COMPGI23/COMPM089: INDIVIDUAL ASSIGNMENT #2

Due Date: Monday 11th December 2017; 5pm

Please email your answer to n.lane@cs.ucl.ac.uk as PDF (and zip file if needed). Limit your answers to concise single paragraphs per question, include graphs/figures as needed. For questions in which code is written, include all code in a single zip file accompanying your PDF. Within the zip file please include a directory structure where the directory names refer to the questions from the assignment, and the code used for a question is included in the appropriate directory.

At the top of your PDF please include your name, student ID, and UCL email address.

There are 4 questions, and a total of 50 marks for this assignment. **Note, not all questions are worth the same number of marks.**

Question 1: Training/Optimization (16 marks)

Recall the simple neural network from Assignment 1 Question 6 that used the MNIST dataset and the following architecture:

- Input layer: 784 values
- Hidden layer 1: 100 fully-connected neurons, with sigmoid activation
- Hidden layer 2: 50 fully-connected neurons, with ReLU activation
- Output layer: 10 fully-connected neurons, with softmax activation

Task. In this question you will revise your earlier code and focus on your own implementation of key parts of the optimization process (i.e., network training). This is designed to get your hands dirty and gain a greater appreciation of this process. For this question your code must include the following elements coded by hand (i.e., not using library support within Keras or TensorFlow):

1. calculate cross-entropy loss;
2. use the backpropagation algorithm to calculate the gradients of this loss with respect to the network weights and update the weights accordingly;
3. randomly initialise the weights
4. train the network architecture described above on the MNIST dataset for several epochs. (The number of epochs is to be determined by an easily altered parameter in your code.)

In addition to submitting code that performs the above logic, please also include a figure that is produced by executing this code that shows the change in loss at each epoch during training. Compare in this figure the same per-epoch evolution of loss under an optimizer built into Keras (please select any optimizer you wish). Briefly comment and contrast regarding the behavior of your implementation and the one selected from Keras, as seen in the figure and more broadly.

Notes. While you may use the TensorFlow/Keras utility functions for loading MNIST data, but do not use library functions for the optimzation stages mentioned above – as an example, TensorFlow's features for gradient calculation or training. While answering this question, you might find it necessary or just easier to re-implement parts of the original network and code that previously relied certain helper-functions. As a source of reference material, you may also find it useful to consult **Chapter 6 and 8** of The Deep Learning Book.

Question 2: Improving Network Accuracy (16 marks)

The aim of this question is to gain experience in some of the techniques discussed in Lecture 6 towards improving the accuracy of a deep network architecture.

Start by cloning the following repository: `https://github.com/PetarV-/L42-Starter-Pack`. Follow the instructions in the README of this repository to setup and execute the model.

Task. Modify the model architecture to exceed 92% accuracy on the validation set by applying some of the techniques described in Lecture 6 to parts of the architecture (for example, dropout and batch normalisation to name just two.)

To answer this question provide: (i) a description of techniques adopted that enabled you to exceed the 92% level; (ii) precisely how they are applied within the architecture; and (iii), justify the design decisions you made.

A bonus 5 marks will be awarded to those who reach 95% accuracy (but please treat this goal only as an optional extension to this question if you have both the time and interest.)

Question 3: Exploding and Vanishing Gradients (6 marks)

Let us clarify first of all the following notation for this question: Capital letters like $\mathbf{A}$ indicate matrices while small letters like $\mathbf{x}$ indicate vectors. We let $\mathbf{W}^\dagger$ denote the transpose or conjugate transpose (for real and complex matrix respectively). In the following we define the norm of a matrix to refer to the spectral radius norm (or operator 2-norm) and the norm of a vector to denote the L_2-norm, which should be familiar from calculus.

Recall that in the Lecture 6 (slide 47) we discussed the Orthogonal Initialization of the weights, i.e. an initialization of the weight matrix $\mathbf{W}$ such that $\mathbf{W}^\dagger\mathbf{W} = \mathbf{W}\mathbf{W}^\dagger = \mathbf{I}$. We mentioned that this transformation helps to combat vanishing or exploding gradients. You will find this issue is described in Chapters 8 and 10 of The Deep Learning Book, and specific discussions present in subsections: **8.2.5** and **10.7**. It is likely benefical to review this material before proceeding.

In this question, we will guidance that will allow you to analyze why such weights $\mathbf{W}$ help to avoid these issues in more general, from a more theoretical perspective.

Recall that orthogonal (in particular unitary matrices for complex numbers) matrices have the useful property to preserve the norm, i.e. $\|\mathbf{W}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$, and hence repeated iterative multiplication of a vector by an orthogonal matrix leaves the norm of the vector unchanged.

These facts will allow us to derive a formal explanation of the vanishing or exploding gradient problem. Note that this won't require you to do any 'advanced' matrix calculus, but simply the application of given relationships!

Assume a network with n hidden layers. Let $\mathbf{h}_i$ be the output vector of layer i where $\mathbf{h}_{i+1} = \sigma(\mathbf{W}_i\mathbf{h}_i + \mathbf{b}_i)$, where $\sigma(\cdot)$ is the pointwise non-linear (e.g. sigmoid) function. If C is the objective we are trying to minimize, then the vanishing and exploding gradient problems refer to the decay or growth of $\partial C/\partial \mathbf{h}_t$ as the number of layers, n, grows.

Task. We ask you to derive a bound for the growth of the norm of this gradient of the cost function, i.e. we will try to bound

$$\left\| \frac{\partial C}{\partial \mathbf{h}_t} \right\|. \tag{1}$$

It is recommended you approach this task by following the basic steps described below. We will supply you with some useful tips to calculate the gradient, but basic knowledge of linear algebra (like the chain rule) is required. **Your answer to this question should focus on working through each of the steps now described.**

1. Begin by applying the chain rule to equation 1. Start by expanding
$$\frac{\partial C}{\partial \mathbf{h}_t} = \frac{\partial C}{\partial \mathbf{h}_n}\frac{\partial \mathbf{h}_n}{\partial \mathbf{h}_t} \tag{2}$$
and now expand the latter term in terms of the intermediate layers $n-1, \ldots, t+1$ which will give you a product of the equation. Do not evaluate any of the derivatives yet!

2. Now take the plug in the derivatives
$$\frac{\partial \mathbf{h}_{i+1}}{\partial h_i} = \mathbf{D}_{i+1}\mathbf{W}_i^\dagger,$$
where $\mathbf{D}_{i+1} = \mathrm{diag}(\sigma'(\mathbf{W}_i\mathbf{h}_i + \mathbf{b}_i))$, i.e. a matrix with diagonal entries according to the vector outcome of $\sigma'(\cdot)$.

3. Use the following identities in this step to prove a bound on equation 1 building on the above expansion and now (!) using the fact that orthonormal matrices preserve the norm:

   - For any vector $\mathbf{x}$ and matrix $\mathbf{M}$ and vector $\mathbf{x}$ we have that $\|\mathbf{Mx}\| \le \|\mathbf{M}\|\,\|\mathbf{x}\|$ and for any matrices $\mathbf{M}, \mathbf{N}$, we have $\|\mathbf{MN}\| \le \|\mathbf{M}\|\,\|\mathbf{N}\|$.

   You should end up with an equation of the form
$$\left\|\frac{\partial C}{\partial \mathbf{h}_t}\right\| \le \|\mathbf{X}_1\| \prod_{k=t}^{n-1} \|\mathbf{X}_2\|,$$
   where the $\mathbf{X}_1, \mathbf{X}_2$ needs to be derived using above results.

4. Now observe that $\|\mathbf{D}_i\| = \max_{j=1,\ldots,n} |\sigma'(\mathbf{W}_i\mathbf{h}_i + \mathbf{b}_i)_j| =: \tau$. Please explain in words (or equations) what happens for large n for the bound on the gradient. Explain here case-wise what happens for $0 \le \tau < 1$ and for $\tau > 1$.

Note that you just derived an explanation why the rectified linear unit (ReLU) nonlinearity is such an attractive choice for the activation function and particularly independent of the depth of the actual neural network!

For those who want to do the actual calculation (purely out of interest, not required by this question) and need a refresher on the matrix derivatives, have a look at: `https://atmos.washington.edu/~dennis/MatrixCalculus.pdf`

Question 4 (12 marks)

Recall in part of Lecture 5, we examined a representative piece of code for training a hot keyword recognition model. An interesting aspect of the design was how each mini-batch was formed. Each batch was not just simply comprised of labeled training data containing examples of spoken words, but would include data that was altered by some transformation (e.g., zero-padding) or even data that did not even contain spoken words at all (e.g., unknown class). Mini-batch generation performed in this way is all in an effort to produce a model that has more robust recognition rates. The aim of this question is to learn more about a few of these techniques, and consider why/how they assist accuracy.

Task. Select three techniques from the list at the end of this question. For each of these techniques provide a seperate relatively brief paragraph that: (i) describes the technique and why it can help model robustness, (ii) discusses if and how it would be applicable outside the context of an audio model. Finally provide paragraph (include a diagram if this is helpful) that describes how mini-batch generation occurs within hot keyword recognition implementation we examined in class, and the role the three techniques you selected fit within it. If any do not occur within the batch generation phase explain where they fit within the overall training process.

Note. If you discover a technique not on the list below that you wish to discuss, please feel free to do so. We recommend consulting the slides from Lecture 5, consulting the code directly, as well as the tutorial from TensorFlow: https://www.tensorflow.org/versions/master/tutorials/audio_recognition. Examining academic paper that build commerical hot keyword recognition models will also provide insights, one mentioned in class is: http://www.isca-speech.org/archive/interspeech_2015/papers/i15_1478.pdf – although there are many, feel free to search and read papers of this type if you have interest. Finally, there are two versions of the code base for this model. We showed a simplified version of the original released by Google. For the purpose of this question using either of these is fine (assuming if inspecting the code is necessary for the particular answer you are providing).

Techniques.

- Data Augmentation
- Class Balancing
- Random Time Shifting
- Zero Padding
- Scaling og Amplitude
- Adding Background Noise
- An *'other'* or *'unknown'* Class
- Controlling Amplitude Across Noise and Signal
- Silence Insertion