Introduction to Deep Learning
Assignment 1

David Barber

November 13, 2018

General

- You may work in groups of up to 5 students.
- Only students that are taking the module for credit may be part of a group.
- You must submit only a single pdf. This must include all of your working and code and any printouts or plots required from your code.
- We will not run your code, so you need to print the output of the code, for example the plots required.
- You may use whichever programming language you wish.
- Except where stated, you must not use any AutoDiff package (such as Tensorflow) in this assignment.

Marking

Each question has a number of *s. Each question will be marked as :

α Essentially a fully correct answer. Base score is 10 marks.

β A good effort, largely correct, but some significant errors. Base score is 6 marks.

γ A weak effort, largely incorrect, but having demonstrated reasonable effort. Base score is 3 marks.

δ A trivial effort or missing answer. Base score is 0 marks.

Each question then scores the Base score multiplied by the number of *s for that question. Hence, a question $[*]$ can obtain a maximum mark of 10, whereas a question $[***]$ can obtain a maximum mark of 30. The scores for each piece of coursework are added up and the total score for all coursework is simply the percentage of the maximum possible score attainable.

In this marking scheme the number of stars is the weight of each question. The score $\alpha, \beta, \gamma, \delta$ evaluates how well each question has been answered. For this assignment, the maximum possible number of marks is 540 and you will be awarded $100 * n/540$ percent for this assignment, where n is the number of marks you get.

Exercise 1

We define the transfer function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

1. [*] Show that for a classification problem with a set of input-outputs $\{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$, $\mathbf{x}^n \in \mathbb{R}^D$, $c^n \in \{0, 1\}$ the (scaled) negative log likelihood is given by[1]

$$E_{lik}(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} \left(\mathbb{I}\left[c^n = 1\right] \log \sigma\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n\right) + \mathbb{I}\left[c^n = 0\right] \log\left(1 - \sigma\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n\right)\right)\right) \tag{2}$$

2. [*] Show that $E_{lik}(\mathbf{w})$ is a convex function of $\mathbf{w}$.

[1] The "cross entropy" loss between distributions p and q is defined as $H(p, q) = -\mathbb{E}\left[\log p\right]_q$. The likelihood loss is therefore $H(p^n, c^n)$ where p^n is in state 1 with probability $\sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n)$.

3. [*] Show that

$$E_{sq}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(c^n - \sigma\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n\right)\right)^2 \tag{3}$$

is not a convex function of $\mathbf{w}$.

4. [*] Explain why

$$\sigma(x) = \begin{cases} \frac{e^x}{1+e^x} & x \leq 0 \\ \frac{1}{1+e^{-x}} & x > 0 \end{cases} \tag{4}$$

avoids numerical overflow in calculating $\sigma(x)$.

5. [*] Explain why

$$\log\sigma(x) = \begin{cases} x - \log\left(1+e^x\right) & x \leq 0 \\ -\log\left(1+e^{-x}\right) & x > 0 \end{cases} \tag{5}$$

avoids numerical overflow in calculating $\log\sigma(x)$.

6. [**] Similar to code presented in appendix(A), plot 1-dimensional slices of the error surface for the logloss and the square loss, demonstrating that the likloss is convex, but that the squareloss is not convex[2]

7. [**] Similarly, define D-dimensional vectors $\mathbf{v}_0$, $\mathbf{v}_1$ and $\mathbf{v}_2$ and make a plot of the two dimensional surface

$$F(z_1, z_2) = E(\mathbf{v}_0 + z_1\mathbf{v}_1 + z_2\mathbf{v}_2) \tag{6}$$

for the likelihood loss and square loss separately, demonstrating that the likelihood loss is convex in z_1, z_2, whereas the square loss is not convex in z_1, z_2.

Exercise 2

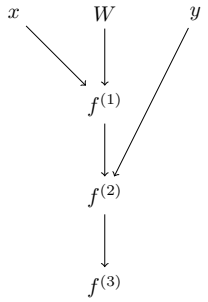
1. [**] Consider the function

$$E = \sum_i \left(y_i - \sum_j W_{ij}x_j\right)^2 \tag{7}$$

and that we wish to calculate

$$\frac{\partial E}{\partial W_{ab}} \tag{8}$$

We can write a computation graph for the function as



where

$$f_i^{(1)} = \sum_j W_{ij}x_j \tag{9}$$

$$f_i^{(2)} = y_i - f_i^{(1)} \tag{10}$$

$$f^{(3)} = \sum_i \left(f_i^{(2)}\right)^2 \tag{11}$$

[2]Note, of course, that showing that a slice of a function is convex, does not guarantee that the function is convex – it would have to be convex for all possible slices. On the other hand, if any slice is non-convex, then the function is non-convex.

Show that, for reverse mode AutoDiff, the edges of the graph are

$$\frac{\partial f_i^{(1)}}{\partial W_{ab}} = \delta_{ia} x_b \tag{12}$$

$$\frac{\partial f_i^{(2)}}{\partial f_j^{(1)}} = -\delta_{ij} \tag{13}$$

$$\frac{\partial f^{(3)}}{\partial f_i^{(2)}} = 2 f_i^{(2)} \tag{14}$$

Using the reverse propagation schedule :

$$t^{(3)} = 1 \tag{15}$$

$$t_i^{(2)} = \frac{\partial f^{(3)}}{\partial f_i^{(2)}} t^{(3)} \tag{16}$$

$$t_i^{(1)} = \sum_j \frac{\partial f_j^{(2)}}{\partial f_i^{(1)}} t_j^{(2)} \tag{17}$$

$$t_{ab}^{W} = \sum_i \frac{\partial f_i^{(1)}}{\partial W_{ab}} t_i^{(1)} \tag{18}$$

$$\tag{19}$$

show that

$$\frac{\partial E}{\partial W_{ab}} = t_{ab}^W = -2 f_a^{(2)} x_b = -2(y_a - \sum_i W_{ai} x_i) x_b \tag{20}$$

2. [**] Consider the function

$$E = \sum_i \left(y_i - \phi \left(\sum_j W_{ij} x_j \right) \right)^2 \tag{21}$$

where $\phi(x)$ is a transfer function of any form. Write down a computation graph for this function (it should have an additional node to the graph above). Write down the reverse schedule of messages $t^{(4)}, t^{(3)}, t^{(2)}, t^{(1)}, t^W$ (similarly to the above) to calculate

$$\frac{\partial E}{\partial W_{ab}} \tag{22}$$

and verify that this reverse mode schedule correctly computes this gradient.

3. [**] Consider the square loss for a single hidden layer neural network (for a single (x, y) training point)

$$E = \sum_i \left(y_i - \phi_2 \left(\sum_k W_{ik}^{(2)} \phi_1 \left(\sum_j W_{kj}^{(1)} x_j \right) \right) \right)^2 \tag{23}$$

where $\phi_i(x)$ are transfer functions of any form. Draw a computation graph for this function. Write down the reverse schedule of messages to calculate

$$\frac{\partial E}{\partial W_{ab}^{(1)}}, \qquad \frac{\partial E}{\partial W_{ab}^{(2)}} \tag{24}$$

and verify that this reverse mode schedule correctly computes these gradients.

4. [*] Explain how to extend this scheme to calculate the gradient for a collection of input-output training pairs (x^n, y^n):

$$E = \sum_n \sum_i \left(y_i^n - \phi_2 \left(\sum_k W_{ik}^{(2)} \phi_1 \left(\sum_j W_{kj}^{(1)} x_j^n \right) \right) \right)^2 \tag{25}$$

5. [**] Consider a constrained version of the question 3 above in which the matrices $W^{(2)} = W^{(1)} = W$.

$$E = \sum_i \left(y_i - \phi_2 \left(\sum_k W_{ik} \phi_1 \left(\sum_j W_{kj} x_j \right) \right) \right)^2 \tag{26}$$

where $\phi_i(x)$ are transfer functions of any form. Draw a computation graph for this function. Write down the reverse schedule of messages to calculate

$$\frac{\partial E}{\partial W_{ab}} \tag{27}$$

and verify that this reverse mode schedule correctly computes these gradients.

Exercise 3

The MNIST dataset is available from http://yann.lecun.com/exdb/mnist/. Download both the train and test sets. Make a new train and test set so that each pixel is rescaled

$$x_i \to x_i/255 \tag{28}$$

which will make all pixel values now lie between 0 and 1. We wish to make an autoencoder using a single hidden layer network with H hidden units of the form

$$f_i(x) = \sigma \left(b_i^{(2)} + \sum_{k=1}^{H} W_{ik}^{(2)} \phi \left(b_k^{(1)} + \sum_{j=1}^{784} W_{kj}^{(1)} x_j \right) \right) \tag{29}$$

where σ is the logistic sigmoid function, equation(1). The Autoencoder error is then

$$E(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}) = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} \sum_{i=1}^{784} L(x_i^n, f_i(x^n)) \tag{30}$$

for a loss function L.

1. [**] Show that, for $0 < x < 1$ and $0 < y < 1$,

$$2(x-y)^2 \leq x \log x + (1-x)\log(1-x) - x \log y - (1-x)\log(1-y) \tag{31}$$

2. [*] Draw a computation tree for the above Autoencoder.

3. [**] Write down explicitly the reverse messages required to calculate the reverse mode AutoDiff for the gradients

$$\frac{\partial E}{\partial W^{(1)}}, \frac{\partial E}{\partial b^{(1)}}, \frac{\partial E}{\partial W^{(2)}}, \frac{\partial E}{\partial b^{(2)}} \tag{32}$$

4. [*****] Consider the square loss

$$L_{sq}(x, y) = \frac{1}{2}(x-y)^2 \tag{33}$$

Using your explicitly coded reverse mode AutoDiff schedule above, use the ADAM optimisation method (see lecture slides) to learn the parameters $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$ of this Autoencoder for $H = 30$ hidden units. You must explicitly write your own ADAM gradient descent routine. You may choose any transfer function $\phi(x)$ you wish. Plot a graph of the training error versus iteration of ADAM. For 100 randomly chosen images from the test set, plot the original image and its reconstruction using the trained Autoencoder.

5. [*****] Repeat the above for the loss[3]

$$L_{lik}(x, y) = -x \log y - (1-x)\log(1-y) \tag{34}$$

plotting the progression of the ADAM optimiser and the reconstruction for the same 100 test images as for the above question. Comment on any differences in training you noticed between using the square loss L_{sq} in the previous question and L_{lik} in this question.

[3] This can be thought of as the cross entropy loss (expected negative log likelihood) between two Bernoulli distributions whose means are given by x and y respectively.

6. [**] Show that, for the likelihood loss L_{lik}, the Autoencoder error $E_{lik}(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$ is a convex function of $W^{(2)}, b^{(2)}$. Show that, in general, for the square-loss L_{sq}, the Autoencoder error $E_{sq}(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$ is not a convex function of $W^{(2)}, b^{(2)}$.

7. [**] Plot 2-dimensional slices of the Autoencoder error with square loss

$$E_{sq}(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}) \tag{35}$$

Randomly initialise $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$ close to the optimal values found from your ADAM training. Then plot 2D slices of the error $F(z_1, z_2)$ by varying z_1 and z_2, using the following three situations

$$F_{11}(z_1, z_2) = E_{sq}(W^{(1)} + z_1 U^{(1)} + z_2 U^{(2)}, b^{(1)}, W^{(2)}, b^{(2)}) \tag{36}$$

$$F_{12}(z_1, z_2) = E_{sq}(W^{(1)} + z_1 U^{(1)}, b^{(1)}, W^{(2)} + z_2 U^{(2)}, b^{(2)}) \tag{37}$$

$$F_{22}(z_1, z_2) = E_{sq}(W^{(1)}, b^{(1)}, W^{(2)} + z_1 U^{(1)} + z_2 U^{(2)}, b^{(2)}) \tag{38}$$

where $U^{(1)}$ and $U^{(2)}$ are randomly chosen matrices of appropriate size in each case.

8. [**] Repeat the above plotting of slices F, but for the likelihood loss

$$E_{lik}(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}) \tag{39}$$

and comment on your results, relating any findings to convexity.

Exercise 4

1. [**] By using the bound

$$\log x \leq x - 1, \qquad x > 0 \tag{40}$$

show that

$$\mathrm{KL}(q||p) \equiv \int q(x) \log \frac{q(x)}{p(x)} dx \geq 0 \tag{41}$$

2. [*] For a generative model

$$p(x) = \int p(x|z, \theta) p(z) dz \tag{42}$$

show that

$$\log p(x) \geq -\int q(z) \log q(z) dz + \int q(z) \log \left[p(x|z, \theta) p(z) \right] dz \tag{43}$$

3. [**] For a Gaussian distribution

$$q(z) = \frac{1}{\sqrt{\det(2\pi\Sigma)}} \exp\left(-\frac{1}{2} (z - \mu)^{\mathsf{T}} \Sigma^{-1} (z - \mu) \right) \tag{44}$$

show that the entropy is given by

$$-\int q(z) \log q(z) dz = \frac{1}{2} \log \det(2\pi e \Sigma) \tag{45}$$

4. [**] Hence, for X-dimensional variable x,

$$p(x|z, \theta) = \frac{1}{(2\pi\sigma_x^2)^{X/2}} \exp\left(-\frac{1}{2\sigma_x^2} (x - \mu_x(z, \theta))^2 \right) \tag{46}$$

and Z-dimensional variable z,

$$q(z|x, \phi) = \frac{1}{(2\pi\sigma_z^2)^{Z/2}} \exp\left(-\frac{1}{2\sigma_z^2} (z - \mu_z(x, \phi))^2 \right) \tag{47}$$

and zero mean unit covariance Gaussian latent $p(z) = \mathcal{N}(z|0, I)$, show that

$$\log p(x) \geq \frac{Z}{2} \log \sigma_z^2 - \frac{X}{2} \log \sigma_x^2 - \frac{1}{2\sigma_x^2} \int q(z|x, \phi) \left(x - \mu_x(z, \theta) \right)^2 dz - \frac{1}{2} \left(Z\sigma_z^2 + \mu_z^2(x, \phi) \right) + \text{const.} \tag{48}$$

5. [*] Show that

$$\int f(x) \frac{1}{\sqrt{\det(2\pi\Sigma)}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)} dx = \int f(\mu + \Sigma^{1/2}\epsilon) \frac{1}{(2\pi)^{X/2}} e^{-\frac{1}{2}\epsilon^2} d\epsilon \tag{49}$$

6. [*] Show that

$$\log p(x) \geq \frac{Z}{2} \log \sigma_z^2 - \frac{X}{2} \log \sigma_x^2 - \frac{1}{2\sigma_x^2} \mathbb{E}\left[(x - \mu_x(\mu_z(x,\phi) + \sigma_z\epsilon, \theta))^2\right]_\epsilon - \frac{1}{2}\left(Z\sigma_z^2 + \mu_z^2(x,\phi)\right) + \text{const.} \tag{50}$$

where ϵ is a zero mean unit covariance Gaussian distributed random variable.

7. [*] For a set of datapoints $x^1, \ldots, x^N$, explain why (for fixed σ_x^2, σ_z^2) a suitable objective to minimise to fit a generative model to this data is

$$E(\theta, \phi) = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbb{E}\left[(x^n - \mu_x(\mu_z(x^n,\phi) + \sigma_z\epsilon, \theta))^2\right]_\epsilon + \sigma_x^2 \mu_z^2(x^n, \phi) \right) \tag{51}$$

8. [*****] Fit a generative model to the MNIST training using the "variational autoencoder" method (VAE) above using a sampling approximation, by drawing one or more samples ϵ (per datapoint) to approximate the expectation with respect to ϵ. You may use a generative network μ_x and inference network μ_z of your own choice. Plot the progress of the training objective and plot sample images x drawn from your trained network, to demonstrate the quality of the trained model. You may use any software framework you wish to implement this. You may also wish to learn the variances σ_x^2 and σ_z^2, or consider a more general diagonal covariance to improve the results.

A Julia 1.0 : Plotting a 1D slice of the error

```
using PyPlot

function sigmoid(x)  # avoid overflow in calculating sigmoid
y=zeros(size(x))
for i=1:length(y)
if x[i]<0
y[i]=exp(x[i])/(1+exp(x[i]))
else
y[i]=1/(1+exp(-x[i]))
end
end
return y
end

function logsigmoid(x) # avoid overflow in calculating log(sigmoid)
y=zeros(size(x))
for i=1:length(y)
if x[i]<0
y[i]=x[i]-log(1+exp(x[i]))
else
y[i]=-log(1+exp(-x[i]))
end
end
return y
end

NNpred(weights,x) = sigmoid(weights'*x)

SquareLoss(c,pred) = begin; vc=vec(c); vp=vec(pred); sum((vc .- vp).^2)/length(c); end

LikLoss(c,pred) = begin; vc=vec(c); vp=vec(pred); -sum( vc.*log.(vp)  .+ (1 .- vc).*log.(1 .- vp)
)/length(c); end

function LikLoss(c,weights,x) # avoid overflow in calculating likloss
logpred=vec(logsigmoid(weights'*x))
log1mpred=vec(logsigmoid(-weights'*x))
return -sum( c.*logpred  .+ (1 .- c).*log1mpred  )/length(c)
end

N = 200 # number of training points
D = 10  # dimension of input
x = randn(D,N) # training inputs

# make some training data:
w_true=randn(D); c=zeros(Bool,N)
for n=1:N
if NNpred(w_true,x[:,n]) .> 0.5
c[n]=1
end
end

# plot a 1D slice of the loss functions :
vec0=10*randn(D) # a random point
vec1=10*randn(D) # a random direction
I=100
Loss1=zeros(I); Loss2=zeros(I)
for i=1:I
lambda=i/I
Loss1[i] = SquareLoss(c,NNpred(vec0+lambda*vec1,x))
Loss2[i] = LikLoss(c,vec0+lambda*vec1,x)
end
plot(Loss1); title("Square Loss"); figure(); plot(Loss2); title("Lik Loss")
```