TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞLARI İLE MAKİNE ÇEVİRİSİNİN DETAYLI BAŞARIM ANALİZİ

YÜKSEK LİSANS TEZİ

Simla Burcu HARMA

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Oğuz ERGİN

OCAK 2020

Fen Bilimleri Enstitüsü Onayı

.....
Prof. Dr. Osman EROĞUL
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

.....
Prof. Dr. Oğuz ERGİN
Anabilimdalı Başkanı

TOBB ETÜ, Fen Bilimleri Enstitüsü'nün 181111014 numaralı Yüksek Lisans Öğrencisi **Simla Burcu HARMA** 'in ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"YAPAY SİNİR AĞLARI İLE MAKİNE ÇEVİRİSİNİN DETAYLI BAŞARIM ANALİZİ"** başlıklı tezi **23.01.2020** tarihinde aşağıda imzaları olan jüri tarafından kabul edilmiştir.

Tez Danışmanı: **Prof. Dr. Oğuz ERGİN** .....
TOBB Ekonomi ve Teknoloji Üniversitesi

Jüri Üyeleri: **Dr. Öğr. Üyesi Mücahid KUTLU (Başkan)**.....
TOBB Ekonomi ve Teknoloji Üniversitesi

Dr. Öğr. Üyesi Can ALKAN .....
Bilkent Üniversitesi

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

Simla Burcu HARMA

ÖZET

Yüksek Lisans Tezi

YAPAY SİNİR AĞLARI İLE MAKİNE ÇEVİRİSİNİN DETAYLI BAŞARIM ANALİZİ

Simla Burcu HARMA

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Oğuz ERGİN

Tarih: Ocak 2020

Teknoloji çağında yaşıyoruz ve son on yılda Yapay Zeka üzerine en çok çalışılan teknoloji olmuştur. Sayısız alanda uygulaması bulunmakla birlikte, Yapay Sinir Ağları ile Makine Çevirisi (YMÇ) temel araştırma alanlarından birisidir. Google, Facebook, Amazon, Microsoft, Yandex gibi birçok büyük şirket ürünlerinde YMÇ kullanmaktadır ve YMÇ'nin kullanıcılara servis olarak sunulması son zamanlarda benimsenen bir yöntem olmuştur. Bu servislerin, kullanıcı memnuniyeti açısından, hız ve çeviri kalitesi başta olmak üzere bazı kısıtları sağlamaları gerekmektedir. YMÇ modellerini hızlandırmak konusunda birçok çalışma yapılmıştır, ancak bildiğimiz kadarıyla bu çalışmaların hiçbiri detaylı başarım/zaman analizinde bulunmamıştır. Bu çalışmada en gelişmiş YMÇ modellerinden birisi olan, kodlayıcı-kodçözücü yapısını ilgi mekanizmasıyla birleştiren Dönüştürücü modeli ile çalışılmıştır.

Dönüştürücü'nün etrafına bir mikroservis kurulmuş ve sistemin darboğazının modelin kendisi olduğu gösterilmiştir. Bunun üzerine temel yapılandırma parametrelerinin değişimiyle deneyler yapılmış ve bu parametrelerin başarımı hassas bir şekilde etkilediği gözlenmiştir. Bunun üzerine modelin her bir bileşeninin CPU ve GPU'da detaylı zaman dökümü çıkarılmış ve en verimsiz aşamanın ışın araması olduğu gösterilmiştir. Daha sonra ışın aramasının daha iyi anlaşılması adına her bir adımını gösteren zaman dökümü çıkarılmıştır. Ayrıca, ışın boyutunun BLEU skorunu sadece kelime-bazında

etkilediği, türce-bazında bir etkisinin olmadığı gözlemlenmiştir. Son olarak kelime-hazinesi büyüklüğünün ışın aramasının başarımında büyük rolü olduğu gösterilmiştir.

Anahtar Kelimeler: Makine Çevirisi, Mikroservisler, Yapay Sinir Ağları, Derin Öğrenme, Performans Analizi.

ABSTRACT

Master of Science

AN IN-DEPTH PERFORMANCE ANALYSIS OF NEURAL MACHINE TRANSLATION TASKS

Simla Burcu HARMA

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Prof. Dr. Oğuz ERGİN

Date: January 2020

We live in the technology era and over the last decade Artificial Intelligence (AI) has been the most focused technology. It has applications in countless topics and neural machine translation (NMT) is one of the major research areas. Many big companies like Google, Facebook, Amazon, Microsoft, Yandex deploy NMT in their production systems and NMT services has become popular lately. These services need to provide some constraints, especially speed and translation quality, for user satisfaction. There has been significant amount of work on accelerating NMT models however to the best of our knowledge, there is no detailed research giving a detailed performance analysis of each step in a model. In this work, one of the state-of-the-art models the Transformer is used. It has encoder-decoder architecture with an additional attention mechanism.
A microservice is implemented on top of the Transformer model and it is showed that the bottleneck is the model itself. Then, several experiments with different configuration values has been conducted and it is observed that the performance of the model is highly sensitive to the changes in these values. A detailed performance breakdown of the model in CPU and GPU show that beam search is a big source of inefficiency. So a detailed time breakdown of the beam search is obtained in order to have a better understanding. Additionally, it is observed that the beam size only affects BLEU score at word level,

and not at token level. Finally, it is showed that the vocabulary size has a major role on the performance of the beam search.

Keywords: Neural Machine Translation, Microservices, Neural Networks, Deep Learning, Performance Analysis.

TEŞEKKÜR

Lisans ve yüksek lisans çalışmalarım boyunca her türlü yardım ve katkılarıyla beni yönlendiren, her zaman arkamda olan değerli hocam ve tez danışmanım Prof. Dr. Oğuz Ergin'e, kıymetli tecrübelerinden yararlandığım, bana her konuda yardımcı olan TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği Bölümü öğretim üyelerine, tez çalışmamdaki her türlü yardım ve katkılarından ötürü Mario Drumond'a ve École Polytechnique Fédérale de Lausanne (EPFL) öğretim üyesi Prof. Babak Falsafi'ye, eğitimim boyunca bana burs veren TOBB Ekonomi ve Teknoloji Üniversitesi'ne ve TÜBİTAK Bilim İnsanı Destek Programları Başkanlığı'na, destekleriyle her zaman yanımda olan arkadaşlarıma ve bu süreçte gösterdikleri her türlü destek, sabır ve yardımları için biricik anneme ve babama çok teşekkür ederim.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

BLEU : Bilingual Evaluation Understudy Score
CPU : Merkezi İşlem Birimi (Central Processing Unit)
DDİ : Doğal Dil İşleme (Natural Language Processing)
ESA : Evrişimli Yapay Sinir Ağları (Convolutional Neural Networks)
FPGA : Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array)
GPU : Grafik İşlemci Ünitesi (Graphics Processing Unit)
İMÇ : İstatistiksel Makine Çevirisi (Statistical Machine Translation)
MÇ : Makine Çevirisi (Machine Translation)
MUSE : Multilingual Unsupervised or Supervised word Embeddings
ÖSA : Özyineli Sinir Ağları (Recurrent Neural Networks)
ReLU : Doğrultulmuş Doğrusal Ünite (Rectified Linear Unit)
TPU : Tensor Processing Unit
UKVB : Uzun Kısa Vadeli Bellek (Long Short-term Memory)
YMÇ : Yapay Sinir Ağları Bazlı Makine Çevirisi (Neural Machine Translation)
YSA : Yapay Sinir Ağları (Artificial Neural Networks)

1. GİRİŞ

Makine çevirisi, son yıllarda büyük bir veri merkezi iş yükü haline geldi. Çevrimiçi servislerin kullanıcı tarafından üretilen metinleri işleme ve yeni metinler oluşturma gibi işlerdeki kullanımı dünya çapında arttığından, makine çevirisi birçok çevrimiçi servisin bir parçası durumundadır [1]. Yapay Sinir Ağları (YSA) ve Doğal Dil İşleme alanlarındaki gelişmelerle birlikte YSA bazlı makine çevirisi (YMÇ), en başarılı makine çevirisi yöntemlerinden biri oldu. Ancak YMÇ'nin hesaplama gereksinimleri, diğer geleneksel çevrimiçi servislere kıyasla onlarca kat daha büyüktür. Bu da servis sağlayıcılarını, altyapılarının hesaplama gücünü arttırmaya itmiştir.

Mikroservisler [2] ve hızlandırma [3, 1, 4], servis sağlayıcılarının makine çevirisi gibi YSA servislerinin yüksek hesaplama gereksinimlerini karşılayabilmek için kullandığı iki temel tekniktir. Mikroservisler, başarım, kullanılabilirlik ve esneklik kısıtlarını yerine getirmek için; sıkı bir şekilde birleştirilmiş, tek bir büyük uygulamadan oluşan monolitik yaklaşımının aksine daha küçük, gevşek bir şekilde bağlı servislerden tek bir uygulama oluşturur. Servis sağlayıcılar ayrıca enerji verimliliğini arttırmak, aynı güç kısıtları altında daha fazla yükü karşılayabilmek için GPU, FPGA veya TPU gibi hızlandırıcıları da kullanırlar.

Bu sebeplerden dolayı; YMÇ servislerinin iş hacmi, çevrimiçi servis sağlayıcılarının sistemlerini nasıl tasarladığını etkilemektedir. YMÇ servislerinin hesaplama başarımı ile çevirinin kalitesi arasındaki ilişki açık olmamakla birlikte, üzerine fazla çalışılmamıştır. Modelin her parçası çeviri başarımını ve kalitesini etkileyen çok fazla parametreye sahiptir. Daha önceki çalışmalarda bu parametreler tarafından oluşturulan tasarım uzayını araştırılmışken, başarım ve kalite ilişkisi üzerine derinlemesine bir araştırma yapılmamıştır.

YSA mikroservisleri, kullanıcı memnuniyeti açısından, kaliteli olmasının yanında gecikme kısıtlarını da sağlamak durumundadır. Bu nedenle, sistem tasarımı sürecinde, sistemin farklı platformlardaki (CPU ve GPU) darboğazlarını belirlemek ve tasarımı, bu darboğazları göz önünde bulundurarak gerçekleştirmek önemlidir. YSA mikroservislerinde gerçekleşen işlemler sırasında, verinin taşınması veya YSA modelinin çalışması

darboğazları oluşabilir. Bu çalışmada, makine çevirisi yapan YSA mikroservislerinin darboğazının YSA modelinin hızı olduğunu gösterdik.
Makine çevirisinde girdi verisi yalnızca metin olduğundan ve yapay sinir ağının yapısının farklılığından dolayı iş yükünün YSA kısmı, veri işleme ve taşıma kısmından daha çok zaman alır. Çok sayıda büyük boyutlarda matris çarpımları gerektirdiği için ise bilgisayım (computation) yükü fazladır. Bu sebeple, makine çevirisi yapan YSA mikroservislerinin hızlandırılması için öncelikle YSA modellerinin hızlandırılması gerekmektedir. YMÇ modellerini hızlandırmak konusunda birçok çalışma yapılmıştır, ancak bildiğimiz kadarıyla bu çalışmaların hiçbiri detaylı başarım/zaman analizinde bulunmamıştır. En gelişmiş YMÇ modellerinden birisi, "İlgi Mekanizması"nı kullanan, Vaswani vd.nin geliştirdiği "Dönüştürücü" modelidir [5]. Bu alandaki birçok güncel çalışma Dönüştürücü'nün hızını ve çeviri kalitesini geliştirme üzerinedir.

1.1 Tezin Katkıları

Tez kapsamında, YMÇ başarımı derinlemesine incelenmiş ve analiz edilmiştir. Yapılan çalışmalar ve katkıları aşağıda sıralandığı gibidir:

- En gelişmiş YMÇ modellerinden olan Dönüştürücü'nün OpenNMT [6] gerçeklemesinin etrafına bir mikroservis kurularak profili çıkarıldı. İşlem zamanının büyük çoğunluğunun YMÇ sistemine harcandığı, sunucu ile istemci arasındaki iletişimin yoksayılabilecek kadar az zaman aldığı gözlendi.

- YMÇ sistemlerinin CPU ve GPU'daki çalışma zamanlarının detaylı analizi yapıldı. Her bir bileşen için ne kadar zaman harcandığı gösterildi.

- Temel sistem parametrelerinin her birinin veri hacmi (throughput) üzerine etkisi incelendi.

- Işın araması aşamasının darboğaz olduğu gözlemlendi. Işın aramasının detaylı başarım analizi yapıldı.

- Kelime hazinesi boyutu azaltılarak modelin davranışı gözlemlendi.

1.2 Literatür Araştırması

Yapay öğrenmenin her alanda uygulanmaya başlanmasıyla, makine çevirisi alanında da uygulamaları hız kazanmıştır. Yeni modeller önerilmesinin yanında mevcut modellerin analizini yapmak, hız ve kalitesini artırmak üzerine de birçok çalışma yapılmıştır. Junczys-Dowmunt vd. [7] mevcut gelişmiş sistemlerde, birçok çeviri ikilisini analiz etmiş ve yeni, hızlı bir kodçözücü geliştirmişlerdir. Niehues vd. [8] makine çeviri

sistemlerinde modelleme ve arama algoritmalarını ayrıştırarak, mevcut sistemleri analiz etmişler ve çeviri kalitesi açısından yeterliliklerini ölçmüşlerdir. En gelişmiş modellerden biri olan Vaswani vd. [5]'nin önerdiği "Dönüştürücü" modeli literatürde en çok çalışılan modellerden biridir. Dönüştürücü modelinin analizi, hızının artırılması ve çeviri kalitesinin geliştirilmesi üzerine birçok çalışma yapılmıştır [9, 10, 11, 12, 13]. Dönüştürücü modeli de dahil olmak üzere, makine çevirisi modellerinde karşılaşılan sorunlardan bahseden Koehn vd. [13], ışın aramasının büyüklüğünün artırılmasının bir süre sonra çeviri kalitesini düşürdüğünü önermişlerdir. Daha sonra bu problem üzerine yoğunlaşan Yang vd. [14] mevcut ışın araması varyasyonlarını analiz etmişler ve yüksek ışın büyüklüklerinde BLEU skoru yükselten bir yöntem geliştirmişlerdir. Işın aramasının bir başka versiyonunu ise Huang vd. [15] oluşturmuştur. Çince-İngilizce çeviride BLEU skoru 2 puan artırmayı başarmışlardır.

Işın aramasının çeviri kalitesine etkisinin yanında, başarımını artırmak için yaptıkları çalışmada Freitag vd. [16], çeşitli budama (pruning) teknikleri önermişler ve kendi modellerindeki kodçözücüyü %43 hızlandırmışlardır. Shi vd. [17], GPU'larda makine çevirisini hızlandırmak için ışın aramasının kullandığı kelime-hazinesi boyutuna dikkat çekmişler ve kelime hizalama yöntemini önermiş ve kodçözücüyü 2 kat hızlandırmışlardır. Senellart vd. [18], OpenNMT'nin Tensorflow portunu kullanarak modelde, aralarında Shi vd. [17]'nin yönteminin de bulunduğu, CPU çalışmasını hızlandıracak iyileştirmeler yapmışlardır. Sonuç olarak BLEU skorda düşüş gözlemlense de son modelleri CPU üzerinde saniyede 800 kelime çevirebilmektedir.

1.3 Tez Taslağı

Tez kapsamında yapılan çalışmalar şu şekilde sıralanmıştır: Bölüm 2.1'de "Makine Çevirisi" araştırma alanı, makine çevirisi yöntemleri ve modelleri, ve çeviri kalitesi ölçüm metriği olan BLEU Skoru anlatılmıştır. Bölüm 2.2'de, yapay sinir ağı türlerinden ve bunların makine çevirisi alanındaki yerlerinden bahsedilmiştir. Bölüm 2.3'te, doğal dil işleme problemlerinde sıklıkla kullanılan Seq2Seq modeller açıklanmış, Dönüştürücü'nün temel bileşenlerinden olan ilgi mekanizmasına giriş yapılmıştır. Bölüm 2.4'te, Dönüştürücü modeli mimarisi, modelin bileşenleri ve çalışma şekli detaylı bir şekilde anlatılmıştır. Bölüm 3'te, tez kapsamında yapılan deneylerin parametreleri verilmiş, Bölüm 4'te ise bu deney sonuçları raporlanarak yorumlanmıştır. Bölüm 5'te, yapılan çalışmalar özetlenmekte ve gelecekte yapılabilecek çalışmalar hakkında fikirler verilmektedir.

2. ÖN BİLGİ

2.1 Makine Çevirisi

Makine çevirisi (MÇ) üzerine araştırmalar 20. yüzyılın ortalarına doğru başlamış [19], 1990'lardan beri [20] bilgisayarların hafıza ve hesaplama gücünün artmasıyla, diller arası metinlerin miktarının artması ve veriye erişimin kolaylaşması ile makine çevirisi konusundaki gelişmeler ivme kazanmış; günümüzde en çok çalışılan araştırma konularından biri haline gelmiştir. http://www.statmt.org websitesinin Temmuz 2019 verilerine göre makine çevirisi alanında toplam 5008 yayın bulunmaktadır ve bu yayınların yıllara göre dağılımı Şekil 2.1'deki gibidir [21].

Şekil 2.1: Makine çevirisi alanındaki yayınların yıllara göre dağılım grafiği

2.1.1 BLEU Skoru

Kishore Papineni vd. [22] tarafından 2002 yılında tanımlanan BLEU Skoru Bilingual Evaluation Understudy Score'un kısaltmasıdır. Bir metnin "aday çevirisi" ile bir veya daha fazla referans çevirilerinin karşılaştırılmasında kullanılan bir puanlandırmadır. Çeviri için geliştirilmiş olsa da birçok doğal dil işleme (DDİ) görevlerinde üretilmiş metnin değerlendirilmesinde kullanılabilir.

BLEU Skoru otomatik makine çevirisi sistemleri için geliştirilmiş olup mükemmel bir değerlendirme olmasa da birçok fayda sunmaktadır. Hesaplaması hızlı ve ucuz, anlaşılması kolay, dilden bağımsız, insan çevirisiyle oldukça uyumlu ve farklı işler için uyarlanabilir bir yapıya sahiptir. Bu sebeplerden dolayı da mükemmel değerlendirme

yapamamasına rağmen hala makine çevirisi alanındaki güncel makalelerin hemen hemen hepsinde değerlendirme metodu olarak kullanılmaktadır.
Mükemmel eşleşme için 1.0, mükemmel uyumsuzluk için 0.0 skorları verilir. Ancak birçok kaynak ve literatürdeki birçok makale BLEU Skoru aralığını $0-100$ olarak kullanmaktadır.
BLEU Skoru hesaplanırken, öncelikle aday çevirideki n-gram'lar ile referans çevirideki n-gram'lar karşılaştırılır (1-gram, cümledeki her bir kelime; 2-gram, cümledeki her bir kelime çifti; n-gram, cümledeki her bir kelime n'lisidir.). Karşılaştırma kelime sırasından bağımsız olarak yapılır. Makalede, eşleşen n-gram'ların sayılması işleminde sistemlerin ürettiği aday cümlede gereksiz yere "mantıklı" kelimeleri fazlaca bulunması eğiliminde olduğu belirtilmektedir. Bunu ödüllendirmemek için referans kelime, bir kelime eşleşmesi bulunduktan sonra yoksayılmalıdır. Bu işlem, düzenlenmiş unigram kesinliği (modified unigram precision) olarak adlandırılmıştır.
Çeviriler ancak referans cümlenin birebir aynısı olursa tam 1 skorunu alabilmektedir. Bu nedenle, bir insan çevirmen bile 1 skorunu alamayacaktır. Makalede 500 cümleden oluşan bir test metninde, bir insan çevirmen dört referansa karşı 0.3468, iki referansa karşı da 0.2571 skoruna ulaşmıştır.
BLEU Skorunun formülü Denklem 2.1.1'de verilmiştir. Formüldeki kesinlik (precision) değeri aday cümle ile referans cümlenin n-gram kesinlik değeridir (ortak n-gram sayısının toplam eşsiz n-gram sayısına bölümü).

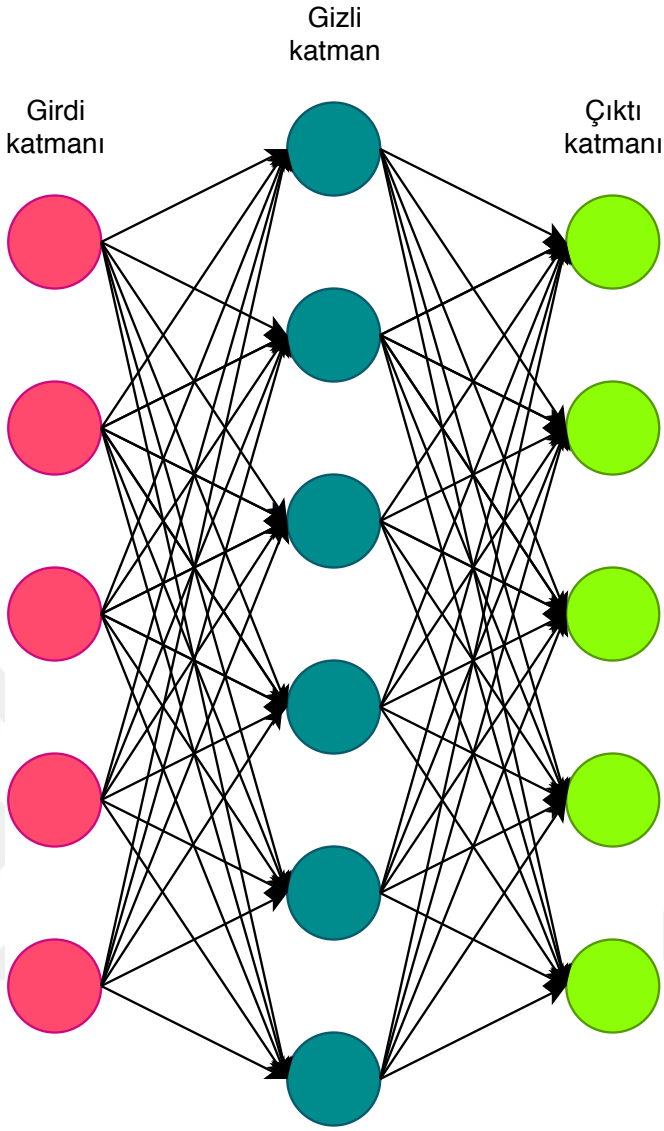
$$BLEU = min\left(1, \frac{\text{aday uzunluğu}}{\text{referans uzunluğu}}\right)\left(\prod_{i=1}^{n} \text{kesinlik}_i^{\frac{1}{n}}\right) \tag{2.1}$$

2.2 Yapay Sinir Ağları ve Türleri

Bu bölümde, tezin geri kalanında sıkça bahsedilecek olan Yapay Sinir Ağlarının (YSA) temellerinden ve özellikle makine çevirisi alanında tercih edilmiş olan türlerinden bahsedilecektir.

2.2.1 Tam Bağlantılı Yapay Sinir Ağları

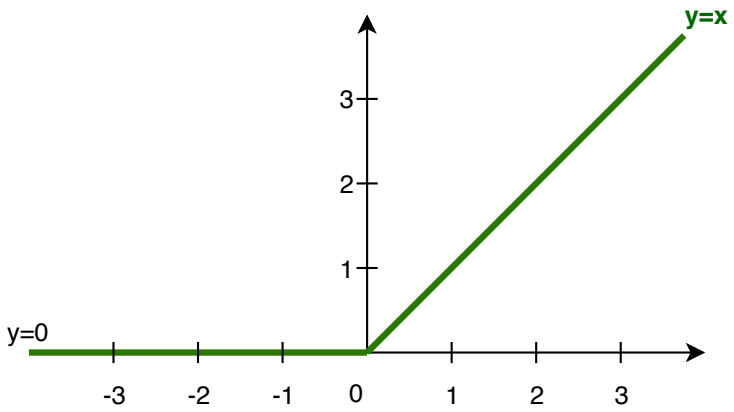
Derin öğrenmenin temeli sayılan Tam Bağlantılı YSA'lar art arda gelen birden fazla Tam Bağlantılı Katmandan oluşan YSA'lardır ve bu alandaki ilk çalışmalar Warren McCulloch [23] ve Walter Pitts [24] tarafından yapılmıştır. Bir Tam Bağlantılı katman, basitçe $\mathbb{R}^m$'den $\mathbb{R}^n$'ye bir fonksiyondur. Şekil 2.2' de de görüldüğü gibi katman içerisindeki nöronlar arasında bir bağlantı olmamasına rağmen, her katmandaki nöron bir önceki ve bir sonraki katmandaki bütün nöronlara bağlıdır.



Şekil 2.2: Tam Bağlantılı Yapay Sinir Ağı

Her katman, girdi vektörünün o katmandaki nöronların ağırlık vektörleriyle (katmanın ağırlık matrisiyle) çarpılması ve sonucun yanlılık değeri (bias) ile birlikte o nöronun aktivasyon fonksiyonundan geçmesi şeklinde ilerler. Birçok aktivasyon fonksiyonu bulunmaktadır, ancak bu çalışmada incelenen YMÇ modeli Dönüştürücü ReLU fonksiyonunu kullandığından sadece bu fonksiyondan bahsedilecektir.

ReLU'nun açılımı "Rectified Linear Unit"tir. YSA'larda en çok kullanılan aktivasyon fonksiyonlarından biridir. Matematiksel olarak $y = \max(0, x)$ şeklinde formülize edilir. Şekil 2.3'te fonksiyonun grafiği verilmiştir.



Şekil 2.3: ReLU aktivasyon fonksiyonu grafiği

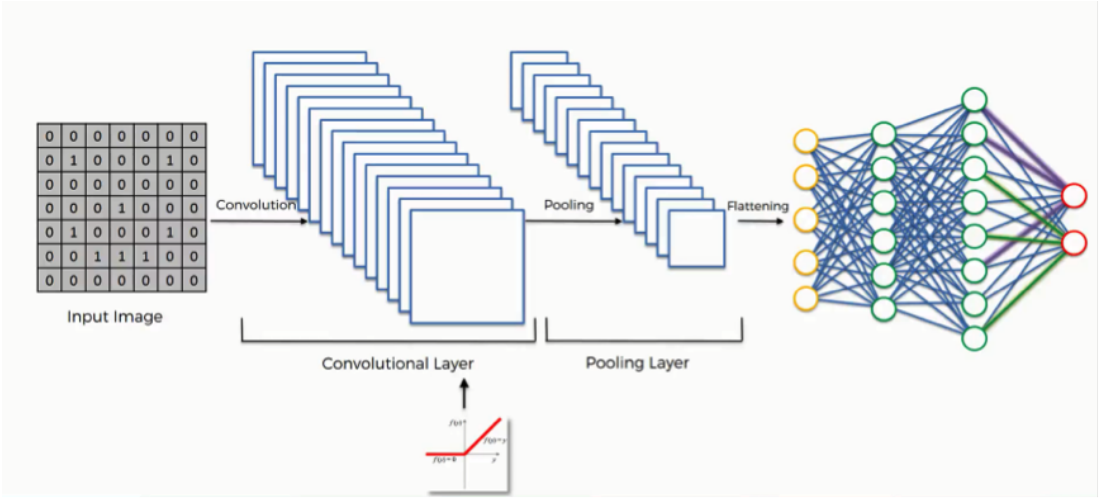
İleri Beslemeli YSA'daki temel fikir, sinir ağına ilk girdileri verdikten sonra katmanlar boyunca, her katmanın çıktısı bir sonrakinin girdisi olacak şekilde ağı besleyip son çıktıyı elde etmektir. Bir YSA'yı eğitirken ise temel amaç çıktının mümkün olduğunca "iyi" olmasını sağlamaktır. Önceden belirlenmiş bir hata fonksiyonu enküçüklenmeye (minimize) çalışılır. Basit problemlerde bu fonksiyon, genellikle asıl çıktı ile tahmin edilmiş çıktı arasındaki farktır.

2.2.2 Evrişimli Sinir Ağları (ESA)

Evrişimli Yapay Sinir Ağları (ESA), birçok yönden klasik yapay sinir ağlarına benzemektedir. Temelleri Fukushima vd.[25] ile Waibel vd. [26] tarafından atılmış ve günümüzdeki haline Lecun vd. [27]'nin yaptığı çalışmada kavuşmuştur. Girdi olarak, çoğunlukla resim alan ESA'lar üç temel katmandan oluşur: Evrişimli katman, örnekleme katmanı ve tam bağlantılı katman.

Resim girdisi 3 boyutlu (yükseklik, genişlik, derinlik) bir matrisle ifade edilir. Bu girdiye evrişimli katmanda filtreler veya özellik haritaları uygulanır, böylece girdinin belirli bölgelerine bağlı nöronların çıktıları elde edilir. Daha sonra resim girdisinin doğrusallığı ReLU fonksiyonu ile kaldırılır. Örnekleme aşamasında ise girdinin, uzaysal boyutlar (yükseklik ve genişlik) boyunca altörneklemesi yapılır. Bu katmanın çıktısı örneklenmiş bir özellik haritasıdır. Son aşama olan tam bağlantılı katman ise bu örneklenmiş özellik haritasını düzleştirir ve çıktı sınıflarının skorlarını hesaplar. Şekil 2.4'te ESA'ların genel mimarisi verilmiştir.

ESA'lar görüntü ve video tanımada, doğal dil işlemede ve tavsiye sistemlerinde oldukça başarılıdır. Ayrıca anlamsal ayrıştırma (semantic parsing) ve yorumlama tespitinde (paraphrase detection) işlerinde de çok iyi sonuçlar verir. Sinyal işleme ve resim sınıflandırmada da uygulamaları vardır.

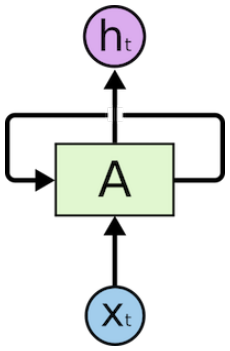


Şekil 2.4: Evrişimli Yapay Sinir Ağı Katmanları [28]

2.2.3 Özyineli Sinir Ağları (ÖSA)

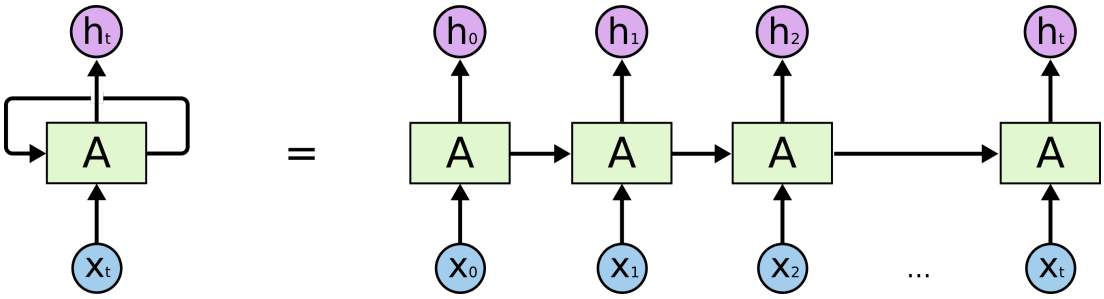
Temelleri Rumelhart vd. [29] tarafından atılan Özyineli Sinir Ağları (ÖSA; İngilizcesi, Recurrent Neural Networks-RNN-), önceki adımın çıktısının mevcut adımın girdisi olarak kullanıldığı sinir ağı türüdür. Klasik yapay sinir ağlarında tüm girdiler ve çıktılar birbirinden bağımsızdır, fakat bir cümlede sonraki kelimenin tahmin edilmesi gibi işlerde önceki kelimelerin "hatırlanması" gereklidir. Bu gibi işler için ÖSA'lar kullanılmaktadır.

ÖSA'larda bilginin korunmasını sağlamak için döngüler bulunmaktadır. Şekil 2.5'te ÖSA'nın t anındaki bir x_t girdisini işleyip h_t çıktısını veren bir parçası (A) gösterilmektedir.



Şekil 2.5: Bir Özyineli Sinir Ağının t anındaki bir x_t girdisini işleyip h_t çıktısını veren bir parçası [30]

Şekilde de görülen döngü, bilginin bir adımdan sonrakine aktarılmasını sağlar. ÖSA'lar aynı ağın (A) birden fazla kopyası şeklinde düşünülebilir, her ağ kendinden sonra gelene bir mesaj iletir. Şekil 2.6'da ağın döngü açıldığındaki görünümü verilmiştir.



Şekil 2.6: Şekil 2.5'teki ÖSA'nın döngü açıldığındaki görünümü [30]

Bu zincir şeklindeki yapı ÖSA'ların dizilerle ve listelerle ilgili olduğunu açıkça göstermektedir. Bu durumda bir metnin çevirisini yapmak için, her bir girdiyi o metindeki kelimeler olarak belirleyebiliriz. ÖSA önceki kelimelerin bilgisini, onu işleyecek ve kullanacak olan sonraki ağa aktarabilir.

2.2.4 Uzun Kısa Vadeli Bellek (UKVB)

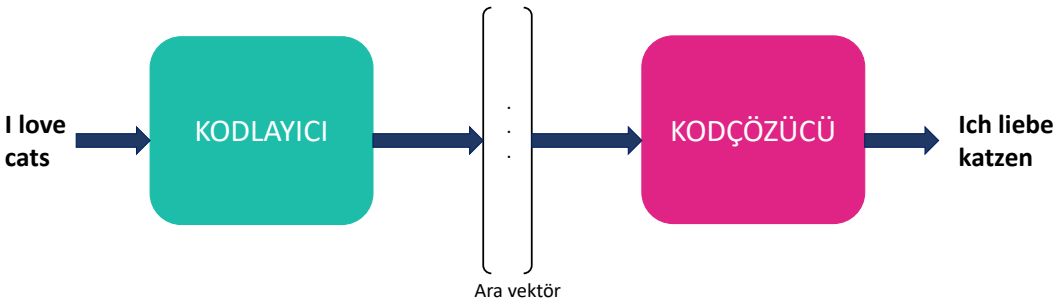
Bölüm 2.2.3'te de bahsedildiği gibi ÖSA'lar bir cümlenin sıradaki kelimesinin tahmini gibi işlerde kullanılabilir. Ancak kelimenin tahmini için, bağlam bilgisine ihtiyaç duyulabilir yani verilen metinde çok önceki cümlelere bakmak gerekebilir. Örneğin "Çocukluğumu Fransa'da geçirdim... Ana dilim gibi konuştuğum dillerden biri..." metninin son kelimesi "Fransızca'dır" olmalı. Bu bilgi metnin ilk cümlesine bakılarak anlaşılabilir. ÖSA'lar, ilgili bilgi ile bilgiye ihtiyaç duyulan yer arasındaki mesafe arttıkça başarısız olmaktadır. Çünkü zincir uzadıkça iletilen bilgi kaybolur.
Bu tarz işler için Hochreiter vd. [31]'nin geliştirdiği özel bir ÖSA türü olan Uzun Kısa Vadeli Bellekler (UKVB; İngilizcesi, Long Short Term Memory-LSTM-) kullanılır. ÖSA'lar ağa yeni bir bilgi eklediğinde bilginin ne kadar önemli olduğunun ayrımını yapmaz, bilgiyi direkt işler ve sonraki adıma iletir. UKVB'lerin ÖSA'lardan farkı birtakım çarpma ve toplama işlemleriyle bu ayrımı yapıyor olmalarıdır. Bu işlemler UKVB hücrelerinde yapılır ve hücre durumu adı verilen mekanizma önemli olan bilginin aktarılmasını, önemsiz bilginin unutulmasını sağlar.
Yine de, UKVB'ler cümle veya metin uzunlukları çok büyükse başarısız olmaktadır. Yani model, dizideki uzak pozisyonların içeriğini unutmaktadır. ÖSA'lar ve UKVB'lerdeki bir diğer sorun ise cümleler işlenirken kelime-kelime işlendiği için işlemi paralelleştirmek mümkün değildir [32].

2.3 Seq2Seq Modeller ve İlgi Mekanizmasına Giriş

İlk defa Google [33] tarafından kullanılan diziden-diziye modeller günlük hayatta kullandığımız birçok uygulamada karşımıza çıkar. Makine çevirisi (Google Çeviri vb.),

ses tanıma, video altyazısı oluşturma gibi işlerde bu model kullanılmaktadır. Klasik Derin Yapay Sinir Ağları, bu tarz dizi girdileri için ideal değildir, çünkü girdi ve çıktıların boyutlarının önceden bilinmesini ve sabit olmasını gerektirir. Diziden-diziye modeller, sabit uzunluktaki girdileri, farklı sabit uzunluktaki çıktılarla eşleştirmeyi amaçlar. Örneğin, Türkçede 3 kelimeden oluşan “Kedileri çok seviyorum.” cümlesinin İngilizcedeki karşılığı “I love cats so much.” 5 kelimeden oluşmaktadır. Bu çeviriyi girdileriyle aynı uzunlukta çıktı veren klasik UKVB ağlarıyla yapmak mümkün değildir, diziden-diziye modeller bunun gibi sorunları çözmek için kullanılır.

Bu modeller, kodlayıcı ve kodçözücü olmak üzere iki temel parçadan oluşur (Bu yüzden “Kodlayıcı-Kodçözücü Modeli” olarak da adlandırılır.). Kodlayıcı, girdi dizisini kelime kelime alır ve çıktı olarak bir ara vektör oluşturur (bir nevi cümlenin anlamını bu vektöre kodlar), kodçözücü bu ara vektörü girdi olarak alır ve hedef çıktı dizisini kelime kelime oluşturur.Şekil 2.7’de kodlayıcı-kodçözücü mimarisine sahip bir modelin genel görünümü verilmiştir.



Şekil 2.7: Kodlayıcı-Kodçözücü ile Makine Çevirisi

Kodlayıcı ve kodçözücü olarak genellikle Özyineli Sinir Ağları, özellikle de Uzun Kısa Vadeli Bellekler kullanılır. Kodlayıcı-kodçözücü mimarisinin iki büyük sorunu bulunur ve ikisi de uzunlukla ilgilidir. Öncelikle, bu mimarinin sınırlı belleği bulunmakta ve çevrilecek olan herhangi bir uzunluktaki cümlenin tamamı, kodlayıcının oluşturacağı sabit uzunluktaki bir ara vektöre sıkıştırılmaya çalışılmaktadır. İkinci olarak, yapay sinir ağı derinleştikçe eğitmek zorlaşır. ÖSA’lar için, dizinin uzunluğu arttıkça yapay sinir ağı derinleşir. Bu da loss fonksiyonunun gradiyentinin sıfıra yaklaşmasına yani "kaybolan gradiyentler" (İngilizcesi, vanishing gradients) problemine yol açar. Özellikle

bu problemi çözmek için tasarlanan ÖSA türlerinde (UKVB gibi) bile bu, hala büyük bir sorundur.

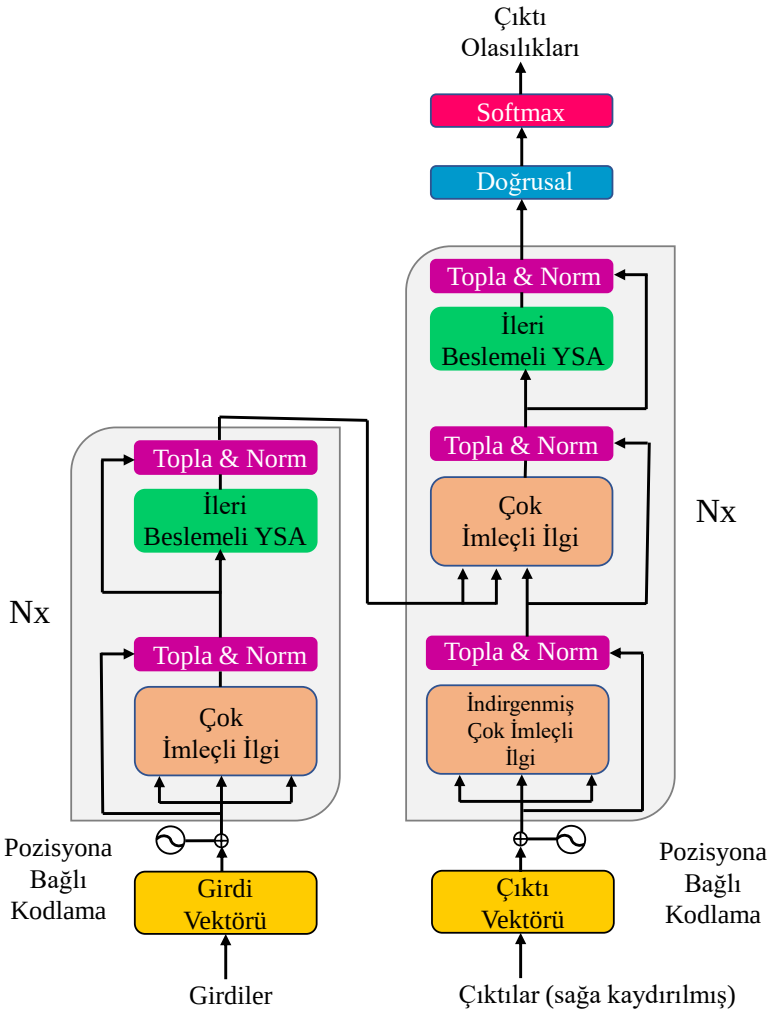
Bu sorunlar, insanların da çeviri yaparken kullandığı bir yöntemle çözülebilir: Girdi cümlesinin alakalı kısımlarına "ilgi" göstererek. Bir insan uzun bir cümle çevirirken, tüm detayları yakaladığından emin olmak için girdi cümlesinin belirli kısımlarına tekrar bakar. Yapay sinir ağlarının da bunu yapması "ilgi mekanizması" ile sağlanabilir. UKVB'nin önceki çıktıları saklanıp ileride kullanılarak yapay sinir ağının kapasitesi, UKVB yapısını değiştirmeden arttırılabilir. Ayrıca, cümlenin belirli bir kısmı ile ilgilendiğimizi biliyorsak, ilgi mekanizması bunu gerçekleştirmek için bir kısayoldur. Bu, adım sayısını azaltarak kaybolan gradiyentler problemini önler.

Mekanizma basitçe şu şekilde işlemektedir: kodlayıcıdan UKVB çıktıları alınıp saklandıktan sonra, kodçözücü tarafında, saklanmış her bir çıktının mevcut hesaplamayla ne kadar alakalı olduğu sorgulanır. Kodlayıcının her çıktısı bu sorgu sonucunda bir skora sahip olur ve bunlar Softmax fonksiyonu yardımıyla toplamı 1 olan bir olasılık dağılımına çevrilir. Bu dağılım kullanılarak kodlayıcı çıktılarının ağırlıklı toplamı olan (ne kadar alakalı olduklarını gösteren) ara vektör elde edilir. İlgi mekanizmasının olumsuz yönü; artık her kodçözücü çıktısı için, kodlanmış cümlenin tamamı üzerinde bu hesaplamanın yapılması gerekliliğidir. Bu, tek bir cümlenin çevirisi için büyük bir sorun değilken daha büyük girdiler için problem olabilir.

2.4 Dönüştürücü

Yukarıda da anlatıldığı gibi İlgi Mekanizması birçok modern derin öğrenme modelinde karşımıza çıkan yaygın bir metottur. İlgi Mekanizması, makine çevirisi uygulamalarının başarımını (performansını) arttırmaya yardımcı olur. Bu çalışmada, ilgi mekanizmasını modelin eğitilme hızını arttırmak için kullanan Dönüştürücü modeli üzerinde durulmuştur. Dönüştürücü'nün en büyük artısı, paralelleşmeye yardımcı olmasıdır. Hatta Google Cloud, Cloud TPU kullanımında referans model olarak Dönüştürücü'yü önermektedir. Şimdi Dönüştürücü modelini ve çalışma prensiplerini daha yakından inceleyelim:

Bir önceki bölümde kodlayıcı ve kodçözücü yapılarından bahsedildi. Burada kodlayıcı, bir sembol gösterimi girdi dizisi olan $x = (x_1, x_2, \ldots, x_n)$ 'yi bir sürekli gösterim dizisi olan $z = (z_1, z_2, \ldots, z_n)$ 'ye eşler. Kodçözücü z'yi girdi olarak alıp sembol gösterimi çıktı dizisi olan $y = (y_1, y_2, \ldots, y_m)$'yi her seferinde bir eleman çıkacak şekilde oluşturur. Model her adımda özbağlanımlı davranır; yani yeni sembol üretirken, girdi olarak daha önce üretilmiş sembolleri de alır. Kabaca bu şekilde işleyen Dönüştürücü modelindeki kodlayıcılar da kodçözücüler de üst üste eklenmiş ilgi mekanizması ve pozisyonlu tam bağlantılı ileri beslemeli yapay sinir ağı katmanlarından oluşur. Şekil 2.8'de Dönüştürücü modelinin yapısı ayrıntılı olarak gösterilmektedir.



Şekil 2.8: Dönüştürücü Modeli

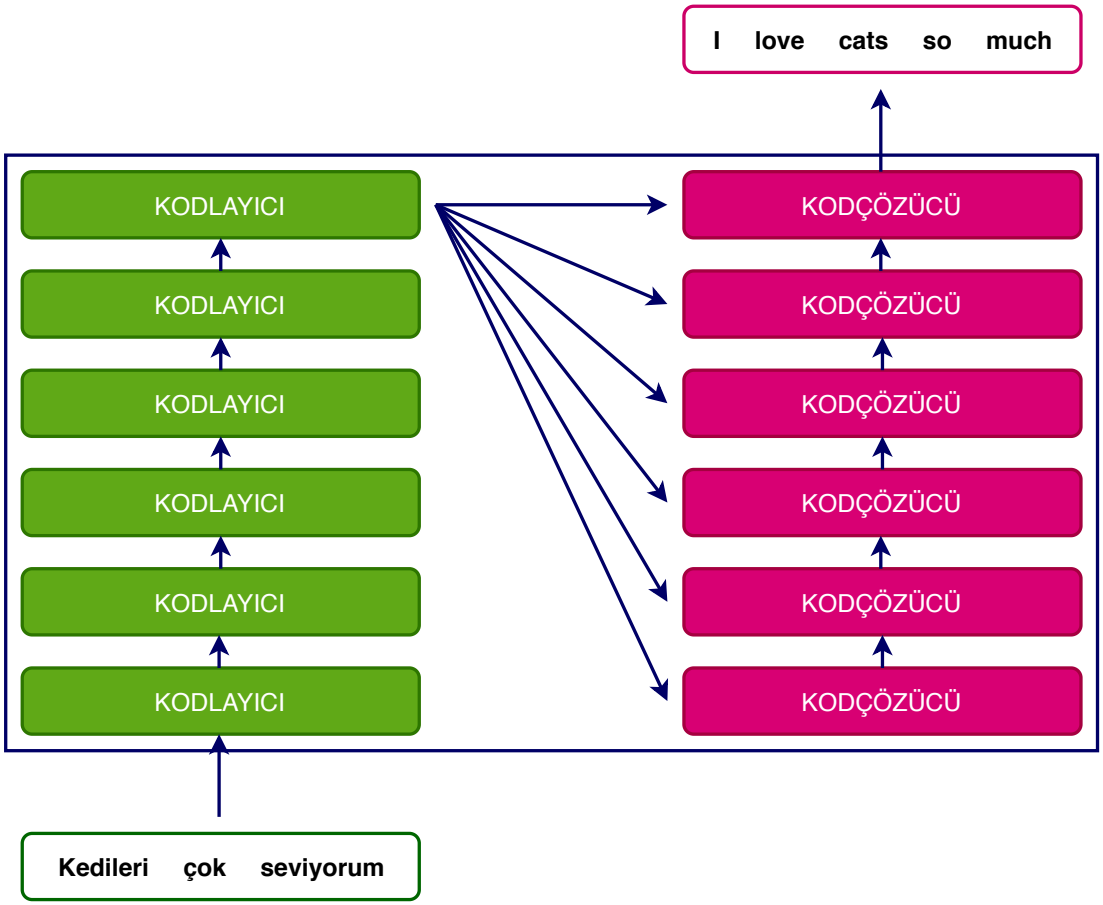
2.4.1 Kodlayıcı ve kodçözücü

Şekil 2.9'da [34] da görüldüğü gibi kodlayıcı $N = 6$ özdeş katmandan oluşur (sadece ağırlıkları aynı değildir.). Her katmanın iki alt-katmanı vardır. İlki, kodlayıcının belli bir kelimeyi çevirirken girdi cümlesindeki diğer kelimelere de bakmasını sağlayan katman olan çok-imleçli özilgi mekanizması; ikincisi ise basit, pozisyonlu tam bağlantılı ileri beslemeli yapay sinir ağıdır. Her iki alt-katmanın da etrafına bir artık bağlantı uygulanır ve hemen ardından katman normalizasyonu yapılır.

Artık bağlantıda, alt katmanın girdisi, aynı alt katmanın çıktısına eklenir ve yeni çıktı bu toplam olur. Yani her alt-katmanın çıktısı $KatmanNorm(x + Altkatman(x))$ şeklindedir. Burada $Altkatman(x)$, alt-katmanın kendisi tarafından ger- çeklenen fonksiyondur. Bu artık bağlantılardan yararlanmak için, gömme katmanları dahil modeldeki tüm katmanlar boyutu $d_{model} = 512$ olan çıktılar üretir.

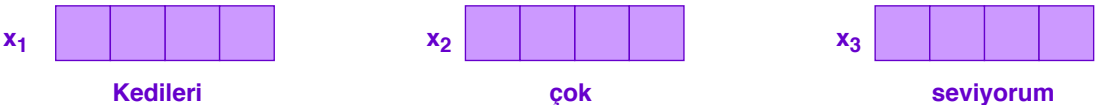
Kodçözücü de aynı şekilde $N = 6$ özdeş katmandan oluşur. Kodçözücüde, kodlayıcıdaki

iki alt-katmana ek olarak, ikisinin arasında üçüncü bir alt-katman daha bulunur.



Şekil 2.9: Dönüştürücü'de Kodlayıcı ve Kodçözücü Yığıtı

Bu alt-katman, kodlayıcının çıktısı üzerinde, kodlayıcının girdi cümlesinin alakalı yerlerine bakmasını sağlayan, çok imleçli ilgi işlemi uygular. Kodlayıcıya benzer olarak, her alt-katmanın etrafına bir artık bağlantı uygulanır ve hemen ardından katman normalizasyonu yapılır. Kodçözücüdeki özilgi alt-katmanı, sonraki pozisyonlara bakılmasını önlemek için maskelenmiştir.
Çeviri yaparken ilk önce, girdi cümlesindeki tüm kelimeler bir gömme algoritması kullanılarak kelime gömmelerine dönüştürülür (Şekil 2.10).



Şekil 2.10: Her kelime 512 boyutlu vektörlere gömülür.

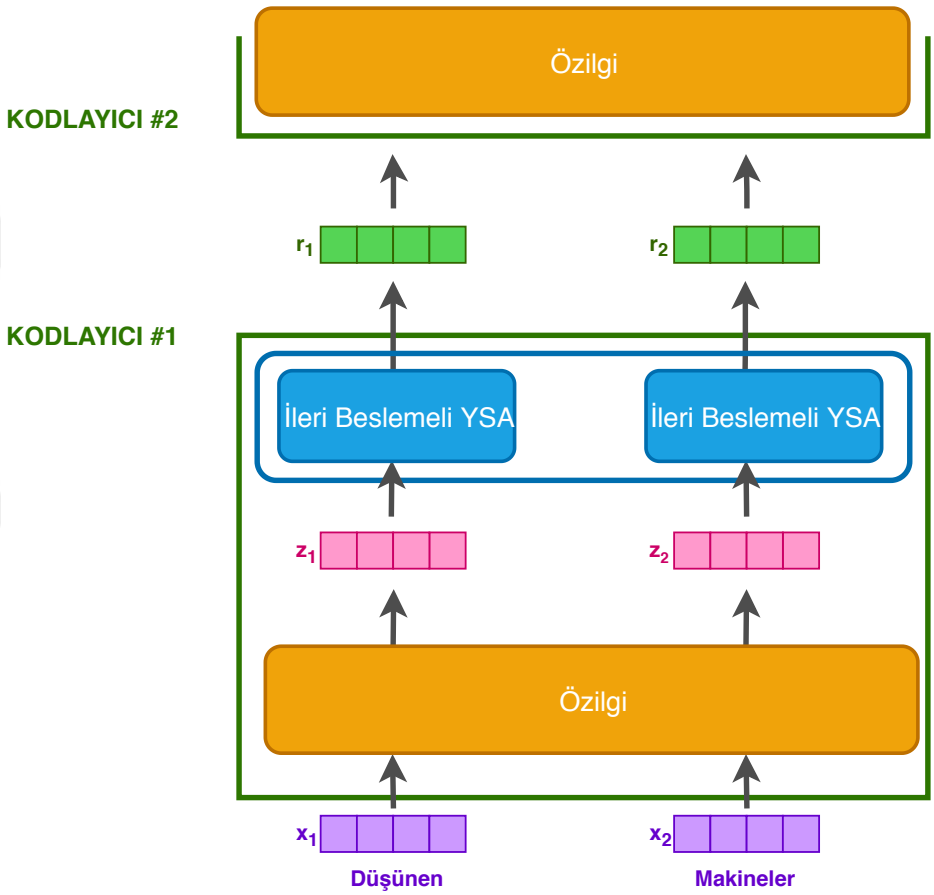
Dönüştürücü'de bu gömmelerin boyutu 512'dir ve ilk kodlayıcının girdisi olarak boyutu 512 olan bu vektörler listesi kullanılır. Bu listenin boyutu, değiştirilebilen

bir hiperparametredir ve genellikle girdi metnindeki en uzun cümlenin kelime sayısı olarak belirlenir. Daha sonraki kodlayıcılar ise kendinden önceki kodlayıcının çıktısını girdi olarak kullanır.

Dönüştürücü'nün en önemli özelliklerinden biri olarak; her pozisyondaki kelime, kodlayıcının içinde kendi yolundan ilerler.

Özilgi katmanında bu yollar arasında bağımlılıklar bulunur, fakat İleri Beslemeli YSA katmanı bu bağımlılıklara sahip değildir. Bu yüzden birçok yol, İleri Beslemeli YSA katmanından geçerken paralel çalıştırılabilir.

Şekil 2.11'de görüldüğü üzere ilk kodlayıcı girdi olarak bir vektör listesi alır.



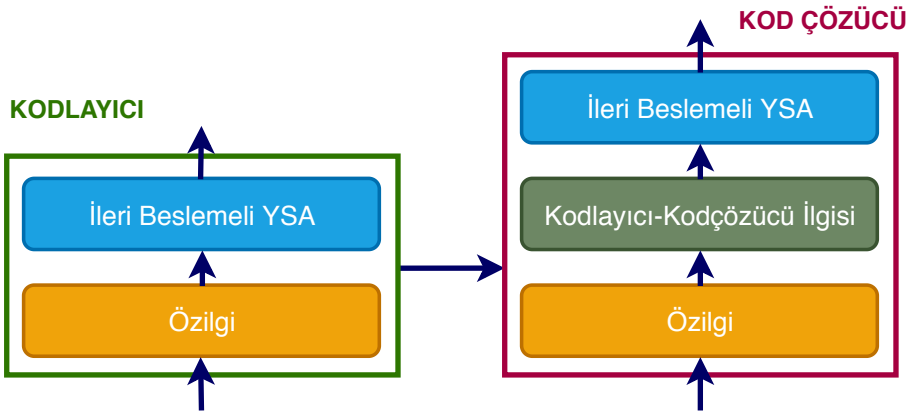
Şekil 2.11: Kodlayıcının Çalışma Prensibi

Bu listeyi içindeki vektörleri özilgi katmanına göndererek işler, daha sonra her birini ayrı ayrı İleri Beslemeli YSA katmanına gönderir (aynı ağ, ancak her vektör ayrı geçiyor.), bunun çıktısını ise sonraki kodlayıcı alır.

Şekil 2.12'de kodlayıcı ve kodçözücünün yukarıda bahsedilen bileşenleri ve iş akışı gösterilmektedir.

Özet olarak girdi cümlesi sırasıyla Özilgi ve İleri Beslemeli YSA katmanlarından oluşan kodlayıcıdan geçecektir. Kodlayıcı çıktısı, kodçözücüye yönlendirilecek ve her

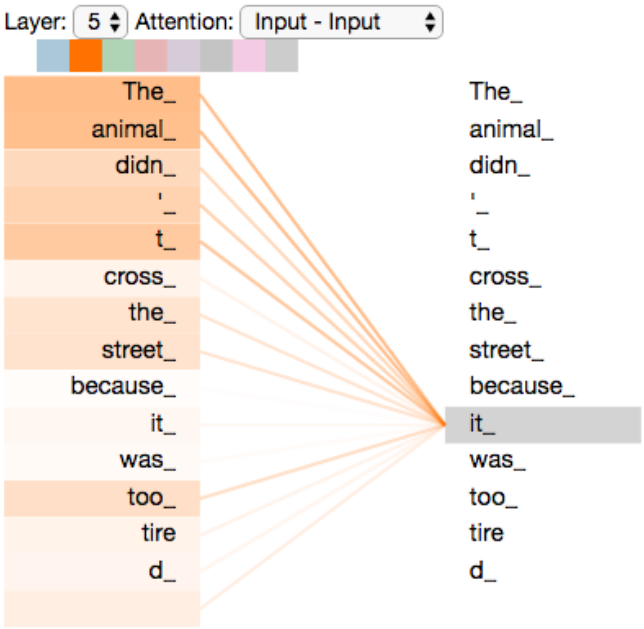
bir çıktı kelimesi için sırasıyla Özilgi, Kodlayıcı-Kodçözücü İlgisi ve İleri Beslemeli YSA katmanlarından oluşan kodçözücü çalıştırılacaktır.



Şekil 2.12: Dönüştürücü'de Kodlayıcı ve Kodçözücü İçeriği

2.4.2 İlgi mekanizması

İlgi Mekanizmasının işleyişini Şekil 2.13 'te de gösterilen bir cümle üzerinde inceleyelim:



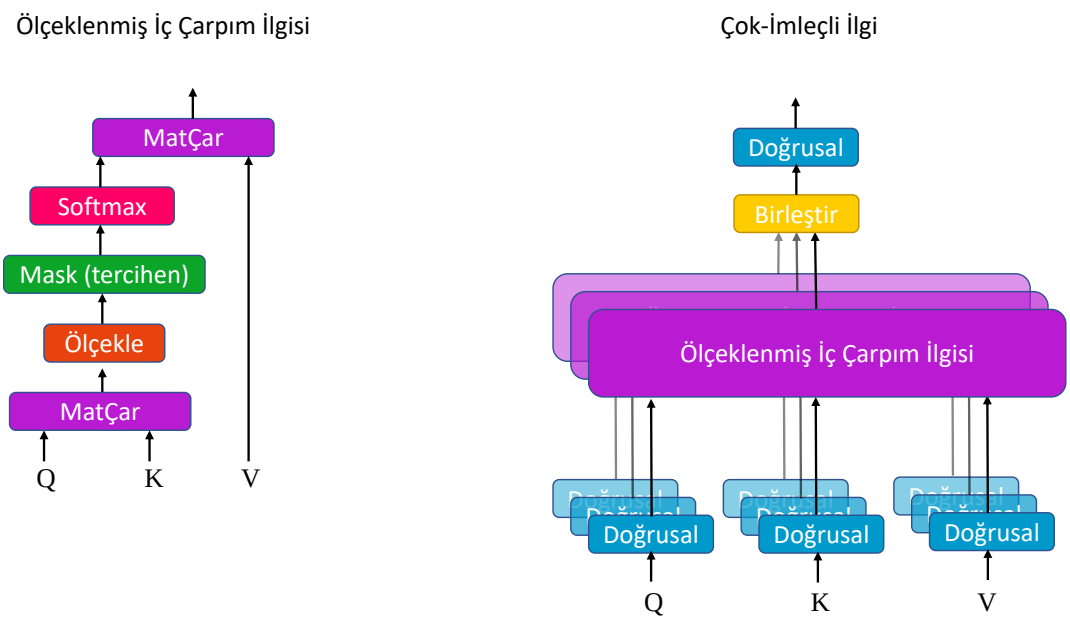
Şekil 2.13: İlgi Mekanizması Görselleştirmesi

"The animal didn't cross the street because it was too tired" cümlesinde "it" kelimesi "street"i mi yoksa "animal"ı mı kastetmektedir? Özilgi, modelin "it" kelimesini işlerken "animal" ile ilişkilendirmesine yardımcı olur. Model her kelimeyi işledikçe, özilgi

modelin diğer pozisyonlardaki kelimelere bakmasını ve bu kelimenin daha iyi kodlanmasını sağlar.
Bir ilgi fonksiyonu bir sorgunun ve anahtar-değer çiftlerinin bir çıktıya eşlenmesi olarak tanımlanabilir. Burada, sorgu, anahtarlar, değerler ve çıktı birer vektördür. Çıktı, değerlerin ağırlıklı toplamıyla hesaplanır. Bahsedilen ağırlıklar, her değer için, sorguyu ilgili anahtar değeri ile karşılaştıran bir uyumluluk fonksiyonu ile hesaplanır.

2.4.2.1 Ölçeklenmiş iç çarpım ilgisi

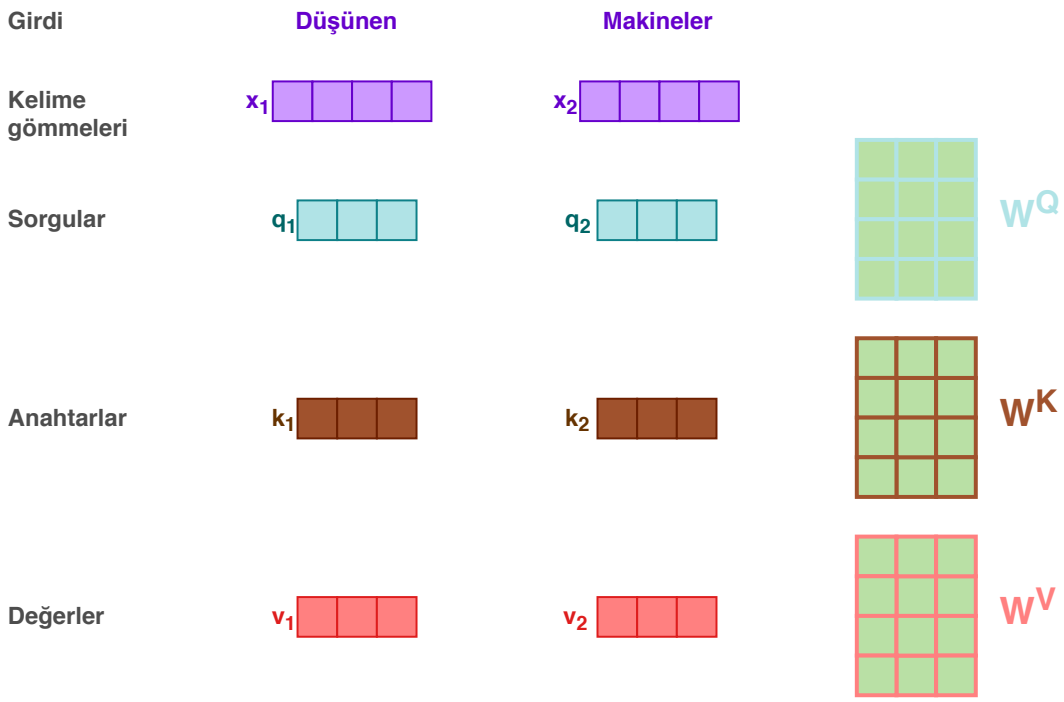
Dönüştürücü modelinde ”Ölçeklenmiş İç Çarpım İlgisi” olarak isimlendirilen bir ilgi türü kullanılmıştır (Şekil 2.14).



Şekil 2.14: İlgi Türleri

Özilgi hesaplanırken ilk adım, kodlayıcının girdi vektörlerinden (her kelimenin gömmesinden) üç vektör oluşturmaktır. Her bir kelime için bir sorgu vektörü, bir anahtar vektörü ve bir değer vektörü oluşturulur (sorgular ve anahtarlar d_k boyutunda, değerler d_v boyutunda). Bu vektörler, kelime gömmesinin eğitim süreci boyunca eğitilmiş olan üç matrisle çarpılarak elde edilir (Şekil 2.15).
x_1'in W^Q ağırlık matrisi ile çarpılmasıyla q_1 sorgu vektörü, W^K ağırlık matrisi ile çarpılmasıyla k_1 anahtar vektörü, W^V ağırlık matrisi ile çarpılmasıyla v_1 değer vektörü

elde edilmiştir. Bu işlem her kelime için yapılır. Daha sonra, her girdi kelimesi için skor hesaplanır. Burada bahsedilen vektörlerin, bir tasarım kararı olarak boyut olarak kelime gömmelerinden küçük olduğuna dikkat edilmesi gerekmektedir. Kelime gömmeleri ve kodlayıcı girdi/çıktı vektörlerinin boyutu 512 iken bahsedilen vektörlerin boyutu 64'tür. Sorgu, anahtar ve değer vektörleri, ilgi hesaplaması için kullanılan soyutlamalardır. İlginin nasıl hesaplandığının anlaşılması, bu vektörlerin görevlerinin anlaşılmasını sağlayacaktır:
Diyelim ki Şekil 2.15'teki örnekte "Makineler" kelimesi için özilgi hesabı yapılacak. Girdi cümlesindeki her kelimenin bu kelimeye karşı skoru bulunmalıdır.



Şekil 2.15: Sorgu, anahtar ve değer vektörlerinin elde edilişi. Bu işlem her kelime için yapılır.

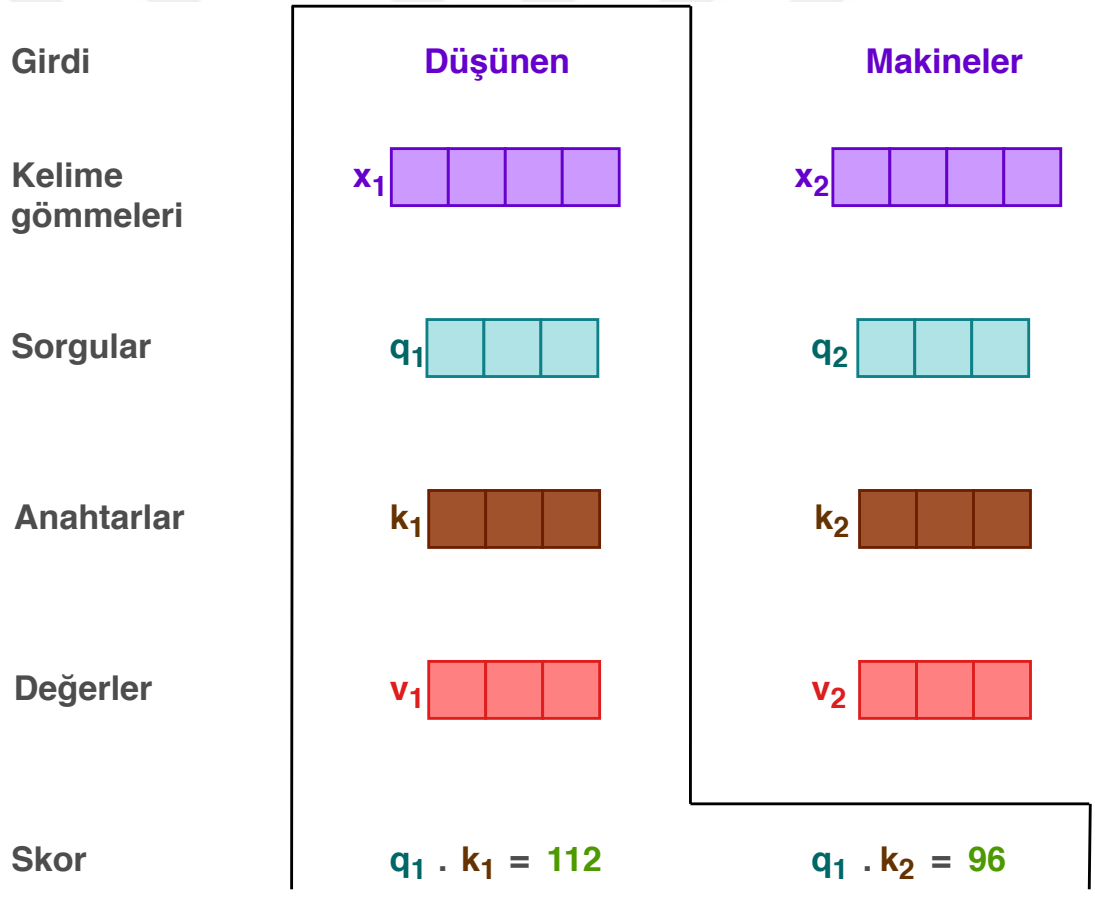
Bu skor, belli bir pozisyondaki kelimeyi kodlarken girdi cümlesinin diğer kısımlarına ne kadar odaklanılması gerektiğini gösterir.
Bir kelimenin skoru, kodlanan kelimenin sorgu vektörünün sorgulanan kelimenin anahtar vektöriyle iç çarpımı alınarak hesaplanır. Örneğin girdi cümlesindeki birinci pozisyondaki kelime için özilgi işlemi yapılırken ilk skor q_1 ve k_1'in iç çarpımı, ikinci skor q_1 ve k_2'nin iç çarpımı ile bulunur (Şekil 2.16).
Daha sonraki aşamada bulunan değerler $\sqrt{d_k} = 8$'e bölünür (bu daha kararlı gradiyentler elde edilmesini sağlar) ve değerlerin ağırlıklarını elde etmek için bulunan sonuçlara softmax fonksiyonu uygulanır (Şekil 2.17).
Softmax fonksiyonu skorları normalize eder, böylece hepsi pozitif ve 1'den küçük

olur. Softmax fonksiyonu çıktısı her kelimenin o pozisyonda ne kadar etkili olduğunu gösterir. Sonrasında her kelimenin softmax skoru o kelimenin değer vektörü ile çarpılır (Şekil 2.18).

Bu işlemin amacı, yüksek skorlu, yani odaklanmak istediğimiz kelimelerin etkisini koruyup alakasız kelimelerin etkisini azaltmaktır ($0,001$ gibi küçük sayılarla çarparak). Son olarak ağırlıklı değer vektörleri toplanır. Bu toplam, o pozisyondaki özilgi katmanı çıktısını verir.

Pratikte tüm bu işlemler birden fazla sorgu üzerinde matris çarpımlarıyla yapılır. Sorgu matrisi Q, anahtarlar matrisi K ve değerler matrisi V olmak üzere, aşağıdaki hesaplama ile çıktılar matrisi elde edilir:

$$Ilgi(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.2}$$



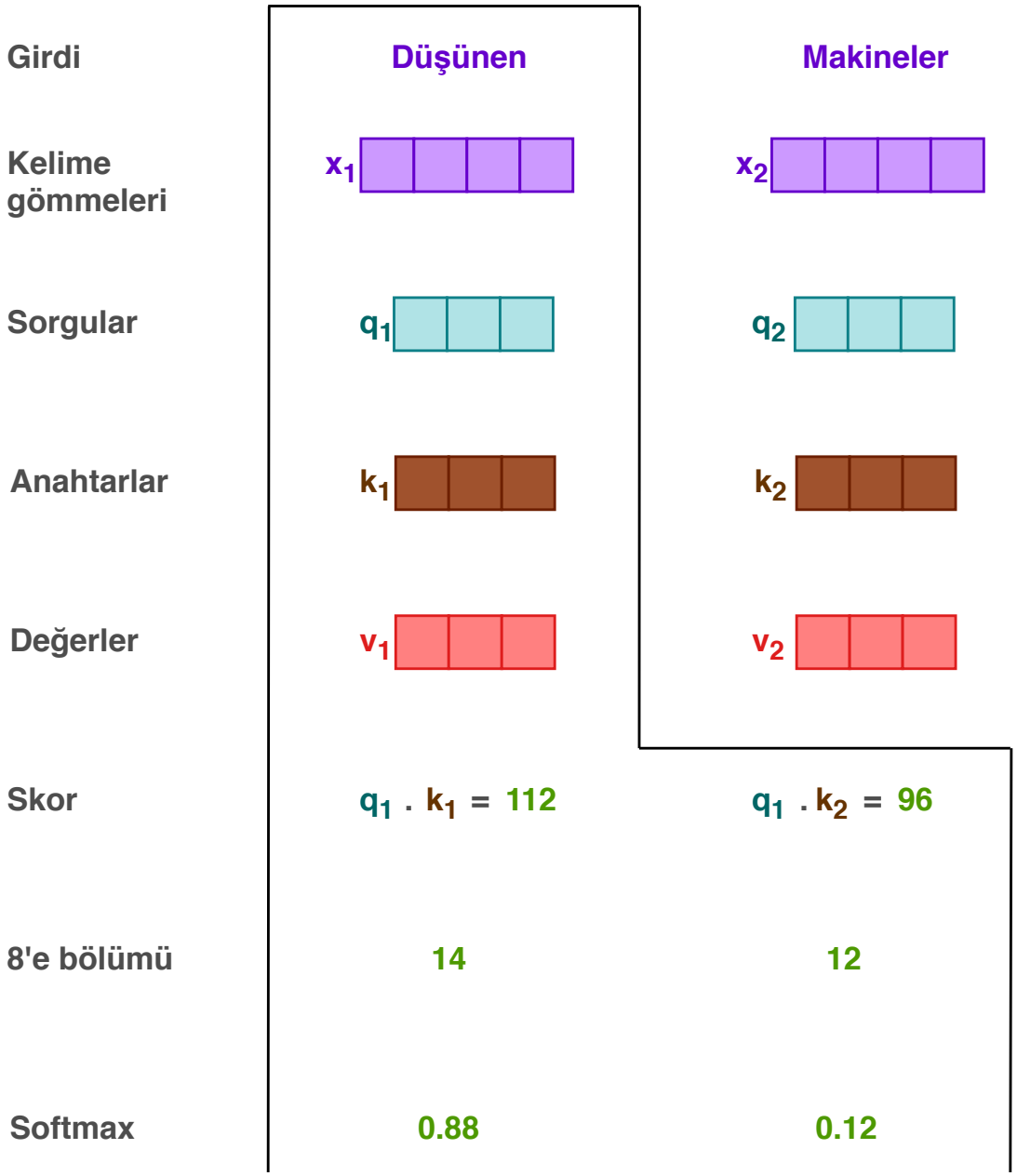
Şekil 2.16: Skor hesabı

En çok kullanılan ilgi fonksiyonları eklemeli ilgi [35] ve iç çarpım ilgisidir.

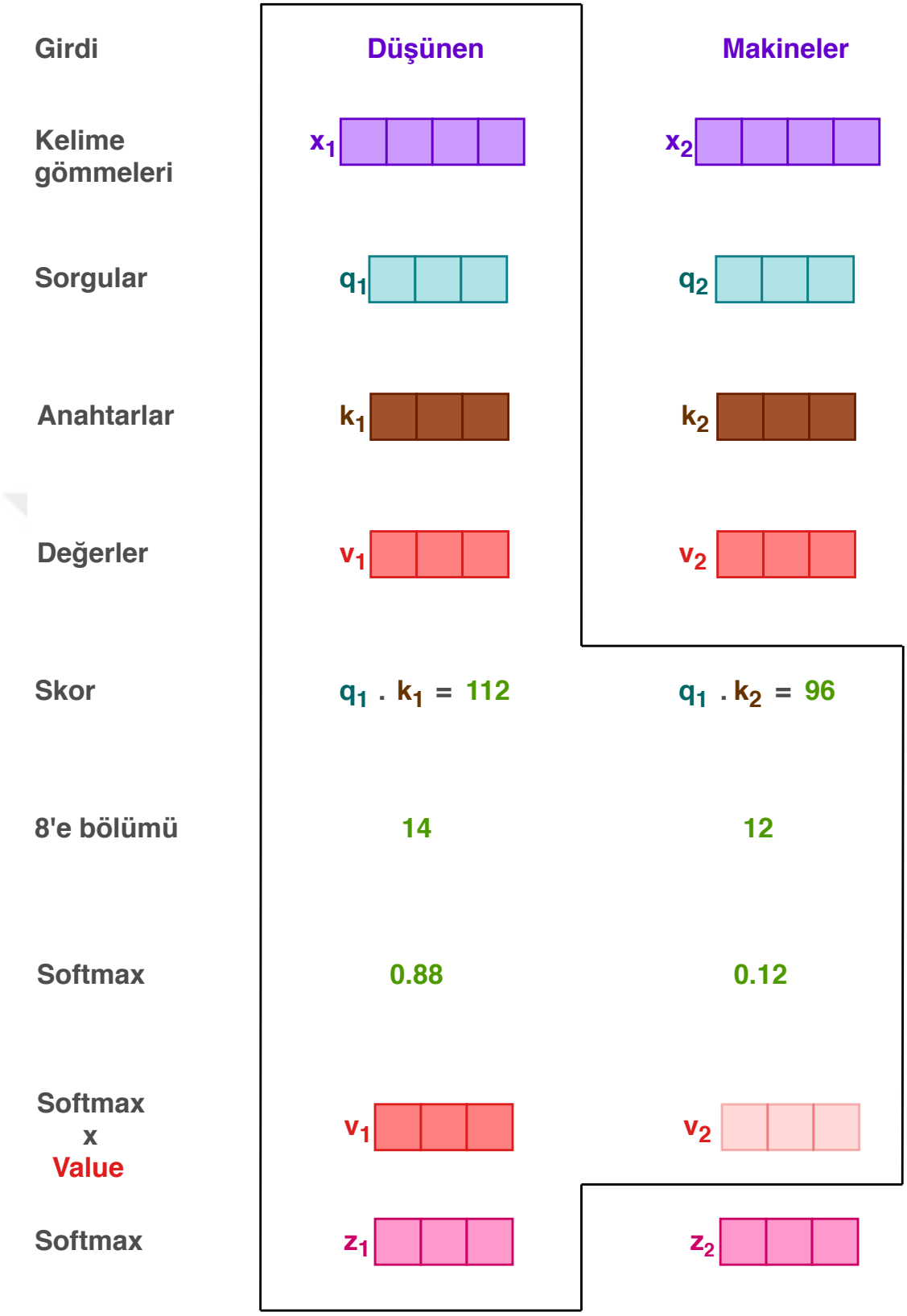
İç çarpım ilgisi yukarıda anlatılan algoritmanın $\sqrt{d_k}$'ya bölünme kısmının bulunmadığı halidir. Eklemeli ilgi ise tek bir gizli katman içeren ileri beslemeli YSA kullanarak

uyumluluk fonksiyonunu hesaplar. Teorik karmaşıklık olarak ikisi benzer olsa da, iç çarpım ilgisi pratikte daha hızlı ve daha alan-verimlidir (space-efficient) çünkü yüksek oranda optimize edilmiş matris çarpımı kodlaması kullanılarak gerçeklenebilir.

d_k'nın küçük değerleri için iki mekanizma da benzer performans gösterirken, d_k'nın büyük değerlerinde eklemeli ilgi ölçeklemesiz olarak iç çarpım ilgisini geçer. Yazarlar bunun sebebi olarak, d_k'nın büyük değerleri için iç çarpımların hızlı büyümesini göstermektedir. Bu, softmax fonksiyonunun çok küçük gradiyentlere sahip olan bölgelerde sonuçlanmasına sebep olur.



Şekil 2.17: Skorlar $\sqrt{d_k}$'ya bölünür ve bulunan sonuçlara softmax fonksiyonu uygulanır.



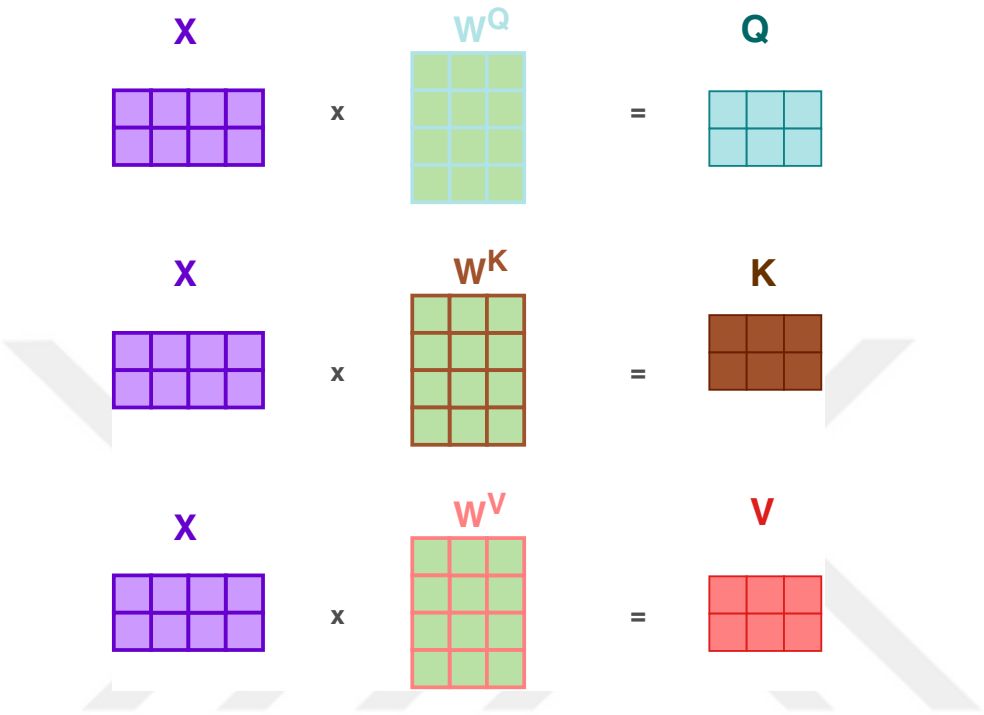
Şekil 2.18: "Düşünen" kelimesi için özilgi katmanı çıktısının elde edilişi.

İç çarpımların hızlı büyümesini örneklemek gerekirse; q ve k ortalamaları 0, varyansları

1 olan bağımsız rastgele değişkenler olsun. İç çarpımları $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$'nin ortalaması 0, varyansı d_k'dir.

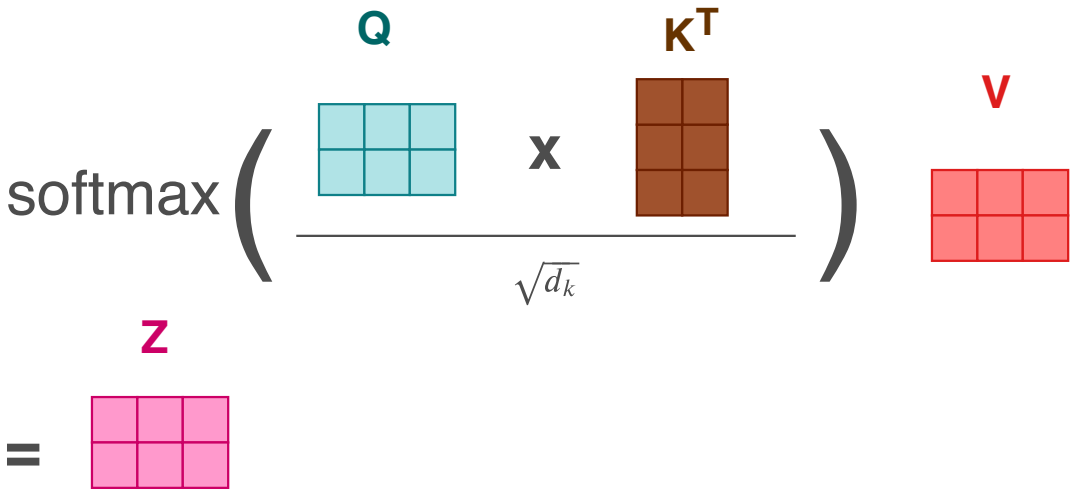
Sonuç olarak buna karşı koymak için iç çarpımlar $\frac{1}{\sqrt{d_k}}$ ile ölçeklendirilmiştir.

Şekil 2.19'da X matrisinin her satırı girdi cümlesinin bir kelimesine karşılık gelir.



Şekil 2.19: Özilgi katmanı matris hesaplaması.

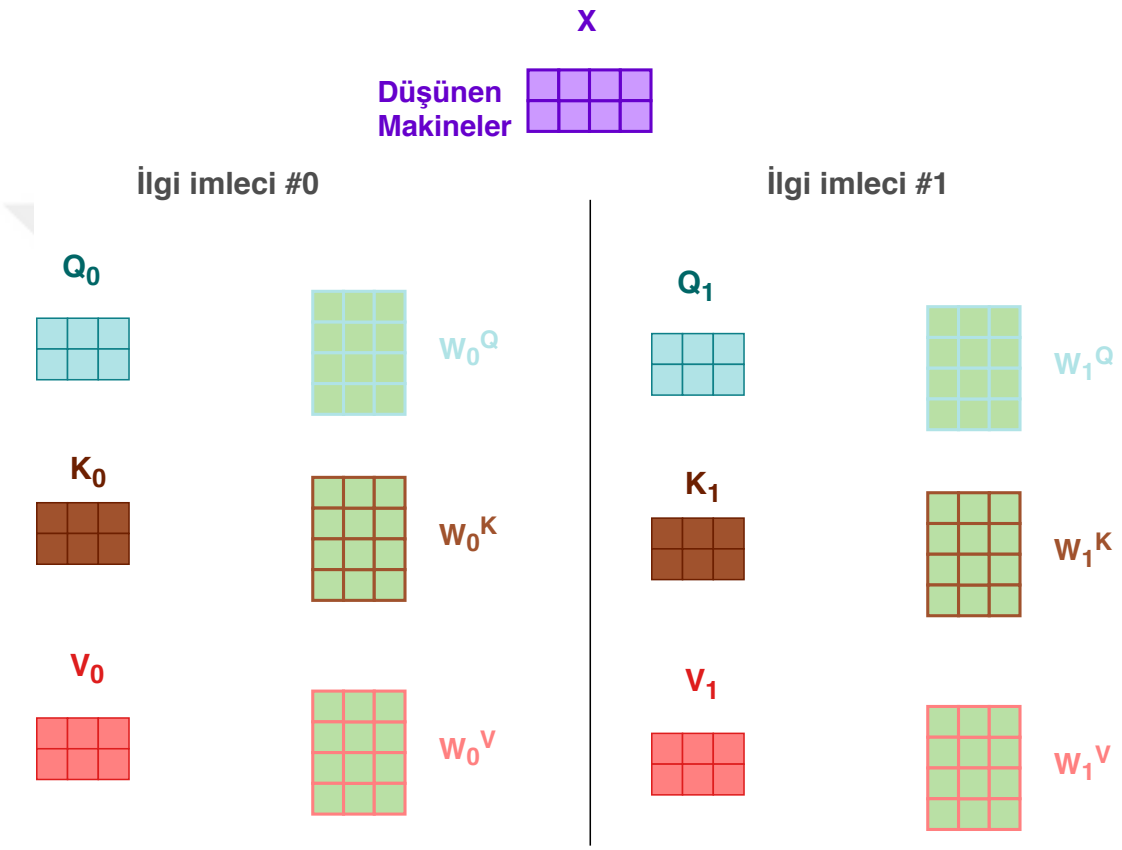
Şekil 2.20'de özilginin matrislerle hesabı tek bir formül olarak görselleştirilmiştir.



Şekil 2.20: Özilginin matrislerle hesaplaması.

2.4.2.2 Çok-İmleçli İlgi

d_{model} boyutundaki anahtarlar, değerler ve sorgularla tek bir ilgi fonksiyonu hesaplamak yerine, anahtarlar, değerler ve sorguların h adet farklı, öğrenilmiş doğrusal izdüşümlerinin (sırasıyla d_k, d_k ve d_v boyutlarına) kullanılması daha etkilidir. Burada her bir imleç için ayrı W^Q, W^K, W^V ağırlık matrisleri kullanılır, böylece her imleç için farklı Q, K, V matrisleri elde edilir (Şekil 2.21). Anahtarlar, değerler ve sorguların bu izdüşümlenmiş versiyonlarının her birinde paralel olarak ilgi fonksiyonu yürütülür.



Şekil 2.21: Çok-İmleçli İlgi için Q, K, V hesabı.

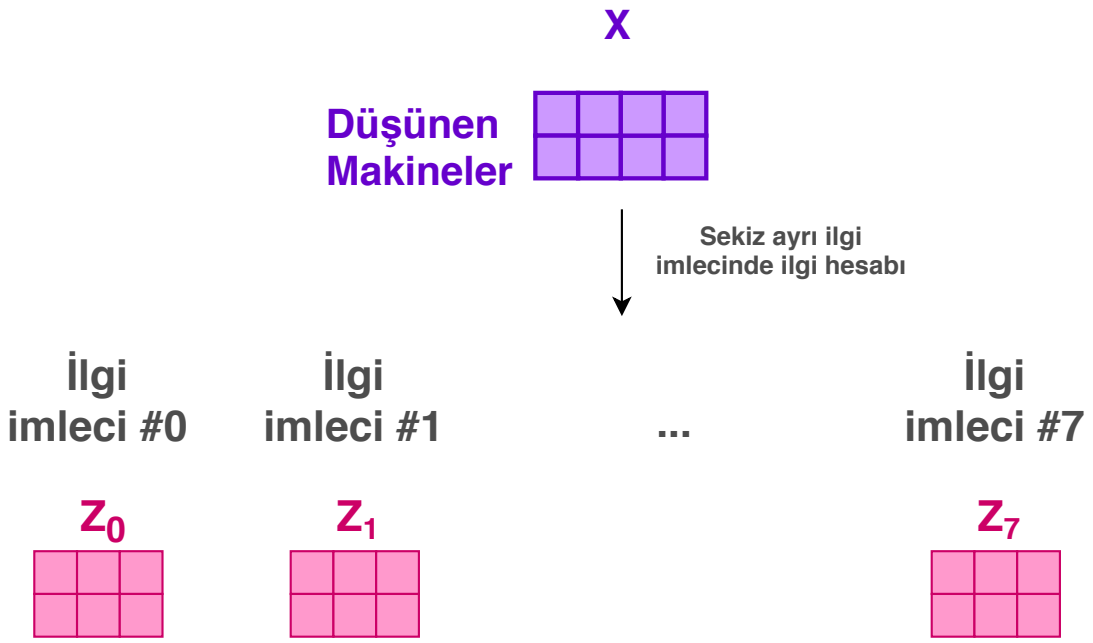
Elde edilen d_v boyutlu çıktılar uç uca eklenir ve bir kere daha izdüşümü alınır. Bu işlemi aşağıdaki gibi denklemle ifade edebiliriz:

$$CokImlecliIlgi(Q,K,V) = Birlestir(imlec_1,...,imlec_h)W^O,$$
$$imlec_i = Ilgi(QW_i^Q, KW_i^K, VW_i^V)$$

İzdüşümler $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ ve $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ parametre matrisleridir.

Dönüştürücü'de $h = 8$ tane paralel ilgi katmanı/imleci bulunmaktadır. Bunların her biri için $d_k = d_v = d_{model}/h = 64$ olarak belirlenmiştir. Her imleçteki boyutlar azaltıldığı için, toplam hesaplama maliyeti tam boyutlu tek bir imleç kullanmayla aynıdır.

Yani her bir imleç için yukarıda anlatılan özilgi işlemi farklı ağırlık matrisleriyle uygulanacak ve $h = 8$ adet farklı Z matrisi elde edilecektir (Şekil 2.22).



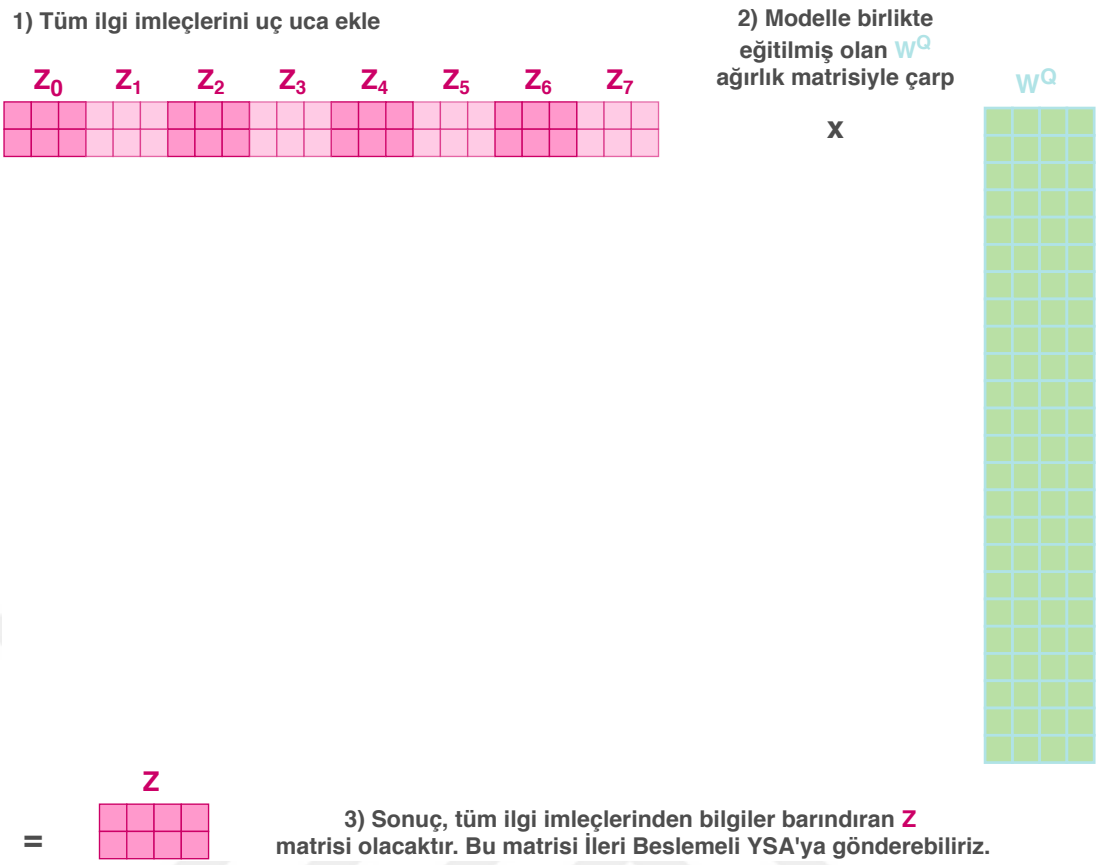
Şekil 2.22: Çok-İmleçli İlgi için Z matrisleri.

Ancak İleri Beslemeli YSA katmanı birden fazla matris girdisi beklememektedir. Bu yüzden elde edilen bu $h = 8$ matris, bir şekilde tek bir matrise sıkıştırılmalıdır. Matrisler uç uca eklenip yeni bir W^O ağırlık matrisi ile çarpılarak İleri Beslemeli YSA katmanının beklediği boyutta bir Z matrisi elde edilir (Şekil 2.23).

Bu yöntem İlgi katmanının başarımını iki yönden arttırır:

Öncelikle modelin farklı pozisyonlara odaklanma kabiliyetini yükseltir. Yukarıda örnek olarak verilen "The animal didn't cross the street because it was too tired" cümlesinde "it" kelimesinin neyi kastettiğinin anlaşılması gibi işlerde kullanışlı olacaktır.

Ayrıca ilgi katmanına birden fazla "temsil alt-uzayı" verir. Çok-İmleçli İlgi uygulanırken birden fazla (Dönüştürücü için 8 tane) W^Q, W^K, W^V ağırlık matrisi kümesi bulunmaktadır. Bunların ilk değerleri rastgele atanmaktadır. Eğitildikten sonra her bir küme, girdi cümlesinin gömmelerini farklı bir alt-uzaya yansıtır.



Şekil 2.23: Çok-İmleçli İlgi için Z matrisinin elde edilişi.

2.4.2.3 Dönüştürücü'deki İlgi Türleri

Dönüştürücü Çok-İmleçli İlgiyi üç farklı şekilde kullanır:

- "Kodlayıcı-Kodçözücü İlgi" katmanlarında sorgular bir önceki kodçözücü katmanından, anahtarlar ve değerler de kodlayıcının çıktısından gelir. Böylece kodçözücüdeki her pozisyon, girdi dizisindeki bütün pozisyonlara bakar. Bu, diziden-diziye modellerdeki klasik kodlayıcı-kodçözücü ilgi mekanizmasının aynısıdır [36, 37, 35]

- Kodlayıcıda özilgi katmanları bulunur. Bir özilgikatmanında, tüm anahtarlar, değerler ve sorgular aynı yerden (kodlayıcıdaki bir önceki katmandan) gelir. Kodlayıcıdaki her pozisyon, kodlayıcının önceki katmanlarındaki tüm pozisyonlara bakar.

- Benzer şekilde, kodçözücüdeki özilgi katmanları, kodçözücüdeki her pozisyonun kendine ve kendinden önceki tüm pozisyonlara bakabilmesini sağlar. Oto-bağlanım özelliğinin korunması için sola doğru bilgi akışının engellenmesi gerekir. Bu özellik, ölçeklenmiş iç çarpım ilgisinin içinde maskeleme yapılarak ($-\infty$'a eşitle-

nerek) gerçeklenmiştir. Softmax fonksiyonunun girdisinde, yasaklı bağlantılara karşılık gelen kısımlar maskelenmiştir.

2.4.3 Pozisyonlu ileri beslemeli ağlar

İlgi alt-katmanlarına ek olarak, kodlayıcı ve kodçözücüdeki katmanların her biri bir tam bağlantılı ileri beslemeli yapay sinir ağı bulundurur ve bu ağ her pozisyona ayrı ayrı ve tamamen aynı şekilde uygulanır. Ağ, aralarında bir tane ReLU aktivasyonu olan iki doğrusal dönüşümden oluşur.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.3}$$

Doğrusal dönüşümler, farklı pozisyonlarda aynı olup katmandan katmana farklı parametreler kullanırlar. Diğer bir deyişle; çekirdek boyutu 1 olan 2 adet evrişim gibidir. Girdi ve çıktının boyutu $d_{model} = 512$, iç katmanın boyutu ise $d_{ff} = 2048$'dir.

2.4.4 Kelime gömme ve Softmax

Dönüştürücü'de, diğer dizi iletim modellerine benzer olarak, girdi ve çıktı türcelerini d_{model} boyutundaki vektörlere çevirmek için öğrenilmiş kelime gömmeleri kullanılmıştır. Ayrıca, öğrenilmiş doğrusal dönüşüm ve softmax fonksiyonları da kodçözücü çıktısını sıradaki türce olasılığı tahminine çevirmek için kullanılmıştır. Dönüştürücü'de de, [38]'teki gibi iki gömme katmanı ile pre-softmax doğrusal dönüşümü aynı ağırlık matrisini paylaşmaktadır. Gömme katmanlarında bu ağırlıklar $\sqrt{d_{model}}$ ile çarpılır.

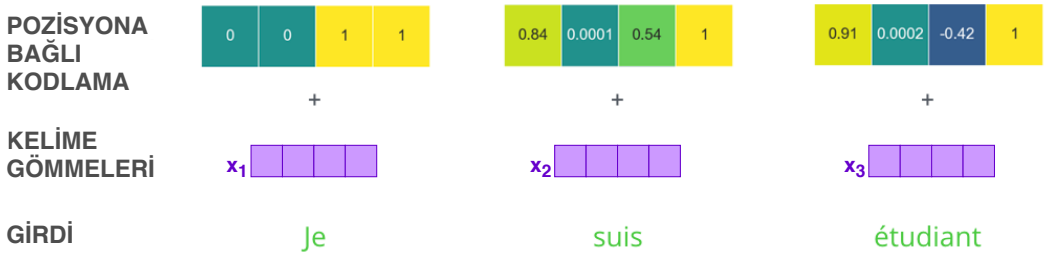
2.4.5 Pozisyona bağlı kodlama

Dönüştürücü hiçbir özyineleme ve evrişim içermediğinden, modelin dizideki sıralamaları kullanabilmesi için türcelerin dizideki göreli veya mutlak pozisyon bilgisinin aktarılması gerekmektedir. Bunun için, Dönüştürücü her girdi kelime gömmesine bir vektör ekler. Bu vektörler modelin öğrendiği belli bir örüntüyü takip eder, bu da modelin her kelimenin pozisyonunu veya dizide farklı kelimeler arasındaki uzaklığı bulmasına yardımcı olur. Bu değerleri gömme vektörlerine eklemek, Q, K, V vektörlerine izdüşümü alındığında ve iç çarpım ilgisi esnasında gömme vektörleri arasında anlamlı uzaklıklar oluşmasını sağlar. Şekil 2.24'te kelime gömmelerinin boyutunun 4 olduğu bir örnek gösterilmiştir.

2.4.6 Dönüştürücü'nün çalışma şekli

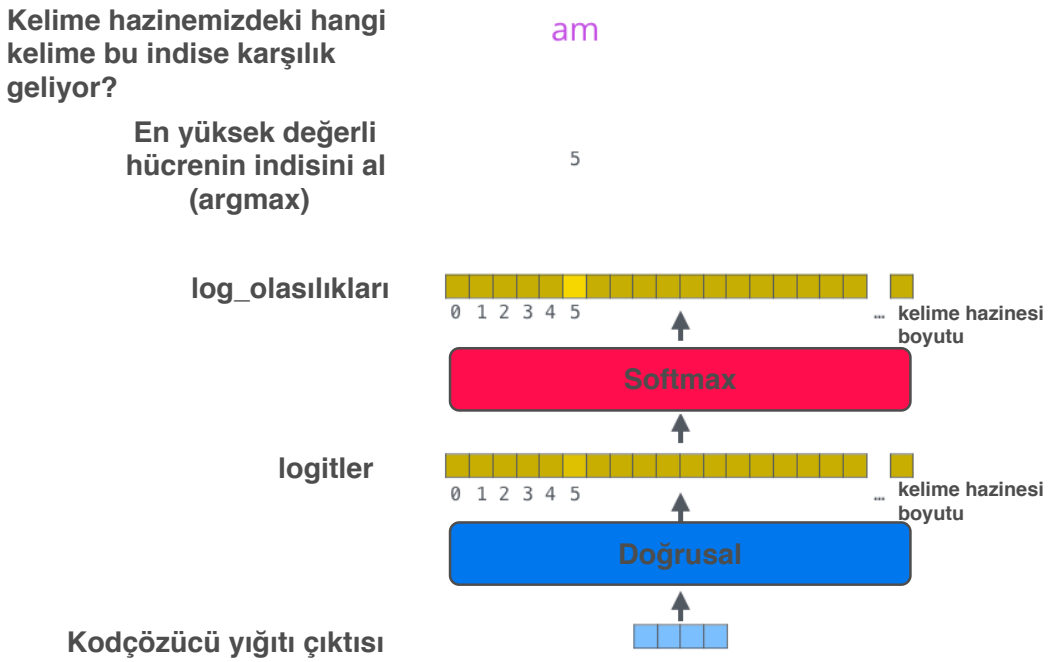
Kodlayıcı girdi dizisini işledikten sonra en üstteki kodlayıcının çıktısı K ve V ilgi vektörlerine dönüştürülür. Bu vektörler her kodçözücünün, doğru pozisyonlara odaklan-

masını sağlayan "Kodlayıcı-Kodçözücü İlgi" katmanında kullanılacaktır. Kodçözücü her adımda çıktı dizisinin bir elemanını üretir. Yukarıda da anlatıldığı gibi kodçözücüdeki özilgi katmanları, kodlayıcıdakinden biraz daha farklı çalışır. Şekil 2.26'te Dönüştürücü'nün çalışmasında kodçözücünün 3. adımı gösterilmiştir.



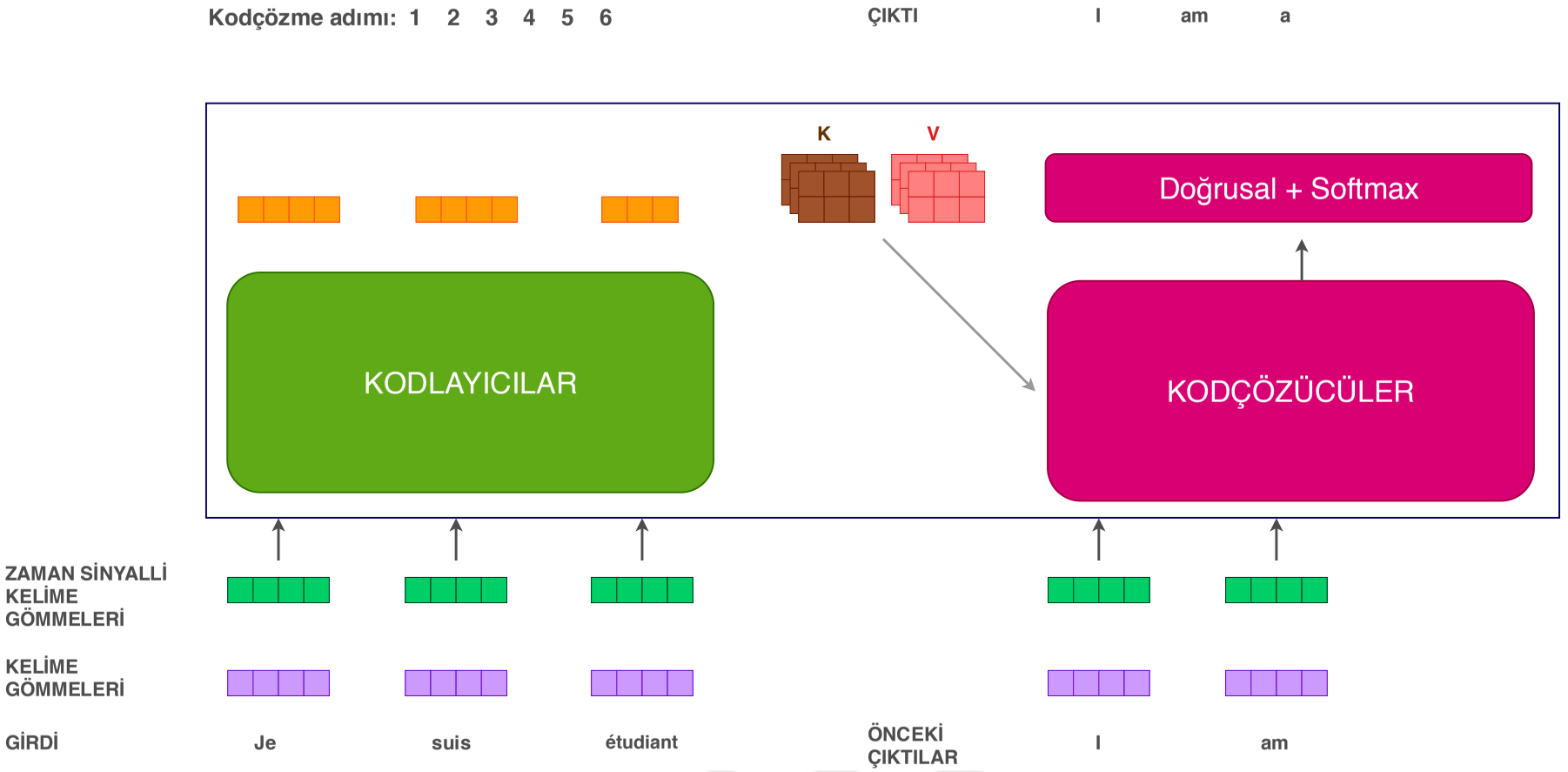
Şekil 2.24: Pozisyona Bağlı Kodlama Örneği.

Kodçözücü yığıtı çıktı olarak ondalıklı sayılardan oluşan bir vektör verir. Bu vektörü bir kelimeye dönüştürmek en sondaki Doğrusal ve Softmax katmanının işidir.



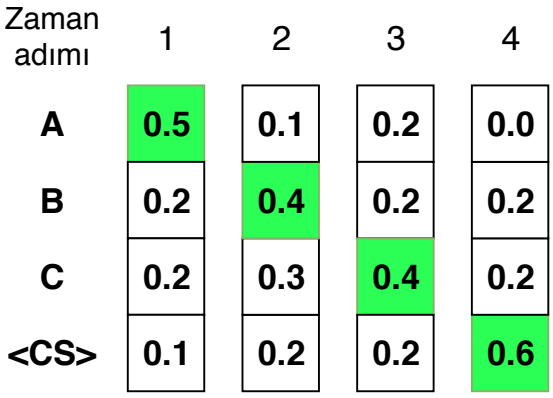
Şekil 2.25: Dönüştürücü'nün çalışmasında Doğrusal ve Softmax katmanları gösterilmiştir.

Doğrusal katman, basit bir tam bağlantılı YSA'dır. Bu katmanın çıktısı, modelin bildiği tüm kelimeler için ("modelin kelime hazinesi") bir skor vektörüdür. Softmax katmanı bu skorları, toplamları 1 olacak şekilde olasılıklara dönüştürür.



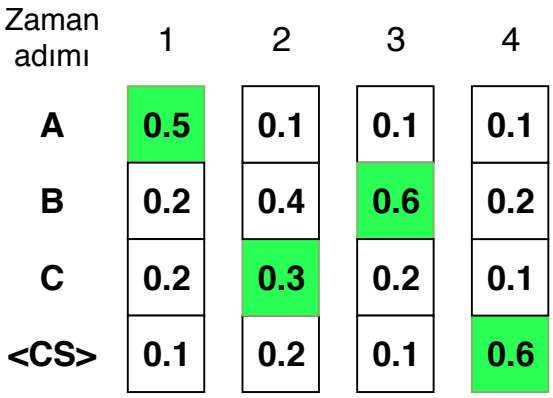
Şekil 2.26: Dönüştürücü'nün Fransızcadan İngilizceye çeviri esnasında Kodçözücünün 3. adımı gösterilmiştir.

En yüksek olasılığa sahip eleman bulunur (açgözlü arama) ve karşılık geldiği kelime, kodçözücünün bu adımının çıktısı olarak verilir. Kodçözücüde çıktı üretirken açgözlü yaklaşım, suboptimal çıktı dizileri ile sonuçlanabilir.



Şekil 2.27: Açgözlü arama örneği

Diyelim ki modelin kelime hazinesinde "A", "B", "C", ve "<CS>" olmak üzere dört kelime bulunuyor. Şekil 2.27'de [39] her zaman adımının altındaki dört sayı bu kelimelerin o zaman adımı için koşullu olasılıklarını göstermekte. Açgözlü arama algoritması her adımda en yüksek olasılığa sahip kelimeyi seçer. Böylece "A", "B", "C", "<CS>" çıktı dizisini elde ederiz. Bu çıktı dizisinin koşullu olasılığı ise $0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$'dir.

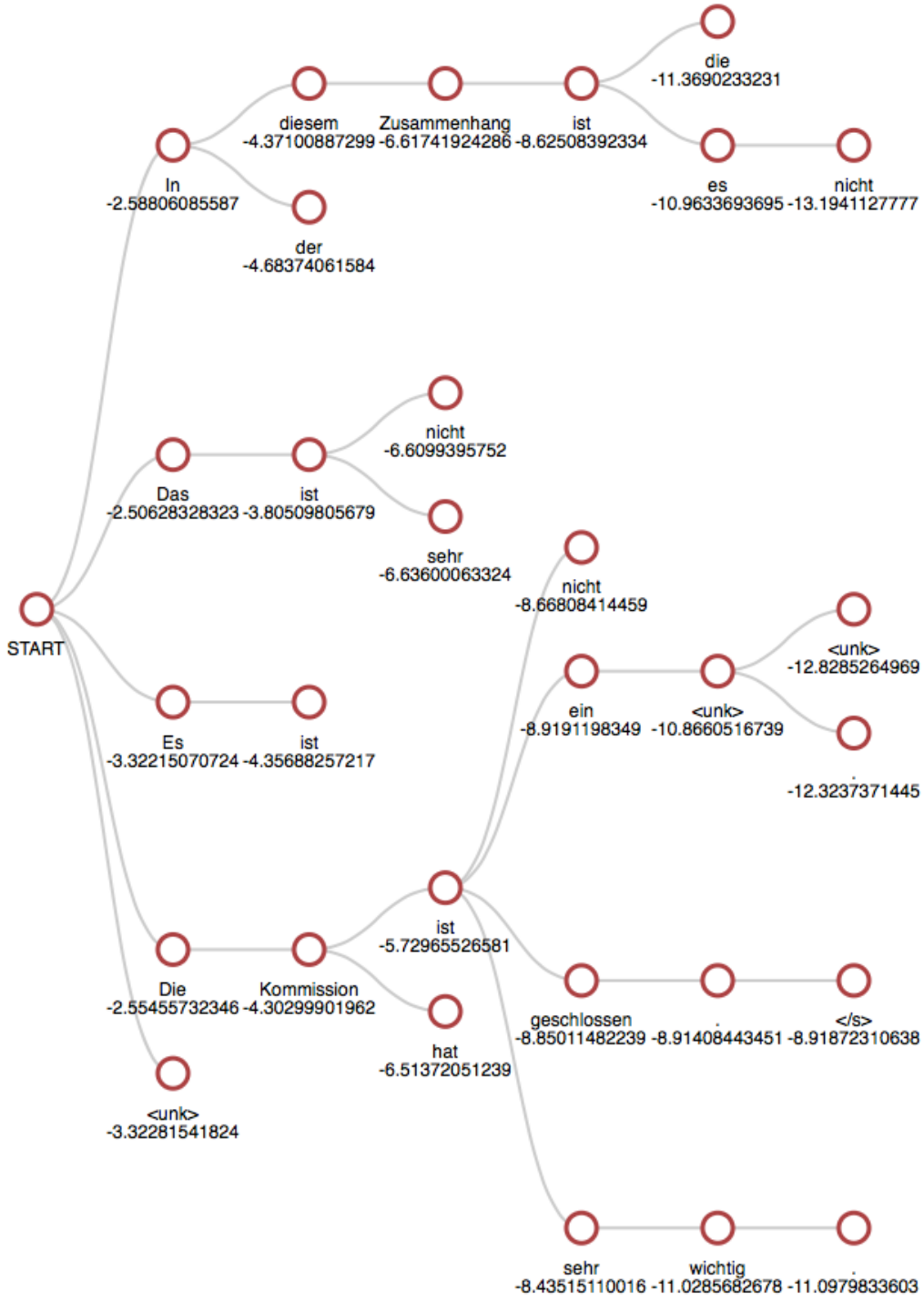


Şekil 2.28: Işın araması örneği

Eğer algoritma ikinci adımda "B" yerine "C"yi seçseydi, üçüncü adımdaki koşullu olasılık değerleri de farklı olacaktı (Şekil 2.28). Bu durumda, algoritma "A", "C", "B", "<CS>" dizisini üretirse $0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$ bir öncekinden daha iyi olasılığa sahip çıktı dizisine erişmiş oluyoruz. Görüldüğü gibi açgözlü yaklaşım optimal olmayan çıktı dizileri ile sonuçlanabiliyor.
Bu sorunu çözmek için Işın Araması (bkz. Bölüm 2.5) adı verilen yöntem kullanılır.

2.5 Işın Araması

Bu yöntem, arama ağacını inşa etmek için sığ öncelikli aramayı (breadth-first search) kullanır, ama her seviyede en yüksek olasılıklı k düğümü hafızada tutar. Şekil 2.29'da



Şekil 2.29: Işın Araması [40]

$k = 5$ olan bir ışın araması örneği gösterilmiştir. Bir sonraki seviye bu k düğüm açılarak bulunur. Hala açgözlü bir algoritmadır, ancak yukarıda bahsedilen, $k = 1$ olan duruma göre daha az açgözlüdür çünkü arama uzayı daha geniştir.

Her düğümün altındaki sayı, dizinin oraya kadarki log olasılığıdır. a_1, a_2, a_3 dizisinin olasılığı, koşullu olasılık olarak hesaplanabilir:

$P(a_1, a_2, a_3) = P(a_1)P(a_2|a_1)P(a_3|a_1, a_2)$.

Bu denklemin doğal logaritması alındığında toplam haline gelir. Arama, cümle-sonu (CS, yani </s>) türcesi en yüksek olasılıklı tahmin olarak çıkınca biter.

İlk seviyede aramayı k düğümle başlattıktan sonra, en basiti modeli bu düğümleri kodçözücü girdisi alarak k defa çalıştırmaktır. Eğer çıktı dizisinin maksimum uzunluğu N ise en kötü durumda modeli $N * k$ kere çalıştırmak gerekecektir.

Daha akıllıca bir çözüm ise; bu k tane düğümü bir yığına koyup modele o şekilde vermektir. Hesaplama paralelleştirilebildiğinden çok daha hızlı olacaktır. Eğer M tane girdi dizisi çevrilecekse, en etkili yığın boyutu $k * M$ olacaktır.

3. DENEYSEL METODOLOJİ

En gelişmiş NMT modellerinden birisi olan Dönüştürücü'nün OpenNMT-py gerçeklemesi analiz edilmiştir. OpenNMT-py, açık kaynaklı (MIT) bir NMT sistemi olan OpenNMT'nin PyTorch portudur. Yapılan deneylerde, OpenNMT-py'ın önceden eğitilmiş İngilizce-Almanca çeviri modeli kullanılmıştır: Standart eğitim tercihlerini (training options) kullanan Temel Dönüştürücü (Base Transformer) modeli. Tablo 3.1'da modelin kullandığı önemli eğitim tercihleri gösterilmiştir.

Çizelge 3.1: Temel Dönüştürücü Standart Eğitim Tercihleri

Parametre	**Değeri**
-layers	6
-rnn_size	512
-word_vec_size	512
-transformer_ff	2048
-heads	8
-encoder_type	transformer
-decoder_type	transformer
-train_steps	200000
-max_generator_batches	2
-dropout	0.1
-batch_size	4096
-batch_type	tokens
-normalization	tokens
-accum_count	2
-optim	adam
-adam_beta2	0.998
-decay_method	noam
-warmup_steps	8000
-learning_rate	2
-max_grad_norm	0
-label_smoothing	0.1
-valid_steps	10000

Model, "SentencePiece" kelime ayrıştırıcısı (tokenizer) [41] ile işlenmiş WMT'17 İngilizce-Almanca doğrulama (validation) ve eğitim verileri ile eğitilmiştir. WMT, her yıl makine çevirisi alanında yapılan bir çalıştay/konferans serisidir ve her yıl bu alandaki bilimsel araştırmalarda kullanılacak veri seti sunar.

SentencePiece

SentencePiece gözetimsiz (unsupervised) bir metin kelime-ayrıştırıcısıdır (text tokenizer). Çoğunlukla YSA-tabanlı metin üretme sistemlerinde, kelime hazinesinin modelin eğitiminden önceden belli olduğu durumlarda kullanılır. Alt-kelime birimleri algoritmasının, işlenmemiş cümlelerden direkt eğitim eklentisi ile yeniden gerçeklenme- sidir. SentencePiece, bayt-çifti-kodlaması [36] ve "unigram" dil modeli [42] olmak üzere iki kelime parçalama (segmentation) algoritmasını desteklemektedir. Örneğin; test verisi cümlelerinden "28-Year-Old Chef Found Dead at San Francisco Mall" cümlesinin SentencePiece ile ayrıştırılmış hali: "_28 - Y ear - O ld _Chef _Found _Dead _at _San _Francisco _Mal l" şeklindedir.
Deneylerde yığın büyüklüğü hep 10 olarak alınmıştır, yani her seferinde 10 tane cümlenin çevirisi yapılmaktadır.
Deneyler, İngilizce-Almanca çevirisi ile, WMT'17 test verisi (newstest2017) ile, Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz ve NVIDIA GeForce GTX TITAN XP GPU üzerinde gerçekleştirilmiştir. Veri kümesi seçimi başarımı etkilemediği için deneylerde 3004 cümleden/satırdan oluşan tek bir veri kümesi kullanılmıştır. Değerlendirme aşamasında çeviri kalitesi için BLEU skorları, hız için ise çıkarsama çalışma zamanları kullanılmıştır.

4. SONUÇLAR

4.1 Farklı Yapılandırma Parametreleri ile Dönüştürücü

OpenNMT-py etrafına bir DYSA mikroservisi kurduk ve uçtan uca başarımı ölçtük. Ağdaki iletişime harcanan zamanın, modelin kendisinin çalışma zamanına kıyasla çok daha az olduğunu ve darboğaz oluşturmadığını gözlemledik. Yani, makine çevirisi yapan DYSA mikroservisi sistemleri için darboğaz modelin kendisidir. Sonuç olarak, sinirsel makine çevirisi mikroservislerini hızlandırmak için sinirsel makine çevirisi modeline odaklanmak gerekir. Dönüştürücü modelini detaylı bir şekilde analiz ettik, bunun için öncelikle OpenNMT-py'ın Dönüştürücü gerçeklemesinin, farklı yapılandırma parametreleri ile CPU'da zaman ölçümünü yaptık. 100 ve 500 satırlık girdi metinlerinde çıkarsama (inference) için sonuçları karşılaştırdık (Çizelge 4.1). Vaswani vd. [5], yapılandırma parametreleri ile BLEU skorlarındaki, yani çeviri kalitesindeki değişimi incelemiştir, ancak bildiğimiz kadarıyla bu parametrelerin başarıma etkisini inceleyen bir çalışma literatürde bulunmamaktadır.

Katman sayısı N arttıkça, çeviri zamanı beklendiği gibi doğrusala yakın bir şekilde artmaktadır. Ancak $N = 8$'den sonra keskin bir artış gözlenmektedir (bkz. Şekil 4.1), aynı davranış GPU'da da gözlenir (bkz. Çizelge 4.2). Bu davranışın sebebi, N arttıkça modelin boyutunun da artmasıdır (bkz. Çizelge 4.2). Deneylerde kullanılan bilgisayarın önbellek boyutları: L1: 32K, L2: 256K, L3: 30720K şeklindedir. Bu yüzden $N = 6$ iken model L2'ye sığmakta ancak $N = 8$ için L3 önbellek kullanılmaktadır, bu da çalışma zamanında büyük bir artışa sebep olmaktadır.

Çizelge 4.1, ilgi imleci sayısının (h) azalmasının büyük başarım kazancı ve az miktarda çeviri kalitesi kaybı ile sonuçlandığını göstermektedir (bkz. 3. satır). Ayrıca katman sayısını (N) azalttığımız zaman, BLEU skorunun azaldığını ancak yüksek miktarda başarım kazancı elde ettiğimizi gözlemledik (bkz. 2, 9 ve 10. satırlar). 2, 11 ve 13. satırlar ise büyük modellerin daha iyi çeviri kalitesine ancak daha düşük başarıma sahip olduğunu göstermektedir.

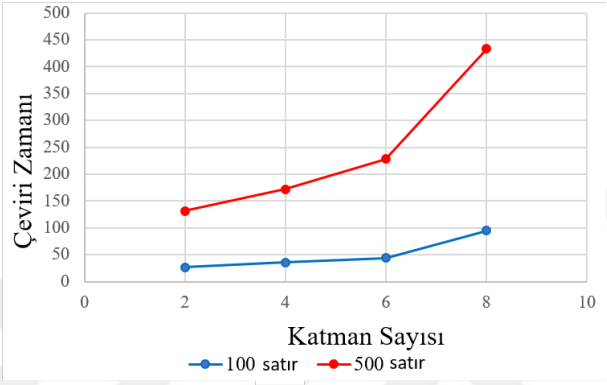
Modelin başarımı yapılandırma parametrelerindeki değişimden kolayca etkilenmektedir. Modelin davranışını daha iyi anlayabilmek için tüm modelin detaylı analizine ihtiyacımız olduğu görülür.

Çizelge 4.1: Yapılandırma parametreleri değişimiyle BLEU Skoru - çeviri zamanı

1		**N**	**d_{model}**	**d_{ff}**	**h**	**d_k**	**d_v**	**BLEU**	**100 satır**	**500 satır**
2	**temel**	6	512	2048	8	64	64	25.8	43.816 s	227.775 s
3	**İlgi imleci sayısında değişim**				1	512	512	24.9	27.371 s	146.750 s
4					4	128	128	25.5	52.202 s	272.670 s
5					16	32	32	25.8	60.981 s	4315.091 s
6					32	16	16	25.4	73.278 s	408.541 s
7	**Katman sayısında değişim**	2						23.7	26.857 s	131.478 s
8		4						25.3	35.845 s	172.184 s
9		8						25.5	94.582 s	433.037 s
10	**Model boyutunda değişim**		256			32	32	24.5	37.398 s	248.761 s
11			1024			128	128	26	96.560 s	667.856 s
12				1024				25.4	60.178 s	412.966 s
13				4096				26.2	64.702 s	564.027 s

Çizelge 4.2: Katman sayısı değişimi ile model boyutundaki değişim

N	**GPU'da 100 satırlık girdi**	**GPU'da 500 satırlık girdi**	**Parametreler** ($\times 10^6$)	**Model boyutu(KB)**
2	8.105 s	43.207 s	36	137.330
4	8.590 s	46.210 s	50	190.735
6	8.710 s	55.451 s	65	247.955
8	14.156 s	72.894 s	80	305.176



Şekil 4.1: Çeviri zamanı - Katman sayısı grafiği

Bu motivasyonla, Temel Dönüştürücü'nün CPU ve GPU'da detaylı zaman analizini yaptık.

4.2 Detaylı Dönüştürücü Başarımı

Tablo 4.3'de gösterildiği gibi, kodçözücü CPU'da toplam çeviri zamanının %60'ını alırken, GPU'da %41'ini almakta; üretici ve ışın araması ise birlikte CPU'da toplam zamanın %32'sini, GPU'da %45'ini almaktadır. Kodçözücünün GPU'da daha az oranda zaman almasının nedeni büyük matris çarpımları içermesidir. Buna karşılık, ışın araması daha önce de belirtildiği üzere sıralı bir işlemdir ve GPU'da geçirdiği zaman oranı daha büyüktür.

4.3 Işın Araması

Işın aramasının temel adımları şu şekildedir: Bir cümlenin çevirisi esnasında, ışın araması uygulanırken ilk iterasyonda sadece türce olasılıkları mevcuttur. Işın araması, en yüksek olasılığa sahip k tane türceyi seçer. İkinci iterasyondan itibaren, ışın araması yeni türce olasılıklarını o dala ait önceki toplama ekler, daha sonra aynı işlemi (en yüksek olanları seçme) bulduğu yeni olasılık değerleri ile uygulayarak devam eder.

Çizelge 4.3: Temel Dönüştürücü'nün CPU ve GPU'da zaman analizi

	Katman	**CPU**	**CPU oranları**	**GPU**	**GPU oranları**
Kodlayıcı	Çok-İmleçli İlgi	7 s	0.56%	1.8 s	0.33%
	Normalization	1.1 s	0.09%	0.2 s	0.04%
	İleri Beslemeli YSA	7.5 s	0.60%	0.8 s	0.15%
Kodçözücü	İndirgenmiş Çok-İmleçli İlgi	358 s	28.70%	87.3 s	16.22%
	Normalizasyon	23.8 s	1.91%	11.9 s	2.21%
	Çok-İmleçli İlgi	242.9 s	19.47%	63.3 s	11.76%
	Normalizasyon	22.8 s	1.83%	12.1 s	2.25%
	İleri Beslemeli YSA	102.3 s	8.20%	48 s	8.92%
Üretici	Üretici	119.3 s	9.56%	2 s	0.37%
Işın Araması	Işın Araması	276 s	22.12%	238 s	44.22%
TOPLAM		1247.6 s		538.19 s	

Çeviri kalitesini yükseltmeyi amaçlayan ışın araması, malesef üç tane başarım darboğazı oluşturur. Öncelikle, sistemin verileri teker teker işlenmesine zorlayan sıralı bir işlemdir. İkinci olarak, en sonda kullanılmayacak olan kelimelerin kodçözümü gibi fazladan birçok gereksiz işlem içerir. Son olarak ise düzensiz bir işlemdir ve bu yüzden yüksek düzey (yavaş) dillerde gerçeklenmeyi gerektirir. Düşük düzey (hızlı) dillerde gerçeklendiğinde ise yüksek miktarda mühendislik masraflarına yol açar.

Bu sebeplerle, ışın araması modelin darboğazıdır. Diğer parametreleri iyileştirmek az bir miktar veri hacmi artışı sağlayacakken; ışın aramasını $\times$ kat hızlandırmak, veri hacmini $5\times$ katına çıkaracaktır (ışın boyutu 5 iken). Bu nedenle ışın araması işlemini derinlemesine inceledik.

4.4 Işın Boyutu - Başarım ve BLEU Skoru

Işın boyutunu arttırmak çeviri zamanının daha uzun olmasına yol açar, fakat aynı zamanda çeviri kalitesini de arttırır. Bu ilişkiyi görmek için ışın boyutunu değiştirerek, çeviri zamanını ölçtük ve BLEU skorlarını hem kelime hem türce bazında hesapladık (Tablo 4.4).

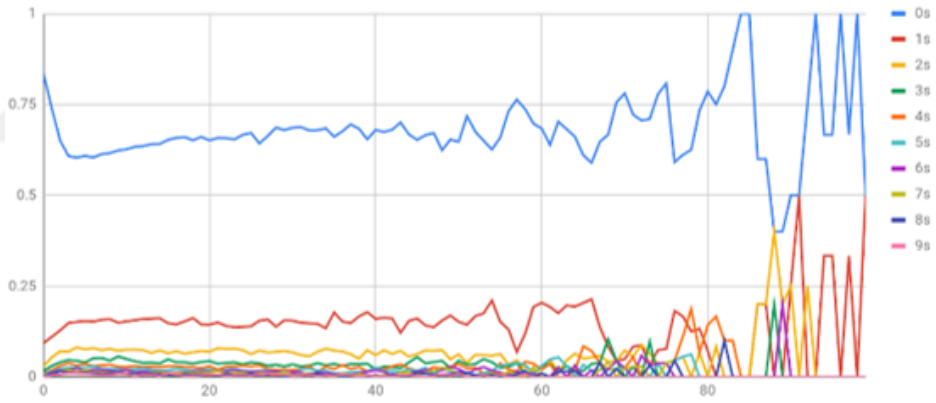
Işın boyutunun başarıma büyük oranda etkisi varken, çeviri kalitesine etkisinin düşük olduğunu gözlemledik. Buna ek olarak, ışın boyutu kelime-seviyesinde BLEU skorunu oldukça etkilerken, türce-bazında hesaplanan BLEU skoruna pek fazla etki etmemektedir. Bunun sebebi üzerine teorimiz: ışın araması sonuçları türce seviyesinde eniyileştirdiğinden (her seferinde türce olasılıklarına bakarak, en yüksek olanlarını seçiyor), daha iyi sonuç elde etmek için daha küçük ışın boyutlarına ihtiyaç duyar. Malesef çeviri kalitesi, kelime-seviyesinde BLEU ile ölçülmektedir. Yani aslında, ışın araması tüm

Çizelge 4.4: CPU'da Işın Boyutu - Çeviri Zamanı ve BLEU Skoru

Işın Boyutu	**Çeviri Zamanı**	**Kelime-bazında BLEU**	**Türce-bazında BLEU**
2	665.51 s	27.58	33.77
3	856.17 s	27.88	33.93
4	1087.96 s	27.99	33.99
5	1247.60 s	28.09	33.97
10	2092.67 s	28.07	33.69
15	2931.36 s	27.83	33.37

arama uzayını düzgün bir şekilde gezecek kadar kuvvetli değildir, bu yüzden de karar verirken eksik bilgi kullanır. Kelime odaklı yapılan ışın aramasının kelime-seviyesinde BLEU skorunun daha yüksek olabileceğini gösterdiği için bu bulgunun önemli olduğuna inanıyoruz.

Işın aramasının davranışını daha iyi anlayabilmek için en yüksek k. olasılığa sahip token'ı seçme sıklıklarını analiz ettik (Şekil 4.2).



Şekil 4.2: Işın aramasının en yüksek k. olasılığa sahip türceyi seçme sıklığı - türcenin cümledeki sırası grafiği.

Işın araması, ortalama olarak %68 sıklıkla en yüksek olasılığa sahip token'ı, %14 sıklıkla 2. en yükseğini seçiyor. Bu frekans 3. en yüksek olasılıktaki token için %6'ya düşüyor. Buna ek olarak 70-80. tokenlardan sonra frekans dağılımının düzensizliği kolaylıkla görülebilir. 90. token ve sonrasında ise ışın araması, sadece en yüksek ilk 3 arasından seçim yapıyor, yani düşük sıralamadaki token'lardan vazgeçmiş durumda. Bu deney, ışın büyüklüğünün 5 olarak seçilmesinin yeterli olduğunu, ve çıkarsama esnasında ışın büyüklüğünün değiştirilmesinin çeviri kalitesini düşürmeyeceğini göstermektedir. Tablo 4.4'teki sonuçlar bu çıkarımımızı doğrular niteliktedir.

4.5 Işın Aramasında Kelime Hazinesi Boyutu

Yukarıda da gösterildiği üzere ışın araması sinirsel makine çevirisinin darboğazıdır. Bu işlemi hızlandırma üzerine birçok araştırma yapılmıştır. Popüler metotlardan bir tanesi kelime hazinesi boyutunu azaltmaktır [17, 18]. Daha küçük bir kelime hazinesi kullanmak daha hızlı çeviri yapılmasını sağlar, ancak literatürde mevcut çalışmalar bunun asıl sebebini açıklamamaktadır.

Işın aramasının, üreticinin olasılık çıktılarını kullandığını biliyoruz. Bu olasılıkları, aktif yolların önceki birikimli toplamlarına ekliyor ve en yüksek k tanesini seçiyor. Ancak kelime hazinesinin kullanımını içeren tüm işlemler gibi, çok büyük bir listenin içinden en büyük k taneyi seçmek de uzun zaman alıyor. Bu davranışı daha iyi anlamak için küçük ve büyük kelime hazineleri kullanımında, ışın aramasının detaylı zaman analizini çıkardık. OpenNMT-py'ın Dönüştürücü'sünde ışın araması aşamasının önemli adımlarının sözde kodu ve satır-satır zaman analizi Tablo 4.5'te verilmiştir (ara basamaklar atlanmıştır).

Çizelge 4.5: 3004 satırlık girdi ile CPU'da ışın aramasının detaylı zaman dökümü

Sözdekod	*Alt-kelime-hazinesi*	*Kelime hazinesi*
Eğer cümle başı ise: ışın_skorları = türce olasılıkları	0.01 s	0.01 s
Değilse: Mevcut türce olasılıklarını, o yoldaki türcelerin olasılıkları toplamına eşit olan ışın_skorlarına ekle	4.1 s	11.3 s
Eğer <CS> (cümle sonu) türcesi gelirse, yolu burada durdur ve çocuğu olmasına izin verme	11.4 s	13.3 s
En yüksek olasılıklı k tane türceyi bul	15.8 s	184.9 s
Kelime hazinesinde bu türcelerin indislerini bul ve kaydet	13.9 s	3.4 s
Tutulan ışın araması bilgilerini güncelle	15.5 s	17.4 s
Eğer en yüksek olasılıklı k türceden biri <CS> ise bu yolu bitmiş yollara ekle	12.3 s	12.8 s
TOPLAM IŞIN ARAMASI ZAMANI	82.73 s	252.78 s
BLEU SKORU	17.42	28.09

Tüm kelime hazinesi kullanıldığında, kelime hazinesi on binlerce kelime/türce içerdiği için en çok zaman alan adım 3. adım olmaktadır (toplam 252.78 saniyelik ışın aramasının 184.9 saniyesi). MUSE (Multilingual Unsupervised or Supervised word Embeddings) [43] yardımıyla bir alt-kelime hazinesi oluşturduk. MUSE modeli bir sözlük oluşturmak amacıyla, iki dilin vektör uzaylarını hizalar. Girdi olarak verilen cümledeki her kelime veya türce için, eğitilmiş MUSE modeli, hedef dilin (bu çalışma için Almanca) hizalanmış

vektör uzayındaki en yakın *n* kelimeyi veya türceyi bulur ve bunları içeren bir alt-kelime hazinesi oluşturulur. Bu şekilde, kelime hazinesinin boyutu yüz veya daha az elemana düşürülmüş olur. Bu çözüm ile 3. adım, toplam 82.73 saniyelik ışın araması aşamasının sadece 15.8 saniyelik kısmını alır. Alt-kelime hazinesi kullanarak, ışın aramasını 3 kat hızlı gerçekleştirebilmemize rağmen, çeviri kalitesi olumsuz etkilenmiştir.

5. DEĞERLENDİRME

Bu tez çalışması kapsamında, öncelikle Vaswani vd.'nin önerdiği YMÇ modeli Dönüştürücü'nün etrafına bir YSA mikroservisi kurulmuştur. Bu mikroservisin darboğazları belirlenmiş ve küçük girdi boyutuna bağlı olarak ağ haberleşmesinin, görüntü işleme yapan mikroservislerin aksine, darboğaz olmadığı görülmüştür. Bu durumda, bu YSA mikroservisinin hızlandırılması için modelin kendisinin hızlandırılması gerektiği çıkarılmıştır.

Modelin başarımını etkileyen faktörleri tespit edebilmek için Dönüştürücü farklı yapılandırma parametreleri ile çalıştırılmış ve modelin başarımının bu yapılandırma parametrelerinin değişimine oldukça hassas olduğu gözlenmiştir. Daha sonra, davranışını daha iyi anlayabilmek adına Dönüştürücü'nün detaylı zaman analizi yapılmıştır. Tek başına büyük bir darboğaz bulunmamakla birlikte çeviri adımlarından biri olan ışın araması aşamasının büyük bir verimsizlik kaynağı olduğu gözlemlenmiştir. Bunun üzerine ışın aramasındaki her bir adımın zaman gereksinimleri detaylı olarak analiz edilmiştir.

Ayrıca çeviri kalitesi metriği olan BLEU Skoru'nun, türce bazında hesaplandığında, ışın boyutuna duyarsız olduğu gösterilmiştir. Son olarak, üretici ve ışın aramasının birlikte toplam çalışma zamanının CPU'da %32'sini, GPU'da %45'ini aldığı gözlemlendi. Bu aşamaların başarımı, kelime hazinesinin boyutuna doğrudan bağlı olduğu için daha küçük bir kelime hazinesi kullanıldığında model hızlanacaktır. Işın araması işlemi, MUSE yardımıyla alt-kelime-hazinesi oluşturularak 3 kat hızlandırılmıştır, ancak alt-kelime-hazinesinin seçimi çeviri kalitesini etkilemektedir.

Sonuç olarak, Dönüştürücü doğal dil işleme alanında bir dönüm noktası olmuş ve dizi modellerinde geniş bir yer edinmiştir. Tez kapsamında Dönüştürücü-tabanlı uygulamaların eniyilenmesi için önerilerde bulunulmuştur. Dönüştürücü'ler, hızlandırıcılar üzerinde yüksek başarım göstermekte, ışın araması aşaması birçok yönden verimsizliğe neden olmakta, ve üretici de eniyileme çalışmaları için üzerine yoğunlaşılabilecek aşamalardan birisi olarak karşımıza çıkmaktadır.

5.1 Gelecek Çalışmalar

Tez kapsamında başarımı detaylı olarak incelenen Dönüştürücü'yü hem hız hem de çeviri kalitesi bakımından iyileştirmek için önerilen alt-kelime-hazinesi yöntemi geliştirilebilir. Mevcut haliyle ışın araması hızını 3 kata çıkaran yöntem, çeviri kalitesini düşürmektedir. Bunun için alt-kelime-hazinesinin seçimi için daha farklı yöntemler uygulanabilir. MUSE modelinin eğitiminde kullanılan veri seti genişletilebilir veya modelin daha iyi eğitilmesi sağlanabilir. Bu şekilde, elde edilen hizalanmış kelime vektörleri birebir çeviriye daha yakın sonuçlar verecek ve elde edilen alt-kelime-hazinesi yüksek çeviri kalitesi için yeterli olacaktır.

Ayrıca ışın aramasının türceler üzerinden ilerleyip çeviri kalitesinin kelimeler üzerinden ölçülmesi ve buna ek olarak BLEU Skoru'nun türce bazında incelendiğinde ışın boyutuna duyarsız olması, ışın araması yönteminde ve BLEU Skoru ölçümünde yapılabilecek iyileştirmeler için yol göstermektedir.

KAYNAKLAR

[1] **Hazelwood**, **K. M.** et al. (2018). Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In: *Proceedings of the 24th IEEE Symposium on High-Performance Computer Architecture (HPCA)*, pp. 620–629.

[2] **Hoff**, **T.** (n.d.). Lessons Learned From Scaling Uber To 2000 Engineers, 1000 Services, And 8000 Git Repositories. (Date last accessed 16-Aug-2019).

[3] **Fowers**, **J.** et al. (2018). A Configurable Cloud-Scale DNN Processor for Real-Time AI. In: *Proceedings of the 45th International Symposium on Computer Architecture (ISCA)*, pp. 1–14.

[4] **Jouppi**, **N. P.** et al. (2017). In-Datacenter Performance Analysis of a Tensor Processing Unit. In: *Proceedings of the 44th International Symposium on Computer Architecture (ISCA)*, pp. 1–12.

[5] **Vaswani**, **A.** et al. (2017). Attention is All you Need. In: *Proceedings of the Thirty-first Conference on Neural Information Processing Systems (NIPS)*, pp. 5998–6008.

[6] **Klein**, **G.** et al. (2017). OpenNMT: Open-Source Toolkit for Neural Machine Translation. In: *ACL (System Demonstrations)*, pp. 67–72.

[7] **Junczys-Dowmunt**, **M.**, **Dwojak**, **T.**, and **Hoang**, **H.** (2016). Is Neural Machine Translation Ready for Deployment? A Case Study on 30 Translation Directions. In: *CoRR* abs/1610.01108.

[8] **Niehues**, **J.** et al. (2017). Analyzing Neural MT Search and Model Performance. In: *First Workshop on Neural Machine Translation*, pp. 11–17.

[9] **Lakew**, **S. M.**, **Cettolo**, **M.**, and **Federico**, **M.** (2018). A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation. In: *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 641–652.

[10] **Kaiser**, **L.** et al. (2018). Fast Decoding in Sequence Models Using Discrete Latent Variables. In: *Proceedings of the Thirty- fth International Conference on Machine Learning (ICML)*, pp. 2395–2404.

[11] **Shazeer**, **N.** and **Stern**, **M.** (2018). Adafactor: Adaptive Learning Rates with Sublinear Memory Cost. In: *Proceedings of the Thirty- fth International Conference on Machine Learning (ICML)*, pp. 4603–4611.

[12] **Chen**, **M. X.** et al. (2018). The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics,* pp. 76–86.

[13] **Koehn**, **P.** and **Knowles**, **R.** (2017). Six Challenges for Neural Machine Translation. In: *First Workshop on Neural Machine Translation*, pp. 28–39.

[14] **Yang**, **Y.**, **Huang**, **L.**, and **Ma**, **M.** (2018). Breaking the Beam Search Curse: A Study of (Re-)Scoring Methods and Stopping Criteria for Neural Machine Translation. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3054–3059.

[15] **Huang**, **L.**, **Zhao**, **K.**, and **Ma**, **M.** (2017). When to Finish? Optimal Beam Search for Neural Text Generation (modulo beam size). In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2134–2139.

[16] **Freitag**, **M.** and **Al-Onaizan**, **Y.** (2017). Beam Search Strategies for Neural Machine Translation. In: *First Workshop on Neural Machine Translation*, pp. 56–60.

[17] **Shi**, **X.** and **Knight**, **K.** (2017). Speeding Up Neural Machine Translation Decoding by Shrinking Run-time Vocabulary. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 574–579.

[18] **Senellart**, **J.** et al. (2018). OpenNMT System Description for WNMT 2018: 800 words/sec on a single-core CPU. In: *Second Workshop on Neural Machine Translation*, pp. 122–128.

[19] **Weaver**, **W.** (1955). Translation. In: *Machine translation of languages* 14, pp. 15–23.

[20] **José Bernardo Mariño Acebal**, **J.** et al. (Dec. 2006). N-gram-based Machine Translation. In: *TC-STAR* 32.

[21] **www.statmt.org** (2019). MT Research Survey Wiki.

[22] **Papineni**, **K.** et al. (Oct. 2002). BLEU: a Method for Automatic Evaluation of Machine Translation. In:

[23] **McCulloch**, **W. S.** and **Pitts**, **W.** (Dec. 1943). A logical calculus of the ideas immanent in nervous activity. In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

[24] **Pitts**, **W.** (Sept. 1942). Some observations on the simple neuron circuit. In: *The bulletin of mathematical biophysics* 4.3, pp. 121–129.

[25] **Fukushima**, **K.** (Apr. 1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. In: *Biological Cybernetics* 36.4, pp. 193–202.

[26] **Waibel**, **A.** et al. (Mar. 1989). Phoneme recognition using time-delay neural networks. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3, pp. 328–339.

[27] **LeCun**, **Y.** et al. (Dec. 1989). Backpropagation Applied to Handwritten Zip Code Recognition. In: *Neural Computation* 1.4, pp. 541–551.

[28] **SuperDataScienceTeam** (2018). Convolutional Neural Networks (CNN): Summary.

[29] **Rumelhart**, **D. E.**, **Hinton**, **G. E.**, and **Williams**, **R. J.** (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. In: ed. by **Rumelhart**, **D. E.**, **McClelland**, **J. L.**, and **PDP Research Group**, **C.** Cambridge, MA, USA: MIT Press. Chap. Learning Internal Representations by Error Propagation, pp. 318–362.

[30] **Olah**, **C.** (Aug. 2015). Understanding LSTM Networks, `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, Alındığı tarih: 22.12.2019.

[31] **Hochreiter**, **S.** and **Schmidhuber**, **J.** (Nov. 1997). Long Short-Term Memory. In: *Neural Comput.* 9.8, pp. 1735–1780.

[32] **Giacaglia**, **G.** (2019). How Transformers Work.

[33] **Sutskever**, **I.**, **Vinyals**, **O.**, and **Le**, **Q.** (Sept. 2014). Sequence to Sequence Learning with Neural Networks. In: *Advances in Neural Information Processing Systems* 4.

[34] **Alammar**, **J.** (2018). `http://jalammar.github.io/illustrated-transformer/`, Alındığı tarih: 22.12.2019.

[35] **Bahdanau**, **D.**, **Cho**, **K.**, and **Bengio**, **Y.** (2014). Neural Machine Translation by Jointly Learning to Align and Translate. In: *CoRR* abs/1409.0473.

[36] **Sennrich**, **R.**, **Haddow**, **B.**, and **Birch**, **A.** (2016). Neural Machine Translation of Rare Words with Subword Units. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

[37] **Dyer**, **C.** et al. (June 2016). Recurrent Neural Network Grammars. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 199–209.

[38] **Press**, **O.** and **Wolf**, **L.** (2016). Using the Output Embedding to Improve Language Models. arXiv: `1608.05859 [cs.CL]`.

[39] **Zhang**, **A.** et al. (2020). `https://d2l.ai`, Alındığı tarih: 22.12.2019.

[40] **group**, **H. N.** and **SYSTRAN** (n.d.). `http://opennmt.net/OpenNMT/translation/beam_search/`, Alındığı tarih: 22.12.2019.

[41] **Kudo**, **T.** and **Richardson**, **J.** (Nov. 2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, pp. 66–71.

[42] **Kudo**, **T.** (2018). Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 66–75.

[43] **Lample**, **G.** et al. (2018). Word translation without parallel data. In: *ICLR (Poster)*.

ÖZGEÇMİŞ

Ad-Soyad : Simla Burcu Harma

Uyruğu : T.C.

Doğum Tarihi ve Yeri : 25.06.1995, Mersin

E-posta : simlaharma@gmail.com

ÖĞRENİM DURUMU:

- **Yüksek Lisans** : 2018, TOBB ETÜ, Bilgisayar Müh.
- **Lisans** : 2014, TOBB ETÜ, Matematik (Çift Anadal) (4,00/4,00)
- **Lisans** : 2013, TOBB ETÜ, Bilgisayar Müh. (4,00/4,00)

MESLEKİ DENEYİM VE ÖDÜLLER:

Yıl	**Yer**	**Görev**
2019 - 2020	EPFL	EDIC Doktora Bursu, Doktora Öğrencisi
2018 - Halen	TÜBİTAK	Yüksek Lisans Bursu
2018 - Halen	TOBB ETÜ	Özel Başarı Burslu Yüksek Lisans Öğrencisi
Haz 2018 - Ağu 2018	EPFL, PARSA Lab.	Stajyer
May 2017 - Ağu 2017	DAI-Labor	Stajyer
Eyl 2016 - Ara 2016	TOBB ETÜ, TCS Lab.	Stajyer
Oca 2016 - Mar 2016	Havelsan Teknoloji Radar	Stajyer

YABANCI DİL: İngilizce (İyi), İspanyolca (Başlangıç)

TEZDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- **Harma, S.,**, Drumond, M., Falsafi, B., & Ergin, O. (2020, Ocak). An in-depth Study of Neural Machine Translation Performance, *HiPEAC Workshop on Accelerated Machine Learning (AccML)*, (In press)
- **Harma, S.,**, Drumond, M., Falsafi, B., & Ergin, O. (2019, Eylül). DNN Mikroservisleri ile Makine Çevirisi Modelleri için Performans Analizi, *İşlemci Tasarım Çalıştayı* (Poster sunumu)