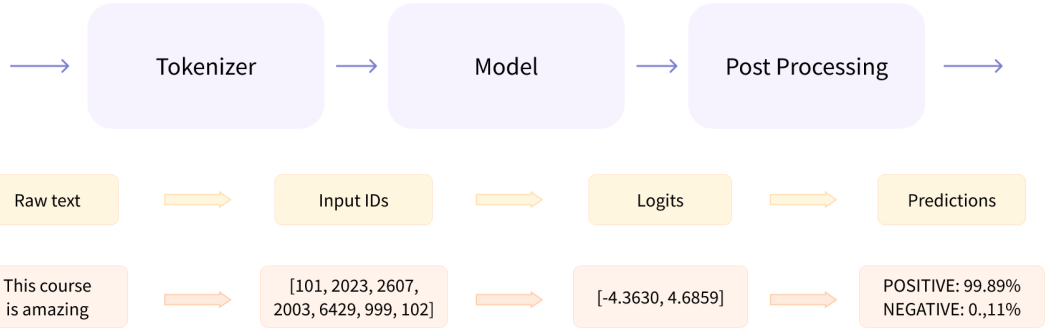
In [ ]:
```
from transformers import pipeline

classifier = pipeline("sentiment-analysis",
                      model="distilbert-base-uncased-finetuned-sst-2-english")
classifier(
    [
        "I've been waiting for a HuggingFace course my whole life.",
        "I hate this so much!",
    ]
)
```

Out[ ]:
```
[{'label': 'POSITIVE', 'score': 0.9598050713539124},
 {'label': 'NEGATIVE', 'score': 0.9994558691978455}]
```

Yukarıda kullanılan fonksiyon, 3 işlemi yürütür:

- Preprocessing
- Inputları modele verme
- Postprocessing.



Preprocessing with Tokenizer

Sürecin ilk aşamasındır. Yapay sinir ağlarına benzer olarak, metinler direkt olarak işleme alınamaz. Bu yüzden, pipeline'ın ilk aşaması verilen metin gruplarını sayılara çevirmektir. Tokenizer'in görevleri şunlardır:

- Verilen cümleyi kelimelere, alt kelimelere, harflere veya sembollere bölmek.
- Elde edilen *token*'lara tamsayı atamak.
- Faydalı olabilecek inputları eklemek (cümle sonunu ve başını anlatan token'lar gibi).

Model pretrain edildiğinde aynı süreçler gerçekleştirilmiştir. Bu yüzden pretrain edilmiş modelin bilgilerini Model Hub üzerinden indirebiliriz. Bu iş için `AutoTokenizer` sınıfından `from_pretrained()` metodunu kullanabiliriz. Modelin checkpoint ismini kullanarak bu metodu kullanırsak, modelin tokenizer'ı bilgisayara inmeye başlayacaktır.

`sentiment-analysis` için default checkpoint ismi `distilbert-base-uncased-finetuned-sst-2-english` dir.

In [ ]:
```
from transformers import AutoTokenizer

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

Yukarıdaki işlem tamamlandıktan sonra cümleyi direkt olarak tokenizer'a verebiliriz. Bu aşamadan sonra yapılması gereken şey cümlenin tamsayılardan oluşan ifadesini tensor formuna sokmaktır.

Huggingface Transformer modellerini kullanırken backend olarak hangi ML framework'u kullanacağınızı düşünmenize gerek yoktur. Ancak Transformer modelleri input olarak *tensor* kabul eder. Dönecek tensor tipini belirtmek için `return_tensors` argumanı kullanılabilir. Örne kullanım aşağıdaki gibidir:

In [ ]:
```
raw_inputs = [
    "I've been waiting for a HuggingFace course my whole life.",
    "I hate this so much!",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)
```

```
{'input_ids': tensor([[  101,  1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12
172,
          2607,  2026,  2878,  2166,  1012,   102],
        [  101,  1045,  5223,  2023,  2061,  2172,   999,   102,     0,     0,
             0,     0,     0,     0,     0,     0]]), 'attention_mask': tensor([[1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]])}
```

Padding ve Truncation argümanları ilerleyen aşamalarda anlatılacaktır. Hatırlamanız gereken şeyler:

- Tokenizer'a cümle veya cümle listesi verilebilir.
- Tokenizer'ın return ettiği tensor tipini belirtebilirsiniz. Aksi takdirde default olan list of lists döndürülecektir.

Yukarıda elde edilen çıktı aslında bir dictionary'dir ve iki adet key değeri vardır: `input_ids` ve `attention_mask`.

`input_ids` iki adet integer satırından oluşur (iki cümle için iki satır) ve bu satırların elemanları, cümledeki tokenleri tanımlayan eşsiz sayılardır.

Model

Tokenizer'ı indirdiğimiz gibi pretrained modeli de indirebiliriz. Bunun için `AutoModel` sınıfından `from_pretrained()` metodunu kullanabiliriz.

In [ ]:
```
from transformers import AutoModel

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModel.from_pretrained(checkpoint)
```

```
Some weights of the model checkpoint at distilbert-base-uncased-finetuned-sst-2-english
were not used when initializing DistilBertModel: ['pre_classifier.weight', 'pre_classifi
er.bias', 'classifier.bias', 'classifier.weight']
- This IS expected if you are initializing DistilBertModel from the checkpoint of a mode
l trained on another task or with another architecture (e.g. initializing a BertForSeque
nceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing DistilBertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a BertForSequenceClassificat
ion model from a BertForSequenceClassification model).
```

Yukarıdaki kod parçası, daha önceden pipeline fonksiyonu ile indirdiğimiz checkpoint'i yükleyecektir ve bir model oluşturacaktır.

Bu mimari sadece base Transformer modülünü içerir: Verilen girdiler için *hidden state* veya *features* denilen çıktılar üretir, yani her model input'u için high-dimensional vektör ifadesi döndürülür. Bu vektör, verilen girdinin içindeki bağlamsal anlayışı'ı (contextual understanding) içerir.

Bu hidden state'ler kendi başlarına da anlamlı olmasına rağmen genelde *head* olarak bilinen modellerin input'larıdır. Bu head'lar tasklara göre değişir.

A high-dimensional vector?

Transformer tarafından üretilen vektör genelde oldukça büyüktür ve üç boyutludur.

- Batch size: Tek seferde işlenen sequence sayısı, yukarıdaki örnek için 2'dir.
- Sequence length: Cümlenin nümerik ifadesinin uzunluğu, yukarıdaki örnek için 16'dır.
- Hidden size: The vector dimension of each model input.

Hidden size boyutu küçük modeller için 768 iken 3072'ye kadar büyüyebilir. Modelin çıktısını görmek için aşağıdaki blok çalıştırılabilir. `outputs` değişkeni namedtuple ya da dictionary olarak davranabilir.

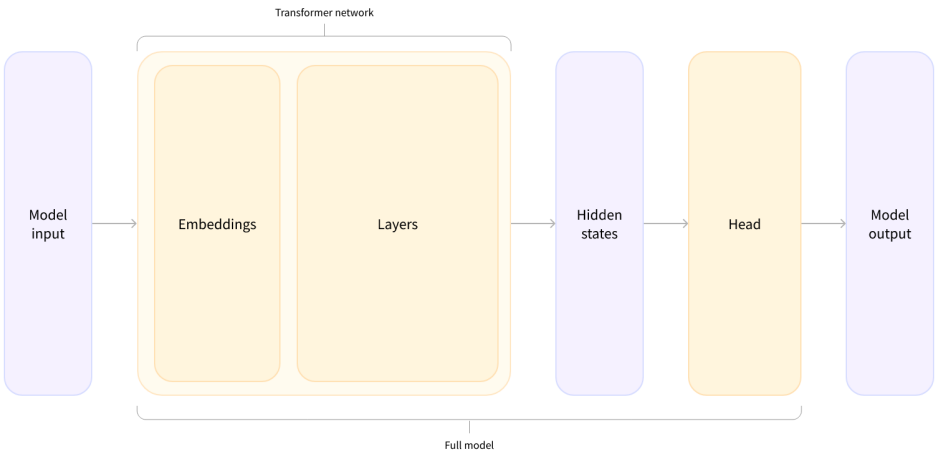
In [ ]:
```
outputs = model(**inputs)
print(outputs.last_hidden_state.shape)
```

```
torch.Size([2, 16, 768])
```

Model heads: Making sense out of numbers

Model heads girdi olarak hidden state içindeki yüksek boyutlu vektörü alır ve farklı bir boyuta "project" eder.

Genelde bir ya da birkaç linear katmandan oluşturulur. Transformer modelin çıktısı, işlenmek için direkt olarak model head'a gönderilir.



Görseldeki model, embedding layer ve subsequent layers olarak ifade edilmiştir. Embedding layer input ID'leri vektöre çevirir. Subsequent layer'lar bu vektörleri attention mechanism ile manipüle ederek final cümlenin representation'unu oluşturur.

Huggingface Transformers'ta farklı mimariler de mevcuttur ve her biri farklı bir iş için tasarlanmıştır. Başlıca mimariler aşağıdaki gibidir:

- `Model (retrieve the hidden states)`
- `ForCausalLM`
- `ForMaskedLM`
- `ForMultipleChoice`
- `ForQuestionAnswering`
- `ForSequenceClassification`
- `ForTokenClassification`

Örneğin devamında bir cümleyi pozitif ya da negatif olarak sınıflandırmak için sequence sınıflandırma head'ı kullanacağız. Bu yüzden AutoModel yerine `AutoModelForSequenceClassification` sınıfı kullanabiliriz.

In [ ]:
```
from transformers import AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
outputs = model(**inputs)
```

Bu noktada output'un boyutunun çok daha küçük olduğunu görebiliriz. Head yüksek boyutlu vektörleri alıp iki değer içeren bir vektör döndürür. Sadece iki cümlemiz ve iki tip etiketimiz olduğu için boyut 2x2 olacaktır.

In [ ]:
```
print(outputs.logits.shape)
```

```
torch.Size([2, 2])
```

Postprocessing the output

Modelden aldığımız çıktıların bir anlam ifade etmesi gerekmez. Örneğin:

In [ ]:
```
print(outputs.logits)
```

```
tensor([[-1.5607,  1.6123],
        [ 4.1692, -3.3464]], grad_fn=<AddmmBackward0>)
```

İlk cümle için [-1.5607, 1.6123] tahmin edilmişken, ikinci cümle için [4.1692, -3.3464] tahmin edilmiştir. Fakat bu sayılar olasılık değil logit'tir(Modelin son katmanı tarafından normalize edilmemiş skorlardır). Olasılığa çevrilebilmeleri için SoftMax katmanından geçmeleri gerekir. Transformer modeller output olarak logit üretir.

In [ ]:
```
import torch

predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
print(predictions)
```

```
tensor([[4.0195e-02, 9.5981e-01],
        [9.9946e-01, 5.4418e-04]], grad_fn=<SoftmaxBackward0>)
```

Burada logit'lerin bir olasılık skoruna dönüştüğünü görülebilir. Bu olasılıkların denk geldiği label'ları hatırlamak için aşağıdaki bloğu çalıştırabilirsiniz:

In [ ]:
```
model.config.id2label
```

Out[ ]:
```
{0: 'NEGATIVE', 1: 'POSITIVE'}
```

Bu durumda sonucu aşağıdaki gibi yorumlayabiliriz:

- Cümle 1: NEGATIVE: 0.0402, POSITIVE: 0.9598
- Cümle 2: NEGATIVE: 0.9995, POSITIVE: 0.0005