ML Special Session Pt.3

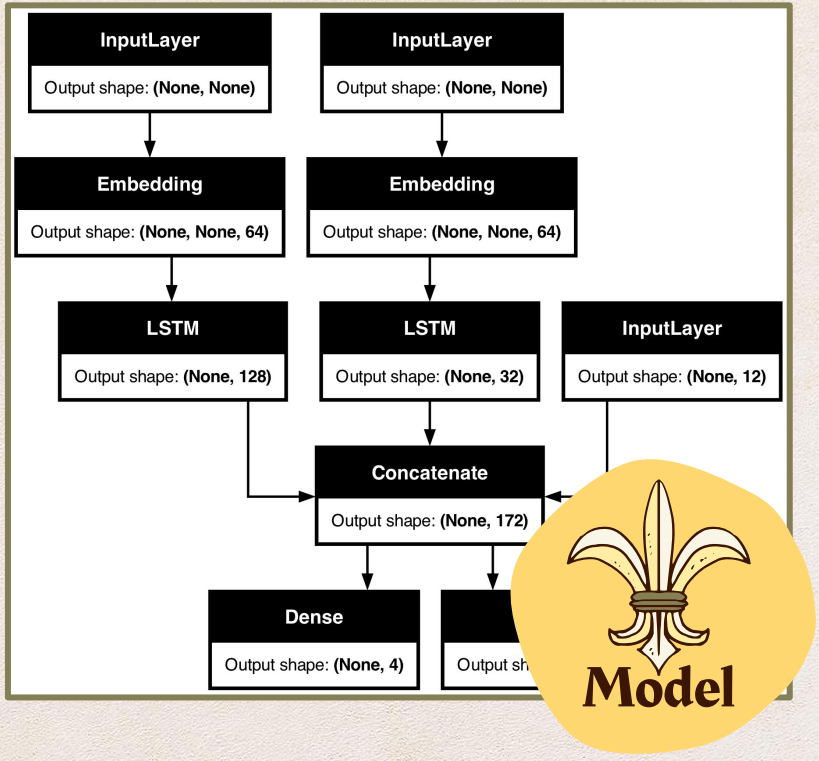
DJS-InfoMatrix ML(Pranav)

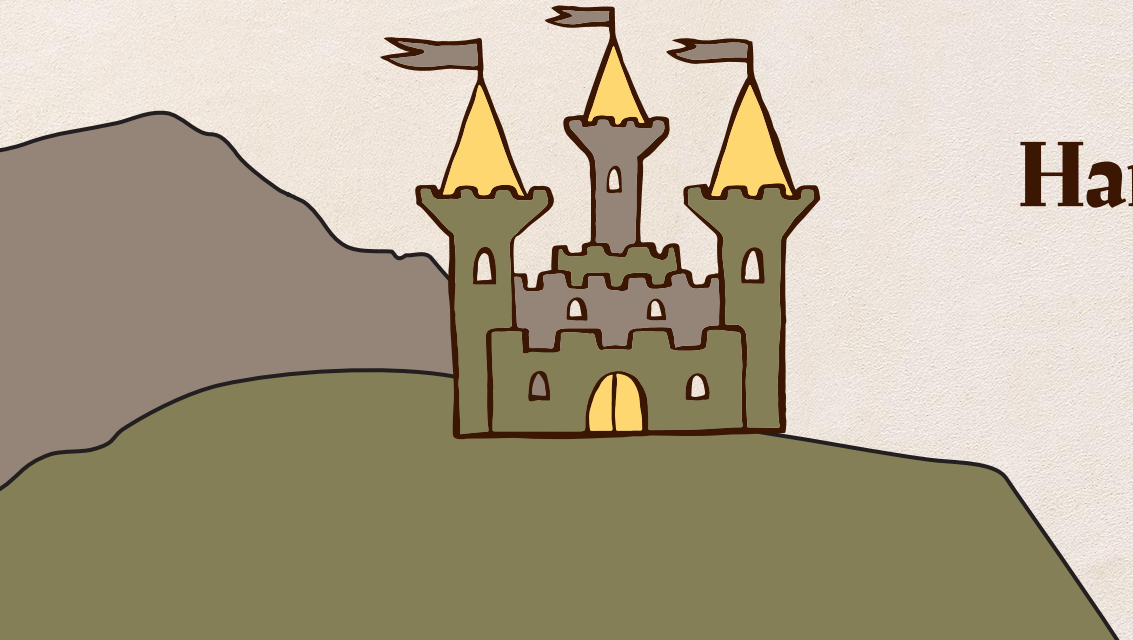
Table of contents

Recap

Situation:- Building a system for ranking customer issue tickets by priority and routing them to the correct department.

3 Inputs & 2 Outputs





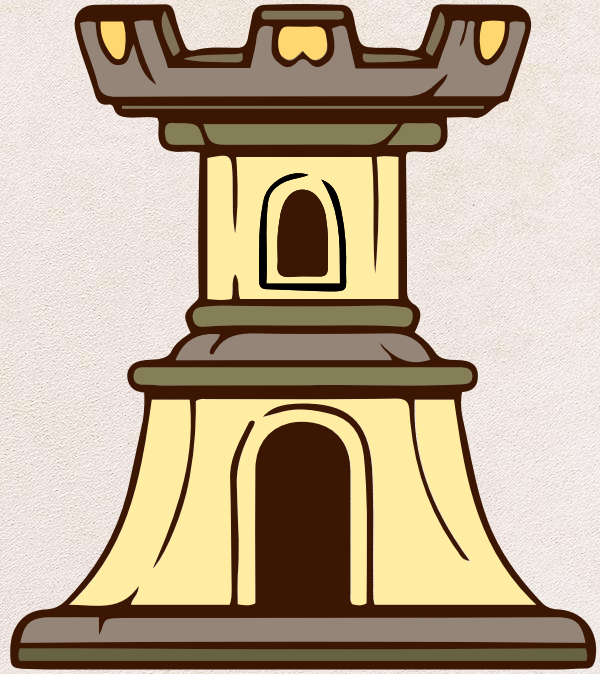
01

Handling Multiple I/O

For NLP

- import numpy as np
- from tensorflow.keras.models import Model
- from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, concatenate
- from tensorflow.keras.preprocessing.text import Tokenizer
- from tensorflow.keras.preprocessing.sequence import pad_sequences
- from tensorflow.keras.utils import to_categorical

Importing Modules



Data Generation

```
# Dummy data
titles = ["Help with order", "Issue with payment", "Login problem"]
bodies = ["I cannot find my order", "Payment method not working", "Cannot log in to my account"]
tags = [0, 1, 2]  # categorical tags

# Tokenize the text inputs
tokenizer = Tokenizer(num_words=100)
tokenizer.fit_on_texts(titles + bodies)
title_sequences = tokenizer.texts_to_sequences(titles)
body_sequences = tokenizer.texts_to_sequences(bodies)

# Pad the sequences
title_data = pad_sequences(title_sequences, maxlen=5)
body_data = pad_sequences(body_sequences, maxlen=20)
```

Topics: Tokenizers, fit_on_texts, text_to_sequences & padding of sequences.

Data Preprocessing

Tokenize

Function to assign tokens to words

fit_on_texts

Do Tokenization

text_to_sequences

Sentences in form of tokens

Padding

Equalize the length

Mission and vision

```
# One-hot encode the tags
tag_data = to_categorical(tags, num_classes=3)

# Dummy outputs
priority_scores = np.random.rand(len(titles), 1)
departments = np.random.randint(0, 3, len(titles))
department_data = to_categorical(departments, num_classes=3)
```

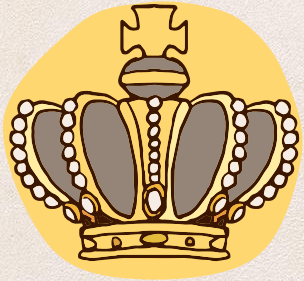
random.rand

random.randint

3 Things from last slide

rand

Randomized real values



randint

Randomized Integer values

to_categorical

GUESS?

ABHI-RIZZ BREAK

Highlight 1

Model Attributes

	Goals	Details
01	INPUT	3 Classes
02	EMBEDDING	Vihaan OVerikar pls XXXplain
03	LSTM	Abhijeet bol bkl
04	CONCATENATE	Combine layers
05	DENSE	Split into 2 Outputs

Model Pt. 1

```
# Model inputs
title_input = Input(shape=(5,), name='title_input')
body_input = Input(shape=(20,), name='body_input')
tag_input = Input(shape=(3,), name='tag_input')

# Embedding and LSTM layers for text inputs
embedding_layer = Embedding(input_dim=100, output_dim=10, input_length=5)
title_embeddings = embedding_layer(title_input)
title_lstm = LSTM(32)(title_embeddings)

embedding_layer_body = Embedding(input_dim=100, output_dim=10, input_length=20)
body_embeddings = embedding_layer_body(body_input)
body_lstm = LSTM(32)(body_embeddings)
```

Model Pt. 2

```
# Concatenate all inputs
concat_layer = concatenate([title_lstm, body_lstm, tag_input])

# Dense layers for output
dense1 = Dense(64, activation='relu')(concat_layer)
priority_output = Dense(1,
                        activation='sigmoid',
                        name='priority_output')(dense1)
department_output = Dense(3,
                          activation='softmax',
                          name='department_output')(dense1)

# Model definition
model = Model(inputs=[title_input, body_input, tag_input],
              outputs=[priority_output, department_output])
```

Compile & Fit

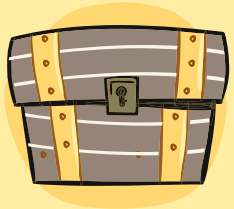
```
# Compile the model
model.compile(optimizer='adam',
              loss={'priority_output': 'binary_crossentropy',
                    'department_output': 'categorical_crossentropy'},
              metrics={'priority_output': 'accuracy',
                       'department_output': 'accuracy'})

# Train the model on dummy data
model.fit([title_data, body_data, tag_data],
          [priority_scores, department_data],
          epochs=45,
          batch_size=2)
```

Multiple I/O, so fitting is different

Multiple I/O, so 2 different loss & metrics as well during compiling.

02

When not to use Sequential Layer



Multiple

I/O

Any layer

Has multiple I/O

Layer

Sharing

Special Case

You want non-linear topology (e.g. a residual connection, a multi-branch model)

Feature Extraction

```
initial_model = keras.Sequential(
    [
        keras.Input(shape=(250, 250, 3)),
        layers.Conv2D(32, 5, strides=2, activation="relu"),
        layers.Conv2D(32, 3, activation="relu"),
        layers.Conv2D(32, 3, activation="relu"),
    ]
)
feature_extractor = keras.Model(
    inputs=initial_model.inputs,
    outputs=[layer.output for layer in initial_model.layers],
)

# Call feature extractor on test input.
x = ops.ones((1, 250, 250, 3))
features = feature_extractor(x)
```

Once a Sequential model has been built, it behaves like a Functional API model. This means that every layer has an input and output attribute. These attributes can be used to do neat things, like quickly creating a model that extracts the outputs of all intermediate layers in a Sequential model.

Transfer Learning using Pre-Trained Models

```python
# Load a convolutional base with pre-trained weights
base_model = keras.applications.Xception(
    weights='imagenet',
    include_top=False,
    pooling='avg')

# Freeze the base model
base_model.trainable = False

# Use a Sequential model to add a trainable classifier on top
model = keras.Sequential([
    base_model,
    layers.Dense(1000),
])

# Compile & train
model.compile(...)
model.fit(...)
```

A **Xception**
Load a pre-trained model

B **Freezing**
Trainable = False

C **Make your model**
Design a model

D **Compile & Fit**
Custom Data needed

ABHI-RIZZ BREAK

Highlight 2

03

Subclassing

1 **Sampling**

Stochastic Function to generate noise

2 **Encoder**

3 **Decoder**

4 **VAE**

End-to-End Model

Sampling

```python
class Sampling(layers.Layer):
    """Uses (z_mean, z_log_var) to sample z, the vector encoding a digit."""

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.seed_generator = keras.random.SeedGenerator(1337)

    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = ops.shape(z_mean)[0]
        dim = ops.shape(z_mean)[1]
        epsilon = keras.random.normal(shape=(batch, dim), seed=self.seed_generator)
        return z_mean + ops.exp(0.5 * z_log_var) * epsilon
```

Encoder

```python
class Encoder(layers.Layer):
    """Maps MNIST digits to a triplet (z_mean, z_log_var, z)."""

    def __init__(self, latent_dim=32, intermediate_dim=64, name="encoder", **kwargs):
        super().__init__(name=name, **kwargs)
        self.dense_proj = layers.Dense(intermediate_dim, activation="relu")
        self.dense_mean = layers.Dense(latent_dim)
        self.dense_log_var = layers.Dense(latent_dim)
        self.sampling = Sampling()

    def call(self, inputs):
        x = self.dense_proj(inputs)
        z_mean = self.dense_mean(x)
        z_log_var = self.dense_log_var(x)
        z = self.sampling((z_mean, z_log_var))
        return z_mean, z_log_var, z
```

Variational Autoencoders

```python
class VariationalAutoEncoder(keras.Model):
    """Combines the encoder and decoder into an end-to-end model for training."""

    def __init__(
        self,
        original_dim,
        intermediate_dim=64,
        latent_dim=32,
        name="autoencoder",
        **kwargs
    ):
        super().__init__(name=name, **kwargs)
        self.original_dim = original_dim
        self.encoder = Encoder(latent_dim=latent_dim, intermediate_dim=intermediate_dim)
        self.decoder = Decoder(original_dim, intermediate_dim=intermediate_dim)

    def call(self, inputs):
        z_mean, z_log_var, z = self.encoder(inputs)
        reconstructed = self.decoder(z)
        # Add KL divergence regularization loss.
        kl_loss = -0.5 * ops.mean(
            z_log_var - ops.square(z_mean) - ops.exp(z_log_var) + 1
        )
        self.add_loss(kl_loss)
        return reconstructed
```

ABHI-RIZZ

HIGHLIGHT PT 3


THANK YOU MI LADS