HuggingFace Model for Text Classification

Sunday, September 25, 2022 8:18 PM

Initialization

[1] pip install datasets transformers

Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch.

Datasets is a lightweight library providing two main features:

- one-line dataloaders for many public datasets: one-liners to download and pre-process any of the number of datasets major public datasets (text datasets in 467 languages and dialects, image datasets, audio datasets, etc.) provided on the HuggingFace Datasets Hub. With a simple command like squad_dataset = load_dataset("squad"), get any of these datasets ready to use in a dataloader for training/evaluating a ML model (Numpy/Pandas/PyTorch/TensorFlow/JAX),
- efficient data pre-processing: simple, fast and reproducible data pre-processing for the above public datasets as well as your own local datasets in CSV/JSON/text/PNG/JPEG/etc. With simple commands like processed_dataset = dataset.map(process_example), efficiently prepare the dataset for inspection and ML model evaluation and training.

```
from huggingface_hub import notebook_login
notebook_login()
```

```
apt install git-lfs
```

An open source Git extension for versioning large files. **Git Large File Storage (LFS)** replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.

Importing Modules

```
[2] import datasets
import random
import pandas as pd
import numpy as np
from IPython.display import display, HTML
from datasets import load_dataset, load_metric
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer
```

Setting Up Variables

```
[3] task = "cola"
model_checkpoint = "distilbert-base-uncased"
batch_size = 16
```

Extracting Datasets and Metrics

```
[4] actual_task = "mnli" if task == "mnli-mm" else task
dataset = load_dataset("glue", actual_task)
metric = load_metric('glue', actual_task)
```

Preprocessing of Data

[5] tokenizer = AutoTokenizer.from_pretrained(model_checkpoint, use_fast = True)
We pass along use_fast=True to the call above to use one of the fast tokenizers (backed by Rust) from the 🤗 Tokenizers library. Those fast tokenizers are available for almost all models, but if you got an error with the previous call, remove that argument.

Generate a Lookup Table for Each GLUE Benchmarks

```
[6] task_to_keys = {
   "cola": ("sentence", None),
   "mnli": ("premise", "hypothesis"),
   "mnli-mm": ("premise", "hypothesis"),
   "mrpc": ("sentence1", "sentence2"),
   "qnli": ("question", "sentence"),
   "qqp": ("question1", "question2"),
   "rte": ("sentence1", "sentence2"),
   "sst2": ("sentence", None),
   "stsb": ("sentence1", "sentence2"),
   "wnli": ("sentence1", "sentence2"),
}
```

Assignment and Validation of Lookup Table

```
[7] sentence1_key, sentence2_key = task_to_keys[task]
if sentence2_key is None:
   print(f"Sentence: {dataset['train'][0][sentence1_key]}")
else:
   print(f"Sentence 1: {dataset['train'][0][sentence1_key]}")
   print(f"Sentence 2: {dataset['train'][0][sentence2_key]}")
```

Create a function to preprocess token/s

```
[8] def preprocess_function(examples):
   if sentence2_key is None:
      return tokenizer(examples[sentence1_key], truncation=True)
   return tokenizer(examples[sentence1_key], examples[sentence2_key], truncation=True)
```

Preprocess all sentences (or pairs of sentences) in our dataset

```
[9] encoded_dataset = dataset.map(preprocess_function, batched=True)
```

Fine-Tuning the Model

```
[10] num_labels = 3 if task.startswith("mnli") else 1 if task=="stsb" else 2
model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint,
num_labels=num_labels)
```

Save the checkpoints of the model

```
[11] metric_name = "pearson" if task == "stsb" else "matthews_correlation" if task == "cola" else
"accuracy"
model_name = model_checkpoint.split("/")[-1]

args = TrainingArguments(
```

```
    f"{model_name}-finetuned-{task}",
    evaluation_strategy = "epoch",
    save_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=5,
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model=metric_name,
    push_to_hub=True,
)
```

Here we set the evaluation to be done at the end of each epoch, tweak the learning rate, use the batch_size defined at the top of the notebook and customize the number of epochs for training, as well as the weight decay. Since the best model might not be the one at the end of training, we ask the Trainer to load the best model it saved (according to metric_name) at the end of training.

Define a compute function using metrics

```
[12] def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    if task != "stsb":
        predictions = np.argmax(predictions, axis=1)
    else:
        predictions = predictions[:, 0]
    return metric.compute(predictions=predictions, references=labels)
```

Pass all the declared variables and predefined functions to our trainer

```
[13] validation_key = "validation_mismatched" if task == "mnli-mm" else "validation_matched" if task ==
"mnli" else "validation"

trainer = Trainer(
    model,
    args,
    train_dataset=encoded_dataset["train"],
    eval_dataset=encoded_dataset[validation_key],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

Fine Tune Model

```
[14] trainer.train()
```

The training loss indicates how well the model is fitting the training data, while the validation loss indicates how well the model fits new data.

Validation of Trainer Model

```
[15] trainer.evaluate()
```

<u>Efficiently Searching for Hyperparameters</u>

Install optuna or Ray Tune modules that supports hyperparameter search of Trainer

[1] pip install optuna
pip install ray[tune]

Losses

Sunday, September 25, 2022 9:34 PM

Deep learning is a branch of machine learning that comprises the use of artificial neural networks. In particular, deep learning algorithms allow computer programs to learn and discover patterns in massive amounts of data.

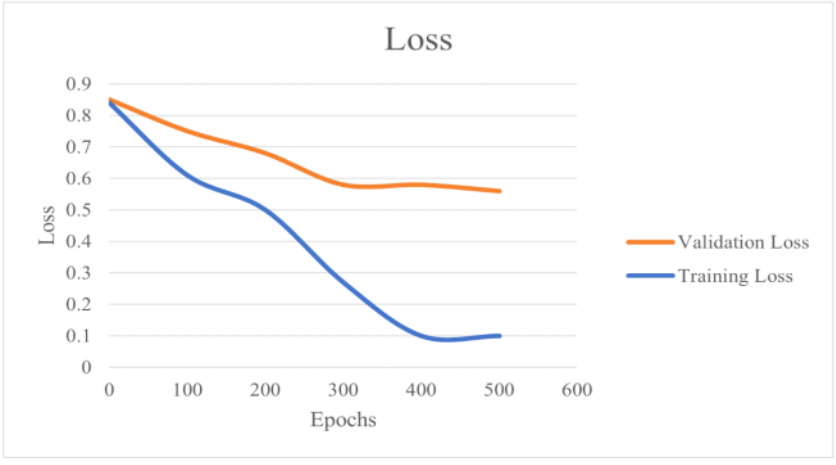
Artificial neural networks are algorithms that are inspired by the workings of the biological neural network in living organisms. An artificial neural network is usually composed of a system of interconnected nodes and weights. As such, the input signals are first passed through nodes known as neurons. Next, these neurons are activated with a function and multiplied with weights to produce an output signal.

Consequently, when we employ deep learning algorithms on a specific dataset, we yield a model that can take in some input and produce an output. Now, to assess the performance of a model, we use a measure known as loss. Specifically, this loss quantifies the error produced by the model.

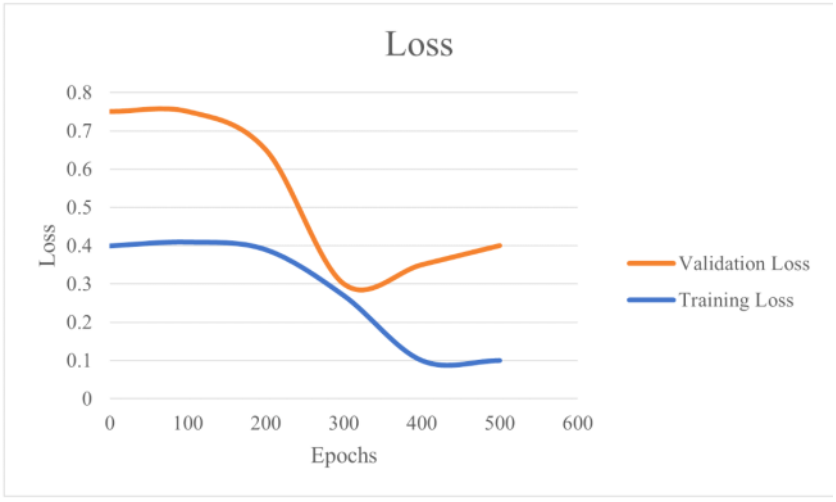
A high loss value usually means the model is producing erroneous output, while a low loss value indicates that there are fewer errors in the model. In addition, the loss is usually calculated using a cost function, which measures the error in different ways. The cost function chosen is usually dependent on the problem being solved and the data being used. For example, cross-entropy is usually used for binary classification problems.

In most deep learning projects, the training and validation loss is usually visualized together on a graph. The purpose of this is to diagnose the model's performance and identify which aspects need tuning. To explain this section, we'll use three different scenarios.

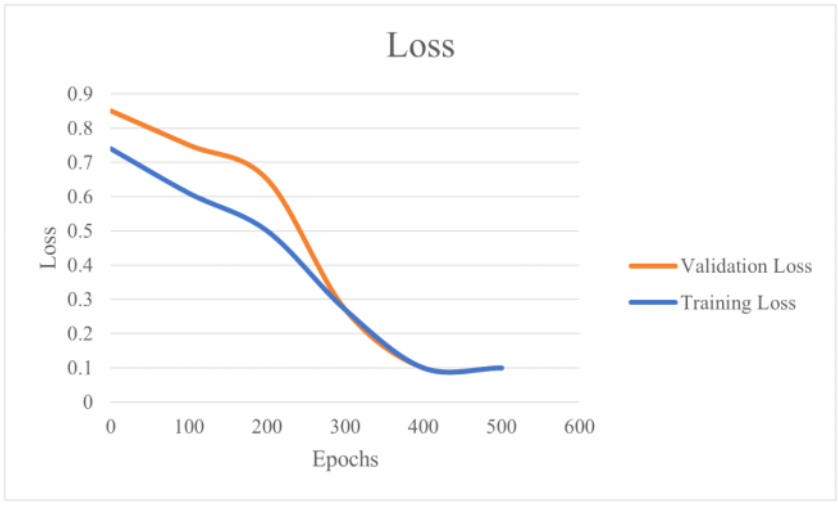
Underfitting



Overfitting


Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Loss
0
100
200
300
400
500
600
Epochs
Validation Loss
Training Loss

Good Fit


Loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Loss
0
100
200
300
400
500
600
Epochs
Validation Loss
Training Loss

Training Loss

Sunday, September 25, 2022 9:35 PM

The training loss is a metric used to assess how a deep learning model fits the training data. That is to say, it assesses the error of the model on the training set. Note that, the training set is a portion of a dataset used to initially train the model. Computationally, the training loss is calculated by taking the sum of errors for each example in the training set.

It is also important to note that the training loss is measured after each batch. This is usually visualized by plotting a curve of the training loss.

Validation Loss

Sunday, September 25, 2022 9:35 PM

On the contrary, validation loss is a metric used to assess the performance of a deep learning model on the validation set. The validation set is a portion of the dataset set aside to validate the performance of the model. The validation loss is similar to the training loss and is calculated from a sum of the errors for each example in the validation set.

Additionally, the validation loss is measured after each epoch. This informs us as to whether the model needs further tuning or adjustments or not. To do this, we usually plot a learning curve for the validation loss

GLUE Benchmarks

Sunday, September 25, 2022 1:37 AM

The General Language Understanding Evaluation (GLUE) benchmark is a collection of resources for training, evaluating, and analyzing natural language understanding systems. GLUE consists of:

- A benchmark of nine sentence- or sentence-pair language understanding tasks built on established existing datasets and selected to cover a diverse range of dataset sizes, text genres, and degrees of difficulty,
- A diagnostic dataset designed to evaluate and analyze model performance with respect to a wide range of linguistic phenomena found in natural language, and
- A public leaderboard for tracking performance on the benchmark and a dashboard for visualizing the performance of models on the diagnostic set.

The GLUE Benchmark is a group of nine classification tasks on sentences or pairs of sentences which are:

- CoLA (Corpus of Linguistic Acceptability)
  - Determine if a sentence is grammatically correct or not. Is a dataset containing sentences labeled grammatically correct or not.
- MNLI (Multi-Genre Natural Language Inference)
  - Determine if a sentence entails, contradicts or is unrelated to a given hypothesis. (This dataset has two versions, one with the validation and test set coming from the same distribution, another called mismatched where the validation and test use out-of-domain data.)
- MRPC (Microsoft Research Paraphrase Corpus)
  - Determine if two sentences are paraphrases from one another or not.
- QNLI (Question-answering Natural Language Inference)
  - Determine if the answer to a question is in the second sentence or not. (This dataset is built from the SQuAD dataset.)
- QQP (Quora Question Pairs2)
  - Determine if two questions are semantically equivalent or not.
- RTE (Recognizing Textual Entailment)
  - Determine if a sentence entails a given hypothesis or not.
- SST-2 (Stanford Sentiment Treebank)
  - Determine if the sentence has a positive or negative sentiment.
- STS-B (Semantic Textual Similarity Benchmark)
  - Determine the similarity of two sentences with a score from 1 to 5.
- WNLI (Winograd Natural Language Inference)
  - Determine if a sentence with an anonymous pronoun and a sentence with this pronoun replaced are entailed or not. (This dataset is built from the Winograd Schema Challenge dataset.)

CoLA

Sunday, September 25, 2022 1:35 AM

CoLA (Corpus of Linguistic Acceptability)

- Determines if a sentence is grammatically correct or not

Data Format

Column 1	Column 2	Column 3	Column 4
the code representing the source of the sentence	the acceptability judgement label 0 - unacceptable 1 - acceptable	the acceptability judgement as originally notated by the author	the sentence

Processing

During gathering of the data and processing, some sentences from the source documents may have been omitted or altered. We retained all acceptable examples, and excluded any examples given intermediate judgments such as “?” or “#”. In addition, we excluded examples of unacceptable sentences not suitable for the present task because they required reasoning about pragmatic violations, unavailable semantic readings, or nonexistent words. We take responsibility for any errors.

MNLI

Sunday, September 25, 2022 8:01 PM

Models thus can be evaluated on both the matched test examples, which are derived from the same sources as those in the training set, and on the mismatched examples, which do not closely resemble any of those
seen at training time.

This task will involve reading a line from a non-fiction article and writing three sentences that relate to it. The line will describe a situation or event. Using only this description and what you know about the world:
• Write one sentence that is definitely correct about the situation or event in the line.
• Write one sentence that might be correct about the situation or event in the line.
• Write one sentence that is definitely incorrect about the situation or event in the line.

Data Collection

The data collection methodology for MultiNLI is similar to that of SNLI: We create each sentence pair by selecting a premise sentence from a preexisting text source and asking a human annotator to compose a novel sentence to pair with it as a hypothesis. This section discusses the sources of our premise sentences, our collection method for hypotheses, and our validation (relabeling) strategy

Hypothesis Collection

To collect a sentence pair, we present a crowdworker with a sentence from a source text and ask them to compose
three novel sentences (the hypotheses):

- one which is necessarily true or appropriate whenever the premise is true (paired with the premise and labeled ENTAILMENT),
- one which is necessarily false or inappropriate whenever the premise is true (CONTRADICTION), and
- one where neither condition applies (NEUTRAL).

This method of data collection ensures that the three classes will be represented equally in the raw corpus

There are five unique prompts in total:

- one for written non-fiction genres (SLATE, OUP, GOVERNMENT, VERBATIM, TRAVEL; Figure 1),
- one for spoken genres (TELEPHONE, FACE-TO-FACE),
- one for each of the less formal written genres (FICTION, LETTERS), and
- a specialized one for 9/11, tailored to fit its potentially emotional content.

Each prompt is accompanied by example premises and hypothesis that are specific to each genre.

Validation

We perform an additional round of annotation on test and development examples to ensure accurate labelling. Workers are presented with pairs of sentences and asked to supply a single label (ENTAILMENT, CONTRADICTION, NEUTRAL) for the pair. Each pair is relabeled by four workers, yielding a total of five labels per example.

Metrics

Sunday, September 25, 2022 7:32 PM

for CoLA: Matthews Correlation Coefficient
for MNLI (matched or mismatched): Accuracy
for MRPC: Accuracy and F1 score
for QNLI: Accuracy
for QQP: Accuracy and F1 score
for RTE: Accuracy
for SST-2: Accuracy
for STS-B: Pearson Correlation Coefficient and Spearman's_Rank_Correlation_Coefficient
for WNLI: Accuracy

Matthew's Correlation Coefficient

Sunday, September 25, 2022 7:33 PM

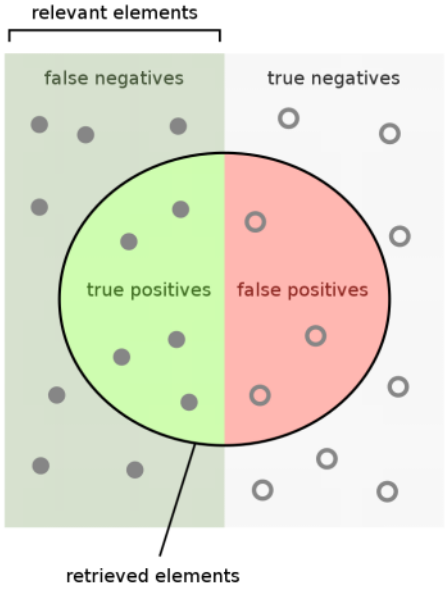
The MCC is defined identically to phi coefficient, introduced by Karl Pearson, also known as the Yule phi coefficient from its introduction by Udny Yule in 1912. Despite these antecedents which predate Matthews's use by several decades, the term MCC is widely used in the field of bioinformatics and machine learning. The coefficient takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. The MCC is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation. However, if MCC equals neither -1, 0, or + 1, it is not a reliable indicator of how similar a predictor is to random guessing because MCC is dependent on the dataset.

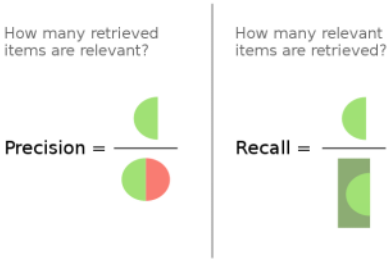
The MCC can be calculated directly from the confusion matrix using the formula:

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

F-Score or F-Measure

Sunday, September 25, 2022 7:32 PM





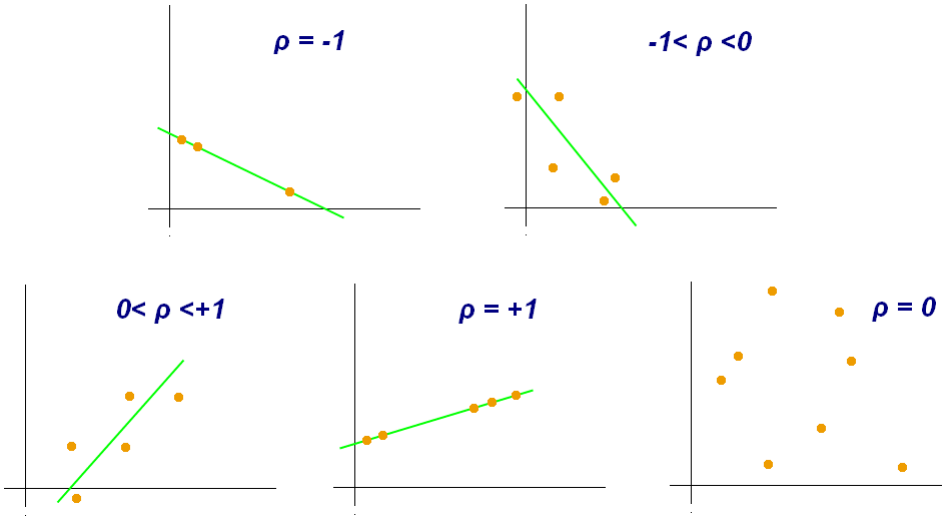
The traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{2\mathrm{tp}}{2\mathrm{tp} + \mathrm{fp} + \mathrm{fn}}.$$

Pearson correlation coefficient

Sunday, September 25, 2022 7:35 PM

is a measure of linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations; thus, it is essentially a normalized measurement of the covariance, such that the result always has a value between −1 and 1. As with covariance itself, the measure can only reflect a linear correlation of variables, and ignores many other types of relationships or correlations. As a simple example, one would expect the age and height of a sample of teenagers from a high school to have a Pearson correlation coefficient significantly greater than 0, but less than 1 (as 1 would represent an unrealistically perfect correlation).



Pearson's correlation coefficient, when applied to a population, is commonly represented by the Greek letter ρ (rho) and may be referred to as the population correlation coefficient or the population Pearson correlation coefficient. Given a pair of random variables (X,Y), the formula for ρ is:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where:

- cov is the covariance
- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y

Model Checkpoints

Sunday, September 25, 2022 2:10 AM

distilbert-base-uncased
Bidirectional Encoder Representations from Transformers (BERT)

Model description

DistilBERT is a transformers model, smaller and faster than BERT, which was pretrained on the same corpus in a self-supervised fashion, using the BERT base model as a teacher. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts using the BERT base model. More precisely, it was pretrained with three objectives:

- Distillation loss: the model was trained to return the same probabilities as the BERT base model.
- Masked language modeling (MLM): this is part of the original training loss of the BERT base model. When taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked sentence through the model and has to predict the masked words. This is different from traditional recurrent neural networks (RNNs) that usually see the words one after the other, or from autoregressive models like GPT which internally mask the future tokens. It allows the model to learn a bidirectional representation of the sentence.
- Cosine embedding loss: the model was also trained to generate hidden states as close as possible as the BERT base model.

This way, the model learns the same inner representation of the English language than its teacher model, while being faster for inference or downstream tasks.

Intended uses & limitations

You can use the raw model for either masked language modeling or next sentence prediction, but it's mostly intended to be fine-tuned on a downstream task. See the model hub to look for fine-tuned versions on a task that interests you.

Note that this model is primarily aimed at being fine-tuned on tasks that use the whole sentence (potentially masked) to make decisions, such as sequence classification, token classification or question answering. For tasks such as text generation you should look at model like GPT2.

Transformers

Monday, September 26, 2022 2:27 AM

Encoder Decoder Framework

Monday, September 26, 2022 2:27 AM

The Encoder-Decoder Framework

Prior to transformers, recurrent architectures such as LSTMs were the state of the art in NLP. These architectures contain a feedback loop in the network connections that allows information to propagate from one step to another, making them ideal for modeling sequential data like text. As illustrated on the left side of Figure 1-2, an RNN receives some input (which could be a word or character), feeds it through the network, and outputs a vector called the hidden state. At the same time, the model feeds some information back to itself through the feedback loop, which it can then use in the next step. This can be more clearly seen if we "unroll" the loop as shown on the right side of Figure 1-2: the RNN passes information about its state at each step to the next operation in the sequence. This allows an RNN to keep track of information from previous steps, and use it for its output predictions.

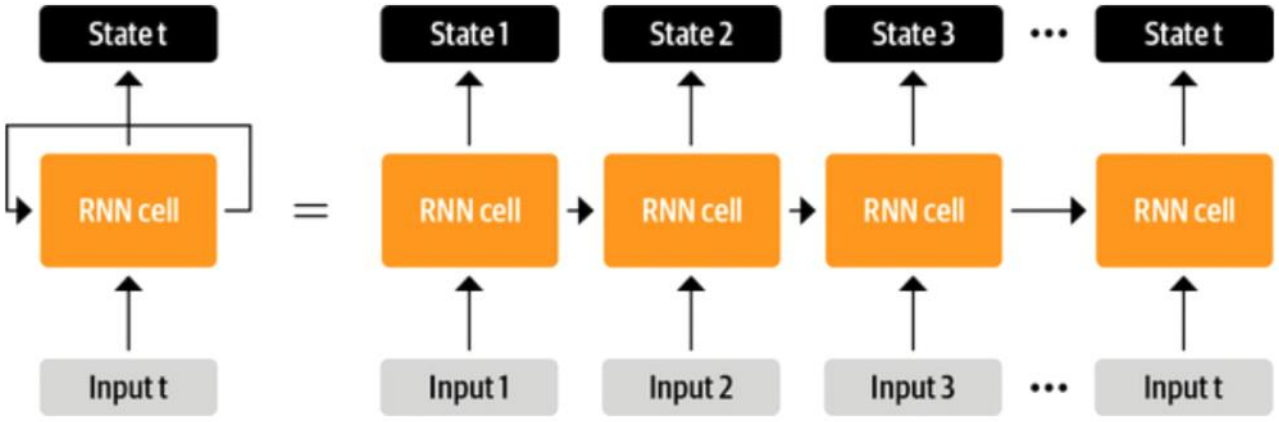


Figure 1-2. Unrolling an RNN in time

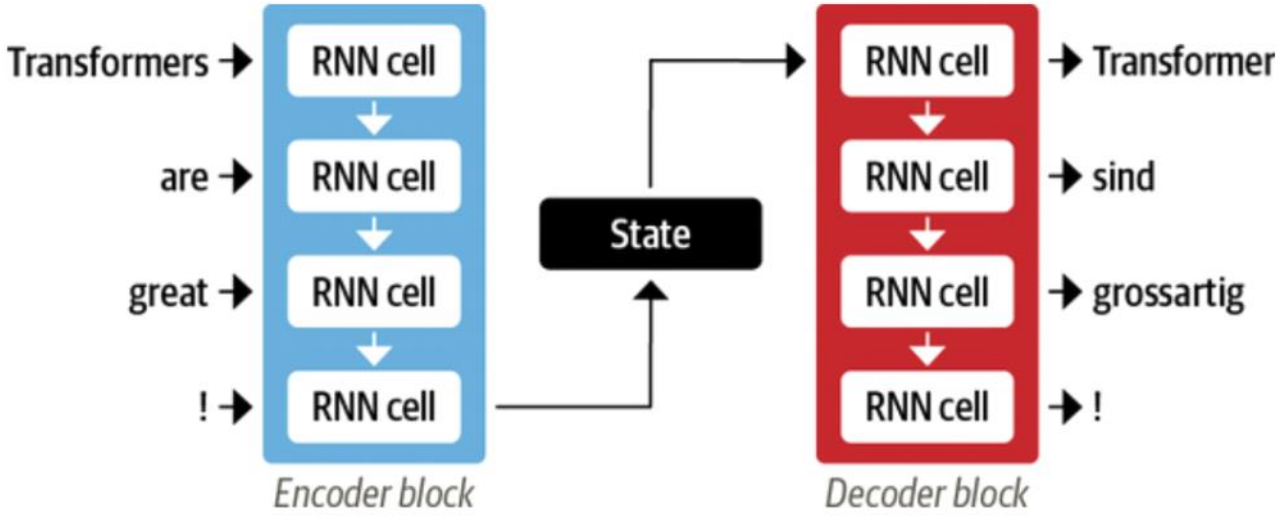


Figure 1-3. An encoder-decoder architecture with a pair of RNNs (in general, there are many more recurrent layers than those shown here)

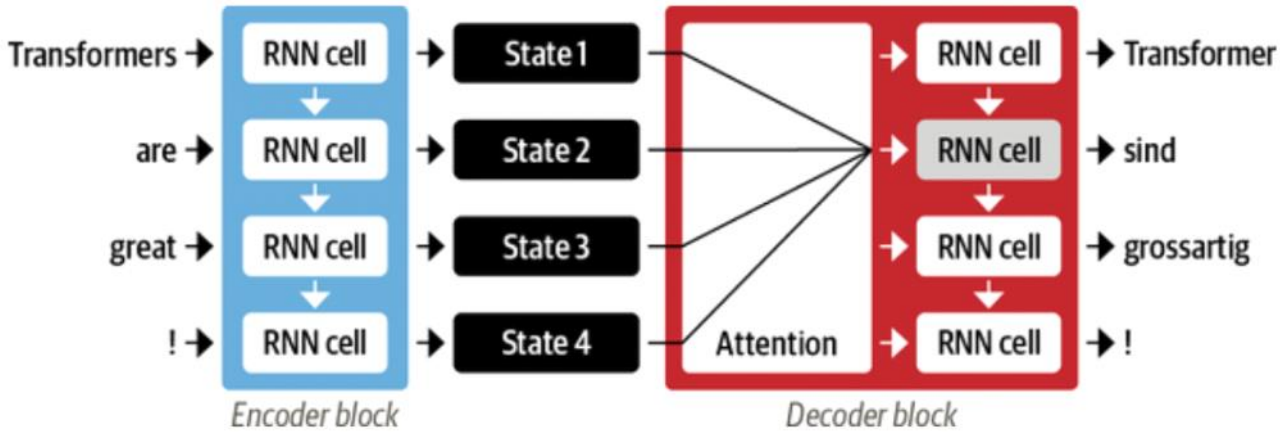


Figure 1-4. An encoder-decoder architecture with an attention mechanism for a pair of RNNs

With the transformer, a new modeling paradigm was introduced: dispense with recurrence altogether, and instead rely entirely on a special form of attention called self-attention. We'll cover self-attention in more detail in Chapter 3, but the basic idea is to allow attention to operate on all the states in the same layer of the neural network. This is shown in Figure 1-6, where both the encoder and the decoder have their own self-attention mechanisms, whose outputs are fed to feed-forward neural networks (FF NNs). This architecture can be trained much faster than recurrent models and paved the way for many of the recent breakthroughs in NLP.

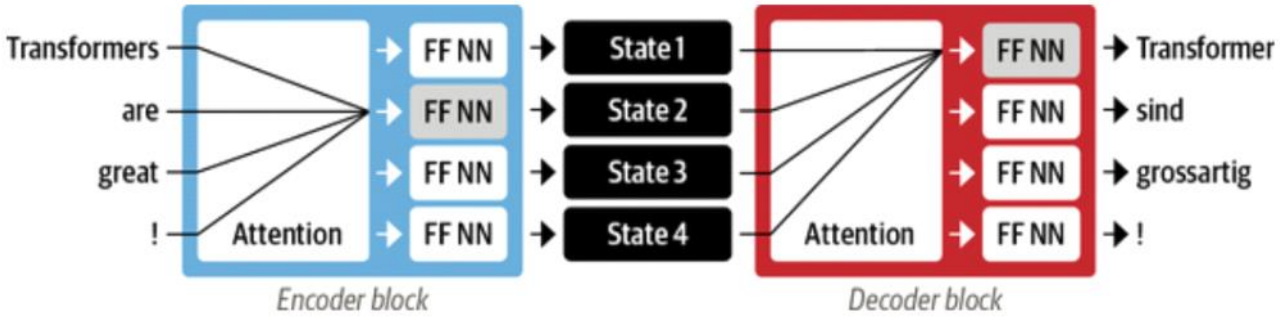


Figure 1-6. Encoder-decoder architecture of the original Transformer

Transfer Learning

Monday, September 26, 2022 2:35 AM

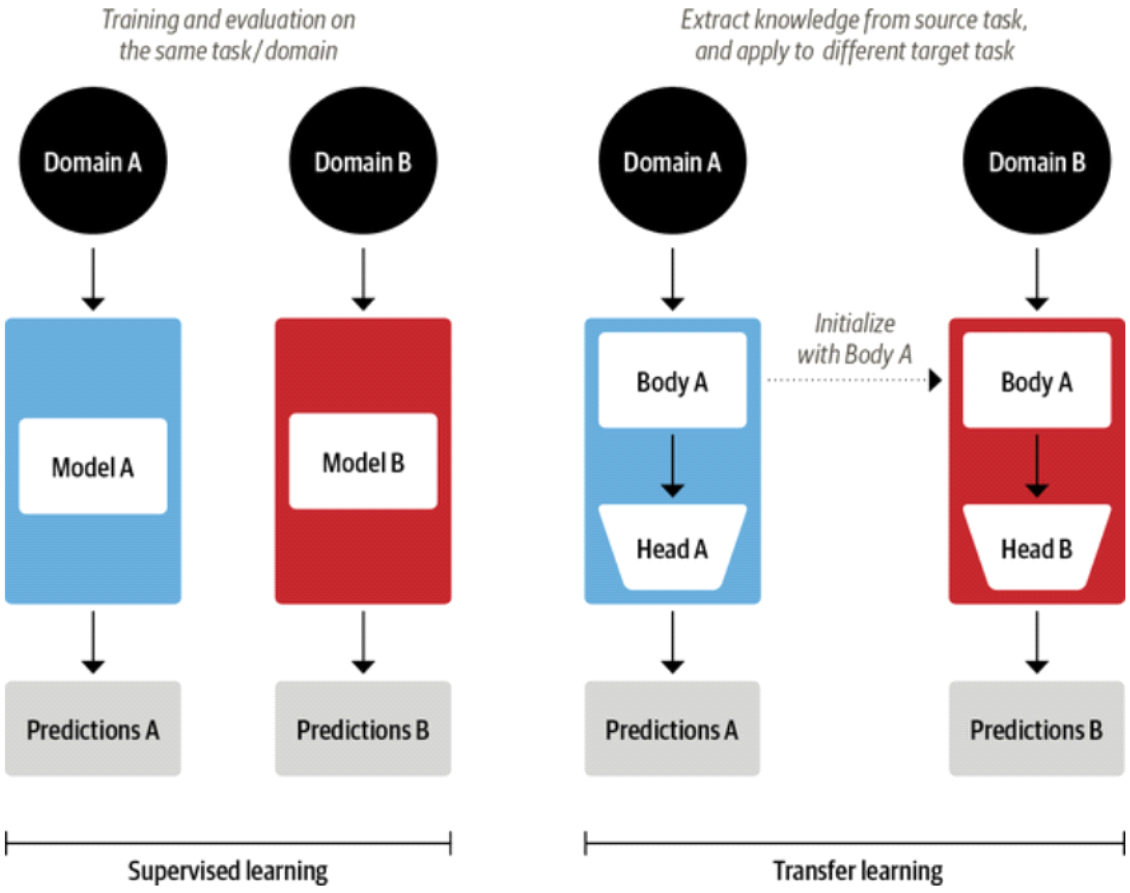


Figure 1-7. Comparison of traditional supervised learning (left) and transfer learning (right)

In computer vision, the models are first trained on large-scale datasets such as ImageNet, which contain millions of images. This process is called pretraining and its main purpose is to teach the models the basic features of images, such as edges or colors. These pretrained models can then be fine-tuned on a downstream task such as classifying flower species with a relatively small number of labeled examples (usually a few hundred per class). Fine-tuned models typically achieve a higher accuracy than supervised models trained from scratch on the same amount of labeled data.

Tokenizer

Monday, September 26, 2022 2:59 AM

Special Token	[PAD]	[UNK]	[CLS]	[SEP]	[MASK]
Special Token ID	0	100	101	102	103