395 Machine Learning
— Assignment 2 —

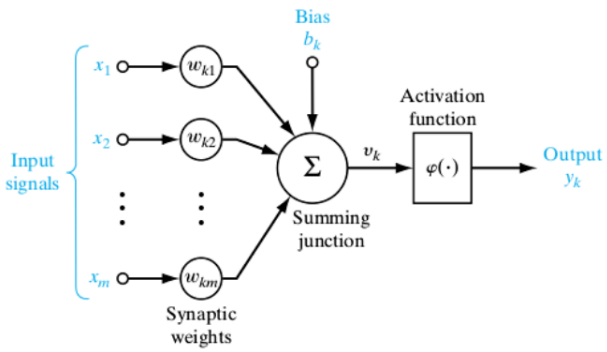
Group 11
Malon AZRIA, Alexandre CODACCIONI, Benjamin MAI, Laura HAGEGE.
ma12917@ic.ac.uk, afc17@ic.ac.uk, bm2617@ic.ac.uk, lmh1417@ic.ac.uk
CBC helper: Pingchaun MA
Course: CO395, Imperial College London

1st March, 2018

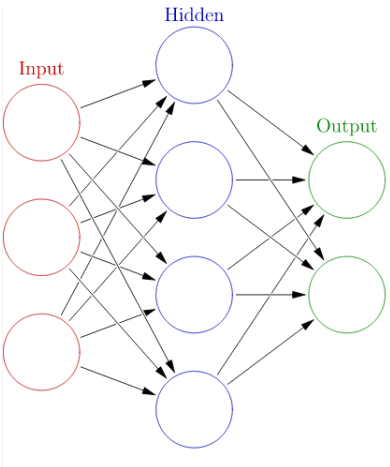
Introduction

The aim of this coursework is to implement and train a fully connected neural net to classifiy emotions using images of faces. There are 7 emotions, namely: anger, disgust, fear, happiness, sadness, surprise and neutral. This report discusses our approach and findings. First, we discuss the theory behind neural nets. Next, we explain how we tuned our neural net's hyperparameters and provide an analysis of our model using metrics such as confusion matrices or recall rates. We conclude with an exploration of neural net state of the art explorat

But first, what is neural network? Neural networks aim to mimic the biological neural network present in the human brain. Let us begin by explaining the concept of a neuron.



(a) neuron



(b) neural net architecture

A neuron takes in several inputs and produces a single output, as represented in the picture above. A neuron sums all inputs weighted by their respective weights, adds a bias and then inputs this into an activation function to produce a single output.

A neural network is is composed of layers of neurons, such that each neuron's output is an input to all of the next layer's neurons. The neural net architecture diagram above, shows three different types of layers. An input layer contains a neuron for each data input's elements, which simply outputs the element it is given. A hidden layer's neuron has a weight for each of the inputs connected to them,

and produces a single output as described earlier. An output layer has an neuron for each output you wish to train your net to predict. In this coursework, when we train a net to predict the emotion of a given image, the net's output layer will have 7 neurons, one for each emotion. Now that we have the basics down, let us now explore how a neural net works.

Q1: Explain the forward and backward pass of linear layers and relu activations

How does a neural net generate an output or in our case, how does it decide which emotion to recognize? The output is produced in a process known as the emphforward pass. Instead of processing one data point at a time, we have implemented our neural net to process data points in batches. As such, the forward pass takes in N examples in a matrix X, where each row contains a data point of dimension D. Let us take a closer look at the neural net's inner workings, assuming it has already been trained. Note that although all the math involved in this coursework is in vector form, I find it more intuitive to illustrate my explanations using scalar equations. Please assume the following mathematical notations for the rest of the report, unless stated otherwise.
- w_{ij} is the weight from neuron i to neuron j.
- b_j is the bias of a neuron j.
- x_i is the i^{th} input of a neuron.

- **Affine transformation**

  Forward pass: $o_j = \sum_{i=1}^{I}(w_{ij} \times x_i) + b_j$,
  When a layer goes through the affine transformation step, every neuron in the layer sums its I inputs weighted by their respective weight, then adds its bias.

  Backward pass: $\frac{\partial o_j}{\partial x_i} = w_{ij}, \quad \frac{\partial o_j}{\partial w_{ij}} = x_i, \quad \frac{\partial o_j}{\partial b_j} = 1$
  For a neuron j in a layer that has just gone through forward affine transformation, we want to compute the derivative of its output o_j with respect to each of its weights, and bias. Take a look at the formulas derived above and notice that I have include $\frac{\partial o_j}{\partial x_i}$. While we are not directly interested in how o_j changes with respect to its inputs, we compute it for other layers that are. Remember that the I inputs of a neuron j are actually the outputs of the previous layers' neurons. Using the chain rule, the previous layer will use $\frac{\partial o_j}{\partial x_i}$ to compute its backward affine transformation for the previous layers.

- **ReLU activation function**

  So what does a neuron do? It computes a weighted sum of its input and must now decide whether it should be "fired". Earlier we spoke of an activation function. It is the activation that decides decides where the neuron should "fire" or not. Without an activation function, a layer's output would simply be a simple linear transformation. Linear transformation are limited in their complexity and have a lesser ability to learn complex functional mappings from data. A neural network without an activation function would simply be a multivariate linear regression model. There exists many activation functions such as the sigmoid function, but in this coursework, we use the function known the ReLU activation function.

  Forward pass: $ReLU(o_j) = max(0, o_j)$
  The ReLU function outputs the neuron's sum if it is greater than 0 or simply outputs 0.

  *Note that only hidden layers go through ReLU. Output layer's neurons always "fire" and output their sum or "score".

  Backward pass: $\frac{\partial ReLU(o_j)}{\partial o_j} = 1$ if $o_j < 0$, else 0.
  This partial derivative is self-explanatory.

Q2: Explain the forward and backward pass of dropout

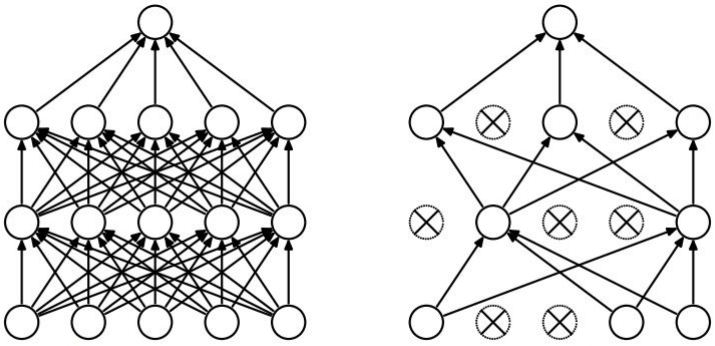
As we create and train the network Dropout layer is used for regularization. Using dropout consist in dropping randomly some neuron from the network. Those neuron are consider ar "turned off" meaning they won't impact the futur weight updates.
While training on one example neuron can co-adapt on becoming really specific to an example. Using dropout regularization we prevent then the neuron to co-adapt too much and the network to be less specific and better in generalization. Dropout prevent then from overfitting and to be stuck in a local minimum of the error gradient.

- **Forward pass :**

  During the forward pass we are modifying the network itself, neuron are dropped from the network with the probability p (and kept with propbability q=1-p) - each neuron has the same probability to be dropped or not. To do so we used a a binomial distribution $\mathcal{B}$(N,p) to decide wether a neuron is dropped or not.
  If we are using dropout during the training phase we need to perform a rescaling (x $\frac{1}{1-p}$) on the neuron in order to make the network operational for testing.

- **Backward pass :**

  During the backpropagation we just need to consider the Dropout, neuron that have been dropped doesn't contribute to further updates, they consquently need to be dropped in the derivative matrix - coresponding weight won't be updated. Therefor mask need to be stored between forward an backward pass to know which neuron have been dropped.



(a) Standard Neural Net (b) After applying dropout.

Q3: Explain the computation of softmax and its gradient

Q4: Training on the cifar10 dataset

Q5: Tuning the hyperparameters

Conclusion