**CSC321 Winter 2015 — Intro to Neural Networks**
**Solutions for afternoon midterm**

Unless otherwise specified, half the marks for each question are for the answer, and half are for an explanation which demonstrates understanding of the relevant concepts.

1. *(2 marks) Briefly explain what is meant by overfitting. Is it true that if you choose the hyperparameters (e.g. number of hidden units) well, then there will be no overfitting? Why or why not? (Either YES or NO is acceptable, as long as you justify your answer.)*

   **Solution.** A learning algorithm overfits when it fits idiosyncrasies in the training set which aren't present in the test set, causing the training performance to be better than the test performance. Possible answers to the second part:

   (a) No. In trying to achieve the best predictive performance, there's a tradeoff between overfitting and underfitting (as well as other factors), and the optimal hyperparameter settings may involve some of each.

   (b) Yes. If your goal is to interpret the learned model, you want to choose a model simple enough that it has only minimal overfitting, so that you know it's only capturing real structure in the data.

   **Marking.** We gave one mark for a good explanation of overfitting, and one mark for an answer to the second part which shows understanding of the relevant concepts.

2. *(1 mark) Recall our study of the weight space geometry of linear regression. For this question, assume there is no bias parameter. We saw that the set of weight vectors* $\mathbf{w}$ *which predict a given target exactly,* i.e. $\mathbf{w}^T\mathbf{x}^{(i)} = t^{(i)}$*, is a hyperplane in weight space. If all the hyperplanes for a given training set intersect at a single point* $\mathbf{w}^\star$*, then must* $\mathbf{w}^\star$ *be an optimal solution to the linear regression problem? Why or why not?*

   **Solution.** Yes. If $\mathbf{w}^\star$ is contained in every hyperplane, then it fits every training case perfectly. It achieves a loss of 0, which is the smallest possible loss.

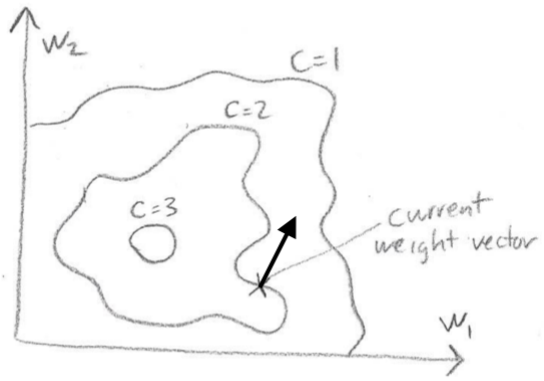
   **Marking.** By far the most common mistake was to get linear regression confused with classification. There is no "feasible region" for linear

regression, since that concept applies to constraint satisfaction (or constrained optimization) problems; regression is unconstrained.

Some people said it might not be an optimal solution since it wouldn't generalize. This isn't technically correct ("optimal solution" refers to a point which minimizes the cost), but we gave it full marks, as long as the answer showed understanding.

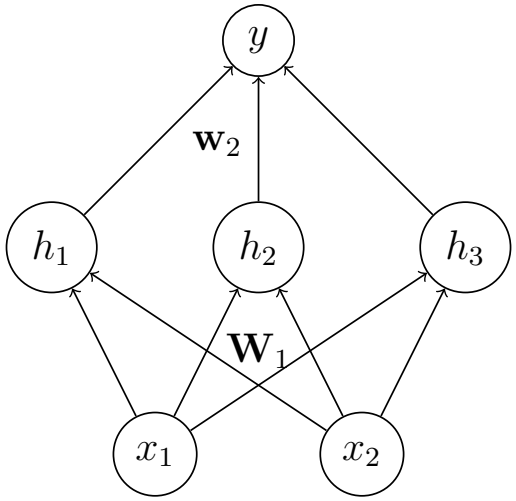
3. *(1 mark) The following diagram shows the level curves in weight space of a cost function $C$ which we are trying to* minimize. *The current weight vector is marked by an $\times$. Sketch the gradient descent update. (We haven't given you enough information to determine the magnitude, so we just want you to get the direction correct.)*

   **Solution:**




   **Marking.** Half a mark for the direction being orthogonal to the level set, and half a point for it being on the correct side of the level set.

4. *(2 marks) In the first week of class, we discussed how linear regression could be made more powerful using a basis function expansion, i.e. a function $\phi$ which maps each data point $\mathbf{x}$ to a feature vector $\phi(\mathbf{x})$. We later saw how this is analogous to fitting a feed-forward neural net with one hidden layer, where one set of weights is held fixed. (Such a network is shown in the following figure.) Which set of weights is held fixed? Briefly explain what the hidden activations and both sets of weights correspond to.*




**Solution.** The weights and hidden units correspond to:

(a) $\mathbf{W}_1$: parameters used in computing the feature vectors

(b) $\mathbf{h}$: the feature vector $\phi(\mathbf{x})$

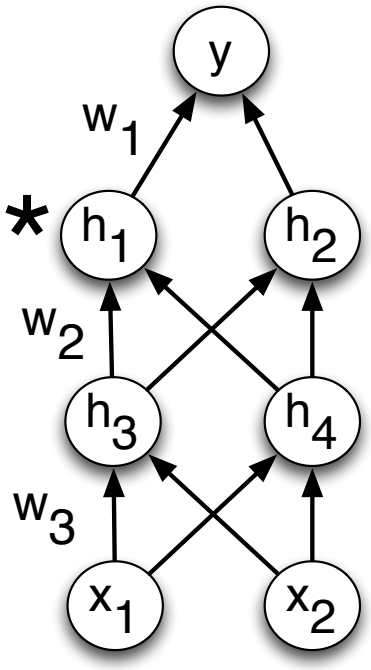
(c) $\mathbf{w}_2$: the regression weights

The weights $\mathbf{W}_1$ are held fixed, since they are used to compute the (fixed) basis function representation.

**Marking.** There were 4 individual questions, each worth half a mark.

5. *(3 marks) Consider the network shown in the figure. All of the hidden units use the linear rectification nonlinearity* $h_i = \max(z_i, 0)$. *We are trying to minimize a cost function* $C$ *which depends only on the activation of the output unit* $y$. *The unit* $h_1$ *(marked with a* $\star$*) receives an input of -1 on a particular training case, so its output is 0. Based only on this information, which of the following weight derivatives are* **guaranteed** *to be 0 for this training case? Write YES or NO for each. Justify your answers informally. Hint: don't work through the backprop computations. Instead think about what the partial derivatives really mean.*

$\partial C/\partial w_1$: YES $\partial C/\partial w_2$: YES $\partial C/\partial w_3$: NO




Note: Each of $w_1$, $w_2$, and $w_3$ refers to the weight on a *single* connection, not the whole layer.

**Solutions:**

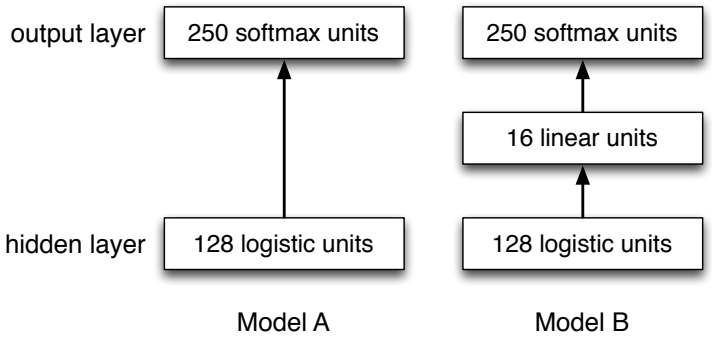
(a) $\partial C/\partial w_1$: YES, because $h_1$ is zero, and therefore changing $w_1$ doesn't affect the input to unit $y$. Therefore it doesn't affect the output of the network, or the cost.

(b) $\partial C/\partial w_2$: YES. Because the input $z_1$ is negative, $\partial h_1/\partial z_1 = 0$, so changing $w_2$ by a small amount doesn't change $h_1$. Therefore it has no effect on the output of the network.

(c) $\partial C/\partial w_3$: NO. Changing $w_3$ by a small amount can change $h_3$, which can change $h_2$, which can change $y$, which can change $C$.

**Marking.** We gave half a mark for each of the three answers and half a mark for having a reasonable justification for each.

6. *(2 marks) Let's compare the following two models. Model A is the neural probabilistic language model from Assignment 1. Model B is the same as Model A, with the following modification: in between the hidden layer (of size 128) and the output layer (of size 250), we insert a layer consisting of 16 linear units. The top layers of both models are shown in the figure. Describe one advantage of Model A and one advantage of Model B.*




**Solution.** In general, adding a linear hidden layer does not make a network any more powerful, but instead limits the functions that it can represent. The advantage of Model A is that it can learn any function that Model B can learn, plus some additional functions. Another solution we gave full credit for is that it has fewer layers, and therefore less of a problem with exploding/vanishing gradients. Possible advantages of Model B include:

(a) It has fewer parameters ($128 \times 16 + 16 \times 250$ as opposed to $128 \times 250$), so it is less likely to overfit.

(b) It is computationally cheaper, since a matrix-vector product of size $16 \times 128$, followed by one of size $250 \times 16$, requires fewer arithmetic operations than one of size $250 \times 128$.

(c) The added linear layer is essentially another word embedding layer. Therefore, after we train the model, we get an embedding of the target words which we can analyze and visualize.

**Marking.** One mark for each part. One common mistake was to say Model B was more complex/powerful than Model A. Some people said

an advantage of A is that it's simpler to implement (since you have fewer things to derive); we didn't give credit for this since the implementation difficulty isn't very different. We also didn't give credit for saying A has fewer units, unless there was an argument for why more units is worse. (In some models like conv nets, reducing the number of units helps since the activations take up a lot of memory; we would have given credit for this answer even though it's not really an issue for this model.)

7. *(1 mark) Briefly explain one method for dealing with the problem of exploding and/or vanishing gradients in recurrent nets. Why does this method help?*

   **Solution.** There are a lot of possible answers, including:

   (a) Clipping the gradients to be no larger than a certain norm prevents the optimization algorithm from taking extremely large steps.

   (b) In tasks that require memorization, reversing the input sequence makes it so some of the dependencies are short-term. The RNN can learn these first, before it learns to model the longer-distance ones.

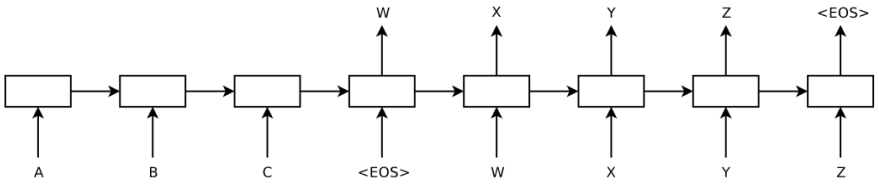
   (c) The Long-term Short Term Memory architecture makes it easy to maintain the state of the hidden units over a long time range. This drastically simplifies the form of the dependency between distant time steps, making the updates more stable.

   (d) Hessian-free optimization corrects for the curvature of the loss function in computing the updates. Since exploding gradients also tend to involve directions of high curvature, H-F will drastically reduce the size of the step in those directions.

   (e) You can carefully initialize the weights of the network so that the largest eigenvalue of the Jacobian at each time step is approximately 1. This prevents the gradients from exploding at the start of optimization, yet still preserves information over many time steps.

   **Marking.** Some people mentioned generic techniques for optimization which don't address RNNs specifically (e.g. momentum, using a smaller learning rate). We didn't give credit for these.

8. *(1 mark) We saw that we can apply a recurrent net to machine translation by feeding it an English sentence, and then having it generate the French sentence the same way an RNN language model generates text. This setup is shown in the figure. Would it work to use the neural probabilistic language model from Assignment 1 in the same way? Why or why not?*

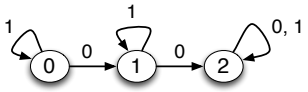



**Solution.** This would not work, since the neural language model is memoryless, which means it only uses the previous few inputs in making its predictions. Once it starts producing the French sentence, it will ignore all but the last few words of the English sentence.

**Marking.** The important idea here is memorylessness. We also gave half a mark for arguing that the language model only captures syntactic information, whereas you also need semantic information for translation. Other answers which would be just as much a problem for an RNN as for the Assignment 1 model received no credit. A common mistake was to argue that it wouldn't work because the sentences don't necessarily align one-to-one; people probably misread the problem and thought we were using the naive strategy of predicting the first French word from the first English word, etc.

9. *(1 mark) Design a finite state machine which determines if a given sequence of binary digits contains at least 2 zeros. Specify which state is the initial state and which state(s) correspond to an answer of YES. You do not need to justify your answer.*

**Solution.** The following FSM counts the number of zeros seen so far, up to a maximum of 2. The initial state is 0, and the state 2 corresponds to YES.

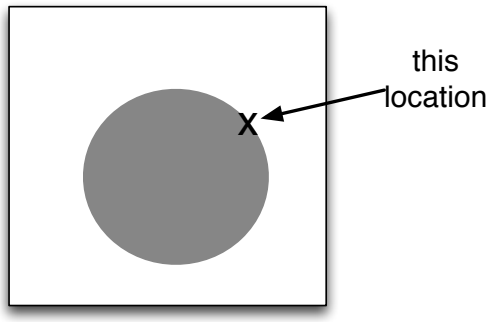


**Marking.** Most people got full credit on this one. A lot of the wrong answers may have resulted from misreading the question, e.g. looking for two *consecutive* zeros rather than two zeros total.

10. *(1 mark) Suppose we compute the convolution $I*w$, where $I$ is a grayscale image (white pixels correspond to values of 1 and black pixels correspond to 0) and*

$$w = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

*is a convolution kernel. For what parts of the image will the output be farthest from zero? In the image shown, will the output at the location marked by $\times$ take a positive or a negative value?*




**Solution, and marking.** Each part was worth half a point, and no justification was required. We compute the convolution using the flip-and-filter interpretation: we flip $w$ horizontally and vertically, and then compute the dot product for every window of the image.

This question was badly worded, and people interpreted the first part in a lot of different ways. Fortunately, all of the interpretations tested the relevant concepts equally well, so this wasn't a problem for marking.

(a) We intended to ask: *in general*, what parts of an image will have output values farthest from zero? Answer: horizontal edges.

(b) If you interpret it as applying to this particular image, the answer is the very top and bottom points on the circle. We also accepted things like "top and bottom parts."

(c) We were vague about whether the convolution operator used only windows within the input, or whether we padded. We gave full marks for saying it's the top and bottom edges of the image, assuming the image is padded with zeros.

At the location shown, it will take a positive value, since the negative part of the filter overlaps with the dark region, and the positive part overlaps with the light region. (This is true because $w$ is flipped as part of computing the convolution.)