Lecture 9

Gidon Rosalki

2025-12-14

1 Introduction

This is probably the last year where NN will be taught in image processing. This is because it is mostly covered in other courses. We will only briefly discuss it, one lecture, and two tutorials. There will still be a homework, but it will be less covered in classes.
Neural networks are parts of most computer vision systems today. They have enabled many tasks, previously thought impossible. It's great with tasks where we have available examples, and without well defined algorithms. Consider things like image recognition / classification. It is easy for humans to create many examples, that the computers may then learn. To be a clear, a well defined algorithm, or an exact equation are preferred to Neural Networks where possible, like solving equations, geometric computations, physics principles, and so on. As a result, in most cases, NNs are part of a solution, not the entire solution.
We are recommended to watch from the *Graphics in 5 minutes* course

- Large Language Models from Scratch
- Large Language Models part 2
- Text to Image in 5 minutes
- Text to image part 2

1.1 Example

A common example is to learn examples from a function f, such that it can then emulate the function. This is good when it is hard to hand craft an example of f, but we can generate examples from which the NN can learn (like classifying if an image is of a man, or a woman). The model may then be tested on a labelled test set.

2 Machine learning frameworks

We are looking to create a model F such that

$$Y = F(x)$$

Where Y is the output for x which may be an image or something, and F predicts what the output should be.
To achieve this, there are two types of data:

1. Training: Given a training set of labelled examples $\{(x_1, y_1), \dots, (x_n, y_n)\}$, estimate the prediction function F by minimising the *prediction error* on the training set.
2. Testing: Apply F to a never seen before test set x and output the predicted value $y = F(x)$

For LLMs, instead of giving them some x like an image, we instead give a large corpus of text. They may be trained through a set of all the texts available from the internet, estimate the prediction function F of predicting the next word given the prior text. No need for labelling.

2.1 Mathematics

Systems may learn functions, for example, a linear regression. This may be done by minimising a cost function, like

$$E_1 = \sum_i (y_i - (a_0 + a_1 x_i))^2$$

Or from the line equation:

$$E_2 = \sum_i (a_0 + a_1 x_i + a_2 y_i)^2$$

Most data ($\approx 80\%$) used for training, and some for testing ($\approx 20\%$). It is important to see that both training, and test errors go down. Training error should always go down (since we are minimising on it), and test error will go down if the learned function is generalised beyond the training set.

We do need to ask about memorisation, vs generalisation. Or, how well does the learned function extend beyond the training set to a new test set.

2.2 Neural networks

These are comprised of artificial neurons. Artificial neurons are comprised of a series of inputs, x_i, multiplied by weights, w_i, brought together into a transfer function (like a sum), and passed through an activation function φ. This may be given as

$$o_j = \varphi\left(\theta_j + \sum_{k=0}^{n} x_k w_{kj}\right)$$

Two examples of activation functions are

- Sigmoid: $\varphi(z) = \dfrac{1}{1+e^{-z}}$
- ReLU: $\varphi(z) = \max\{0, z\}$

In a Neural Network, even a single hidden layer can approximate any function from input to output (given enough neurons). We may use this for classification, where we give it an input (an image), and receive a single neuron output (what figure is drawn in the image). We can also perform regressions, where given a blurred image, we receive pixel values as output, which are a sharpened version of the image.

2.2.1 Deep neural networks

These have multiple hidden layers, rather than just one Each layer i is **fully connected**, meaning that every node in layer i is connected to every node in the layer $i \pm 1$

A special type would be an autoencoder. For these, the output size is the same as the input, one of the hidden layers has far fewer nodes than the input. This forces a compact representation of the image for image comparison, and removes noise, since random noise cannot be compressed into this representation. There may also be bias nodes, that always output 1.