



**BURSA TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ
FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**2024 BAHAR DÖNEMİ
BLM0390 SEMİNER DERSİ RAPORU**

**EMİNE ŞENER
BİLGİSAYAR MÜHENDİSLİĞİ 3.SINIF ÖĞRENCİSİ
21360859058**

Önsöz

Bu çalışma, yapay zekanın en heyecan verici alanlarından biri olan büyük dil modellerine odaklanmaktadır. Yapay zeka dünyasında son yıllarda büyük bir patlama yaşanmıştır ve bu patlamada dil modellerinin rolü oldukça büyüktür. Bu çalışma, büyük dil modellerinin temel ilkelerinden başlayarak, son derece popüler hale gelen ve geniş bir kullanım alanına sahip olan bu teknolojilerin derinlemesine bir incelemesini sunmaktadır.

Emine ŞENER

2024, Bursa

İÇİNDEKİLER

ÖZET.....	3
1.Büyük Dil Modellerine Giriş.....	5
1.1 Üretken Yapay Zeka Tanımı.....	5
1.2 Büyük Dil Modelleri Tanımı.....	6
1.3 Transformer Derin Öğrenme Mimarisi.....	7
1.3.1 Transformer Mimarisi.....	8
1.3.2 Embedding Mimarisi.....	9
1.3.3 Self Attention Mimarisi.....	13
1.4 Büyük Dil Modellerinin Tipleri.....	19
1.4.1 Encoder Only (Yalnızca Encoder İçeren) Büyük Dil Modelleri...19	
1.4.2 Decoder Only (Yalnızca Decoder İçeren) Büyük Dil Modelleri...20	
1.4.3 Encoder ve Decoder İçeren Büyük Dil Modelleri.....	21
2. Büyük Dil Modeli NanoGPT’ nin Görselleştirilmesi	21
2.1.NanoGPT Tanımı.....	21
2.2.NanoGPT’nin Diğer Büyük Dil Modelleriyle Kıyaslanması.....	22
2.3NanoGPT Mimarisinin İncelenmesi.....	23
3.Bert Büyük Dil Modeli Destekli Türkçe Duygu Analizi Uygulaması.....	31
4.Sonuçlar	37
5. Referanslar.....	37

ÖZET

Bu çalışmanın temel amacı, büyük dil modellerinin temel kavramlarını, yapılarını ve uygulamalarını açıklamaktır. İlk olarak, yapay zekanın üretken yönünü ve bu bağlamda büyük dil modellerinin tanımını ele alıyoruz. Ardından, Transformer derin öğrenme mimarisinin temellerini ve büyük dil modellerinin farklı tiplerini inceliyoruz. İkinci bölümde, NanoGPT gibi öne çıkan bir büyük dil modelinin görselleştirilmesi ve diğer modellerle karşılaştırılması yapılmaktadır. Üçüncü bölümde ise, Bert büyük dil modeli destekli Türkçe duygu analizi uygulaması detaylı bir şekilde ele alınmaktadır. Sonuçlar bölümünde ise, çalışmanın ana bulguları ve gelecekteki araştırma alanlarına ilişkin öneriler sunulmaktadır.

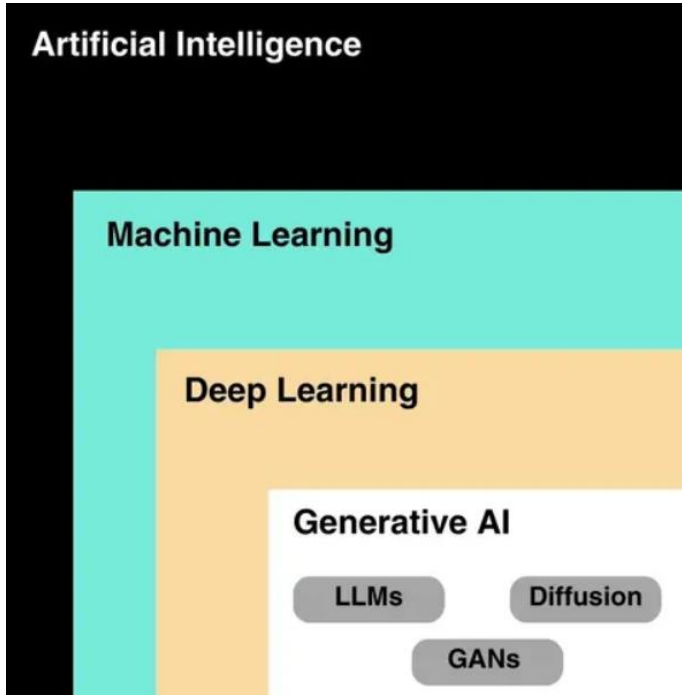
Bu çalışma, büyük dil modelleri ve onların uygulama alanları hakkında genel bir anlayış sağlamak için tasarlanmıştır ve yapay zeka alanında ilgilenen herkesin faydalanabileceği bir kaynak olmayı amaçlamaktadır.

1.Büyük Dil Modellerine Giriş

Büyük dil modelleri (LLM'ler), yapay zeka (AI) geliştirme alanında devrim yaratarak geliştiricilere daha önce ihtiyaç duyulan sürenin çok daha kısa bir sürede benzeri görülmemiş yetenekler sunmasını sağladı. Büyük dil modelleri (LLM), büyük miktarda veri üzerinde önceden eğitilmiş çok büyük derin öğrenme modelleridir. Bu modeller, geniş veri kümeleri üzerinde eğitilmiş olup, çok sayıda parametre içerir ve insan dilini anlama, üretme ve manipüle etme yeteneklerine sahiptir.

1.1 Üretken Yapay Zeka Tanımı

Üretken yapay zeka, insan yeteneğini taklit eden veya ona yaklaşan içerik yaratan makineleri içerir. İnsan tarafından oluşturulan içerikten oluşan devasa veri kümelerinden öğrenen modellere sahip, geleneksel makine öğreniminin bir alt kümesidir. Metin, resim, ses vb. gibi yeni içerik oluşturmaya odaklanan derin öğrenmenin bir alt kümesi veya uygulaması olarak düşünülebilir.



Şekil 1: Llm ve Diğer AI Alanlarının İlişkisi

Yapay zeka:

Normalde insan zekası gerektiren görevleri yerine getirebilen her türlü sistemi içeren en geniş kategori.

Makine Öğrenimi:

Makine Öğrenimi, deneyimlerden öğrenmek için algoritmaların eğitime odaklanan yapay zeka alanıdır.

Derin Öğrenme

Verilerin çeşitli faktörlerini analiz etmek için çok katmanlı sinir ağlarını kullanan bir makine öğrenimi alt kümesidir.

Üretken yapay zeka bu görselin en iç kümesidir ve içerisinde büyük dil modellerini barındırır. Bu durum LLM'lerin üretken yapay zeka uygulamalarının temelini oluşturmaktan kaynaklanır; LLM'ler, geniş veri kümeleri üzerinde eğitilerek insan benzeri metinler üretme yetenekleriyle, yaratıcı ve özgün içerikler oluşturma sürecinde kilit rol oynar. Bu süreçte, diffusion modelleri ve Generative Adversarial Networks (GAN'ler) de önemli katkılar sağlar; diffusion modelleri, karmaşık veri dağılımlarını öğrenerek gerçekçi içerikler üretirken, GAN'ler üretici ve ayırt edici ağlar arasındaki rekabet yoluyla yüksek kaliteli ve yenilikçi içerikler oluşturur.

1.2 Büyük Dil Modelleri Tanımı

Büyük Dil Modelleri (Large Language Models, LLM'ler), adından da anlaşılacağı üzere, büyük veri kümelerinden elde edilen bilgilere dayalı olarak metin ve diğer içerik biçimlerini tanıyabilen, özetleyebilen, tercüme edebilen, çıkarım yapabilen ve yeni içerikler oluşturabilen derin öğrenme algoritmalarıdır.

Bu modeller, özellikle Transformer modellerinin en başarılı uygulamaları arasında yer alır. Transformer modelleri, yalnızca insan dillerini öğrenmekle kalmaz, aynı zamanda sıralı verilerdeki ilişkileri izleyerek bağlamı ve dolayısıyla anlamı öğrenen bir sinir ağı türüdür. Transformer modelleri, araştırmacıların DNA'daki gen zincirlerini ve proteinlerdeki amino asitleri anlayarak ilaç tasarımı hızlandırmalarına yardımcı olurken; dolandırıcılığı önlemek, üretimi kolaylaştırmak, çevrimiçi önerilerde bulunmak veya sağlık hizmetlerini iyileştirmek için eğilimleri ve anomalileri tespit edebilir.

Büyük Dil Modelleri, bu geniş ve derin öğrenme yetenekleri sayesinde çok çeşitli uygulama alanlarında kullanılır ve insan benzeri metinler üretme, yaratıcı ve özgün içerikler oluşturma süreçlerinde kilit rol oynar. Bu modeller, genellikle milyarlarca parametre içerir ve geniş veri kümeleri üzerinde eğitilir, bu da onların dilin karmaşık yapısını ve anlamını derinlemesine öğrenmelerini sağlar.

Büyük dil modellerinde milyonlarca parametre, modelin dilin karmaşıklığını ve inceliklerini öğrenebilmesi için kritik öneme sahiptir. Bu parametreler, modelin esnekliğini, öğrenme kapasitesini ve genelleme yeteneğini artırarak, geniş bir yelpazede yüksek performans göstermesini sağlar. Eğitim sürecinde bu parametrelerin optimize edilmesi, büyük veri kümeleri ve yüksek hesaplama gücü gerektirir, ancak sonuçta ortaya çıkan model, çok çeşitli doğal dil işleme görevlerinde etkili çözümler sunar.

1.3. Transformer Derin Öğrenme Mimarisi

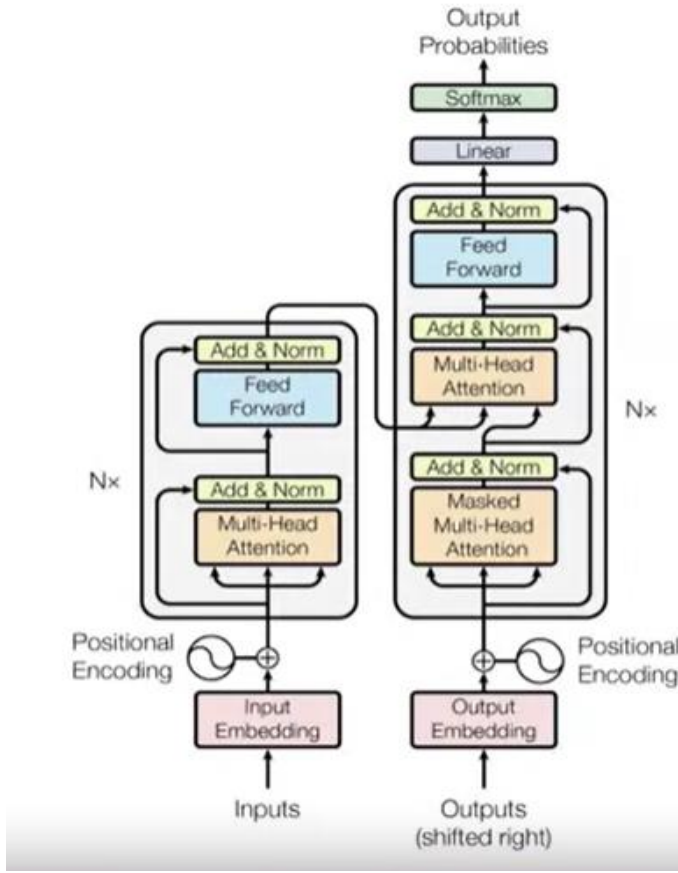
Transformer mimarisi, yapay zeka ve derin öğrenme alanında devrim yaratan bir modeldir. Bu mimari, ilk olarak 2017 yılında Google Research tarafından yayınlanan "Attention Is All You Need" adlı makalede tanıtılmıştır. Bu makale, dil modellemesi ve doğal dil işleme (NLP) alanında o döneme kadar yaygın olarak kullanılan Recurrent Neural Networks (RNN) ve Long Short-Term Memory (LSTM) modellerine alternatif olarak, daha verimli ve güçlü bir yaklaşım sunarak yapay zeka alanında bir devrim yaratmıştır.



Şekil 2:Transformer Mimarisini Tanıtan Makale

"Attention Is All You Need" makalesi, dikkat mekanizmasına dayalı Transformer mimarisini tanıtarak, derin öğrenme ve NLP alanında önemli bir dönüm noktası oluşturmuştur. Bu mimari, dilin karmaşık yapısını ve bağlamını anlamada üstün yetenekleriyle, çeşitli uygulamalarda geniş bir kullanım alanı bulmuştur. Transformer modeli, yüksek hesaplama verimliliği ve esnek yapısı sayesinde, modern yapay zeka uygulamalarının temel taşlarından biri haline gelmiştir. Günümüzde hala geliştirilmekte olan en güçlü yapay zeka araçları temelinde transformer mimarisini kullanmaktadır.

1.3.1 Transformer Mimarisi



Şekil 3:Transformer Mimarisi

Transformer mimarisi temel olarak bir encoder ve bir decoder birleşiminden oluşur. Ancak bu durum opsiyoneldir. Sadece encoder, sadece decoder veya hem encoder hem de decoder içeren LLM'ler bulunabilir. Bu seçim, probleme göre belirlenir. Örneğin, eğer problem bir çeviri problemiyse, encoder ve decoder birlikte kullanılır.

Multi-head attention ve Feed Forward, encoder ve decoder'da bulunan iki ortak yapıdır.

Input embedding, veri setinin modele sunulduğu

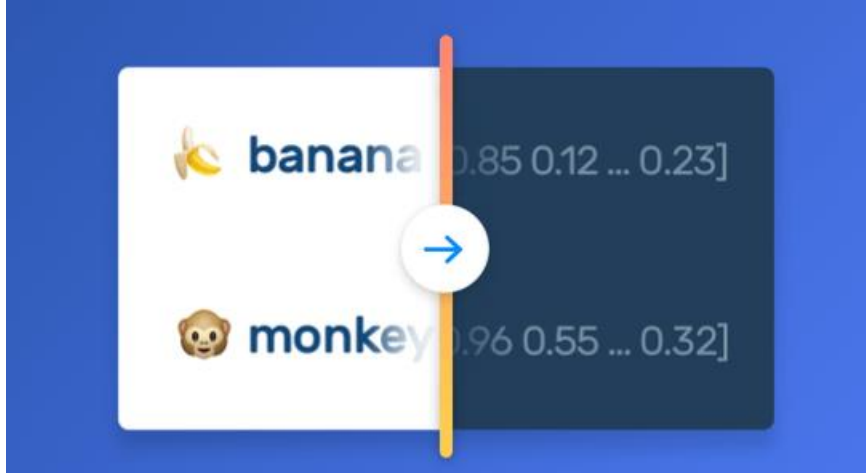
kısımdır. Output embedding ise karşılaştırma amaçlı kullanılan veri setini ifade eder. Kelimeler embedding'e dönüştürüldüğünde, işlenebilir vektörlere dönüştürülürler ve benzer anlamlı kelimeler birbirine yakın vektörlere dönüştürülür. Multi-head attention, self-attention mekanizmasının defalarca uygulanmasıdır. Multi-head attention çıktısı, girişe eklenir.

Şekil 3'te aynı zamanda encoder ve decoder bölümlerinin yanlarında xN ifadeleri de bulunur. N Transformer mimarisi içindeki bu katmanlardaki işlemlerin N kez tekrar edeceğini ifade etmektedir. N değeri problemin yapısına göre değişir. Örneğin sınıflandırma alanında bir sonuç ya da kelimenin diğer kelimelerden sonra gelme olasılığını hesaplama durumları için farklı N değerleri bulunur. Makalede tanıtılan transformer mimarisinde N değeri 6 olarak belirlenmiştir.

Transformer mimarisi kısaca birçok bileşeni birleştirme işlemi yapan bir yapıdır. Günümüzdeki AI modellerinin temelini oluşturur.

1.3.2 Embedding Mimarisi

Büyük dil modelleri, basit sinir ağıları veya makine öğrenimi algoritmaları gibi, kelimelerle iyi performans gösterememektedir. Kelimelerin büyük bir dil modeline, basit bir sinir ağında veya makine öğrenimi algoritmasında kullanılabilmesi için, kelimelerin sayılara dönüştürülmesi gerekir.



Şekil 4: Kelimelerin Sayısallaştırılması

Transformer ve LLM'ler gibi büyük dil modellerinde, embedding genellikle kelime veya token seviyesindeki giriş verilerini sayısal değerlerle ifade etmek için kullanılır.

Transformer'da, embedding katmanı genellikle giriş verilerini (örneğin, kelimeleri veya tokenleri) vektörel bir uzaya dönüştürmek için kullanılır. Bu dönüştürme işlemi, genellikle bir kelime dağarcığını temsil eden bir matrisin içerisinde her bir kelimenin veya tokenin gömülü olduğu bir embedding matrisi kullanılarak gerçekleştirilir. Bu Embedding matrisi, modelin dil yapısını anlamasına ve dil görevlerini gerçekleştirmesine yardımcı olur.

LLM'lerde, Embedding genellikle büyük bir veri kümesi üzerinde önceden eğitilerek elde edilir. Bu önceden eğitilmiş Embedding, kelime ilişkilerini ve anlamlarını yakalamak üzere optimize edilmiştir. LLM, giriş metni veya belgeyi bu önceden eğitilmiş Embedding vektörlerine dönüştürerek, metni sayısal değerlerle temsil eder. Bu sayede, model metinler arasındaki anlamı ve bağlamı daha iyi anlayabilir ve çeşitli doğal dil işleme görevlerini gerçekleştirebilir.

Kelimelerin sayılara dönüştürülmesi için birden fazla teknik bulunmaktadır. Bunlardan birisi kelimelere rastgele sayıların atanmasıdır.

“I saw a cat.” cümlesindeki her bir kelimeye ve de noktaya farklı bir sayı rastgele olarak atanır.

39 1592 10 2548 5
↑ ↑ ↑ ↑ ↑
I saw a cat .

Şekil 5:cümlelerin tokenleştirilmesi

Troll 2 → 12
is → -3.05
great! → 4.2

Dil modellerinin hedefine uygun çalışması için bağlamdan uzaklaşmaması gerekmektedir. Rastgele atanan değerler, kelimeler arasındaki bağlantıyı yok edebilir.

Şekil 6:Kelimelerin Tokenleştirilmesi

“Troll 2 is great!” cümlesindeki her bir kelime resimdeki gibi farklı sayısal değerlerle temsil edilebilir.

“Troll 2 is great!” cümlesine oldukça benzer olan “Troll 2 is awesome!” cümlesindeki awesome kelimesine rastgele sayısal değer atanarak çok farklı bir değer atanabilir.

great! → 4.2 awesome! → -32.1

Şekil 7:Benzer Kelimelerin Tokenleştirilmesi

“great” ve “awesome” kelimeleri benzer anlam taşımlarına ve benzer biçimde kullanılmalarına rağmen rastgele olarak birbirinden çok farklı sayılarla temsil ediliyorlar. Benzer kelimelerin birbirinden çok farklı sayılarla temsil edilmesi embedding’in karmaşıklığını artırır ve eğitim sürecini zorlaştırır.

“great” kelimesine nasıl doğru bir şekilde işleyeceğini öğrenen bir model, “awesome” kelimesini öğrenirken “great” kelimesinden herhangi bir yardım alamaz. Bu durum büyük dil modelinin gücünü azaltır. Benzer kelimelere benzer sayıların verilmesi daha güçlü büyük dil modellerinin tasarlanmasını sağlar. Çünkü benzer kelimeler benzer sayılarla temsil edildiğinde benzer kelimeler içerisinde bir kelimenin nasıl kullanılacağını öğrenmek aynı anda tüm benzer kelimelerin kullanımını öğrenmeye yardımcı olur.

Aynı kelime kökü farklı bağlamlarda kullanılabileceğinden, çoğul hale getirilebileceğinden ya da özellikle Türkçe veriler üzerinde çalışılıyorsa pek çok farklı çekim ekiyle çekimlenebileceği için her farklı kelimeye farklı ama benzer bir sayı atanması önemlidir. Bu nedenle kelimelere rastgele sayısal değerler atama yöntemi kelimelerin bağlam ilişkilerine dikkat etmediği için büyük dil modellerinde kullanılacak girdilerin sayısal değerlere döndürülmesi için sinir ağı eğitmek daha iyi sonuçlar vermektedir.

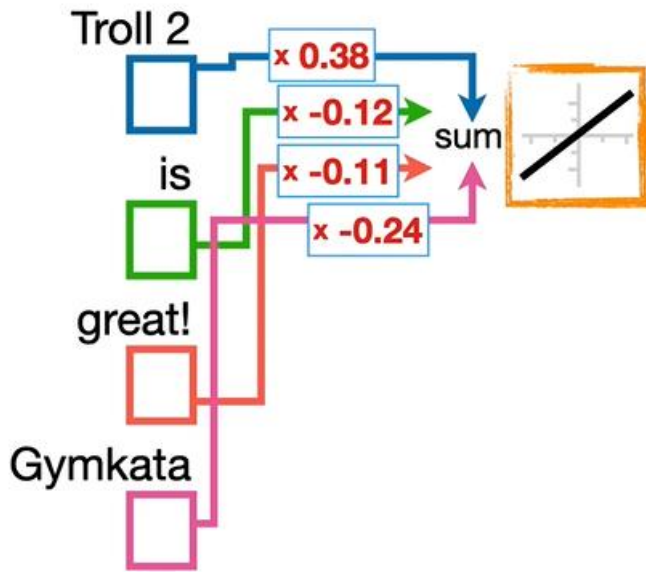
Embedding için sinir ağı eğitmenin en büyük nedeni kelimelerin bağlam ilişkisini göze almaktır. Örneğin “great” kelimesi “It is great!” cümlesinde pozitif bir anlam taşıırken, “My cell phone is broken, great!” negatif bir anlam taşır. Bu sebeple aynı kelimenin olumlu ve olumsuz temsillerine de farklı değerler atanır. Kelimelerin benzer olduğun karar vermek ve benzer bağlamda kullanıldığını tespit etmek yüksek hesaplama gerektiren bir işlemdir ve bu görev için basit sinir ağı eğitilir.

Training Data

Troll 2 is great!
Gymkata is great!

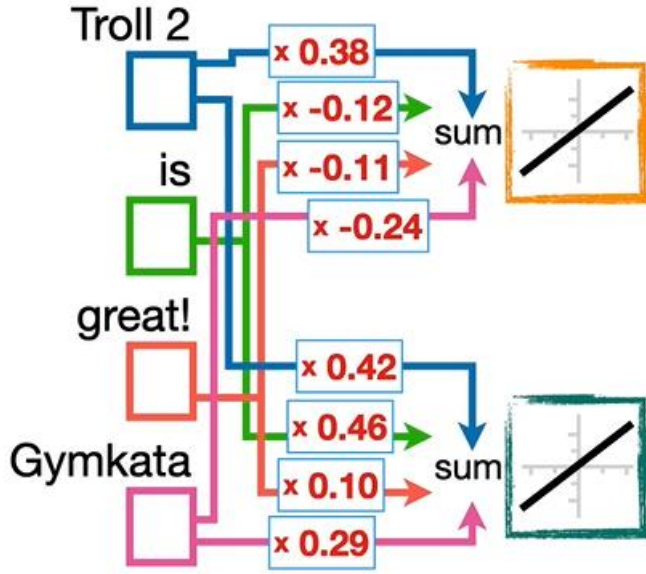
Şekil 8: Basit bir veri seti

Embedding için basit bir sinir ağı eğitmek için gerçek hayatta kullanılan eğitim verilerine göre oldukça basit olan 4 farklı girdiden oluşan veri seti ele alınarak embedding eğitimi ele alınır.



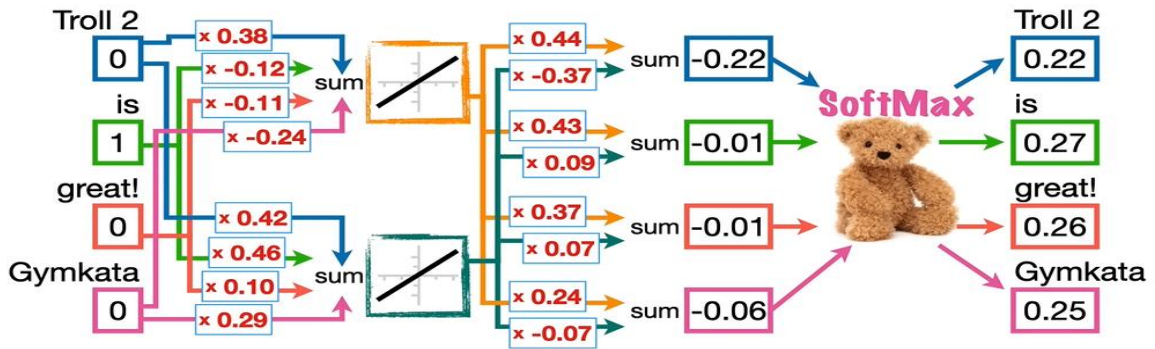
Şekil 9: Sinir Ağı Giriş Katmanı

Sinir ağı oluşturmak için her benzersiz kelime için farklı bir girdi oluşturulmaktadır. Her giriş farklı ağırlık değerleriyle çarpılır ve aktivasyon fonksiyonundan geçirilir. Embedding için kullanılan aktivasyon fonksiyonu her farklı input için farklı bir output üretmeyi sağlar.



Şekil 10: Aktivasyon fonksiyonları

Embedding eğitilirken her girişe bir ya da birden fazla aktivasyonu ilişkilendirilir. Aktivasyon fonksiyonu sayısı her kelimeyle kaç farklı değeri ilişkilendirmek istenilmesine göre belirlenir. Şekilde temsili gösterilen sinir ağı için her biri giriş için iki farklı ağırlık değeri ve ilişkilendirilir ve iki adet aktivasyon fonksiyonu bulunur



Şekil 11: Embedding Sinir Ağı

Şekilde “is” kelimesinden sonra gelmesi gereken kelimenin tahmin edilme süreci yer alır. Giriş değerlerinde “is” tokeni 1 diğerleri 0 ile temsil edilir. Başlangıç ağırlık değerleri rastgele olarak belirlenir. Her girdi için iki farklı aktivasyon fonksiyonunun ürettiği değerler önemlidir. Aktivasyon fonksiyonları her giriş değerini işledikleri için kelimelerin sayılara dönüştürülme sürecinde bağlamları, anlamları ve benzerlikleri korur.

Başlangıç ağırlık değerleri rastgele atandığı için sinir ağının eğitim süreci başlangıçta yanlış sonuçlar üretir. Örneğin şekilde “is” kelimesinden sonra gelecek kelime olarak 0.27 en büyük değeri ile yine “is” değeri belirlenmiş. Yanlış sonuç üretiminin iyileştirilmesi için back-propagation kullanarak ağırlık değerlerimizi, temsili yüksek değerler haline getirmemiz gerekir.

1.3.3 Self Attention Mimarisi

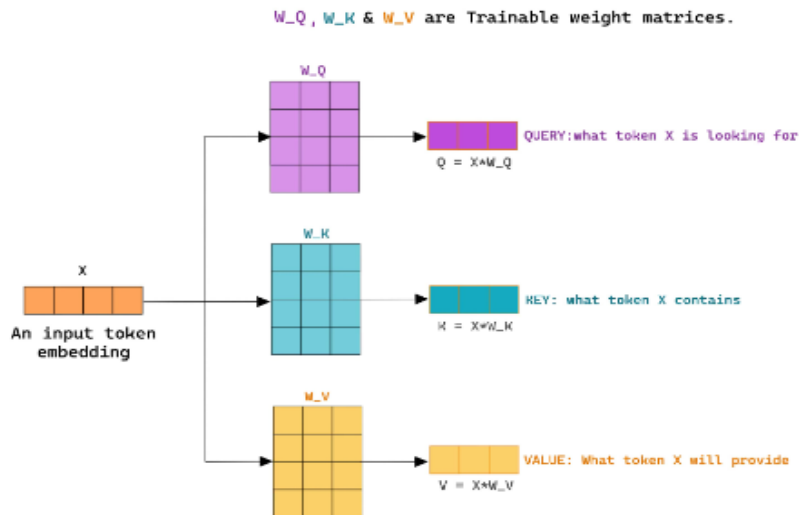
Bert, ALBERT, RoBERTa, DistilBERT gibi modellerin ortak noktası isimlerindeki benzerlik BERT değil temelde hepsinin transformer mimarisini kullanıyor olamalarıdır. Son zamanlarda AI gelişmelerinin temel yapı taşını transformers mimarisi oluşturmaktadır. NLP, Computer Vision gibi AI tekniklerinin insan algısına yakın bir düzeye gelmesindeki en büyük etken bu Transformers mimarisi. Transformer mimarisinin temelini de “self attention” kavramı oluşturur.

Yapay sinir ağlarında her şey sayısallaştırılır. Transformer modellerine beslenecek giriş kelimeleri, vektörler ile temsil edilir. Kelime şeklinde bir girdi yerine vektörler şekline girdiler veriyoruz. Verilen bu girdiler daha önceden eğitilmiş Embedding modelleri kullanılarak elde ediliyor.

```
Input 1: [1, 0, 1, 0]
Input 2: [0, 2, 0, 2]
Input 3: [1, 1, 1, 1]
```

Self attention mekanizmasına şekildeki gibi (gerçek hayat problemlerine göre oldukça basitleştirilmiş)sayısal değerlerle temsil edilen 3 farklı giriş verilir. 3 farklı giriş için başlangıç parametresi olan 4 değerinde uzunluğa sahip vektörler üretilir.

Kelime temsil eden her adet vektör uygun matrislerle çarpılarak key, value, query şeklinde 3 yeni vektör elde edilir. Key, value ve query vektörlerinin üretilmesi için kullanılan 3 farklı matris model vocabulary içerisinde bulunan diğer tokenler için de kullanılır. Burada parametre paylaşımı söz konusudur(Ağırlık matrislerinin tüm giriş değerleri için kullanılabilmesi).



Şekil 12:Eğitilebilir Ağırlık Matrisleri

Query, Value ve Key için ayrı eğitilebilir weight matrisleri bulunur. Eğitilebilir matris, genelde 0'a yakın değerler seçilerek başlatılan ardından eğitim sürecinde iyileştirilen değerler içeren matrislerdir.

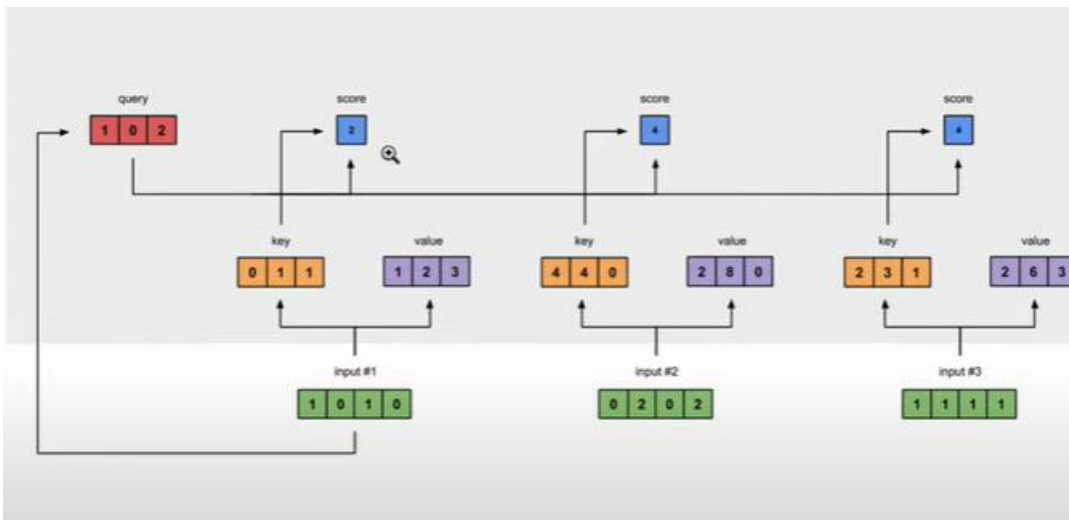
Matrisin eğitim öncesi ilk değerlerine parametre değerleri gözüyle bakılabilir. Her bir giriş için hesaplanan key, value, query değerleri ayrı key, value ve query matrislerinde tutulabilir. Veya aşağıdaki şekildeki gibi 3 girişin değerlerini tek bir key, value ve query matrislerinde tutabilir. Veya aşağıdaki şekildeki gibi 3 girişin değerlerini tek bir key, value ve query matrislerinde tutabilir.

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 4 & 4 & 0 \\ 2 & 3 & 1 \end{bmatrix}$$

Hesaplanan ayrı key vektörleri birbirinden bağımsız işlenmelidir ve ayrıca bu vektörler genelde tensör olarak adlandırılırlar. Bağımsız oldukları için tensörler matrisi olarak bir araya getirilebilir.

Elde edilen her bir query vektörünün vocabulary içerisinde bulunan her bir token vektörünün key vektörüyle ilişkisinin hesaplanması gerekir. Çünkü “self attention” mekanizması cümledeki kelimelere ve kelimelerin bağlamına odaklanır. Bunun için key ve query vektörleri arasında iç çarpımı yapılır. İç çarpım kullanılabilecek tek yöntem değildir, iki vektörün benzerliğini bulmak için pek çok farklı yöntem mevcut.

Key ve query değerleri iç çarpım ile çarpılır ve tek hücreli bir score vektörü oluşturulur.



Şekil 13:Self Attention Yapısı

Tek bir token vektörünün query ağırlık matrisi ile çarpılması sonucu elde edilen query vektörü için her bir giriş tokeninin key ağırlık matrisiyle çarpılması sonucu elde edilen key vektörü çarpılır. İlk token vektörünün query vektörü ile her bir vektörün key vektörü çarpılır ve ilk vektörün kendisiyle ve diğer giriş token vektörleriyle olan ilişkisi ortaya çıkar.

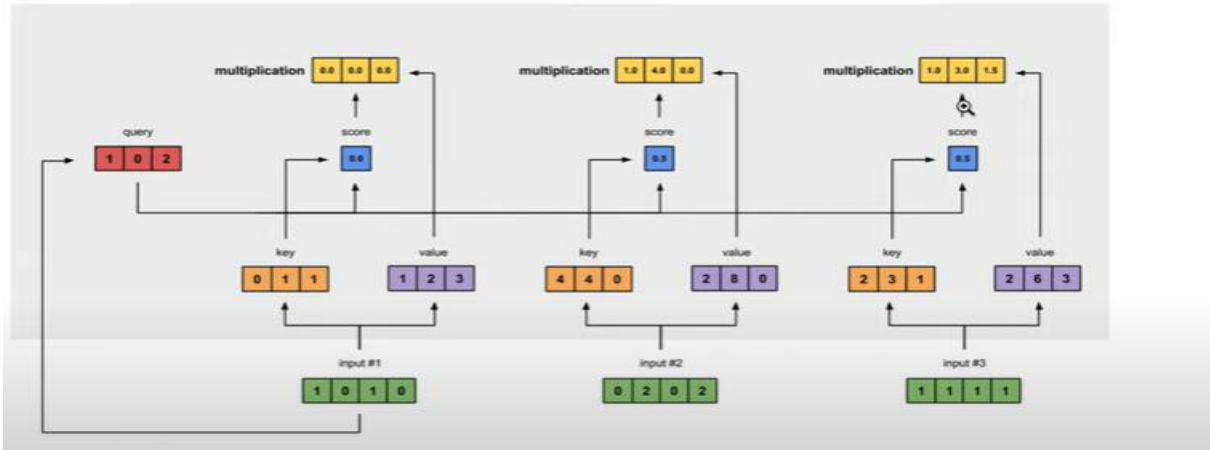
Bu işlem aynı zamanda çok büyük değerler üretebilir ve ürettiği bu sayıların yönetimi oldukça zor olabilir. Bu yüzden üretilen bu sayıların olasılık gibi yorumlanabilmesi için softmax fonksiyonundan geçirilmesi şarttır. Softmax sayıları 1 ve 0 arasında sıkıştırır ve sonuçların toplamını 1'e eşitler.

$$\text{softmax}([2, 4, 4]) = [0.0, 0.5, 0.5]$$

Şekildeki örnekte 2, 4, 4 “attention skorları” softmax fonksiyonundan geçirilir ve 0.0, 0.5 ve 0.5 değerleri elde edilir.

İlk giriş vektörü için yapılan her bir işlem model vocabulary içerisinde bulunan tüm giriş token vektörleri için yapılır. Bu süreç oldukça fazla işlem gerektirmesine rağmen, her farklı giriş tokeni için aynı işlemler yapıldığı için işlemler paralelleştirilebilir ve GPU üzerinde çalıştırılabilir. Böylelikle eğitim süreci tahmin edildiği kadar uzun sürmez.

Her giriş vektörü için bir score değeri oluşturulduktan sonra her bir score değeri kendi giriş token vektöründen üretilen value token vektörü ile skaler çarpılır.



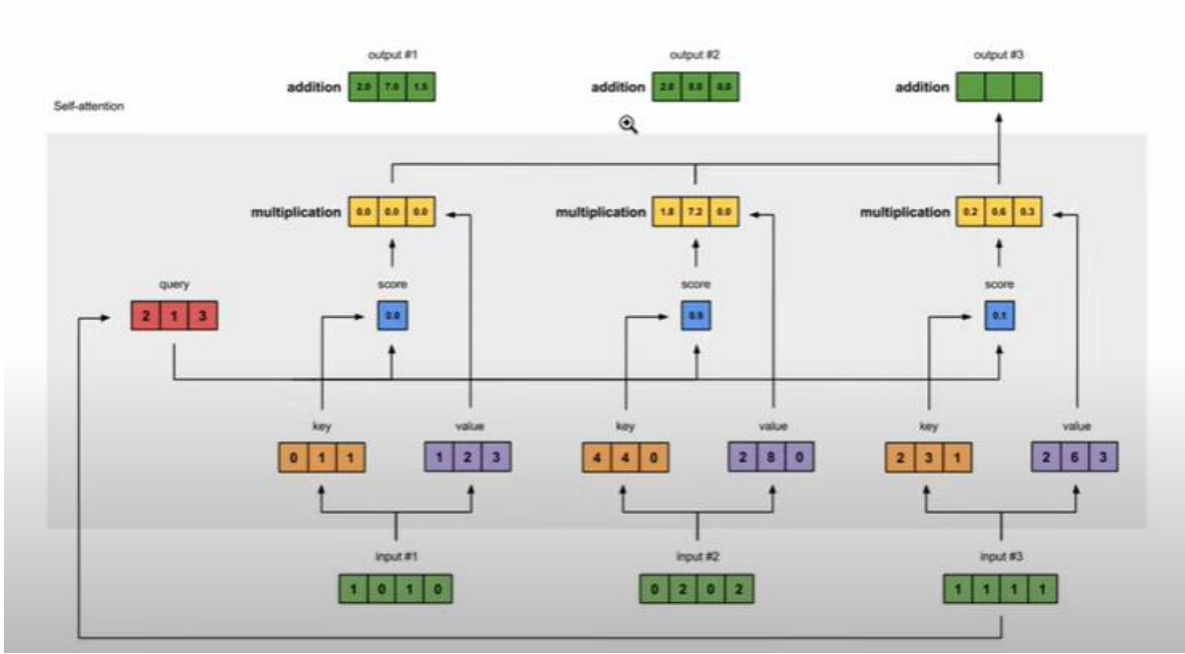
Şekil 14:Self Attention

$$\begin{aligned} 1: & 0.0 * [1, 2, 3] = [0.0, 0.0, 0.0] \\ 2: & 0.5 * [2, 8, 0] = [1.0, 4.0, 0.0] \\ 3: & 0.5 * [2, 6, 3] = [1.0, 3.0, 1.5] \end{aligned}$$

Her input vektöründen elde edilen score ve value vektörleri çarpılır.

$$\begin{array}{r}
[0.0, 0.0, 0.0] \\
+ [1.0, 4.0, 0.0] \\
+ [1.0, 3.0, 1.5] \\
\hline
= [2.0, 7.0, 1.5]
\end{array}$$

Elde edilen bu vektörler toplanır. Böylelikle çıkış vektörü 1, birinci giriş token vektörünün yeni temsil vektörü elde edilmiş olur. Tüm işlemler, her input token vektörü ile tekrar edilir.



Şekil 15: Temel Self Attention Mimarisi

“Self Attention” süreci tamamlandığında girişteki 4 boyutlu vektörler yerine “self attention” ile üretilen 3 boyutlu yeni vektörler kullanılır.

Self Attention Mekanizmasının Python ile Kodlanması

<pre> 1 import torch 2 3 x = [4 [1, 0, 1, 0], # Input 1 5 [0, 2, 0, 2], # Input 2 6 [1, 1, 1, 1] # Input 3 7] 8 x = torch.tensor(x, dtype=torch.float32) </pre>	<p>Self attention mekanizması tensör değerleri kullanarak çalışır. Tensör değerleri Pytorch kütüphanesi ile kullanılabilir. Input token vektörleri temsili değerler olarak başlatılır. Vocabular x matrisindeki her bir token torch.float32 ile tensörlere dönüştürülür. Bu süreçte float32 gibi hafıza için uygun bir değer seçmek de oldukça önemlidir.</p>
<pre> 1 w_key = [2 [0, 0, 1], 3 [1, 1, 0], 4 [0, 1, 0], 5 [1, 1, 0] 6] 7 w_query = [8 [1, 0, 1], 9 [1, 0, 0], 10 [0, 0, 1], 11 [0, 1, 1] 12] 13 w_value = [14 [0, 2, 0], 15 [0, 3, 0], 16 [1, 0, 3], 17 [1, 1, 0] 18] </pre>	<p>Key, Value ve Query vektörlerini belirlemek için kullanılacak olan ağırlık matrisleri başlatılır. Başlangıç değerleri 0'a yakın seçilir ve eğitilebilir parametre değerleridir. Sınır ağının eğitim sürecinin başlangıcında yanlış sonuçlar üretmesine back-propagation yöntemi ile eğitim boyunca iyileştirilerek doğru sonuçlar üretebilir değerlere dönüştürülürler.</p>
<pre> 19 w_key = torch.tensor(w_key, dtype=torch.float32) 20 w_query = torch.tensor(w_query, dtype=torch.float32) 21 w_value = torch.tensor(w_value, dtype=torch.float32) </pre>	<p>Her bir ağırlık matrisi tensörler haline getirilir.</p>
<pre> keys = x @ w_key querys = x @ w_query values = x @ w_value print(keys) # tensor([[0., 1., 1.], # [4., 4., 0.], # [2., 3., 1.]]) print(querys) # tensor([[1., 0., 2.], # [2., 2., 2.], # [2., 1., 3.]]) print(values) # tensor([[1., 2., 3.], # [2., 8., 0.], # [2., 6., 3.]]) </pre>	<p>Query, key ve value vektörleri üretilerek matrislere depolanır. Bu süreçte de tüm matrisler tensör olarak temsil edilir. Giriş değerlerini depolayan x matrisi her bir ağırlık matrisi ile çarpılır.</p>
<pre> 1 attn_scores = querys @ keys.T 2 3 # tensor([[2., 4., 4.], # attention scores from Query 1 4 # [4., 10., 12.], # attention scores from Query 2 5 # [4., 12., 10.]]) # attention scores from Query 3 </pre>	<p>Query çarpımları ile attention skorları hesaplanır.</p>

```

1 from torch.nn.functional import softmax
2
3 attn_scores_softmax = softmax(attn_scores, dim=-1)
4 # tensor([[6.3379e-02, 4.6831e-01, 4.6831e-01],
5 #        [6.0337e-06, 9.8201e-01, 1.7986e-02],
6 #        [2.9539e-04, 8.8054e-01, 1.1917e-01]])
7
8 # For readability, approximate the above as follows
9 attn_scores_softmax = [
10     [0.0, 0.5, 0.5],
11     [0.0, 1.0, 0.0],
12     [0.0, 0.9, 0.1]
13 ]
14 attn_scores_softmax = torch.tensor(attn_scores_softmax)

```

Attention skorları Pytorch içerisinde bulunan ve normalizasyon sağlayan softmax fonksiyonundan geçirilir.

```

1 weighted_values = values[:,None] * attn_scores_softmax.T[:,None]
2
3 # tensor([[0.0000, 0.0000, 0.0000],
4 #        [0.0000, 0.0000, 0.0000],
5 #        [0.0000, 0.0000, 0.0000],
6 #        [1.0000, 4.0000, 0.0000],
7 #        [2.0000, 8.0000, 0.0000],
8 #        [1.8000, 7.2000, 0.0000],
9 #        [1.0000, 3.0000, 1.5000],
10 #        [0.0000, 0.0000, 0.0000],
11 #        [0.2000, 0.6000, 0.3000]])

```

Elde edilen softmax fonksiyonundan geçirilmiş olan attention skorları value vektörleriyle çarpılır. Ağırlıklandırılmış değerler elde edilir.

```

1 outputs = weighted_values.sum(dim=0)
2
3 # tensor([[2.0000, 7.0000, 1.5000], # Output 1
4 #        [2.0000, 8.0000, 0.0000], # Output 2
5 #        [2.0000, 7.8000, 0.3000]]) # Output 3

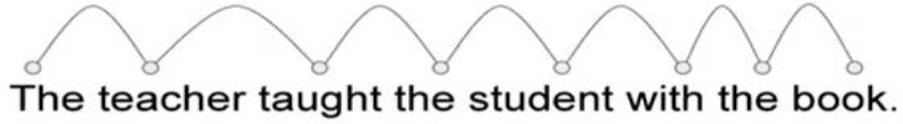
```

Son olarak bulunna ağırlıklandırılmış değerler toplanır ve yeni temsil vektörü elde edilmiş olur.

“Self Attention” mekanizmasındaki tüm işlemlerin matrislerle yapılması GPU ve TPU gibi donanımları kullanabilmek anlamına gelir. Self Attention mekanizmasındaki neredeyse tüm işlemler eş zamanlı işlemlerdir. Dolayısıyla bu karmaşık işlemler GPU ve TPU ile kısa sürede tamamlanabilir.

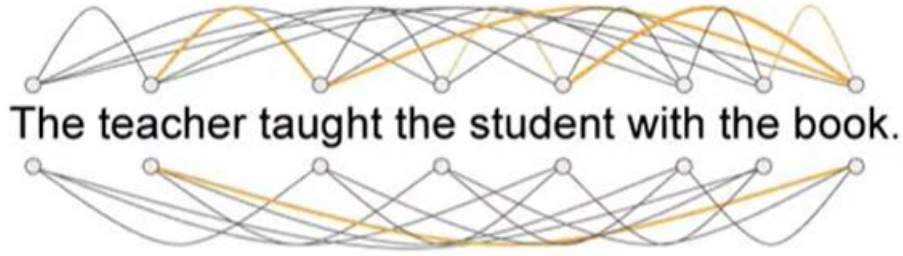
Transformer mimarisinin içindeki “Multiheadattention” bu dikkat mekanizmasının çoklu kullanımıdır ve Pytorch içerisindeki Keras kütüphanesinde hazır olarak bulunmaktadır.

“Self attention” kavramı somut örnek ile açıklanabilir. “Self attention” öncesi her bir kelimenin diğer tüm kelimelerle ilişkisi ele alınmadığı için cümle bağlamı işlenemiyordu.



Şekil 16:Self Attention Olmadan İlişkiler

“Self Attention” mekanizması ile her bir kelimenin diğer tüm kelimelerle ilişkisi değerlendirilir.

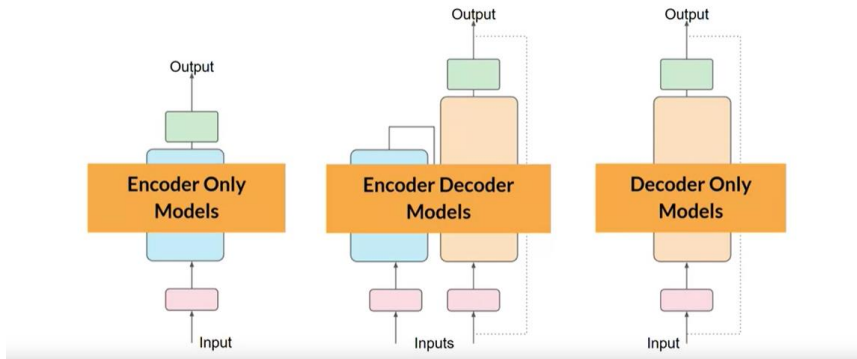


Şekil 17:Self Attention İle İlişkiler

1.4 Büyük Dil Modellerinin Tipleri

Büyük dil modellerinin temelini Trasnformer mimarisi oluşturur. Transformer pek çok bileşen içeren ve temelinde encoder ve decoder olmak üzere iki bölümden oluşan bir yapıdır. Büyük dil modelleri transformer mimarisini kullanırken her iki yapıyı da kullanmak zorunda değildir. Hatta bu encoder ve decoder yapılarını kullanıp kullanmamalarına göre büyük dil modelleri farklı görevler için üretilebilir.

Transformers

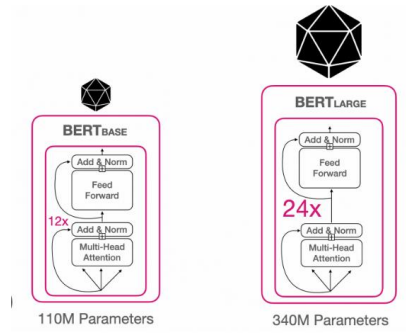


Şekil 18:Transformer Tipleri

1.4.1 Encoder Only (Yalnızca Encoder İçeren) Büyük Dil Modelleri

Encoder-only(yalnızca encoder içeren) modeller, girişleri alıp kodlayan ve genellikle girdi dizilerini başka bir formata dönüştürmeden işlem yapan modellerdir. Bu modeller, girdi dizilerinin uzunluğunu değiştirmeden çıktılar üretirler. Encoder-only modeller metin sınıflandırma, duygu analizi, adlandırılmış varlık tanıma gibi görevlerde kullanılır.

BERT (Bidirectional Encoder Representations from Transformers) modeli, bir encoder-only modeldir. BERT, bir cümleyi alır ve bu cümlemin anlamını kodlar, böylece sınıflandırma veya etiketleme gibi görevlerde kullanılabilir.

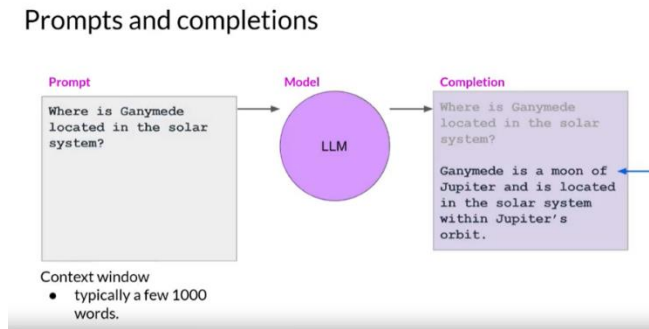


Şekil 19: Bert

1.4.2 Decoder Only (Yalnızca Decoder İçeren) Büyük Dil Modelleri

Decoder-only(yalnızca decoder içeren) modeller, genellikle sadece çıktı üretmeye odaklanırlar ve otoregresif olarak çalışırlar, yani her adımda önceki adımın çıktısını kullanarak bir sonraki adımı tahmin ederler. Bu modeller, çeşitli görevlerde genelleme yapabilirler.

Decoder-only metin üretimi, dil modelleme, sohbet botları gibi görevlerde kullanılır. GPT-3 (Generative Pre-trained Transformer 3) modeli, bir decoder-only modeldir. Örneğin, bir başlangıç cümlesi verildiğinde, devam eden metni tahmin edebilir ve üretebilir.

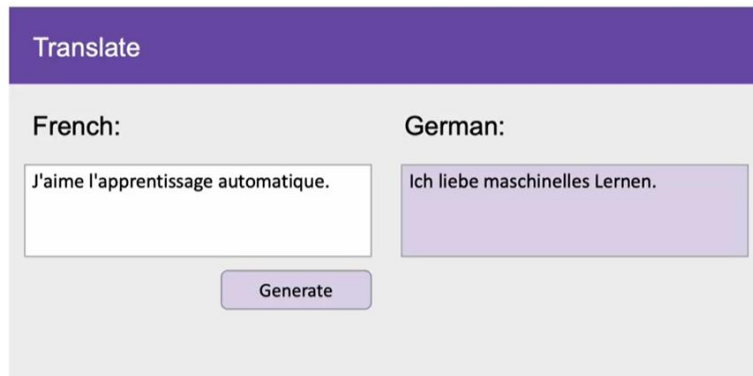


Şekil 20: Decoder Only Model

1.4.3 Encoder ve Decoder İçeren Büyük Dil Modelleri

Encoder-decoder(mimaride hem encoder hem de decoder bulunur) modelleri, girdileri kodlayan (encode) ve ardından bu kodlamaları çözerek (decode) farklı uzunluklarda çıktı dizileri üreten modellerdir. Bu tür modeller, girdi ve çıktı dizilerinin uzunluklarının farklı olabileceği görevlerde başarılıdır.

Encoder- decoder modeller makine çevirisi, metin özetleme, soru-cevap gibi görevlerde kullanılır. T5 (Text-to-Text Transfer Transformer) modeli, bir encoder-decoder modelidir. Örneğin, Fransızca bir cümleyi alıp onu almanca bir cümleye çevirme görevi için kullanılır.



Şekil 21:Translate Örneği

3. Büyük Dil Modeli NanoGPT' nin Görselleştirilmesi

Raporun bu aşamasında büyük dil modellerinden NanoGPT kullanılarak büyük dil modellerinin mimarisi görselleştirilecektir.

3.1.NanoGPT Tanımı

NanoGPT, GPT-2'nin minimal bir uygulaması olup, basit ve anlaşılabilir yapısı ile özellikle eğitim ve araştırma amaçları için uygundur. Küçük boyutu ve hızlı çalışması sayesinde, transformer tabanlı dil modellerinin temellerini öğrenmek ve yeni fikirleri denemek isteyenler için ideal bir araçtır.

3.2.NanoGPT’nin Diğer Büyük Dil Modelleriyle Kıyaslanması

NanoGPT, büyük dil modelleri (BDM) ile karşılaştırıldığında belirgin farklılıklara sahiptir. Model boyutu ve karmaşıklığı açısından NanoGPT, GPT-2'nin sadeleştirilmiş bir versiyonu olarak, daha az parametre içerir ve daha basit bir yapıya sahiptir. Bu özellikler, NanoGPT'yi hafif ve hızlı çalışır hale getirirken, GPT-3 ve GPT-4 gibi büyük dil modelleri milyarlarca parametreye sahip olup çok daha karmaşıktır ve büyük donanım kaynakları gerektirir.

NanoGPT, basitliği ve erişilebilirliği ile öne çıkar ve eğitim, öğretim, araştırma ve küçük ölçekli projeler için mükemmeldir. Buna karşın, büyük dil modelleri yüksek performans, doğruluk ve geniş kullanım alanları sunar, ancak daha yüksek maliyet ve karmaşıklık ile birlikte gelirler. Bu modeller, büyük ölçekli ve karmaşık dil işleme görevlerinde üstün performans sergilerler.

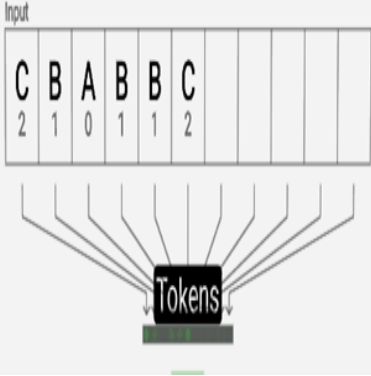
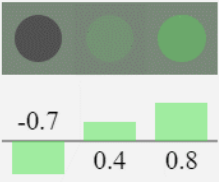
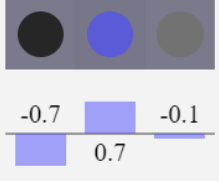
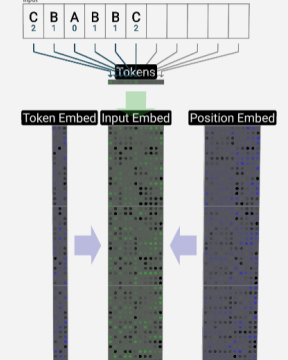
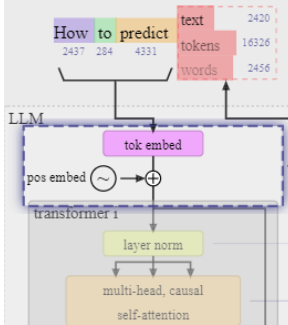


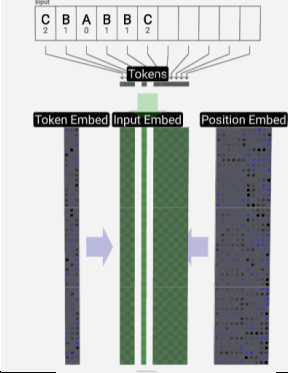

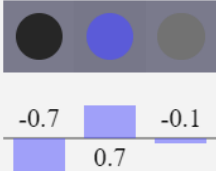
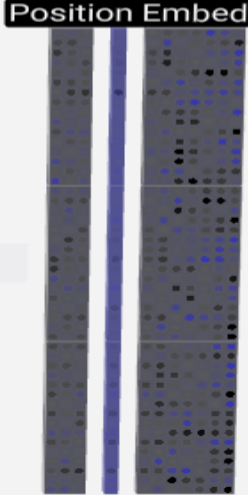
Şekil 22:NanoGpt

3.3 NanoGPT Mimarisinin İncelenmesi

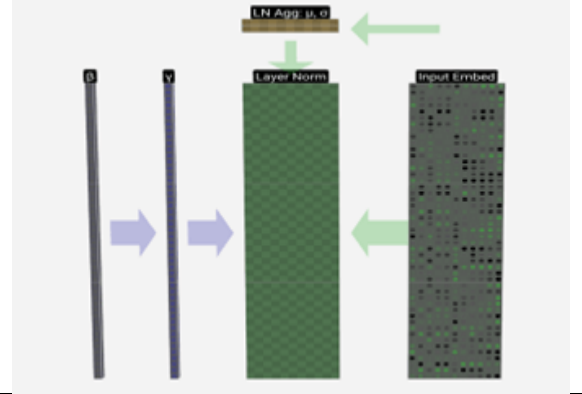
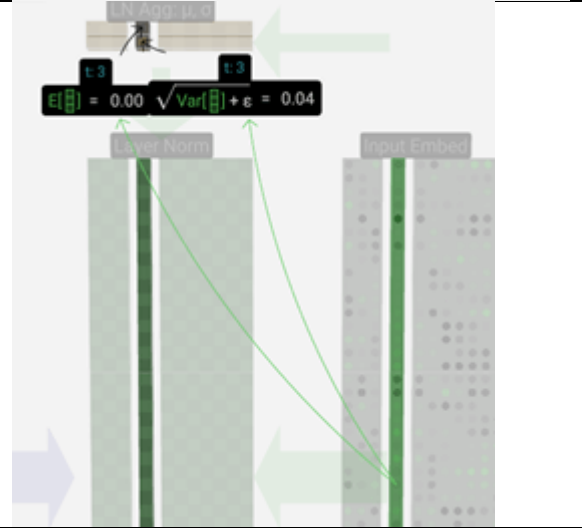
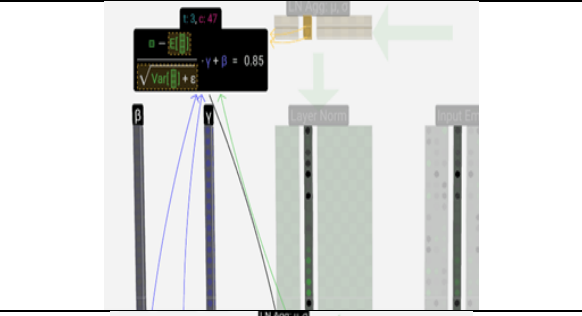
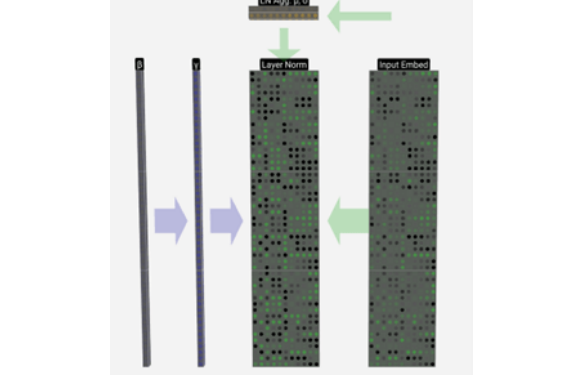
Bu aşamada sadece 85.000 parametrelili nano-gpt modelini incelenecektir. Modelin çalışmasını incelemek için gerçek hayat problemlerinin karmaşıklığına göre oldukça basit olan 6 harften oluşan bir dizi alınmaktadır.

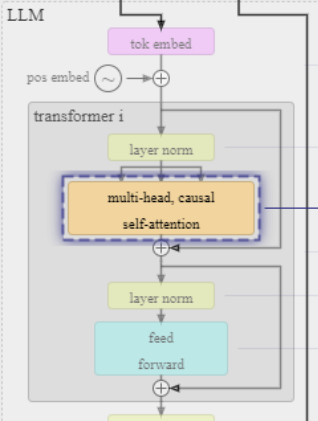
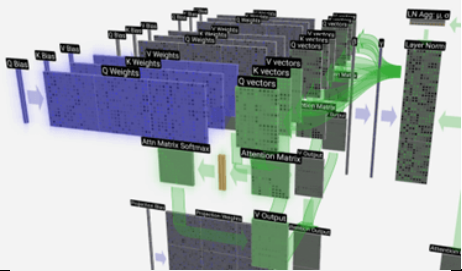
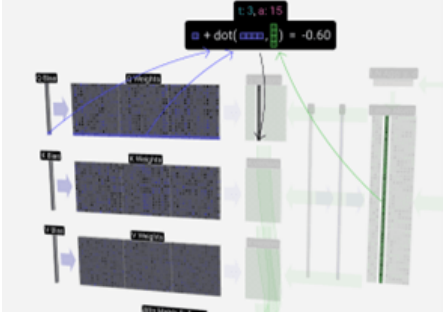
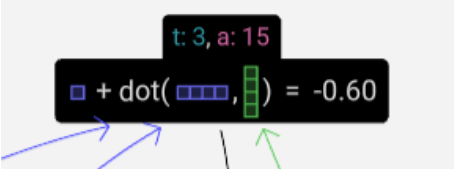
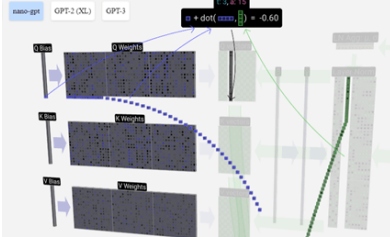
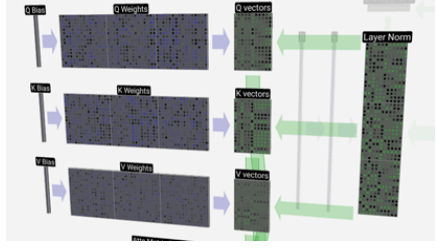
<p>CBABBC</p>	Başlangıç olarak 6 harften oluşan ve gerçek hayat problemlerine göre oldukça basit olan bir dizi alınır ve bu dizinin alfabetik olarak ters sırada sıralanması hedeflenir.																				
<p>Input</p> <table><tr><td>C</td><td>B</td><td>A</td><td>B</td><td>B</td><td>C</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td>1</td><td>0</td><td>1</td><td>1</td><td>2</td><td></td><td></td><td></td><td></td></tr></table> <p>Tokens</p>	C	B	A	B	B	C					2	1	0	1	1	2					Giriş dizesi öncelikler her harfinin birbirinden ayırmak adına tokenleştirilir ve birbirinden ayrılan bu harflerin her biri farklı sayısal değerlerle temsil edilir. Token uzunluğu büyük dil modellerine verilen parametreler aracılığıyla belirlenir. Oluşturulan token dizisinde önemli kısımlar yeşil olarak, dizinin uzunluğunu diğer tüm tokenlerle sabit tutmak için yapılan eklemeler renksiz hücre olarak temsil edilmektedir.
C	B	A	B	B	C																
2	1	0	1	1	2																
<table><tr><td>token</td><td>A</td><td>B</td><td>C</td></tr><tr><td>index</td><td>0</td><td>1</td><td>2</td></tr></table>	token	A	B	C	index	0	1	2	Dizedeki her farklı token için farklı bir index değeri belirlenir ve bu tokenler ve index'ler modelin “vocabulary” kelime dağarcığını oluşturur.												
token	A	B	C																		
index	0	1	2																		
<p>2 1 0 1 1 2</p>	Sekans tokenleştirme ve indexleme işleminden sonra yalnızca sayısal değerler içeren bir dizeye dönüştürülür. “CBABBC” dizisi modele 210112 olarak beslenir.																				

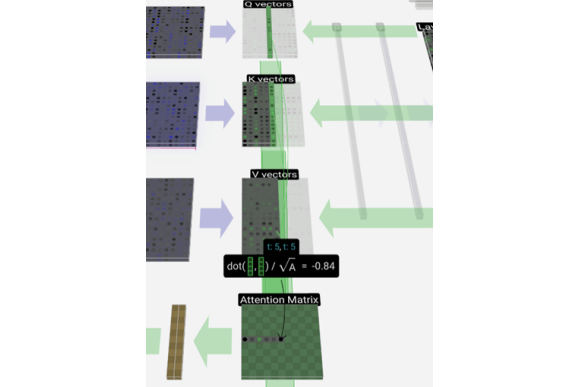
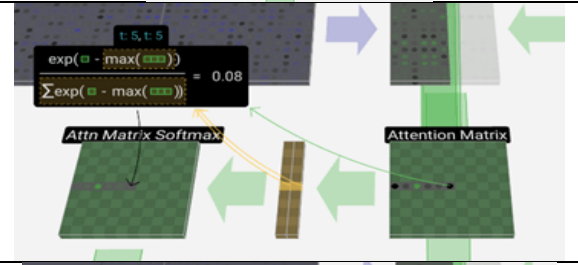
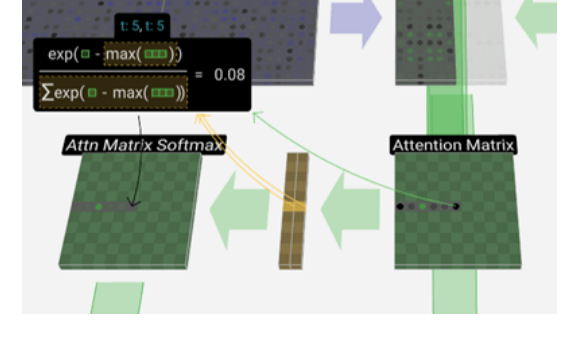
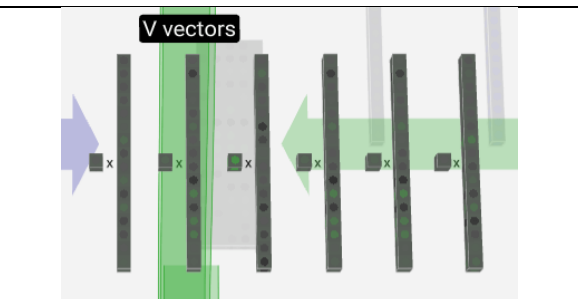
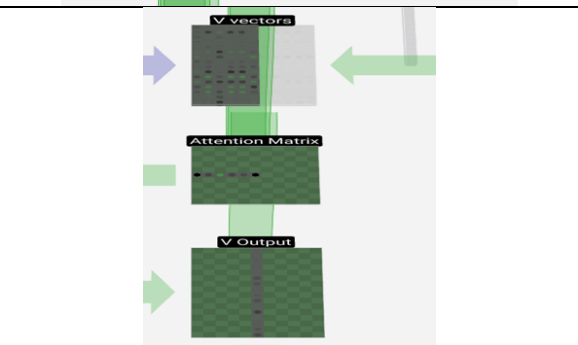
	<p>Başlangıçta tüm tokenler tam sayı değerlerle temsil edilmişti ama tam sayı değerler büyük dil modellerinin karmaşıklığını karşılamakta yetersizdir. Verinin transformer yapılarına beslenmeden önce float sayılarla temsili oluşturulmalıdır.</p>
 <p>being processed</p>	<p>Tokenleştirme sürecinde her yeşil hücre işlenmekte olan bir sayıyı temsil eder ve her mavi hücre bir ağırlıktır.</p>
 <p>weights</p>	<p>Sıradaki her sayı ilk önce 48 elemanlı bir vektöre dönüştürülür (bu özel model için seçilen boyut). Buna gömme denir.</p>
	<p>Her sayı öncelikle 48 elemanlı bir vektöre dönüştürülür. Bu işleme "embedding" denir. Dizideki her tam sayı, sırasıyla 48 elemanlı bir vektöre dönüştürülür. Bu dönüşüm sonucunda "input embed" yapısı oluşturulur.</p>
	<p>Embedding süreci, büyük dil modellerinde veriler transformer mimarisine verilmeden önce gerçekleşir.</p>

	<p>Embedding süreci, dizinin 3. indeksinde bulunan ve 1 ile temsil edilen B tokeni ile incelenecektir.</p>
	<p>Sol kısımda yer alan token embedding matrisi, her bir token için 48 boyutlu yeni bir vektör üretir. Burada 48 olarak belirlenen boyut, büyük dil modeline verilen parametrelere göre farklı değerlerde olabilir. Token embedding matrisi, modelin kelime dağarcığındaki token sayısı kadar sütuna sahiptir. Bu örnekte, basit olarak 'ABC' tokenleri '012' olarak temsil edildiği için yalnızca 3 sütundan oluşan ve 'ABC' harflerini temsil eden bir token embedding matrisi kullanılır.</p>
 <p>weights</p>	<p>Token embedding matrisi oluşturulma süreci ve ağırlıklarının belirlenmesi, raporun embedding kısmında ayrıntılı olarak anlatılmıştır.</p>
	<p>Embedding sürecinin bir diğer önemli matrisi ise position embedding matrisidir. Embedding sürecinde başlangıçta belirlenen 48 değeri tüm matrislerde aynı olmak zorundadır. Position embedding matrisi, her bir tokenin hangi konumda olduğunu belirtir. Giriş dizisinin 3. indeksindeki B değerinin embedding süreci ele alındığı için, position embedding matrisinin 3. sütunu incelenir.</p>

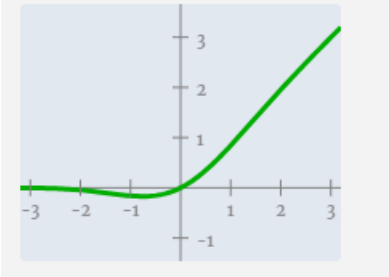
	<p>Giriş sekansı "CBABBC" için, 3. indekste bulunan B tokenini embedding matrisinde temsil etmek üzere, token embedding içerisinde 'B' tokenini temsil eden 1. sütun ile position embedding'de 3. indeksi temsil eden 3. sütun toplanır. Bu işlem sonucunda, input embedding matrisinin 3. indeksindeki B harfini temsil eden kısım oluşturulmuş olur.</p>
	<p>Bahsedilen süreç, tüm tokenler için gerçekleştirilir ve bu sayede transformer mimarisine teslim edilecek input embedding matrisi oluşturulur.</p>
	<p>Büyük dil modellerinin çalışma sürecindeki bir diğer önemli adım, katman normalizasyonu (layer normalization) sürecidir. Önceki bölümde oluşturulan giriş embedding matrisi, ilk Transformer bloğunun girişidir. Transformer bloğundaki ilk adım, bu matrise katman normalizasyonu uygulamaktır. Bu, matrisin her sütunundaki değerleri ayrı ayrı normalleştiren bir işlemdir.</p>

	<p>Normalleştirme, derin sinir ağlarının eğitiminde önemli bir adımdır ve eğitim sırasında modelin kararlılığının artırılmasına yardımcı olur. Her sütunu ayrı ayrı ele alabildiğimiz için, şimdilik 4. sütuna ($t=3$) odaklanacağız.</p>
	<p>Normalizasyonun amacı, sütundaki ortalama değeri 0'a ve standart sapmayı 1'e eşitlemektir. Bunun için, sütun için bu iki niceliği (ortalama (μ) ve standart sapma (σ)) buluruz ve sonra ortalama değeri çıkarıp standart sapmaya böleriz.</p>
	<p>Burada kullandığımız notasyon, ortalama için $E[x]$ ve varyans için (C uzunluğundaki sütunun) $Var[x]$'dir. Varyans basitçe standart sapmanın karesidir. Epsilon terimi ($\epsilon = 1 \times 10^{-5}$), sıfıra bölünmeyi önlemek için eklenmiştir.</p>
	<p>Her satır ve sütun için bu işlemler yapılır ve sonuç matrisi elde edilir. Normalleştirilmiş değerlere sahip olduğumuzda, sütundaki her öğeyi öğrenilmiş bir ağırlıkla (γ) çarpıyoruz ve ardından bir sapma (β) değeri ekleriz, böylece normalleştirilmiş değerler elde edilir. Bu normalleştirilmiş embedding matrisi, Self-Attention katmanına iletilir.</p>

	<p>Kişisel dikkat katmanını, belki de Transformer'ın ve GPT'nin kalbidir. Bu aşama, girdi yerleştirme matrisindeki sütunların birbirleriyle "konuştuğu" aşamadır. Şu ana kadar ve diğer tüm aşamalarda sütunlar bağımsız olarak değerlendirilebilir.</p>
	<p>Büyük dil modellerinde kullanılan self-attention yapısı, çoklu başlıklı dikkat (multi-head attention) şeklinde birçok katmandan oluşur.</p>
	<p>İlk adım, normalleştirilmiş girdi gömme matrisinden T sütunları için üç vektör üretmektir. Bu vektörler Q, K ve V vektörleridir: S:Sorgu vektörü, K: Anahtar vektör, V: Değer vektörü. Bu vektörlerden birini üretmek için önyargı eklenmiş bir matris-vektör çarpımı gerçekleştirilir. Her çıkış hücresi, giriş vektörünün bazı doğrusal birleşimidir.</p>
	<p>Q vektörleri için bu, Q-ağırlık matrisinin bir satırı ile giriş matrisinin bir sütunu arasındaki nokta çarpımı ile oluşturulur.</p>
	<p>Birinci vektördeki her elemanı ikinci vektördeki karşılık gelen elemanla eşleştiririz, bu çiftleri birbiriyle çarpıyoruz ve ardından sonuçları topluyoruz.</p>
	<p>Bu, her bir çıktı ögesinin giriş vektöründeki tüm öğelerden etkilenebilmesini sağlamak için genel ve basit bir yoldur. Bu nedenle sınır ağlarında sıklıkla görülür. Bu işlemi Q, K, V vektörlerindeki her çıkış hücresi için tekrarlanır.</p>

	<p>{K, V} girişleri, geçmişteki 6 sütunu, Q değeri ise şimdiki zamanı göstermektedir. İlk olarak, mevcut sütunun Q vektörü ($t = 5$) ile önceki sütunların her birinin K vektörleri arasındaki nokta çarpımını hesaplıyoruz. Bu hesaplamalar daha sonra dikkat matrisinin karşılık gelen satırında ($t = 5$) saklanır.</p>
	<p>Bu nokta çarpımlar, iki vektör arasındaki benzerliği ölçmenin bir yoludur. Eğer vektörler çok benzerse, nokta çarpımı büyük olacaktır. Eğer vektörler çok farklıysa, nokta çarpımı küçük veya negatif olacaktır.</p>
	<p>Diğer bir unsur da nokta çarpımı aldıktan sonra \sqrt{A}'ya bölmemizdir; burada A, Q/K/V vektörlerinin uzunluğudur. Bu ölçeklendirme, bir sonraki adımda büyük değerlerin normalizasyona (softmax) hakim olmasını önlemek için yapılır. Her satırın toplamı 1 olacak şekilde normalleştirilir.</p>
	<p>Normalleştirilmiş öz-dikkat matrisinin ($t = 5$) satırına bakıyoruz ve her öğe için diğer sütunların karşılık gelen V vektörünü öğe bazında çarpıyoruz.</p>
	<p>Daha sonra çıktı vektörünü oluşturmak için bunları toplayabiliriz. Böylece çıktı vektörüne yüksek puanlara sahip sütunlardan gelen V vektörleri hakim olacaktır.</p>

	<p>Kişisel dikkat katmanının başı için olan süreçtir. Yani öz-dikkatin temel amacı, her bir sütunun diğer sütunlardan ilgili bilgileri bulmak ve değerlerini çıkarmaktır. Bu, sorgu vektörünü diğer sütunların anahtarlarıyla karşılaştırarak gerçekleştirilir. Ancak, bu sürecin yalnızca geçmişe bakabilmesine ilişkin ek kısıtlamaları da vardır.</p>
	<p>Transformatör bloğunun kişisel dikkatten sonraki bir sonraki yarısı MLP'dir (çok katmanlı algılayıcı). Bu, biraz ağız dolusu olabilir, ancak burada iki katmanlı basit bir sinir ağı bulunur. Öz-dikkatte olduğu gibi, vektörler MLP'ye girmeden önce bir katman normalizasyonu gerçekleştirilir</p>
	<p>MLP'de, $C = 48$ uzunluğundaki sütun vektörlerimizi (bağımsız olarak) aşağıdakilere koyarız.</p>
	<p>4 * C uzunluğundaki bir vektöre önyargı eklenmiş doğrusal bir dönüşüm.</p>
	<p>GELU aktivasyon fonksiyonu (öge bazında) 3. bias eklenmiş, C uzunluğundaki bir vektöre geri dönen doğrusal bir dönüşüm.</p>
	<p>Öncelikle matris-vektör çarpımını önyargı eklenmiş olarak çalıştırılır ve vektör 4 * C uzunluğuna kadar genişletilir.</p>

	<p>ReLU aktivasyon fonksiyonu vektörün her elemanına uygulanır. Bu, modele non-linear yapı eklemenin önemli bir parçasıdır ve herhangi bir sinir ağında yaygın olarak kullanılır.</p>
	<p>Kullanılan özel işlev GELU, ReLU işlevine çok benzer şekilde çalışır ($\max(0, x)$ olarak hesaplanır), ancak keskin bir köşe yerine düzgün bir eğriye sahiptir</p>

3.Bert Büyük Dil Modeli Destekli Türkçe Duygu Analizi Uygulaması

Bert büyük dil modelleri içerisinde en basit modellerden birisidir ve transformer mimarisi duygu analizi için uygundur.

Öncelikle gerekli tüm kütüphaneler import edilir.

```
from sklearn.model_selection import train_test_split
from transformers import AutoModel, AutoTokenizer
import pandas as pd
import numpy as np
from tqdm import tqdm
import torch
```

Modeli Türkçe veriler üzerinde duygu analizi yapacağı eğitmek için Türkçe dataset kullanılır.

```
In [2]: df = pd.read_csv("labelled_dataset_with_FLAN.csv")
```

```
In [3]: df.head()
```

```
Out[3]:
```

	Unnamed: 0	review	sentiment
0	0	bayiler satar artık e burası türkiye	positive
1	1	fiyatlara bin ekleyin bayide bulamayacaksınız ...	positive
2	2	motor kasaya yakışmış	positive
3	3	tl eksik aracı almam neyse sene bekleyem alırım	positive
4	4	eskiden milyoner olmak vardı derlerdi nerden g...	positive

Veri seti eğitim ve test olmak üzere 0.75:0.25 oranında ikiye ayrılır.

```
1 X_train, X_test = train_test_split(df, test_size=0.25, random_state=42)
2
3 X_test
```

		review	sentiment
36262	36262	yardımlarınız gerçekten teşekkür ederim halen ...	positive
36783	36783	kardeş ben yıldır dizel logan kullanıyorum zat...	positive
32633	32633	ben bir göz gezdirdim bin liraya çıkan modelle...	positive
14564	14564	tdi fiyat alan oldu arkadaşlar bin liste fiyat...	positive
50638	50638	hayırlı olsun	positive
...
14471	14471	model tdi coupe almayı düşünüyorum civarı km g...	negative
48558	48558	avrupa an satılıyor	positive
7831	7831	tsi polonun performansı gayet tatminkar kadar ...	positive
47091	47091	maalesef çalışmadı dizel bir modeline cc olmuy...	negative
18393	18393	tesekkur ederim beklemedeyiz	positive

16112 rows x 3 columns

Türkçe duyarlılık analizi gerçekleştirmek için önceden eğitilmiş Türkçe BERT modeli kullanılmaktadır. Önceden eğitilmiş model koda aktarılır.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_name = "dbmdz/bert-base-turkish-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
bert_model = AutoModel.from_pretrained(model_name).to(device)
```

BERT ile özellik çıkarma yapılır. BERT tokenizer, tüm metinsel verileri BERT modeli tarafından işlenebilecek tokenlara dönüştürür. Maksimum token uzunluğu belirlenir ve kalan tokenlara uzunlukları birbirine eşit olacak şekilde dolgu uygulanır. Dolgu uygulanan tokenların büyük boyutu nedeniyle hepsi aynı anda işlenemez, bu nedenle bu amaçla bir toplu iş boyutu olarak 8 seçilmiştir. İşleme giren her toplu, dolgu tokenlarını içeren bir tensöre dönüştürülür ve ardından GPU'ya aktarılır. Dikkat maskesi bölümü de, hangi tokenların dikkate alınması gerektiğini belirleyen şekilde bir tensöre dönüştürülür ve GPU'ya aktarılır. Önceden eğitilmiş bir model kullanıldığından, işlemi hızlandırmak için geri yayılım devre dışı bırakılmıştır. Her toplu için, BERT modelinin sonuçları, anlamsal anlamı belirten kısımları dizinleyerek biriktirilir.


```
def feature_extraction(df):
    tokenized = df['review'].apply((lambda x: tokenizer.encode(str(x), add_special_tokens=True)))
    max_len = 0
    for i in tokenized.values:
        if (len(i) > max_len):
            max_len = len(i)

    last_hidden_states = []
    padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
    attention_mask = np.where(padded != 0, 1, 0)
    all_hidden_states = []
    batch_size = 8
    for i in range(0, len(padded), batch_size):
        batch_input_ids = torch.tensor(padded[i:i+batch_size]).to(device)
        batch_attention_mask = torch.tensor(attention_mask[i:i+batch_size]).to(device)

        with torch.no_grad():
            last_hidden_states += list(bert_model(batch_input_ids, attention_mask=batch_attention_mask)[0][:,0,:].cpu().numpy())
    return last_hidden_states
```

```
X_train_tokens = feature_extraction(X_train)
```

```
X_test_tokens = feature_extraction(X_test)
```

Her satır için oluşturulan tokenler ve duygu değerleri, modeli eğitmek için hazırlanır. Yorumlar, feature_extraction fonksiyonunu kullanarak tensörlere dönüştürülerek model için hazırlanmıştır. Veri setinde 'negatif', 'pozitif' ve 'nötr' olarak etiketlenmiş duygular, doğru etiketin dizinine 1 değeri atanarak etiket vektörlerine dönüştürülerek eğitime hazırlanır.

```
def process_data(df,tokens):
    mapping = {'negative':0, 'neutral':1, 'positive':2}
    X = []
    y = []
    for idx, review in enumerate(tqdm(df["review"].values)):
        X.append(tokens[idx])
        y_val = np.zeros(3)
        y_val[mapping[df.iloc[idx]['sentiment']]] = 1
        y.append(y_val)
    return np.array(X), np.array(y)
```

```
X_train, y_train = process_data(X_train,X_train_tokens)
```

```
100%|██████████| 48333/48333 [00:02<00:00, 19570.64it/s]
```

```
X_test, y_test = process_data(X_test, X_test_tokens)
```

```
100%|██████████| 16112/16112 [00:00<00:00, 18975.09it/s]
```

Model eğitmek için gerekli fonksiyonlar import edilir.

```
import keras
from keras.models import Sequential,load_model
from keras.layers import Dense,Dropout,BatchNormalization,Activation
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping,ModelCheckpoint
```

Sequential() ile, her katmanı ayrı ayrı ekleyebileceğimiz bir sıralı model oluşturuyoruz. num_tokens tarafından temsil edilen tokenların uzunluğu, giriş katmanının boyutunu belirlemek için kullanılır. Veri setinde benzer yorumlar olduğundan ve verilerin çoğunluğunun pozitif duyguya sahip olduğundan, aşırı uyum riski vardır. Bu sorunu çözmek için, her katmandaki nöronların %50'sini devre dışı bırakan Dropout(0.5) katmanını dahil ediyoruz. Eğitimi hızlandırmak ve aşırı uyum riskini azaltmak için BatchNormalization() da ekliyoruz. Batch Normalization, nöronların aktivasyonunu normalize eder. Çıkış katmanında, softmax fonksiyonunu kullanarak çoklu sınıf sınıflandırması yaparak (negatif, pozitif, nötr) her duygu için bir olasılık değeri hesaplarız. Bu işlem, 3 nöron içeren bir katmanla yapılır.

```
# create model
model = Sequential()
num_tokens = len(X_train_tokens[0])
model.add(Dense(128, input_shape=(num_tokens,), activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())

model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(3, activation='softmax'))
model.summary()
```

Model oluşturulmuş olur.

```
Model: "sequential"
_____
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	98432
dropout (Dropout)	(None, 128)	0
batch_normalization (Batch Normalization)	(None, 128)	512
dense_1 (Dense)	(None, 64)	8256
dropout_1 (Dropout)	(None, 64)	0
batch_normalization_1 (Batch Normalization)	(None, 64)	256
dense_2 (Dense)	(None, 32)	2080
batch_normalization_2 (Batch Normalization)	(None, 32)	128
dropout_2 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 3)	99

```
_____
Total params: 109,763
Trainable params: 109,315
Non-trainable params: 448
```

Modeli optimize etmek için geniş çapta benimsenmiş olan Adam optimizer'ı kullanıyoruz ve öğrenme oranını 0.001 olarak belirliyoruz. Daha iyi sonuçlar elde etmek için öğrenme oranını ayarlamak potansiyel olarak mümkündür. Öğrenme oranını düşürmek, öğrenmeyi iyileştirebilir, ancak model eğitim sürecini uzatabilir. Kategorik çapraz entropi (categorical_crossentropy), çoklu sınıf sınıflandırması için uygundur.

```
optimizer = Adam(lr=1e-3)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

Training verilerindeki yorumlar birbirine benzediği için ve duyguların çoğunluğu pozitif olduğu için aşırı uyum riski bulunmaktadır. Bunun önüne geçmek için, EarlyStopping ve ModelCheckpoint'i geriçağırma işlevleri olarak kullanıyoruz. EarlyStopping, kayıp değeri 50 ardışık epoch boyunca azalmadığında 500 epoch tamamlanmadan önce eğitimi sonlandırır. ModelCheckpoint ise, eğitim süresince val_accuracy değeri en yüksek olduğunda ağırlık değerlerini 'best_model' olarak kaydeder.

```
# es = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=1)
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=50)
```

```
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
```

Eğitim, tensöre dönüştürülmüş train veri kümesinde gerçekleşecek. Eğitim süreci test_dataset ile değerlendirilecek. Her epoch'ta 64 örnek işlenecek. Veriler, her epoch başında karıştırılacak. EarlyStopping (ES) etkinleştirilmediği sürece eğitim 500 epoch boyunca devam edecek. Eğitim sonucunda en yüksek doğruluk değerine sahip model kaydedilecek.

```
history =model.fit(X_train,
                    y_train,
                    validation_data=(X_test, y_test),
                    batch_size=64,
                    shuffle=True,
                    verbose=1,
                    epochs=500,
                    callbacks=[es, mc])
```

Model eğitim süreci tamamlanır.

```
Epoch 180/500
749/756 [=====>.] - ETA: 0s - loss: 0.2664 - accuracy: 0.9094
Epoch 180: val_accuracy did not improve from 0.91193
756/756 [=====] - 4s 5ms/step - loss: 0.2663 - accuracy: 0.9094 - val_loss: 0.2968 - val_accuracy:
0.9112
Epoch 181/500
745/756 [=====>.] - ETA: 0s - loss: 0.2659 - accuracy: 0.9094
Epoch 181: val_accuracy did not improve from 0.91193
756/756 [=====] - 4s 5ms/step - loss: 0.2659 - accuracy: 0.9094 - val_loss: 0.2823 - val_accuracy:
0.9111
Epoch 181: early stopping
```

Model tekrar kullanımlar için kaydedilir.

```
! pip install h5py
```

```
WARNING:tensorflow: The current process just got forked, after parallelism has already been used. Disabling parallelism t
o avoid deadlocks...
To disable this warning, you can either:
 - Avoid using "tokenizers" before the fork if possible
 - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
Requirement already satisfied: h5py in /opt/conda/lib/python3.10/site-packages (3.9.0)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/lib/python3.10/site-packages (from h5py) (1.23.5)

Save the resulting best model for analysis
```

```
saved_model = load_model('best_model.h5')
```

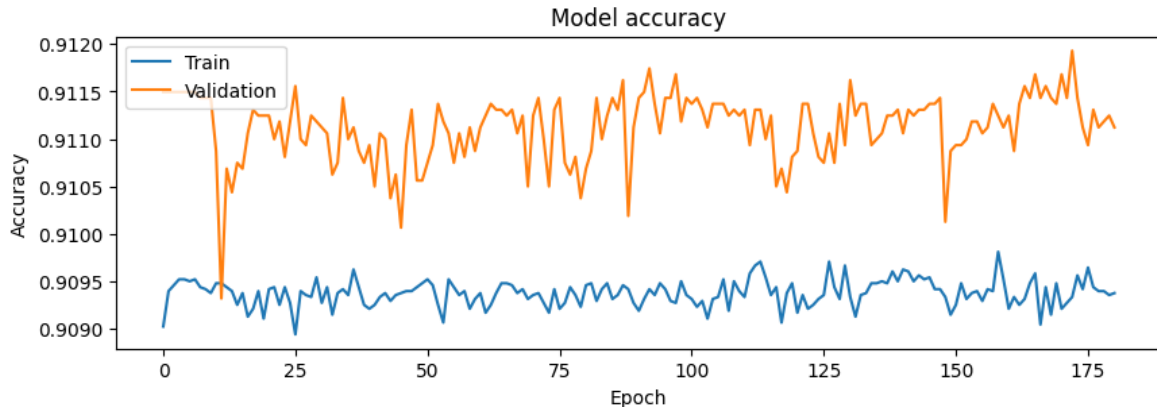
```
saved_model
```

```
<keras.engine.sequential.Sequential at 0x7fd47cdd9f60>
```

Model değerleri incelenir. Modelin doğruluk değerine accuracy ile ulaşılır.

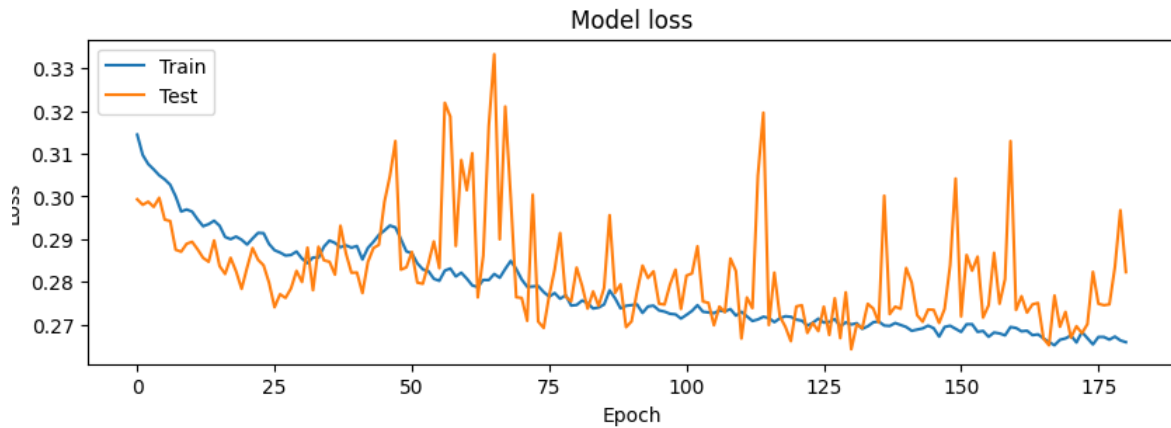
```
plt.figure(figsize=(10,3))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
```

<matplotlib.legend.Legend at 0x7fd47d72bca0>



Modelin loss değeri değerlendirilir.

```
plt.figure(figsize=(10,3))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



4.Sonuçlar

Bu rapor, büyük dil modellerinin genel bir tanımını sunarak, özellikle Transformer mimarisine odaklanmıştır. Büyük dil modellerinin çeşitli tipleri ve yapısal özellikleri üzerinde durulmuş ve NanoGPT gibi özel bir modelin incelenmesi yapılmıştır.

Ayrıca, Bert büyük dil modeli destekli Türkçe duygu analizi uygulaması ele alınmıştır. Bu uygulama, Bert modelinin kullanımını örneklemektedir ve Türkçe metinlerde duygusal analiz yapma yeteneği sunmaktadır.

Sonuç olarak, büyük dil modellerinin yapay zeka alanında önemli bir role sahip olduğunu ve çeşitli uygulamalarda başarıyla kullanılabildiğini göstermiştir. Ancak, bu modellerin eğitimi ve kullanımıyla ilgili bazı zorluklar ve sınırlamalar da vardır. Gelecekte, bu alandaki araştırmaların ve gelişmelerin devam edeceği ve büyük dil modellerinin daha da geliştirileceği öngörülmektedir.

5.Referanslar

- [1]. *Attention*. <https://nlp.seas.harvard.edu/2018/04/03/attention.html> adresinden alındı
- [2]. *Generative AI with Large Language Models*. <https://www.coursera.org/learn/generative-ai-with-llms/home/welcome> adresinden alındı
- [3]. *Introduction to LLM*. <https://medium.com/@yash9439/introduction-to-llms-and-the-generative-ai-part-1-a946350936fd> adresinden alındı
- [4]. *LLM Visualization*. <https://bbycroft.net/llm> adresinden alındı
- [5]. *Transformer_(deep_learning_architecture)*. [https://en.wikipedia.org/wiki/Transformer_\(deep_learning_architecture\)](https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)) adresinden alındı
- [6]. *Word Embedding*. <https://monkeylearn.com/blog/word-embeddings-transform-text-numbers/> adresinden alındı