Sebastian Borgeaud dit Avocat

Brain tumour segmentation using Convolutional Neural Networks

Computer Science Tripos – Part II

Fitzwilliam College

March 18, 2017

Proforma

Name:	**Martin Richards**
College:	**St John's College**
Project Title:	**How to write a dissertation in LaTeX**
Examination:	**Computer Science Tripos – Part II, July 2001**
Word Count:	**1587**[1] **(well less than the 12000 limit)**
Project Originator:	Dr M. Richards
Supervisor:	Dr Markus Kuhn

Original Aims of the Project

To write a demonstration dissertation[2] using LaTeX to save student's time when writing their own dissertations. The dissertation should illustrate how to use the more common LaTeX constructs. It should include pictures and diagrams to show how these can be incorporated into the dissertation. It should contain the entire LaTeX source of the dissertation and the makefile. It should explain how to construct an MSDOS disk of the dissertation in Postscript format that can be used by the book shop for printing, and, finally, it should have the prescribed layout and format of a diploma dissertation.

Work Completed

All that has been completed appears in this dissertation.

Special Difficulties

Learning how to incorporate encapulated postscript into a LaTeX document on both Ubuntu Linux and OS X.

[1] This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

[2] A normal footnote without the complication of being in a table.

Declaration

I, [Name] of [College], being a candidate for Part II of the Computer Science Tripos [or the Diploma in Computer Science], hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed [signature]

Date [date]

Contents

List of Figures

Acknowledgements

This document owes much to an earlier version written by Simon Moore [**?**]. His help, encouragement and advice was greatly appreciated.

Chapter 1

Introduction [14%]

Convolutional Neural Networks have gained an enormous amount of traction in recent years, as they have been used in many different areas to obtain state-of-the-art results. The areas of computer vision and biomedical imaging analysis are no exception to this and the number of entries using Convolutional Neural Networks in the BraTS [CITATION] challenge has risen accordingly, with some of these entries being at the top of the evaluation board.

Chapter 2

Preparation [26%]

During the first phase of my project, the aim was to replicate the method used by Pereira et al [1], so it was crucial to first fully understand the steps taken in the paper to then be able to reimplement them. Unfortunately, the paper didn't include any source code which meant that if something wasn't fully explained in details, I would have to find out what was actually done. This turned out to be a problem for the pre-processing step as the proposed method uses a normalisation developed by Nyul [CITATION]. This normalisation requires human input, preferably from a domain expert, and I was therefore not able to use that normalisation method. The paper also proposed a second normalisation method, which used a combination of winsorizing and N4 normalization. This method performed slightly worse but had the advantage of being fully automated, which is why I chose to use this method.

2.1 Convolutional Neural Networks

Neural networks

To understand how convolutional neural networks work, it is important to be familiar with ordinary neural networks. These are made up of layers of neurons that have trainable weights in a sequence of layers. An example of the structure of such a neural network can be found in figure 2.1.

Each neuron in layer $n+1$ is connected to every neuron in layer n and computes as an output

$$y = f_{act}((\sum_{i=1}^{n} y_i w_i) + b)$$

where f_{act} is a non-linear, differentiable activation function and y_i is the output of neuron i in the previous layer. A neuron is connected to every neuron in the previous layer, which is why this layer is also referred to as a fully connected layer.

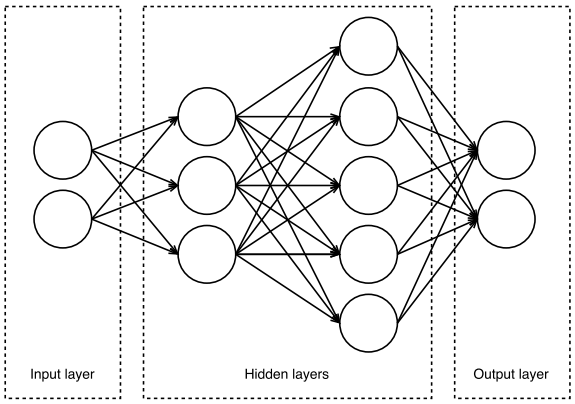

Figure 2.1: Structure of a simple neural network with two hidden layers.

Activation functions

The most common activation functions are the Sigmoid function,

$$S(x) = \frac{1}{1 + e^x}$$

the hyperbolic tangent

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

the rectifier

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

and the leaky rectifier, for some $0 < \alpha < 1$

$$f(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

This functions can be seen in figure 2.2 Historically the hyperbolic tangent function or the sigmoid function have been used as activation functions. However, as the magnitude of the gradients of those functions is always below 1, these activation function create a problem of vanishing gradients for deeper networks, as we have to multiply those gradients together for each layer. [CITATIONS!!!!] This is why it is now preferred to use the rectifier or the leaky rectifier functions. [PLOT OF FUNCTIONS]

Loss function

The next step is to compute how well our network approximates our training data with a loss function, to then decide how to change the weights of the network in order to minimise the loss. Since the aim of the network is to classify the central pixel(s) of the patch, we use the categorical cross-entropy loss function

$$\mathcal{L}(\theta) = \frac{1}{m}\Big[\sum_{i=1}^{m}\sum_{j=1}^{k} \mathbb{1}[y^{(i)} = k] \log(P(y^{(i)} = k \mid x^{(i)}; \theta))\Big] \tag{2.1}$$

where $\mathbb{1}$ is the indicator function, $\log(P(y^{(i)} = k \mid x^{(i)}; \theta))$ is the probability outputted by the network.

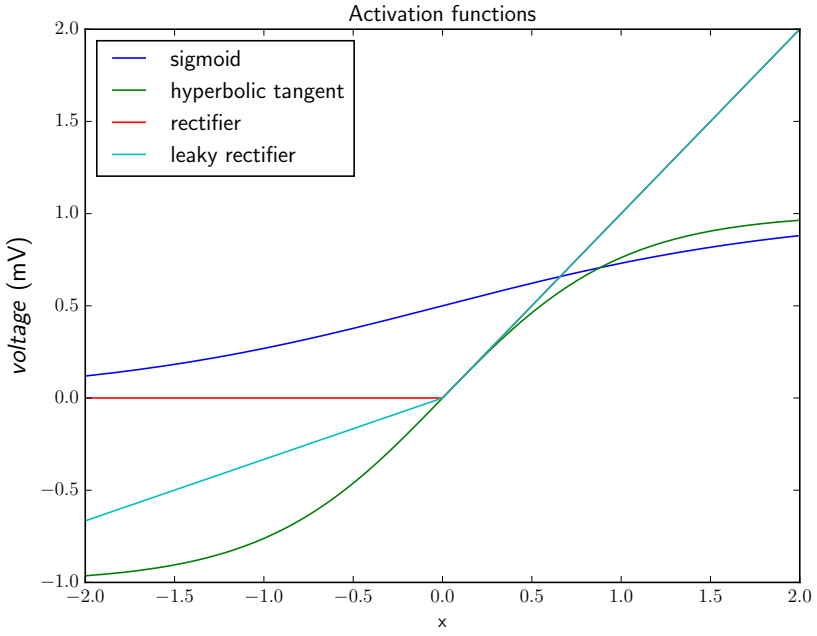

Figure 2.2: Plot of 4 different activation functions.

Backpropagation

Since every basic operation used in the neural network is differentiable, the entire network will also be differentiable, which in turn makes it possible to calculate the gradient of a loss function with respect to the weights in the network. This then will allow us to minimise this loss function by using a gradient descent approach.

SGD

Momentum

Further optimizations

2.1.1 Adding convolutions

Convolutional neural networks are similar to ordinary neural networks, sometimes also called artificial neural networks. Convolutional neural networks are also made up for neurons that have trainable weights. The main difference is that

2.1.2 Dropout

2.1.3 Batch Normalization

2.2 Data pre-processing

2.2.1 N4ITK

2.2.2 Patch extraction

2.3 Data source

Chapter 3

Implementation [40%]

3.1 CNN models

I implemented the convolutional neural networks using the Keras (Add reference) library, which allows users to create neural networks by combining different layers together. The more common layers come as part of the library but Keras allows the user to define layers as well. The architecture proposed by Pereira et al [1] used the following layers:

- two-dimensional convolutional layers
- fully connected layers
- two-dimensional max pooling layers

Futhermore, we can easily add Dropout to the model by adding an instance of a Dropout layer where required.

My implementation of the model proposed by Pereira [1] looks as follows in Keras:

```
1   model = Sequential()
2
3   model.add(Convolution2D(64, 3, 3, border_mode='same', init=init,
    input_shape = input_shape, W_regularizer=l2(l)))
4   model.add(LeakyReLU(alpha))
5
6   model.add(Convolution2D(64, 3, 3, border_mode='same', init=init,
    W_regularizer=l2(l)))
7   model.add(LeakyReLU(alpha))
8
9   model.add(Convolution2D(64, 3, 3, border_mode='same', init=init,
    W_regularizer=l2(l)))
10  model.add(LeakyReLU(alpha))
11
12  model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), border_mode='
    valid'))
13
14  model.add(Convolution2D(128, 3, 3, border_mode='same', init=init,
    W_regularizer=l2(l)))
15  model.add(LeakyReLU(alpha))
16
```

```
17     model.add(Convolution2D(128, 3, 3, border_mode='same', init=init,
       W_regularizer=l2(l)))
18     model.add(LeakyReLU(alpha))
19
20     model.add(Convolution2D(128, 3, 3, border_mode='same', init=init,
       W_regularizer=l2(l)))
21     model.add(LeakyReLU(alpha))
22
23     model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2), border_mode='
       valid'))
24
25     model.add(Flatten())
26
27     model.add(Dense(256, init=init, W_regularizer=l2(l)))
28     model.add(LeakyReLU(alpha))
29     model.add(Dropout(0.1))
30
31     model.add(Dense(256, init=init, W_regularizer=l2(l)))
32     model.add(LeakyReLU(alpha))
33     model.add(Dropout(0.1))
34
35     model.add(Dense(5, init=init, W_regularizer=l2(l)))
36     model.add(Activation('softmax'))
```

This makes it easy to create deep, convolutional networks and to experiment with them. This reason, together with the active community of Keras users is why I chose to use the Keras library.

Before training the models, we further need to specify which loss function and which algorithm should be used to respectively specify and minimise the loss function. Again, keras makes this very simple:

```
1     sgd = SGD(lr = 3e-5, decay =0.0, momentum = 0.9, nesterov = True)
2     model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=[
      'accuracy', dice])
```

3.2 Data preprocessing

3.3 Training

3.4 Segmentation

Chapter 4

Evaluation + Conclusion [20%]

4.1 Metrics used for the evaluation

4.1.1 Dice score

4.1.2 Positive predictive value

4.1.3 Sensitivity

4.2 Evaluation of the model proposed by Pereira et al.

4.3 Evaluation of the my model

4.4 Comparison

Chapter 5

Conclusion

I hope that this rough guide to writing a dissertation is LaTeX has been helpful and saved you time.

Bibliography

[1] Sergio Pereira, Adriano Pinto, Victor Alves, and Carlos A. Silva. Brain tumor segmentation using convolutional brain tumor segmentation using convolutional neural networks in mri images. *IEEE Transactions on medical imaging*, 35(5):1240–251, May 2016.