

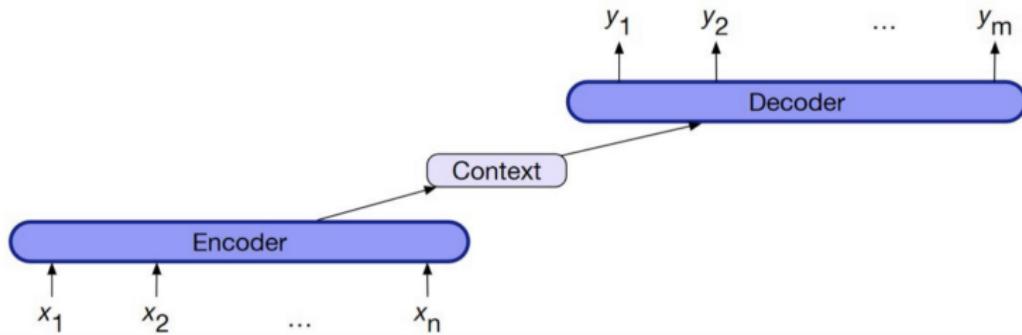
# Transformers

Dr.Furkan Göz

Aralık 18, 2023

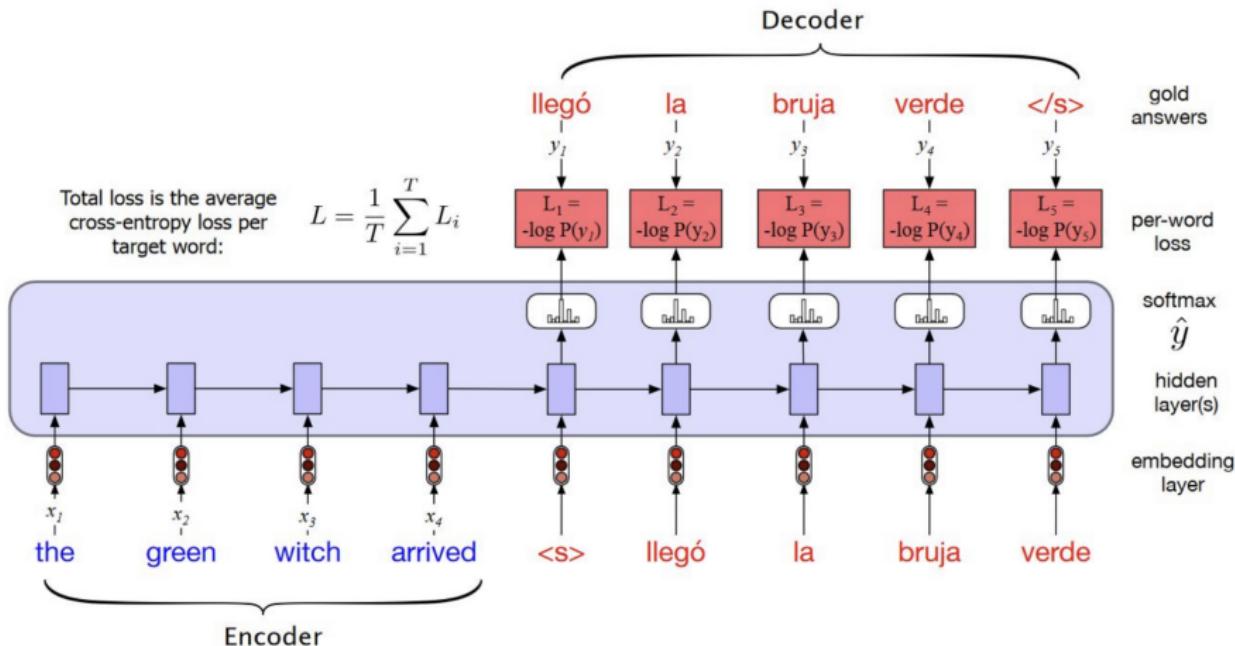
- **Encoder (Kodlayıcı):** Giriş verisini (örneğin bir metin) alır ve bu veriyi bir dizi sayısal değere (korteks vektörü) dönüştürür. Encoder giriş verisinin önemli özelliklerini ve bağlamlarını öğrenmek için tasarlanmıştır.
- **Decoder (Çözücü):** Encoder tarafından oluşturulan vektörü alır ve bunu hedef veriye (örneğin, başka bir dildeki metne) dönüştürmek için kullanır.
- Makine çevirisinde encoder kaynak dildeki bir cümleyi kodlar ve decoder bu kodlanmış bilgiyi hedef dilde bir cümleye çevirir.

# Encoder-Decoder



- Bir **encoder**, bir giriş dizisi  $x_1^n$  kabul eder ve buna karşılık gelen bağlamsal temsiller dizisi  $h_1^n$  üretir.
- Bir **context vector** ( $c$ ),  $h_1^n$  fonksiyonudur ve bilgiyi decoder'a ileter.
- Bir **decoder**,  $c$ 'yi girdi olarak kabul eder ve rastgele uzunlukta gizli durumlar dizisi  $h_1^m$ 'den karşılık gelen çıktı durumları  $y_1^m$  elde edilir.

# seq2seq (training)



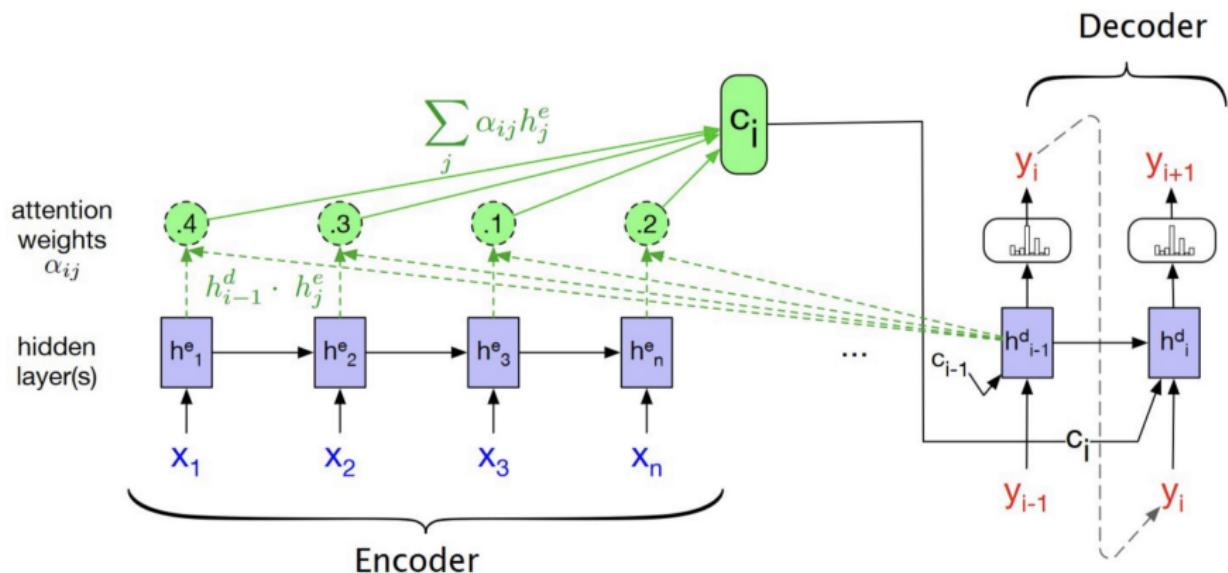
- **Context Vector:** Context vector  $h_n$ , encoder tarafından işlenen kaynak metnin son zaman adımının gizli durumunu ifade eder. Kaynak metnin tüm bilgisini sıkıştırılmış bir formda içerir ve decoder'ın çıktı üretmesi için gerekli bağlamsal bilgileri sağlar.
- **Decoder Bilgisi:** Decoder kaynak metin hakkında yalnızca context vector üzerinden bilgi sahibidir. Özellikle uzun cümlelerde cümlenin başındaki bilgilerin kaybolabileceği veya yeterince temsil edilemeyeceği anlamına gelir.
- Attention: Her çıktı kelimesi için kaynak metnin ilgili kısmına dikkat etmesini sağlar böylece modelin her bir kelimeye odaklanabilmesine olanak tanır.
- Uzun cümlelerde kaynak metnin başındaki bilgilerin de decoder tarafından dikkate alınmasını sağlar.

- Derin öğrenme modellerinde bir modelin girdi dizisindeki belirli bölmelere dikkat etmesine izin veren bir tekniktir.
- Önemli bilgileri öne çıkarabilir ve daha az önemli olanları göz ardı edebilir.
- Transformerlar dikkat modülleri içerecek şekilde geliştirilmiştir.
- Transformer modellerinde modelin tüm girdi elemanları arasındaki etkileşimleri aynı anda değerlendirmesini sağlar.

Üç bileşeni vardır:

- **Sorgu (Query):** Mevcut ya da odaklanılan girdi elemanını temsil eder. Modelin dikkatini yönlendirdiği noktadır. Modelin hangi bilgilere odaklanması gerektiğini belirler. Örneğin, bir cümledeki belirli bir kelimeyi işlerken bu kelime sorgu olarak kullanılır.
- **Anahtar (Key):** Karşılaştırma yapılacak girdi elemanlarını temsil eder. Modelin sorguya karşılaştırdığı bilgilerdir. Sorgunun hangi elemanlarla ilişkilendirileceğini belirler. Örneğin sorgunun bir kelime olduğu durumda cümledeki diğer kelimeler anahtar olarak işlev görür.
- **Değer (Value):** Anahtar ve sorgu birbiriyle yüksek uyum gösteriyorsa ilgili değerler çıktıyı oluşturmak üzere kullanılır. Modelin son çıktısını etkileyen gerçek bilgidir.

# encoder-decoder with Attention



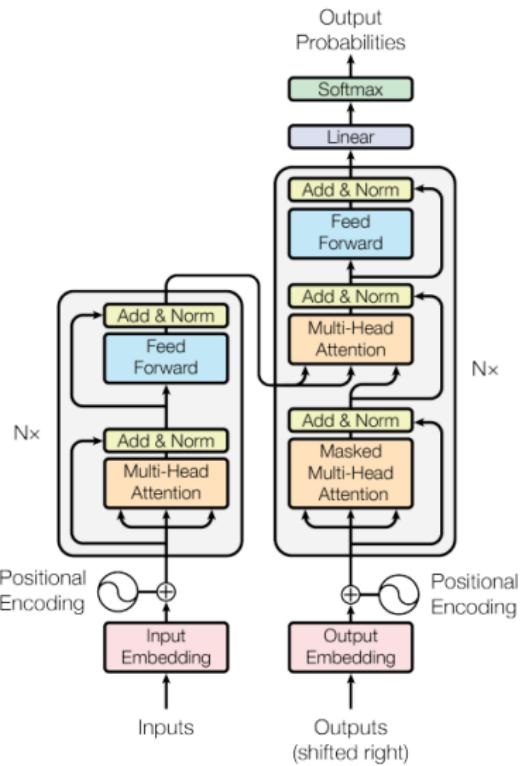
- Dikkat mekanizmalarına dayalı derin öğrenme modelleridir.
- 2017 yılında Google tarafından geliştirildi. İlk önemli model Attention Is All You Need makalesi ile tanıtılmıştır.
- Modelin bir cümledeki her kelimenin diğer kelimelerle olan ilintinin anlaşılmasını sağlar.

# LSTM vs Transformers



- LSTM gibi Transformers da uzak bilgileri işleyebilir.
- LSTM'den farklı olarak, Transformers tekrarlayan bağlantınlara dayanmaz.
- Transformers:
  - basit doğrusal katmanlar
  - ileri beslemeli ağlar
  - self-attention katmanları
- Self-attention RNN'lerde olduğu gibi ara tekrarlayan bağlantınlardan geçirmeye gerek kalmadan büyük bağamlardan bilgi çıkarmasını ve kullanmasını sağlar.

# Transformers

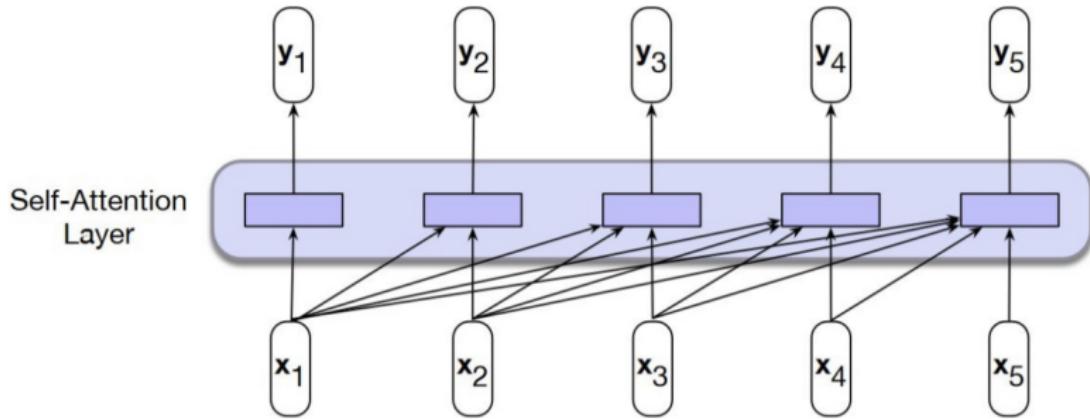


- Encoder: Girdiler ilk olarak kelime gömme (embedding) vektörlerine dönüştürülür.
- Positional Encoding: Girdi gömmelere konum bilgisi eklenir.
- Self-Attention: Bir kelimenin cümle içindeki diğer kelimelerle olan ilişkisi değerlendirilir.
- Multi-Head Attention: Farklı dikkat mekanizmalarını paralel olarak uygulamasına ve girdi dizisindeki farklı özelliklerin öğrenilmesini sağlar.

- Masked Multi-Head Attention: Bir sonraki kelimenin tahmin edilmesi gerekīinde mevcut ve önceki kelimelerin bilgisini kullanmayı sağlar. Sonraki kelimelerin bilgisini maskeleyerek görmesini engeller.
- Add & Norm: Her dikkat ve besleme ileri katmanından sonra normalizasyon uygulanır.
- Feed Forward: her bir dikkat katmanından sonra bir lineer dönüşüm uygulanır.

- Girdideki her bir öğe işlenirken model dikkate alınan öğeye kadar tüm girdilere erişebilir
- Mevcut öegenin ötesindeki girdiler hakkında bilgiye erişim yoktur
- Attention temelli yaklaşım: Bir öğeyi mevcut bağlamdaki ilgilerini ortaya çıkaracak şekilde diğer öğelerden oluşan bir koleksiyonla karşılaştırır
- Self-Attention: Karşılaştırmalar kümesi belirli bir dizideki diğer öğelerle yapılır. Bu karşılaştırmaların sonucu daha sonra mevcut girdi için bir çıktı hesaplamak için kullanılır

# Self-Attention



# Transformers: Attention

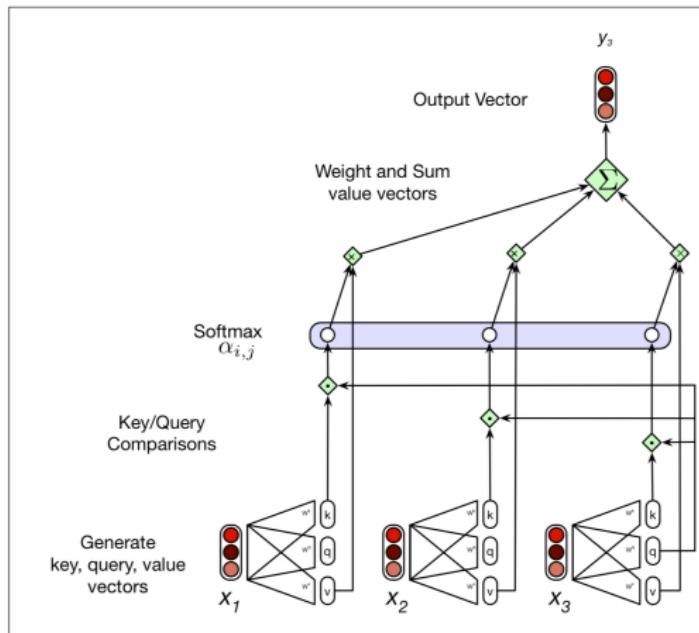


- Sorğu: Dikkatin şu anki odak noktasıdır ve tüm önceki girdilere kıyasla değerlendirilir.
- Anahtar: Mevcut dikkat odak noktasıyla karşılaştırılan önceki girdi olarak rolüdür.
- Değer: Mevcut dikkat odak noktası için çıktıının hesaplanmasıında kullanılır.

Bu üç farklı rolü yakalamak için transformers  $W^Q$ ,  $W^K$ , ve  $W^V$  ağırlık matrisleri:

- $q_i = W^Q x_i;$
- $k_i = W^K x_i;$
- $v_i = W^V x_i.$

# Self-Attention Layer

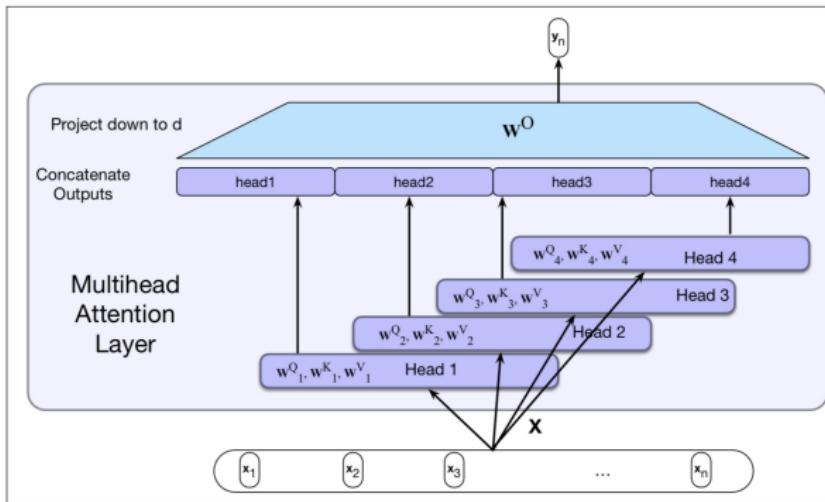


# Multi-head Attention



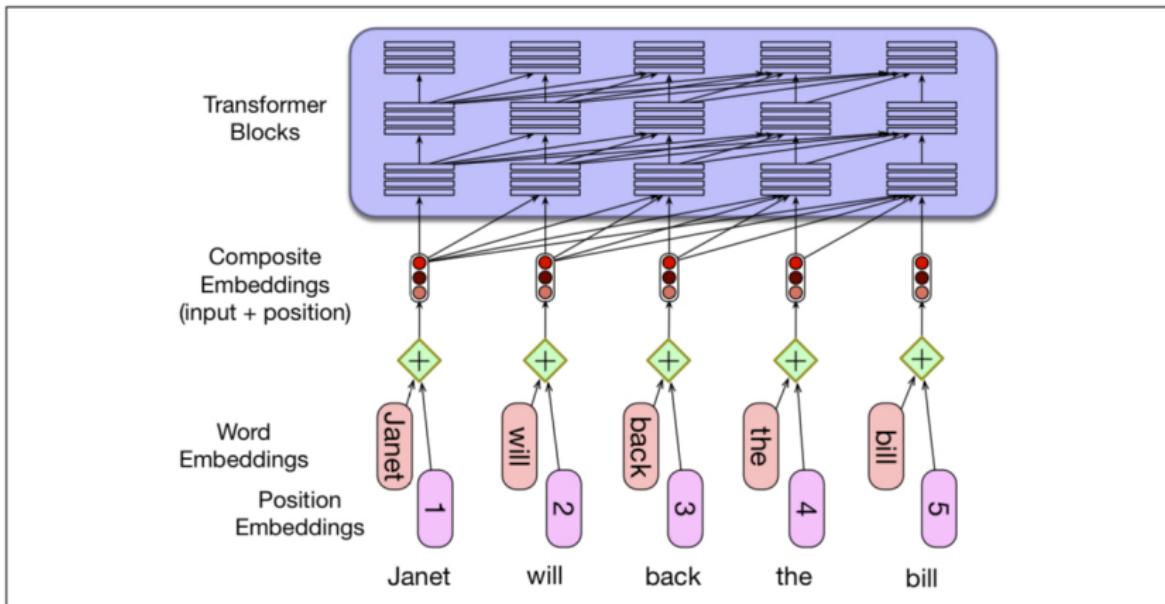
- Tek bir transformer bloğunda girdiler arasındaki farklı ilişkileri öğrenmek zordur.
- Çözüm: Transformers Multi-head Attention kullanıyor
- Self-attention katmanları kümesi head olarak adlandırılır ve modelde aynı derinlikte paralel katmanlarda bulunur. Her birinin farklı parametre değerleri var.
- Her bir head girdiler arasında var olan ilişkilerin farklı yönlerini öğrenir.
- $\text{MultiHeadAttention}(X) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$
- Her bir  $\text{head}_i = \text{SelfAttention}(Q_i, K_i, V_i)$  burada:
  - $Q_i = XW_i^Q$
  - $K_i = XW_i^K$
  - $V_i = XW_i^V$

# Multi-head Attention

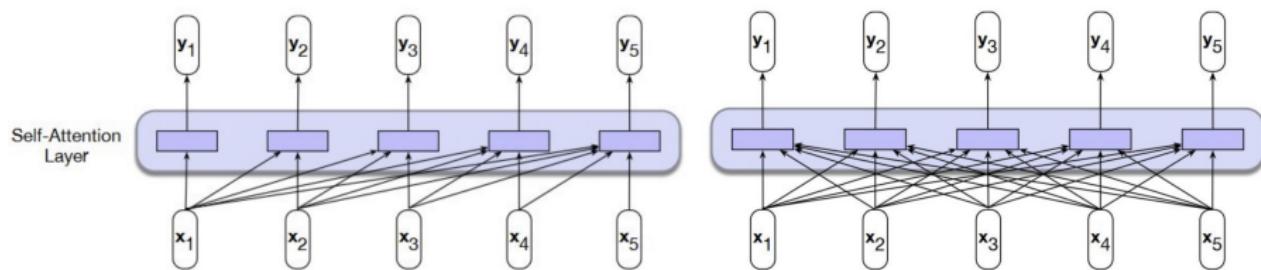


- Transformers girdi dizideki her bir elemanın sırasını nasıl belirler?
- Positional embeddings: Girdi gömülerini girdi dizisindeki her bir konuma özgü konumsal gömülerle birleştirerek değiştirir.
- Transformer modelleri girdilerin sıralı bilgisini doğrudan modelin yapısına dahil etmez.
- Her bir giriş tokenine özel bir konumsal gömme eklenir. Bu konumsal gömme vektörleri, genellikle rastgele başlatılır ve model eğitimi sırasında öğrenilir.

# Positional Embedding



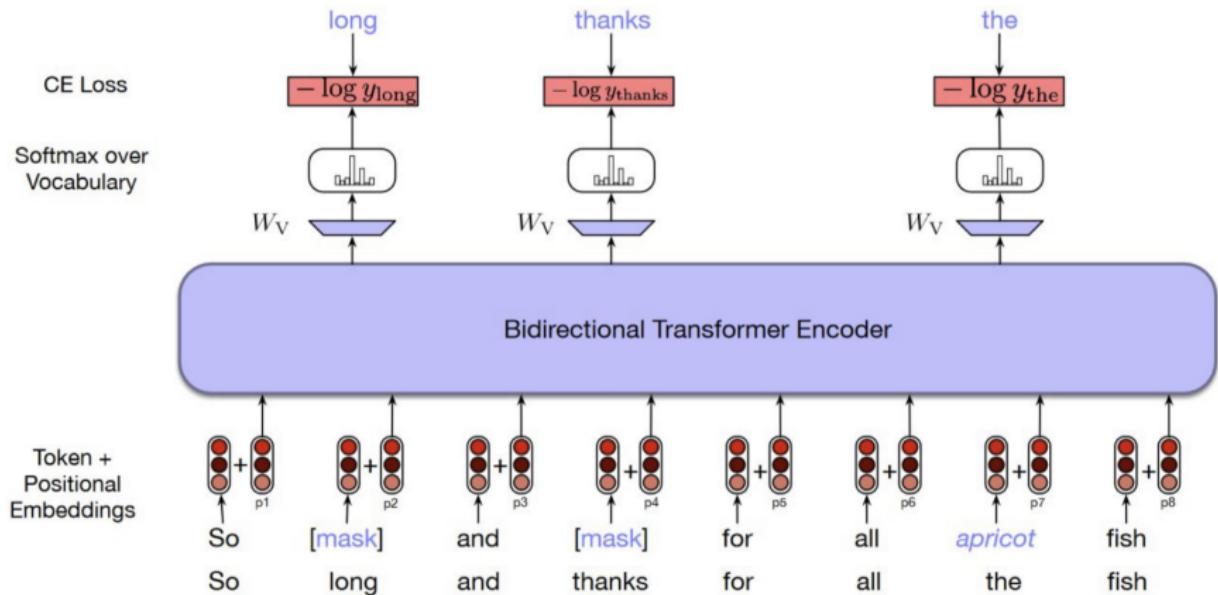
- BERT bir metindeki kelimelerin arasındaki bağlamsal ilişkileri bir dikkat mekanizması ile öğrenen Transformers modelidir.
- MLM ve NSP yaklaşımlarını kullanır.



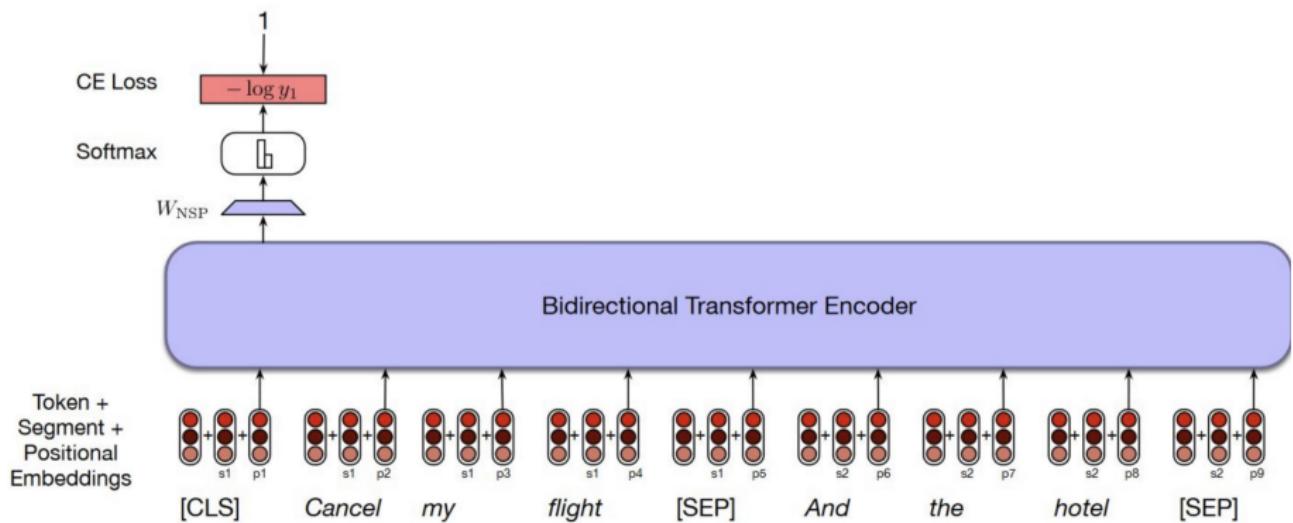
backward looking, transformer model

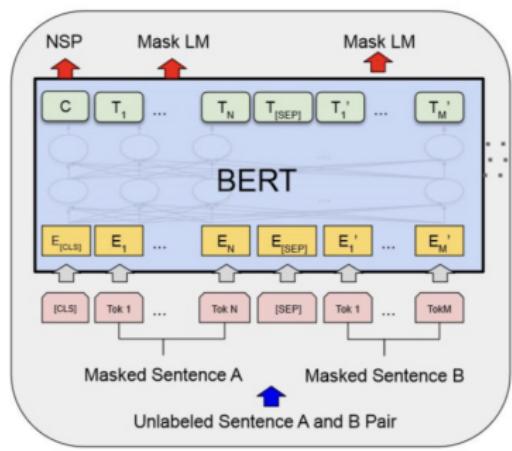
bidirectional transformer model

- Masked Language Modeling - Maskeli Dil Modellemesi (MLM) olarak adlandırılır.
- Model eğitim korpusundan alınan cümleler ile rastgele seçilen token örnekleri öğrenme görevinde kullanılır. Bir token seçildiğinde üç farklı şekilde kullanılır:
  - a. [MASK] ile değiştirilir.
  - b. Kelime dağarcığından rastgele seçilmiş başka bir token ile değiştirilir.
  - c. Değiştirilmeden bırakılır.
- BERT'te bir eğitim dizisindeki girdi tokenlerinin %15'i öğrenme için seçilir.
  - Bu tokenlerin %80'i [MASK] ile değiştirilir, %10'u rastgele seçilen tokenlerle değiştirilir ve kalan %10'u değiştirilmeden bırakılır.

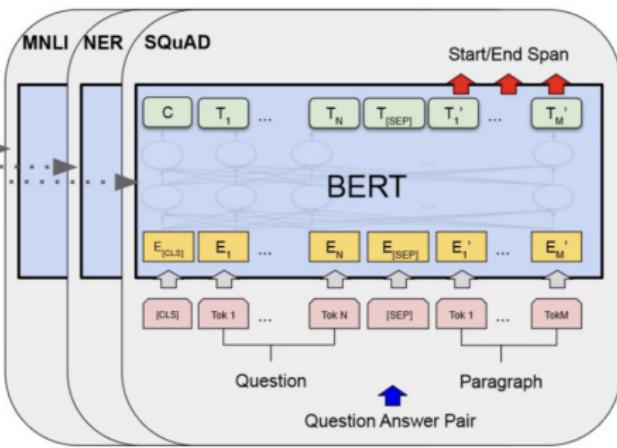


- NSP - Next Sequence PRediction iki ayrı cümleyi girdi olarak alır. Bu cümleler modelin anlamasını sağlamak için birbirleriyle ilişkilendirilir.
- Model ikinci cümlenin birinci cümlenin mantıksal ve semantik olarak devamı olup olmadığını tahmin eder.
- Model gerçek cümle çiftleri (ardışık cümleler) ve rastgele eşleştirilmiş cümle çiftleriyle eğitilir.





Pre-training



Fine-Tuning

- **BERT**
- **RoBERTa**
- **Sentence-BERT (SBERT)**: Anlamsal olarak anlamlı cümle gömülerini türeterek ve bu gömülerin kozinüs benzerliği ile karşılaştırarak semantik arama ve kümeleme gibi görevlerde kullanılır.
- **DistilBERT**: BERT'in daha hızlı ve daha küçük bir versiyonudur, daha az kaynak kullanarak benzer performansı hedefler.
- **ALBERT**: Model boyutunu ve eğitim maliyetlerini azaltmak için geliştirilmiştir.
- **BerTurk**

```
▶ from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from transformers import DistilBertTokenizer, DistilBertModel
import torch
import numpy as np

data = fetch_20newsgroups(subset='all', categories=['comp.sys.mac.hardware', 'comp.windows.x'],
                         shuffle=True, random_state=42, remove=('headers', 'footers', 'quotes'))
texts = data.data[:200]
labels = data.target[:200]

tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertModel.from_pretrained('distilbert-base-uncased')

batch_size = 10
embeddings = []

for i in range(0, len(texts), batch_size):
    batch_texts = texts[i:i+batch_size]
    encodings = tokenizer(batch_texts, padding=True, truncation=True, return_tensors="pt")
    with torch.no_grad():
        batch_outputs = model(input_ids=encodings['input_ids'], attention_mask=encodings['attention_mask'])
        batch_embeddings = batch_outputs.last_hidden_state[:, 0, :].numpy()
    embeddings.append(batch_embeddings)

embeddings = np.vstack(embeddings)
X_train, X_test, y_train, y_test = train_test_split(embeddings, labels, test_size=0.2, random_state=42)
log_reg = LogisticRegression(max_iter=1000)
param_grid = {'C': [0.01, 0.1, 1, 10]}
clf = GridSearchCV(log_reg, param_grid, cv=5)
clf.fit(X_train, y_train)

print(f'Best parameters: {clf.best_params_}')
print(f'Test accuracy: {clf.score(X_test, y_test)})')
```

Best parameters: 'C': 10 Test accuracy: 0.925