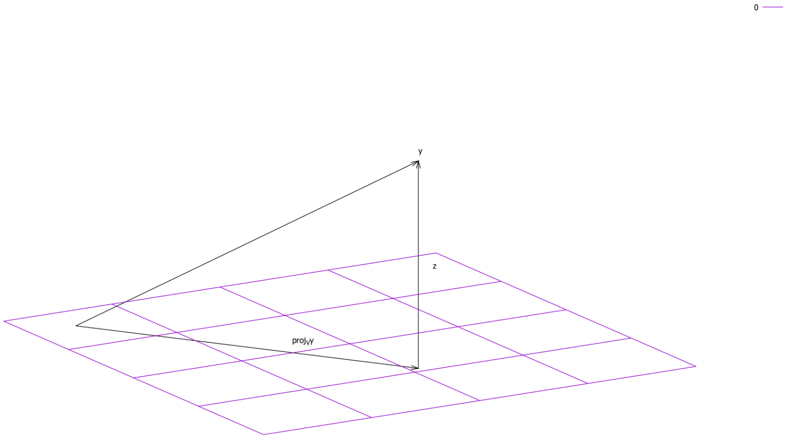
In other words, it tries to learn a mapping from itself to itself.
What is the point?

Autoencoders - The bottleneck



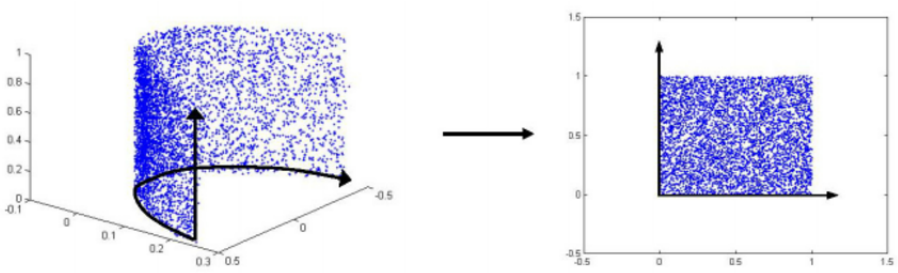
Autoencoders - Manifold projection

The main idea is to do the nonlinear equivalent of a linear projection



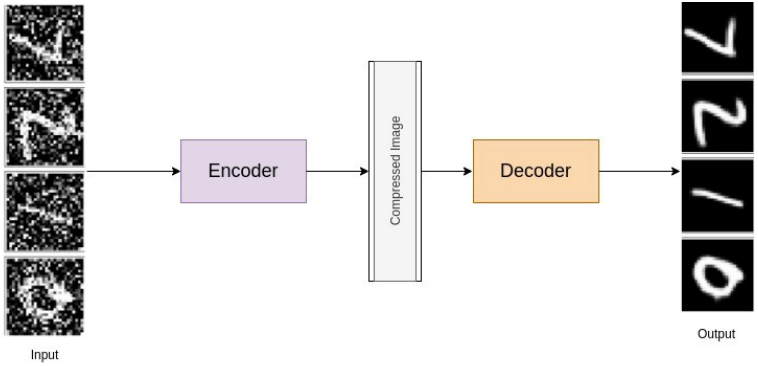
Autoencoders - Manifold projection

The main idea is to do the nonlinear equivalent of a linear projection



Autoencoders - The denoising autoencoder

An alternative to the bottleneck is to add noise to the data and then learn the network to clean it up. This will force the neural network to learn the underlying representation of the signal



Autoencoders - Applications

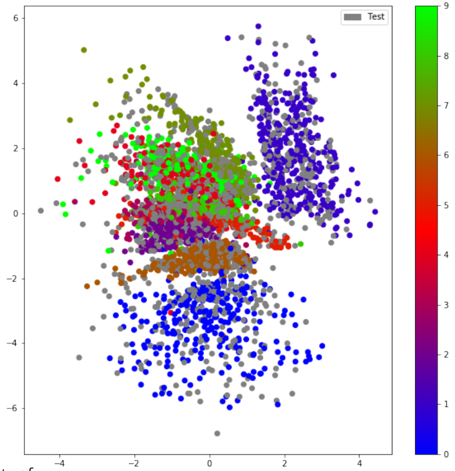
- Feature extraction: Use an autoencoder to find a low-level representation of the data. Then use the first half of the network as a feature extractor
- Speech enhancement: Noisy spectrograms are used as inputs, and the NN learns to recover the clean spectrograms
- Anomaly detection: Clean data is used to train a predictor $\hat{y}_{n+1} = f(y_1^n)$. If the example data is different from the training data, the variance of the predictor will increase, indicating an anomaly.

Autoencoders - Example (from kaggle.com)

The MNIST handwritte digits are 28×28 pixels $\Rightarrow$ 784 dimensional input

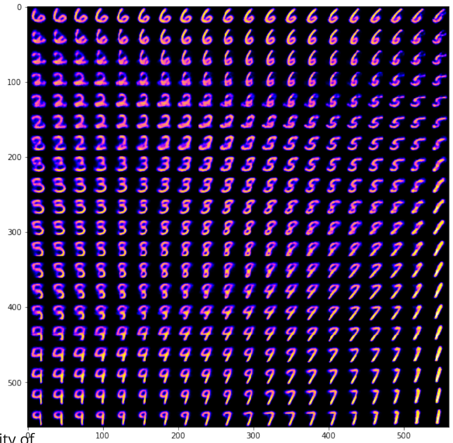
Autoencoders - Example (from kaggle.com)

We can use an autoencode to project it into a 2-dimensional space



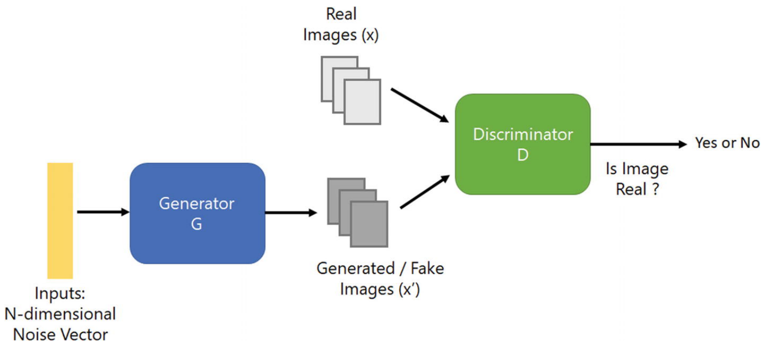
Autoencoders - Example (from kaggle.com)

Using the trained autoencoded we can generate new digits using (x, y) as inputs



Generative Adversarial Networks - Simple GAN

In its simplest form, a GAN learns to generate data from an empirical distribution:



Generative Adversarial Networks - Simple GAN

Training a GAN can be formulated as a game:

- The true distribution is represented by an empirical distribution given as p_{data}
- The adversarial distribution is p_g, which is obtained by transforming noise z drawn from $p_z(z)$ by a neural network $G(z; \theta_g)$
- The detector is a neural network $D(x; \theta_d)$, that outputs the probability of the data x being drawn from the *true* distribution $p_{\text{data}}(x)$, and not $p_g(x)$
- We train $\{\theta_g, \theta_d\}$ jointly by playing the minimax game

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))]$$

Thank you for your attention

NTNU | Norwegian University of Science and Technology