In the Name of God

Introduction to Machine Learning (25737-2)

Problem Set 03

Spring Semester 1402-03

Department of Electrical Engineering

Sharif University of Technology

Instructor: Dr. R. Amiri

Due on Ordibehesht 28th, 1403 at 23:59

($*$) starred problems are just optional!

1 Design a Neural Network

Design a Neural network capable of generating Hamming codes for 4-bit inputs. Describe the network, including the number of neurons in each layer and determine all the weights in the network. (note that the neural net should have 4 inputs and 3 outputs.)

2 ($*$) Design Simple Neural Network

Design a neural network with one hidden-layer that implements the following function:

$$(A \vee \bar{B}) \oplus (\bar{C} \vee \bar{D})$$

Draw the network and determine all its weigths.

3 ($*$) Vector Derivative

Consider following functions:

$$\mathbf{f}_1\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{\pi}\sin(\pi x_2) \\ e^{x_1-1}x_2^2 \\ x_1x_2 \end{bmatrix}, \quad \mathbf{f}_2\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} x_1+x_2+x_3 \\ x_1^2+x_2^2+x_3^2 \end{bmatrix}$$

$$\mathbf{f}(\mathbf{x}) = (\mathbf{f}_2 \circ \mathbf{f}_1)(\mathbf{x})$$

Determine $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ at point $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

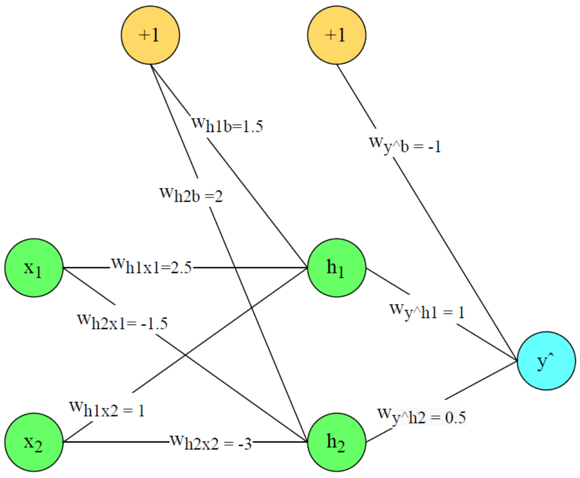
*Note that your solution must follow the methods mentioned in the course slides.

4 ($*$) Backpropagation Algorithm1

The following image shows a two-layer neural network with two nodes in the hidden layer and one node in the output. x_1 and x_2 are the two inputs to the network. Each node has a bias with value of 1.

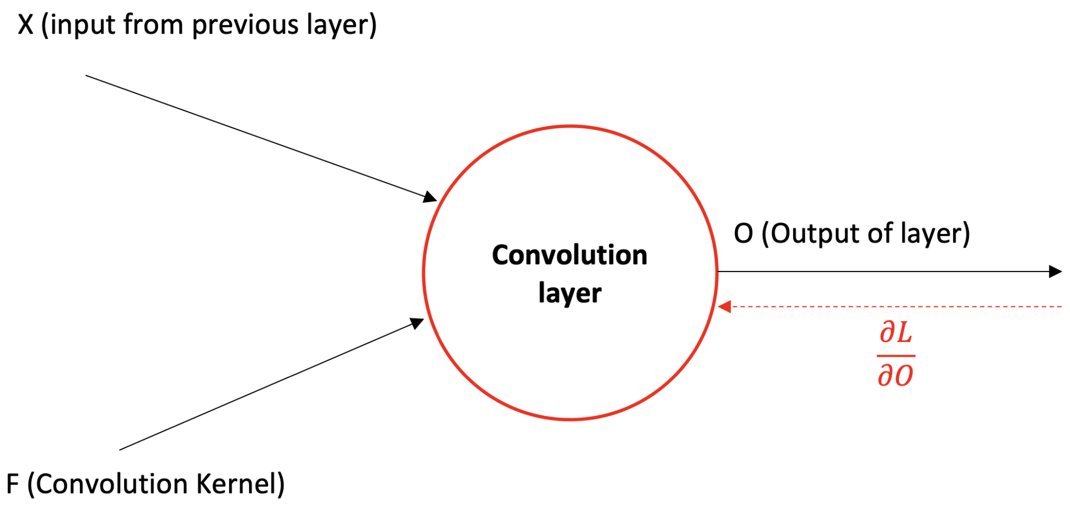
Assume that the value of the learning rate is 0.1 and the activation function is sigmoid in both hidden-layer and output-layer.

1. Calculate the value at nodes $\hat{y}, h_1, h_2$ for input $\{x_1 = 0, x_2 = 1\}$.
2. Execute one step of backpropagation algorithm for the previous input in part (a) and output $y = 1$.
3. Calculate the updated weights for the hidden-layer and output-layer (a total of 9 weights) by executing one step of the gradient descent algorithm.



5 Back propagation in CNN - Convolution in each direction!

In this question, we are going to do back propagation for a Convolutional layer and come up with a closed form answer for our required derivatives. First, let's look at the general schema of our layer:



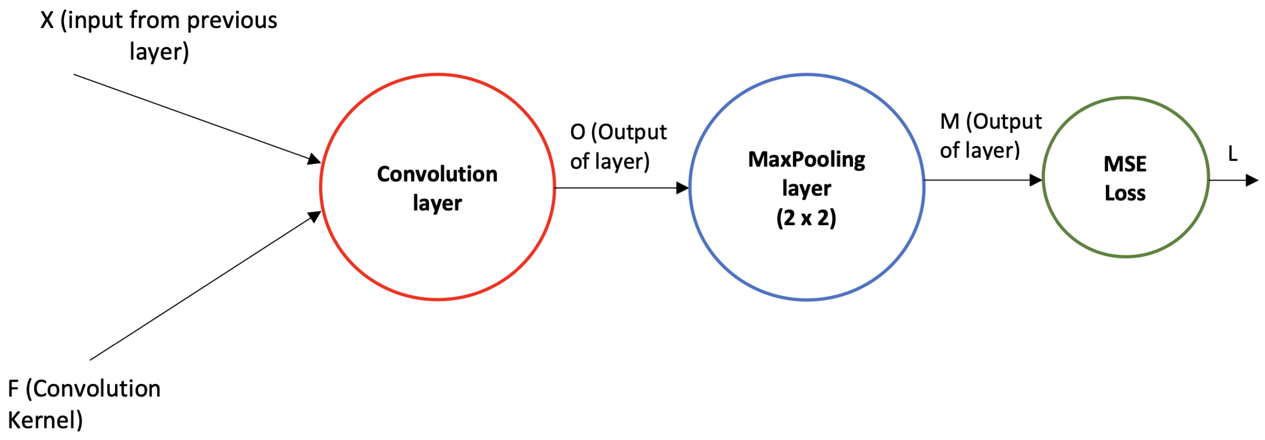
We know how *Forward Pass* for convolution layers are computed (if not, please refer to the course's slides!). Suppose X is a 2D matrix and thus, F is also 2D. Please **note** that the dimensions of these 2 matrices can arbitrarily change. Prove that:

$$\frac{\partial L}{\partial F} = X * \frac{\partial L}{\partial O}$$
$$\frac{\partial L}{\partial X} = F \otimes \frac{\partial L}{\partial O}$$

It is worth noting that $*$ sign is *Convolution* operation and $\otimes$ is ***full** convolution.*

6 Back propagation in CNN - an example

In this question, we are going to do *back propagation* operation on a sequence of layers, shown below; and write the update rules (and update weights) for each of the weights using *gradient descent.*



Here, please write the update rules and then update weights corresponding to variables/matrices X, F, O, M. The required information is provided below:

$$X = \begin{bmatrix} 1 & 7 & -1 & -7 & 10 & 11 \\ 2 & 8 & 0 & 0 & 12 & 13 \\ 3 & 9 & 0 & 0 & 0 & 0 \\ 4 & 10 & -4 & -10 & 0 & 0 \\ 5 & 11 & -5 & -11 & 16 & 17 \\ 6 & 12 & -6 & -12 & 14 & 15 \end{bmatrix}$$
$$F = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$
$$\hat{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where $\hat{L}$ is the matrix to calculate MSE loss with.

7 CNNs are universal approximators

We know that **CNNs are universal approximators**.It means that any function can be approximated using CNNs. We know the same thing about **MLP**s and we know MLPs are universal approximators, too. (For more information on this subject and seeing the proof for these statements, see this link and this link).

Now, we are looking to find out that if we can make an equivalent MLP from a CNN or not? If yes, please explain and say under what situation and circumstance we can find out the equivalent MLP. If not, please explain why under no circumstances, we can not find an equivalent MLP.
(For the sake of simplicity, you can explain your reasons and/or ideas on a 2D image X with dimensions $3 \times 3 \times 1$ and kernel F with dimensions 2×2 and then, explain how can your idea/explanations be generalized to higher dimensions)

8 Backpropagation Algorithm2

Consider the following convolutional network with given layers.

$$\text{Layer1 : Convolutional} \begin{cases} \mathit{Input:} & \mathbf{x} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \\ \mathit{Filter:} & \mathbf{v}_1 = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \\ \mathit{Output:} & \mathbf{z}_1 = tanh(\mathbf{x} * \mathbf{v}_1) \end{cases}$$

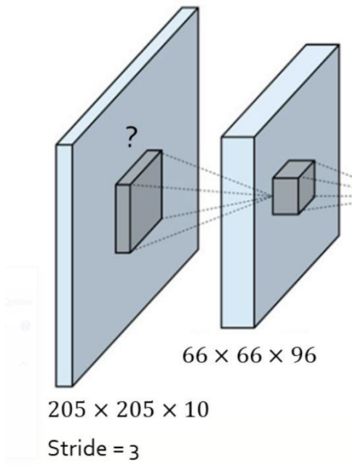
$$\text{Layer2 : Fully-connected} \begin{cases} \mathit{Input:} & \mathbf{z}_1 \\ \mathit{Filter:} & \mathbf{v}_2 = \begin{bmatrix} w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} \\ \mathit{Output:} & \mathbf{z}_2 = \sigma(\mathbf{v}_2 \times \mathbf{z}_1) \end{cases}$$

$$\text{Layer3 : Fully-connected} \begin{cases} \mathit{Input:} & \mathbf{z}_2 \\ \mathit{Filter:} & \mathbf{v}_3 = \begin{bmatrix} w_5 & w_6 \end{bmatrix} \\ \mathit{Output:} & y^* = \mathbf{v}_3 \times \mathbf{z}_2 \end{cases}$$

using backpropagation algorithm, obtain the derivative of $(y^* - y)^2$ with respect to w_1.

9 (∗)Model Parameters

Consider the following two-layer convolutional network.



1. Based on the input and output dimensions shown in the figure, determine the size of the kernel used for this operation.
2. Determine the number of trainable parameters in this layer.
3. Calculate the number of multiplication operations required to obtain the output.

10 (*) Receptive field

Determine a closed and non recursive formula for the receptive field of a 1*1 square in the nth convolution layer of a neural network in the first layer in terms of kernel size and stride.

11 Optimizating Deep Learning Models

In this question, we are going to discuss the *Momentum* in optimization algorithms.

11.1 Momentum in Optimization

Recall the *Gradient Descent* algorithm:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$$

Where θ_t is the parameters at time t, α is the learning rate, and $\nabla f(\theta_t)$ is the gradient of the loss function with respect to the parameters.
The *Momentum* algorithm is an extension of the *Gradient Descent* algorithm that adds a momentum term to the update rule:

$$\begin{aligned} v_{t+1} &= \beta v_t + (1-\beta)\nabla f(\theta_t) \\ \theta_{t+1} &= \theta_t - \alpha v_{t+1} \end{aligned}$$

Where v_t is the momentum term, and β is the momentum parameter.

11.1.1 Questions 1

The *Nesterov Accelerated Gradient* (NAG) algorithm is an extension of the *Momentum* algorithm that adds a correction term to the update rule:

$$\begin{aligned} v_{t+1} &= \beta v_t + (1-\beta)\nabla f(\theta_t - \alpha v_t) \\ \theta_{t+1} &= \theta_t - \alpha v_{t+1} \end{aligned}$$

Explain how the *NAG* algorithm works and how it improves the training process compared to the *Momentum* algorithm.

11.1.2 Questions 2

The *Adagrad* algorithm is an adaptive learning rate optimization algorithm that scales the learning rate based on the historical gradients:

$$\begin{aligned} g_{t+1} &= g_t + (\nabla f(\theta_t))^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{g_{t+1} + \epsilon}} \nabla f(\theta_t) \end{aligned}$$

Explain how the *Adagrad* algorithm works and how it improves the training process compared to the *Momentum* algorithm.

11.2 (*)Approximate Newton Methods

The *Newton's Method* is an optimization algorithm that uses the second derivative of the loss function to update the parameters. Write the update rule of the *Newton's Method* and explain how it works.

11.3 Adam Optimizer

The *Adam* optimizer is an adaptive learning rate optimization algorithm that combines the ideas of *Momentum* and *Adagrad*. Here is the update rule of the *Adam* optimizer:

$$\begin{aligned}
g_t &= \nabla_\theta f(\theta_{t-1}) \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_t &= \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}$$

1. Explain how the *Adam* optimizer works and how it improves the training process compared to the *Momentum* algorithm, line by line.
2. Explain why m_t have a bias towards zero in the early stages of training and how $\hat{m}_t$ corrects this bias.

12 Quadratic Error Function

Consider a quadratic error function of the form

$$E = E_0 + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H} (\mathbf{w} - \mathbf{w}^*)$$

where $\mathbf{w}^*$ represents the minimum, and the Hessian matrix $\mathbf{H}$ is positive definite and constant. Suppose the initial weight vector $w^(0)$ is chosen to be at the origin and is updated using simple gradient descent

$$\mathbf{w}^\tau = \mathbf{w}^{\tau-1} - \rho \nabla E$$

where τ denotes the step number, and ρ is the learning rate (which is assumed to be small). Show that, after τ steps, the components of the weight vector parallel to the eigenvectors of $\mathbf{H}$ can be written

$$w_j^\tau = \{1 - (1 - \rho\eta_j)^\tau\} w_j^*$$

where $w_j = \mathbf{w}^T \mathbf{u}_j$ and $\mathbf{u}_j$ and η_j are the eigenvectors and eigenvalues, respectively, of $\mathbf{H}$ so that

$$\mathbf{H}\mathbf{u}_j = \eta_j \mathbf{u}_j$$

Show that as $\tau \to \infty$, this gives $\mathbf{w}^\tau \to \mathbf{w}^*$ as expected, provided $|1 - \rho\eta_j| < 1$. Now suppose that training is halted after a finite number τ of steps. Show that the components of the weight vector parallel to the eigenvectors of the Hessian satisfy

$$\begin{aligned}
\mathbf{w}_j^{(\tau)} &\approx \mathbf{w}_j^* \quad \text{when} \quad \eta_j \gg (\rho\tau)^{-1} \\
|\mathbf{w}_j^{(\tau)}| &\ll |\mathbf{w}_j^*| \quad \text{when} \quad \eta_j \ll (\rho\tau)^{-1}
\end{aligned}$$

13 LSTM

Consider a standard LSTM cell with input dimension n, hidden state dimension m, and forget gate, input gate, and output gate activation functions denoted as $f(t)$, $i(t)$, and $o(t)$ respectively. Let W_f, W_i, and W_o represent the weight matrices associated with these gates. Additionally, let U_f, U_i, and U_o represent the recurrent weight matrices.

Given an input sequence $x(t)$, an initial hidden state $h(0)$, and the LSTM equations:

$$\begin{aligned}
f(t) &= \sigma(W_f x(t) + U_f h(t-1) + b_f) \\
i(t) &= \sigma(W_i x(t) + U_i h(t-1) + b_i) \\
o(t) &= \sigma(W_o x(t) + U_o h(t-1) + b_o) \\
\tilde{c}(t) &= \tanh(W_c x(t) + U_c h(t-1) + b_c) \\
c(t) &= f(t) \odot c(t-1) + i(t) \odot \tilde{c}(t) \\
h(t) &= o(t) \odot \tanh(c(t))
\end{aligned}$$

where σ is the sigmoid function, $\odot$ denotes element-wise multiplication, and $\tilde{c}(t)$ is the candidate cell state.

Prove that the LSTM cell can learn to remember information over long sequences by showing that the derivative of the cell state $c(t)$ with respect to $c(t-1)$ does not vanish as t increases.

14 (∗)RNN

14.1 1. Complex Dynamics of RNNs:

Derive the Jacobian matrix of the hidden state transitions in an RNN and discuss its role in understanding the stability of the network during training. Explain how the spectral radius of this matrix influences the behavior of the RNN, particularly in relation to the vanishing and exploding gradient issues.

14.2 2. BPTT in Depth:

Provide a detailed mathematical derivation of the Backpropagation Through Time (BPTT) process specifically for an RNN handling variable-length input sequences. Discuss the computational challenges and implications of applying BPTT in terms of memory and time complexity, especially when dealing with long sequences.

15 GANs

In this question, we are going to discuss the *GANs* and their training process.

15.1 GANs, a game theoritic approach

Game theory is a branch of mathematics that deals with the analysis of games. In this question, we are going to discuss the *GANs* from a game theoritic approach.
A Game is defined by the following elements:

- Players: The agents who are playing the game; row player and column player in a two-player game.
- Actions: The actions that each player can take; the strategies of each player.

- Payoffs: The rewards that each player receives based on the actions taken by all players; the utility function $u_i(\mathbf{a}_1, \mathbf{a}_2)$ for player i in a two-player game where $\mathbf{a}_1$ and $\mathbf{a}_2$ are the actions taken by player 1 and player 2, respectively.

Nash Equilibrium is a concept in game theory that describes a situation in which no player can increase their payoff by changing their strategy, assuming that all other players keep their strategies unchanged. For example, in a two-player game, a Nash Equilibrium is a pair of strategies, one for each player, such that each player's strategy is the best response to the other player's strategy.
For example, consider a two-player game called *Prisoner's Dilemma.* In this game, two players can either *Cooperate* or *Defect.* The payoffs for each player are as follows:

- If both players *Cooperate*, they both receive a payoff of 3,
- If one player *Cooperates* and the other *Defects*, the player who *Cooperates* receives a payoff of 0, and the player who *Defects* receives a payoff of 5,
- If both players *Defect*, they both receive a payoff of 1.

We can define the *Prisoner's Dilemma* as a game with the following matrix:

	Cooperate	Defect
Cooperate	3,3	0,5
Defect	5,0	1,1

Now, Lets discuss how to find the Nash Equilibrium of a game. The Nash Equilibrium of a game can be found by solving the following optimization problem:

$$\max_{\mathbf{a}_1} \min_{\mathbf{a}_2} \mathbf{u}_1(\mathbf{a}_1, \mathbf{a}_2)$$
$$\max_{\mathbf{a}_2} \min_{\mathbf{a}_1} \mathbf{u}_2(\mathbf{a}_1, \mathbf{a}_2)$$

Where $\mathbf{a}_1$ and $\mathbf{a}_2$ are the actions taken by player 1 and player 2, respectively, and $\mathbf{u}_1$ and $\mathbf{u}_2$ are the payoffs of player 1 and player 2, respectively.
Now, let's define the GANs as a game. In a GAN, we have two players, the *Generator* and the *Discriminator.* The Generator tries to generate samples that are similar to the real data, while the Discriminator tries to distinguish between the real and generated samples.

15.1.1 Questions 1

Define the GANs as a game. What are the players, actions, and payoffs in this game?

15.1.2 Questions 2

To find the Nash Equilibrium of the GANs game, we need to solve the following optimization problem:

$$\max_{\theta_G} \min_{\theta_D} \mathbf{u}_G(\theta_G, \theta_D) = \max_{\theta_G} \min_{\theta_D} \mathbf{E}_{\mathbf{x} \sim p_{\text{data}}}[\log(\mathbf{D}(\mathbf{x}))] + \mathbf{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - \mathbf{D}(\mathbf{G}(\mathbf{z})))]$$

Explain this optimization problem and how it relates to the training process of the GANs.

15.1.3 Questions 3

To solve the optimization problem, we calculus of variations. Let $y = y(x)$ be a function of x and the function we need to optimize is:

$$\int F(x, y, y')dx$$

For the function $y = y(x)$ that minimizes the integral, we know that:

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\left(\frac{\partial F}{\partial y'}\right) = 0$$

You do not need to prove this formula.
Now, solve the optimization problem of the GANs game using the calculus of variations. You need to find a closed form solution for θ_D.

15.2 GANs, you don't want to train them!

Training GANs is a challenging task due to the instability of the training process. In this question, we are going to discuss the challenges of training GANs.
You can solve question 1 and 2 or you can solve question 3. NO extra points will be given for solving all three questions.

15.2.1 Questions 1

Recall the optimization problem of the GANs game:

$$\max_{\theta_G} \min_{\theta_D} \mathbf{u}_G(\theta_G, \theta_D) = \max_{\theta_G} \min_{\theta_D} \mathbf{E}_{\mathbf{x}\sim p_{\text{data}}}[\log(\mathbf{D}(\mathbf{x}))] + \mathbf{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - \mathbf{D}(\mathbf{G}(\mathbf{z})))]$$

When using `Gradient Descent` to solve this optimization problem, first term is the *discriminator loss*. When you update the parameters of the discriminator, if you take the gradient of the GANs loss with respect to the generator parameters, first term will not be affected. Explain why the first term is not affected when updating the generator parameters. How does this affect the training process of GANs?

15.2.2 Questions 2

It's known that the training process of GANs is unstable and can suffer from the *mode collapse* problem. Explain the *mode collapse* problem and how it affects the training process of GANs.

15.2.3 Questions 3

If you solve the optimization problem of the GANs for optimal *Discriminator*, you will get the following optimization problem:

$$\begin{aligned}\mathbf{u}_D(\theta_G, \theta_D) &= 2D_{JS}(p_{\text{data}}||p_{\text{model}}) - 2log(2)\\ &= \frac{1}{2}D_{KL}(p_{\text{data}}||\frac{p_{\text{data}} + p_{\text{model}}}{2}) + \frac{1}{2}D_{KL}(p_{\text{model}}||\frac{p_{\text{data}} + p_{\text{model}}}{2}) - 2log(2)\end{aligned}$$

Where D_{JS} is the Jensen-Shannon divergence, D_{KL} is the Kullback-Leibler divergence (if you aren't familiar with *KL divergence*, it's a measure of how one probability distribution is different

from a second, reference probability distribution).
Another problem with training GANs is suffered from the *KL divergence* term in the loss function. For a better intuition, check the following image

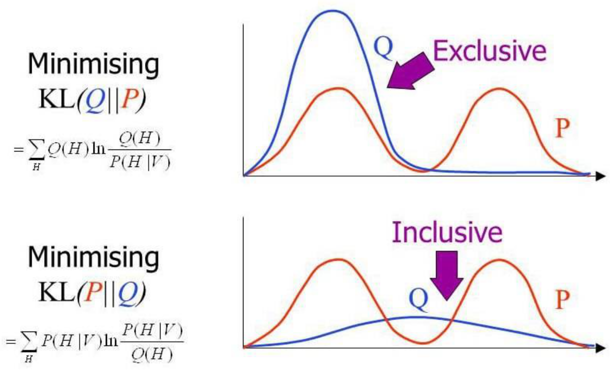


Figure 1: $p(x)$, the data distribution and $q(x)$, model distribution

To solve this problem, we can use the *Wasserstein GANs* (WGANs) that replace the *KL divergence* term with the *Wasserstein distance.* Explain the *Wasserstein distance* and how it solves the problem of the *KL divergence* term in the loss function of GANs.