Dept. of Computer Science, BITS Pilani, K.K. Birla Goa Campus
Neural Networks (BITS F312)
B.E. Comprehensive Test, Sem-I, AY 2019-20
December 14, 2019
Total Marks: 40
Time Limit: 3 Hour

Question 1 [9 marks]

Consider neural networks with a single hidden layer. Each unit is associated with a set of weights $\mathbf{w}$ and uses the following activation functions:

- Hard-threshold activation function with a constant threshold θ:

$$h(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

- Linear activation function:

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

where, $\mathbf{x} = (x_1, x_2)$ is the input vector.

(a) Let $x_{1,2} \in \{0, 1\}$. What is the size of the space of all boolean functions $f(\mathbf{x})$? Show working for your answer. [1]

(b) For each boolean function $f(\mathbf{x})$ above, can we build a neural network to compute f. If yes, give a constructive proof. Otherwise, justify. [6]

(c) Justify whether and how can the neural network described in question 1 represent a polynomial function of (i) degree 1, and (ii) degree greather than 1. [2]

Solution:

(a) There are 4 posible inputs: $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$. For each input there can be 4 possible outputs: $(0, 0, 0, 0)$ throughto $(1, 1, 1, 1)$. Each output corresponds to a boolean function. Therefore, there are 16 ($2^4 =$) boolean functions.

(b) Any boolean function can be expressed using three basic boolean functions such as NOT, AND, and OR. If we can give constructive proof that we can build neural network for these three functions, we are done. For this, we have to satisfy the following equations that uses the hard-threshold activation:

$$h(\mathbf{x}) = \begin{cases} 1 & w_1x_1 + w_2x_2 \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

Goal is to find w_1, w_2, and θ. For the boolean functions AND and OR, let us fix $\boxed{w_1 = 1}$ and $\boxed{w_2 = 1}$. Can we find a θ that would satisfy the following equations for AND:

x_1	x_2	y	*rule* for y
0	0	0	$0 < 0$
0	1	0	$w_2 < \theta$
1	0	0	$w_1 < \theta$
1	1	1	$w_1 + w_2 \geq \theta$

This table concludes, $\theta > 1$ and $\theta \geq 2$. Therefore, the final value is: $\boxed{\theta \geq 2}$.

For OR: These equations are $1 \geq \theta$ (twice) and $2 \geq \theta$. The final value is: $\boxed{\theta \geq 1}$.

For NOT: The value of w_i (it doesn't matter 1 or 2) has to be found out. Fixing it to 1 does not work.

x_1	$y = \neg x_1$	*rule* for y
0	1	$0 \geq \theta$
1	0	$w_1 < \theta$

The above table results in the inequality: $0 \geq \theta > w_1$. If we fix $\boxed{w_1 = -1}$, then $\boxed{\theta = -0.5}$.

(c) (a) Polynomial of degree one can be represted using the linear activation function $f(\mathbf{x}) = h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + w_0$. (b) Polynomial of degree greather than one cannot be represented by this neural network.

Question 2 [3 marks]

One-word type answers are expected.

(a) We want to perform the following four different operations on a neural net model. State what effect will these operations generally have on the bias and variance of the model. Use three options: 'increases', 'decreases', 'no effect' to fill the table appropriately.

Operation	Bias	Variance
Regularising the model parameters	?	?
Increasing the size of the hidden layers	?	?
Training the network with dropout	?	?
Getting more training data	?	?

[2]

(b) Because pooling layers do not have parameters, they do not affect the backpropagation (derivatives) calculation. Is this statement 'True' or 'False'? [1]

Solution:

(a)

Operation	Bias	Variance
Regularising the model parameters	Increases	Decreases
Increasing the size of the hidden layers	Decreases	Increases
Training the network with dropout	Increases	Decreases
Getting more training data	No change	Decreases

(b) False: Because, the cell from which the value was selected (in case of max-pool) can only pass the gradient backward. Other cells don't pass any gradient. Similar argument for average-pool. Pooling layer uses a masking for gradient passing — a kind of flagging method.

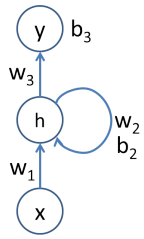
Question 3 [7 marks]

Story-type answers are strongly discouraged.

(a) Recall that a RNN takes in an input vector $\mathbf{x}(t)$ and a state vector $\mathbf{h}(t-1)$ and returns a new state vector $\mathbf{h}(t)$ and an output vector $\mathbf{y}(t)$. Refer the side figure. The equations are:

$$h(t) = f(w_1 x(t) + w_2 h(t-1) + b_2)$$
$$y(t) = g(w_3 h(t) + b_3)$$

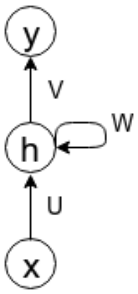
Where f and g are activations functions at hidden and output layers respectively. Now, given



$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$
$$g(x) = x$$
$$\text{and } h(0) = 0$$

Find the values or conditions on weights such that RNN initially outputs 0, and as soon as it receives input 1, it starts producing 1 as output no matter what the input is. For example, if input sequence is 00100101, then output sequence would be 00111111. [3]

(b) Consider a vanilla RNN network as shown below.



(i) In standard feed-forward network, we used to have a loss function L. For RNN, if we use the same loss function, will it yield consistent result? If your answer is YES, then justify your answer. If your answer is NO, then what is the modification required in L? [2]

(ii) For a dataset $\mathbf{D} = (\mathbf{x}_t, y_t)_1^k$ (for some $k \in \mathbb{N}$), show how information is propagated in the RNN for $k = 3$. [2]

Solution:

(a) The conditions are:

$$b_3 = 0$$
$$w_3 + b_3 = 1 \ (i.e.\ w_3 = 1)$$
$$b2 < 0$$
$$w_1 + w_2 + b_2 \geq 0$$
$$w_1 + b_2 \geq 0$$
$$w_2 + b_2 \geq 0$$

(b) Answers:

(i) We have a data $D = (\mathbf{x}_t, y_t)$, where we assume that the data is ordered temporally. Thus, we define the loss function to be

$$L(U, W, V; h_0) = \sum_{t=1}^{T} (y(t) - f(\mathbf{x}_t, h_{t-1}; U, W, V))$$

where h_{t-1} is the previous reccurent state. The initial state h_0 needs to be specified and the problem depends on it as well.

(ii) The unfolded diagram is required: It should contain 3 RNN cells connected with weight W.

Question 4 [10 marks]

These questions expect mathematical and conceptual justifications.

(a) You have constructed a very large and complex neural network model for representation learning. Learning sparse representations for inputs has a lot of benefits in deep learning such as extracting discriminating representation for any given input. Let $L(\mathbf{x}, \hat{\mathbf{x}})$ denote the loss function for your present model. How will you modify this loss function such that your model is forced to learn sparse representation? [2]

(b) A contractive autoencoder is a heavily regularized encoder[1] in which the hidden representation is not very sensitive to changes in input values. Consider an autoencoder with single hidden layer. Let the input vector be defined as $\mathbf{x} = (x_1, \ldots, x_d)$, and the hidden units be $\mathbf{h} = (h_1, \ldots, h_k)$ (usually, $k \geq d$). Let's denote the weights of the autoencoder as a vector of w_{ij}s, where $1 \leq i \leq d; 1 \leq j \leq k$.

The reconstruction loss for a single input instance is defined as:

$$L = \sum_{i=1}^{d} (x_i - \hat{x}_i)^2$$

where, $\hat{\mathbf{x}}$ is the reconstructed version of the input.

(i) The contractive autoencoder defines a new loss (let's call it L') by adding a regularising term to L. Mathematically define L'. [2]

(ii) Derive the simplest expression of L' when the hidden layer uses a sigmoid nonlinearity. [3]

(iii) Show that when hidden layer uses a linear activation, the new loss (L') is equal to the weight-decayed version of L. [3]

Solution:

(a) Sparse representations can be learned by enforcing a constraint on the number of hidden (representation) units activated given an input instance. This is done by penalizing the *activations* of the neural network, so that only a small subset of the neurons fire for any given data instance. The simplest way to achieve sparsity is to impose an L_1-penalty on the hidden units. Therefore, the original loss function L is modified to the regularized loss function L' as

$$L' = L + \lambda \sum_{i=1}^{m} |h_i|$$

Here, m is the total number of units in the representation layer, and h_i is the value of the ith hidden unit, and λ is the regularisation parameter.

(b) Here are the solutions:

[1] normally, overcomplete

(i) The new loss adds the derivatives of h_js w.r.t. to x_is to L. So,

$$L' = \sum_{i=1}^{d}(x_i - \hat{x}_i)^2 + \frac{\lambda}{2}\sum_{i=1}^{d}\sum_{j=1}^{k}\left(\frac{\partial h_j}{\partial x_i}\right)^2$$

where, λ is the penalty factor.

(ii) When the hidden units use sigmoid activation, the value

$$h_j = \sigma\left(\sum_{i=1}^{d} w_{ij}x_i\right) \ \forall j$$

so,

$$\frac{\partial h_j}{\partial x_i} = w_{ij}h_j(1-h_j) \ \forall i,j$$

rewriting L':

$$L' = \sum_{i=1}^{d}(x_i - \hat{x}_i)^2 + \frac{\lambda}{2}\sum_{j=1}^{k} h_j^2(1-h_j)^2\sum_{i=1}^{d} w_{ij}^2$$

(iii) When the hidden layer uses the linear activation,

$$\frac{\partial h_j}{\partial x_i} = w_{ij}$$

So, the simplest form is

$$L' = \sum_{i=1}^{d}(x_i - \hat{x}_i)^2 + \frac{\lambda}{2}\sum_{i=1}^{d}\sum_{j=1}^{k} w_{ij}^2$$

which resembles with L_2-regularisation or weight decay.

Question 5 [11 marks]

Let $\mathbf{x} = (x_1, \ldots, x_m)$ denote the visible (input) units, and $\mathbf{h} = (h_1, \ldots, h_n)$ denote the hidden units. Restricted Boltzmann Machine (RBM) is an energy-based stochastic neural network that is fully connected between visible and hidden units, and is parameterised by $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$; where, $\mathbf{W} = \{w_{ij} : 1 \le i \le m, 1 \le j \le n\}$ are the weight parameters between visible and hidden units; $\mathbf{b} = \{b_j : 1 \le j \le n\}$ are biases in the hidden layer; $\mathbf{c} = \{c_i : 1 \le i \le m\}$ are the biases in the visible layer.

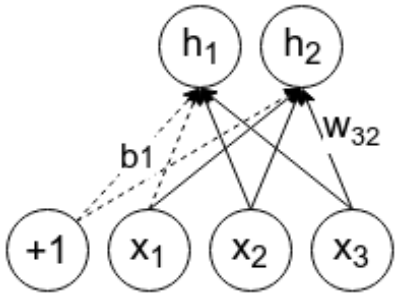
(a) Mathematically describe how does a RBM model the joint distribution $\mathsf{P}(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta})$? [2]

(b) Let $\mathbf{x} = (x_1, x_2, x_3)$ and $x_{1,2,3} \in \{0, 1\}$. We define a 'unidirectional' RBM with two hidden units $\{h_1, h_2\}$ as shown in the figure below. The weight parameters of this networks are as follows: $w_{11} = 0.5$, $w_{12} = 0.5$, $w_{21} = -0.5$, $w_{22} = 0.5$, $w_{31} = -0.5$, $w_{32} = -0.5$; hidden layer biases: $b_1 = 1$, $b_2 = -1$; visible layer biases: $c_1 = c_2 = c_3 = 0$. The hidden units have sigmoid activation function. Given the inputs and model parameters above:

(i) Write the expression for Boltzmann energy for this unidirectional RBM. [1]

(ii) Obtain the joint distribution $\mathsf{P}(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta})$. [6]

(iii) Calculate the marginal probability $P(x_1 = 1, \mathbf{h})$. [2]



(The arrow direction is $\mathbf{x} \rightarrow \mathbf{h}$. b_1 and w_{32} are for representation purpose only.)

Solution:

(a) RBM uses an energy function to model the joint distribution. The energy function is called the Boltzmann energy defined as:

$$\begin{aligned} E(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta}) &= -(\mathbf{hWx} + \mathbf{bh} + \mathbf{cx}) \\ &= -\left(\sum_{i=1}^{m} \sum_{j=1}^{n} h_j w_{ij} x_i + \sum_{j=1}^{n} b_j h_j + \sum_{i=1}^{m} c_i x_i \right) \end{aligned}$$

The joint distribution is defined as:

$$\mathsf{P}(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta}) = \frac{e^{-E(\mathbf{x},\mathbf{h};\boldsymbol{\theta})}}{Z(\boldsymbol{\theta})}$$

where, $Z(\boldsymbol{\theta})$ is the normalising constant (also called *partition function*) and is calculated by summing the energy over all visible and hidden units:

$$Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}'} \sum_{\mathbf{h}'} e^{-E(\mathbf{x}',\mathbf{h}';\boldsymbol{\theta})}$$

(b) (i) The unidirectional RBM will not have the last term as in question(a):

$$\begin{aligned} E(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta}) &= -(\mathbf{hWx} + \mathbf{bh}) \\ &= -\left(\sum_{i=1}^{m} \sum_{j=1}^{n} h_j w_{ij} x_i + \sum_{j=1}^{n} b_j h_j \right) \end{aligned}$$

(ii) The goal is to fillup a table of probabilities. I have written a code to do this.

```
x = [0 0 0;0 0 1;0 1 0;0 1 1;1 0 0;1 0 1;1 1 0;1 1 1];
W = [0.5 0.5;-0.5 0.5;-0.5 -0.5];
b = [1 -1];
Z = 0; %normalizing constant
Z1 = 12.137593; %final Z is used to print last column
P = 0; %to check whether it's a distro
for i=1:8
    a(1) = x(i,:) * W(:,1) +  b(1);
```

```
    a(2) = x(i,:) * W(:,2) +  b(2);
    h(1) = sigmoid(a(1));
    h(2) = sigmoid(a(2));

    E = -((a - b) * h' + b * h');
    expE = exp(-E);

    Z = Z + expE;
    P = P + expE/Z1;
    fprintf('%d %d %d | %f %f | %f | %f | %f\n',...
    x(i,1),x(i,2),x(i,3),h(1),h(2),E,expE,expE/Z1);
end
Z1 = Z;
fprintf('Z = %f\n',Z);
fprintf('P = %f\n',P);
```

x_1	x_2	x_3	h_1	h_2	E	e^{-E}	$p(\mathbf{x},\mathbf{h})$
0	0	0	0.731059	0.268941	-0.462117	1.587431	0.130786
0	0	1	0.622459	0.182426	-0.037591	1.038307	0.085545
0	1	0	0.622459	0.377541	-0.122459	1.130273	0.093122
0	1	1	0.5	0.268941	0.268941	0.764188	0.06296
1	0	0	0.817574	0.377541	-1.037591	2.822411	0.232535
1	0	1	0.731059	0.268941	-0.462117	1.587431	0.130786
1	1	0	0.731059	0.5	-0.731059	2.077278	0.171144
1	1	1	0.622459	0.377541	-0.122459	1.130273	0.093122
						$Z = 12.137593$	$\sum = 1.0$

The last column is the answer. (Step marks: 4 marks for columns 4–7; 1 mark for Z; 1 marks for column 8)

(iii) Marginal probability $P(x_1, \mathbf{h}) = \sum_{x_1} \mathsf{P}(\mathbf{x} = x_1, \mathbf{h}) = 0.250393 + 0.143975 + 0.149534 + 0.082499 = 0.626401 \approx 0.63$.

—END—