**4.**

Question 4

Why do dropouts help avoid overfitting?



Because neighbor neurons can have similar weights, and thus can skew the final training



Having less neurons speeds up training

**Question 8**

The dropout rate determines how many neurons are removed from the network during training. Which of the two cases below do you think will happen if it is set too high?



The network would lose specialization to the effect that it would be inefficient or ineffective at learning.



Training time would increase due to the extra calculations being required for higher dropout.

**Question 1**

When using image augmentation with image\_dataset\_from\_directory, what happens to your raw image data on-disk?



A copy will be made, and the copies are augmented



A copy will be made, and the originals will be augmented



Nothing



The images will be edited on disk, so be sure to have a backup

**Question 3**

The diagram for traditional programming had Rules and Data in, but what came out?



Answers



Binary



Machine Learning



Bugs

Question 6

Applying convolutions on top of a DNN will have what impact on training?



It will be slower



It will be faster



There will be no impact



It depends on many factors. It might make your training faster or slower, and a poorly designed convolutional layer may even be less efficient than a plain DNN!

train\_dataset = tf.keras.utils.image\_dataset\_from\_directory(

directory=TRAIN\_DIR,

batch\_size=32,

image\_size=(28,28),

label\_mode='categorical',

color\_mode = "grayscale", # Use this argument to get just one color dimension, because it is greyscale

)

Your model has 68,888 total parameters and the reference is 30,000

Warning! this exceeds the reference which is 30,000. If the kernel crashes while training, switch to a simpler architecture.

Your model has 68,888 trainable parameters and the reference is 30,000

Warning! this exceeds the reference which is 30,000. If the kernel crashes while training, switch to a simpler architecture.

* The number of trainable parameters should align with the size of your dataset to avoid overfitting.
* A general **rule of thumb** is to have fewer parameters than the total number of data points in your training set.

**Guidelines**:

* **Small datasets (<10k samples)**: ~10k–100k parameters.
* **Medium datasets (~50k samples)**: ~100k–500k parameters.
* **Large datasets (>1M samples)**: Models with millions of parameters.

In Con2D the number of learnable parameters depends on filters and not on input data size but in DNN the learnable parameters depend on input data size. That’s why some times if you have two models one has 3 con2d layers following maxpooling layers if it passes to fullyconnected layes would generate more learnable parameters than model has 3 con2d following maxpooling layers, because input size for fullyconnected layer for shallower model is larger than deep model.

Question 8

Using the default settings, how does the TextVectorization standardize the string inputs?



By lowercasing the strings.



By arranging the strings in alphabetical order.



By stripping punctuation.



By lowercasing and stripping punctuation.

tf.keras.layers.StringLookup(num\_oov\_indices=0)

Question 2

Using the default settings, what does the 'max\_tokens' parameter do when initializing the TextVectorization layer?



It errors out if there are more than max\_tokens distinct words in the corpus



It specifies the maximum size of the vocabulary, and picks the most common ‘max\_tokens - 2’ words



It specifies the maximum size of the vocabulary, and picks the most common ‘max\_tokens’ words



It specifies the maximum size of the vocabulary, and picks the most common ‘max\_tokens - 1’ words

How many reviews are there in the IMDB dataset and how are they split?



60,000 records, 80/20 train/test split



50,000 records, 50/50 train/test split



60,000 records, 50/50 train/test split



50,000 records, 80/20 train/test split

**Question 2**

How does an LSTM help understand meaning when words that qualify each other aren’t necessarily beside each other in a sentence?



They load all words into a cell state



They don’t



They shuffle the words randomly



Values from earlier words can be carried to later ones via a cell state

An **LSTM (Long Short-Term Memory)** network is designed to handle sequences of data by maintaining a **cell state** that can carry information from earlier words to later ones, even if they are not adjacent in a sentence. This mechanism allows the LSTM to capture long-term dependencies and relationships in sequential data. The cell state, along with the gates (forget gate, input gate, and output gate), helps the network decide what information to keep, update, or forget, enabling it to understand context in sentences where related words are not close to each other.

**Question 3**

What’s the best way to avoid overfitting in NLP datasets?



Use LSTMs



Use GRUs



Use Conv1D



None of the above

Avoiding overfitting in NLP datasets typically requires strategies like:

1. **Data Augmentation**: Increasing the diversity of the training dataset.
2. **Regularization Techniques**: Using dropout, L2 regularization, or other techniques to reduce overfitting.
3. **Early Stopping**: Halting training once validation performance stops improving.
4. **Hyperparameter Tuning**: Adjusting model parameters to balance bias and variance.
5. **Increasing Training Data**: Expanding the dataset size helps the model generalize better.

### 5.

**Question 5**

Why does sequence make a large difference when determining semantics of language?



Because the order in which words appear dictate their meaning



Because the order of words doesn’t matter



Because the order in which words appear dictate their impact on the meaning of the sentence



It doesn’t

The sequence of words is crucial in language because it determines how meaning is constructed. For example:

* "The cat chased the dog" has a different meaning than "The dog chased the cat," even though the same words are used.

The position and relationship of words in a sentence influence its overall semantics, which is why sequence models like LSTMs, GRUs, and Transformers are widely used in NLP tasks.

**7.**

Question 7

What’s the output shape of a bidirectional LSTM layer with 64 units?



(128,None)



(None, 64)



(None, 128)



(128,1)

**2.**

**Question 2**

What is a major drawback of word-based training for text generation instead of character-based generation?



Because there are far more words in a typical corpus than characters, it is much more memory intensive



There is no major drawback, it’s always better to do word-based training



Character based generation is more accurate because there are less characters to predict



Word based generation is more accurate because there is a larger body of words to draw from

### Explanation:

* **Word-based training** requires a larger vocabulary, which increases memory usage and computational cost.
* **Character-based training** deals with a much smaller set of possible tokens (characters), making it less memory-intensive, though it may require more training to understand context effectively.
* Question 3
* What are the critical steps in preparing the input sequences for the prediction model?
* 
* Splitting the dataset into training and testing sentences.
* 
* Pre-padding the subphrases sequences.
* 
* Generating subphrases from each line using n\_gram\_sequences.
* 
* Converting the seed text to a token sequence using texts\_to\_sequences.

### 

### Explanation:

1. **Generating subphrases using n\_gram\_sequences**: This step creates smaller sequences from a larger text to train the model on varying lengths of context.
2. Pre-padding may also be necessary, but it is not explicitly required in this case unless the sequences are of variable length and need alignment. Splitting into training and testing is a general step for any ML workflow but not specific to preparing input sequences.

**4.**

Question 4

When predicting words to generate poetry, the more words predicted the more likely it will end up gibberish. Why?



It doesn’t, the likelihood of gibberish doesn’t change



Because the probability of prediction compounds, and thus increases overall



Because the probability that each word matches an existing phrase goes down the more words you create



Because you are more likely to hit words not in the training set

When predicting a sequence of words, each prediction depends on the prior ones, and the probabilities of selecting the "correct" word compound. Over time, even small errors in predictions can accumulate, leading to sentences that drift away from coherent or meaningful text, resulting in gibberish. This phenomenon is particularly common in generative models that rely heavily on statistical predictions.

**Question**

True or False: When building the model, we use a sigmoid activated Dense output layer with one neuron per word that lights up when we predict a given word.

**False.**

In text generation tasks, typically, the output layer is a **softmax** activated Dense layer with as many neurons as the vocabulary size, not one per word. The softmax activation function provides a probability distribution over all the possible words in the vocabulary, and the model selects the word with the highest probability.

A sigmoid activation would not be suitable for this task because it's used for binary classification (outputting a probability between 0 and 1 for each class) rather than multi-class problems like text generation where you want to select from a large set of possible words.