Q1:
Optimizers are algorithms that reduce the loss of model by adjusting W and B. Gradient descent is also a optimiser, but we have faster algorithms like SGD, Adam optimiser etc. For example Adam optimiser works similar to gradient descent but there is a catch, it adjusts the learning the rate for each step such that it reaches faster to minimum point than normal gradient descent which just keeps the learning rate constant. They take the difference between actual and predicted data and adjust weights and biases to produce more correct results.

Q2:
GD:Considers the entire training dataset in each iteration to calculate the average gradient of the loss function. This average gradient is then used to update the model's parameters. Its advantage is that it provides more accurate descent direction due to averaging, leading to smoother convergence to a minimum. But it is computationally expensive as it considers whole dataset at every step.
SGD: SGD randomly picks one data point from the whole data set at each iteration to reduce the computations enormously. The advantage of using it is very fast, especially for large datasets, as it only processes one data point per update, but disadvantage is that the updates can be noisy due to the randomness of using a single data point. This can lead to a more zigzag path towards the minimum.
Mini-batch GD:
It lies in between of GD and SGD. It processes the data in small batches (mini-batches) of a pre-defined size. The gradient is calculated based on the average within each mini-batch, and then used to update the parameters. Advantage: Faster than GD as it doesn't use the entire dataset, and reduces noise compared to SGD by averaging over a mini-batch. Disadvantage: Finding the optimal mini-batch size can be a balancing act – too small and it approaches SGD's noise, too large and it approaches GD's computational cost.

Q3:
Adam stands for Adaptive Moment Estimation. It is an optimisation algorithm that combines the benefits of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp, .key features of adam include:

- Adaptive Learning Rates: Adam calculates individual learning rates for each model parameter based on their historical gradients. This helps the optimizer adjust to different parameter sensitivities and speeds up convergence.
- Momentum with Decay: Similar to momentum, Adam maintains an exponentially decaying average of past gradients (called the first moment). This term incorporates the direction of previous updates, smoothing the optimization process.
- RMSprop with Bias Correction: Adam also keeps an exponentially decaying average of the squared gradients (called the second moment). However, it applies a bias correction factor to account for the initial bias towards smaller values at the beginning of training.
- Disadvantages are:
- May Not Always Find the Best Minimum**:** In some cases, Adam might settle for a local minimum instead of the global minimum of the loss function.

- Sensitive to Hyperparameters: While less sensitive than SGD, Adam's performance can still be impacted by the choice of learning rate and decay rates.

Mathematical Formulation

Adam maintains two moving averages of the gradient: the first moment (mean) and the second moment (uncentered variance).

1. **Initialize Parameters:** - $m_0 = 0$ (first moment vector) - $v_0 = 0$ (second moment vector) - $t = 0$ (time step) 2. **Hyperparameters:** - α: Learning rate - β_1: Decay rate for the first moment (typically 0.9) - β_2: Decay rate for the second moment (typically 0.999) - ϵ: A small constant to prevent division by zero (typically 10^{-8})

3. **Update Rule:** - Increment time step: $t = t + 1$ - Compute gradient: $g_t = \nabla_\theta L(\theta_t)$ - Update biased first moment estimate:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

- Update biased second moment estimate:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- Correct bias in first moment:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

- Correct bias in second moment:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- Update parameters:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Q4:
Both rmsprop and adam are optimization algorithms that are used to minimize loss in neural network training process. Though they have some simimlrities but they have some key differences that affects their usage in various scenarios. Some similarities between them include features like adaptive learning rates, but various differences are as follows:

- Incorporation of Momentum: Rmsprop mainly focuses on adapting LR based on square gradients of each parameter, it does not not explicitly incorporate momentum. On the other hand Adam combines adaptive learning rates with momentum. It maintains an exponentially decaying average of past gradients, giving the updates a sense of direction and aiding convergence in situations where SGD gets stuck.
- Bias correction: rmsprop does not apply any bias correction to the historical squared gradients. This can lead to underestimating the variance in the initial stages of training, while Adam oncludes bias correction for both the first moment (moving average of gradients) and the second moment (moving average of squared gradients). This ensures more reliable estimates of these moments from the beginning.

Impact of differences:

- Convergence: Adam often exhibits faster convergence compared to Rmsprop, especially for problems with non-convex loss functions. The momentum term in Adam helps the updates overcome shallow valleys or oscillations that might slow down Rmsprop.
- Hyperparameter Tuning: Adam generally requires less tuning of hyperparameters like the learning rate. The bias correction and momentum features make it less sensitive to these values compared to Rmsprop.

Q5:

There is no best algorithm, we have to decide upon what kind of data we have and how much resources we have. Moreover you might also need to experiment with different optimisers to get the best:

SGD: It is computationally cheap and can be used when you have very large datasets but not very much resources to run it, but one tradeoff is that it takes a very zigzag path to optimal point.

Rmsprop: Adapts learning rates for each parameter based on squared gradients, addressing fluctuating gradients better than SGD. Good for problems with non-convex loss functions and fluctuating gradients, can be an alternative to Adam when momentum is not necessary. Disadvantage is that it lacks momentum, might converge slower than Adam in some cases.

Adam: Combines adaptive learning rates with momentum, converges faster than SGD or Rmsprop, generally requires less hyperparameter tuning. It is generally used by default in ML applications. Some disadvantages are that it may not always find the global minimum, can be computationally expensive compared to SGD.

Q6:
Overfitting: It is the situation where model performs very well on the training data but it performs badly on the cross validation data. It learns training data very well but it is not able to generalize results on unseen data. It is referred to as High variance.
Underfititng: It is the situation where model is not even able to learn training data well, so training loss is itself very high than baseline error. It is referred to as High bias.

Q7:
Vanishing and exploding gradients are two issues that can arise during training in deep neural networks.

- Vanishing gradients: During backpropagation, the error (loss) is propagated backward through the layers to calculate the gradients for each weight. In vanishing gradients, these gradients become progressively smaller as they travel back through the network towards the earlier layers. This small gradients hinder the changes in parameters very slow, this cann arise due to usage of certain activation functions that attain almost const value for large values of input, like sigmoid function for very large and very small x.
- Exploding gradients: On the opposite end of the spectrum, exploding gradients occur when the gradients become increasingly larger as they propagate backward. Its causes include using certain activation function that can grow exponentially for large values. The

weight updates in the earlier layers become excessively large, potentially causing the weights to explode (overflow) and become unusable. The learning process becomes unstable and erratic.

Q8:

Batch normalization and layer normalization are techniques used in deep learning to address internal covariate shift and improve the stability and speed of training neural networks.

Batch normalization: It focuses on normalizing the activations of a hidden layer within a mini-batch during training. It subtracts the mean of the activations within the mini-batch from each activation then it divides each activation by the standard deviation of the mini-batch activations. It introduces two learnable parameters, gamma and beta, which allow the model to scale and shift the normalized activations back to the original data range, if necessary. Benefits include Reduces Internal Covariate Shift, reduces Sensitivity to Learning Rates. Disadvantages: effectiveness depends on batch size, Smaller batch sizes can lead to less accurate estimates of the mean and standard deviation, reducing its effectiveness. Increased Training Time by normalization step adds a small overhead to the training process.

Layer normalization: It normalizes the activations across each feature in the input layer or a hidden layer, independent of the mini-batch. Similar to BN, LN subtracts the mean and divides by the standard deviation, but it does this for each feature vector independently across the mini-batch dimension. It also uses learnable gamma and beta parameters for scaling and shifting. Drawbacks: Potentially Slower Training, Less Effective for Very Large Feature Sizes.

Q9:

Regularization in machine learning is a set of techniques to prevent models from becoming overly complex and prone to overfitting. Overfitting occurs when a model memorizes the training data too well, losing its ability to generalize and perform well on unseen data. Regularization techniques work by introducing a penalty term to the learning process. This penalty discourages the model from having overly large weights or complex structures. For eg in Linear regression:

Regularized linear regression

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 \boxed{+\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}\right]$$

Here the Lambda is regularization factor, If you increase it’s value then it will penalize model from over estimating weights. Higher values of w will increase the error term so model will try to keep them low.

Q10:

Dropout layers are regularization techniques that are used to reduce model overfitting .Dropout layers work by randomly dropping out a certain proportion of neurons (along with their connections) from a layer during training. This essentially creates a thinned-out version of the network for each training iteration. It prevents model from learning very complex patterns that are actually redundant and can make model prone to overftting. If the model becomes too focused on the specific details of the training data (e.g., a small imperfection in a particular digit), it might struggle to recognize similar unseen digits that lack those imperfections. Benefits: prevents overfitting, improve robustness, reduces training time.

Q11:

L1 regularization (Lasso regression): L1 regularization adds the absolute value of the sum of the weights in your model to the loss function. This penalty term is like a cost for having large weights.

L2 Regularization (Ridge Regression): L2 regularization adds the square of the sum of the weights in your model to the loss function. This penalty term discourages large weights, but doesn't necessarily drive them to zero.

Advantages:

Improved Generalization**:** By reducing model complexity, L1 regularization helps the model focus on the most important features, leading to better performance on unseen data.

Interpretability**:** Since some weights become zero, it's easier to understand which features are most important for the model's predictions.

Disadvantages:

Not all features are binary: L1 might set irrelevant features to zero, but some features might have small non-zero values, making interpretation less clear-cut in some cases.

Q12:

Validation accuracy is metric that teels us whether our model performs well on unseen data or not. We cannot consider training accuracy as metric to judge the models performance because that is seen data, the model is put to test when it performs well on unseen data or not. It is also used to check if the model is overfit or not. If model performs weel on training data but not on validation data it means that it is overfitt. By using validation accuracy we can also do hyper parameter tuning. Early Stopping: Validation accuracy allows us to employ a technique called early stopping. Here, we stop training the model once the validation accuracy starts to decline. This prevents the model from further overfitting on the training data.

Q13:

Data augmentation is method that increases size of training set by transforming, scaling, adding noise etc. to existing set. There can be certain usecases like for eg you are training a model for OCR now In you dataset letters like A,H,I,X,V etc are symmetric about vertical axis, then we can take mirror images of that specific data points to increase training set size, other example include transforming, random cropping, etc. For speech recognition we can add various noise in background like crowd noise, car horn noise, weak signal noise etc.

Disadvantages of Data Augmentation:

- Potential for Introducing Noise**:** If not applied carefully, data augmentation techniques can introduce unrealistic or nonsensical distortions to the data. This can confuse the model and hinder its performance.
- Increased Training Time and Computational Cost: Transforming existing data points takes additional processing power, which can lead to longer training times. This can be a concern for very large datasets or computationally expensive models.
- Domain-Specific Effectiveness: The effectiveness of data augmentation techniques depends on the specific problem and data domain. What works well for images might not be suitable for text data or other domains.
- Limited Applicability: Data augmentation is most effective for tasks where the underlying patterns can be preserved under certain transformations. It might not be as beneficial for tasks that rely on very specific details in the data.

Q14:

Transfer learning is a powerful technique in machine learning where knowledge gained from a pre-trained model on one task (source task) is applied to a different but related task (target task). It leverages the learned features and representations from the source task as a starting point for the target task, often leading to faster training times and improved performance on the target task compared to training a model from scratch. It is especially useful when you have less amount of data for training.

There are two main approaches to using transfer learning:

- Feature Extraction: In this approach, you freeze the weights of the pre-trained model and only train the final layers on your target task dataset. The pre-trained layers act as feature extractors, providing informative representations for your target task.
- Fine-tuning**:** Here, you unfreeze some or all of the pre-trained model's weights and fine-tune them on your target task dataset. This allows the model to adjust the learned features to better suit the target task while still benefiting from the pre-trained knowledge.

Overall, transfer learning is a valuable technique that can significantly improve the performance and efficiency of machine learning models. By leveraging pre-trained models, you can achieve better results with less data and training time.

Q15:

The bias-variance tradeoff is a fundamental concept in machine learning that deals with the relationship between a model's complexity, its accuracy on the training data (bias), and its ability to generalize well to unseen data (variance). It essentially forces you to find a sweet spot between a model that's too simple and a model that's too complex.

Bias: This refers to the systematic underfitting of the model. A high bias means the model is too simple and underestimates the complexity of the relationship between the features and the target variable. This results in an inability to capture the true patterns in the data, leading to consistently wrong predictions on unseen data.

Variance: This refers to the model's sensitivity to the specific training data. A high variance means the model is too complex and overfits the training data. It memorizes the specifics of the training set but fails to capture the generalizable patterns. This leads to the model performing well on the training data but performing poorly on unseen data.

Imagine a target function you're trying to learn with your model. As you increase the model's complexity (by adding more parameters, using a more complex architecture), the variance tends to increase. The model becomes more flexible and can fit the training data better, potentially reducing bias. However, this flexibility can also lead to the model memorizing noise or irrelevant details in the training data, hurting its ability to generalize.

Conversely, if you keep the model simple (with fewer parameters), the bias will likely be higher. The model might not be able to capture the intricacies of the relationship between features and target, leading to underfitting.

Q16: I would prefer one neural network approach, reasons for it include:

A single network can learn a unified representation of the patient data (weight, temperature, CBC, etc.) This allows the model to identify features that might be relevant across multiple diseases, potentially improving overall performance.

If you train separate model for all the diseases then every modl will have its own error which will separately affect the final result, but If i going with only one network then it would all be included in single network, stopping propagation of error.

Data Imbalance: Hospitals likely see more frequent cases of some diseases (e.g., gastrointestinal issues) compared to others (e.g., heart problems). Training separate models might lead to performance bias towards the more frequent diseases due to data imbalance.

Q17:

(a) Three weight bias matrix:

  Between input and A1 : Weight = (n1*n0), bias = (n1*1)

  Between A1 and A2 : Weight = (n2*n1), bias = (n2*1)

  Between A2 and output : Weight = (n3*n2), bias = (n3*1)

  Verification: first layer W*X ⇒ (n1*n0)*(n0*m) = (n1*m)

  second layer W*X ⇒ (n2*n1)*(n1*m) = (n2*m), similarly for third layer.

(b) Hidden layers: ReLU, it is a popular choice for hidden layers, as it is computationally effective, adds non linearity and does not face problem of vanishing gradient.
  Output layer: Sigmoid as it is well suite for binary classification.

(c) Hidden Layers: ReLU (same as above)
  Output layer: Softmax, as it converts linear values to probabilities that sum upto one, ans is equivalent to sigmoid for multiclass classification.

(d) Hidden layers: ReLU (same as above)
  Output layers: Linear activation function as regression models aim to predict continuous values, so Linear activation is generally used.