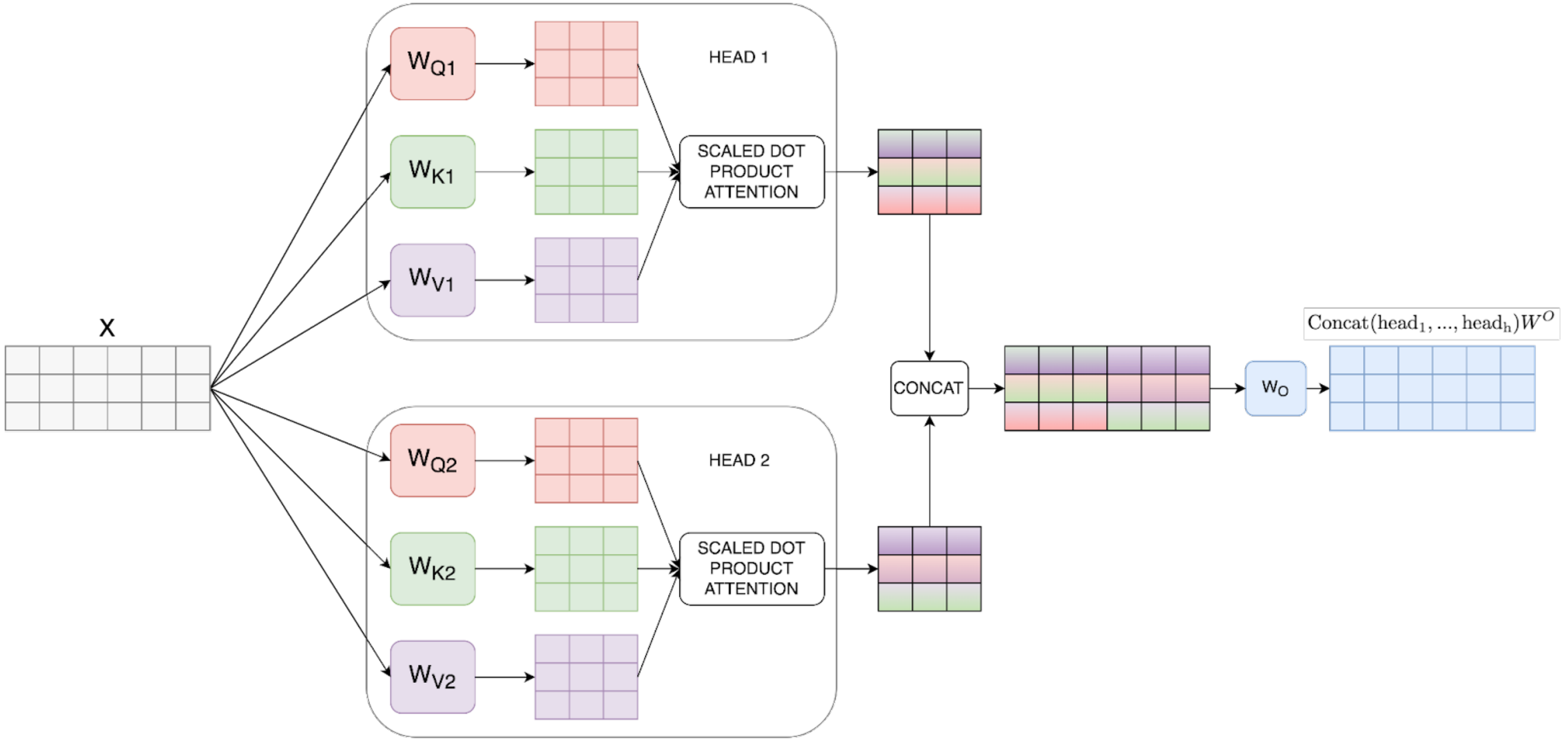
Transforming NLP

Axel Sirota

AI and Cloud Consultant

@AxelSirota


X
W_{Q1}
W_{K1}
W_{V1}
HEAD 1
SCALED DOT PRODUCT ATTENTION
W_{Q2}
W_{K2}
W_{V2}
HEAD 2
SCALED DOT PRODUCT ATTENTION
CONCAT
W_O
$\text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$

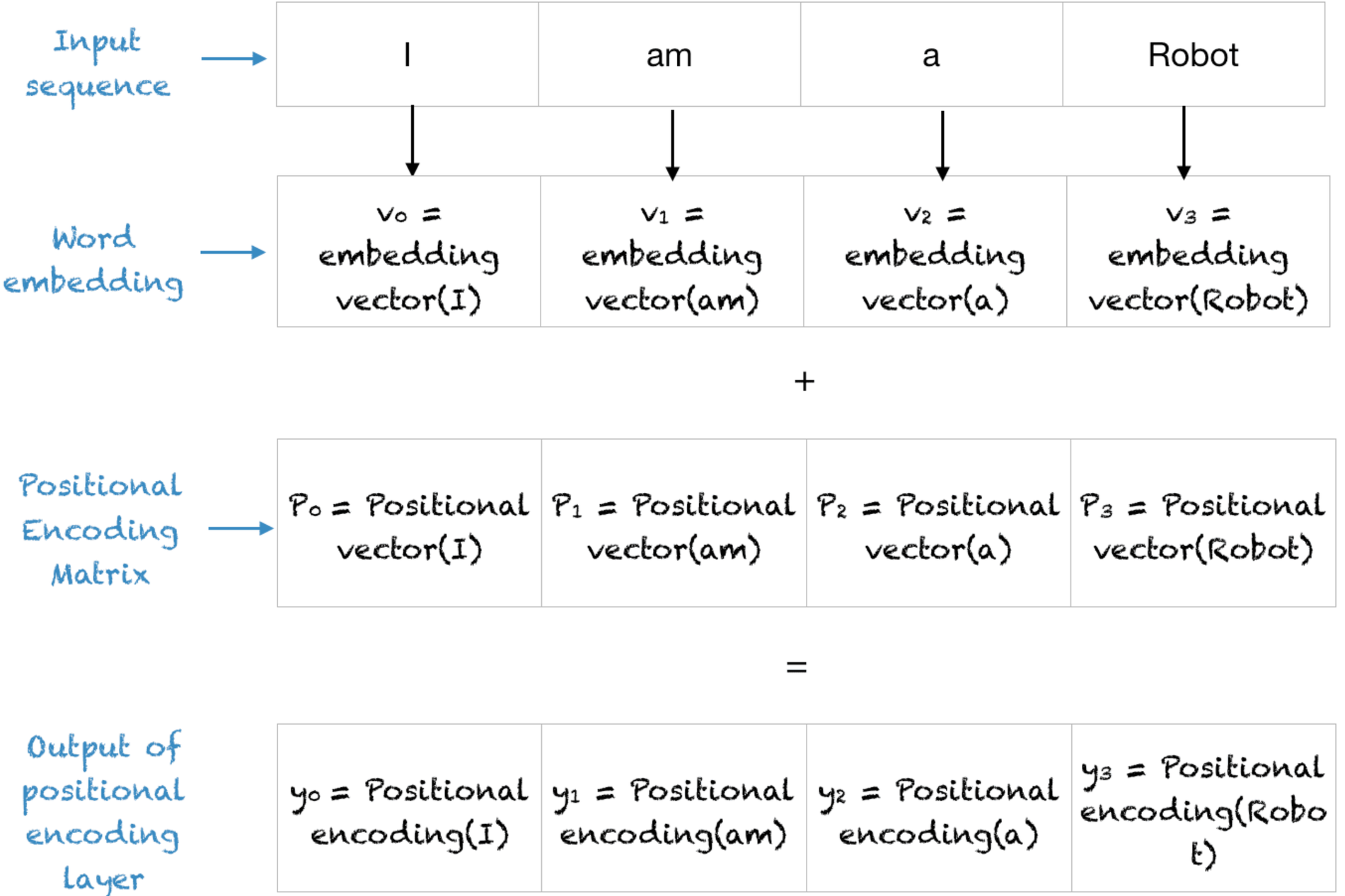
Multi-head Attention is a vital component within Transformers.

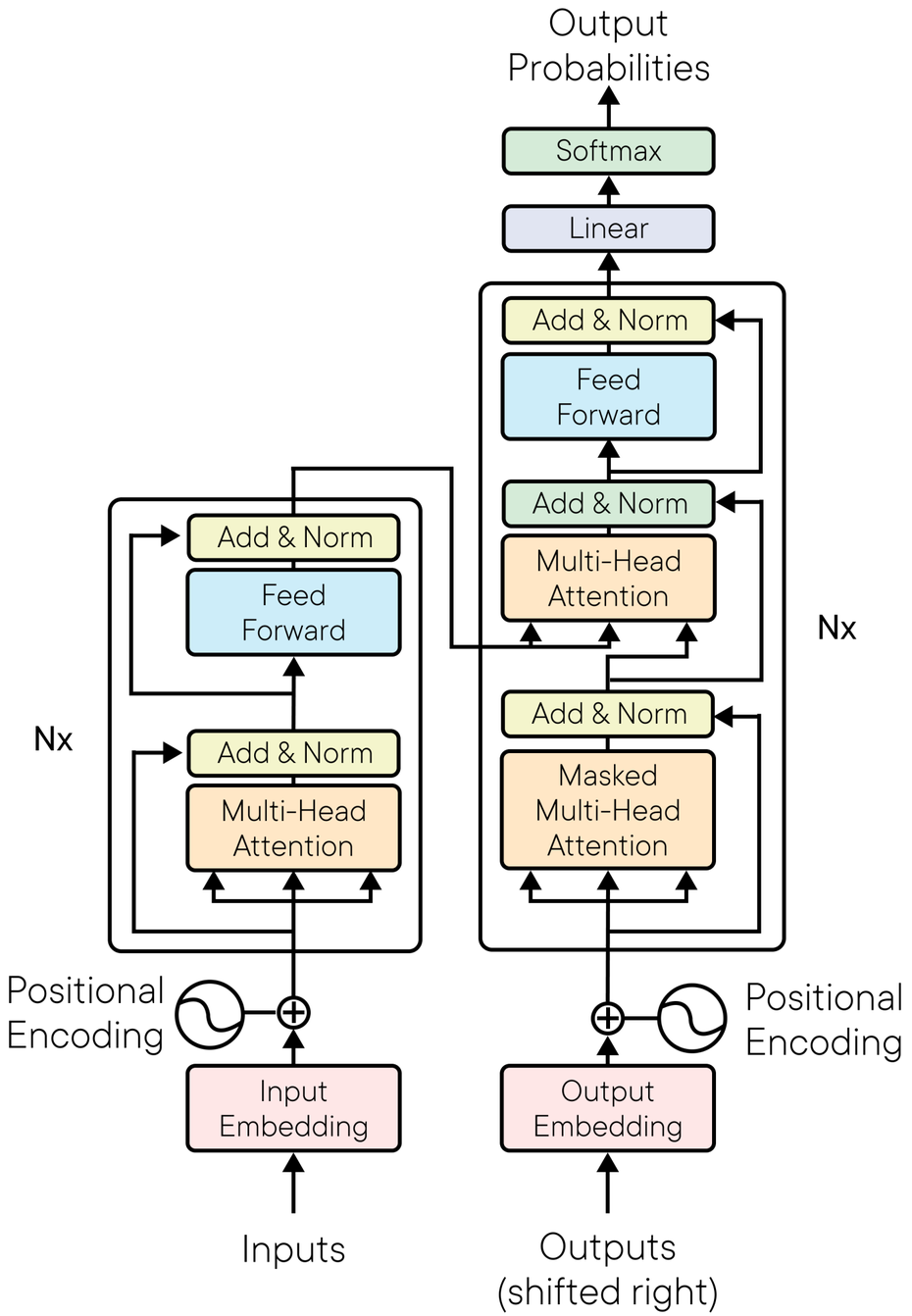
POSITIVE

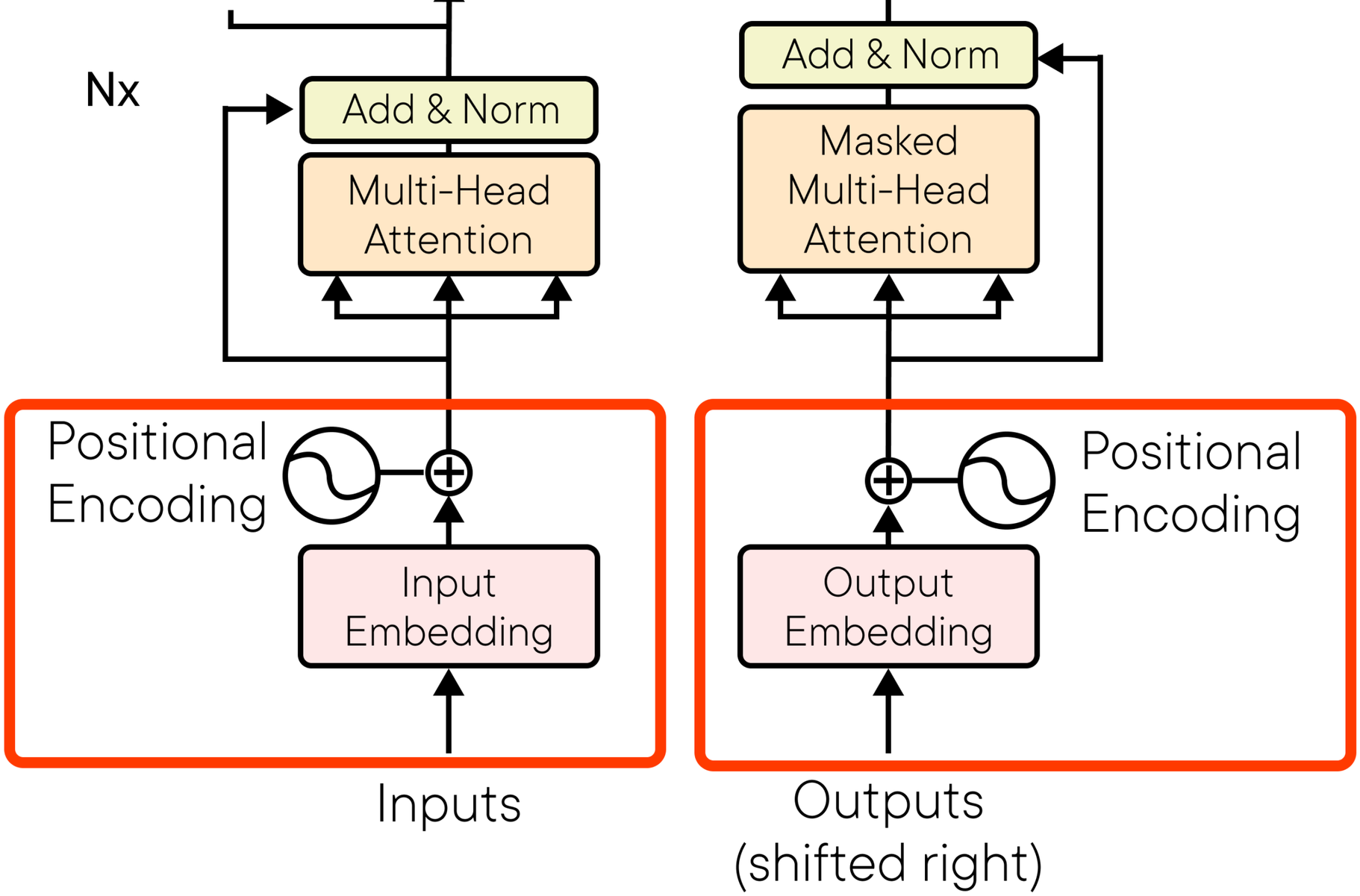
I loved the restaurant even though the pasta was mediocre

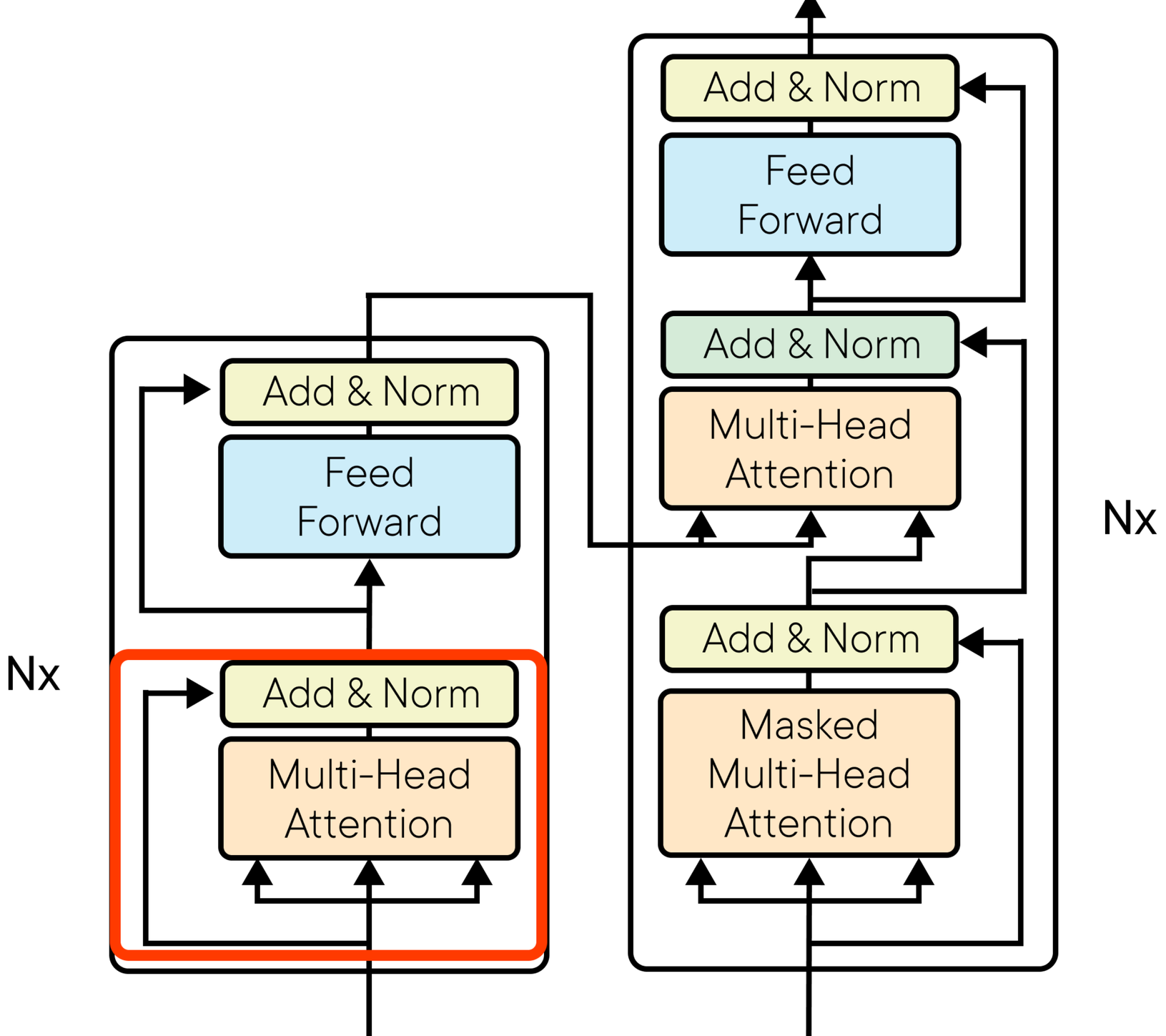
NEGATIVE

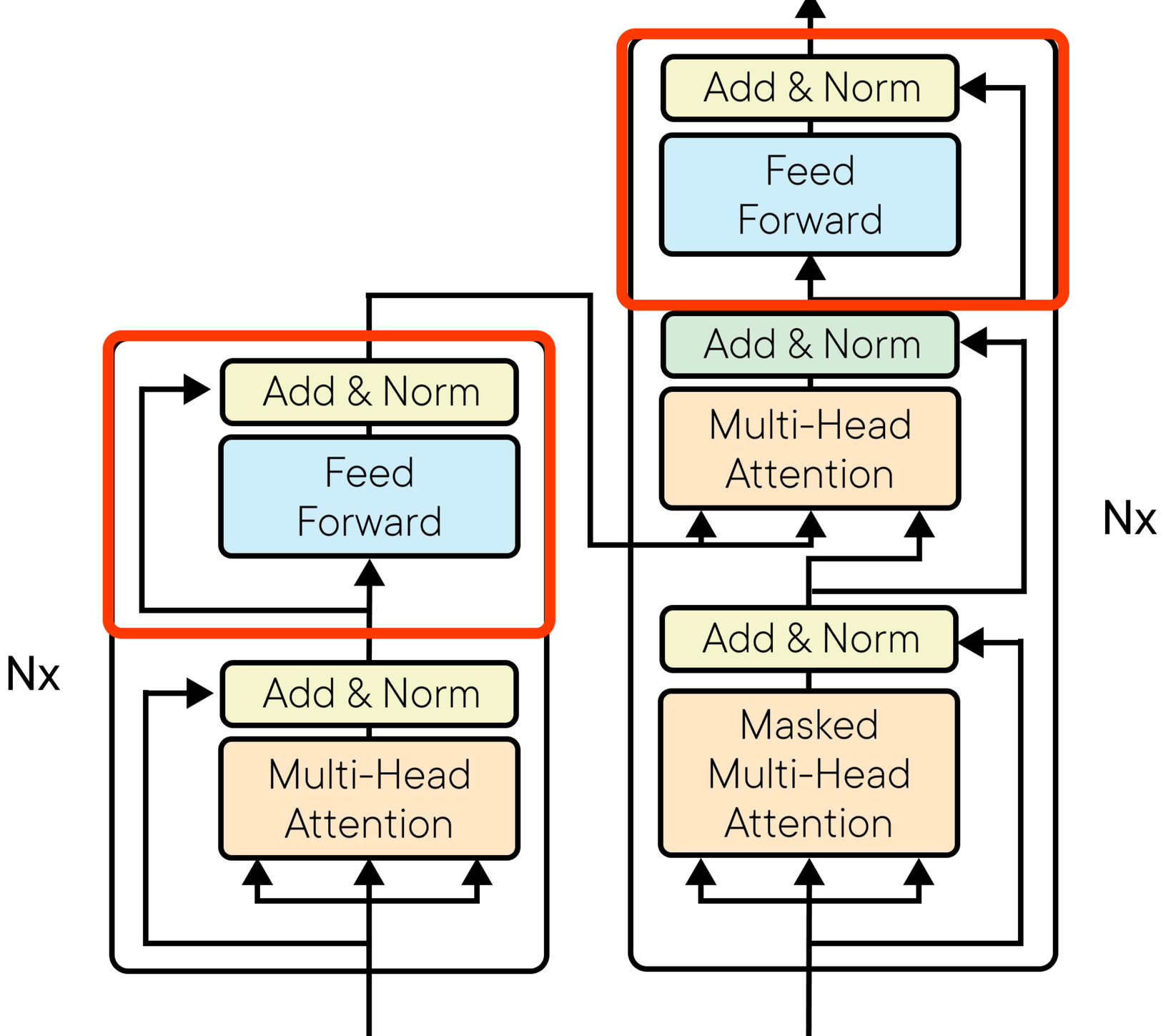
I hate this place even though it has fantastic steack

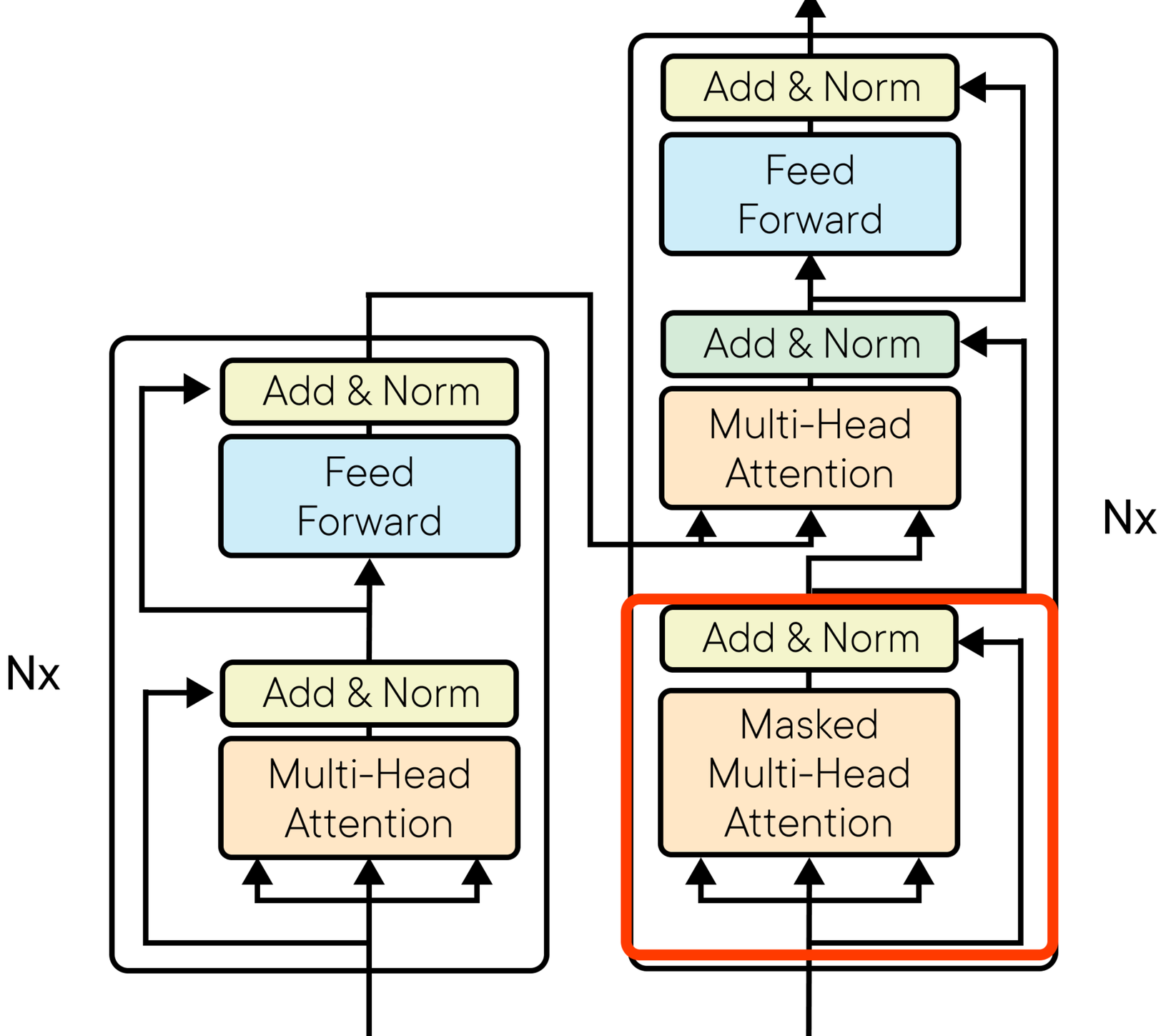

Input sequence
I
am
a
Robot
Word embedding
v_0 = embedding vector(I)
v_1 = embedding vector(am)
v_2 = embedding vector(a)
v_3 = embedding vector(Robot)
+
Positional Encoding Matrix
P_0 = Positional vector(I)
P_1 = Positional vector(am)
P_2 = Positional vector(a)
P_3 = Positional vector(Robot)
=
Output of positional encoding layer
y_0 = Positional encoding(I)
y_1 = Positional encoding(am)
y_2 = Positional encoding(a)
y_3 = Positional encoding(Robot)

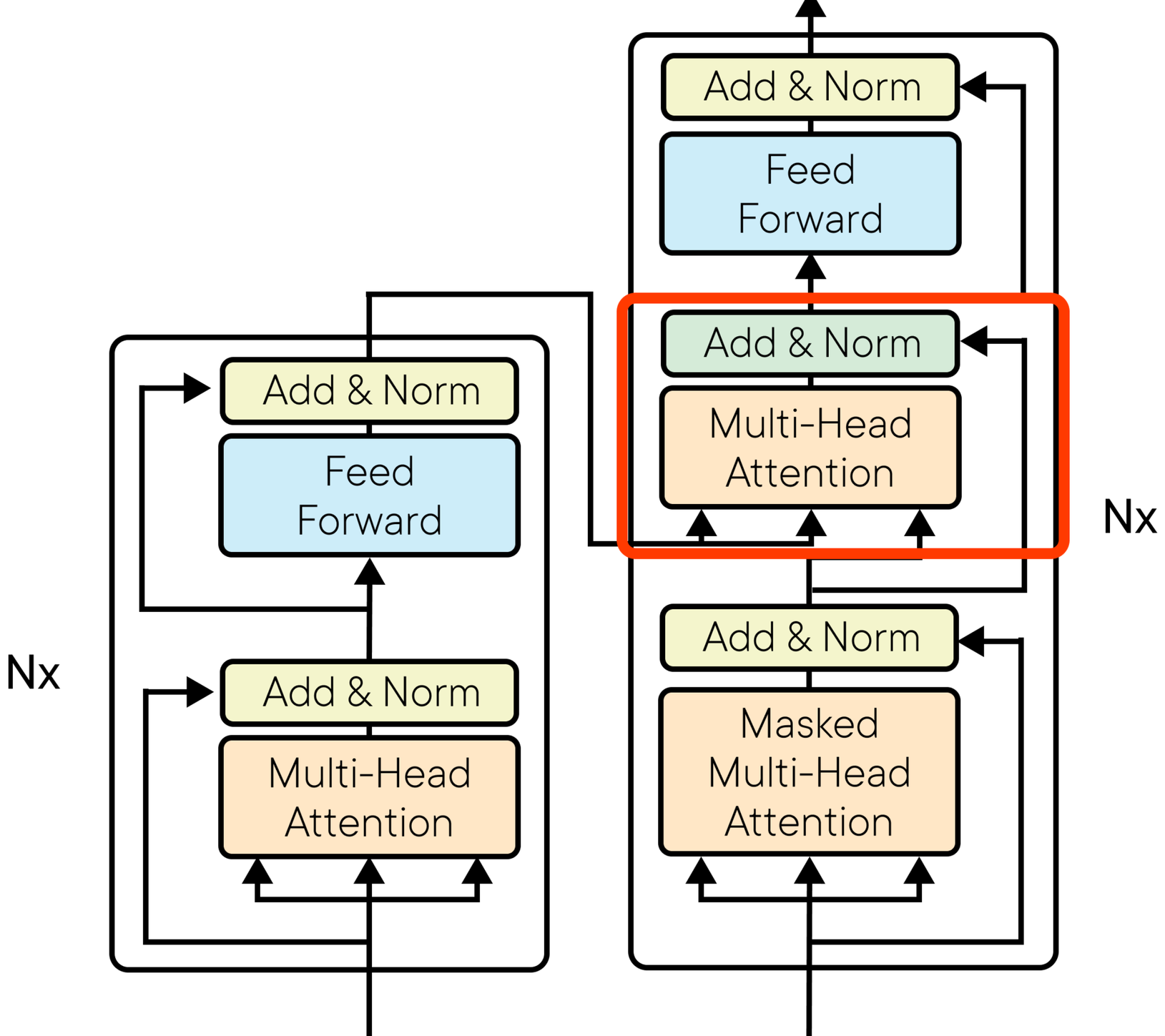

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

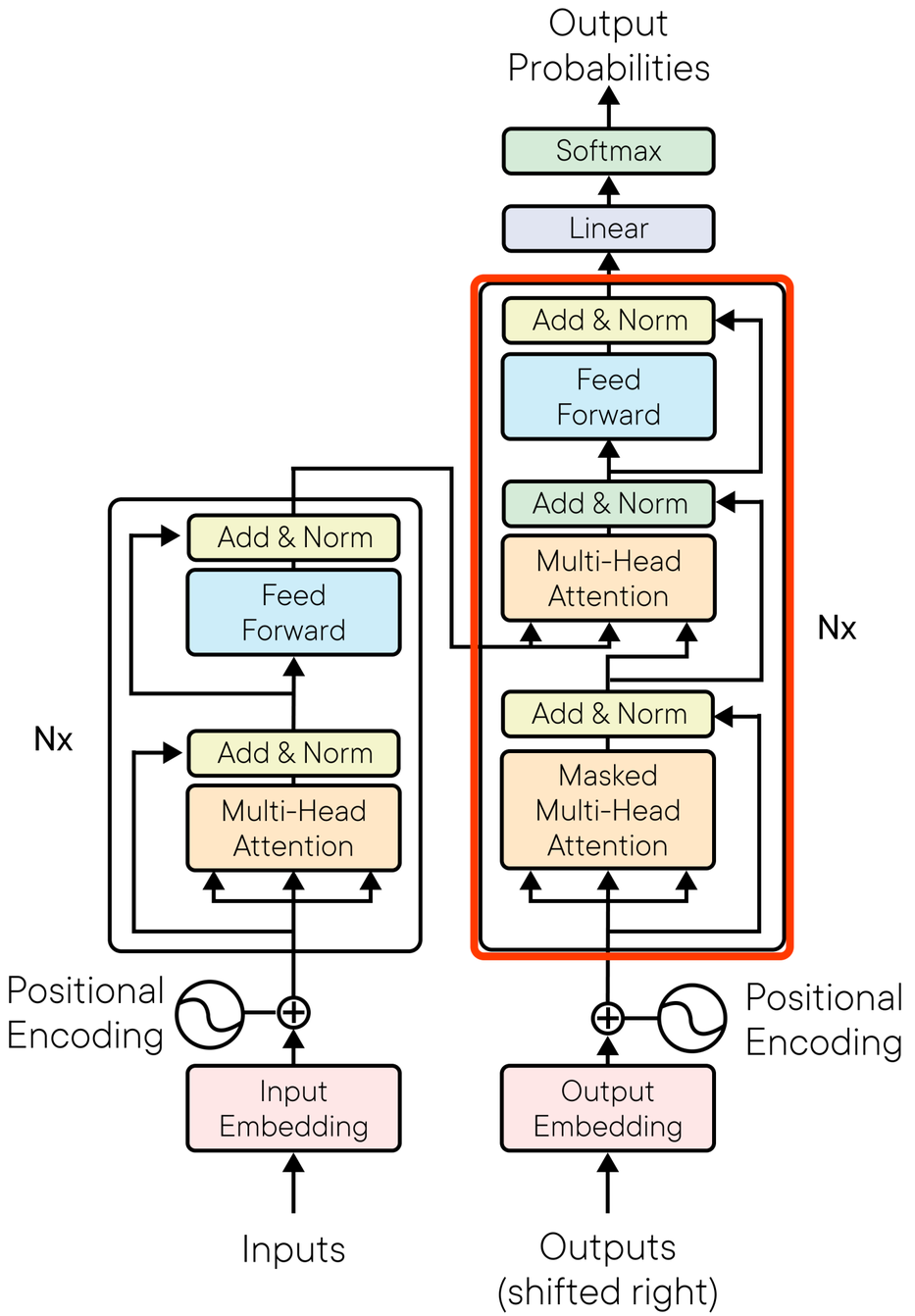

Nx
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Positional Encoding
Input Embedding
Positional Encoding
Output Embedding
Inputs
Outputs (shifted right)


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Why Do We Need It?

Transformers can parallelize computations across a GPU, which RNNs cannot

Transformers do not have the informational bottleneck

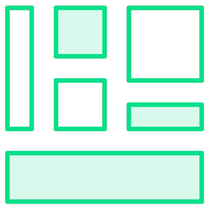
And Transformers have much fewer parameters for the same size of architecture than an RNN

Our First Transformer

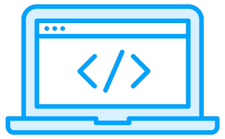
Introducing Hugging Face

Hugging Face is an open-source library and plataforma that provides a wide range of tolos and models for NLP tasks.

Features



Pretrained Models: Hugging Face provides a vast collection of pretrained models, including popular architectures like BERT, GPT and many more.



Transformers Library: The Transformers Library, developed by Hugging Face, is a go-to resource for NLP practitioners.

Datasets library: At your disposal, you have the datasets library to download tons of famous datasets along with their metrics to compare your models with the current state of the art.

How To Use

Index.py

```
from transformers import pipeline

sentiment_análisis = pipeline(“sentiment-analysis”)
result = sentiment_análisis(“I love Hugging Face!”)
print(result)
```

1. Download and load the tokenizer and model

Index.py

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Download and load the tokenizer and model
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

# Load the IMDb dataset from Hugging Face datasets library
dataset = load_dataset("imdb")

# Split the dataset into training and testing sets
train_dataset, test_dataset = train_test_split(dataset['train'], test_size=0.2,
random_state=42)

# Tokenize the training dataset
train_encodings = tokenizer(train_dataset['text'], truncation=True, padding=True)
train_labels = train_dataset['label']

# Tokenize the testing dataset
test_encodings = tokenizer(test_dataset['text'], truncation=True, padding=True)
test_labels = test_dataset['label']
```

2. Load the IMDb dataset from Hugging Face datasets library

Index.py

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Download and load the tokenizer and model
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

# Load the IMDb dataset from Hugging Face datasets library
dataset = load_dataset("imdb")

# Split the dataset into training and testing sets
train_dataset, test_dataset = train_test_split(dataset['train'], test_size=0.2,
random_state=42)

# Tokenize the training dataset
train_encodings = tokenizer(train_dataset['text'], truncation=True, padding=True)
train_labels = train_dataset['label']

# Tokenize the testing dataset
test_encodings = tokenizer(test_dataset['text'], truncation=True, padding=True)
test_labels = test_dataset['label']
```

3. Split the dataset into training and testing sets

Index.py

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Download and load the tokenizer and model
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

# Load the IMDb dataset from Hugging Face datasets library
dataset = load_dataset("imdb")

# Split the dataset into training and testing sets
train_dataset, test_dataset = train_test_split(dataset['train'], test_size=0.2,
random_state=42)

# Tokenize the training dataset
train_encodings = tokenizer(train_dataset['text'], truncation=True, padding=True)
train_labels = train_dataset['label']

# Tokenize the testing dataset
test_encodings = tokenizer(test_dataset['text'], truncation=True, padding=True)
test_labels = test_dataset['label']
```

4. Tokenize the datasets

Index.py

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Download and load the tokenizer and model
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

# Load the IMDb dataset from Hugging Face datasets library
dataset = load_dataset("imdb")

# Split the dataset into training and testing sets
train_dataset, test_dataset = train_test_split(dataset['train'], test_size=0.2,
random_state=42)

# Tokenize the training dataset
train_encodings = tokenizer(train_dataset['text'], truncation=True, padding=True)
train_labels = train_dataset['label']

# Tokenize the testing dataset
test_encodings = tokenizer(test_dataset['text'], truncation=True, padding=True)
test_labels = test_dataset['label']
```

Create TensorFlow datasets

Index.py

```
# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
train_labels)).batch(16)
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
test_labels)).batch(16)
```

```
# Configure training parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])
```

```
# Train the model
model.fit(train_dataset, epochs=3)
```

```
# Evaluate the model
results = model.evaluate(test_dataset)
print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

Configure training parameters

Index.py

```
# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
train_labels)).batch(16)
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
test_labels)).batch(16)

# Configure training parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])

# Train the model
model.fit(train_dataset, epochs=3)


# Evaluate the model
results = model.evaluate(test_dataset)
print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

Train the model

Index.py

```
# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
train_labels)).batch(16)
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
test_labels)).batch(16)


# Configure training parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])


# Train the model
model.fit(train_dataset, epochs=3)


# Evaluate the model
results = model.evaluate(test_dataset)
print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

Evaluate the model

Index.py

```
# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
train_labels)).batch(16)
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
test_labels)).batch(16)


# Configure training parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])


# Train the model
model.fit(train_dataset, epochs=3)


# Evaluate the model
results = model.evaluate(test_dataset)
print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

Transfer Learning on Transformers with Hugging Face

Text Summarisation with Fine-Tuned T5

Keys, Takeaways and Tips

Takeaways

The Transformer architecture solved all of the significant challenges we had with RNNs for sequence models

With Hugging Face, now it most straightforward tan ever just to graba checkpoint from someone and finetune it to your needs

Some models like T5 can perform on multiple task because that’s how they were trained and we can leverage it

Keys

Try to test around with another model of tasks and finetune it using Hugging Face, play a little

Practise using the original Transformer we created and try to Neural Machine Translation with it!

What Comes Next?

Finish the Path created here at Pluralsight on Large Language Models.

Do the course by Hugging Face theselves on how to use their tooling.

Check the book “Natural Language Processing with Transformers” by Lewis Tuntstall, Leandro von Werra and Thomas Wolf.