1 Neural Networks

1.1 Neural network for regression

Figure 1 shows a two-layer neural network which learns a function \( f : X \to Y \) where \( X = (X_1, X_2) \in \mathbb{R}^2 \). The weights \( w = \{w_1, \ldots, w_6\} \) can be arbitrary. There are two possible choices for the function implemented by each unit in this network:

- \( S \): signed sigmoid function \( S(a) = \text{sign} [\sigma(a) - 0.5] = \text{sign} [\frac{1}{1+\exp(-a)} - 0.5] \)
- \( L \): linear function \( L(a) = ca \)

where in both cases \( a = \sum_i w_i X_i \).

1. Assign proper activation functions (S or L) to each unit in Figure 1 so this neural network simulates a linear regression: \( Y = \beta_1 X_1 + \beta_2 X_2 \).

2. Assign proper activation functions (S or L) for each unit in Figure 1 so this neural network simulates a binary logistic regression classifier: \( Y = \arg \max_y P(Y = y | X) \), where \( P(Y = 1 | X) = \frac{\exp(\beta_1 X_1 + \beta_2 X_2)}{1+\exp(\beta_1 X_1 + \beta_2 X_2)} \), and \( P(Y = -1 | X) = \frac{1}{1+\exp(\beta_1 X_1 + \beta_2 X_2)} \). Derive \( \beta_1 \) and \( \beta_2 \) in terms of \( w_1, \ldots, w_6 \).

3. Assign proper activation functions (S or L) to each unit in Figure 1 so this neural network simulates a boosting classifier which combines two logistic regression classifiers, \( f_1 : X \to Y_1 \) and \( f_2 : X \to Y_2 \), to produce its final prediction: \( Y = \text{sign}[\alpha_1 Y_1 + \alpha_2 Y_2] \). Use the same distribution in problem 1.1.2 for \( f_1 \) and \( f_2 \). Derive \( \alpha_1 \) and \( \alpha_2 \) in terms of \( w_1, \ldots, w_6 \).

1.2 Convolutional neural networks

1. Count the total number of parameters in LeNet. How many parameters in all of the convolutional layers? How many parameters in all of the fully-connected layers?

Note:

(a) The filter size of each convolutional and pooling(subsampling) layer:
   - C1: 5 × 5 (i.e., each unit of C1 has a 5 × 5 receptive field in its preceding layer);
   - S2: 2 × 2;
   - C3: 5 × 5;
   - S4: 2 × 2;
3 Neural Network and Regression (18 pts)

Consider a two-layer neural network to learn a function $f : X \rightarrow Y$ where $X = \langle X_1, \ldots \rangle$.

$P(Y = -1|X) = \frac{1}{1+\exp(\beta_1 X_1 + \beta_2 X_2)}$.

3. (3 pts) Following problem 3.2, derive $\beta_1$ and $\beta_2$ in terms of $w_1, \ldots, w_6$.

4. (4 pts) Assign proper activation functions (S or L) for each unit in this network:

- $S$: signed sigmoid function
- $L$: linear function

function implemented by each unit in this network.

2. In a convolutional layer the units are organized into planes, each of which is called a feature map. The units within a feature map (indexed $q$) have different inputs, but all share a common weight vector, $w^{(q)}$. A convolutional network is usually trained through backpropagation. Let $J^{(q)}$ be the number of units in the $q$th feature map, $z_j^{(q)}$ the activation of the $j$th unit, $x_j^{(q)}$ the $i$th input for the $j$th feature map, $w_i^{(q)}$ the $i$th element of $w^{(q)}$, $L$ the training loss. Derive the gradient of $w_i^{(q)}$.

1.3 Gradient vanishing/explosion

In this problem we will study the difficulty of back-propagation in training deep neural networks. For simplicity, we consider the simplest deep neural network: one with just a single neuron in each layer, where the output of the neuron in the $j$th layer is $z_j = \sigma(a_j) = \sigma(w_j z_{j-1} + b_j)$. Here $\sigma$ is some activation function whose derivative on $x$ is $\sigma'(x)$. Let $m$ be the number of layers in the neural network, $L$ the training loss.

1. Derive the derivative of $L$ w.r.t. $b_1$ (the bias of the neuron in the first layer).

2. Assume the activation function is the usual sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$. The weights $w$ are initialized to be $|w_j| < 1$ ($j = 1, \ldots, m$).

   (a) Explain why the above gradient ($\partial L/\partial b_1$) tends to vanish ($\rightarrow 0$) when $m$ is large.
   (b) Even if $|w|$ is large, the above gradient would also tend to vanish, rather than explode ($\rightarrow \infty$). Explain why. (A rigorous proof is not required.)

3. One of the approaches to (partially) address the gradient vanishing/explosion problem is to use the rectified linear (ReL) activation function instead of the sigmoid. The ReL activation function implemented by each unit in this network: one with just a single neuron in each layer, where the output of the neuron in the $j$th layer is $z_j = \max(0, a_j) = \max(0, w_j z_{j-1} + b_j)$. Here $\max(0, x)$ is a common implementation of the ReL activation function.

   Figure 1: A two-layer neural network.

   Figure 2: LeNet network.
function is $\sigma(x) = \max\{0, x\}$. Explain why ReL can alleviate the gradient vanishing problem as faced by sigmoid.

4. A second approach to (partially) address the gradient vanishing/explosion problem is layer-wise pre-training. Restricted Boltzmann machine (RBM) is one of the widely-used models for layer-wise pre-training. Figure 3 shows an example of RBM which includes $K$ hidden units $h_i$ and $J$ input units $v_j$. Let us define the joint distribution as the following general form:

$$P(v, h) = \frac{1}{Z} \exp\left(\sum_i \theta_i \phi_i(v, h)\right)$$

where $Z = \sum_{v,h} \exp\left(\sum_i \theta_i \phi_i(v, h)\right)$ is the normalization term; $\phi_i(v, h)$ are some features; $\theta_i$ are the parameters corresponding to the weights in the RBM. Consider the simplest learning algorithm, gradient descent. Show that

$$\frac{\partial \log P(v)}{\partial \theta_i} = \sum_h \phi_i(v, h)P(h|v) - \sum_{v,h} \phi_i(v, h)P(v, h).$$

Figure 3: A restricted Boltzmann machine.

2 Recurrent networks

Follow this guide to learn more on RNN. Then, write a complete training procedure for a word-level LSTM recurrent network on Penn Treebank dataset. Design and train your network so that it will satisfy the 2 following goals:

- Final word-level perplexity on the test-set should be $< 120$
- Number of trainable parameters (weights) within the network should be $< 3M$

In addition to the requirements set, use the networks achieving these goals to generate 5 random sentences continuations to (no apostrophes): "Buy low, sell high is the"...

2.1 Recurrent implementation

You may use any Torch recurrent implementation (or write your own). You might use:

Element-Research’s rnn

We recommend Torch, but we also allow other packages like

\footnote{(Adapted from 096260 Deep learning course)}
2.2 Data

The Penn Treebank data (in text format, preprocessed to word-level) is available here under:
- ptb.train.txt
- ptb.valid.txt
- ptb.test.txt

You should train your network only with **ptb.train.txt**

2.3 Submission instructions

Submission will be in pairs (course partners) and will contain a short pdf report containing:

- Model architecture description, training procedure (data augmentation, regularization, optimization details etc).
- Two convergence graphs for your final model - for error and loss as a function of time (epochs). Each graph should depict both training and test performance.
- A short summary of your attempts and conclusions.

In addition, you should also supply:

- Code able to reproduce your results - we might test it on different variants on these datasets.