CSC321 Winter 2015 - Introduction to Neural Networks
Questions from 2014 midterm

We don't have an electronic copy of the midterm from 2014, so we have transcribed a subset of the questions.

Section A

This section had 6 questions, each worth one mark. These are short questions. Don't write a whole page of text. The main idea, in one or two sentences, is enough.

1. When we were discussing perceptrons, we talked about a way of creating one extra connection weight that has the same effect as having a bias; thus we eliminated the need for a separate bias that's different from the weights. Describe that trick, and say whether or not it can also be applied to multilayer neural networks trained with backpropagation.
2. What do we mean when we say that a particular unit in a neural network is an "invariant feature detector"?
3. What does backpropagation-based learning of a recurrent neural network have in common with backpropagation-based learning of a convolutional neural network? Mention something that does not happen in backpropagation-based learning of regular multilayer neural networks, but does happen in RNN's and CNN's.
4. What is an "autoregressive model" and why do we call it "memoryless"?
5. What is "pooling" in a convolutional neural network?
6. What problem do we run into when we use too large a learning rate? What problem do we run into when we use too small a learning rate?

Section B

This section had 6 questions. We have listed 5 of them. Students were supposed to pick 3 of them to answer, each worth 2 marks. Like in Section A, write brief answers.

1. This is about non-standard types of hidden unit, with error backpropagation. Most neural networks use hidden layers of logistic neurons. However, we could try other types of hidden unit.
   (a) In theory we could use binary threshold units, where y is 1 if $z \geq 0$ and $y = 0$ if $z < 0$. However, this is not a good idea. How would backpropagation-based training suffer if we did this?

(b) We could use rectified linear units, where $y = \max(z, 0)$. In what way do these have the same problem as binary threshold units, and in what way do they not have that problem?

2. This is about what perceptrons can't do. Perceptrons are pretty good, but they can't do everything. If no weight vector exists that gets every training case right, a perceptron will just keep changing its weights forever. In class we studied a classic example of a training set that has this problem. This is for a perceptron with two input units and a bias, and it's a training set of four training cases:

$$\begin{aligned}(0,0) &\to 1\\ (1,1) &\to 1\\ (0,1) &\to 0\\ (1,0) &\to 0\end{aligned}$$

The number after the arrow indicates the intended output.

(a) Why is there no weight vector & bias that gets every case right? Explain either with a mathematical description, or by drawing the relevant diagram and explaining the situation in English.

(b) Let's get rid of the first data point: the one with input values $(0, 0)$. Now a good weight vector & bias can be found. Find one, and make very explicit what the numbers are, below. There is no need to explain, although a brief explanation might earn you partial marks if the numbers you write down are incorrect.

- w_1, the weight from the first input unit to the output unit
- w_2, the weight from the second input unit to the output unit
- b, the bias

3. This is about the softmax formula. Let's look at a softmax group of just two units: unit 1 and unit 2. Both units get an input, which we usually call z, so we have z_1 and z_2. These input values lead to output values called y (y_1 and y_2), with the guarantee that $y_1 + y_2 = 1$.

(a) Write down the formula that describes how y_1 depends on the z values, for this case with just 2 units. There is no need to simplify or manipulate the formula in any way, nor to explain.

(b) Let's say that we fix $z_2 = 0$. Now z_1 is the only remaining input. We're not interested in y_2, but we are interested in y_1. Prove that y_1, as a function of z_1, is the logistic function that we've seen in class. It's a simple derivation, but show clearly what the steps are.

4. This is about the guarantees of learning with error backpropagation. Let's say that we're training a network to read images (28×28 pixels) of handwritten digits. The network consists of $28 \times 28 = 784$ input units, a hidden layer of logistic units, and a softmax group of 10 units as the output layer. We're using the cross-entropy loss function. We have many training cases and we always compute our weight update based on the entire training set, using the error backpropagation algorithm. We use a learning rate that's small enough for all practical purposes, but not so small that the network doesn't learn. We stop when the weight update becomes zero. For each of the following questions, answer yes or no, and explain very briefly (one short sentence can be enough).

   (a) Does this training set-up guarantee that the weight vector gets closer, on every step, to the weight vector that gives the smallest possible *loss* value (or to one such weight vector if there are multiple)?

   (b) Does this training set-up guarantee that the loss gets better (i.e. smaller) on every step?

   (c) Does this training set-up guarantee that eventually we'll reach a weight vector that gives the smallest possible *loss* value?

   (d) We use our network to classify images by simply seeing which of the 10 output units gets the largest probability when the network is presented with the image, and declaring the number of that output unit to be the network's guessed label. Does our training set-up guarantee that the number of mistakes that the network makes on training data (by following this classification strategy) never increases?

5. Here you draw the feasible region of a perceptron. Let's say we have a perceptron with two input units and no bias. We have a training set of two training cases:

$(1, 1) \to 1$	i.e. this is a "positive example"
$(0, 3) \to 0$	i.e. this is a "negative example"

   The perceptron has two learnable parameters: w_1 (the weight on the first input) and w_2 (the weight on the second input).

   (a) Draw the two-dimensional weight space, with w_1 on the horizontal axis and w_2 on the vertical axis. In that weight space, draw the line that separates weight vectors that correctly classify the first training case from weight vectors that incorrectly classify the first training case. Indicate which side is the "good" side (which has the correct weight vectors) and which is the "bad" side.

   (b) Now add the good-vs-bad dividing line of the second training case, and again indicate which side is good and which is bad. Indicate clearly which area of the diagram is the feasible region of this training set.