The ability to communicate with one another is a fundamental part of being human. There are nearly 7,000 different languages worldwide. As our world becomes increasingly connected, language translation provides a critical cultural and economic bridge between people from different countries and ethnic groups. Some of the more obvious use-cases include

In NMT, RNN is a stacked multi-layer of **Long short-term memory**(LSTMs). LSTM is a type of RNN which is capable of perform sequence-to-sequence transfer.

But there are problems with RNN. Different languages have different grammars. The sequence of words in a sentence might not match between languages. Additionally, RNN has difficulty to remember long sentences.

**MACHINE TRANSLATION**

Most of us if not all of us have used google translate or perhaps interacted with it.

When some your international friends send you a text, you use google translate.

You are in a foreign country and want to ask some thing you use google translate.

Google translate as a lot of applications but have you ever wondered how it works?

today I am going to take through how it works and also will be able to get our hands on real code to fine tune, test and deploy a translation model.

**LANGUAGE TRANSLATION**

How do we translate a sentence from on language to another language. To make things clear let us say we are translating from English to Swahili.

Our first trial is to take every word in the English sentence and find the corresponding translation in Swahili and spit it out and we repeat this for every word in the sentence.

It is a simple strategy and honestly, we don’t need machine learning for this because we can create a database of English Swahili translation and we are all set. For every English word look up in the database get the corresponding translation and repeat this for every word.

But there is a problem with this approach, every language has two components

1. Tokens – tokens are the smallest units of language
2. Grammar - it defines how this token should appear so that they make sense

In this context tokens are words and words are tokens.

“how are you” has four word tokens

Grammar is a set of rules that define the ordering for these words such as adjectives should follow nouns.

Syntax – does the structure of this sentence look correct

Semantics – does the sentence make sense in context

Grammar ensures syntax and semantics of sentence so that a sentence could make sense.

For a translation to make sense we need to incorporate grammar into our model. This is where Neural networks comes in.

Neural network in layman’s language is just a component that learns to solve a problem by looking at 100 of 1000 of examples, which allows the network to learn patterns in data

eventually it will be able to translate a given English sentence into Swahili all on its own.

This sounds interesting but what exactly is this network.

There are different neural networks used to solve different problems in this case we need a neural network to solve a problem of translation and since we are dealing with sentences we use a recurrent neural network

**How neural networks work**

In this case we need a neural network that solves the problem of translation some English sentence is the inputs and Swahili sentence is the output.

A recurrent neural network in simple terms is a neural network that learns to solve problems that involve sentences and since we are dealing with a problem of language translation which requires sentences the we use recurrent neural networks.

The first thing you notice is that the inputs and outputs are both sequence of words but computers don’t understand sentences like humans do so we need to convert them into a form the understand, numbers more specifically matrices and vectors.

This gives us the first part of our network which takes an English sentence and outputs a sequence of vectors that the computer can understand. This network is called a recurrent neural network.

Now that we have a vector, we need to convert it into a Swahili sentenced this vector to sentence mapping is done by another neural network and since we are dealing with sentence transformation, we use another recurrent Neural network

These two neural networks make up a basic structure of a basic translation model and it’s called the encoder decoder architecture. The first network encodes English sentences to vectors and the second networks decodes vectors into Swahili sentences.

But this architecture has a number of limitations the first on is the are slow because they are processed sequentially, we need the previous state to calculate the current state

They do not perform well with long sequences, this is because they cannot understand which part of an English sentence to translate and which part to ignore.

ATTENTION MECHANISM

It sits between decoder and encoder so during translation an English word is fed into the encoder, it is encoded into some vector which is just numbers the computer understands its basically the same English sentences in the computers eyes the we use an attention mechanism basically asking which Swahili word will be generated by which English word the decoder will then generate Swahili translation one word at a time focusing on the word determined by the attention mechanism.

CONCLUSION

So this is how popular machine translation models work the only difference is it uses up to 8 RNN instead of one this will help in solving more complex problems.

To recap on this you want to translate English to Swahili you pass the English texts word by word to the encoder and it converts this word in to vectors this vectors are then passed into an attention mechanism and this determines which English words to focus when translating into Swahili then this data is passed through a decoder which then generates Swahili sentence one word at a time. And that’s how a translation model works.

This allows computer to translate longer sentence that LSTM

**GRADIO**

It is python library which provides UI for your machine learning projects.

Gradio is the fastest way to demo your machine learning model with a friendly web interface so that anyone can use it, anywhere! With a few lines of code.

FINE TUNING

It the process that takes a model that has already been trained for a specific task and changing it to perform a second similar task.

This allows us to take advantage of what the model has already been trained on without developing it from scratch

NOTEBOOK

! pip install -q sacremoses ----- package is a Python library that provides a collection of tokenization and normalization tools for processing natural language text. It is specifically designed for handling text data in the context of natural language processing (NLP) and machine learning tasks.

Sacrebleu - is a Python library specifically used for evaluating the quality of machine translation outputs using the BLEU (Bilingual Evaluation Understudy) metric.

pip install -q evaluate -- Evaluate is a library that makes evaluating and comparing models and reporting their performance easier and more standardized.

!pip install -q transformers[sentencepiece] - SentencePiece is an unsupervised text tokenizer and detokenizer mainly for Neural Network-based text generation systems where the vocabulary size is predetermined prior to the neural model training.  a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input (which includes the recursive output) data.

Import string - allows you to create and customize your own string formatting behaviors using the same implementation as the built-in

Import warnings - lt controls whether warnings are ignored, displayed, or turned into errors

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. I

IMPORTING DATASET

**To fine-tune or train a translation model from scratch, we will need a dataset suitable for the task**

remove numbers

convert the data to lower case

remove apostrophes

convert back to csv

**preparing the dataset**

we load a dataset using load\_dataset function to load the dataset which it converts the csv file into a datasetdict object

We have 8492 pairs of sentences, but in one single split, so we will need to create our own validation set

We will use train\_test split method to split the dataset, we ll use 20% of the data for validation and the remaining 80% for training. The seed is used to provide reproducibility… ie you will get the same result if you are the model more than once

Now let us take a look on one of the elements in our data. We get a dictionary with two sentence pairs of languages we requested

**Load the pretrained model**

For this notebook we have picked a model from Helsinski NLP. This model is a sequence to sequence translation model which has already been trained on a large corpus of English and Swahili sentences.

We test the model by creating a translation pipeline the pass a sentence in English into the model. The model will give us this output tommorow ni sikukuu which is a Swahili translation but not very good. By the end of this fine tuning well try improving it.

Now that the datasets are ready we need to convert the in a way the computer understands this is done by tokenizing both the inputs and the targets

We will the test if the tokenozer process the target language In the output language.The tokenizer gives the output has inputIDS, attention mask and labels.

Since inputs is a dictionary with our usual keys (input IDs, attention mask, etc.), the last step is to define the preprocessing function we will apply on the datasets.

Note that we set the same maximum length for our inputs and outputs. Since the texts we’re dealing with seem pretty short, we use 128.

We can now apply that preprocessing in one go on all the splits of our dataset:

FINE TUNING WITH KERAS

First things first, we need an actual model to fine-tune. We’ll use the usual AutoModel API.

This model we are using was trained on a translation task and can be used already so well not have set any weights for the model

DATA COLLATOR

This meethos takes the tokenizer used to preprocess the inputs, but it also takes the model. This is because this data collator will also be responsible for preparing the decoder input IDs, which are shifted versions of the labels with a special token at the beginning. Since this shift is done slightly differently for different architectures, the DataCollatorForSeq2Seq needs to know the model object:

EVALUATION OF THE MODEL

The BLEU(Bilingual Evaluation Understusy) is a metric for evaluating machine translations score evaluates how close the translations are to their labels. 0 is the lowest score which means low quality translation while 100 is the highest score. With score of between 70 and 80 Very high quality, adequate, and fluent translations and often more accurate than humans

Fine tuning the translation model

# The number of training steps is the number of samples in the dataset, divided by the batch size then multiplied

# by the total number of epochs. Note that the tf\_train\_dataset here is a batched tf.data.Dataset,

# not the original Hugging Face Dataset, so its len() is already num\_samples // batch\_size.

Before we train we first need to login to hugging face.. this is not a compulsory step but it help us to save our model which we can share and retrieve any time. Note that you can specify the name of the repository you want to push to with the hub\_model\_id argument (in particular, you will have to use this argument to push to an organization)

Training

we define a PushToHubCallback to upload our model to the Hub during training, as we saw in [section 2](https://huggingface.co/learn/nlp-course/chapter7/(/course/chapter7/2)), and then we simply fit the model with that callback:

num\_epochs = 2: This line sets the number of epochs for training. An epoch refers to a complete iteration over the entire training dataset during model training. In this case, the model will be trained for 2 epochs.

num\_train\_steps = len(tf\_train\_dataset) \* num\_epochs: This line calculates the total number of training steps based on the number of epochs and the length of the training dataset (tf\_train\_dataset). A training step is a single update of the model's weights based on a batch of training data.

optimizer, schedule = create\_optimizer(init\_lr=5e-5, num\_warmup\_steps=0, num\_train\_steps=num\_train\_steps, weight\_decay\_rate=0.01): This line creates an optimizer and a learning rate schedule for model training. The create\_optimizer function is called with several parameters:

init\_lr=5e-5: This sets the initial learning rate for the optimizer to 5e-5, which is a commonly used learning rate value in machine learning models.

num\_warmup\_steps=0: This sets the number of warm-up steps for the learning rate schedule to 0, indicating that there will be no warm-up phase.

num\_train\_steps=num\_train\_steps: This sets the total number of training steps for the learning rate schedule to the value calculated earlier based on the number of epochs and the length of the training dataset.

weight\_decay\_rate=0.01: This sets the weight decay rate for the optimizer to 0.01, which is a regularization technique used to prevent overfitting in machine learning models.

The create\_optimizer function returns an optimizer and a learning rate schedule, which are stored in the variables optimizer and schedule, respectively.

model.compile(optimizer=optimizer): This line compiles the machine learning model with the optimizer that was created earlier. The optimizer parameter specifies the optimizer to be used during model training. Other parameters, such as loss function, metrics, etc., can also be specified during model compilation.

These lines of code set up the training parameters for the model, including the number of epochs, learning rate, optimizer, and weight decay, and compile the model for training using TensorFlow. Once these settings are in place, you can proceed with training the model using the compiled optimizer and other specified parameters.

Top of Form

**metrics after training**

Finally, let’s see what our metrics look like now that training has finished:

At this stage, you can use the inference widget on the Model Hub to test your model and share it with your friends. You have successfully fine-tuned a model on a translation task — congratulations!

**Using the pretrained model**

Let us use the same text we had already asked the model before translation

As expected, our pretrained model adapted its knowledge to the data we fine-tuned it on, and instead of leaving the English word “Tommorow” alone, it now translates it to the Swahili.

**Deployment**

There are several ways to deploy a machine learning model including flask, Streamlit and others we are looking at deploying the model using Streamlit because it gives more freedom to edit its iinterface.

But for now we have deployed using gradio which is a python ui for proving interfaces for machine learning projects using short lines of code

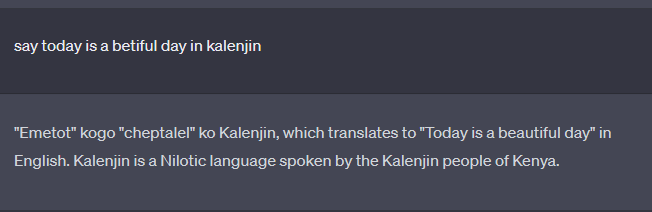
CHAT GPT UNDERSTANDS ONLY 95 LANGUAGES

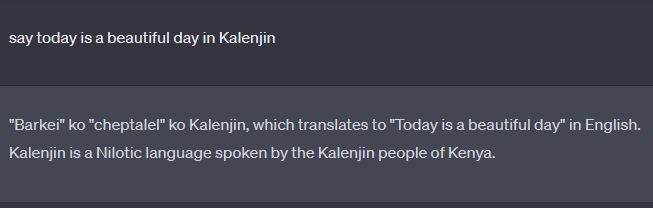
Africa – 2 (Hausa and Swahili)

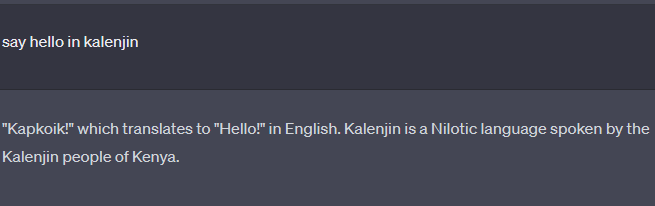
America - 3

Europe - 37

Asia - 40







Kikuyu native speakers is 6 million

Kalenjin native speakers is around 5 million

 93% of GPT-3’s pre-training data is English ([Brown et al., 2020](https://arxiv.org/pdf/2005.14165.pdf)).