

Transformer'ların Kalbi: Attention Mekanizması

Bir Cümplenin Veriye Dönüşüm Yolculuğu

Kelimeler Yalnızken Anlamsızdır

O tarlada bir **ben** gördüm.



Bugün çok yorgunum, **ben** artık uyuyacağım.



Aynı "ben" kelimesi, farklı cümlelerde tamamen farklı anımlara gelir. Bir makine öğrenmesi modeli için bu ilk "gömme" (embedding) işlemi, bağlama referans vermeyen bir arama tablosudur; yani her iki "ben" kelimesi de başlangıçta aynı vektörle temsil edilir.

Peki bir makine, bir kelimenin anlamını zenginleştirmek için çevresindeki kelimeleri nasıl kullanır?

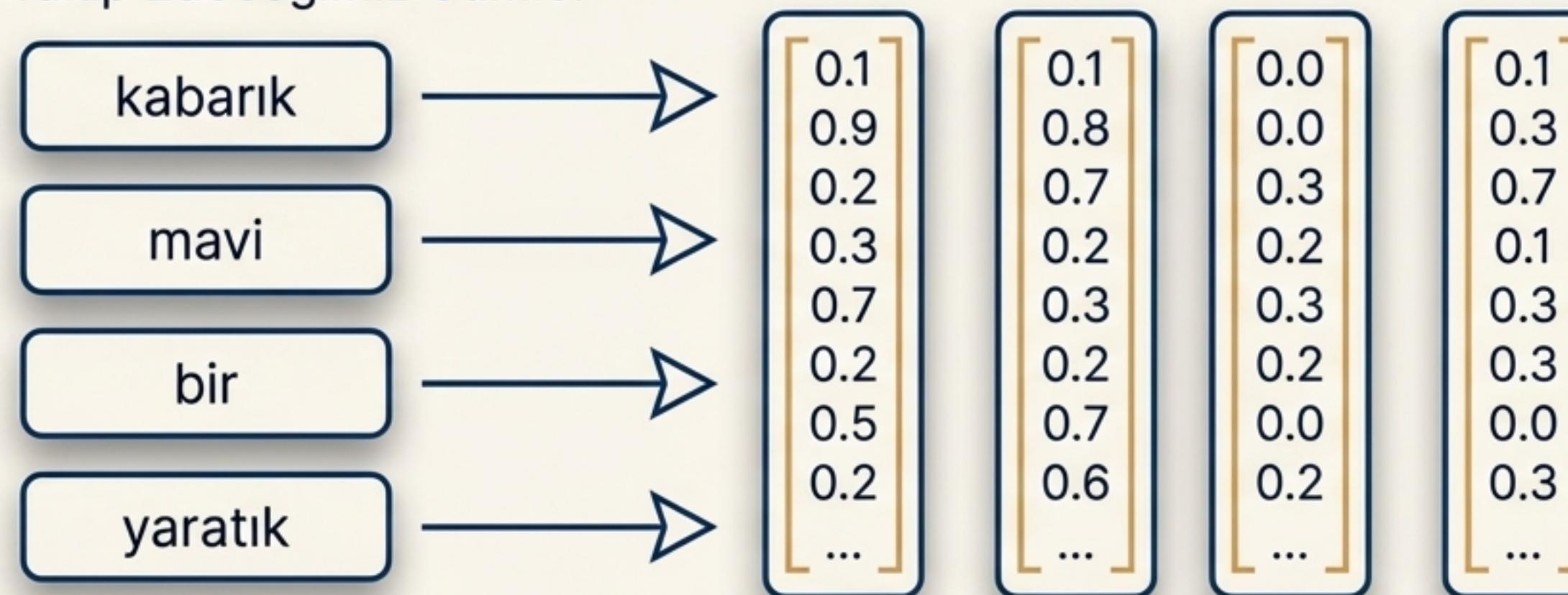
Hedef: Kelimelerin, çevrelerindeki diğer kelimelerden nasıl anlam kazandığını modellemek.

Adım 1: Kelimelerden Vektörlere (Embedding)

Modelin yolculuğu, her kelimeyi (veya "token"ı) yüksek boyutlu bir uzayda bir noktaya, yani bir "vektöre" dönüştürmekle başlar. Bu işleme **Embedding** denir. Bu vektörler, kelimelerin anlamsal ilişkilerini kodlar. Eğitimli bir modelde, bu uzaydaki yönler anlamsal bir anlama karşılık gelir.

Örneğin: $\text{vektör(Kral)} - \text{vektör(Erkek)} + \text{vektör(Kadın)} \approx \text{vektör(Kraliçe)}$

Yolculuğunu Takip Edeceğimiz Cümle:

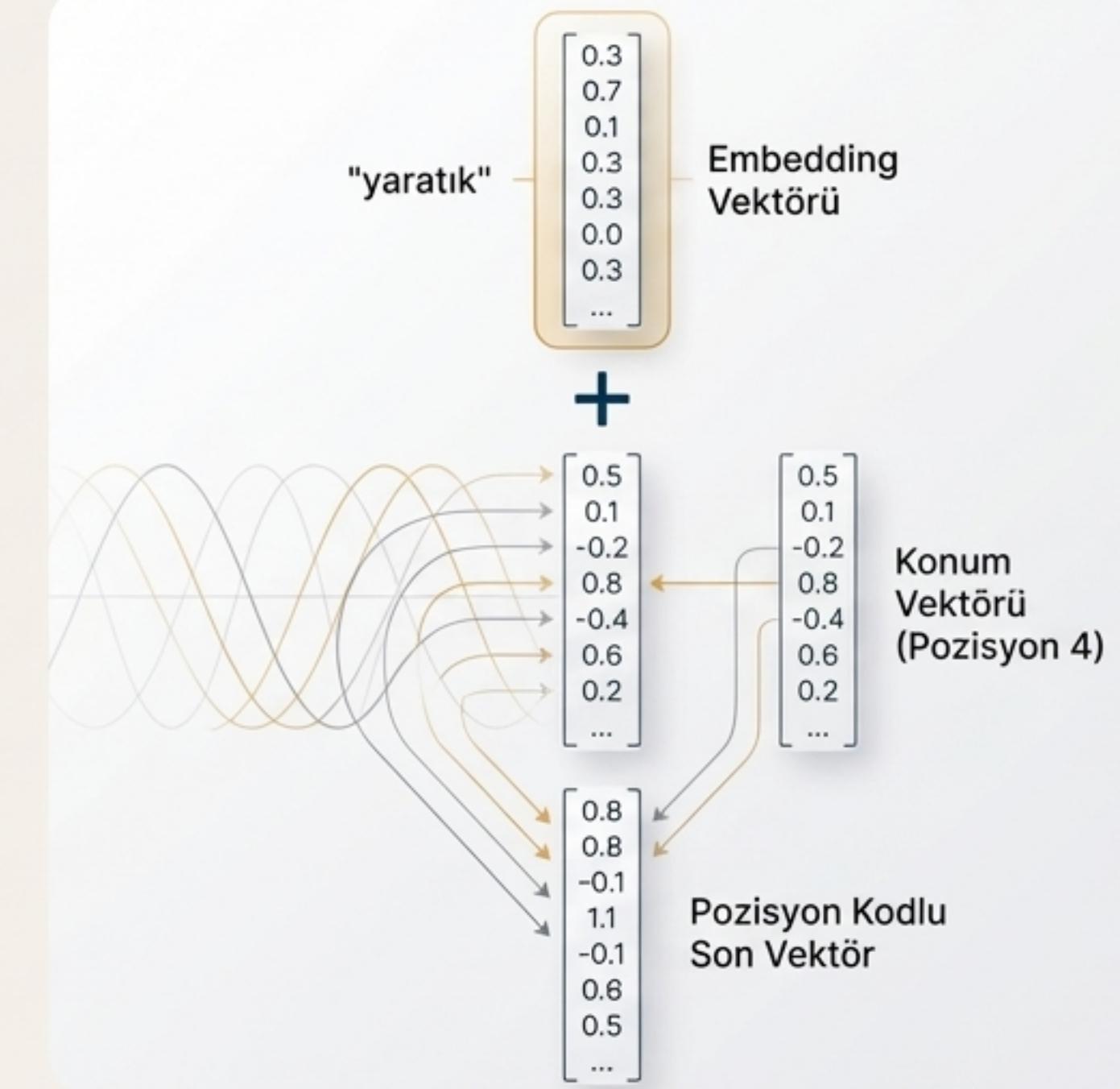


Adım 2: Kelimelerin Sırasını Eklemek (Positional Encoding)

Mevcut kelime vektörleri (embedding'ler) anlamı taşıır, ancak sıralamayı taşımaz. "Mavi kabarık yaratık" ile "Kabarık mavi yaratık" arasındaki farkı anlamak için, her kelimenin vektörüne, o kelimenin pozisyonunu belirten özel bir "konum vektörü" eklenir.

Denklem: **‘Son Vektör = Embedding Vektörü + Konum Vektörü’**

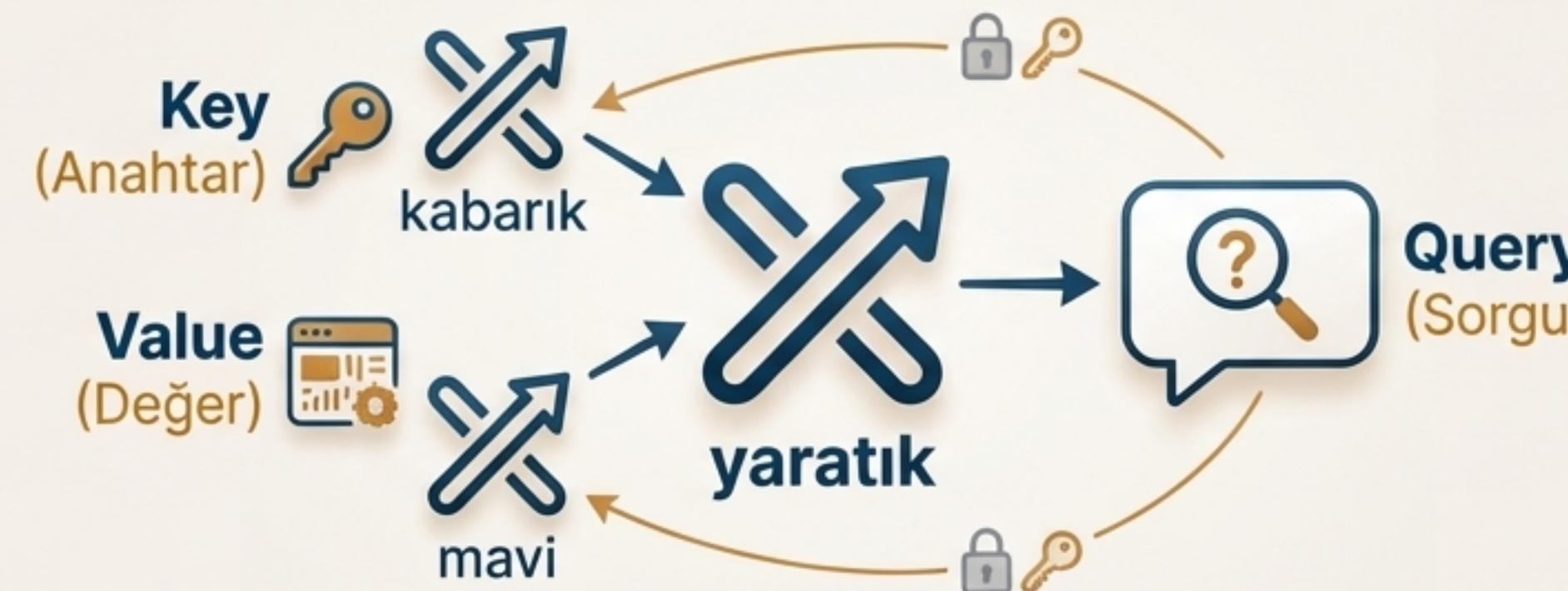
Bu işlem için genellikle farklı freksnlardaki sinüs ve kosinüs fonksiyonları kullanılır. Bu sayede model, kelimelerin göreceli pozisyonlarını öğrenebilir.



Attention'ın Ana Fikri: Vektörlerin Sohbeti

Attention, her kelime vektörünün, cümlenin diğer tüm kelime vektörleriyle etkileşime girerek kendi anlamını zenginleştirmesini sağlayan bir mekanizmadır. Bu etkileşim, bir "sohbet" metaforuyla anlaşılabılır.

Örnek: "yaratık" kelimesi, anlamını netleştirmek için diğer kelimelere sorar: "Beni en iyi hangi kelimeler tanımlıyor?"



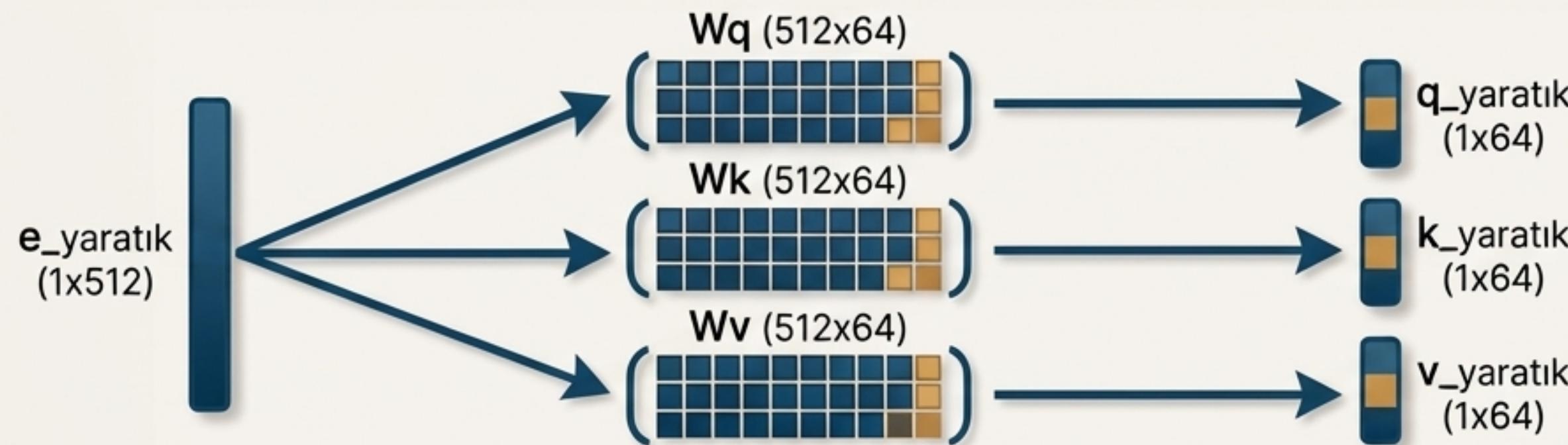
Bu "sohbet" için her kelime üç farklı role bürünür:

1. **Query (Sorgu):** 🔎 Aradığım şey ne? ("Ben bir isimim, bana uygun sıfatlar nerede?")
2. **Key (Anahtar):** 🗝️ Sunduğum şey ne? ("Ben bir sıfatım, 'mavilik' anlamını taşııyorum.")
3. **Value (Değer):** 📈 Eğer Sorgu ve Anahtar eşleşirse, sana vereceğim asıl bilgi/anlam ne? ("Sana 'mavi' bilgisini aktaracağım.")

Q, K, V Vektörlerinin Oluşturulması

Her kelimenin pozisyon kodlu vektörü, modelin eğitim sırasında öğrendiği üç farklı ağırlık matrisi (W_q , W_k , W_v) ile çarpılarak o kelimeye özel Q, K ve V vektörleri oluşturulur. Bu matrisler, modelin verilerden öğrendiği parametrelerdir.

Bu işlem sayesinde model, bir kelimenin nasıl "soru soracağını" (Query), ne tür bilgilere "cevap verebileceğini" (Key) ve eşleşme durumunda hangi bilgiyi "aktaracağını" (Value) öğrenir.



$$q_{\text{yaratik}} \text{ (1x64)} = e_{\text{yaratik}} \text{ (1x512)} * W_q \text{ (512x64)}$$

$$k_{\text{yaratik}} \text{ (1x64)} = e_{\text{yaratik}} \text{ (1x512)} * W_k \text{ (512x64)}$$

$$v_{\text{yaratik}} \text{ (1x64)} = e_{\text{yaratik}} \text{ (1x512)} * W_v \text{ (512x64)}$$

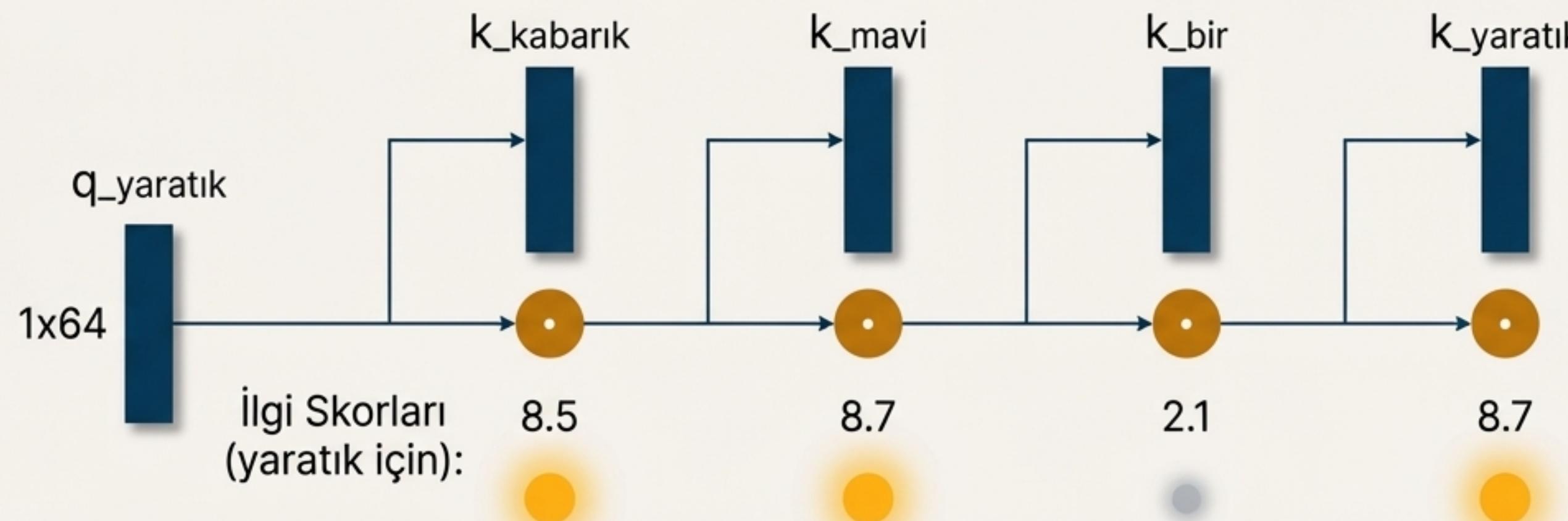
Adım 3: İlgi Skorlarının Hesaplanması ($Q \cdot K$)

Bir kelimenin diğer kelimelerle ne kadar “ilgili” olduğunu bulmak için, o kelimenin **Query (Q)** vektörü, cümleinin diğer tüm kelimelerinin **Key (K)** vektörleri ile tek tek çarpılır. Bu işleme **dot product (nokta çarpım)** denir.

Dot product, iki vektörün ne kadar hizalı veya benzer olduğunu ölçen ölçeklenmemiş bir metriktir. Yüksek sonuç, yüksek ilgi ve benzerlik demektir.

‘Skor(yaratık, mavi) = $q_{\text{yaratık}} \cdot k_{\text{mavi}}$ ’ \rightarrow Yüksek bir sayı çıkar.

‘Skor(yaratık, bir) = $q_{\text{yaratık}} \cdot k_{\text{bir}}$ ’ \rightarrow Düşük bir sayı çıkar.

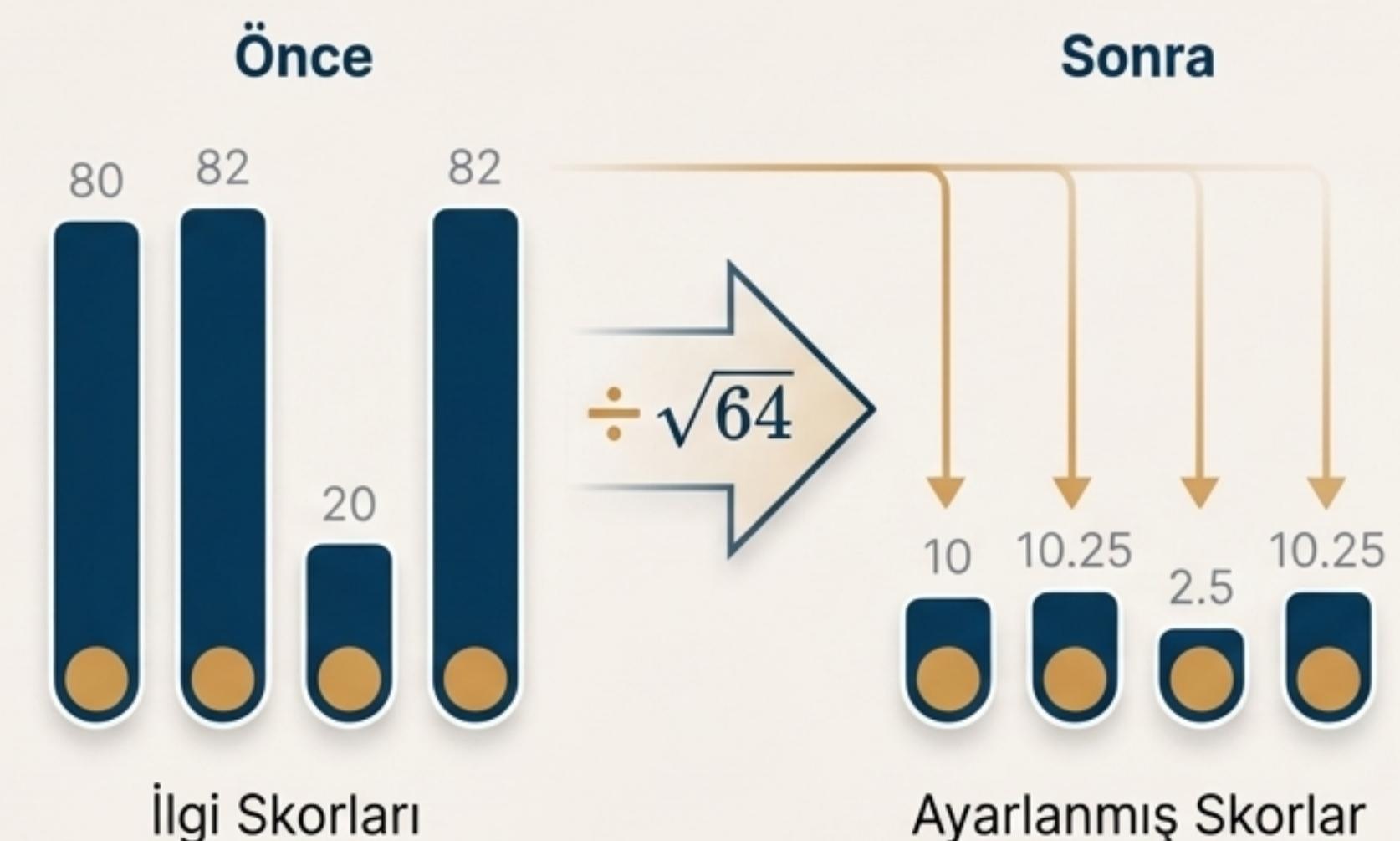


Skorların Ayarlanması (Scaling)

Dot product sonuçları, vektör boyutları büyükçe çok büyük değerlere ulaşabilir. Bu durum, softmax fonksiyonunu küçük gradyanlı bölgelere iterek öğrenme sürecini kararsız hale getirebilir. Bunu önlemek için, skorları Q ve K vektörlerinin boyutunun (d_k) kareköküne böleriz.

Bu işlem, gradyanları daha stabil hale getirerek modelin daha iyi ve verimli öğrenmesini sağlar.

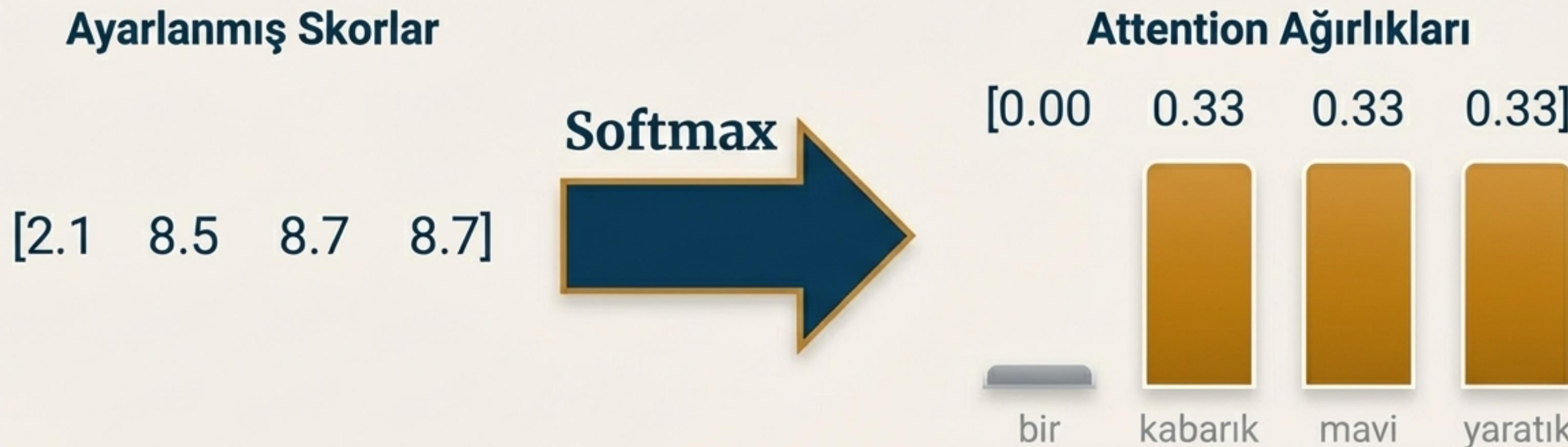
$$\text{Ayarlanmış Skor} = \frac{\mathbf{Q} \cdot \mathbf{K}^T}{\sqrt{d_k}}$$



Adım 4: Skorları Ağırlıklara Dönüşütmek (Softmax)

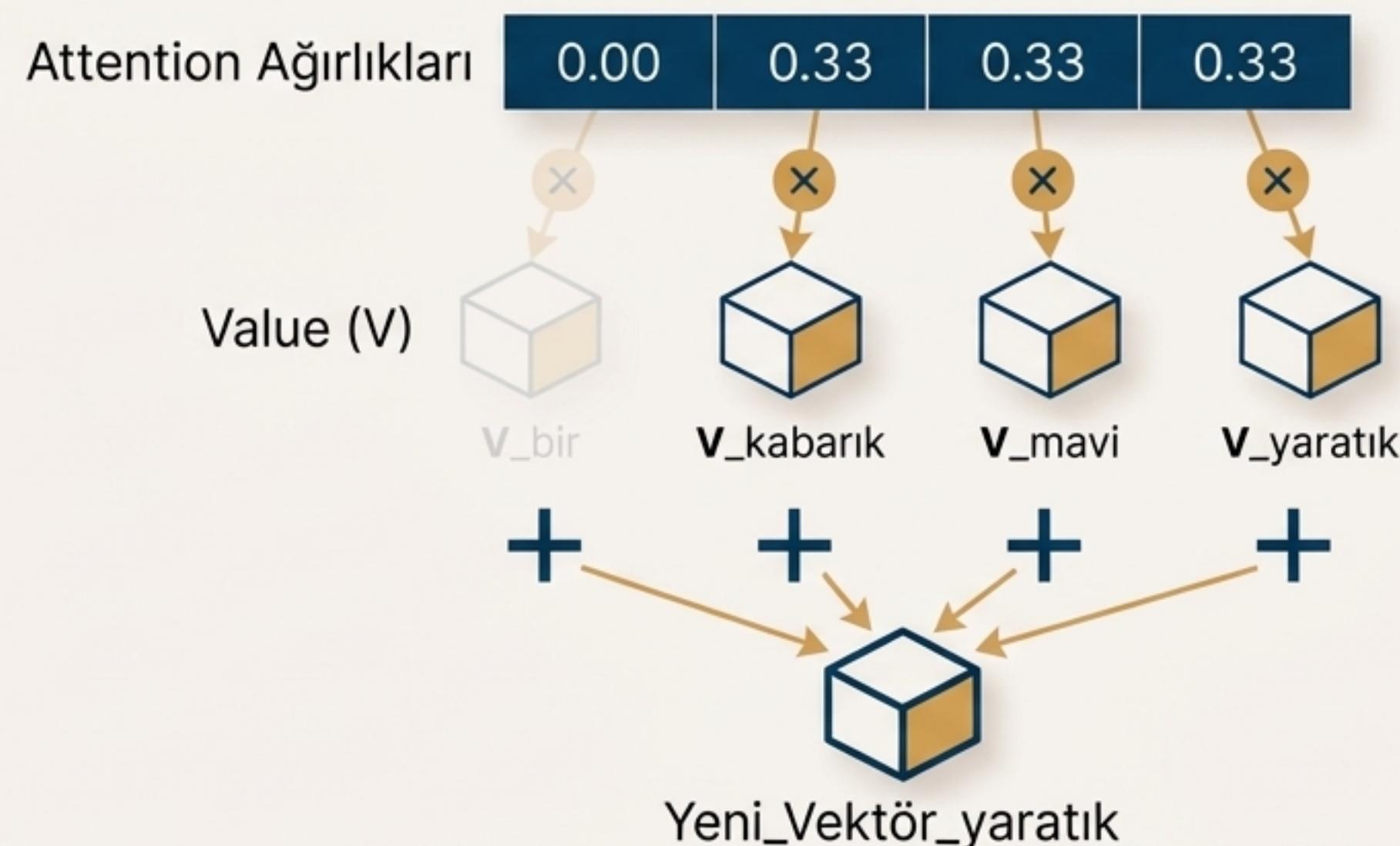
Ayarlanmış skorlar, henüz sezgisel bir anlam taşımaz. Bunları, toplamları 1 olan ve 0 ile 1 arasında değer alan olasılık benzeri **ağırlıklara** dönüştürmek için **Softmax** fonksiyonu kullanılır.

Artık "yaratık" kelimesinin, kendi anlamını oluştururken hangi kelimeye yüzde kaç oranında "dikkat" etmesi gerektiğini biliyoruz. Bu ağırlık dağılımına "**Attention Patern**" (**Dikkat Deseni**) denir.



Adım 5: Değerlerin (V) Ağırlıklı Toplamı

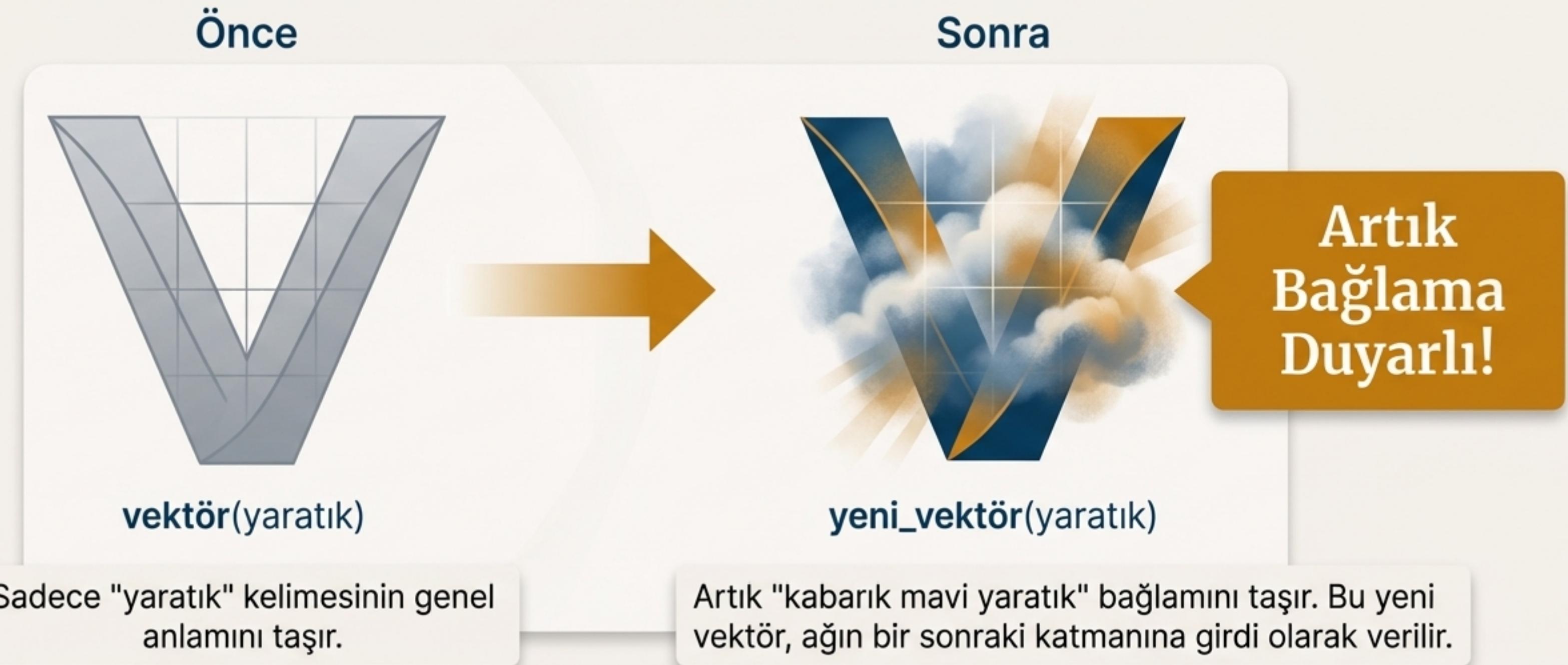
Son adımda, hesapladığımız attention ağırlıklarını, her kelimenin **Value (V)** vektörü ile çarparız ve sonuçları toplarız. Bu işlem, en çok dikkat edilen kelimelerin "anlamını" çeken yeni bir vektör oluşturur.



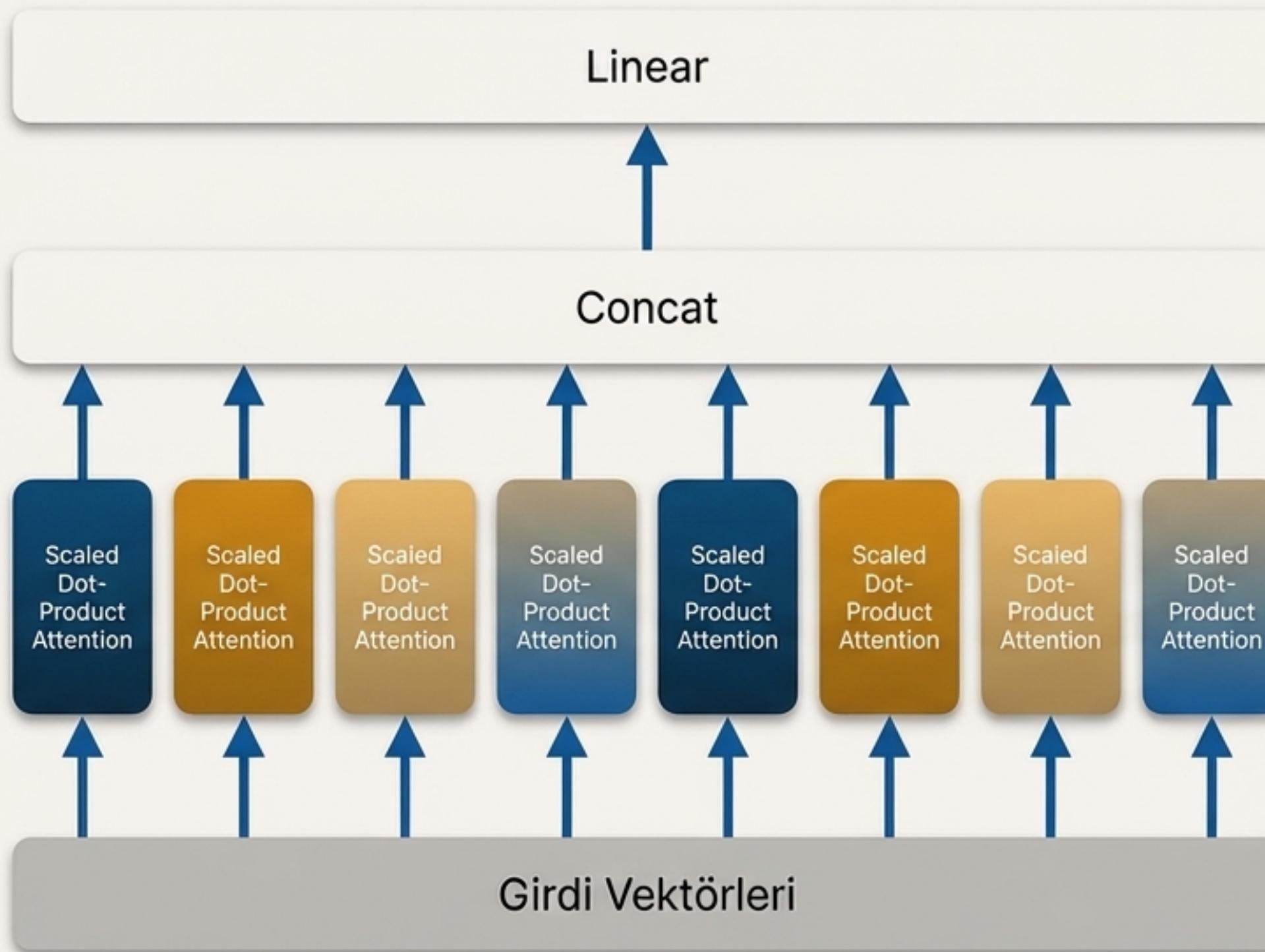
$$\text{Yeni_Vektör_yaratık} = (0.00 * v_{\text{bir}}) + (0.33 * v_{\text{kabarık}}) + (0.33 * v_{\text{mavi}}) + (0.33 * v_{\text{yaratık}})$$

Dönüşüm Tamamlandı: Öncesi ve Sonrası

Attention mekanizması sayesinde, "yaratık" kelimesinin yolculuğun başındaki orijinal vektörü, artık "kabarık" ve "mavi" kelimelerinin anlamsal bilgisini de içeren zenginleştirilmiş yeni bir vektöre dönüştü.



Büyük Resim: Tek Bir Bakış Açısı Yeterli Değil (Multi-Head Attention)



Az önce anlattığımız tüm süreç, tek bir '**Attention Head**' (Dikkat Kafası) idi.

Transformer'lar bu işlemi, farklı ve bağımsız W_q , W_k , W_v matris setleriyle **paralel olarak** birçok kez (örn. GPT-3'te 96 kez) yapar.

Her 'kafa', farklı türde anlamsal ilişkilere odaklanmayı öğrenir:

- Bir kafa sıfat-isim ilişkisine odaklanabilir.
- Başka bir kafa fiil-nesne ilişkisine odaklanabilir.
- Diğer bir kafa, daha az dilbilgisel olan tematik bağlantılarla (örn. 'büyücü' ve 'Harry') odaklanabilir.

Bu, modelin dili çok daha zengin ve katmanlı bir şekilde anlamasını sağlar.

Mimarinin Tamamı: Transformer Bloğu

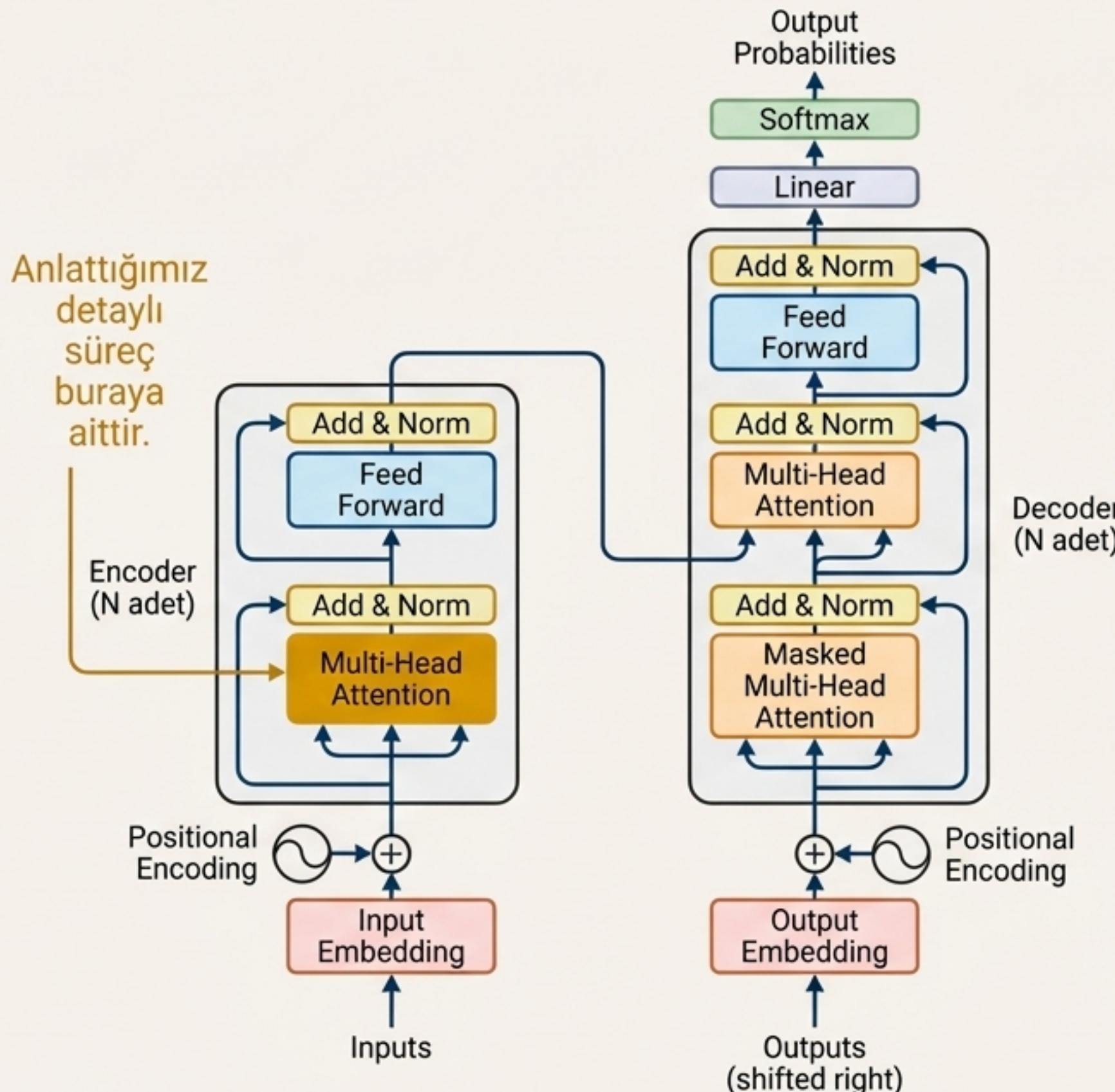
Multi-Head Attention, bir Transformer bloğunun sadece ilk alt katmanıdır. Her blokta ayrıca şunlar bulunur:

1. Residual Connection (Artık Bağlantı) & Layer Norm:

Her alt katmanın (Attention ve Feed-Forward) çıktısı, girdisiyle toplanır ('Add') ve normalize edilir ('Norm'). Bu, bilgi kaybını önler ve eğitimi stabilize eder.

2. Feed-Forward Network:

Her kelimenin vektörünü ayrı ayrı işleyerek ek bir hesaplama ve dönüşüm katmanı sağlar.

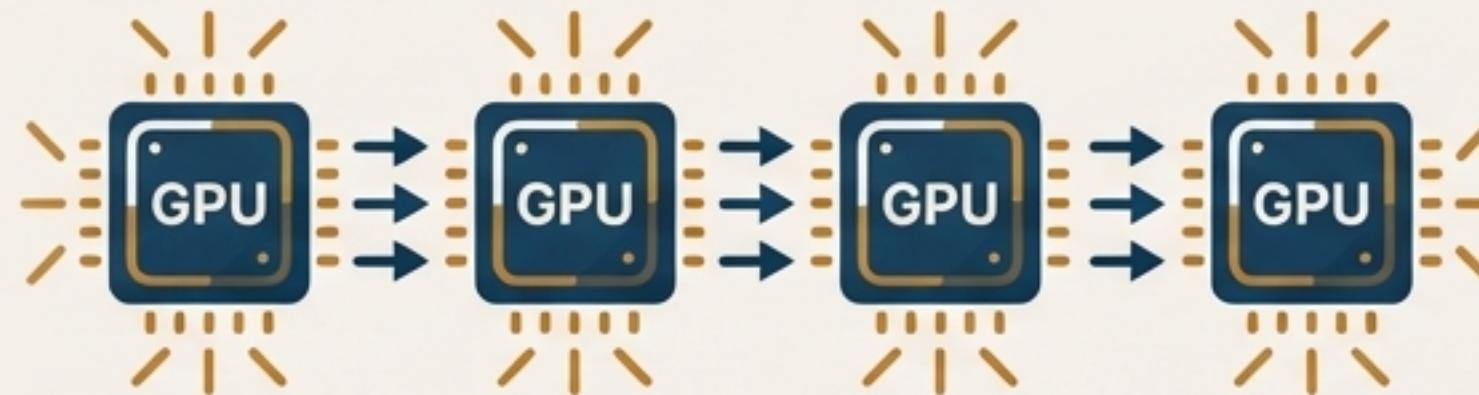


Bu bloklar, hem **Encoder (Kodlayıcı)** hem de **Decoder (Kod Çözücü)** katmanlarında üst üste N kez (orijinal makalede 6 kez) tekrarlanır.

Attention Neden Devrimsel?

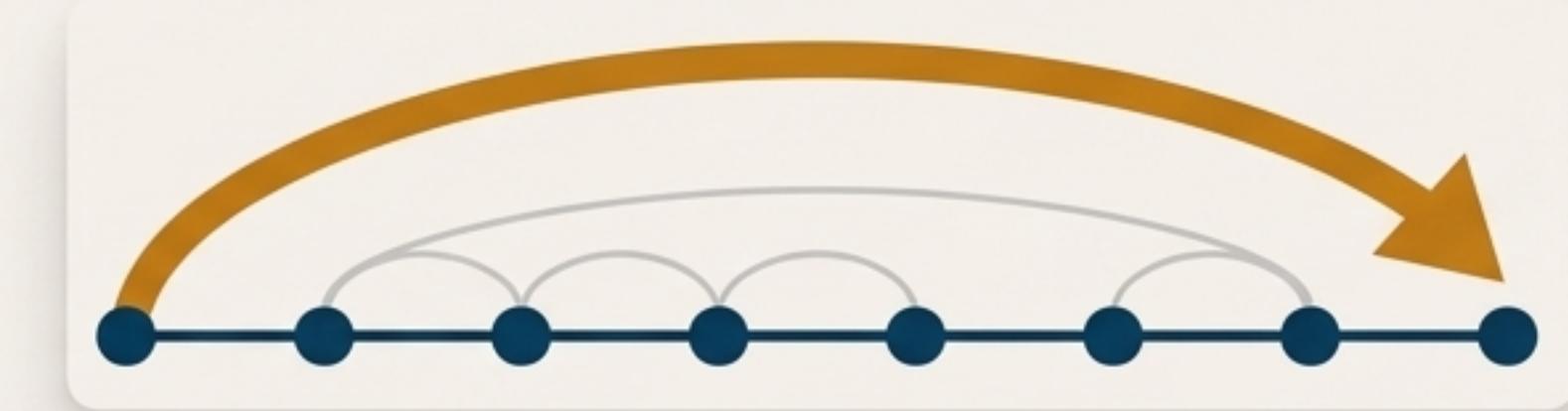
Transformer mimarisi, RNN gibi eski yaklaşımlara göre iki büyük avantaj sunarak yapay zeka alanında bir devrim yarattı:

1. Paralelleştirme



Her kelimenin attention skoru diğerlerinden bağımsız ve aynı anda hesaplanabilir. Bu, modern GPU'ların paralel işlem gücünden tam olarak yararlanmayı sağlar ve eğitim sürelerini büyük ölçüde kısaltır.

2. Kısa Bağlantı Yolları



Cümlenin başındaki ve sonundaki iki kelime arasındaki etkileşim yolu direkt ve sabittir (sabit sayıda işlem). RNN'lerde bu yol, aradaki kelime sayısı kadar uzundur. Bu sayede Transformer'lar, **uzun mesafeli anlamsal bağlantıları** (long-range dependencies) çok daha kolay öğrenir.

Anahtar Çıkarımlar

- **Attention**, kelimelerin bağlam içindeki anlamını zenginleştirmek için bir "ilgi" ve "benzerlik" hesaplama mekanizmasıdır.
- **Query, Key, Value**, kelimelerin birbirleriyle etkileşime girmesini sağlayan, eğitimle öğrenilmiş "rollerdir".
- Süreç, **ilgi skoru** bulma ($Q \cdot K$), **skorları ayarlama** (scale & softmax) ve **değerlerin (V) ağırlıklı toplamını** alarak yeni, bağlama duyarlı bir vektör oluşturma adımlarından oluşur.
- Bu basit ama son derece **paralelleştirilebilir** ve **ölçeklenebilir** fikir, günümüzün en gelişmiş yapay zeka modellerinin temelini oluşturur.

