Instructor's Solution Manual for "Neural Networks and Deep Learning"

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY

May 17, 2018

Contents

Chapter 1

An Introduction to Neural Networks

1. *Consider the case of the XOR function in which the two points* $\{(0,0),(1,1)\}$ *belong to one class, and the other two points* $\{(1,0),(0,1)\}$ *belong to the other class. Show how you can use the ReLU activation function to separate the two classes in a manner similar to the example in the chapter.*

   We assume that the hidden layer contains two units. The first layer should implement the transformations $x_1 - x_2$ and $x_2 - x_1$ to create the pre-activation values. On applying the ReLU activation to the two pre-activated values, one obtains the representation $\{(0,0),(0,0)\}$ for the first pair of data points, and the representation $\{(1,0),(0,1)\}$ for the second pair of the data points. Clearly, the two classes become separable.

2. *Show the following properties of the sigmoid and tanh activation functions (denoted by* $\Phi(\cdot)$ *in each case):*

   (a) Sigmoid activation: $\Phi(-v) = 1 - \Phi(v)$

   (b) Tanh activation: $\Phi(-v) = -\Phi(v)$

   (c) Hard tanh activation: $\Phi(-v) = -\Phi(v)$

   For sigmoid activation $\Phi(-v) = 1/(1+\exp(v)) = \exp(-v)/(1+\exp(-v))$. The last value can easily be shown to be $1 - 1/(1+\exp(-v)) = 1 - \Phi(v)$.

   For tanh activation, the proof is similar; the numerator and denominator should be multiplied with $\exp(2v)$ to obtain the result.

   The proof for hard tanh is even simpler because it is a thresholded linear function.

3. *Show that the tanh function is a re-scaled sigmoid function with both horizontal and vertical stretching, as well as vertical translation:*

$$tanh(v) = 2sigmoid(2v) - 1$$

   This identity is easy to show by plugging in the values on both sides.

4. *Consider a data set in which the two points $\{(-1,-1),(1,1)\}$ belong to one class, and the other two points $\{(1,-1),(-1,1)\}$ belong to the other class. Start with perceptron parameter values at $(0,0)$, and work out a few stochastic gradient descent updates with $\alpha = 1$. While performing the stochastic gradient descent updates, cycle through the training points in any order.*

(a) Does the algorithm converge in the sense that the change in objective function becomes extremely small over time?

(b) Explain why the situation in (a) occurs.

The algorithm will not converge because the two classes are not linearly separable.

5. *For the data set in Exercise 4, where the two features are denoted by (x_1, x_2), define a new 1-dimensional representation z denoted by the following:*

$$z = x_1 \cdot x_2$$

Is the data set linearly separable in terms of the 1-dimensional representation corresponding to z? Explain the importance of nonlinear transformations in classification problems.

The points in one class map to 1, whereas the points in the other class map to -1. Therefore, the transformation makes the points linearly separable. Note that this is the XOR function, and therefore nonlinear transformations are required to map inseparable points to separable values.

6. *Implement the perceptron in a programming language of your choice.*

This is an implementation exercise.

7. *Show that the derivative of the sigmoid activation function is at most* 0.25*, irrespective of the value of its argument. At what value of its argument does the sigmoid activation function take on its maximum value?*

The derivative is of the form $o(1-o)$, where $o \in (0,1)$. By differentiating, it is easy to show that this function takes on its maximum value at $o = 0.5$ (i.e, argument value of 0), and the maximum value of 1. Furthermore $o(1-o)$ always lies in $(0,1)$ because each of the terms in the product is less than 1.

8. *Show that the derivative of the tanh activation function is at most 1, irrespective of the value of its argument. At what value of its argument does the tanh activation take on its maximum value?*

The output is of the form $1 - o^2$, which is always at most 1 at $o = 0$. The maximum gradient is 1. Note that the tanh activation function tends to have less problem as compared to the sigmoid with respect to the vanishing gradient problem.

9. *Consider a network with two inputs x_1 and x_2. It has two hidden layers, each of which contain two units. Assume that the weights in each layer are set so that top unit in each layer applies sigmoid activation to the sum of its inputs and the bottom unit in each layer applies tanh activation to the sum of its inputs. Finally, the single output node applies ReLU activation to the sum of its two inputs. Write the output of this neural network* in closed form *as a function of x_1 and x_2. This exercise should give you an idea of the complexity of functions computed by neural networks.*

The function that is computed by this neural network is the following:

$$\max\{0, \text{sigmoid}(\text{sigmoid}(x_1+x_2)+\tanh(x_1+x_2))+\tanh(\text{sigmoid}(x_1+x_2)+\tanh(x_1+x_2))\}$$

11. *Consider a 2-dimensional data set in which all points with $x_1 > x_2$ belong to the positive class, and all points with $x_1 \leq x_2$ belong to the negative class. Therefore, the true separator of the two classes is linear hyperplane (line) defined by $x_1 - x_2 = 0$. Now create a training data set with 20 points randomly generated inside the unit square in the positive quadrant. Label each point depending on whether or not the first coordinate x_1 is greater than its second coordinate x_2.*

- *(a) Implement the perceptron algorithm without regularization, train it on the 20 points above, and test its accuracy on 1000 randomly generated points inside the unit square. Generate the test points using the same procedure as the training points.*
- *(b) Change the perceptron criterion to hinge-loss in your implementation for training, and repeat the accuracy computation on the same test points above. Regularization is not used.*
- *(c) In which case do you obtain better accuracy and why?*
- *(d) In which case do you think that the classification of the same 1000 test instances will not change significantly by using a different set of 20 training points?*

(a) and (b) and implementation exercises. In general, the margin-based objective function of hinge less tries to find separators in which marginal classification is discouraged. Therefore, it generalizes better to unseen test instances and is also more stable with choice of training data. Therefore, the hinge-loss will usually be more accurate for (c) and also more stable for (d).

Chapter 2

Machine Learning with Shallow Neural Networks

1. *Consider the following loss function for training pair $(\overline{X}, y)$:*

$$L = max\{0, a - y(\overline{W} \cdot \overline{X})\}$$

Assume that test instances are predicted using $y = sign\{\overline{W} \cdot \overline{X}\}$. A value of $a = 0$ corresponds to the perceptron criterion and a value of $a = 1$ corresponds to the SVM. Show that any value of $a > 0$ leads to the SVM with an unchanged optimal solution when no regularization is used. What happens when regularization is used?

Any solution $\overline{W}$ to the SVM has a corresponding solution $a\overline{W}$ with an objective function value that is scaled up by a. In other words, the optimal value of $\overline{W}$ does not change. When regularization is used, the regularization parameter needs to be divided by a to ensure the same optimal solution. In other words, when regularization is used, the problem is still an SVM, but its regularization parameters need to be changed.

2. *Based on your answer to Exercise 1, can you formulate a generalized objective function for the Weston-Watkins SVM?*

The generalized objective function for the ith training instance is defined as follows:

$$J_i = \sum_{r:r \neq c(i)} \max(\overline{W_r} \cdot \overline{X_i} - \overline{W}_{c(i)} \cdot \overline{X_i} + a, 0)$$

Here, $a > 0$ is a free hyper-parameter.

3. *Consider the unregularized perceprton update for binary classes with learning rate α. Show that using any value of α is inconsequential in the sense that it only scales up the weight vector by a factor of α. Show that these results also hold true for the multiclass case. Do the results hold true when regularization is used?*

Any value of α only scales up the weights of all the separators by α and leaves their relative proportions unchanged. This is only true for the perceptron and is not true for the other learning methods.

4. *Show that if the Weston-Watkins SVM is applied to a data set with $k = 2$ classes, the resulting updates are equivalent to the binary SVM updates discussed in this chapter.*

If we start with weight vector values of 0, the updates will be the negative of each other and will be the same as that in te binary SVM.

5. *Show that if multinomial logistic regression is applied to a data set with $k = 2$ classes, the resulting updates are equivalent to the logistic regression updates discussed in this chapter.*

Same as above.

6. *Implement the softmax classifier using a deep learning library of your choice.*

This is an implementation exercise.

7. *In linear-regression-based neighborhood models the rating of an item is predicted as a weighted combination of the ratings of other items of the same user, where the item-specific weights are learned with linear regression. Show how you can construct an autoencoder architecture to create this type of model. Discuss the relationship of this architecture with the matrix factorization architecture.*

The inputs correspond to the ratings of the other items, and missing inputs are implicitly assumed to have rating values of 0. Note that this approach is likely to have bias because of setting missing values to 0. On the other hand, the matrix factorization architecture discussed in the book will not have bias. In fact, a recent recommender architecture, referred to as *AutoRec*, is based on this principle.

8. *Consider an autoencoder which has an input layer, a single hidden layer containing the reduced representation, and an output layer with sigmoid units. The hidden layer has linear activation:*

(a) Set up a negative log-likelihood loss function for the case when the input data matrix is known to contain binary values from $\{0, 1\}$.

(b) Set up a negative log-likelihood loss function for the case when the input data matrix contains real values from $[0, 1]$.

For the case of the binary matrix the loss for each entry with value of 1 is $-\log(\sigma((UV^T)_{ij}))$ and the loss for each entry with a value of 0 is $-\log(1-\sigma((UV^T)_{ij}))$. Here, $\sigma(\cdot)$ is the sigmoid function. For fractional entries at a fraction of f, we combine the loss for 0 and 1 as follows:

$$J_i = -f \cdot \log(\sigma((UV^T)_{ij})) - (1 - f) \cdot \log(1 - \sigma((UV^T)_{ij})) \tag{2.1}$$

In other words, we use a convex combination of the losses at 0 and 1.

9. **Non-negative matrix factorization with autoencoders:** *Let D be an $n \times d$ data matrix with non-negative entries. Show how you can approximately factorize $D \approx UV^T$ into two non-negative matrices U and V, respectively, by using an autoencoder architecture with d inputs and outputs. [Hint: Choose an appropriate activation function in the hidden layer, and modify the gradient-descent updates.]*

The ReLU activation will ensure that the output of the hidden layer is non-negative. In addition, during gradient-descent any negative value of the parameter from the hidden-to-output layer is set to 0 after each gradient-descent step. The outputs of the hidden layer can be stacked to create the matrix U, and the weights from hidden to output layer provide the matrix V^T. The data is reconstructed as UV^T. Note that the weights from input-to-hidden layer are not tied to the hidden-to-output layer. These weights are allowed to be negative, as the pseudo-inverse of V might have negative entries.

10. *Propose a modification of the approach in Exercise 9 for probabilistic latent semantic analysis.*

The only difference is that the I-divergence objective needs to be used instead of the squared error. The gradient-descent steps are appropriately modified.

11. **Simulating a model combination ensemble:** *In machine learning, a model combination ensemble averages the scores of multiple models in order to create a more robust classification score. Discuss how you can approximate the averaging of an Adaline and a logistic regression classifier with a two-layer neural network. Discuss the similarities and differences of this architecture with an actual model combination ensemble when backpropagation is used to train it. Show how you can modify the training process of the neural network so that the final result is a fine-tuning of the model combination ensemble.*

The first layer contains a linear activation and a sigmoid activation, and the second layer contains fixed weights that are set to 1. However, if backpropagation is used to train it, the loss functions used in the first layer are not the same, which makes the model somewhat different. One can more closely simulate the ensemble by first pre-training the first layer with their relevant loss functions, and then attaching the second layer. Backpropagation is used to fine-tune the weights with any reasonable loss function and activation associated with the final layer for classification. The result is still not an exact simulation of the model combination ensemble, but is actually a fine-tuned version of the pre-trained weights, which create a model combination ensemble. The pre-training actually helps in regularization, if data is limited.

12. **Simulating a stacking ensemble:** *In machine learning, a stacking ensemble creates a higher level classification model on top of features learned from first level classifiers. Discuss how you can modify the architecture of Exercise 11, so that the first level of classifiers correspond to an Adaline and a logistic regression classifier and the higher-level classifier corresponds to a support vector machine. Discuss the similarities and differences of this architecture with an actual stacking ensemble when backpropagation is used to train it. Show how you can modify the training process of the neural network so that the final result is a fine-tuning of the stacking ensemble.*

The answer to this problem is the same as the previous exercise, except that the final layer also contains weights, and loss function of the output node is set to that of the SVM. In this case, the first layer is pre-trained (with loss functions of Adaline and logistic regression) and then the second layer is pre-trained with the outputs of the first layer and the SVM loss. Back-propagation is used for fine-tuning with only the loss of the output layer.

13. *Show that the stochastic gradient-descent updates of the perceptron, Widrow-Hoff learning, SVM, and logistic regression are all of the form $\overline{W} \Leftarrow \overline{W}(1-\alpha\lambda) + \alpha y[\delta(\overline{X}, y)]\overline{X}$. Here, the mistake function $\delta(\overline{X}, y)$ is $1 - y(\overline{W} \cdot \overline{X})$ for least-squares classification, an indicator variable for perceptron/SVMs, and a probability value for logistic regression. Assume that α is the learning rate, and $y \in \{-1, +1\}$. Write the specific forms of $\delta(\overline{X}, y)$ in each case.*

This result can be easily shown by going back to the text and substituting the appropriate term in the update with $\delta(\overline{X}, y)$.

14. *The linear autoencoder discussed in the chapter is applied to the rows of the $n \times d$ data set D to create a k-dimensional representation of each row. The encoder weights contain the $k \times d$ weight matrix W and the decoder weights contain the $d \times k$ weight matrix V. Therefore, the reconstructed representation is $DW^T V^T$, and we are trying to minimize the aggregate loss value $||DW^T V^T - D||^2$ over the entire training data set.*

(a) For a fixed value of V, show that the optimal matrix W must satisfy $D^T D(W^T V^T V - V) = 0$.

(b) Use (a) to show that if the matrix D has full rank d, then we must have $W^T V^T V = V$.

(c) Use (b) to show that $W = (V^T V)^{-1} V^T$. Assume that $V^T V$ is invertible.

(d) Work out the corresponding conditions in (a), (b), and (c), when the encoder-decoder weights are tied as $W = V^T$. Show that the columns of V must be orthonormal.

(a) Setting the gradient to zero results in the condition of (a). It is relatively easy to show the result using matrix calculus.

(b) If D has rank d, then $D^T D$ must also have rank d. The only way in which condition (a) can be satisfied is if $W^T V^T V = V$.

(c) By premultiplying the condition in (b) on both sides by $(V^T V)^{-1}$, we get the desired result.

(d) If the encoder and decoder weights are tied, then we have $VV^T V = V$. Premultiplying both sides with V^T, we obtain $V^T V(V^T V - I) = 0$. Since $V^T V$ is invertible, we can multiply both sides with $(V^T V)^{-1}$ in order to obtain the desired result.

Chapter 3

Training Deep Neural Networks

1. *Consider the following recurrence:*

$$(x_{t+1}, y_{t+1}) = (f(x_t, y_t), g(x_t, y_t)) \tag{3.1}$$

Here, $f()$ and $g()$ are multivariate functions.

(a) Derive an expression for $\frac{\partial x_{t+2}}{\partial x_t}$ in terms of only x_t and y_t.

(b) Can you draw an architecture of a neural network corresponding to the above recursion for t varying from 1 to 5? Assume that the neurons can compute any function you want.

(a) One can first use the chain rule by summing over the different paths from x_t to x_{t+2}.

$$\frac{\partial x_{t+2}}{\partial x_t} = \frac{\partial x_{t+2}}{\partial x_{t+1}}\frac{\partial x_{t+1}}{\partial x_t} + \frac{\partial x_{t+2}}{\partial y_{t+1}}\frac{\partial y_{t+1}}{\partial x_t} \tag{3.2}$$

Now, one can recursively substitute the functional forms to obtain the appropriate expression.

(b) The resulting architecture will have five layers in which there are two nodes in each layer. The functions f and g are computed in each layer.

2. *Consider a two-input neuron that multiplies its two inputs x_1 and x_2 to obtain the output o. Let L be the loss function that is computed at o. Suppose that you know that $\frac{\partial L}{\partial o} = 5$, $x_1 = 2$, and $x_2 = 3$. Compute the values of $\frac{\partial L}{\partial x_1}$ and $\frac{\partial L}{\partial x_2}$.*

One can use the chain rule to obtain the fact that $\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial x_1} = \frac{\partial L}{\partial o} x_2$. Therefore, the value is $5 \times 3 = 15$. Similarly, the value of $\frac{\partial L}{\partial o}\frac{\partial L}{\partial x_2} = 5 \times 2 = 10$.

3. *Consider a neural network with three layers including an input layer. The first (input) layer has four inputs x_1, x_2, x_3, and x_4. The second layer has six hidden units corresponding to all pairwise multiplications. The output node o simply adds the values in the six hidden units. Let L be the loss at the output node. Suppose that you know that $\frac{\partial L}{\partial o} = 2$, and $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, and $x_4 = 4$. Compute $\frac{\partial L}{\partial x_i}$ for each i.*

One can use a similar approach as exercise 2 with the use of the chain rule. In this case, one can show that the partial derivative $\frac{\partial L}{\partial x_1}$ is simply the product of $(x_2 + x_3 + x_4$ with $\frac{\partial L}{\partial o}$. This value is $9 \times 2 = 18$. A similar approach can be used for the other variables.

4. *How does your answer to the previous question change when the output o is computed as a maximum of its six inputs rather than its sum.*

In such a case, one just backpropagates along the maximum of the nodes in the last hidden layer. Here, the maximum product is $x_3 \cdot x_4 = 12$. Therefore, the partial derivatives of the loss with respect to x_1 and x_2 are 0. The partial derivative with respect to x_3 is $2 \times x_4 = 8$, and the partial derivative with respect to x_4 is $2 \times x_3 = 6$.

5. *The chapter discusses how one can perform a backpropagation of an arbitrary function by using the multiplication with the Jacobian matrix. At first sight, this seems like a very useful approach. However, computing the Jacobian is rarely done in practice. Discuss why this is the case. [Hint: Compute the Jacobian with respect to sigmoid function]*

This is because the Jacobian matrix is very sparse, and it would be inefficient to compute the entire matrix.

6. *Consider the loss function $L = x^2+y^{10}$. Implement a simple steepest descent algorithm to plot the coordinates as they vary from the initialization point to the optimal value of 0. Consider two different initialization points of $(0.5, 0.5)$ and $(2, 2)$ and plot the trajectories in the two cases at a constant learning rate. What do you observe about the behavior of the algorithm in the two cases?*

In one case, the trajectory is more sensitive to x, and the other case it is more sensitive to y. Little progress is often made with respect to one of the variables.

7. *Show that if the Hessian is positive definite, then all conjugate directions are linearly independent.*

Suppose that the conjugate directions are not linearly independent. Without loss of generality, assume that $\overline{q}_d$ can be expressed in terms of $\overline{q}_1 \dots \overline{q}_{d-1}$.

$$\overline{q}_d = \sum_{i=1}^{d-1} \lambda_i \overline{q}_i \tag{3.3}$$

Then, we have $\overline{q}_d^T H \overline{q}_d = 0$ by expanding one of the two $\overline{q}_d$ in the above expression in terms of its linear dependents. However, this is in contradiction to the fact that H is positive definite.

8. *Show that if the dot product of a d-dimensional vector $\overline{v}$ with d linearly independent vectors is 0, then $\overline{v}$ must be the zero vector.*

Let $\overline{x}_1 \dots \overline{x}_d$ be the d linearly independent vectors. These vectors form a basis for d-dimensional space, and therefore the vector $\overline{v}$ must be expressed as a linear combination of $\overline{x}_1 \dots \overline{x}_d$. Therefore, we have $\overline{v} = \sum_{i=1}^{d} \alpha_i \overline{x}_i$. By taking the dot product of both sides with $\overline{v}$ we get the following:

$$||\overline{v}||^2 = \sum_{i=1}^{d} \alpha_i (\overline{v} \cdot \overline{x}_i) = \sum_{i=1}^{d} \alpha_i (0) = 0 \tag{3.4}$$

The modulus of a vector is zero, when the vector is itself zero.

9. *This chapter discusses two variants of backpropagation, which use the pre-activation and the postactivation variables, respectively, for the dynamic programming recursion. Show that these two variants of backpropagation are mathematically equivalent.*

This result can be shown by showing that both the pre-activation as well as pre-activation updates are equivalent to the decoupled updates in which both pre-activation and post-activation variables are used.

10. *Consider the softmax activation function in the output layer, in which real-valued outputs $v_1 \ldots v_k$ are converted into probabilities as follows:*

$$o_i = \frac{exp(v_i)}{\sum_{j=1}^{k} exp(v_j)} \quad \forall i \in \{1, \ldots, k\}$$

(a) Show that the value of $\frac{\partial o_i}{\partial v_j}$ is $o_i(1 - o_i)$ when $i = j$. In the case that $i \neq j$, show that this value is $-o_i o_j$.

(b) Use the above result to show the correctness of the loss derivative:

$$\frac{\partial L}{\partial v_i} = o_i - y_i$$

Assume that we are using the cross-entropy loss $L = -\sum_{i=1}^{k} y_i log(o_i)$, where $y_i \in \{0, 1\}$ is the one-hot encoded class label over different values of $i \in \{1 \ldots k\}$.

(a) Both results are straightforward differentiation. The first one is similar to differentiation of sigmoid.

(b) In this case, we can use the following:

$$\frac{\partial L}{\partial v_i} = \sum_j \frac{\partial L}{\partial o_j}\frac{\partial o_j}{\partial v_i} = \sum_{j \neq i} \frac{y_j}{o_j} o_j o_i - y_i(1 - o_i) = o_i(\sum_j y_j) - y_i \tag{3.5}$$

Since we have $\sum_j y_j = 1$, the result follows.

11. *The chapter uses steepest descent directions to iteratively generate conjugate directions. Suppose we pick d* arbitrary *directions $\overline{v}_0 \ldots \overline{v}_{d-1}$ that are linearly independent. Show that (with appropriate choice of β_{ti}) we can start with $\overline{q}_0 = \overline{v}_0$ and generate successive conjugate directions in the following form:*

$$\overline{q}_{t+1} = \overline{v}_{t+1} + \sum_{i=0}^{t} \beta_{ti}\overline{q}_i \tag{3.6}$$

Discuss why this approach is more expensive than the one discussed in the chapter.

Pre-multiply both sides of Equation 3.6 with the row vector $\overline{q}_i^T H$ and use the conjugacy condition to set the LHS to 0. This results in the following value of β_{ti}:

$$\beta_{ti} = -\frac{\overline{q}_i^T H \overline{v}_{t+1}}{\overline{q}_i^T H \overline{q}_i} \tag{3.7}$$

This approach requires us to maintain *all* previous directions, and also requires us to compute $O(d)$ values of β_{ti} for varying i. Therefore, the approach is not as time- and space-efficient.

12. *The definition of β_t ensures that $\overline{q}_t$ is conjugate to $\overline{q}_{t+1}$. This exercise systematically shows that* **any** *direction $\overline{q}_i$ for $i \leq t$ satisfies $\overline{q}_i^T H \overline{q}_{t+1} = 0$:*

(a) Recall that $H\overline{q}_i = [\nabla L(\overline{W}_{i+1}) - \nabla L(\overline{W}_i)]/\delta_i$ for quadratic loss functions, where δ_i depends on step-size. Show the following for all $i \leq t$:

$$\delta_i[\overline{q}_i^T H \overline{q}_{t+1}] = -[\nabla L(\overline{W}_{i+1}) - \nabla L(\overline{W}_i)]^T [\nabla L(\overline{W}_{t+1})] + \delta_i \beta_t (\overline{q}_i^T H \overline{q}_t)$$

Also show that $[\nabla L(\overline{W}_{t+1}) - \nabla L(\overline{W}_t)] \cdot \overline{q}_i = \delta_i \overline{q}_i^T H \overline{q}_t$.

(b) Show that $\nabla L(\overline{W}_{t+1})$ is orthogonal to each $\overline{q}_i$ for $i < t$. [The case for $i = t$ is trivial because of line-search.]

(c) Show that the loss gradients at $\overline{W}_0 \ldots \overline{W}_{t+1}$ are mutually orthogonal.

(d) Show that $\overline{q}_i^T H \overline{q}_{t+1} = 0$ for $i \leq t$.

Since $H\overline{q}_i = [\nabla L(\overline{W}_{i+1}) - \nabla L(\overline{W}_i)]/\delta_i$, we can use the transpose of the vector condition while keeping in mind that H is symmetric:

$$\overline{q}_i^T H = [\nabla L(\overline{W}_{i+1}) - \nabla L(\overline{W}_i)]^T/\delta_i \tag{3.8}$$

Now note that successive conjugate directions are generated as $\overline{q}_{t+1} = -\nabla L(\overline{W}_{t+1}) + \beta_t q_t$. Pre-multiplying both sides with $\overline{q}_i^T H$ for $i \leq t$, we get the following:

$$\overline{q}_i^T H \overline{q}_{t+1} = -\overline{q}_i^T H[\nabla L(\overline{W}_{t+1})] + \beta_t \overline{q}_i^T H \overline{q}_t \tag{3.9}$$

Now substituting for only the first $\overline{q}_i^T H$ in the right-hand side of Equation 3.9 using Equation 3.8, we get the desired result.

Also since we have $H\overline{q}_t = [\nabla L(\overline{W}_{t+1}) - \nabla L(\overline{W}_t)]/\delta_t$, we can pre-multiply both sides with the row vector $\overline{q}_i^T$ to get the second result of (a). Note that the right-hand side can also be written as a dot product, since it is the product of a row vector and a column vector.

Joint proof of (b), (c), (d): We will make the inductive assumption that all the statements of (b), (c), and (d) are true for values of t that are less than or equal to $k-1$, and we will show the results at $t = k$.

First, from (a), we have already proved that $\nabla L(\overline{W}_{k+1}) \cdot \overline{q}_i = \nabla L(\overline{W}_k) \cdot \overline{q}_i + \delta_k \overline{q}_i^T H \overline{q}_k$ for $i < k$. Both the terms on the right-hand side are zero because of the inductive assumption in the case where $i < k-1$. For the case when $i = k-1$ both terms are also zero because of the line-search condition $\nabla L(\overline{W}_k) \cdot \overline{q}_{k-1} = 0$ and also because of the fact that $\overline{q}_{k-1}^T H \overline{q}_k = 0$ by definition. Therefore, we have shown the induction for (b).

Next, we will examine the orthogonality of gradient $\nabla L(\overline{W}_{k+1})$ with gradient of $\nabla L(\overline{W}_i)$. In the case where $i = k$, we know that $\nabla L(\overline{W}_{k+1}) \cdot \overline{q}_k = 0$ because of the line-search condition. Now expanding $\overline{q}_k = \beta_{k-1}\overline{q}_{k-1} - \nabla L(\overline{W}_k)$ and taking the dot product of both sides with $\nabla L(\overline{W}_{k+1})$, we get $0 = \nabla L(\overline{W}_{k+1}) \cdot [\beta_{k-1}\overline{q}_{k-1} - \nabla L(\overline{W}_k)]$. Note that we have used the line-search condition to set the LHS to 0. Now note that the first of the two terms on the RHS sets to 0 because of the inductive assumption, which leaves only the term involving successive gradients. Therefore, we get the result that *immediately successive gradients* $\nabla L(\overline{W}_k)$ and $\nabla L(\overline{W}_{k+1})$ are orthogonal.

Next, we will show the result that $\nabla L(\overline{W}_i)$ is orthogonal to $\nabla L(\overline{W}_{k+1})$ for $i < k$. By rearranging the recursive definition of $\overline{q}_i$ in terms of gradient at $\overline{W}_i$ and $\overline{q}_{i-1}$, the gradient $\nabla L(\overline{W}_i)$ can be expressed as a linear combination of $\overline{q}_i$ and $\overline{q}_{i-1}$. Both $\overline{q}_i$ and $\overline{q}_{i-1}$ are orthogonal to $\nabla L(\overline{W}_{k+1})$ based on the inductive assumption, and therefore any linear combination of them must also be orthogonal to $\nabla L(\overline{W}_{k+1})$. In other words, $\nabla L(\overline{W}_i)$ is orthogonal to $\nabla L(\overline{W}_{k+1})$ for $i < k$. Therefore, we have shown the induction for (c).

Next, we will show the induction for (d) only for the case when $i < k$. This is because conjugacy between immediately successive directions follows from the way in which $\overline{q}_{k+1}$ is defined as a function of $\overline{q}_k$ and how β_k is chosen. We restate the result we showed in (a) using the inductive index k.

$$\delta_i[\overline{q}_i^T H \overline{q}_{k+1}] = -[\nabla L(\overline{W}_{i+1}) - \nabla L(\overline{W}_i)]^T [\nabla L(\overline{W}_{k+1})] + \delta_i \beta_k (\overline{q}_i^T H \overline{q}_k)$$

Now observe that the first term on the RHS is 0 because of what we proved in (c). The second term on the RHS is 0 because of the inductive assumption. Therefore, we have shown that the LHS is 0 as well. This completes the induction for (d), which is the conjugacy condition.

The initialization conditions of the induction can be shown in a relatively simple way by using the fact that the gradient at the optimal point found by line search is always orthogonal to the original direction, and also the fact that immediately successive directions are always conjugate (by definition).

Chapter 4

Teaching Deep Learners to Generalize

1. *Consider two neural networks used for regression modeling with identical structure of 10 hidden layers containing 100 units each. In both cases, the output node is single unit with linear activation. The only difference is that one of them uses linear activations in the hidden layers and the other uses sigmoid activations. Which model will have higher variance in prediction?*

   The neural network with nonlinear activations is inherently more powerful, and it will also have higher variance. As discussed in Chapter 1, using only linear activations will result in a linear model.

2. *Consider a situation in which you have four attributes $x_1 \dots x_4$, and the dependent variable y is such that $y = 2x_1$. Create a tiny training data set of 5 distinct examples in which a linear regression model without regularization will have infinite number of coefficient solutions with $w_1 = 0$. Discuss the performance of such a model on out-of-sample data. Why will regularization help?*

   The training pairs are as follows:

   $[1, 1, 0, 0]2$
   $[2, 0, 1, 0]4$
   $[3, 1, 1, 0]6$
   $[5, 1, 2, 0]10$
   $[4, 2, 1, 0]8$

   In this case, it can be shown that setting $w_1 = 0, w_2 = 2, w_3 = 4$ and w_4 to anything will result in zero error on the training data. The performance on out-of-sample data will be poor. However, adding a penalty on the coefficients will choose the most compact solution, which is $w_1 = 2$.

3. *Implement a perceptron with and without regularization. Test the accuracy of both variations of the perceptron on both the training data and the out-of-sample data on the* Ionosphere *data set of the* UCI Machine Learning Repository*. What do you observe about the effect of regularization in the two cases. Repeat the experiment with smaller samples of the* Ionosphere *training data, and report your observations.*

The performance without regularization will be better on the training data, and the performance with regularization will be better on out-of-sample test data. This effect will be amplified for smaller training data sets.

4. *Implement an autoencoder with a single hidden layer. Reconstructs inputs for the* Ionosphere *data set of the previous exercise with (a) no added noise and weight regularization, (b) added Gaussian noise and no weight regularization.*

This is an implementation exercise.

5. *The discussion in the chapter uses an example of sigmoid activation for the contractive autoencoder. Consider a contractive autoencoder with a single hidden layer and ReLU activation. Discuss how the updates change when ReLU activation is used. Discuss the relationship of this setting with straightforward weight regularization.*

The updates are almost like weight regularization except that the multiplicative weight decay is applied only when the ReLU unit that it is incident to is activated.

6. *Suppose that you have a model that provides around* 80% *accuracy on the training as well as on the out-of-sample test data. Would you recommend increasing the amount of data or adjusting the model to improve accuracy?*

This is clearly a case of under-fitting because the training and test accuracy are very close. Furthermore, training accuracy is usually close to 100% in over-fitting models with large gaps in accuracy. Therefore, I would recommend increasing the capacity of the model by adding more units, which preferably have nonlinearity.

7. *In the chapter, we showed that adding Gaussian noise to the input features in linear regression is equivalent to L_2-regularization of linear regression. Discuss why adding of Gaussian noise to the input data in a de-noising single-hidden layer autoencoder with linear units is roughly equivalent to L_2-regularized singular value decomposition.*

Consider the case in which noise with variance λ is added to each attribute. Note that the prediction of each of the d attributes in an autoencoder can be viewed as an independent linear regression problem, except that it goes through a linear constriction layer. However, since all units a linear, the overall model is still linear. The expected covariance matrix after added noise only changes in terms of the diagonal matrix, where variance of λ is added to each diagonal element. Therefore, the eigenvectors of $D^T D + \lambda I$ are returned. Note that conditioning the diagonal elements of the covariance matrix in this way is equivalent to L_2-regularization of SVD.

8. *Consider a network with a single input layer, two hidden layers, and a single output predicting a binary label. All hidden layers use the sigmoid activation function and no regularization is used. The input layer contains d units, and each hidden layer contains p units. Suppose that you add an additional hidden layer between the two current hidden layers, and this additional hidden layer contains q linear units.*

(a) Even though the number of parameters have increased by adding the hidden layer, discuss why the capacity of this model will decrease when $q < p$.

(b) Does the capacity of the model increase when $q > p$?

(a) Linear layers do not add to the representation power of the network, as discussed in Chapter 1. However, by forcing the activations to pass through a constricted layer, we are effectively performing a low-rank factorization of the weight matrix between the hidden layers in the original model. This is equivalent to regularization that reduces the capacity of the model.

(b) When $q > p$, the capacity of the resulting model will be the same. This is because we can collapse the two weight matrices in the changed model into a single linear weight matrix without changing the predictions in any way.

9. *Bob divided the labeled classification data into a portion used for model construction and another portion for validation. Bob then tested 1000 neural architectures by learning parameters (backpropagating) on the model-construction portion and testing its accuracy on the validation portion. Discuss why the resulting model is likely to yield poorer accuracy on the out-of-sample test data as compared to the validation data, even though the validation data was not used for learning parameters. Do you have any recommendations for Bob on using the results of his 1000 validations?*

This approach can lead to overfitting because the best of the 1000 models in in-sample data might not be the best of the 1000 models in out-of-sample data. It might be advisable to select the top-k best performing models (say, $k = 10$) and then average the results.

10. *Does the classification accuracy on the training data generally improve with increasing training data size? How about the point-wise average of the loss on training instances? Explain your answer. At what point will training and testing accuracy become similar?*

The classification accuracy on the training data generally worsens with increasing training data size, because the model can no longer memorize the training data with limited capacity. Similarly, the average loss per training instance, generally increases with training data size, as the model becomes less specific to particular training instances. This result is in contrast to the fact that *test* accuracy generally improves with increasing training data size. When the amount of training data is very large, the training and testing accuracy will tend to become similar.

11. *What is the effect of increasing the regularization parameter on the training and testing accuracy? At what point do training and testing accuracy become similar?*

Increasing the regularization parameter almost always reduces training accuracy. However, it first increases testing accuracy by reducing overfitting. At some point, increasing the regularization parameter further will cause underfitting. At this point, the testing accuracy will reduce. When the regularization parameters are very large, all parameters will be set to values close to 0, and training/testing accuracy will tend to become similar.

Chapter 5

Radial Basis Function Networks

1. *Consider the following variant of radial basis function networks in which the hidden units take on either 0 or 1 values. The hidden unit takes on the value of 1, if the distance to a prototype vector is less than σ. Otherwise it takes on the value of 0. Discuss the relationship of this method to RBF networks, and its relative advantages/disadvantages.*

By using this approach, one is essentially using a discrete kernel function. The disadvantage of this approach is that there may be many points that take on the value of 0 for all hidden units. Such points will not be discriminated at all.

2. *Suppose that you use the sigmoid activation in the final layer to predict a binary class label as a probability in the output node of an RBF network. Set up a negative log-likelihood loss for this setting. Derive the gradient-descent updates for the weights in the final layer. How does this approach relate to the logistic regression methods discussed in Chapter 2? In which case will this approach perform better than logistic regression.*

The updates are identical to those of logistic regression used in Chapter 2, except that we are using the hidden units rather than the input units for modeling.

3. *Discuss why an RBF network is a supervised variant of a nearest-neighbor classifier.*

Consider an RBF network in which each prototype is a point from the training data. The RBF network is a weighted nearest neighbor classifier, where the weight is the product of the RBF similarity to training point (prototype) and the corresponding neural network weight from the prototype to the output node. The latter is learned in a supervised fashion.

4. *Discuss how you can extend the three multi-class models discussed in Chapter 2 to RBF networks. In particular discuss the extension of the (a) multi-class perceptron, (b) Weston-Watkins SVM, and (c) softmax classifier with RBF networks. Discuss how these models are more powerful than the ones discussed in Chapter 2.*

All gradient-descent updates and neural architectures are identical to the ones in Chapter 2, except that the RBF-engineered features are used instead of the input features. These models have nonlinearity, and therefore they are more powerful than the ones in Chapter 2, which cannot model nonlinear decision boundaries.

5. *Propose a method to extend RBF networks to unsupervised learning with autoencoders.*

In this case, we create an output layer with as many units as the number of RBF units. The target values in the output layer are set to the same values as the RBF units. We also have a *trained* hidden layer between the *unsupervised* RBF hidden layer and the output layer, which has fewer units than the layers on either side of it. The weights on either side of this layer are tied, as in a conventional autoencoder. These weights are trained with backpropagation.

6. *Suppose that you change your RBF network so that you keep only the top-k activations in the hidden layer, and set the remaining activations to 0. Discuss why such an approach will provide improved classification accuracy with limited data.*

This approach is a form of regularization of the hidden layer, and it will remove the effect of low activations that are noisy anyway. Therefore, the accuracy of the approach will improve when there is limited data.

7. *Combine the top-k method of constructing the RBF layer in Exercise 6 with the RBF autoencoder in Exercise 5 for unsupervised learning. Discuss why this approach will create representations that are better suited to clustering. Discuss the relationship of this method with spectral clustering.*

This approach reconstructs the hidden activations as in Exercise 5, except that the hidden activations are modified by keeping only the top-k activations and setting the remaining to 0. The approach is closely related to spectral clustering because it will not favor weak relationships between points and prototypes. The prototypes are already performing a type of soft clustering, and the weak connections between prototypes and points are broken in order to create a more well-defined clustering. The approach is closely related to unnormalized spectral clustering, which also breaks the inter-cluster links by setting weak entries in the kernel matrix to 0. In the case of spectral clustering, the prototypes are assumed to be the set of original data points. One difference is that spectral clustering uses mutual k-nearest neighbors, whereas our approach uses uni-directional nearest neighbors.

8. *The manifold view of outliers is to define them as points that do not naturally fit into the nonlinear manifolds of the training data. Discuss how you can use RBF networks for unsupervised outlier detection.*

The unsupervised learning method of Exercise 5 can be used. Points that have very high levels of residual errors are flagged as outliers. Note that this approach is also used in conventional neural networks. The main difference is that RBF networks allow nonlinear feature engineering with an RBF layer, rather than by using multiply layers of nonlinear units.

9. *Suppose that instead of using the RBF function in the hidden layer, you use dot products between prototypes and data points for activation. Show that a special case of this setting reduces to a linear perceptron.*

We use the linear activation in the output layer and then use the perceptron criterion. Furthermore, the prototypes are the individual data points. This approach reduces to a linear perceptron, because the dot product corresponds to the linear kernel, and an RBF network with all points as prototypes simulates a kernel perceptron.

10. *Discuss how you can modify the RBF autoencoder in Exercise 5 to perform semi-supervised classification, when you have a lot of unlabeled data, and a limited amount of labeled data.*

The RBF autoencoder of Exercise 5 is first trained with unlabeled data. Then, its decoder is removed and the innermost hidden layer (with reduced representation) is capped with a classification layer (with sigmoid or softmax activation, depending on class label). The weights of this added layer are trained with the labeled data. Furthermore, the RBF autoencoder weights can also be fine-tuned. The resulting approach provides a semi-supervised model for classification.

Chapter 6

Restricted Boltzmann Machines

1. *This chapter discusses how Boltzmann machines can be used for collaborative filtering. Even though discrete sampling of the contrastive divergence algorithm is used for learning the model, the final phase of inference is done using real-valued sigmoid and softmax activations. Discuss how you can use this fact to your advantage in order to fine-tune the learned model.*

   One can use the learned weights as initializations for the back-propagation algorithm. Since the trained RBM is being used like a conventional neural network at prediction time, it can be fine-tuned during training as well.

2. *Implement the contrastive divergence algorithms of a restricted Boltzmann machine. Also implement the inference algorithm for deriving the probability distribution of the hidden units for a given test example. Use Python or any other programming language of your choice.*

   This is an implementation exercise.

3. *Consider a Boltzmann machine without a bipartite restriction (of the RBM), but with the restriction that all units are visible. Discuss how this restriction simplifies the training process of the Boltzmann machine.*

   For computing the data-specific correlations in an update, one no longer needs to use Monte Carlo sampling. However, the model-specific correlations remain the same.

4. *Propose an approach for using RBMs for outlier detection.*

   It has been shown in the chapter how RBMs can be used for dimensionality reduction. Points with large reconstruction error are outliers.

5. *Derive the weight updates for the RBM-based topic modeling approach discussed in the chapter. Use the same notations.*

   The updates are similar to the case of collaborative filtering, because the models are similar.

6. *Show how you can extend the RBM for collaborative filtering (discussed in the chapter) with additional layers to make it more powerful.*

   We would add an additional hidden layer between the users and the movies. [One can also add multiple hidden layers, which are fully connected with one another.] The modeling is similar in terms of sharing the weights between different users and movies. Otherwise, the training is similar to a multi-layer Boltzmann machine.

7. *A discriminative Boltzmann machine is introduced for classification in the chapter. However, this approach is designed for binary classification. Show how you can extend the approach to multi-way classification.*

The binary logistic unit is replaced with a softmax unit. Subsequently, the log-likelihood objective function is set up with the softmax unit, and gradient descent is applied. The modifications to the multiway case are analogous to those in a conventional neural network.

8. *Show how you can modify the topic modeling RBM discussed in the chapter in order to create a hidden representation of each node drawn from a large, sparse graph (like a social network).*

Conceptually, a document-term binary matrix is similar to a node-node adjacency matrix in terms of sparsity. One can simply treat each node as a "document" with a bag of other node identifiers. This principle can be used to create the appropriate modifications of the RBM.

9. *Discuss how you can enhance the model of Exercise 8 to include information about a list of keywords drawn from each node.*

In this case, we have additional visible units for the keywords. Conceptually, each node is a "document" containing both node identifiers and keyword identifiers.

10. *Discuss how you can enhance the topic modeling RBM discussed in the chapter with multiple layers.*

Additional hidden layers are added between the visible units and the hidden units. The layers between these units are trained in the same way as in traditional RBMs.

Chapter 7

Recurrent Neural Networks

1. *Download the character-level RNN, and train it on the "*tiny Shakespeare*" data set available at the same location. Create outputs of the language model after training for (i) 5 epochs, (ii) 50 epochs, and (iii) 500 epochs. What significant differences do you see between the three outputs?*

   The outputs will tend to improve over time, as the parameters of the neural network get better trained.

2. *Consider an echo-state network in which the hidden states are partitioned into K groups with p/K units each. The hidden states of a particular group are only allowed to have connections to its own group in the next time-stamp. Discuss how this approach is related to an ensemble method in which K independent echo-state networks are constructed and the predictions of the K networks are averaged.*

   One can roughly decompose this network into K independent networks each of which interacts within its own subnetwork. However, the difference is that the output layer is trained jointly.

3. *Show how you can modify the feedforward architecture discussed in Chapter 2 in order to create a recurrent neural network that can handle temporal recommender systems. Implement this adaptation and compare its performance to the feedforward architecture on the Netflix prize data set.*

   We create a time-layered representation of this architecture in which we have hidden to hidden connections across time-layers. The architecture of each layer is similar to what is shown in Chapter 2. One can train this network with the BPTT algorithm.

4. *Consider a recurrent network in which the hidden states have a dimensionality of 2. Every entry of the 2×2 matrix W_{hh} of transformations between hidden states is 3.5. Furthermore, sigmoid activation is used between pairs of hidden states. Would such a matrix be more prone to the vanishing or the exploding gradient problem?*

   It is easy to show that the largest eigenvalue of the linear transformation matrix is 3.5. However, the derivative of the sigmoid activation is always less than 0.25. Since the value of 0.25×3.5 is less than 1, this recurrent network would be more prone to the vanishing gradient problem.

5. *Suppose that you have a large database of biological strings containing a sequence of nucleobases drawn from $\{A, C, T, G\}$. Some of these strings contain unusual mutations*

representing changes in the nucleobases. Propose an unsupervised method (i.e., neural architecture) using RNNs in order to detect these mutations.

Use a simple alphabet-centric language model like the tiny Shakespeare example in the text. Mutations will have low log likelihood.

6. *How would your architecture for the previous question change if you were given a training database in which the mutation positions in each sequence were tagged, and the test database was untagged?*

In this case, we use a model that classifies individual positions as "mutation" or "no mutation." Therefore, the output layer uses binary sigmoid units. Use the training database to learn the model, and use the test database to perform prediction.

7. *Discuss the types of pre-training in the machine translation approach with sequence-to-sequence learning.*

One can train *word2vec* in both the source and destination languages. Note that *word2vec* is a shallow model that is easily trained without the training problems typical in an RNN. Furthermore, pre-trained features are often available off-the-shelf. The pre-trained features can be used for both the input and output layer. The *word2vec* encoder can be used for the RNN encoder in input layer, and the *word2vec* decoder can be used for the RNN decoder in output layer. Note that we will need different *word2vec* models for the two languages.

8. *Consider a social network with a large volume of messages sent between sender-receiver pairs, and we are interested only in the messages containing an identifying keyword, referred to as a hashtag. Create a real-time model using an RNN, which has the capability to recommend hashtags of interest to each user, together with potential followers of that user who might be interested in receiving messages related to that hashtag. Assume that you have enough computational resources to incrementally train an RNN.*

There are several ways of constructing the architecture of the RNN. Therefore, the answer to this problem is not unique. However, for recommendations, it might make sense to train two RNNs separately. One architecture has input as the sender/receiver identifier, and the output as the hashtag-identifier and the receiver/sender identifier at the other end. Another uses the sender/receiver and hashtag as input and outputs the receiver/sender. Therefore, for each message we obtain two training points, as we treat them in a symmetric way (asymmetric models are also possible). An RNN is trained incrementally as sender-receiver pairs and hash-tags are received. The input of the first RNN uses the sender/receiver as input. The output of the first RNN uses two softmax units, one for the hash-tag and the other for the receiver/sender identifier. The input of the second RNN uses the sender/receiver and hashtag as input. The output of the second RNN outputs a receiver/sender identifier. At any given point, we can use both the models in "test mode" where a potential sender is input to the first RNN in order to obtain receiver identifier and hashtag recommendations. The potential sender and the recommended hastag is then input to the second RNN in order to obtain possible receivers for that hashtag.

9. *If the training data set is re-scaled by a particular factor, do the learned weights of either batch normalization or layer normalization change? What would be your answer if only a small subset of points in the training data set are re-scaled? Would the learned weights in either normalization method be affected if the data set is re-centered?*

Data set rescaling does not affect either normalization method. However, if only a subset of the data is scaled it will affect batch normalization but not layer normalization. This is because the normalization factors across a batch will change. Data set re-centering will not change batch normalization, but it will affect layer normalization. This is because layer normalization recenters with respect to activations in a layer, rather than over a mini-batch of instances.

10. *Consider a setting in which you have a large database of pairs of sentences in different languages. Although you have sufficient representation of each language, some* pairs *might not be well represented in the database. Show how you can use this training data to (i) create the same universal code for a particular sentence across all languages, and (ii) have the ability to translate even between pairs of languages not well represented in the database.*

We create an encoder for each language and a decoder for each language. All encoder-decoder pairs have the same dimensionality. For each pair of languages, we hook up the relevant encoder and decoder, and update their weights. This process is performed over the entire training database. The result will be a set of encoders that map to the same code for a particular sentence (across languages), and a set of decoders that read from the same code.

Chapter 8

Convolutional Neural Networks

1. *Consider a 1-dimensional time-series with values 2, 1, 3, 4, 7. Perform a convolution with a 1-dimensional filter 1, 0, 1 and zero padding.*

   Output is a sequence of size $5 - 3 + 1 = 3$. The sequence is $5, 5, 10$.

2. *For a one-dimensional time series of length L and a filter of size F, what is the length of the output? How much padding would you need to keep the output size to a constant value?*

   The problem is not different from the two-dimensional case. The length is $(L - F + 1)$. One must pad by $(F - 1)/2$ on both sides. Therefore, the length of the sequence will increase by $(F - 1)$.

3. *Consider an activation volume of size* $13 \times 13 \times 64$ *and a filter of size* $3 \times 3 \times 64$*. Discuss whether it is possible to perform convolutions with strides 2, 3, 4, and 5. Justify your answer in each case.*

   The value of $13 - 3$ must be divisible by the stride. Therefore, strides of 2 and 5 are possible, but strides of 3 and 4 are not.

4. *Work out the sizes of the spatial convolution layers for each of the columns of the table shown in the text on* VGGNet*. In each case, we start with an input image volume of* $224 \times 224 \times 3$.

   Since convolutional volumes are maintained by the use of 3×3 filters with a padding of 1, the spatial sizes remain fixed in convolutional layers and depth is given by the number of filters. The maxpooling always reduces spatial footprint by a factor of 2.

5. *Work out the number of parameters in each spatial layer for column D of the table.*

   A $3 \times 3 \times d$ filter has $9d$ parameters. With biases, we get $10d$. Substitute the value of d for each spatial layer to get the number of parameters for each filter. Then, multiply with the number of filters in the layer.

6. *Download an implementation of the* AlexNet *architecture from a neural network library of your choice. Train the network on subsets of varying size from the* ImageNet *data, and plot the top-5 error with data size.*

   This is an implementation exercise.

7. *Compute the convolution of the input volume in the upper-left of Figure 8.2 with the horizontal edge detection filter of Figure 8.1(b). Use a stride of 1 without padding.*

The final result of the convolution is a 5×5 output volume with the following entries:

4	6	4	-3	-3
0	-1	0	1	-2
-5	-6	1	-1	0
6	11	1	-3	4
3	3	4	4	2

8. *Perform a 4×4 pooling at stride 1 of the input volume in the upper-left corner of Figure 8.4.*

The result is another $4\times$ output volume, for which the entries are as follows:

7	7	5	7
8	8	5	7
8	8	6	7
8	8	6	7

9. *Discuss the various type of pretraining that one can use in the image captioning application discussed section 7.7.1 of Chapter 7.*

A pretrained convolutional neural network can be used for the image processing part. Any pretrained model on ImageNet, such as AlexNet may be used. Word2vec can be used to pretrain the input and output layers of the RNN. Furthermore, it is possible to pretrain a language model on the RNN, without hooking it up to the CNN. Therefore, the RNN receives two levels of pretraining, first with word2vec, and then with a language model. At this point, the models are hooked together and training is performed.

10. *You have a lot of data containing ratings of users for different images. Show how you can combine a convolutional neural network with the collaborative filtering ideas discussed in Chapter 2 to create a hybrid between a collaborative and content-centric recommender system.*

The answer to this question is not unique, as there are many possible ways of building such an architecture. The method in chapter 2 uses users as input and item ratings as output, while allowing incompleteness in the outputs for each user. Therefore, the method in chapter 2 implicitly trains neural networks that are specific to users. Here, we will train neural networks that are specific to items. Therefore, the input is an engineered representation of the item, and the output is the rating of each user for that item. However, since only a subset of the ratings are available for a particular item, the outputs for users who have not rated that item will be missing. We first pretrain a convolutional neural network on ImageNet, and then remove the output layer, so that the final layer of the convolutional neural network is the set of features from the fully connected layer. Then, we connect m outputs to that , where m is the number of users. The outputs can be linear units providing the numerical ratings. First, we will pre-train the weights between the features and the specified ratings of the users. Then, we will fine-tine the convolutional network and the weights from fully connected layer to user ratings together. In this way, the features learned by the convolutional network will become more sensitive to the recommendation application. In fact, it is the final stage of training the convolutional neural network and the ratings jointly that incorporate the collaborative aspects of the ratings into the learned features. As a result, the system will have characteristics of both a content-centric and collaborative system.

Chapter 9

Deep Reinforcement Learning

1. *The chapter gives a proof of the likelihood ratio trick for the case in which the action a is discrete. Show the result for the case in which the action a is continuous.*

   The proof is identical to that discussed in the chapter, except that the integral needs to be used instead of discrete summation.

2. *Throughout this chapter, a neural network, referred to as the policy network, has been used in order to implement the policy gradient. However, one can also use softmax regression to predict the same probabilities. Discuss the advantage of using a neural network over softmax regression in different settings.*

   A softmax regression can be viewed as a special type of neural network with one layer. However, a generic neural network allows more complex representation learning with greater number of layers, when the inputs are not easily related to the outputs. In other words, using a neural network provides better capacity of learning.

3. *You have two slot machines, each of which has an array of 100 lights. The probability distribution of the reward from playing each machine is an unknown (and possibly machine-specific) function of the pattern of lights that are currently lit up. Playing a slot machine changes its light pattern in some well-defined but unknown way. Discuss why this problem is more difficult than the multi-armed bandit problem. Design a deep learning solution to optimally choose machines in each trial that will maximize the average reward per trial at steady-state.*

   This problem is more difficult because we have a notion of a state, and the payoffs of the slot machines depend on the state. We can have a deep learner, which takes in the state of each machine as input and output an optimal value for each action. The Bellman's equations can be used to decide the optimal action.

4. *Consider the well-known game of rock-paper-scissors. Human players often try to use the history of previous moves to guess the next move. Would you use a Q-learning or a policy-based method to learn to play this game? Why? Now consider a situation in which a human player samples one of the three moves with a probability that is an unknown function of the history of 10 previous moves of each side. Propose a deep learning method that is designed to play with such an opponent. Would a well-designed deep learning method have an advantage over this human player? What policy should a human player use to ensure probabilistic parity with a deep learning opponent?*

Policy gradients is the appropriate approach to this game because of its stochastic nature. A deterministic policy can be exploited. The optimal policy is to simply output one of the three options at random. Any other policy can be exploited by a deep learner.

5. *Consider the game of tic-tac-toe in which a reward drawn from* $\{-1, 0, +1\}$ *is given at the end of the game. Suppose you learn the values of all states (assuming optimal play from both sides). Discuss why states in non-terminal positions will have non-zero value. What does this tell you about credit-assignment of intermediate moves to the reward value received at the end?*

States in non-terminal positions have non-zero values because of their strategic value. For example, setting a trap in tic-tac-toe earns a reward.

6. *Write a Q-learning implementation that learns the value of each state-action pair for a game of tic-tac-toe by repeatedly playing against human opponents. No function approximators are used. Assume that you can initialize each Q-value to 0 in the table.*

This is an implementation exercise.

7. *The two-step TD-error is defined as follows:*

$$\delta_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$$

(a) Propose a TD-learning algorithm for the 2-step case.

(b) Propose an on-policy n*-step learning algorithm like SARSA. Show that the update is truncated variant of the equation in the book after setting* $\lambda = 1$*. What happens for the case when* $n = \infty$*?*

(c) Propose an off-policy n*-step learning algorithm like Q-learning and discuss its advantages/disadvantages with respect to (b).*

The updates are made to state $\overline{X}_{t-2}$ using $\delta_t^{(2)}$. One can also use a truncated version of the update used for the TD-lambda algorithm in the book, except that λ is set to 1, and we use only the most recent two terms. One can generalize this idea to the n-step algorithm as well. The off-policy n-step algorithm is the same as TD-Leaf. It is computationally more expensive, but it is better optimized.

Chapter 10

Advanced Topics in Deep Learning

1. *What are the main differences in the approaches used for training hard-attention and soft-attention models?*

   Hard-attention models are not differentiable, and therefore they generally use reinforcement learning models. Soft attention models can often be implemented with conventional backpropagation.

3. *Discuss how the k-means algorithm is related to competitive learning.*

   The k-means algorithm is related to competitive learning in the sense that updates to the centroids are not done using gradient descent or a learning rate. Rather, an approach is leveraged in which the centroids are set to minimize the current value of the objective function.

4. *Implement a Kohonen self-organizing map with (i) a rectangular lattice, and (ii) a hexagonal lattice.*

   This is an implementation exercise.

5. *Consider a two-player game like GANs with objective function $f(x, y)$, and we want to compute $min_x max_y f(x, y)$. Discuss the relationship between $min_x max_y f(x, y)$ and $max_y min_x f(x, y)$. When are they equal?*

   The max-min is always less than or equal to min-max. The best example is to consider the function $\sin(x + y)$, for which the min-max is 1, but the max-min is -1. In general, the function needs to be convex in the minimization variables and concave in the maximization variables for the two to be equal. This is a well known result in minimax optimization.

6. *Consider the function $f(x, y) = sin(x + y)$, where we are trying to minimize $f(x, y)$ with respect to x and maximize with respect to y. Implement the alternating process of gradient descent and ascent discussed in the book for GANs to optimize this function. Do you always get the same solution over different starting points?*

   In this case, the final result will be sensitive to the starting point. One can expect to have stable termination points only in cases where the function $f(x, y)$ is convex in the minimization variables and concave in the maximization variables. This is not the case for a function such as $\sin(x + y)$. (See previous exercise.) This result also means that the performance of the GAN gradient descent will be sensitive to the topology of the loss function.