Due Date : February 22nd, 24:00

Instructions

- *For all questions, show your work!*
- *Use a document preparation system such as LaTeX.*
- *Submit your answers electronically via the course gradescope*
- *TA for this assignment are :* ***Andjela Mladenovic*** *(IFT6135B) and* ***Ghait Boukachab*** *(IFT6135A).*

1. **Selection of Activation Function (10 pts)** We will compare two different activation functions in the following question. Recall the definition of $\sigma(x) = \frac{1}{1+e^{-x}}$ and $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
   (a) (2 pts) Find the derivative of the sigmoid function $\sigma'(x)$ and express it in terms of the sigmoid function $\sigma(x)$.
   (b) (2 pts) Find the derivative of the $\tanh'(x)$ function and express it in terms of the $\tanh(x)$ function.
   (c) (2 pts) Upper bound the value of $\sigma'(x)$ with a constant (you can use AM–GM inequality).
   (d) (2 pts) Upper bound the value of $\tanh'(x)$ with a constant (you can use GM-HM inequality or the property that the square of real number is always non-negative).
   (e) (2 pts) Compare the two upper bounds and explain what impact would this difference have on optimization.

Answer 1 *a. we know the sigmoid function is defined as :*

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

So the derivative of sigmoid with respect to x using chain rule is :

$$\begin{aligned}\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\frac{1}{1+e^{-x}} = \frac{d}{dx}(1+e^{-x})^{-1} \\ &= -(1+e^{-x})^{-2}\frac{d}{dx}(1+e^{-x}) \\ &= -(1+e^{-x})^{-2}(-e^{-x}) \\ &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}} \\ &= \frac{1}{1+e^{-x}}(1 - \frac{1}{1+e^{-x}}) = \sigma(x)(1-\sigma(x))\end{aligned}$$

b.
we know the tanh function is defined as :

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

So the derivative of tanh with respect to x using quotient rule is :

$$\begin{aligned}
\tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\
\frac{d}{dx}\tanh(x) &= \frac{d}{dx}\frac{e^x - e^{-x}}{e^x + e^{-x}} \\
&= \frac{(e^x + e^{-x})\frac{d}{dx}(e^x - e^{-x}) - (e^x - e^{-x})\frac{d}{dx}(e^x + e^{-x})}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\
&= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\
&= 1 - (\frac{e^x - e^{-x}}{e^x + e^{-x}})^2 \\
&= 1 - \tanh(x)^2
\end{aligned}$$

c.
Using the AM-GM inequality and using the derivative we derived in the first part if we set x1 =$\sigma(x)$ *and x2=* $1-\sigma(x)$*. according to the formula below we have :*

$$\frac{x_1 + x_2}{2} \geq \sqrt{x_1 x_2}$$

so substituting x1 and x2 by the values above we have :

$$\begin{aligned}
\frac{\sigma(x) + 1 - \sigma(x)}{2} &\geq \sqrt{\sigma(x)(1-\sigma(x))} \\
&= \frac{1}{2} \geq \sqrt{\frac{d}{dx}\sigma(x)} \\
&= \frac{d}{dx}\sigma(x) \leq \frac{1}{4}
\end{aligned}$$

which means that $\sigma'(x)$ *has an upper bound value of* $\frac{1}{4}$

d.
Using the property that the square of real number is always non-negative and using the derivative we derived in the second part we have :

$$\begin{aligned}
\tanh(x)^2 &\geq 0 \\
-\tanh(x)^2 &\leq 0 \\
1 - \tanh(x)^2 &\leq 1 \\
= \frac{d}{dx}\tanh(x) &\leq 1
\end{aligned}$$

which means that $\tanh'(x)$ *has an upper bound value of 1.*

e.
Both functions belong to the S-like functions that suppress the input value to a bounded range. This helps the network to keep its weights bounded and prevents the exploding gradient problem where the value of the gradients becomes very large.

The range of the tanh function includes negative values. it maps the input value to a value between -1 and 1. whereas the range of the sigmoid function is limited to positive values between 0 and 1. This can be important for certain types of neural networks, such as those used for classification, where the output needs to be a value between 0 and 1. Another important difference between the two functions is their curvature. The sigmoid function has a maximum slope of 0.25 at the origin, which means that the gradient can become very small for large or small inputs. The tanh function, on the other hand, has a maximum slope of 1 at the origin (steeper gradient at the origin), which means that using the tanh activation function results in higher values of gradient during training and higher updates in the weights of the network. This can be important for deeper neural networks, where the vanishing gradient problem can be a significant issue. So, if we want strong gradients and big learning steps, we should use the tanh activation function. However, the sigmoid function is better for certain types of problems where the output needs to be constrained to a specific range

in addition the output of tanh is symmetric around zero leading to faster convergence.

Despite their benefits, both functions present the so-called vanishing gradient problem. In neural networks, the error is backpropagated through the hidden layers of the network and updates the weights. In case we have a very deep neural network and bounded activation functions, the amount of error decreases dramatically after it is backpropagated through each hidden layer. So, at the early layers, the error is almost zero, and the weights of these layers are not updated properly. The ReLU activation function can fix the vanishing gradient problem.

Useful inequalities:

Inequality of Arithmetic and Geometric Means (AM-GM)

$$\frac{x_1 + x_2 + \dots x_n}{n} \geq \sqrt[n]{x_1 x_2 \dots x_n} \tag{1}$$

Inequality of Geometric and Harmonic Means (GM-HM)

$$\sqrt[n]{x_1 x_2 \dots x_n} \geq \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \dots \frac{1}{x_n}} \tag{2}$$

The above inequalities hold for any real positive numbers $x_1, x_2, \dots x_n$ with equality if and only if $x_1 = x_2 = \dots = x_n$.

2. **Cross Entropy Properties (12 pts)**

Cross-entropy loss function (a popular loss function) is given by:

$$\mathrm{CE}(p, x) = -x \log(p) - (1 - x) \log(1 - p)$$

.

Please refer to this loss for (a) and (b) parts.

(a) (2 pts) **Cross Entropy and Maximum Likelihood** For this derivation, we assume that x is binary, i.e. $x \in \{0, 1\}$. Derive the cross-entropy loss function using the maximum likelihood principle for $x \in \{0, 1\}$.

(b) (2 pts) **Cross Entropy and KL divergence** Suggest a probabilistic interpretation of the cross-entropy loss function when $x \in (0, 1)$. (Hint: KL divergence between two distributions)

(c) (4 pts) **Discrete distribution - Maximum Entropy** Let X be a random variable which takes n values with probabilities $p_1, p_2, \ldots, p_n$ with $p_i > 0, \forall i$. What is the distribution that maximizes entropy $H(X) = -\sum_{i=1}^{n} p_i \log p_i$? Derive the upper bound for the entropy $H(X)$ expressed as a function of n. (Hint : use Jensen Inequality)

(d) (4 pts) **Continuous distribution (known mean μ and variance σ^2) - Maximum Entropy** Given mean μ and variance σ^2, what is the continuous distribution that maximizes differential entropy $h(X) = -\int_x f(x) \log f(x) dx$? Prove it.

Answer 2 *a.*
Binary Classification presents a unique problem where :
1.each example(x', x) belongs to one of two complementary classes, 2.each example is independent of each other(i.e result of one example does not affect the outcome of the other example) and, 3.all the examples are generated are from the same underlying distribution/process.
Further, we only need to predict for the positive class i.e P(x=1 | x') = p because the probability for the negative class can be derived from it i.e P(x=0 | x') =1-P(x=1 | x') = 1-p.

$$P(x|x') = \begin{cases} p & if\ x = 1 \\ 1 - p & if\ x = 0 \end{cases}$$

which can be written in more compact form :

$$P(x|x') = p^x (1 - p)^{(1-x)} for x \in \{0, 1\}$$

.

Using a concept called maximum likelihood estimation(MLE) we can extend the Bernoulli Distribution and come up with the Binary Cross Entropy Cost function. Recall that for a single data point we are maximizing a Bernoulli trial, for multiple data points we will

maximize the product of multiple Bernoulli trials.
Applying this concept to multiple independent Bernoulli trials (a distribution with multiple independent Bernoulli trials is called a Binomial distribution) and maximizing the probability for each of the "m" examples in a dataset/batch we get : first we write the likelihood function :

$$L(p|(x'^1, x^1), ((x'^2, x^2)), ..., (x'^m, x^m)) = \prod_{i=1}^{m} P(x^i|p) = \prod_{i=1}^{n} (p)^{x^i} ((1-p))^{(1-x^i)}$$

The likelihood function in its current form is prone to numerical overflow because of multiple products. So we will instead take the natural log of the likelihood function.

$$\log L(p|(x'^1, x^1), ((x'^2, x^2)), ..., (x'^m, x^m)) = \log(\prod_{i=1}^{n} (p)^{x} ((1-p))^{(1-x^i)})$$

$$= \sum_{i=1}^{n} (x^i \log p + (1-x^i) \log(1-p))$$

Recall that maximizing a function is the same as minimizing the negative of that function.

$$-\log L(p|(x'^1, x^1), ((x'^2, x^2)), ..., (x'^m, x^m)) = \sum_{i=1}^{n} (-x^i \log p - (1-x^i) \log(1-p))$$

MLE estimate is the set of model parameters that minimize

$$-\log L(p|(x'^1, x^1), ((x'^2, x^2)), ..., (x'^m, x^m)) = \sum_{i=1}^{n} (-x^i \log p - (1-x^i) \log(1-p))$$

Turns out it's the Cross-Entropy(CE) Cost function that we've been using.

b.

The cross-entropy and Kullback-Leibler (KL) divergence are related concepts in information theory and are often used in machine learning.
Let $p(x)$ and $q(x)$ be two probability distributions over the same discrete random variable x (they could be the probability distribution of x for the true and predicted distribution). The cross-entropy between p and q is defined as :

$$H(p,q) = -\sum_x p(x)\log q(x)$$

The KL divergence from p to q is defined as :

$$D_{KL}(p||q) = \sum_x p(x)\log\frac{p(x)}{q(x)}$$

We can see that the cross-entropy and KL divergence are related by the following equation :

$$H(p,q) = H(p) + D_{KL}(p||q)$$

where $H(p)$ is the entropy of p, given by $H(p) = -\sum_x p(x)\log p(x)$.
The above equation can be derived by using the property of logarithms and the definition of KL divergence :

$$\begin{aligned} H(p,q) &= -\sum_x p(x)\log q(x) \\ &= -\sum_x p(x)\left(\log p(x) - \log\frac{p(x)}{q(x)}\right) \\ &= -\sum_x p(x)\log p(x) + \sum_x p(x)\log\frac{p(x)}{q(x)} \\ &= H(p) + D_{KL}(p||q) \end{aligned}$$

This equation shows that the cross-entropy between p and q is equal to the entropy of p plus the KL divergence from p to q. In machine learning, the cross-entropy is often used as a cost function to train a model to predict a probability distribution that is as close as possible to the true distribution. The KL divergence is then used to measure the difference between the predicted distribution and the true distribution.
c.

To find the distribution that achieves this upper bound, we can use Lagrange multipliers to maximize the function

$$g(p_1, p_2, ..., p_n) = -\sum_{i=1}^{n} p_i log(pi)$$

subject to the constraint

$$\sum_{i=1}^{n} p_i = 1$$

and

$$p_i \geq 0 for all i$$

The Lagrangian is :

$$L(p(X), \lambda_0) = \sum_{x \in X} -p(x) \log p(x) + \lambda_0 * (\sum_{x \in X} p(x) - 1)$$

taking derivative with respect to one of these p(x) :

$$\frac{\partial L}{\partial p(x)} = 0 = -\log p(x) - 1 + \lambda_0$$

$$\frac{\partial L}{\partial \lambda_0} = 0 = \sum_{x \in X} p(x) - 1$$

note that i map the corresponding probabilities of each n *values of* X *which are* $p_1, p_2, ..., p_3$ *to probability of occurring the corresponding event* $x \in X$ *which is p(X=x).*
from the last two equations we have :

$$p(x) = \exp(-1 + \lambda_0)$$

while $\sum_{x \in X} p(x)$ *=1*
so :

$$\sum_{x \in X} \exp(-1 + \lambda_0) = 1$$

$$n * \exp(-1 + \lambda_0) = 1$$

$$\log n + \lambda_0 - 1 = 0$$

$$\lambda_0 = 1 - \log n$$

now we substitute λ_0 *into* $p(x) = \exp(-1 + \lambda_0)$ *which yields :*

$$p(x) = \exp(-1 + 1 - \log n) = \exp(-\log n) = \exp(\log \frac{1}{n}) = \frac{1}{n}$$

finally the distribution that takes n *values with probabilities* $p_1, p_2, ..., p_n$ *and maximize entropy* $H(X) = \sum_{x \in X} -p(x) \log_2 p(x)$ *is :* $p(x) = \frac{1}{n}$

To derive the upper bound for the entropy H(X), we can use Jensen's inequality, which *states that for a convex function f and a random variable X, we have :*

$$f(E[X]) \leq E[f(X)]$$

we need to show that the entropy H(X) is a convex function in order to apply jensen's inequality.

the entropy H(X) is defined over a discrete probability distribution, which means that its second derivative may not exist everywhere. To show that H(X) is a convex function, we need to show that its discrete version satisfies the definition of convexity.

to show that the entropy of X, H(X), is convex, we need to show that for any two probability distributions p and q over X, and for any $\lambda \in$ [0,1], the following inequality holds :

$$H(\lambda * p + (1-\lambda) * q) \leq \lambda * H(p) + (1-\lambda)H(q)$$

where $\lambda * p + (1-\lambda) * q)$ is the convex combination of the probability distributions p and q.

Let p and q be two probability distributions over the same set of n outcomes, and let $0 \leq \lambda \leq 1$. Then the convex combination of p and q is defined as :

$$r = \lambda p + (1-\lambda)q$$

The entropy of r is : $H(r) = -\sum_{i=1}^{n} r_i log r_i$

Using the fact that log is a concave function, we can apply Jensen's inequality to get :

$$H(r) = -\sum_{i=1}^{n} r_i log r_i \leq -\lambda \sum_{i=1}^{n} p_i log_i - (1-\lambda) \sum_{i=1}^{n} q i_i log q_i = \lambda H(p) + (1-\lambda)H(q)$$

This shows that the entropy of a convex combination of probability distributions is less than or equal to the convex combination of their entropies. Therefore, the entropy H(X) is a convex function.

now we want to derive an upper bound on the entropy of a random variable X that takes n values with probabilities $p_1, p_2, \ldots, p_n$ using Jensen's inequality.
Recall that the entropy of X is given by :

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i$$

Let Y be a random variable such that $Y = \frac{1}{n}\sum_{i=1}^{n} p_i$. We can interpret Y as the expected value of p_i when all probabilities are assumed to be equal. That is, if we replace all p_i's with Y, we get :

$$H(X) = -\sum_{i=1}^{n} Y \log Y$$

Now, let's apply Jensen's inequality to the function $f(Y) = -Y \log Y$. Since f is a concave function, Jensen's inequality tells us that :

$$H(X) = -\sum_{i=1}^{n} Y \log Y \leq -\log\left(\frac{1}{n}\sum_{i=1}^{n} Y\right)$$

Simplifying the right-hand side, we get :

$$H(X) \leq -\log\left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{n}\sum_{j=1}^{n} p_j\right)$$

which is equivalent to :

$$H(X) \leq -\log\left(\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} p_j\right)$$

Since $\sum_{j=1}^{n} p_j = 1$, we have :

$$H(X) \leq -\log\left(\frac{1}{n^2}\sum_{i=1}^{n} 1\right) = -\log\left(\frac{n}{n^2}\right) = \log n$$

Therefore, we have shown that :

$$H(X) \leq \log n$$

This inequality tells us that the entropy of a random variable taking n values with probabilities $p_1, p_2, \ldots, p_n$ is at most $\log n$. This upper bound is achieved when all probabilities are equal, i.e., $p_1 = p_2 = \cdots = p_n = \frac{1}{n}$.
d.

$$L(f(x), \lambda_0, \lambda_1) = \int_x f(x) * \log f(x) dx + \lambda_0(\int_x f(x) dx - 1) + \lambda_1(\int_x (x-\mu)^2 f(x) - \sigma^2)$$

$$\frac{\partial L}{\partial f(x)} = 0 = -\log f(x) - 1 + \lambda_0 + \lambda_1 (x-\mu)^2$$

$$\frac{\partial L}{\partial \lambda_0} = 0 = \int_x f(x)dx - 1$$

$$\frac{\partial L}{\partial \lambda_1} = 0 = \int_x (x-\mu)^2 f(x) - \sigma^2$$

from the first derivative we have :

$$f(x) = \exp(\lambda_1(x-\mu)^2 + \lambda_0 - 1) = \exp(\lambda_1(x-\mu)^2) * \exp(\lambda_0 - 1)$$

$$C = \exp(\lambda_0 - 1) -> f(x) = C * \exp(\lambda_1(x-\mu)^2)$$

from the second derivative we have :

$$\int_{-\infty}^{\infty} f(x)dx = 1$$

$$\int_{-\infty}^{\infty} C * \exp(\lambda_1(x-\mu)^2)dx = 1$$

$$\int_{-\infty}^{\infty} \exp(\lambda_1(x-\mu)^2)dx = \frac{1}{c}$$

we set $\lambda_1 = -\beta$

$$\frac{\partial L}{\partial \lambda_0} = 0$$

$$\int_{-\infty}^{\infty} \exp(-\beta(x-\mu)^2)dx = \frac{1}{c}$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \exp(-\beta((x-\mu)^2 + (y-\mu)^2))dxdy = \frac{1}{c^2}$$

$=$

$$\int_{\theta=0}^{2\pi}\int_{r=0}^{\infty} \exp(-\beta((x-\mu)^2 + (y-\mu)^2))drd\theta = \frac{1}{c^2}$$

$$[r = \sqrt{(x-\mu)^2 + (y-\mu)^2)}]$$

=

$$\int_{\theta=0}^{2\pi}\int_{r=0}^{\infty}\exp(-\beta(r^2)rdrd\theta = \frac{1}{c^2}$$

=

$$2\pi * (-\frac{\exp(-\beta(r^2)}{2\beta})|_0^{\infty} = \frac{1}{c^2}$$

$$\beta = c^2\pi$$

$$\frac{\partial L}{\partial \lambda_1} = 0$$

$$\int_{-\infty}^{\infty} C*(x-\mu)^2*\exp(-\beta(x-\mu)^2)dx = \sigma^2$$

$$y = x - \mu dx = dy$$

$$\int_{-\infty}^{\infty} C*(y)^2*\exp(-\beta(y)^2)dy = \sigma^2$$

$$y = \frac{z}{\sqrt{\beta}} - > y^2 = \frac{z^2}{\beta} - > dy = \frac{dz}{\sqrt{\beta}}$$

$$\int_{-\infty}^{\infty} C*\frac{z^2}{\beta}*\exp(-z^2)\frac{dz}{\sqrt{\beta}} = \sigma^2$$

$$= \frac{C}{\beta*\sqrt{\beta}}\int_{-\infty}^{\infty} z^2*\exp(-z^2)dz = \sigma^2$$

since it is symmetric

$$= 2*\frac{C}{\beta*\sqrt{\beta}}\int_{0}^{\infty} z^2*\exp(-z^2)dz = \sigma^2 h = z^2 - > z = \sqrt{h} - > dz = \frac{dh}{2\sqrt{h}}$$

=

$$\frac{C}{\beta*\sqrt{\beta}}\int_{0}^{\infty}\sqrt{h}*\exp(-h)dh = \sigma^2$$

=

$$\frac{C}{\beta*\sqrt{\beta}}\int_{0}^{\infty} h^{(\frac{3}{2}-1)}*\exp(-h)dh = \sigma^2$$

$$\gamma(x) = \int_{0}^{\infty} h^{x-1}*\exp(-h)dh$$

and we know

$$\gamma(\frac{3}{2}) = \frac{\sqrt{\pi}}{2}$$

so putting all these together we have :

$$\frac{C}{\beta * \sqrt{\beta}} * \frac{\sqrt{\pi}}{2} = \sigma^2$$

and we already have $\beta = c^2\pi$

$$\frac{C}{C^3 * \pi\sqrt{\pi}} * \frac{\sqrt{\pi}}{2} = \sigma^2$$

$$C = \frac{1}{\sigma\sqrt{2\pi}}$$

and

$$\beta = \frac{1}{2\sigma^2}$$

$$p(X) = C * \exp(-\beta(X-\mu)^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2\sigma^2}(X-\mu)^2)$$

3. **Output size and Parameters of Convolution Layers (5 pts)**
Consider a 3 hidden-layer convolutional neural network. Assume the input is a color image of size 128×128 in the RGB representation. The first layer convolves 64 8×8 kernels with the input, using a stride of 2 and zero-padding of 4. The second layer downsamples the output of the first layer with a 2×2 non-overlapping max pooling. The third layer convolves 128 4×4 kernels with a stride of 2 and zero-padding of 2.
   (a) (3 pts) What is the dimensionality of the output of the third layer?
   (b) (2 pts) Not including the biases, how many parameters are needed for the last layer?

Answer 3 *a.*
The output size(height and width of the image) of a convolutional layer can be calculated using the formula

$$\lfloor \frac{input\_shape + 2 * padding - kernel\_size}{stride} \rfloor + 1$$

where input_ shape is the size of the input feature map, padding is the amount of zero-padding applied to the input feature map, kernel_ size is the size of the convolutional kernel, and stride is the stride of the convolutional kernel. so according to this fomrula we can calculate the dimensionality of the output of each layer ://
first layer : input_ shape is 128, padding is 4, kernel_ size is 8 and stride is 2

$$\lfloor \frac{128 + 2 * 4 - 8}{2} \rfloor + 1 = 65$$

*so the dimensionality of the output of the first layer is : 65*65*64 (because the first layer contains 64 kernels)*
second layer : input_shape is 65, padding is 0, kernel_size is 2 and stride is 2

$$\lfloor \frac{65 + 2 * 0 - 2}{2} \rfloor + 1 = 32$$

*so the dimensionality of the output of the second layer is : 32*32*64.*
third layer : input_shape is 32, padding is 2, kernel_size is 4 and stride is 2

$$\lfloor \frac{32 + 2 * 2 - 4}{2} \rfloor + 1 = 17$$

*so the dimensionality of the output of the third layer is : 17*17*128 (because third layer contains 128 kernels)*
b.
*in the last layer we have 128 kernels each of size 4*4 each are connected to the 64 filters we have as the input to this layer. so number of parameters excluding the biases are : 128*4*4*64*

4. **MLP Mixer (16 pts)**

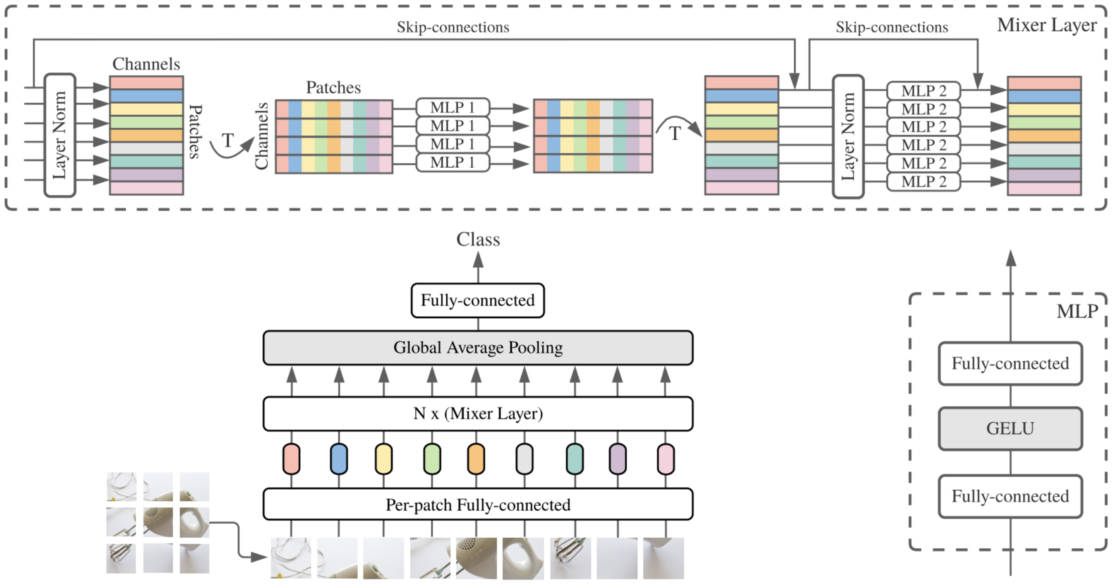

FIGURE 1 – (Borrowed from the MLPMixer paper.) MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include : skip-connections, dropout, and layer norm on the channels.

(a) (2 pts) **MLP Mixer Dimensions** Let's assume that Mixer architecture is being applied to an input image of size 64×64. The Mixer's output is of size 16×128. Determine the patch resolution P, number of patches S, as hidden dimension C(channels).

(b) (2 pts) **MLP Mixer Complexity** Show that the computational complexity of the MLP Mixer is linear in terms of number of input patches.

(c) (6 pts) **Input Transformation - Channel Mixing MLP** Consider the following scenario : The original input image A is of size 9×9. We convert the input image into non-overlapping patches of size 3×3, and then linearly project all patches with the same projection matrix. The result of these operations is a matrix X of size 9×6. Then we apply the *channel-mixing MLP* that acts on rows of X, and is shared across all rows. The result of this operation is matrix U size 9×6. Now consider a modified image A such that $A_{\text{modified}} = PA$, where we define matrix P in the following manner:

$$P = \begin{bmatrix} e_{\pi(1)} \\ e_{\pi(2)} \\ \vdots \\ e_{\pi(9)} \end{bmatrix} \tag{3}$$

Here e_k is k-th basis vector and π represents the permutation of indices from $1 \dots 9$. Find all possible P such that by permuting rows of U_{modified} we can get back matrix U.

(d) (6 pts) Select one of your solutions for P and find $P_{reverse}$ such that $P_{\text{reverse}} U_{\text{modified}} = U$.

Answer 4 *a.*
According to the paper "Mixer takes as input a sequence of S non-overlapping image patches, each one projected to a desired hidden dimension C. This results in a two-dimensional real-valued input table, $X \in R^{S*C}$. If the original input image has resolution (H, W),
and each patch has resolution (P, P), then the number of patches is $S = \frac{H*W}{P^2}$". in this setting we have H=W=64, S=16, C=128 so P is : $P = \sqrt{\frac{H*W}{S}} = \sqrt{\frac{64*64}{16}} = 16$
b.
s= number of input patches c = number of channels according to the equation 1 of the paper. Each MLP block contains two fully-connected layers and a nonlinearity applied independently to each row of its input data tensor. Mixer layers can be written as follows (omitting layer indices) :

$$U_{*,i} = X_{*,i} + W_2 * \sigma(W_1 Layernorm(X)_{*,i})$$

for i=1,...c

$$Y_{j,*} = U_{*,i} + W_4 * \sigma(W_4 Layernorm(U)_{*,i})$$

for j=1,...s
D_s = number of channels = c
D_c = number of input patches = s

which σ is an element-wise nonlinearity (GELU). D_S and D_C are tunable hidden widths in the token-mixing and channel-mixing MLPS, repectively. Note that D_S is selected independently of the number of input patches. Therefore, the computational complexity of the network is linear in the number of input patches. Since D_C is independent of the patch size, the overall complexity is sum of the complexity of layer D_s and D_C which are O(c) *and $O(s)$ respectively. consequently, the overall complexity is linear in terms of the number of patches.*

c.

*our input original image when it is converted to 9 patches of 3*3 patch is:*

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \tag{4}$$

*where A_{ij} is the j's patch in the i's row of the A matrix(i=1,2,3 and j=1,2,3). so overall we have $\frac{9*9}{3^2} = 9$ patches. Then we linearly project all 9 patches with the same projection matrix which results to the matrix X of size 9*6 :*

$$X = \begin{bmatrix} L(A_{11}) \\ L(A_{12}) \\ \vdots \\ L(A_{32}) \\ L(A_{33}) \end{bmatrix} \tag{5}$$

*Then we apply the channel-mixing MLP (denoted as F below)that acts on rows of X, and is shared across all rows. The result of this operation is matrix U of size 9*6 :*

$$U = \begin{bmatrix} F(L(A_{11})) \\ F(L(A_{12})) \\ \vdots \\ F(L(A_{32})) \\ F(L(A_{33})) \end{bmatrix} \tag{6}$$

when we multiply input image in the column representation form with a permutation matrix P as defined in the question, it will permute the rows of the input image with respect to the permutation π, according to the property of the permutation matrix :

$$\begin{bmatrix} e_{\pi(1)} \\ e_{\pi(2)} \\ \vdots \\ e_{\pi(9)} \end{bmatrix} \cdot \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_9 \end{bmatrix} = \begin{bmatrix} A_{\pi(1)} \\ A_{\pi(2)} \\ \vdots \\ A_{\pi(9)} \end{bmatrix} \tag{7}$$

Acording to equation 6 if we change the position of each A_{ij} (which is equivalent to each of our 9 patches) in the matrix A, just the order of rows in U will change because all operations are shared among all rows. So the patches must remain the same but the order of patches can change in $A_{modified}$.

each three patch consist three rows of matrix A.
A_{1j}*, consist first three rows : 1, 2, 3*
A_{2j}*, consist second three rows : 4, 5, 6*
A_{3j}*, consist third three rows : 7, 8, 9*

We group matrix A into three groups, each consisting three consecutive rows of matrix A. $A = \begin{bmatrix} A'_1 \\ A'_2 \\ A'_3 \end{bmatrix}$

where $A'_1 = [A_{11}, A_{12}, A_{13}]$, $A'_2 = [A_{21}, A_{22}, A_{23}]$, *and* $A'_3 = [A_{31}, A_{32}, A_{33}]$.
so if we change the order of A'_i *i=1,2,3 in matrix A, patches won't change but their order will be different. so there are 5 ways to permute these three groups to have matrix different form the original one and at the same time by permuting rows of* $U_{modified}$ *we can reach U.*
these are the 5 different ways :

$$A_{m_1} = \begin{bmatrix} A'_1 \\ A'_3 \\ A'_2 \end{bmatrix} = \begin{bmatrix} A_{1j} \\ A_{3j} \\ A_{2j} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{31} & A_{32} & A_{33} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} \text{ -> } P_1 = \begin{bmatrix} e_{\pi(1)} \\ e_{\pi(2)} \\ e_{\pi(3)} \\ e_{\pi(7)} \\ e_{\pi(8)} \\ e_{\pi(9)} \\ e_{\pi(4)} \\ e_{\pi(5)} \\ e_{\pi(6)} \end{bmatrix}$$

$$A_{m_2} = \begin{bmatrix} A'_2 \\ A'_1 \\ A'_3 \end{bmatrix} = \begin{bmatrix} A_{2j} \\ A_{1j} \\ A_{3j} \end{bmatrix} = \begin{bmatrix} A_{21} & A_{22} & A_{23} \\ A_{11} & A_{12} & A_{13} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \text{ -> } P_2 = \begin{bmatrix} e_{\pi(4)} \\ e_{\pi(5)} \\ e_{\pi(6)} \\ e_{\pi(1)} \\ e_{\pi(2)} \\ e_{\pi(3)} \\ e_{\pi(7)} \\ e_{\pi(8)} \\ e_{\pi(9)} \end{bmatrix}$$

$$A_{m_3} = \begin{bmatrix} A'_2 \\ A'_3 \\ A'_1 \end{bmatrix} = \begin{bmatrix} A_{2j} \\ A_{3j} \\ A_{1j} \end{bmatrix} = \begin{bmatrix} A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{11} & A_{12} & A_{13} \end{bmatrix} \text{ -> } P_3 = \begin{bmatrix} e_{\pi(4)} \\ e_{\pi(5)} \\ e_{\pi(6)} \\ e_{\pi(7)} \\ e_{\pi(8)} \\ e_{\pi(9)} \\ e_{\pi(1)} \\ e_{\pi(2)} \\ e_{\pi(3)} \end{bmatrix}$$

$$A_{m_4} = \begin{bmatrix} A'_3 \\ A'_2 \\ A'_1 \end{bmatrix} = \begin{bmatrix} A_{3j} \\ A_{2j} \\ A_{1j} \end{bmatrix} = \begin{bmatrix} A_{31} & A_{32} & A_{33} \\ A_{21} & A_{22} & A_{23} \\ A_{11} & A_{12} & A_{13} \end{bmatrix} \text{->} P_4 = \begin{bmatrix} e_{\pi(7)} \\ e_{\pi(8)} \\ e_{\pi(9)} \\ e_{\pi(4)} \\ e_{\pi(5)} \\ e_{\pi(6)} \\ e_{\pi(1)} \\ e_{\pi(2)} \\ e_{\pi(3)} \end{bmatrix}$$

$$A_{m_5} = \begin{bmatrix} A'_3 \\ A'_1 \\ A'_2 \end{bmatrix} = \begin{bmatrix} A_{3j} \\ A_{1j} \\ A_{2j} \end{bmatrix} = \begin{bmatrix} A_{31} & A_{32} & A_{33} \\ A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} \text{->} P_5 = \begin{bmatrix} e_{\pi(7)} \\ e_{\pi(8)} \\ e_{\pi(9)} \\ e_{\pi(1)} \\ e_{\pi(2)} \\ e_{\pi(3)} \\ e_{\pi(4)} \\ e_{\pi(5)} \\ e_{\pi(6)} \end{bmatrix}$$

d.
consider A_{m_1} form the previous part

$$A_{m_1} = \begin{bmatrix} A'_1 \\ A'_3 \\ A'_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{31} & A_{32} & A_{33} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} \textit{ after linear projection } X_{m_1} \textit{ is :}$$

$$X_{m_1} = \begin{bmatrix} L(A_{11}) \\ L(A_{12}) \\ L(A_{31}) \\ L(A_{32}) \\ L(A_{33}) \\ L(A_{21}) \\ L(A_{22}) \\ L(A_{23}) \end{bmatrix} \text{->} U_{m_1} = \begin{bmatrix} F(L(A_{11})) \\ F(L(A_{12})) \\ F(L(A_{31})) \\ F(L(A_{32})) \\ F(L(A_{33})) \\ F(L(A_{21})) \\ F(L(A_{22})) \\ F(L(A_{23})) \end{bmatrix}$$

now we need a $P_{reverse}$ such that $P_{reverse} * U_{modified} = U$.

$$P_{reverse} \cdot \begin{bmatrix} F(L(A_{11})) \\ F(L(A_{12})) \\ F(L(A_{31})) \\ F(L(A_{32})) \\ F(L(A_{33})) \\ F(L(A_{21})) \\ F(L(A_{22})) \\ F(L(A_{23})) \end{bmatrix} = U = \begin{bmatrix} F(L(A_{11})) \\ F(L(A_{12})) \\ F(L(A_{13})) \\ F(L(A_{21})) \\ F(L(A_{22})) \\ F(L(A_{23})) \\ F(L(A_{31})) \\ F(L(A_{32})) \\ F(L(A_{33})) \end{bmatrix} \textit{ it can be seen that } P_{reverse} = P_1 = \begin{bmatrix} e_{\pi(1)} \\ e_{\pi(2)} \\ e_{\pi(3)} \\ e_{\pi(7)} \\ e_{\pi(8)} \\ e_{\pi(9)} \\ e_{\pi(4)} \\ e_{\pi(5)} \\ e_{\pi(6)} \end{bmatrix}$$

5. **Gradient Descent Convergence (12 pts)**

(a) (6 pts) **Convex Function Convergence** Consider the following function:

$$f(x) = \begin{cases} \frac{3}{4}(1-x)^2 - 2(1-x) & \text{if } x > 1 \\ \frac{3}{4}(1+x)^2 - 2(1+x) & \text{if } x < -1 \\ x^2 - 1 & \text{otherwise} \end{cases} \tag{8}$$

Show that f is a convex function. Find its unique minimizer and its gradient. Consider the following algorithm : $x_t = x_{t-1} - \eta f'(x_{t-1})$ where $\eta = 1$. Will this algorithm converge to a stationary point if it starts at point x_0, where $x_0 > 1$? Why or why not?

(b) (6 pts) **Prove Convergence of Gradient Descent to Stationary Point in Non-Convex case** Suppose we are trying to minimize the function $F(w)$ that is L-smooth. Let F_* be the minimal function value (i.e. the value at the global minima). Using $\eta = \frac{1}{L}$, prove that gradient descent will "almost" converge to a stationary point in a bounded (and polynomial) number of steps. Precisely,

$$\min_{k<K} \|\nabla F(w^{(k)})\|^2 \leq \frac{2L}{K}(F(w^{(0)}) - F_*) \tag{9}$$

Hints:

i. L-smoothness implies that:

$$F(w^{(k+1)}) \leq F(w^{(k)}) - \eta\|\nabla F(w^{(k)})\|^2 + \frac{1}{2}\eta^2 L\|\nabla F(w^{(k)})\|^2 \tag{10}$$

Combine this with $\eta = \frac{1}{L}$

ii. Use the fact that the minimum of a sequence of elements is less than the average of the sequence.

Answer 5 *a : A function is convex if and only if its second derivative is non-negative over its entire domain. In other words, a function is convex if its graph always curves upward, or equivalently, if every chord connecting two points on the graph lies above the graph. So we find the second derivative of this function :*

$$\frac{d}{dx}f(x) = \begin{cases} \frac{3}{2}(x-1) + 2 & \text{if } x > 1 \\ \frac{3}{2}(x+1) - 2 & \text{if } x < -1 \\ 2x & \text{otherwise} \end{cases} \tag{11}$$

$$\frac{d}{dx}f'(x) = \begin{cases} \frac{3}{2} & \text{if } x > 1 \\ \frac{3}{2} & \text{if } x < -1 \\ 2 & \text{otherwise} \end{cases} \tag{12}$$

as it can be seen the second derivative of f(x) is non-negative over its entire domain in all 3 different regions. so f(x) is a convex function. in order to find the unique minimmizer wee need to first determine the critical points of the function. These are the points where the derivative of the function is zero or undefined. Once we have found the critical points, we need to check which of them correspond to local minima, and then determine which of these is the absolute minimum. so we set the first derivative of each region to zero to find the critical points of each region :

$$\frac{d}{dx}f(x) = \begin{cases} \frac{3}{2}(x-1)+2=0 & \textit{if } x>1 \\ \frac{3}{2}(x+1)-2=0 & \textit{if } x<-1 \\ 2x=0 & \textit{otherwise} \end{cases} \tag{13}$$

for $x > 1$ and $x < -1$ the critical point doesn't lie in the defined region but for the $-1 <= x <= 1$ the ciritical point is when x=0 which the function has value -1 and it is the unique minimizer beacause the second derivative of f(x) at x=0 is positive so it is the unique minimmizer of f(x). the gradient of f(x) is :

$$\vec{\nabla} f(x,y) = [\frac{\partial f}{\partial x}] = [\begin{cases} \frac{3}{2}(x-1)+2 & \textit{if } x>1 \\ \frac{3}{2}(x+1)-2 & \textit{if } x<-1] \\ 2x=0 & \textit{otherwise} \end{cases}$$

the update algorithm for each region is :

$$x_t = \begin{cases} x_{t-1} - \frac{3}{2}(x_{t-1}-1) - 2 & \textit{if } x_{t-1} > 1 \\ x_{t-1} - \frac{3}{2}(x_{t-1}+1) + 2 & \textit{if } x_{t-1} < -1 \\ x_{t-1} - 2{*}x_{t-1} & \textit{otherwise} \end{cases} = \begin{cases} \frac{-1}{2}(x_{t-1}) - \frac{1}{2} & \textit{if } x_{t-1} > 1 \\ \frac{-1}{2}(x_{t-1}) + \frac{1}{2} & \textit{if } x_{t-1} < -1 \\ -x_{t-1} & \textit{otherwise} \end{cases}$$

so if we start by a $x_0 > 1$, we lie in the first region which is if $x_{t-1} > 1$. So to know the next x (x_1) will be in which region :
if $x_0 > 1$ then $\frac{-1}{2}(x_0) - \frac{1}{2} < -1$ so $x_1 < -1$
so the next x will lie in the second region which is if $x_{t-1} < -1$. Then like before if we resume by a $x_1 < -1$, we lie in the second region which is if $x_{t-1} < -1$. So to know the next x (x_2) will be in which region :
if $x_1 < -1$ then $\frac{-1}{2}(x_1) + \frac{1}{2} > 1$ so $x_2 > 1$
like the start of the process we now have an $x_{t-1} > 1$ and we want to know the next x which now we know it will lie in the second region where $x_{t-1} < -1$ and after that the next x will lie in the first region and so on. This process keeps continuing forever and never x lies in the region where $-1 <= x <= 1$ in which we have our unique minimmizer which is 0. Consequently this function could never reach this unique minimmum therefore it won't converge to a stationary point if it starts at point $x_0 > 1$.

b :
Smoothness implies that :

$$F(w^{(k+1)}) \leq F(w^{(k)}) - \eta||\nabla F(w^{(k)})||^2 + \frac{1}{2}\eta^2 L||\nabla F(w^{(k)})||^2$$

Our setting of η (Using $\eta = \frac{1}{L}$) implies :

$$F(w^{(k+1)}) \leq F(w^{(k)}) - \frac{1}{2L}||\nabla F(w^{(k)})||^2$$

Using the fact that the minimum of a sequence of elements is less than the average of the sequence by summing over k we have :

$$min_{0\leq k<K}||\nabla F(w^{(k)})||^2 \leq \frac{1}{K}\sum_{t=0}^{K-1}||\nabla F(w^{(k)})||^2$$

from the above equation we have : $||\nabla F(w^{(k)})||^2 \leq 2L * (F(w^{(k)}) - F(w^{(k+1)}))$

$$\begin{aligned}
&\leq \frac{2L}{K}\sum_{t=0}^{K-1}(F(w^{(k)}) - F(w^{(k+1)})) \\
&= \frac{2L}{K}(F(w^{(0)}) - F(w^{(k)})) \\
&\leq \frac{2L}{K}(F(w^{(0)} - F_*)
\end{aligned}$$