Neural nets

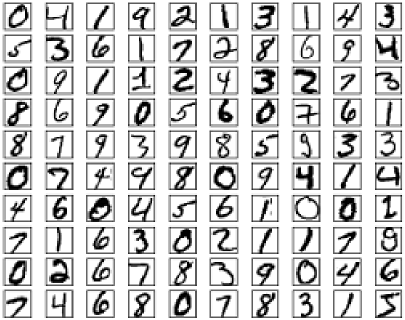
Aurélie Herbelot

University of Trento
Centre for Mind/Brain Sciences

Trento 2016

Introduction

Neural networks: a motivation



How to recognise digits?

- Rule-based: a ‘1’ is a vertical bar. A ’2’ is a curve to the right going down towards the left and finishing in a horizontal line...
- Feature-based: number of curves? of straight lines? directionality of the lines (horizontal, vertical)?
- Well, that’s not gonna work...

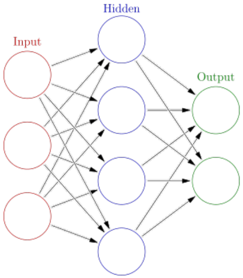
Learning your own features

- We don't know what people pay attention to when recognising digits (which features to use).
- Don't try to guess. Just let the system decide for you.
- A nice architecture to do this is the neural network.

A simple introduction to neural nets

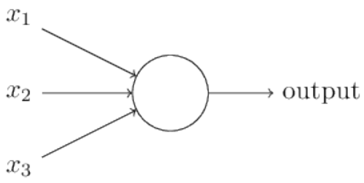
Neural nets

- A neural net is a set of interconnected neurons organised in 'layers'.
- Typically, we have one input layer, one output layer and a number of hidden layers in-between.



By Glosser.ca - Own work, Derivative of File:Artificial neural network.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24913461

The perceptron: an artificial neuron



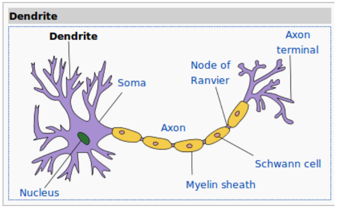
- The output of the neuron is given by:

$$y = \varphi\left(\sum_{j=0}^{m} w_j x_j\right) \tag{1}$$

- If this output is over a threshold, the neuron ‘fires’.

Comparison with a biological neuron

- Dendrite: Take input from other neurons (>1000). Acts as an input vector.
- Soma: The equivalent of the summation function. The (positive and negative – exciting and inhibiting) ions from the input signal are mixed in a solution inside the cell.
- Axon: The output, connecting to other neurons. The axon transmits a signal once the the soma reaches enough potential.



By Quasar Jarosz at English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=7616130

A (simplified) example

- Should you bake a cake? It depends on the following features:
  - Wanting to eat cake (0/+1)
  - Having a new recipe to try (0/+1)
  - Having time to bake (0/+1)
- How much weight should each feature have?
  - You like cake. Very much. Weight: 0.8
  - You need practice, as become a pastry chef is your professional plan B. Weight: 0.3
  - Baking a cake will take time away from your computational linguistics project. Weight: 0.1

A (simplified) example

- We'll ignore φ for now, so our equation for the output of the neuron is:

$$y = \sum_{j=0}^{m} w_j x_j \tag{2}$$

- Assuming you want to eat cake (+1), you have a new recipe (+1) and you don't really have time (0), our output is: $0.8 * 1 + 0.3 * 1 + 0.1 * 0 = 1.1$
- Let's say our threshold is 0.5, then the neuron will fire (output 1). You should definitely bake a cake.

From threshold to bias

- We can write $\sum_{j=0}^{m} w_j x_j$ as the dot product $\vec{w} \cdot \vec{x}$.
- We usually talk about bias rather than threshold – which is just a way to move the value to the other side of our inequality:
  - if $\vec{w} \cdot \vec{x} > t$ then 1 (fire) else 0
  - if $\vec{w} \cdot \vec{x} - t$ then 1 (fire) else 0

Neural nets as machine learning algorithm

- NNs are typical ML algorithms:
  - Our problem is to predict output *y* given input *x*.
  - We give the NN lots of *training* examples (instances for which we know the output).
  - The system learns appropriate weights for each neuron in the network: its *parameters*.
  - Test on unseen examples.

But hang on...

- Didn't we say we didn't want to encode features? Those inputs look like features...
- Right. In reality, what we will be inputting are not human-selected features but simply a vectorial representation of our input (see previous classes on representing words and images as vectors).
- Typically, we have one neuron per value in the vector.
- Similarly, we have a vectorial representation of our output (which could be as simple as a two-element vector representing a binary decision).

How about real-valued input?

- The values in the vectors we are dealing with are not normally between 0 and 1.
- Don’t use a perceptron but a sigmoid neuron, which takes real-valued input and also outputs real values (not just 0 or 1).
- This representation is less sensitive to changes in weights and thresholds.

The role of the hidden layer

Recognising a 9

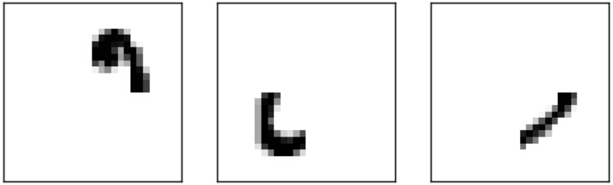
- Let's assume that the image is a 64 by 64 pixels image (4096 inputs, with a value between 0 and 1).
- The output layer has just one single neuron: an output value > 0.5 indicates a 9 has been recognised, < 0.5 there is no 9.
- What about the hidden layer?

The hidden layer

- The hidden layer allows the network to make more complex decisions.
- Intuition: the first layer processes the input and extracts some preliminary features, which will themselves be used by the second layer, etc.
- Setting the parameters of the hidden layer(s) is an art... For instance, number of neurons.

The hidden layer: example

- A hidden layer neuron might learn to recognise a particular element of an image:



- If this neuron fires, then that element has been found in the image.
- By learning which elements are relevant to recognising numbers in the hidden layer, the network can produce a system which, given an input image, identifies the relevant ‘features’ (whatever those should be) and maps certain combinations to a particular digit.

How does learning work?

Calculating the error

- We have a set a training examples (*n* instances with input and desired output).
- We want to set the parameters of the network (weights and biases).
- We feed the *n* examples to the network and see what the current parameters output.
- Using a ‘cost function’, we calculate how far the predictions are from the ‘real’ values.

The cost function

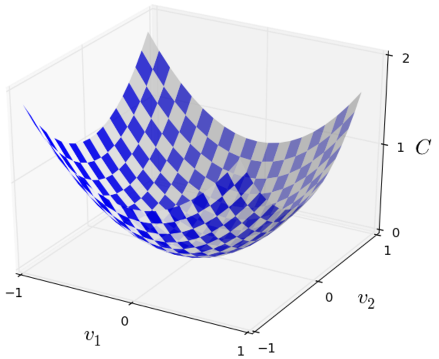
- A typical cost function is the mean squared error (MSE):

$$C(w,b) = \frac{1}{2n}\sum_x ||y(x) - pred||^2 \tag{3}$$

- The cost is small when $y(x)$ is approximately *pred*.

Gradient descent

- The gradient descent algorithm: find the state of the network that gives the minimum error.
- I.e. find the minimum of the cost function.
- Here is some function defined in terms of two variables:

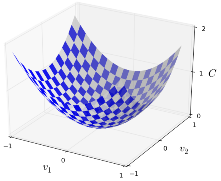


Gradient descent

- To find the minimum of a function, we can just do maths.
- But... a NN typically has *a lot* of parameters, which makes the maths computationally demanding.
- I.e. the *pred* term below might be very complex:

$$C(w,b) = \frac{1}{2n}\sum_x ||y(x) - pred||^2 \tag{4}$$

Gradient descent



- Imagine that we are somewhere on the mountains near this valley, and try to get to the bottom as fast as possible (like a rolling ball).
- One way to do it is to take a step and see whether this step is ascending or descending. Then repeat.
- Using derivatives, calculus has a way to tell us the difference in cost when we move in one particular direction. So we can choose where to go.

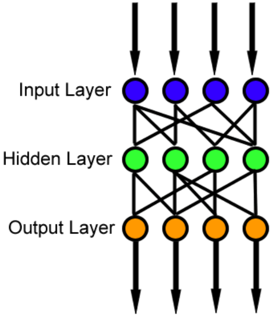
Gradient descent

- When to stop? If we take too big steps down the valley, we might overshoot the minimum.
- Set the *learning rate* μ so that our steps are small enough.
- Once our cost cannot decrease anymore, we assume we have found our minimum and the network's parameters are optimal.

Network architectures

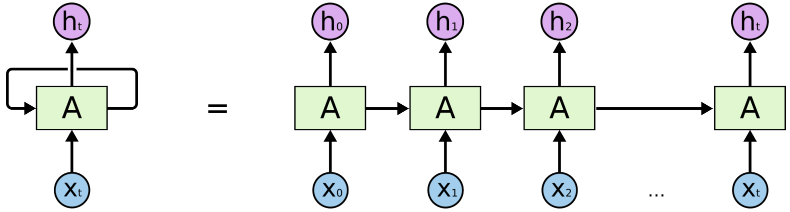
Feedforward networks

- In a feedforward NN, information flows from input to output. Each layer feeds information into the next layer. No loop.



Recurrent Neural Networks

- The problem with feedforward networks is that they don't have persistence: they process each input independently of the previous input.
- In language, we take into account the previous word to understand the new one: *Kim has a cat and a...*
- RNNs take time information into account.



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory Networks

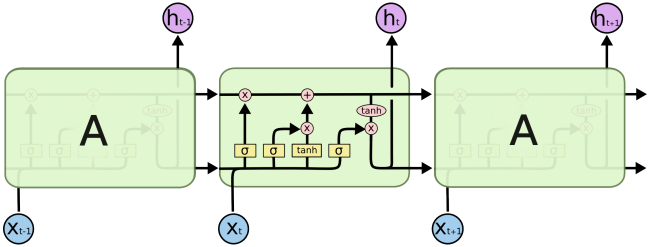
- Problem with RNNs: they don't have such a long memory!
- **Example:** *Kim is now going out with Sandy, who's crazy about horses. They've moved to the countryside. So now, Kim has a cat and a...*
- It isn't possible to set an RNN's parameters so that they remember those long-term dependencies.

Long Short Term Memory Networks

- An RNN has just one layer in its repeating module.
- An LSTM has four layers that interact, each one with a gate. Gates are ways to let information through (or not):
  - Forget gate layer: look at previous cell state and current input, and decide which information to throw away.
  - Input gate layer: see which information in the current state we want to update.
  - Update layer: propose new values for the cell state.
  - Output layer: Filter cell state and output the filtered result.

Long Short Term Memory Networks

- **Example:** *Kim is now going out with Sandy, who's crazy about horses. They've moved to the countryside. So now, Kim has a cat and a...*
- We can remember stuff about horses and countryside, and forget about going out or moving.



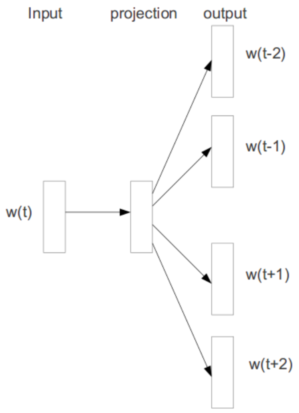
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Word2Vec

The intuition behind Word2Vec

- Up to now, we have looked at ‘count-based’ models of distributional semantics.
- That is, we built our word vectors by counting co-occurrences of words which each other.
- Word2Vec (Mikolov et al 2013) is a neural network, *predictive* model. It has two possible architectures:
  - given some context words, predict the target (CBOW)
  - given a target word, predict the contexts (Skip-gram)
- In an NN setting, distributional vectors are usually referred to as *embeddings*.

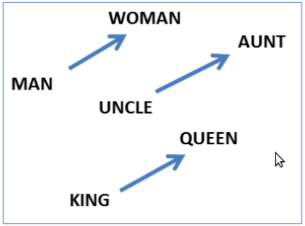
The Skip-gram model

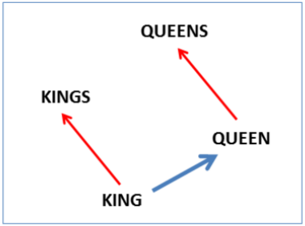


Features of Word2Vec representations

- A representation is learnt at the reduced dimensionality straightaway: we are outputting vectors of a chosen dimensionality (parameter of the system).
- Usually, a few hundred dimensions: dense vectors.
- The dimensions are not interpretable: it is impossible to look into ‘characteristic contexts’.
- For many tasks (but not all!) outperform count-based vectors. But it is possible to ‘tweak’ count-based vectors to make them work like predict ones.

What Word2Vec is famous for





The actual components of Word2Vec

- A vocabulary. (Which words do I have in my corpus?)
- A table of word probabilities.
- Negative sampling: tell the network what *not* to predict.
- Subsampling: don't look at all words and all contexts.

Negative sampling

- Whenever considering a word-context pair, also give the network contexts which are not the actual observed word.
- Sample from the vocabulary. The probability to sample something more frequent in the corpus is higher.
- The number of negative samples will affect results.

Subsampling

- Instead of considering all words in the sentence, transform it by randomly removing words from it:
  considering all sentence transform randomly words
- The subsampling function makes it more likely to remove a frequent word.
- This kind of recreate the PMI effect in a count-based model.
- Note that this affects the window size around the target.

Criticisms of neural networks

The black box criticism

- Because neural nets learn their own features and parameters, through the use of one or more ‘hidden’ layers, it is not always easy to know what they are learning (or not).
- In some cases, it is possible to ‘look’ inside the hidden layers, but that’s not always the case.

Real-life parameters

```
– Default configuration
config =
config.corpus = "corpus.txt" – input data
config.background = "none" – background space
config.window = 5 – (maximum) window size
config.dim = 100 – dimensionality of word embeddings
config.alpha = 0.75 – smooth out unigram frequencies
config.table_size = 1e8 – table size from which to sample neg samples
config.neg_samples = 5 – number of negative samples for each positive sample
config.minfreq = 10 –threshold for vocab frequency
config.lr = 0.025 – initial learning rate
config.min_lr = 0.001 – min learning rate
config.epochs = 3 – number of epochs to train
config.gpu = 0 – 1 = use gpu, 0 = use cpu
config.stream = 1 – 1 = stream from hard drive 0 = copy to memory first
config.batch_size = 5
```