**Assignment-1 (neural networks)**

**1. Modify an existing neural network model to improve performance?**

There are several ways to modify an existing neural network model to improve its performance, some of which are:

* Add more layers: Adding more layers to the model can increase its capacity and allow it to learn more complex patterns in the data.
* Change activation functions: Changing the activation functions of the layers can improve the model's ability to model non-linear relationships in the data.
* Adjust the learning rate: Changing the learning rate can improve the convergence speed and stability of the training process.
* Modify the loss function: Modifying the loss function can make the model more sensitive to certain types of errors or prioritize different types of performance metrics.
* Change the optimizer: Changing the optimizer can help the model converge faster and achieve better results on the validation data.
* Use pre-trained models: Using pre-trained models, such as those from transfer learning, can improve the performance of the model by leveraging the knowledge learned from other datasets.
* Increase or decrease the batch size: Changing the batch size can affect the convergence speed and stability of the model and can help prevent overfitting.
* Use regularization: Regularization techniques, such as L1, L2, or dropout, can be used to prevent overfitting and improve the model's ability to generalize.

It's important to note that not all modifications may improve the performance of the model, and some may even decrease it. It's best to experiment with different modifications and evaluate their effect on the performance of the model on the validation data.

**2. Explain how different approaches affect the performance of the model?**

Different approaches can have a significant impact on the performance of a machine learning model. Here are some examples of how different approaches can affect the performance of a model:

**Model architecture:** The architecture of the model can significantly affect its performance. Adding more layers to the model can increase its capacity and ability to learn more complex patterns in the data. However, adding too many layers can cause overfitting, which can decrease performance. Choosing an appropriate architecture, including the number of layers and their sizes, can significantly impact the performance of the model.

**Regularization:** Regularization techniques, such as L1, L2, or dropout, can help prevent overfitting by adding a penalty term to the loss function. These techniques can help the model generalize better, which can improve its performance on validation data.

**Optimizers:** Optimizers, such as SGD, Adam, or RMSprop, can affect the rate of convergence and stability of the training process. Some optimizers may work better than others on specific types of problems or data.

**Learning rate:** The learning rate controls how much the weights are updated during each iteration of training. A learning rate that is too high can cause the model to overshoot the optimal weights, while a learning rate that is too low can cause the model to converge slowly or not at all.

**Batch size:** The batch size determines how many samples are used in each iteration of training. A larger batch size can result in faster convergence but may decrease the model's ability to generalize to new data.

**Preprocessing:** Preprocessing techniques, such as normalization or scaling, can affect the performance of the model. These techniques can help improve the stability of the training process and can help the model converge faster.

**Data augmentation:** Data augmentation techniques, such as random cropping, flipping, or rotation, can help improve the performance of the model by increasing the size and diversity of the training dataset.

In summary, the choice of approach can significantly affect the performance of a machine learning model. It's important to experiment with different approaches and evaluate their impact on the performance of the model on the validation data.

**For the IMDB example that we discussed in class, do the following:**

1. **You used two hidden layers. Try using one or three hidden layers and see how doing so affects validation and test accuracy.**

When I used one hidden layer, the validation loss: 0.2697 and Validation accuracy: 0.8903 which is highest in Epoch 5.

When I used three hidden layers, the validation loss: 0.2897 and Validation accuracy: 0.8674 which is highest in epoch 4.

1. **Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on?**

Here, I have tried using 32 units,

Text

Description automatically generated with medium confidence

Validation loss: 0.2783, Validation accuracy: 0.889

Here, I have tried using 64 units,

Text

Description automatically generated

Validation loss: 0.2746, Validation accuracy: 0.8878

1. **Try using the mse loss function instead of binary\_crossentropy.**

When I use MSE loss function as below,

**model.Compile (optimizer='adam', loss='mse', metrics=['accuracy'])**

got below results:

Validation loss: 0.0838, Validation accuracy: 0.8863

1. **Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.**

As mentioned, I have used tanh activation,

model = Sequential([

Dense (64, input\_shape=(input\_shape,), activation='tanh'),

Dense (32, activation='tanh'),

Dense (num\_classes, activation='softmax`')

])

This sets the activation function to tanh in the first two layers of the model and sets the output layer's activation function to softmax for multi-class classification problems.

It's important to note that changing the activation function can affect the performance of the model. It's best to experiment with different activation functions and evaluate their impact on the performance of the model on the validation data to determine the best choice for the specific problem at hand.

1. **Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.**

**Code:**

from tensorflow.keras.layers import Dropout

model = Sequential([

Dense(64, input\_shape=(input\_shape,), activation='relu'),

Dropout(0.5),

Dense(32, activation='relu'),

Dropout(0.5),

Dense(num\_classes, activation='softmax')

])

Dropout is a commonly used regularization technique that randomly drops out (sets to zero) a certain percentage of the inputs to a layer during each training iteration. This helps prevent overfitting by reducing the interdependencies between neurons and promoting more robust feature representations.

To add dropout regularization to the model, we can add a Dropout layer after each fully connected layer. Here is an example of how to add dropout regularization to a two-layer fully connected neural network:

we add a Dropout layer with a rate of 0.5 after each fully connected layer. The rate parameter controls the percentage of inputs that are randomly set to zero during each training iteration.

It's important to note that adding dropout regularization can also affect the convergence of the model, so it may be necessary to adjust the learning rate or other hyperparameters accordingly. It's also important to experiment with different dropout rates and evaluate their impact on the performance of the model on the validation data to determine the best choice for the specific problem at hand.