COMPSCI 671D Fall 2019
Homework 4
Vinayak Gupta
vg101

1 EM Algorithm for Topic Modeling (35 points)

In this question we will try to design an algorithm for discovering the abstract topics that occur in a set of documents. In the problem, we have a set of M abstract documents in the universe, $\mathcal{D}$, which are mixtures of latent topics. We also have a dictionary of words, $\mathcal{W}$, with size N. Intuitively, $\mathcal{W}$ is the list of all unique words that occur at least once within $\mathcal{D}$. Using these two sets we can denote the number of times word w_j occurs in document d_i as $n(w_j, d_i)$. It follows that we can represent a document d_i as a vector of size N, each entry of which corresponds to how many times word w_j appears in it. The topics z are latent variables in a topic set $\mathcal{Z}$ sized K. Intuitively, if a document is about a certain topic, it will include some particular words with higher probability. For example, "music", "show" and "film" appear frequently in documents about Arts while "school", "student" and "teachers" usually occur if the document is about Education. Meanwhile, a document can be a mixture of several topics, e.g., a document can be about arts education for high school students. Inspired by the intuition, a reasonable approach to model the generative process is as follows.

$$\Pr(d_i, z_k, w_j) = \Pr(d_i)\Pr(z_k|d_i)\Pr(w_j|z_k)$$

which means the topics only depend on the documents and the words only depend on topics. For computational convenience, let $\Pr(d_i)$ be a uniform distribution and $\Pr(z_k|d_i)$ and $\Pr(w_j|z_k)$ be Multinomial distributions. i.e.

$$\Pr(d_i) = \frac{1}{M}$$

$$\Pr(z_k|d_i) = \alpha_{ik},\ \sum_{k=1}^{K} \alpha_{ik} = 1$$

$$\Pr(w_j|z_k) = \beta_{kj},\ \sum_{j=1}^{N} \beta_{kj} = 1.$$

We are interested in learning α_{ik} and β_{kj} because they are substantively interesting: α_{ik} represents the proportion of topic k that makes up document i, and β_{kj} the probability of seeing word j under topic k. Therefore, a single word w_j in a document d_i is generated in this way: (1) Choose a topic z_k from the topic distribution $Pr(z_k|d_i)$, the probability of choosing topic k is α_{ik}; (2) Choose a word from the vocabulary distribution $p(w_j|z_k)$ in this topic, the probability is β_{kj}. We want to learn these parameters by maximizing the likelihood of the data using the EM algorithm.

a) For our set of N documents and W word of choices, write down the log-likelihood of the model above, i.e. $\log(p(D = d_i, W = w_j|\alpha, \beta))$.

Remember that in the EM (Expectation-Maximization) algorithm, we can figure out the parameters and the hidden variables by iteratively computing (1) the distribution of latent variables using the old parameters and (2) find new parameters that maximize the likelihood using the old latent variable distribution. So, you can do the following:

b) E-step: Derive the distribution of latent variable $p(z_k|w_j, d_i, \alpha^{old}, \beta^{old})$ by fixing the old parameters.

c) M-step: Find the α^{new} and β^{new} that optimize the log likelihood using $p(z_k|w_j, d_i, \alpha^{old}, \beta^{old})$ as the distribution of z.

You would iterate these until convergence to get the final parameters in practice.

Just so you know, the model you have just been working with in this problem is a very famous and very popular model. :)

1. EM Algorithm for Topic Modeling

Universe : D => M documents
Dictionary : W => N words
Number of times w_j in document d_i => $n(w_j, d_i)$
Topic set : Z => K topics

$$Pr(d_i, z_k, w_j) = Pr(d_i)\, Pr(z_k|d_i)\, Pr(w_j|z_k)$$
$$Pr(d_i) = 1/M$$
$$Pr(z_k|d_i) = \alpha_{ik}, \quad \sum_{k=1}^{K} \alpha_{ik} = 1$$
$$Pr(w_j|z_k) = \beta_{kj}, \quad \sum_{j=1}^{N} \beta_{kj} = 1$$

(a) likelihood $= P(\{D_1, W_1\} = \{d_1, w_1\}, \{D_1, W_2\} = \{d_1, w_2\}, \ldots, \{D_M, W_N\} = \{d_M, w_N\} | \Theta)$
where $\Theta = (\alpha, \beta)$

$$= \prod_{i=1}^{M} \prod_{j=1}^{N} Pr(D = d_i, W = w_j|\Theta)^{n(w_j, d_i)}$$

where $n(w_j, d_i)$ term comes from the fact that there are $n(w_j, d_i)$ occurences of word w_j in document d_i; so we can group them (multiply them) together while calculating likelihood

$$\text{log likelihood} = \sum_{i=1}^{M} \sum_{j=1}^{N} \log Pr(D = d_i, W = w_j|\Theta)^{n(w_j, d_i)}$$

$$= \sum_{i=1}^{M} \sum_{j=1}^{N} n(w_j, d_i) \log Pr(D = d_i, W = w_j|\Theta) \quad (1)$$

By law of total probability (Defination of joint probability)

$$\log Pr(D = d_i, W = w_j|\Theta) = \log Pr(W = w_j|D = d_i, \Theta) \cdot Pr(D = d_i|\Theta)$$

$Pr(D = d_i|\Theta) = Pr(d_i)$, since Θ and d_i are independent
Also, $Pr(d_i) = 1/M$

so $\log Pr(D = d_i, W = w_j|\Theta) = \log \frac{Pr(W = w_j|D = d_i, \Theta)}{M}$

We also have latent variables (topics):

$z_1, \ldots, z_K$

By law of Total Probability:

$$Pr(W=w_j \mid D=d_i, \theta) = \sum_{k=1}^{K} Pr(W=w_j \mid z=z_k, D=d_i, \theta) \cdot Pr(z=z_k \mid D=d_i, \theta)$$

$$= \sum_{k=1}^{K} \beta_{kj} \alpha_{ik} \quad \text{as } Pr(W=w_j \mid z=z_k, D=d_i, \theta) = \beta_{kj}$$
$$Pr(z=z_k \mid D=d_i, \theta) = \alpha_{ik}$$

So (1) becomes:

Ans

$$\text{log likelihood} = \sum_{i=1}^{M} \sum_{j=1}^{N} n(w_j, d_i) \log \sum_{k=1}^{K} \frac{\beta_{kj} \alpha_{ik}}{M}$$

(b) Since there is a log of sum in the likelihood, we can use Jensen's Inequality and EM method to maximize the log likelihood.

$$\log \sum_{k=1}^{K} \frac{\beta_{kj} \alpha_{ik}}{M} \Rightarrow \log \sum_{k=1}^{K} Pr(z=z_k \mid W=w_j, D=d_i, \theta_t) \frac{\beta_{kj} \alpha_{ik}}{M \cdot Pr(z=z_k \mid W=w_j, D=d_i, \theta_t)}$$

By Jensen's Inequality:

$$\geq \sum_{k=1}^{K} Pr(z=z_k \mid W=w_j, D=d_i, \theta_t) \log \frac{\beta_{kj} \alpha_{ik}}{M \cdot Pr(z=z_k \mid W=w_j, D=d_i, \theta_t)}$$

$$\text{so } A(\theta, \theta_t) = \sum_{i=1}^{M} \sum_{j=1}^{N} n(w_j, d_i) \sum_{k=1}^{K} \gamma_{ijk} \log \frac{\beta_{kj} \alpha_{ik}}{M \gamma_{ijk}}$$

where $\gamma_{ijk}^{t} = Pr(z=z_k \mid W=w_j, D=d_i, \theta_t)$

E-step: compute γ_{ijk}^{t+1}

$$\gamma_{ijk}^{t+1} = P_{\theta_t}(z=z_k \mid W=w_i, D=d_i, \theta_t) = \frac{P_{\theta_t}(W=w_i \mid z=z_k, D=d_i, \theta_t)\, P_{\theta_t}(z=z_k \mid D=d_i, \theta^t)}{\sum_{k=1}^{K} P_{\theta_t}(W=w_j \mid z=z_k, D=d_i, \theta_t)\, P_{\theta_t}(z=z_k \mid D=d_i, \theta^t)}$$

$$\boxed{\gamma_{ijk}^{t+1} = \frac{\beta_{kj}^{t}\, \alpha_{ik}^{t}}{\sum_{k=1}^{K} \beta_{kj}^{t}\, \alpha_{ik}^{t}}} \quad \underline{\text{Ans}} \Rightarrow \gamma_{ijk}^{t+1} = \frac{\beta_{kj}^{old}\, \alpha_{ik}^{old}}{\sum_{k=1}^{K} \beta_{kj}^{old}\, \alpha_{ik}^{old}}$$

(c)

M-step: Update α, β by setting derivatives of A to 0

Since, there are constraints for α and β, we need to use Lagrangian:

for α $$L(\theta, \theta_t) = A(\theta, \theta_t) + \lambda\left(1 - \sum_{k=1}^{K} \alpha_{ik}\right)$$

$$\frac{dL(\theta,\theta_t)}{d\alpha_{(ik)'t}} = \frac{dA}{d\alpha_{(ik)'t}} - \lambda = 0$$

$$= \frac{d}{d\alpha_{(ik)'t}}\left(\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{K} n(w_j, d_i)\, \gamma_{ijk} \log \frac{\beta_{kj}^{new}\, \alpha_{ik}^{new}}{M\, \gamma_{ijk}}\right) = \lambda$$

Since β_{kj}^{new}, M, γ_{ijk} do not depend on $\alpha_{(ik)'t}$, remove them from above eqn.

$$= \frac{d}{d\alpha_{(ik)'t}}\left(\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{K} n(w_j, d_i)\, \gamma_{ijk} \log \alpha_{ik}^{new}\right) = \lambda$$

$$= \sum_{j=1}^{N} \frac{n(w_j, d_i)\, \gamma_{ijk}}{\alpha_{ik}^{new}} = \lambda$$

$$\alpha_{ik}^{new} = \sum_{j=1}^{N} \frac{n(w_j, d_i)\, \gamma_{ijk}}{\lambda}$$

Also, $$\sum_{k=1}^{K} \alpha_{ik}^{new} = 1 = \sum_{k=1}^{K}\sum_{j=1}^{N} \frac{n(w_j, d_i)\, \gamma_{ijk}}{\lambda}$$

$$\lambda = \sum_{k=1}^{K}\sum_{j=1}^{N} n(w_j, d_i)\, \gamma_{ijk}$$

$$\therefore \quad \alpha_{ik}^{new} = \frac{\sum_{j=1}^{N} n(w_j, d_i)\, \gamma_{ijk}}{\sum_{k=1}^{K} \sum_{j=1}^{N} n(w_j, d_i)\, \gamma_{ijk}}$$

for β $\quad L(\theta, \theta_t) = A(\theta, \theta_t) + \lambda \left(1 - \sum_{j=1}^{N} \beta_{kj}\right)$

$$\frac{dL(\theta, \theta_t)}{d\beta_{kj't}} = \frac{dA}{d\beta_{kj't}} - \lambda = 0$$

$$= \frac{d}{d\beta_{kj't}} \left(\sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=1}^{K} n(w_j, d_i)\, \gamma_{ijk} \log \beta_{kj}^{new} \right) = \lambda$$

because m, α_{ik}^{new}, γ_{ijk} do not depend on $\beta_{kj't}$, so we removed them

$$\Rightarrow \sum_{i=1}^{M} \frac{n(w_j, d_i)\, \gamma_{ijk}}{\beta_{kj}^{new}} = \lambda$$

$$\beta_{kj}^{new} = \frac{\sum_{i=1}^{M} n(w_j, d_i)\, \gamma_{ijk}}{\lambda}$$

$$\sum_{j=1}^{N} \beta_{kj}^{new} = \sum_{j=1}^{N} \sum_{i=1}^{M} \frac{n(w_j, d_i)\, \gamma_{ijk}}{\lambda} = 1$$

$$\text{So} \quad \lambda = \sum_{j=1}^{N} \sum_{i=1}^{M} n(w_j, d_i)\, \gamma_{ijk}$$

$$\beta_{kj}^{new} = \frac{\sum_{i=1}^{M} n(w_j, d_i)\, \gamma_{ijk}}{\sum_{j=1}^{N} \sum_{i=1}^{M} n(w_j, d_i)\, \gamma_{ijk}}$$

2 Neural Networks (65 points)

In this problem you will get the chance to construct a neural network with architecture $784 - H - H - 1$, where H is the number of hidden nodes you choose. This neural network can be used for binary classification, such 0, 1 digits classification for MNIST. The model can be represented as

$$f(\mathbf{x}; \theta) : \mathbb{R}^{784} \to [0, 1]$$

where $\theta = \{\mathbf{W}_1 \in \mathbb{R}^{784 \times H}, \mathbf{b}_1 \in \mathbb{R}^H, \mathbf{W}_2 \in \mathbb{R}^{H \times H}, \mathbf{b}_2 \in \mathbb{R}^H, \mathbf{w}_3 \in \mathbb{R}^H, b_3 \in \mathbb{R}, \}$. Explicitly, $f(\mathbf{x})$ is

$$\mathbf{h}_1 = \sigma\left(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1\right)$$

$$\mathbf{h}_2 = \sigma\left(\mathbf{W}_2^\top \mathbf{h}_1 + \mathbf{b}_2\right)$$

$$f(\mathbf{x}) = \sigma\left(\mathbf{w}_3^\top \mathbf{h}_2 + b_3\right)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$, and it is an element-wise operator, which means if $\mathbf{x} = (x^1, x^2, ..., x^d) \in \mathbb{R}^d$, we have $\sigma(\mathbf{x}) = (\sigma(x^1), \sigma(x^2), ..., \sigma(x^d))$.

The loss function for this model (or the negative log-likelihood) is the usual one:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} (y_i \log f(\mathbf{x}_i) + (1 - y_i) \log(1 - f(\mathbf{x}_i)))$$

a) Backpropogation

Derive the gradients $\nabla_{\mathbf{w}_3}\mathcal{L}(\theta)$, $\nabla_{b_3}\mathcal{L}(\theta)$, $\nabla_{\mathbf{W}_2}\mathcal{L}(\theta)$, $\nabla_{\mathbf{b}_2}\mathcal{L}(\theta)$, $\nabla_{\mathbf{W}_1}\mathcal{L}(\theta)$, $\nabla_{\mathbf{b}_1}\mathcal{L}(\theta)$ by backpropagation.

2(a)

$z_1 = W_1^T x + b_1$; $z_2 = W_2^T h_1 + b_2$; $z_3 = W_3^T h_2 + b_3$

where $h_1 = \sigma(z_1)$; $h_2 = \sigma(z_2)$ and σ is a sigmoid function

$$L(\theta) = -\sum_{i=1}^{N} (y_i \log f(x_i) + (1-y_i) \log (1 - f(x_i)))$$

$$\frac{\partial L(\theta)}{\partial f(x)} = \frac{f(x) - y}{f(x)(i_N - f(x))^T} \quad ; \text{ where } i_N \text{ is } (1 \ldots\ldots 1)$$

$$\frac{\partial L(\theta)}{\partial w_3} = \nabla_{w_3} L(\theta) = \frac{\partial L(\theta)}{\partial f(x)} \frac{\partial f(x)}{\partial z_3} \frac{\partial z_3}{\partial w_3}$$

$$\frac{\partial z_3}{\partial w_3} = h_2 \quad ; \quad \frac{\partial f(x)}{\partial z_3} = f(x)(i - f(x))^T$$

$$\Rightarrow \nabla_{w_3} L(\theta) = \frac{f(x) - y}{f(x)(i_N - f(x))^T} \cdot f(x)(i - f(x))^T \cdot h_2$$

$$= \boxed{(f(x) - y) h_2}$$

$$\nabla_{b_3} L(\theta) = \frac{\partial L(\theta)}{\partial f(x)} \frac{\partial f(x)}{\partial z_3} \frac{\partial z_3}{\partial b_3}$$

$$= (f(x) - y) \cdot 1 = \boxed{f(x) - y}$$

$$\nabla_{w_2} L(\theta) = \frac{\partial L(\theta)}{\partial f(x)} \frac{\partial f(x)}{\partial z_3} \frac{\partial z_3}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial w_2}$$

$$= (f(x) - y) w_3 h_2 (i_H - h_2)^T h_1$$

$$\nabla_{b_2} L(\theta) = \frac{\partial L(\theta)}{\partial f(x)} \frac{\partial f(x)}{\partial z_3} \frac{\partial z_3}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial b_2}$$

$$= (f(x) - y) w_3 h_2 (i_H - h_2)^T$$

$$\nabla_{w_1} L(\theta) = \frac{\partial L(\theta)}{\partial f(x)} \frac{\partial f(x)}{\partial z_3} \frac{\partial z_3}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

$$= (f(x) - y) w_3 h_2 (i_H - h_2)^T w_2 h_1 (i - h_1)^T x$$

$$\nabla_{b_1} L(\theta) = \frac{\partial L(\theta)}{\partial f(x)} \frac{\partial f(x)}{\partial z_3} \frac{\partial z_3}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial b_1}$$

$$= (f(x) - y) w_3 h_2 (i_{H} - h_2)^T w_2 h_1 (i - h_1)^T$$

As we can see, the formulas recursively are

$$\nabla_{z_{l-1}} L(\theta) = \nabla_{z_l} L(\theta) w_l h_{l-1} (i - h_{l-1})^T$$

$$\nabla_{w_l} L(\theta) = \nabla_{z_l} L(\theta) h_{l-1}$$

$$\nabla_{b_l} L(\theta) = \nabla_{z_l} L(\theta)$$

b) Weight Initialization

Neural Networks are normally trained by gradient descent based optimization algorithms, for which we have to give initial values for our parameters (weight and bias). The bias parameters are usually initialized as 0's, while the weight parameters are usually initialized by independently sampling from $N(0, \sigma^2)$.

(i) What are the potential problems if we choose σ to be either too small or too large?

(ii) One nice property we would like our weights to have is that by proper initialization, we want the values before activation to be similar in distribution for each layer. It turns out we can achieve this by choosing σ properly. Now suppose we use ReLU activation in our neural network and the inputs are normalized. We want to choose the proper σ_l for the l-th hidden layer such that

$$Var(z_l) = Var(z_{l-1}).$$

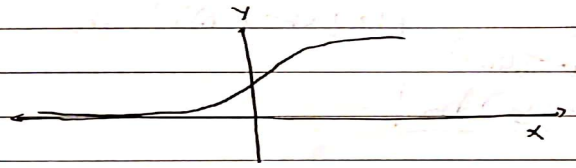
We assume the neurons in each layer are mutually independent and share the same distribution and we use z_l to represent the random variable that generates $\mathbf{z}_l$, where $\mathbf{z}_l = \mathbf{W}_l^T \mathbf{h}_{l-1} + \mathbf{b}_l$ and $\mathbf{h}_{l-1} = \max(0, \mathbf{z}_{l-1})$ and $\mathbf{W}_l \in \mathbb{R}^{n_{l-1} \times n_l}$.

Find σ_t such that $Var(z_l) = Var(z_{l-1})$ and prove your result.

(b) (i) Weight parameters are initialized from $N(0, \sigma^2)$

If σ is too small, all parameters are initialized close to 0. This causes the derivative wrt loss function nearly same for all weights. This cascades into subsequent iterations and all hidden neural nodes act similarly for all iterations thus defeating the purpose of neural networks.

If σ is very large, since the mean is 0, the individual (magnitudes) weights are initialized with very high values. If the weights are very high or very low, $z = W^T h + b$ is also very high or very low. Activation fns. like sigmoid have a vanishing gradient at extremely high/low values as can be seen by their graph. This causes slow learning and network might get stuck at a local optima.



Sigmoid graph.

(ii) $\text{Var}(\text{Relu}) = \frac{1}{2}\text{Var}(x)$ where $\text{RELU} = \max(0, x)$

because $f(x) = x$'s right part (+ve part) is same as RELU's +ve part and the negative part is just 0. This can also be shown as:

$$\text{Var}(\text{Relu}) = \underset{\substack{\text{(inputs normalized)} \\ E(\text{Relu}) = 0}}{E(\text{Relu}^2)} = \int_{-\infty}^{\infty} \text{Relu}^2 \, p(\text{Relu}) \, dx$$

$$= \int_{-\infty}^{\infty} \max(0, x)^2 \, p(x) \, dx$$

$$= \int_{0}^{\infty} \max(0, x)^2 \, p(x) \, dx = \int_{0}^{\infty} x^2 p(x) \, dx$$

$$= \frac{1}{2} \int_{-\infty}^{\infty} x^2 p(x) \, dx \quad \text{since } x^2 \text{ is symmetric}$$

Also, $E(x) = 0$ as for $f(x) = x$, $E(x) = 0$

so $= \frac{1}{2} \int_{-\infty}^{\infty} (x - E(x))^2 p(x) \, dx$

$= \frac{1}{2}\text{Var}(x)$

Hidden layers

$$Var(z_k^d) = Var\left(\sum_j w_{kj}^l a_j^{l-1}\right)$$

z_k^d: kth observation

$$= \sum_{j=1}^{H} \left(E[w_{kj}^d]^2 Var(a_j^{d-1}) + Var(w_{kj}^l) E[a_j^{d-1}]^2 + Var(w_{kj}^d) Var(a_j^{d-1}) \right)$$

$$= \sum_{j=1}^{H} Var(w_{jk}^d) Var(a_j^{d-1})$$ because $E[w_{kj}^d] = 0$ (mean = 0)

$E[a_j^{d-1}] = 0$ (normalized = 0)

$$= h\, Var(W^d) Var(a^{d-1})$$

as variance of all individual weights same and variance of all a's also same

so $Var(z^d) = h\, Var(W^d) \dfrac{Var(z^{d-1})}{2}$ $\left(\text{as } Var(a^{d-1}) = \dfrac{Var(z^{d-1})}{2}\right)$

so $Var(W^d) = \dfrac{2}{h}$ if $Var(z^d) = Var(z^{d-1})$

so $\sigma_k^2 = \dfrac{2}{h}$

$$\boxed{\sigma_k = \sqrt{\frac{2}{h}}}$$ Ans

c) Implement your Neural Network

Use the preprocessed MNIST data provided in the attachment. MNIST is a dataset of images of handwritten digits that also contain labels for the number they correspond to. We want to train a neural network to learn to recognize these handwritten digits. Use gradient descent with the gradients you have derived in part a) to train a neural network on the data. Report the accuracy for the network on the train and test data. Since the sample size is large, we can approximate the true gradient with stochastic gradient for computational convenience:

$$\nabla \mathcal{L}(\theta) \approx \frac{1}{B} \sum_{j=1}^{B} \nabla \mathcal{L}(\theta)_j$$

which means at each step, instead of calculating the gradients with all samples in the dataset, we randomly pick a mini-batch of B samples and calculate the gradient using these samples. In implementing the neural network, it would be beneficial to build and train the model in a general way. That is, build a NN with d hidden layers such that we can easily modify d.

The code was developed using the framework from the coursera deeplearning.ai course I did https://www.coursera.org/learn/neural-networks-deep-learning?fbclid=IwAR0bRdW3V3cUGbR1vi1v7-SCgJBGpg4ag9_2mmbOX0n-FEJtu8FDsHtUtLQ

```
import numpy as np
import h5py
import matplotlib.pyplot as plt
import time
import scipy.io as sio
from PIL import Image
from scipy import ndimage
import pylab
np.random.seed(1)

def sigmoid(Z):
    A = 1/(1+np.exp(-Z))
    cache = Z
    return A, cache


def sigmoid_diff(Z):

    s = 1/(1+np.exp(-Z))
    dZ =  s * (1-s)
    return dZ

def initialize_parameters_deep(layer_dims):
    parameters = {}
    L = len(layer_dims)
    np.random.seed(1)
    for l in range(1, L):

        parameters['W' + str(l)] = np.random.randn(layer_dims[l], layer_dims[l-1])
    * 0.01
        parameters['b' + str(l)] = np.zeros((layer_dims[l], 1))

    return parameters

def linear_forward(A, W, b):
```

```
    Z = np.dot(W,A)+b
    cache = (A, W, b)
    return Z, cache

def linear_activation_forward(A_prev, W, b):

    Z, linear_cache = linear_forward(A_prev, W, b)
    A, activation_cache = sigmoid(Z)

    cache = (linear_cache, activation_cache)

    return A, cache

def L_model_forward(X, parameters):
    caches = []
    A = X
    L = len(parameters) // 2


    for l in range(1, L):
        A_prev = A
        A, cache = linear_activation_forward(A_prev,parameters['W' + str(l)] ,
    parameters['b' + str(l)])
        caches.append(cache)

    AL, cache = linear_activation_forward(A, parameters['W' + str(L)] , parameters
    ['b' + str(L)])
    caches.append(cache)

    return AL, caches

def compute_cost(AL,Y):
    m=Y.shape[1]
    cost=-(1/m)*(np.dot(Y,np.transpose(np.log(AL)))+np.dot(1-Y,np.transpose(np.log
    (1-AL))))
    cost=np.squeeze(cost)
    return cost

def linear_backward(dZ, cache):

    A_prev, W, b = cache
    m=A_prev.shape[1]
    dW = (1/m)*(np.dot(dZ,np.transpose(A_prev)))
    db = (1/m)*(np.dot(dZ,np.ones((dZ.shape[1],1))))
    dA_prev = np.dot(np.transpose(W),dZ)

    return dA_prev, dW, db

def linear_activation_backward(dA, cache):

    linear_cache, activation_cache = cache


    dZ = dA*sigmoid_diff(activation_cache)
    dA_prev, dW, db = linear_backward(dZ, linear_cache)

    return dA_prev, dW, db

def L_model_backward(AL, Y, caches):
```

```
91
92      grads = {}
93      L = len(caches) # the number of layers
94      m = AL.shape[1]
95      Y = Y.reshape(AL.shape) # after this line, Y is the same shape as AL
96
97      dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))
98
99      current_cache = caches[L-1]
100     grads["dA" + str(L-1)], grads["dW" + str(L)], grads["db" + str(L)] =
        linear_activation_backward(dAL, current_cache)
101
102     for l in reversed(range(L-1)):
103
104         current_cache = caches[l]
105         dA_prev_temp, dW_temp, db_temp = linear_activation_backward(grads["dA" +
        str(l + 1)], current_cache)
106         grads["dA" + str(l)] = dA_prev_temp
107         grads["dW" + str(l + 1)] = dW_temp
108         grads["db" + str(l + 1)] = db_temp
109
110     return grads
111
112 def update_parameters(parameters, grads, learning_rate):
113
114     L = len(parameters) // 2
115
116     for l in range(L):
117         parameters["W" + str(l+1)] = parameters["W" + str(l+1)]-(learning_rate*
        grads["dW" + str(l + 1)])
118         parameters["b" + str(l+1)] = parameters["b" + str(l+1)]-(learning_rate*
        grads["db" + str(l + 1)])
119
120     return parameters
121
122 def L_layer_model(X, Y, layers_dims, check,learning_rate = 0.1,num_iterations
        =3000):
123
124
125     costs = []
126     parameters = initialize_parameters_deep(layers_dims)
127
128
129     for i in range(0, num_iterations):
130
131
132         AL, caches = L_model_forward(X, parameters)
133
134
135         cost = compute_cost(AL, Y)
136
137         grads = L_model_backward(AL, Y, caches)
138
139         if(check==1):
140             return np.linalg.norm(grads['dW1'],ord='fro')
141
142         parameters = update_parameters(parameters, grads, learning_rate)
143
144
```

```
    return parameters

def predict(X, y, parameters):


    m = X.shape[1]
    n = len(parameters)
    p = np.zeros((1,m))


    probas, caches = L_model_forward(X, parameters)


    for i in range(0, probas.shape[1]):
        if probas[0,i] > 0.5:
            p[0,i] = 1
        else:
            p[0,i] = 0


    print("Accuracy: "  + str(np.sum((p == y)/m)))

    return p

tmp = sio.loadmat("mnist_all.mat")
x1=tmp['train0']
x2=tmp['train1']
x3=tmp['test0']
x4=tmp['test1']

y1=[[0] for i in range(x1.shape[0])]
y2=[[1] for i in range(x2.shape[0])]
y3=[[0] for i in range(x3.shape[0])]
y4=[[1] for i in range(x4.shape[0])]

train_x_orig=np.concatenate((x1,x2))
test_x_orig=np.concatenate((x3,x4))
train_y=np.concatenate((y1,y2))

test_y=np.concatenate((y3,y4))


layers_dims = [784]
d=2
for j in range(d):
    layers_dims.append(20)
layers_dims.append(1)

train_y=np.transpose(train_y)
test_y=np.transpose(test_y)

m_train = train_x_orig.shape[0]
num_px = train_x_orig.shape[1]
m_test = test_x_orig.shape[0]

print ("Number of training examples: " + str(m_train))
print ("Number of testing examples: " + str(m_test))
print ("Each image is of size: (" + str(num_px) + ", " + str(num_px) + ", 3)")
print ("train_x_orig shape: " + str(train_x_orig.shape))
```

```
print ("train_y shape: " + str(train_y.shape))
print ("test_x_orig shape: " + str(test_x_orig.shape))
print ("test_y shape: " + str(test_y.shape))



train_x_flatten = train_x_orig.reshape(train_x_orig.shape[0], -1).T
test_x_flatten = test_x_orig.reshape(test_x_orig.shape[0], -1).T


train_x = train_x_flatten/255.
test_x = test_x_flatten/255.

print ("train_x's shape: " + str(train_x.shape))
print ("test_x's shape: " + str(test_x.shape))

parameters = L_layer_model(train_x, train_y, layers_dims,0, num_iterations = 3000)



pred_train = predict(train_x, train_y, parameters)

pred_test = predict(test_x, test_y, parameters)

layers_dims = [784]
for i in range(1,11):

    layers_dims.append(20)
    layers_dims.append(1)
    if i==1:
        normval=[L_layer_model(train_x, train_y, layers_dims, 1,num_iterations =
    1)]
    else:
        normval.append(L_layer_model(train_x, train_y, layers_dims,1,
    num_iterations = 1))
    layers_dims.pop()

print(normval)


fig = plt.figure(figsize=(20,10))

ax = fig.add_subplot(2, 2, 1)

line, = ax.plot(normval, color='blue', lw=2)

ax.set_yscale('log')
plt.xlabel('Number of hidden layers')
plt.ylabel('Log of frobenius norm of dW1')
pylab.show()
```

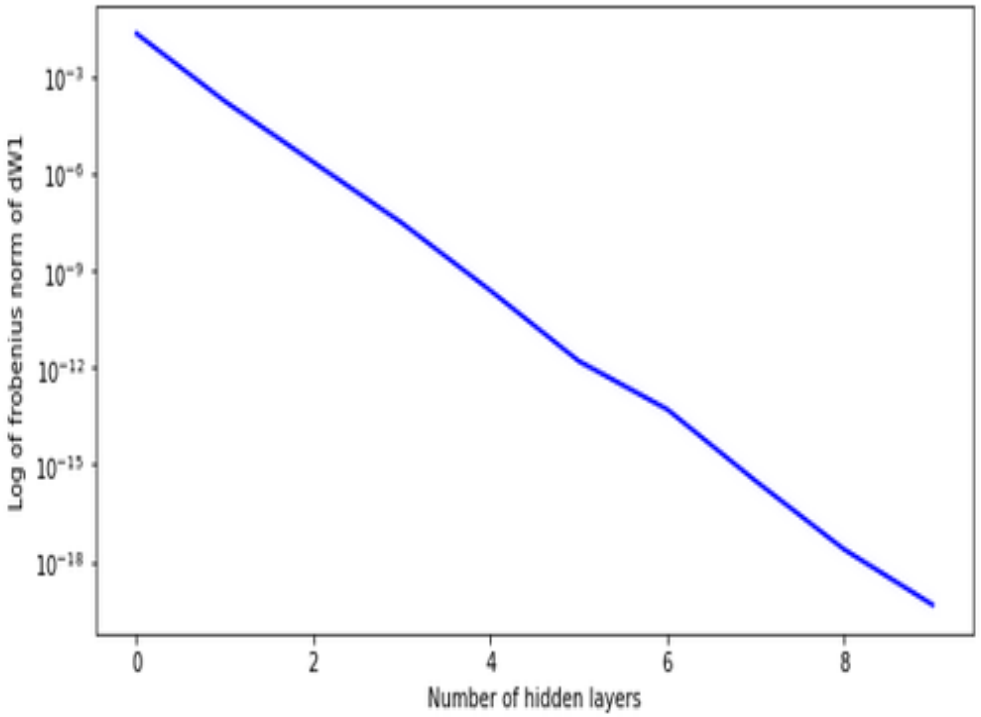
```
Accuracy: 0.9978681405448085
Accuracy: 0.9990543735224583
```

d) Vanishing Gradient

(i) In our current model, we have 2 hidden layers. What is the magnitude of $\nabla_{\mathbf{W}_1}\mathcal{L}(\theta)$ (you can use Frobenius norm or induced matrix norm)? Now try to modify the the number of hidden layers d, from 1 to 10. What happens to the magnitude of $\nabla_{\mathbf{W}_1}\mathcal{L}(\theta)$ when we increase d? Plot the magnitude against d on log scale. In this question, you don't need to train the whole neural network, just initialize it with random weights, e.g. $\mathcal{N}(0, 1)$, and calculate the gradient.

The above code in c also contains the code for d(i). The graph of log of Frobenius Norm with number of hidden layers are shown below along with their actual values. We can see that the Norm decreases with increasing hidden layers.

```
[0.020929657601947354,
 0.0001685396683827383 4,
 2.206016544453013e-06,
 2.9407276490753038e-08,
 2.408563213511275e-10,
 1.5851636785460938e-12,
 5.1170501426214545e-14,
 3.127652475763494e-16,
 2.3980611774373226e-18,
 4.6875823257695946e-20]
```



(ii) Suppose that all weights are finite and the weight matrices $\mathbf{W}_2, \mathbf{W}_3, ..., \mathbf{W}_d$ are diagonalizable matrices in which the absolute value of the eigenvalues are upper-bounded by $4 - \epsilon$. Prove that

$$\lim_{d \to \infty} \nabla_{\mathbf{W}_1} \mathcal{L}(\theta) = \mathbf{0}$$

d (ii) In a), we got the following relation:

$$\nabla_{z_{\ell-1}} L(\theta) = \nabla_{z_\ell} L(\theta)\, w_\ell\, h_{\ell-1} (i - h_{\ell-1})^T$$

where $i = [1 \ldots 1]$

We can use Cauchy - Schwartz inequality on the following eqn:

$$\frac{\nabla_{z_{\ell-1}} L(\theta)}{\nabla_{z_\ell} L(\theta)} = w_\ell\, h_{\ell-1} (i - h_{\ell-1})^T$$

Using inequality:

$$\left\| \frac{\nabla_{z_{\ell-1}} L(\theta)}{\nabla_{z_\ell} L(\theta)} \right\|_2 = \| w_\ell\, h_{\ell-1} (i - h_{\ell-1})^T \|_2 \leq \| w_\ell \|_2 \, \| h_{\ell-1} (i - h_{\ell-1})^T \|_2$$

$\| h_{\ell-1} (i - h_{\ell-1})^T \|_2$ can have maximum value $1/4$ as $h(1-h)$ if $h \in (0,1)$ is $1/4$.

This means that $\left\| \frac{\nabla_{z_{\ell-1}} L(\theta)}{\nabla_{z_\ell} L(\theta)} \right\|_2$ is bounded by $1 - \left(\frac{4-\epsilon}{4}\right)$ as $\| w_\ell \|_2$ is bounded by $4 - \epsilon$. (Because eigenvalues are bounded by $4 - \epsilon$)

Now, suppose there are d hidden layers, then

$$\left\| \frac{\nabla_{z_{\text{some}_2}} L(\theta)}{\nabla_{z_{d+1}} L(\theta)} \right\|_2 = \left\| \frac{\nabla_{z_1} L(\theta)}{\nabla_{z_2} L(\theta)} \cdot \frac{\nabla_{z_2} L(\theta)}{\nabla_{z_3} L(\theta)} \cdots \frac{\nabla_{z_d} L(\theta)}{\nabla_{z_{d+1}} L(\theta)} \right\|_2$$

By using cauchy - schwartz inequality, we get

$$\leq \left\| \frac{\nabla_{z_1} L(\theta)}{\nabla_{z_2} L(\theta)} \right\|_2 \left\| \frac{\nabla_{z_2} L(\theta)}{\nabla_{z_3} L(\theta)} \right\|_2 \cdots \left\| \frac{\nabla_{z_d} L(\theta)}{\nabla_{z_{d+1}} L(\theta)} \right\|_2$$

$$= \left(\frac{4-\epsilon}{4}\right)^d < 1^d$$

so this $\frac{4-\epsilon}{4} < 1$ and as $d \to \infty$ $\left(\frac{4-\epsilon}{4}\right)^d \to 0$

Hence, $\lim_{d \to \infty} \left\| \frac{\nabla_{z_1} L(\theta)}{\nabla_{z_{d+1}} L(\theta)} \right\|_2 = 0$

Also, $\| \nabla_{z_{dH}} L(\theta) \| = \| f(x) - y \|$

This value is finite as it depends only on observations which are constant, so $\| \nabla_{z_d} L(\theta) \|_2 \to 0$

Also, $\nabla_{w_d} L(\theta) = \nabla_{z_d} L(\theta) h_{d-1}$

Again by Cauchy-Schwartz:

$$\| \nabla_{w_d} L(\theta) \| = \| \nabla_{z_d} L(\theta) h_{d-1} \|_2 \leq \| \nabla_{z_d} L(\theta) \|_2 \| h_{d-1} \|_2$$

Any Sigmoid fn. can have max. 2 norm 1 so $\| h_{d-1} \|_2 \leq 1$

$$\therefore \| \nabla_{w_1} L(\theta) \|_2 \leq \| \nabla_{z_1} L(\theta) \|_2 \| h_{d-1} \|_2 < \| h_{d-1} \|_2$$

$\| \nabla_{w_1} L(\theta) \|_2 \to 0$ as $d \to \infty$ because $\| \nabla_{z_1} L(\theta) \|_2 \to 0$

so $\nabla_{w_1} L(\theta) \to 0$

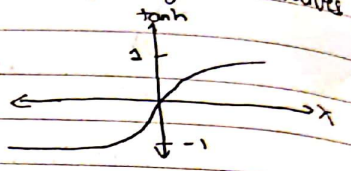
Hence, $\boxed{\lim_{d \to \infty} \nabla_{w_1} L(\theta) = 0}$,

(iii) The result above is known as the vanishing gradient problem in the feedforward neural network. Will there still be such a problem if we use a tanh activation? What if we use a ReLU activation?

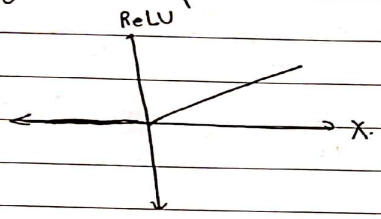
Give an intuitive explanation of why vanishing gradient happens in light of the shapes of activation functions and provide 2 ways of dealing with this problem.

d(iii) Yes, tanh behaves in a similar fashion to sigmoid function. While sigmoid function maps $R \to [0,1)$, while tanh maps $R \to (-1, 1)$. Other than that the shapes are very similar with flat gradients for very high/low values. Also, the formula for tanh is:

$$\tanh(x) = 2\sigma(2x) - 1$$



However, in case of ReLU, the gradient is non zero for activated part ($x > 0$). There is no vanishing gradient problem in case of ReLU.

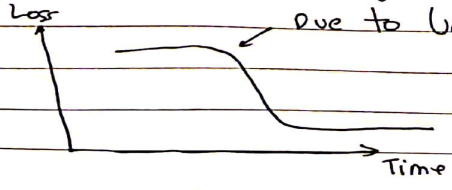


Vanishing Gradient can be dealt with in the following ways:

1) Batch Normalization: Outputs of several nodes (mini-batch) in same layer are normalized. The mean and standard deviation are treated as separate parameters which are required to be learned. This process is done before non-linear activation function and adds regularization and helps gradient flow in the network.

2) Leaky ReLU: It also removes vanishing gradients as it works similar to ReLU ($\max(0.1x, x)$).

3. Initialization of network weights is also important.



e) Sigmoid Activation

One potential problem of using the sigmoid activation function for hidden layers is that it maps from $\mathbb{R}$ to $[0, 1]$, which means the output of the sigmoid function is always positive. What happens to the signs of the gradients of hidden layer weights in this situation if we only have one example? What if we are adding gradients up across a batch of data?

2(e) If we are using only one example, the gradients are given by the following formula:

$$\frac{dE}{dw_{a,b}} = \frac{dE}{do_b} \cdot \frac{do_b}{dnet_b} \cdot \frac{dnet_b}{dw_{a,b}}$$

$$= \left(\sum_{l \in L} \delta_l w_{b,l} \right) o_b(1 - o_b) o_a$$

where l are the output nodes from b, a is the node from where weight originates, b is the node to which it comes.

Suppose, there are 2 nodes (a_1, a_2 from which weight originates) and one to which converges b, then

$$\frac{dE}{dw_{a_1,b}} = \left(\sum_{l \in L} \delta_l w_{b,l} \right) o_b(1 - o_b) o_{a_1}$$

$$\frac{dE}{dw_{a_2,b}} = \left(\sum_{l \in L} \delta_l w_{b,l} \right) o_b(1 - o_b) o_{a_2}$$

The formula shows that sign of $\frac{dE}{dw_{a,b}}$ will remain same for all incoming weights to a node. This is a limitation as either all weights increase / decrease simultaneously.

If we are adding gradients up across a batch of data, this issue is AVOIDED. Since, we sum gradients, while the sign of gradients for individual observation for all incoming weights are same, their sum may have different signs. This can be shown by the following example.

Obs 1: $\frac{dE}{dw_{a_1,b}} = -2$ $\frac{dE}{dw_{a_2,b}} = -2$

Obs 2: $\frac{dE}{dw_{a_1,b}} = +1$ $\frac{dE}{dw} = +3$

The signs of gradients are same but while summing across observations:

$$\frac{dE}{dW_{a_1,b}} = -2+1 = -1 \quad ; \quad \frac{dE}{dW_{a_2,b}} = -2+3 = +1$$

We are able to get different signs (because of different magnitudes of σ_{a_1} and σ_{a_2})