Yayınlanan Veri Bilimine Doğru

Utkarsh Ankit Follow
25 Nis 2020 · 13 dakika okuma · Dinlemek

Save

Transformer Neural Network: Canavarın Adım Adım Dağılımı

kaynak: unsplash'ta arseny togulev .

Transformer Sinir Ağı, çözmeyi amaçlayan yeni bir mimaridir.diziden diziye görevleruzun menzilli bağımlılıkları kolaylıkla işlerken. *"Dikkat Tek İhtiyacınız Olan"* 2017 [1] makalesinde önerildi . NLP alanında günümüzün en gelişmiş tekniğidir.

Doğrudan Transformer'a geçmeden önce, onu neden kullandığımızı ve resme nereden geldiğini açıklamak için biraz zaman ayıracağım. (Bu kısmı atlamak

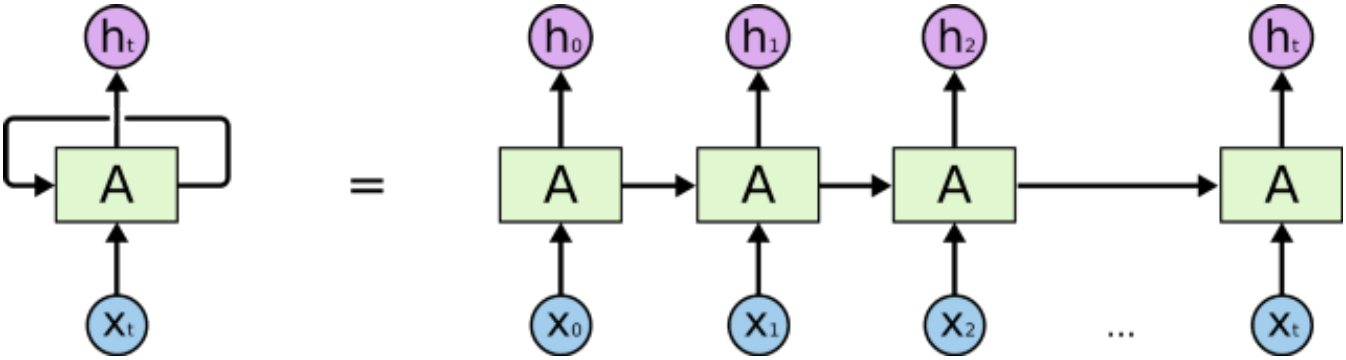
istiyorsanız doğrudan Transformer konusuna gidin, ancak daha iyi anlamak için sırayla okumanızı öneririm).

Yani hikaye RNN (Recurrent Neural Networks) ile başlıyor.

RNN

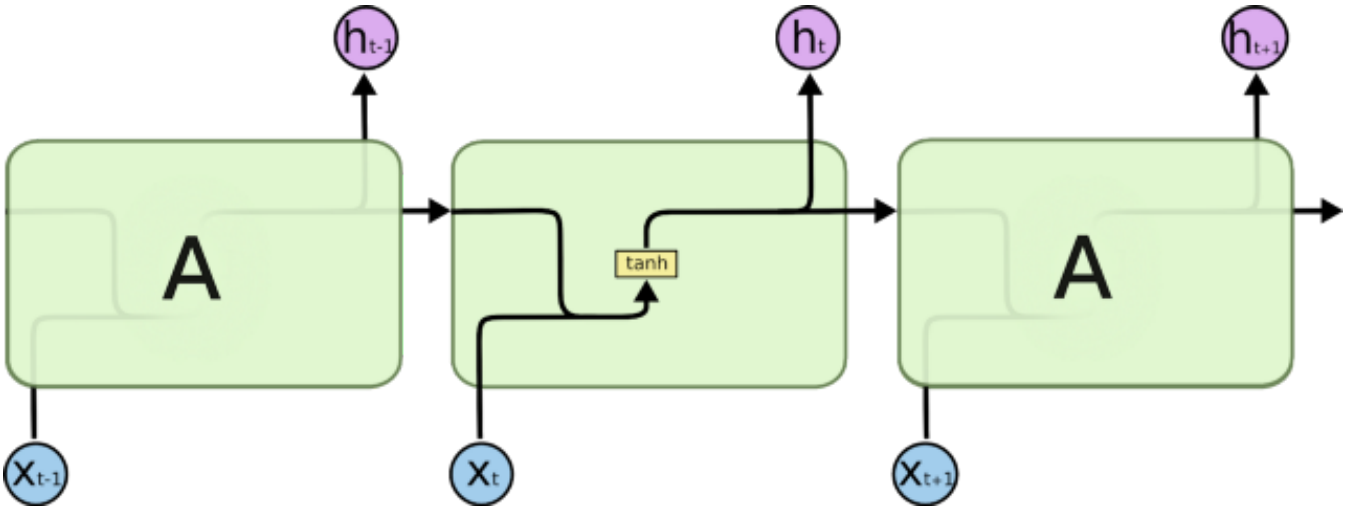
RNN nedir? Basit YSA'dan farkı nedir? En büyük fark nedir?

RNN'ler, zaman içinde kullanıma sunulan İleri Beslemeli Sinir Ağlarıdır.



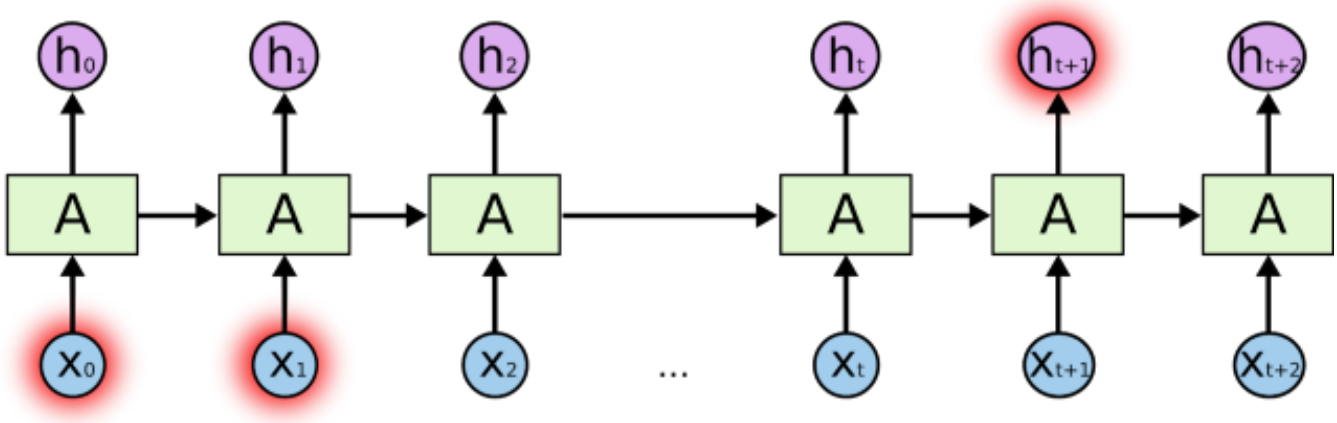
kaynak: cola's (CC0).

Normal Sinir Ağlarından farklı olarak, RNN'ler, önceden belirlenmiş bir boyut sınırı olmaksızın bir **dizi girdi** alacak şekilde tasarlanmıştır . Bu dizinin herhangi bir girdisinde olduğu gibi "dizi", komşularıyla bir tür ilişkiye sahiptir veya onlar üzerinde bir etkiye sahiptir.



RNN mimarisi. kaynak: cola's (CC0).

Temel ileri beslemeli ağlar da bazı şeyleri "hatırlar", ancak eğitim sırasında öğrendiklerini hatırlarlar. RNN'ler eğitim sırasında benzer şekilde öğrenirken, ek olarak çıktı(lar) üretirken önceki girdi(ler)den öğrenilenleri hatırlar.



Uzun vadeli bağımlılıkları gösteren resim. kaynak: cola's (CC0).

Farklı modellerde kullanılır-

1.) **Vektör-Dizi Modelleri**- Sabit boyutlu vektörleri herhangi bir uzunluktaki girdi ve çıktı vektörleri olarak alırlar, örneğin resim alt yazısında görüntü bir girdi olarak verilir ve çıktı görüntüyü tanımlar.

2.) **Sekans-Vektör Modeli**- Herhangi bir boyutta bir vektör alın ve sabit boyutta bir vektör çıktısı alın. Örneğin. Bir filmin duygu analizi, herhangi bir filmin incelemesini sabit boyutlu bir vektör olarak olumlu veya olumsuz olarak derecelendirir.

3.) **Diziden Diziye Model**- En popüler ve en çok kullanılan varyant, girdiyi bir dizi olarak alır ve değişken boyutlarıyla başka bir dizi olarak çıktı verir. Örneğin. Borsa tahmini için zaman serisi verileri için dil çevirisi.

Dezavantajları-

1. Eğitmek için yavaş.
2. Uzun dizi, yok olan gradyanlara veya örneğin uzun vadeli bağımlılıklar sorununa yol açar. Basit bir ifadeyle, iş eski bağlantıyı hatırlamaya geldiğinde hafızası o kadar güçlü değildir.

Örn . _ "Bulutlar ____ içinde."

Bulutlarla bağlantılı olduğu için bir sonraki kelimenin gökyüzü olacağı açıktır. Burada bulutlar ile tahmin edilen kelime arasındaki mesafenin daha az olduğunu görüyoruz, böylece RNN kolayca tahmin edebiliyor.

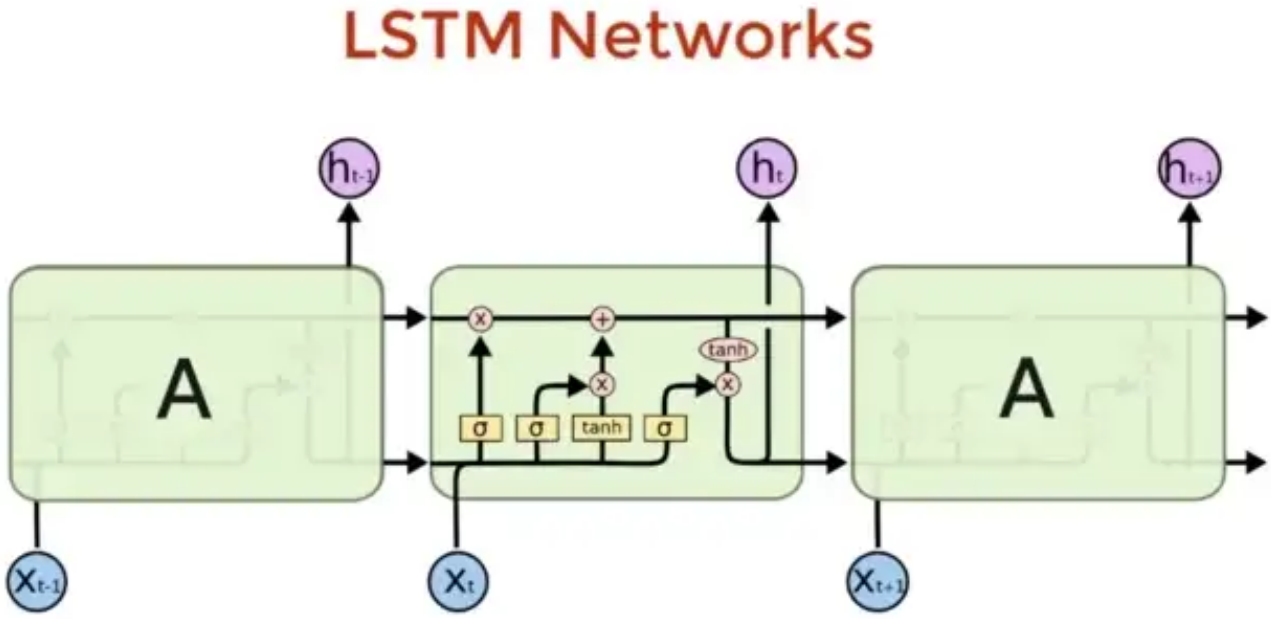
Ama başka bir örnek için,

"Almanya'da ailemle birlikte büyüdüm, uzun yıllar geçirdim ve kültürleri hakkında doğru bilgiye sahibim, bu yüzden akıcı ____ konuşuyorum."

Burada tahmin edilen kelime, Almanya ile doğrudan bağlantılı olan Almanca'dır ancak Almanya ile tahmin edilen kelime arasındaki mesafe bu durumda daha fazladır, bu nedenle RNN'nin tahmin etmesi zordur.

Ne yazık ki, bu boşluk büyüdükçe, RNN'ler bağlanamaz hale gelir, çünkü bellekleri mesafeyle birlikte zayıflar.

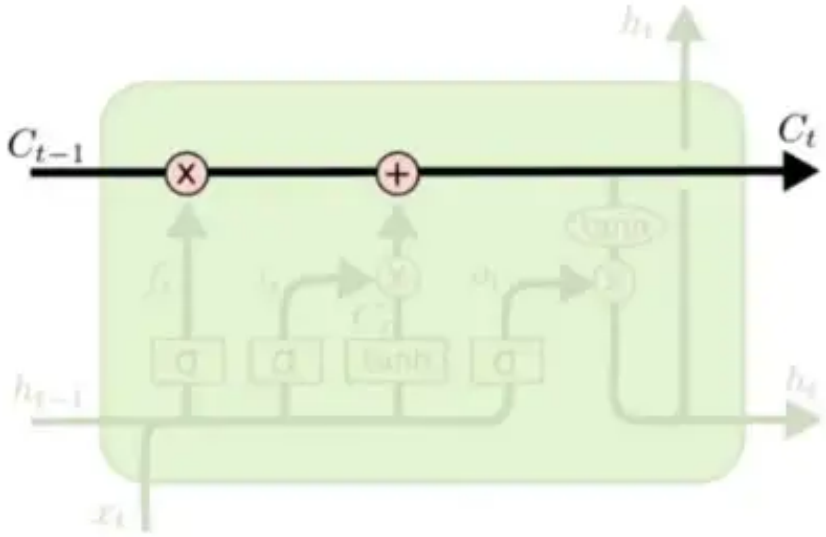
LSTM



The repeating module in an LSTM contains four interacting layers.

kaynak: cola's (CC0).

Uzun Kısa Süreli Bellek - Kaybolan gradyan problemlerini çözmek için özel olarak yapılmış özel bir RNN türü. Uzun Süreli Bağımlılıkları öğrenme yeteneğine sahiptirler. Bilgileri uzun süre hatırlamak, öğrenmek için mücadele ettikleri bir şey değil, pratik olarak varsayılan davranışlarıdır!



Bu dal, bilgi iletmeye ve hücrenin uzun süre işlenmesini atlamaya izin verir. kaynak: cola's (CC0).

LSTM Nöronları, normal nöronların aksine, bilgi iletmeye ve mevcut hücrenin uzun işlemlerini atlamaya izin veren bir dala sahiptir, bu, hafızanın daha uzun süre tutulmasına izin verir. Kaybolan gradyan probleminin durumunu iyileştiriyor ama o kadar da şaşırtıcı değil, 100 kelimeye kadar işe yarayacak gibi, ama 1000 kelime gibi, kavramasını kaybetmeye başlıyor.

Ancak basit RNN gibi eğitilmesi de çok yavaştır, hatta daha yavaştır.

Paralel hesaplama için tasarlanmış GPU'ları çok iyi kullanamayan, sırayla tek tek girdi alırlar.

Sıralı verileri nasıl paralel hale getirebiliriz? *(Bu soruya geri döneceğim.)*

Şimdilik, iki sorunla uğraşıyoruz-

- kaybolan gradyan
- Yavaş eğitim

Kaybolan gradyan sorununu çözme:

Dikkat

Girdinin hangi kısmına odaklanmamız gerektiği sorusuna cevap verir.

Dikkati biraz farklı bir şekilde açıklayacağım. Bir durumu ele alalım-

Birinin bize bir makine öğrenimi kitabı verdiğini ve bizden kategorik çapraz entropi hakkında bilgi vermemizi istediğini varsayalım. Bunu yapmanın iki yolu var, önce tüm kitabı okuyun ve cevapla geri dönün. İkinci olarak dizine gidin, 'kayıplar' bölümünü bulun, çapraz entropi bölümüne gidin ve Kategorik Çapraz Entropi bölümünü okuyun.

Sizce daha hızlı yöntem nedir?

İlk yöntemde olduğu gibi, tüm kitabı okumak bir haftayı bulabilir. İkinci sıradayken, neredeyse 5 dakika sürmez. Ayrıca, birinci yöntemdeki bilgilerimiz çok fazla bilgiye dayandığı için daha belirsiz ve çeşitli olurken, ikinci yöntemdeki bilgiler ihtiyaca göre doğru olacaktır.

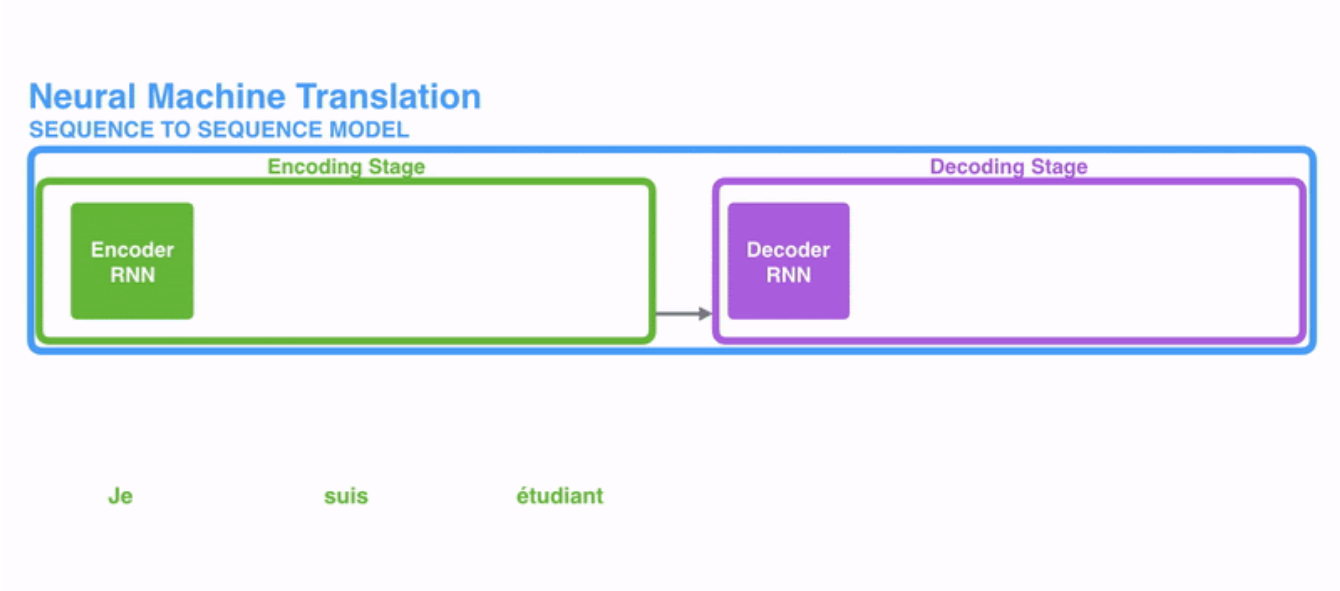
Burada neyi farklı yaptık?

İlk durumda, kitabın herhangi bir kısmına özel olarak odaklanmadık, oysa ikinci durumda, dikkatimizi kayıplar bölümüne odakladık ve ardından dikkatimizi, Kategorik Çapraz kavramının yer aldığı çapraz entropi kısmına odakladık. Entropi açıklanır. Aslında, çoğu insanın yapacağı yol budur.

Sinir ağlarındaki dikkat, insanlarda bulduğumuza biraz benzer. Girişlerin belirli kısımlarında yüksek çözünürlüğe odaklanırken, girişin geri kalanı düşük çözünürlüktedir [2].

Diyelim ki bir NMT(Neural Machine Translator) yapıyoruz,

Bu animasyona göz atın, bu basit bir ardışık sıralı modelin nasıl çalıştığını gösterir.

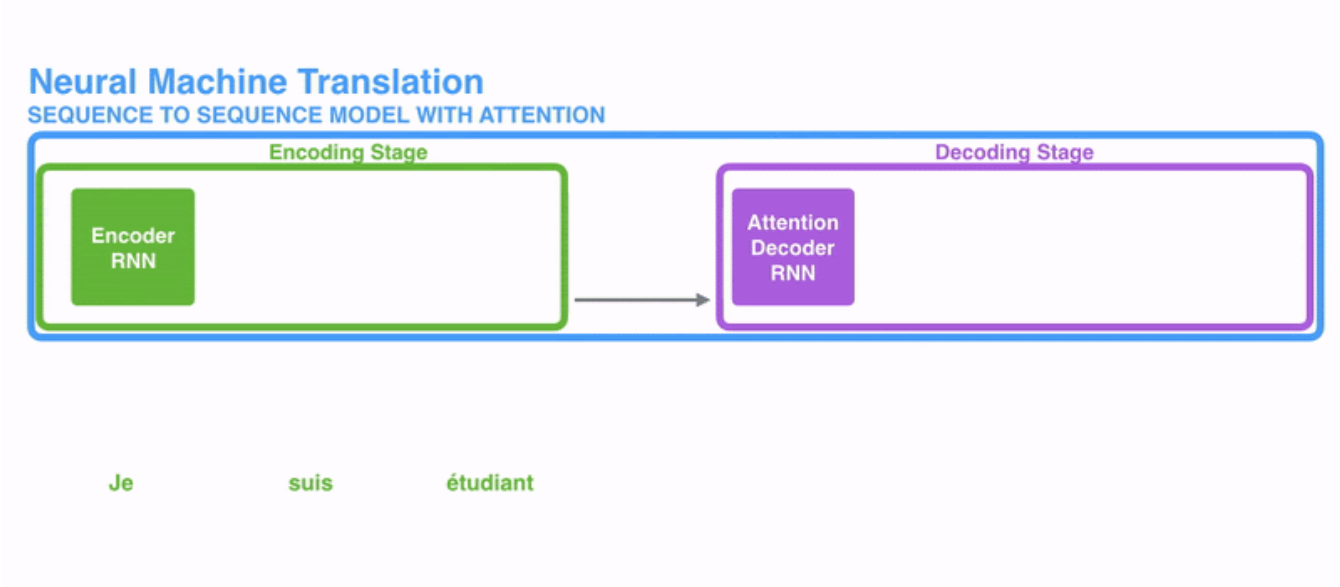


Klasik bir Seq-to-Seq modelinin çalışması. kaynak: jalammar's (CC BY-NC-SA 4.0).

Kodlayıcı veya kod çözücünün her adımı için RNN'nin girdilerini işlediğini ve o zaman adımı için çıktı ürettiğini görüyoruz. Her zaman adımında RNN, gördüğü girdilere ve önceki çıktılara göre gizli durumunu günceller. Animasyonda gizli durumun aslında kod çözücüye ilettiğimiz **bağlam vektörü olduğunu görüyoruz.**

"Dikkat" zamanı .

Bağlam **vektörünün** bu tür modeller için sorunlu olduğu ortaya çıktı. Modeller uzun cümlelerle uğraşırken sorun yaşarlar. Ya da uzun cümlelerde yok olan gradyan sorunuyla karşılaştıklarını söyleyin. Böylece, bir makalede [2] bir çözüm ortaya çıktı, Dikkat tanıtıldı. Modelin, gerektiğinde girdi dizisinin ilgili kısmına odaklanmasına izin verdiği için makine çevirisinin kalitesini büyük ölçüde iyileştirdi.



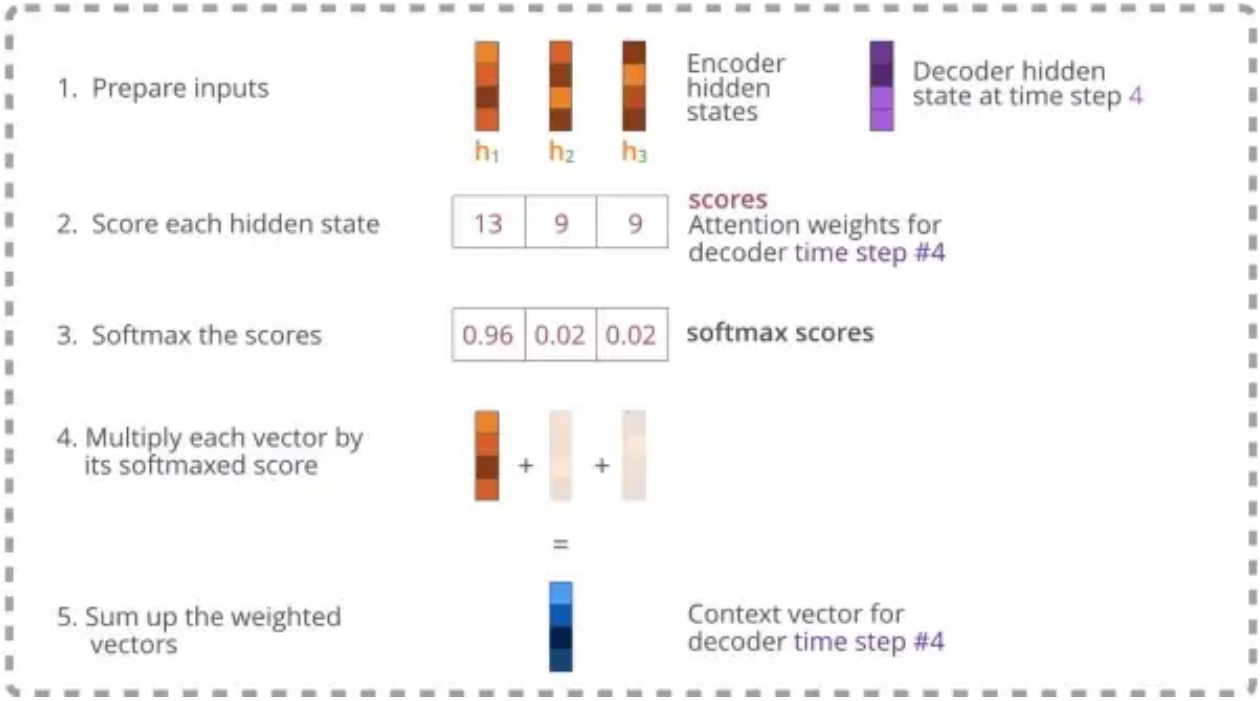
Bir Seq-to-Seq Modelinin Dikkatle Çalışması. kaynak: jalammar's (CC BY-NC-SA 4.0).

Bu dikkat modeli, klasik sıralı modelden iki şekilde farklıdır:

- Basit bir ardışık modele kıyasla burada kodlayıcı, kod çözücüye çok daha fazla veri aktarır. Önceden, kodlama bölümünün yalnızca son, son gizli durumu kod çözücüye gönderilir, ancak şimdi kodlayıcı tüm gizli durumları (hatta ara durumları) kod çözücüye iletir.
- Kod çözücü kısmı, çıktısını üretmeden önce fazladan bir adım atıyor. Aşağıda açıklanan-

Kod çözücülerin son adımı şu şekilde ilerler:

1. Kodlayıcının her gizli durumu çoğunlukla giriş cümlesindeki belirli bir kelimeyle ilişkilendirildiğinden, aldığı her gizli durumu kontrol eder.
2. Her gizli duruma bir puan veriyorum.
3. Daha sonra her puan ilgili softmax puanıyla çarpılır, böylece yüksek puanlarla gizli durumlar yükseltilir ve düşük puanlarla gizli durumlar boğulur. *(Net görselleştirme için aşağıdaki resme bakın.)*



kaynak: jalammar's (CC BY-NC-SA 4.0).

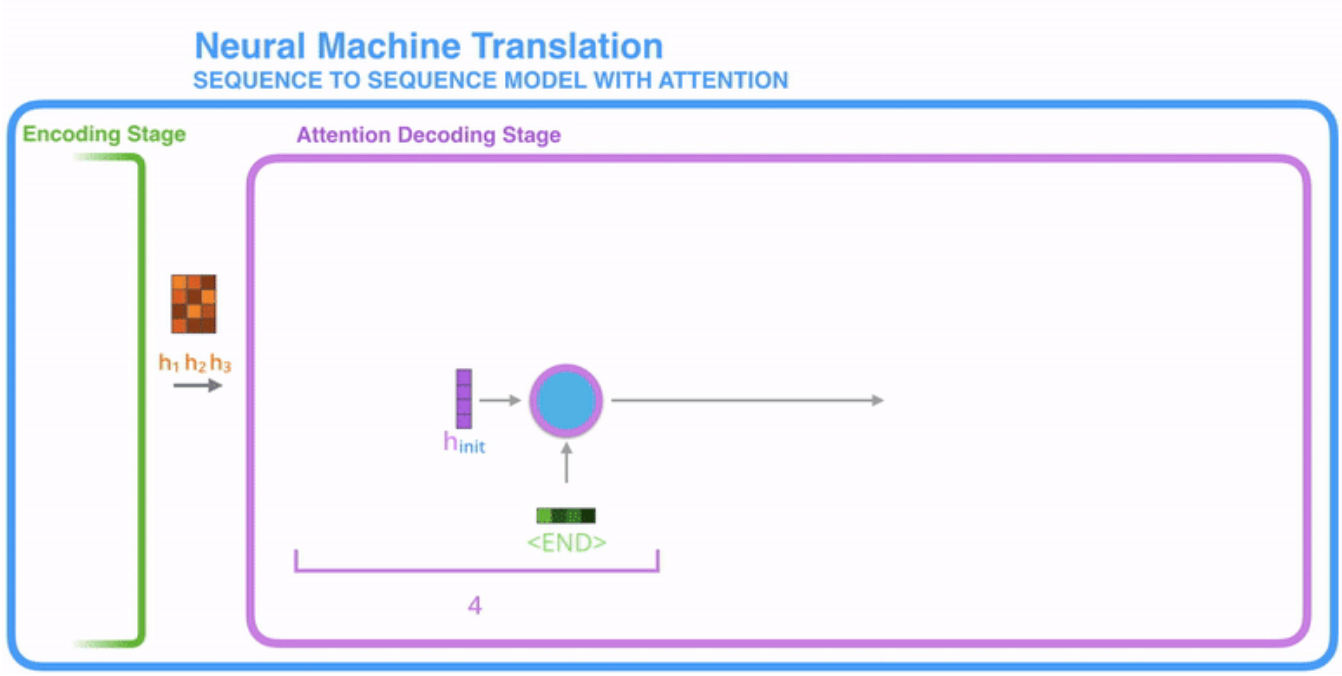
Bu puanlama alıştırması, kod çözücü tarafında her zaman adımında yapılır.

Şimdi her şeyi bir araya getirdiğimizde:

1. Dikkat kod çözücü katmanı, <END> belirtecinin gömülmesini alır ve bir ilk kod çözücü gizli durumu, RNN girdisini işler ve bir çıktı ve yeni bir gizli durum vektörü (h4) üretir.
2. Şimdi, bu zaman adımı için bir bağlam vektörü C4'ü hesaplamak için kodlayıcı gizli durumlarını ve h4 vektörünü kullanıyoruz. Dikkat kavramının uygulandığı yer burasıdır, bu yüzden buna Dikkat Adımı denir.
3. (h4) ve C4'ü tek bir vektörde birleştiriyoruz.

4. Şimdi, bu vektör bir ileri beslemeli sinir ağına geçirilir, ileri beslemeli sinir ağlarının çıktısı bu zaman adımının çıktı kelimesini gösterir.

5. Bu adımlar bir sonraki zaman adımları için tekrarlanır. *(Net görselleştirme için aşağıdaki slayda bakın.)*



Son Adım. kaynak: jalammar's (CC BY-NC-SA 4.0).

Dikkat böyle çalışır.

Örneğin. Bir Resim Yazısı Probleminde Dikkat Çalışması: -

Bir Resim Yazısı Probleminde dikkatin çalışması. kaynak: CodeEmporium (CC0).

Şimdi, daha önce belirttiğim soruyu hatırlayın-

Sıralı verileri nasıl paralel hale getirebiliriz?

İşte cephanemiz geliyor-

Transformatörler

2017'de yayınlanan "Attention Is All You Need" adlı bir makale devreye giriyor, dikkat katmanlarına dayanan ve dönüştürücü olarak adlandırılan bir kodlayıcı-kod çözücü mimarisini tanıtıyor.

Temel farklardan biri, giriş dizisinin paralel olarak geçirilebilmesi, böylece GPU'nun etkin bir şekilde kullanılabilmesi ve eğitim hızının da artırılabilmesidir. Ve çok başlı dikkat katmanına dayalıdır, kaybolan gradyan sorunu da büyük bir farkla aşılır. Makale, NMT'de (Neural Machine Translator) transformatör uygulamasına dayanmaktadır.

Yani, burada daha önce altını çizdiğimiz her iki problemimiz de burada bir dereceye kadar çözüldü.

Örneğin, basit RNN'den oluşan bir tercümanda olduğu gibi, kelime gömmelerini oluşturmak için her seferinde bir kelime olmak üzere dizimizi veya cümleyi sürekli bir şekilde giriyoruz. Her kelime bir önceki kelimeye bağlı olduğu için gizli hali buna göre hareket eder, bu nedenle her seferinde bir adım vermek gerekir. Transformatörde ise öyle değil, bir cümlenin tüm kelimelerini aynı anda geçirebiliyoruz ve aynı anda gömülen kelimeyi belirleyebiliyoruz. Peki, aslında nasıl çalışıyor, ileriyi görelim-

Mimarlık:-

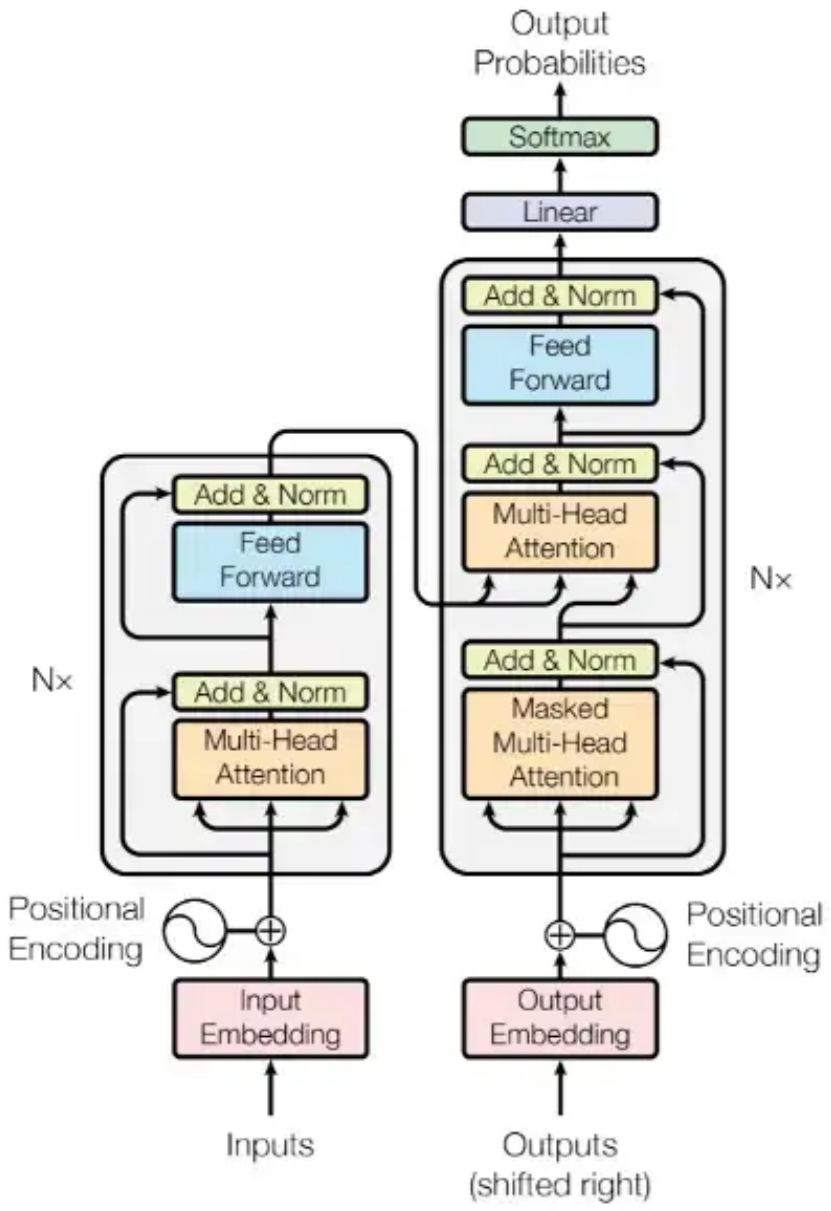
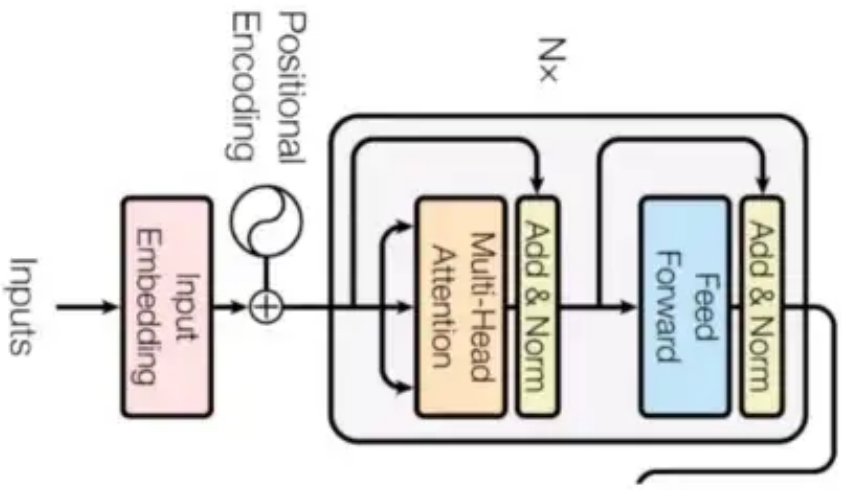


Figure 1: The Transformer - model architecture.

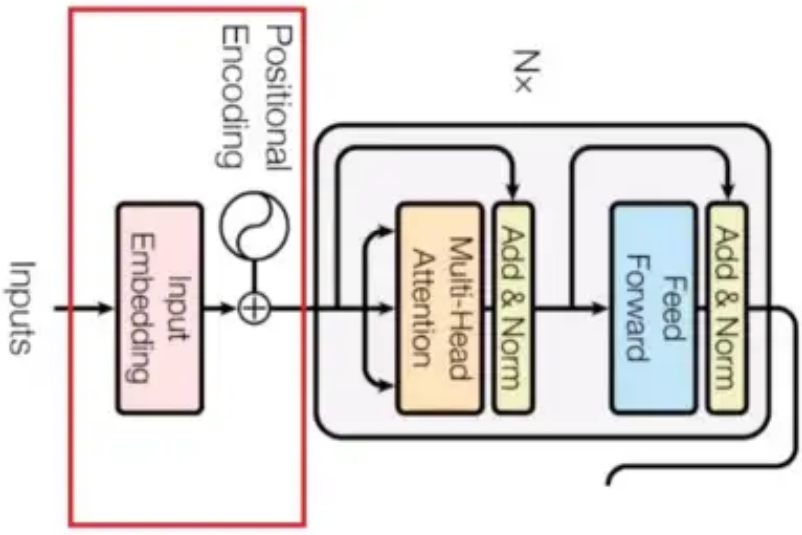
kaynak: arXiv:1706.03762 [cs.CL].

> *1. Kodlayıcı Bloğu -*



kaynak: arXiv:1706.03762 [cs.CL].

Bilgisayarların sözcükleri anlamadığı, sayılar, vektörler veya matrisler üzerinde çalıştığı bir gerçektir. Yani, kelimelerimizi bir vektöre dönüştürmemiz gerekiyor. Ama bu nasıl mümkün olabilir. **İşte Gömme Alanı** kavramı burada devreye giriyor. Benzer anlamlara sahip kelimelerin bir araya toplandığı veya o boşlukta birbirine yakın olarak bulunduğu açık bir alan veya sözlük gibidir. Bu boşluk, gömme boşluk olarak adlandırılır ve burada her kelime, anlamına göre haritalanır ve belirli bir değerle atanır. Yani, burada kelimelerimizi vektörlere çeviriyoruz.



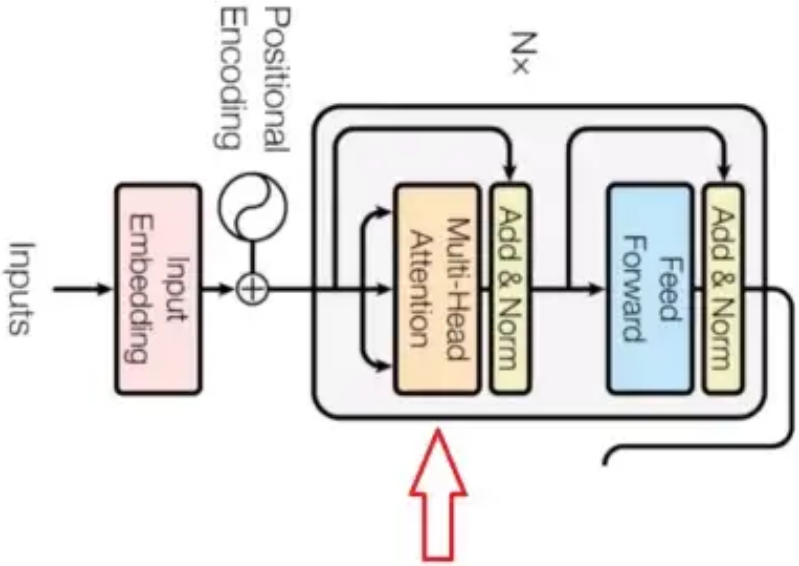
kaynak: arXiv:1706.03762 [cs.CL].

Ancak karşılaşacağımız bir diğer konu da, farklı cümlelerdeki her kelimenin farklı anlamlara sahip olmasıdır. **Dolayısıyla, bu sorunu çözmek için Konumsal Kodlayıcılardan** yardım alıyoruz . Kelimenin cümle içindeki konumuna göre bağlam veren bir vektördür.

Kelime → Gömme → Konumsal Gömme → Nihai Vektör, Bağlam olarak adlandırılır.

Yani girdimiz hazır, artık kodlayıcı bloğuna gidiyor.

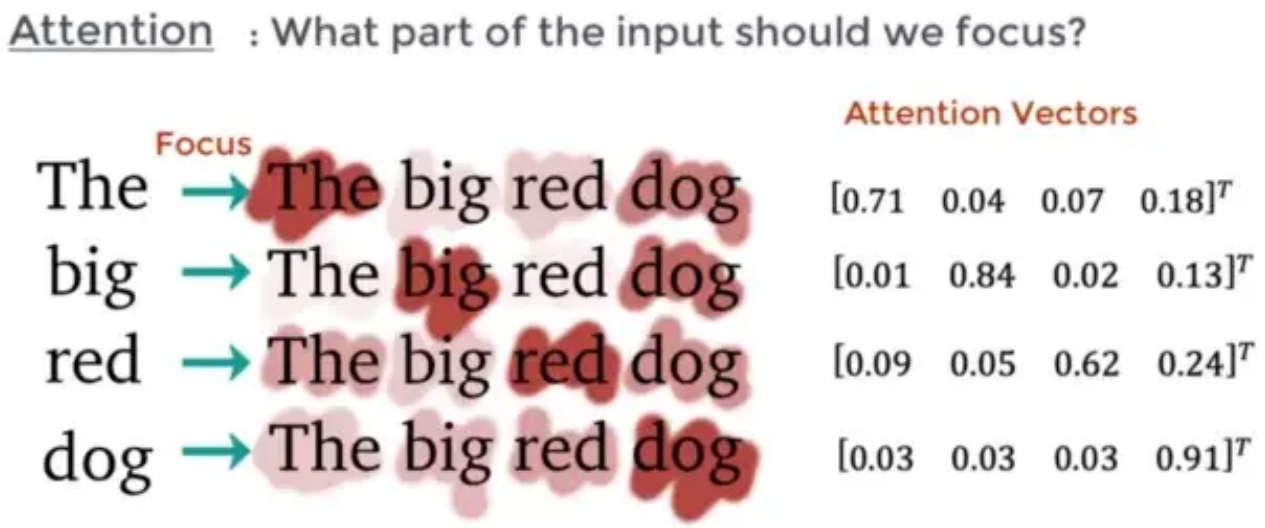
> *Çok Başlı Dikkat Parçası -*



kaynak: arXiv:1706.03762 [cs.CL].

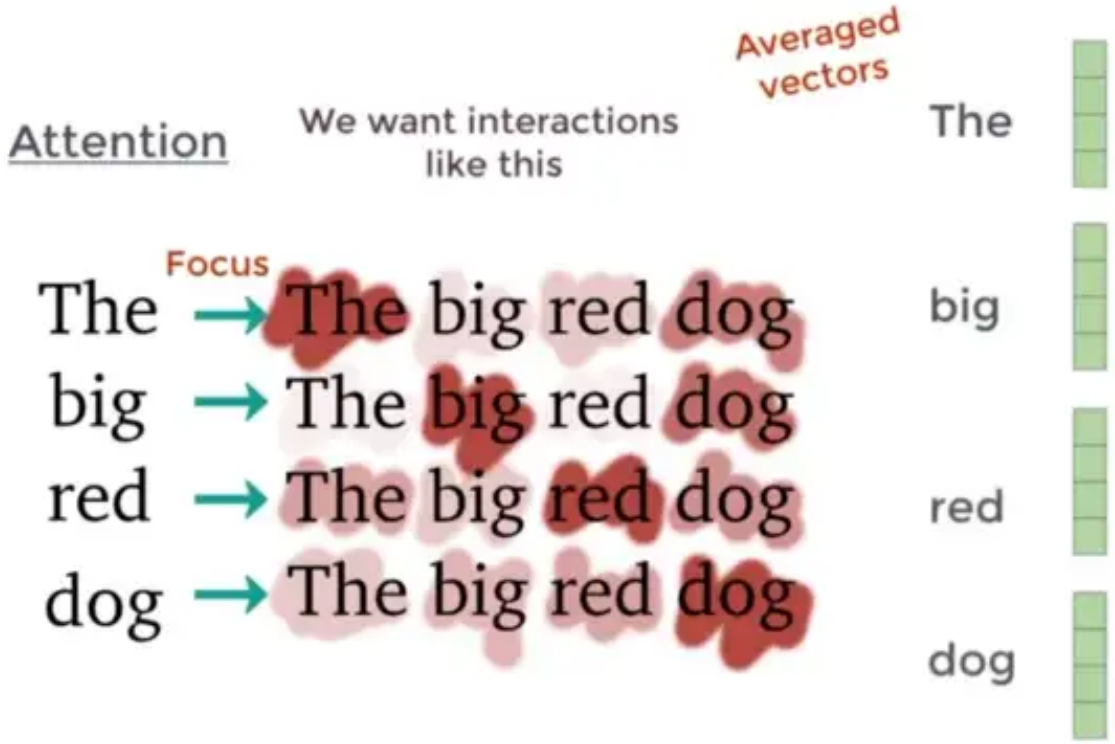
Şimdi transformatörün ana özü, "Öz Dikkat" geliyor.

Belirli bir kelimenin o cümledeki diğer kelimelerle ne kadar alakalı olduğu kısmına odaklanır. Bir dikkat vektörü olarak temsil edilir. Her kelime için, o cümledeki kelimeler arasındaki bağlamsal ilişkiyi yakalayan bir dikkat vektörü üretebiliriz.



kaynak: CodeEmporium (CC0).

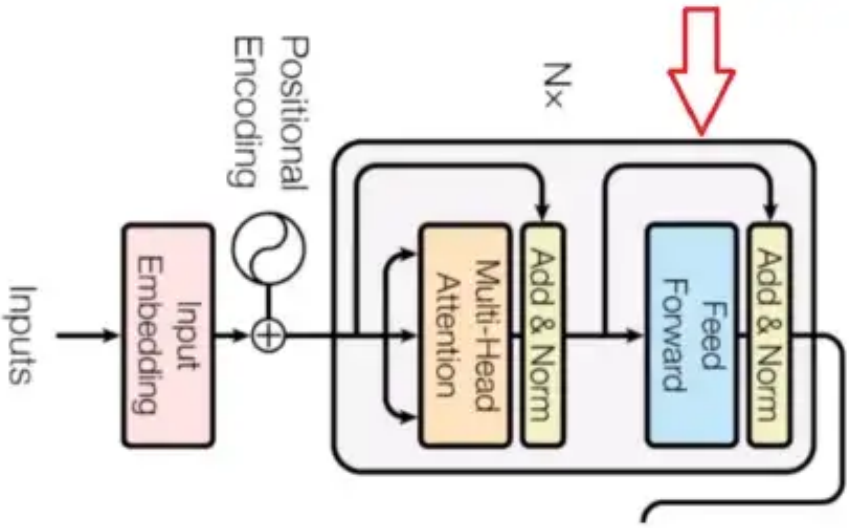
Karşılaştığı tek sorun, o cümlenin diğer kelimeleriyle etkileşime girmeye meyilli olsak da, cümledeki her kelimenin değerini kendi başına çok daha ağır basmasıdır. Bu nedenle, her kelime için birden çok dikkat vektörü belirliyoruz ve her kelimenin son dikkat vektörünü hesaplamak için ağırlıklı bir ortalama alıyoruz.



kaynak: CodeEmporium (CC0).

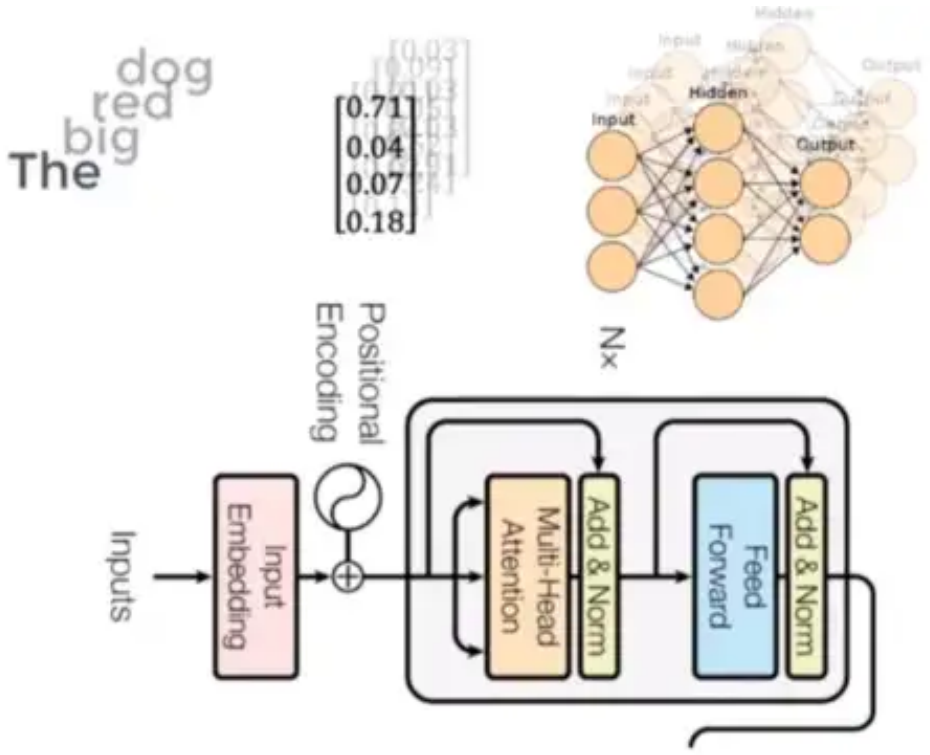
Birden çok dikkat vektörü kullandığımız için buna **Çok Başlı Dikkat Bloğu** denir .

> *İleri Besleme Ağı -*



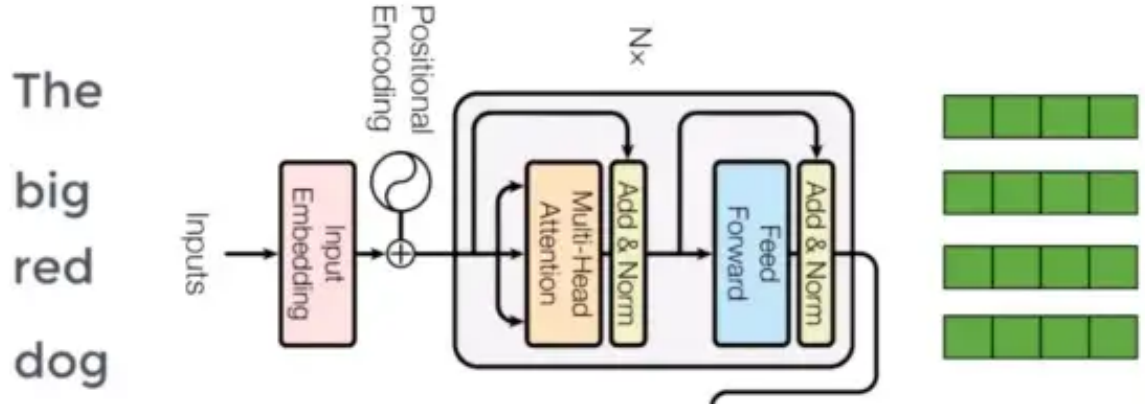
kaynak: arXiv:1706.03762 [cs.CL].

Şimdi, ikinci adım İleri Beslemeli Sinir Ağı. Bu, her dikkat vektörüne uygulanan basit ileri beslemeli Sinir Ağı'dır, asıl amacı dikkat vektörlerini bir sonraki kodlayıcı veya kod çözücü katmanı tarafından kabul edilebilir bir forma dönüştürmektir.



kaynak: arXiv:1706.03762 [cs.CL].

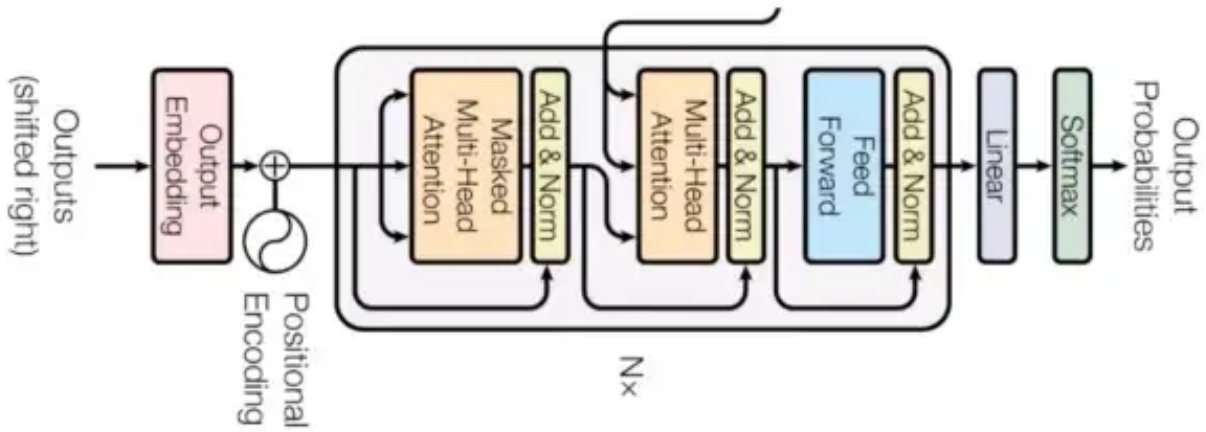
İleri Besleme Ağı, dikkat vektörlerini "birer birer" kabul eder. Ve buradaki en iyi şey, RNN'den farklı olarak, burada bu dikkat vektörlerinin her biri birbirinden *bağımsızdır* . Dolayısıyla burada **paralelleştirme** uygulanabilir *ve tüm farkı yaratan da budur* .



Enkoder Çıkışı. kaynak: arXiv:1706.03762 [cs.CL].

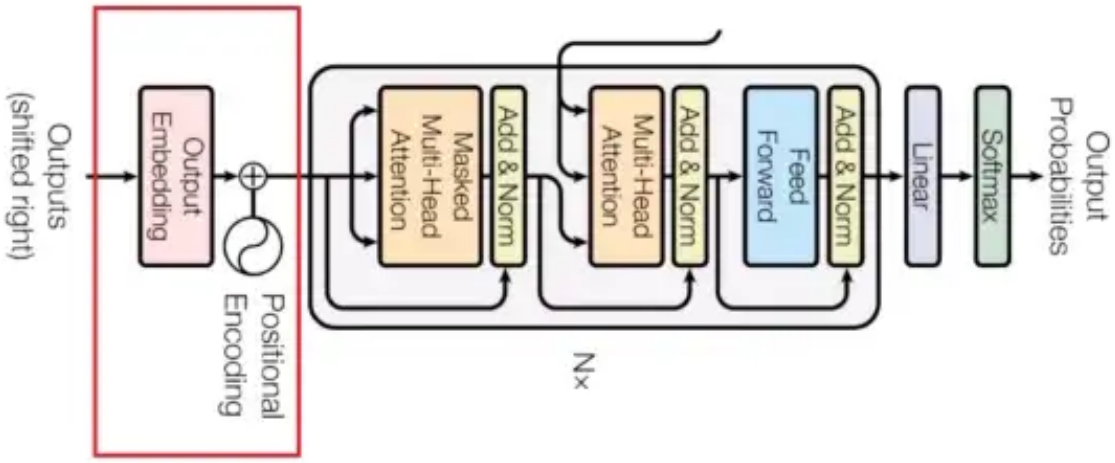
Artık tüm kelimeleri aynı anda kodlayıcı bloğuna geçirebilir ve her kelime için aynı anda Kodlanmış Vektörler setini alabiliriz.

> *2. Kod Çözücü Bloğu -*



kaynak: arXiv:1706.03762 [cs.CL].

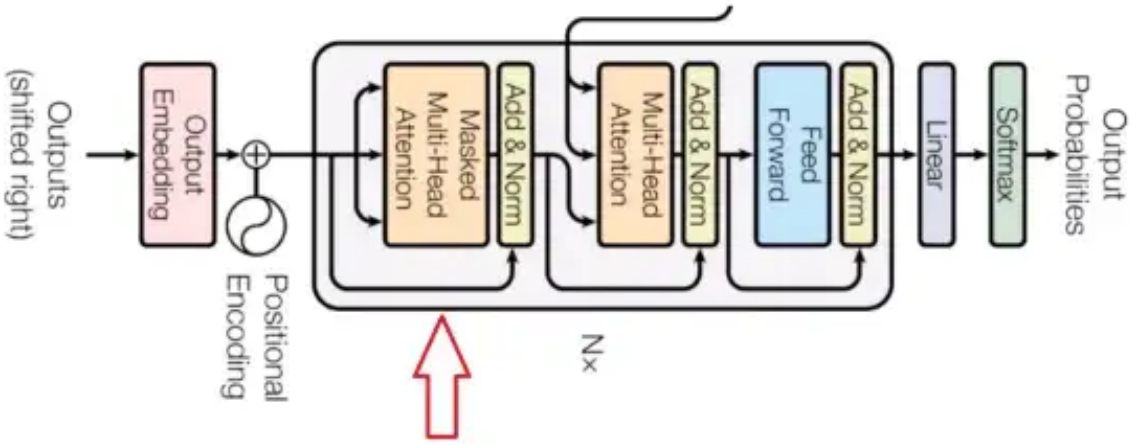
Şimdi, İngilizce için Fransızca'ya çevirmen yetiştiriyormuşuz gibi, bu nedenle eğitim için modelin öğrenmesi için çevrilmiş Fransızca cümlesiyle birlikte İngilizce bir cümle vermemiz gerekiyor. Yani İngilizce cümlelerimiz Encoder Block'tan, Fransızca cümleler Decoder Block'tan geçer.



kaynak: arXiv:1706.03762 [cs.CL].

İlk başta, kelimeleri ilgili vektörlere dönüştüren Gömme katmanı ve Konumsal kodlayıcı bölümümüz var. Bu, Kodlayıcı bölümünde gördüğümüze benzer.

> *Maskeli Çok Başlı Dikkat Parçası -*



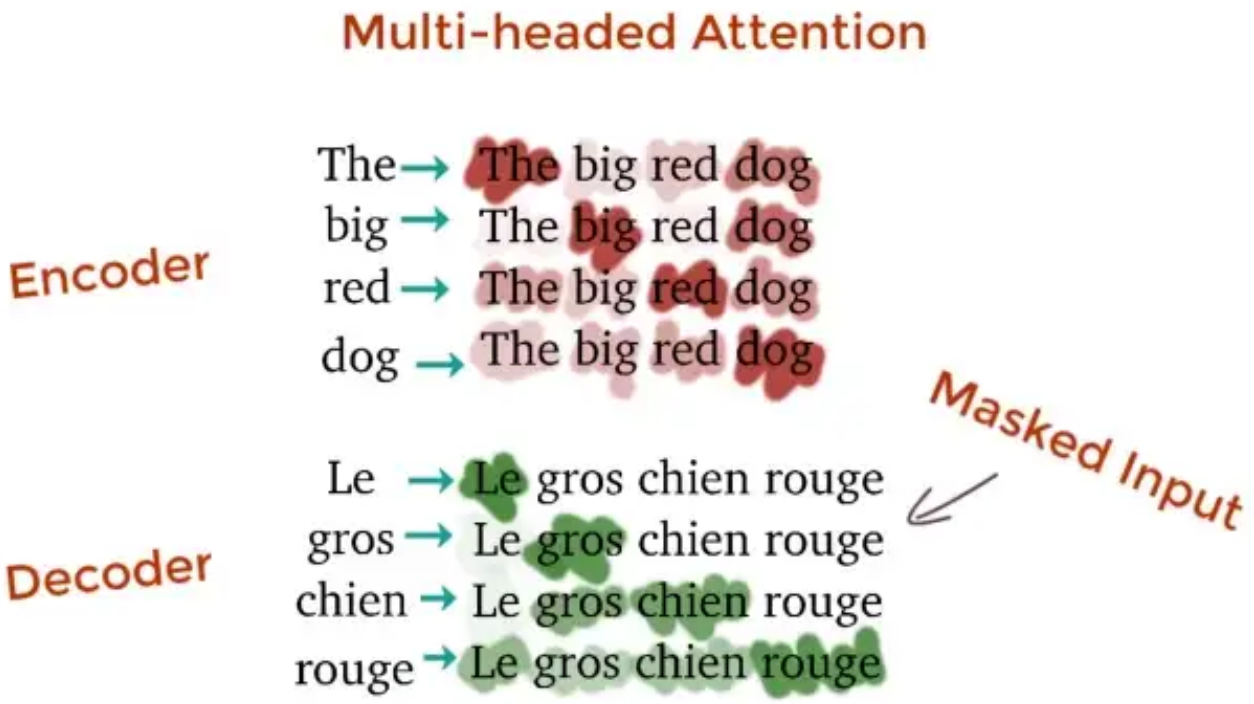
kaynak: arXiv:1706.03762 [cs.CL].

Şimdi, Fransızca cümlelerdeki her kelime için aynı cümledeki her kelimeyle ne kadar ilişkili olduğunu temsil eden dikkat vektörlerinin üretildiği kendine dikkat bloğundan geçecek. (Tıpkı Encoder Kısmında gördüğümüz gibi).

Ancak bu bloğa **Maskeli Çok Başlı Dikkat Bloğu** denir ve basit terimlerle açıklayacağım.

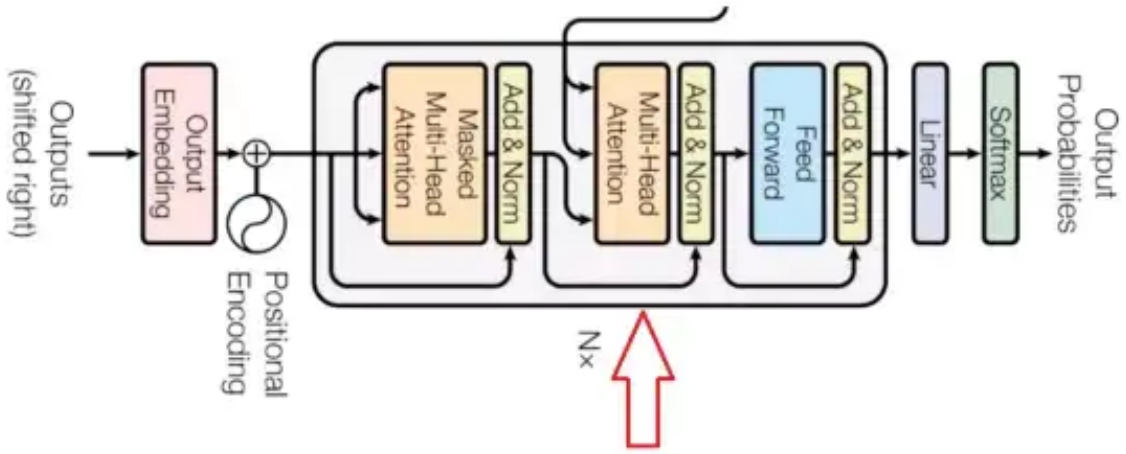
Bunun için öğrenme mekanizmasının nasıl çalıştığını bilmemiz gerekiyor. İlk olarak, İngilizce bir kelime veriyoruz, önceki sonuçları kullanarak Fransızca versiyonunda tercüme edecek, ardından (kod çözücü bloğunda beslediğimiz) gerçek Fransızca tercümeyle eşleşecek ve karşılaştıracaktır. Her ikisini de karşılaştırdıktan sonra matris değerini güncelleyecektir. Birkaç yinelemeden sonra bu şekilde öğrenecektir.

Gözlemlediğimiz şey, bir sonraki Fransızca kelimeyi gizlememiz gerektiğidir, böylece ilk önce gerçek çevrilmiş kelimeyi bilmeden önceki sonuçları kullanarak bir sonraki kelimenin kendisini tahmin edecektir. Öğrenmenin gerçekleşmesi için, bir sonraki Fransızca kelimeyi zaten biliyorsa hiçbir anlamı olmayacaktır. Bu nedenle, onu saklamamız (maskelememiz) gerekiyor.



Bu İngilizce-Fransızca Çeviri örneğidir. kaynak: CodeEmporium (CC0).

İngilizce cümleden herhangi bir kelimeyi alabiliriz ama öğrenmek için sadece Fransızca cümlenin önceki kelimesini alabiliriz. Böylece matris işlemi ile paralelleştirmeyi yaparken matrisin sonradan gelen kelimeleri 0'a çevirerek **maskelemesini** dikkat ağının kullanmamasını sağlıyoruz.



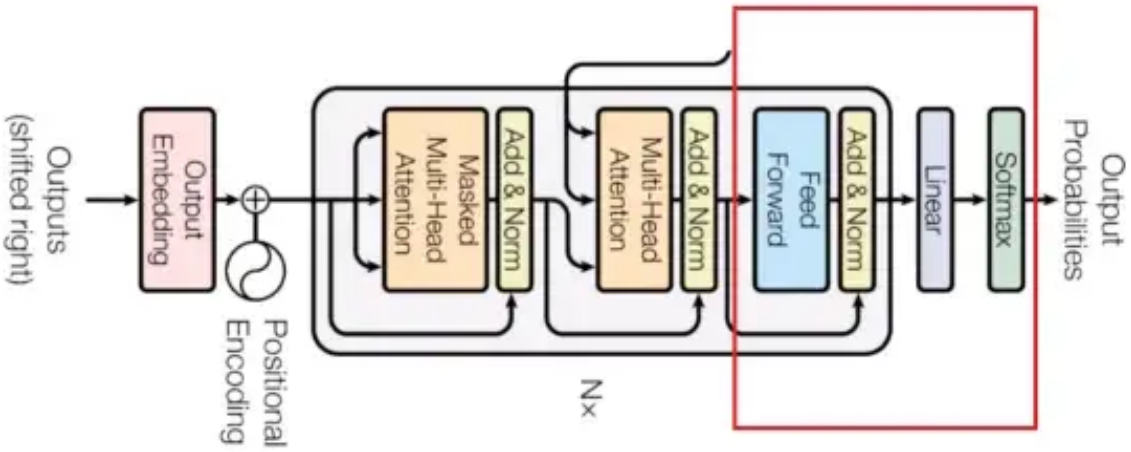
kaynak: arXiv:1706.03762 [cs.CL].

Şimdi, önceki katmandan elde edilen dikkat vektörleri ve Kodlayıcı Bloğundan gelen vektörler, başka bir **Çok Başlı Dikkat Bloğuna** geçirilir . (*Encoder bloğundan*

gelen sonuçların da devreye girdiği kısım burasıdır. Diyagramda ayrıca Encoder bloklarından gelen sonuçların buraya geldiği açıkça görülmektedir.). Bu yüzden **Kodlayıcı-Kod Çözücü Dikkat Bloğu** olarak adlandırılır .

Her İngilizce ve Fransızca cümle için her kelimenin bir vektörüne sahip olduğumuz için. Bu blok aslında İngilizce ve Fransızca kelimelerin eşlemesini yapar ve aralarındaki ilişkiyi bulur. Bu, İngilizce'den Fransızca'ya ana kelime eşlemenin gerçekleştiği kısımdır.

Bu bloğun çıktısı, İngilizce ve Fransızca cümlelerdeki her kelime için dikkat vektörleridir. *Her vektör, her iki dildeki diğer kelimelerle olan ilişkiyi temsil eder* .



kaynak: arXiv:1706.03762 [cs.CL].

Şimdi her bir dikkat vektörünü bir ileri besleme birimine aktarıyoruz, bu, çıktı vektörlerinin başka bir kod çözücü bloğu veya doğrusal bir katman tarafından kolayca kabul edilebilir bir şeye dönüşmesini sağlayacaktır.

Doğrusal bir katman, başka bir ileri beslemeli katmandır. Çeviriden sonra Fransızca dilinde boyutları kelime sayılarına genişletmek için kullanılır.

Şimdi, girişi insan tarafından yorumlanabilen bir olasılık dağılımına dönüştüren bir Softmax Katmanından geçirilir.

Ve ortaya çıkan kelime, çeviriden sonra en yüksek olasılıkla üretilir.

Google'ın AI Blogunda [6] gösterilen örnek aşağıdadır, referans olması için buraya koydum.

Genel Bakış - Trafo Şebekesinin Çalışması. kaynak: Google AI (CC0).

Transformer, her kelime için ilk temsilleri veya katıştırmaları üreterek başlar. Bunlar doldurulmamış dairelerle temsil edilir. Ardından, öz-dikkatini kullanarak, diğer tüm kelimelerden bilgi toplar ve dolu toplarla temsil edilen tüm bağlam tarafından bilgilendirilmiş kelime başına yeni bir temsil üretir. Bu adım daha sonra tüm kelimeler için paralel olarak birçok kez tekrarlanır ve art arda yeni temsiller üretilir.

Kod çözücü benzer şekilde çalışır, ancak soldan sağa doğru her seferinde bir sözcük üretir. Yalnızca önceden oluşturulmuş diğer sözcüklerle değil, aynı zamanda kodlayıcı tarafından oluşturulan son temsillerle de ilgilenir.

Transformatör böyle çalışır ve bu artık NLP'deki en gelişmiş tekniktir. ***Öz-dikkat mekanizması*** kullanarak harika sonuçlar veriyor ve aynı zamanda paralelleştirme sorununu da çözüyor. Google bile , yaygın NLP uygulamaları için modelleri önceden eğitmek üzere bir transformatör kullanan **BERT'yi kullanır.**

. . .

Referanslar -

1. *Tüm ihtiyacınız olan dikkat* : Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob

Thieu Pham, Christopher D. Manning.

3. Colah'ın Blogu.
4. Jay Alammar'ın Blogu.
5. CodeEmporium: Youtube.

NLP Dikkat Nmt Doğal dil işleme Derin Öğrenme