


LSTM Ağları ile Türkçe Kök Bulma

Araştırma Makalesi/Research Article

 Burcu CAN

Bilgisayar Mühendisliği Bölümü, Hacettepe Üniversitesi, Ankara, Türkiye

burcucan@cs.hacettepe.edu.tr

(Geliş/Received:20.11.2018; Kabul/Accepted:22.05.2019)

DOI: 10.17671/gazibtd.486042

Özet— Türkçe, morfem adı verilen birimlerin art arda eklenmesiyle sözcüklerin oluşturulduğu sondan eklemeli bir dildir. Sözcüklerin farklı parçaların birleştirilmesiyle oluşturulması makine tercümesi, duygu analizi ve bilgi çıkarımı gibi birçok doğal dil işleme uygulamasında seyreklik probleminde yol açmaktadır çünkü sözcüğün her farklı formu farklı bir sözcük gibi algılanmaktadır. Bu makalede, sözcüklerin yapım ve çekim eklerinden arındırılarak köklerinin otomatik olarak bulunabilmesi için bir yöntem öneriyoruz. Kullandığımız yöntem tekrarlayan sinir ağları kullanarak oluşturulan kodlayıcı-kod çözücü yaklaşımına dayanmaktadır. Verilen herhangi bir sözcük, oluşturduğumuz sinir ağı yapısı ile öncelikle kodlanmakta, ardından kodu çözülerek köküne ulaşılabilir. Bu yöntem şimdiye kadar etiketleme veya makine tercümesi gibi problemlerde kullanılmıştır. Diğer Türkçe kök bulma modelleriyle karşılaştırıldığında sonuçların oldukça iyi olduğu gözlenmiştir. Diğer modellerde olduğu gibi, herhangi bir kural kümesi elle tanımlanmadan, sadece sözcük ve kök ikililerinden oluşan bir eğitim veri kümesi kullanılarak kök bulma işlemi önerdiğimiz bu model ile gerçekleştirilebilmektedir.

Anahtar Kelimeler— doğal dil işleme, hesaplamalı dilbilimi, gözetimsiz öğrenme, morfoloji, morfolojik bölümlenme, derin öğrenme, tekrarlayan sinir ağları

Stemming Turkish Words with LSTM Networks

Abstract— Turkish is an agglutinative language that builds words by concatenating the units called morphemes. Building words by concatenating various units together leads to sparsity problem in many natural language processing tasks such as machine translation, sentiment analysis, and information extraction because each different form of the same word is considered as a different word token. In this paper, we propose a method that can find the stems of words automatically by filtering out any derivational or inflectional suffixes attached to words. The proposed method is based on an encoder-decoder model built by recurrent neural networks. Any given word is first encoded by the neural network and then its stem is extracted by decoding it. This method has been used in problems such as tagging or machine translation so far. We obtain comparative results compared to other Turkish stemmers. Moreover, unlike the other models, stemming could be performed without defining a rule set manually, and by just using a train set that involves word and stem pairs.

Keywords— natural language processing, computational linguistics, unsupervised learning, morphology, morphological segmentation, deep learning, recurrent neural networks

1. GİRİŞ (INTRODUCTION)

Sondan eklemeli bir dil olan Türkçede sözcükler morfem adı verilen anlamlı parçaların art arda eklenmesiyle oluşturulur. Morfemler, sözcüklerin anlamlı en küçük parçalarıdır. Örneğin, *kitaplarımda* sözcüğü, *kitap*, *lar*, *ım*, *da* morfemlerinden oluşmaktadır. Her bir morfemin, sözcüğe kattığı bir anlam bulunmaktadır. Örneğin, *-lar* çoğul anlamı verirken, *-ım* birinci tekil kişiye ait iyelik

anlamını vermekte, *-da* ise o nesnenin bir yerde bulunduğunu işaret etmektedir. Dolayısıyla Türkçe gibi sondan eklemeli dillerde anlam ve sözdizimi çoğunlukla sözcükler ve sözcüklere eklenen morfemler sayesinde oluşturulur. Morfem tabanlı olmayan veya morfolojik bakımdan zengin olmayan diğer dillerde ise anlam ve sözdizimi morfemlerden ziyade sözcükler ile oluşturulmaktadır. Örneğin, *kitaplarımda* sözcüğü İngilizcede *in my books* olarak ifade edilmektedir.

Dolayısıyla hem iyelik eki, hem de bulunma hali için iki farklı sözcük (*my* ve *in*) kullanılmaktadır. Aynı örnekte –s sözcüğe çoğul anlamı vermektedir. Dolayısıyla İngilizcenin Türkçeye göre morfolojik olarak daha zayıf olduğu söylenebilir.

Doğal dil işleme uygulamaları metin üzerinde bazı işlemleri gerçekleştirirken genelde sözcüklere sadık kalırlar. Örneğin, *kitap* sözcüğü bir metinde, *kitapta*, *kitaplarda*, *kitabıyla*, *kitaptanmış*, *kitaplıklarda* gibi farklı formlarda bulunabilmektedir. Herhangi bir makine öğrenme yöntemi ile bu metin üzerinde herhangi bir işaretleme, sözcük anlamının ortaya çıkarılması (gibi) bu formların her biri farklı bir özelliğe (feature) denk gelecek ve sistem tarafından ayrı birer sözcük gibi algılanacaktır. Halbuki her bir form aslında tek bir anlama işaret etmektedir ve burada temel olan *kitap* sözcüğünün anlamıyken, çekim eklerinin daha çok sözdizimsel bir görevi olmaktadır. Anlamı sözdizimsel bağlamda (nesnenin bir konumda bulunması, nesnenin bir şeye ait olması, nesnenin bir şeye işaret etmesi vs.) değişse de sözcüğün anlamında radikal bir değişiklik olmamaktadır. Sözdizimini soyutlayan ama daha çok anlamı ilgilendiren doğal dil işleme uygulamalarında sözcüklerin formlarından ziyade anlamı önem taşımaktadır. Örneğin bir duygu analizi uygulamasında *kötü* sözcüğünün farklı formlarından ziyade (*kötüye*, *kötülük*, *kötüdür*, *kötünün* vs.) bu sözcüğün o metinde geçiş geçmemesi önemli bir ipucudur. Sözdizimini ilgilendiren uygulamalarda ise sözcüklerin formlarının yanında formlardaki morfemlerin ayrı ayrı varlığı da önem taşımaktadır. Örneğin bir soru cevap sisteminde, sorulan nesnenin bulunabilmesi için morfemler, nesnenin ve öznenin ayırt edilebilmesi için önemli bir ipucu taşımaktadır.

Biz bu çalışmada, daha çok sözcük anlamının önemli olduğu uygulamalar için bir Türkçe kök bulma yöntemi öneriyoruz. Çalışmada, çekim eklerinin yanında yapım eklerinin de bulunması hedeflenmektedir. Ancak önerilen yöntem tamamen kullanılan eğitim kümesine göre öğrendiği için, kullanılan veri kümesinde sadece sözcük ve sözcüğün çekim eklerinden arındırılmış hali var ise, model buna göre eğitilmiş olacak ve sadece çekim eklerini bulabilecektir. Eğitim kümesinde, yapım eklerinin de atılmış hallerini dahil etmemizin nedeni, sözcüklerin yapım eki almış dahi olsa sahip oldukları anlamdan tamamen uzaklaşmamış olmalarıdır. Örneğin, *kitap* ve *kitaplık* sözcükleri anlamsal olarak farklı olmasına rağmen ilişkilidir. Kök bulma problemi için çoğu çalışmada da, çekim eklerinin yanında yapım ekleri de sözcükten atılarak sözcüğün en yalın hali bulunur. Sözcüğün yapım ekli haline gövde ve bu halini elde etmeye *gövdeleme* adı verilirken, yapım ekleri de atıldıktan sonraki en yakın haline kök ve bu halini elde etmeye *kök bulma* adı verilmektedir. Çoğu çalışmada bu ayırım yapılmadan, her iki probleme de *stemming* adı verilmekte ve bu işlem ya *kök bulma* ya da *gövdeleme* olarak ifade edilmektedir. Bu makalede, gövdeleme ve kök bulma ifadeleri birbiri yerine kullanılmış olup, her iki ifade için de yapım eklerinin de atıldığı varsayılmıştır.

Bu makalede, son zamanlarda doğal dil işleme de dahil olmak üzere farklı alanlarda sıkça kullanılan derin öğrenme yöntemleri kullanılmıştır. Türkçe gövdeleme üzerine şimdiye kadar yapılan çalışmalar çoğunlukla kural-tabanlıdır ve kuralların elle önceden tanımlanmasını gerektirmektedir. Derin öğrenme yöntemlerinin makine öğrenmesinde özellikleri kendi çıkarabilmesi, kuralların elle tanımlanması ihtiyacını ortadan kaldırmıştır. Biz de bu çalışmada, elle herhangi bir kural tanımlamadan, sadece sözcük ve kök çiftlerinden oluşan bir eğitim kümesi kullanarak kuralları otomatik olarak çıkarabilen bir kodlayıcı-kod çözücü (encoder-decoder) yapay sinir ağı mimarisi kullandık. Kodlayıcı ile her bir sözcük sınırlı bir özellik vektörü ile ifade edilebilecek ve bu özellik vektörü kullanılarak sözcüğe ait kökün bir dil modeli dahilinde oluşturulması sağlanacaktır. Bu yöntem şimdiye kadar en fazla makine tercümesi probleminde kullanılmıştır [1] [2] [3].

Herhangi bir kural tanımlanmadığı için önerilen model başka dillerde de kullanılabilir. Bunun için sadece sözcük ve kök çiftlerinden oluşan bir eğitim kümesi yeterli olacaktır.

2. KONUYA İLİŞKİN ÖNCEKİ ÇALIŞMALAR (RELATED WORK)

Kök bulma için şimdiye kadar uygulanmış çalışmalar üç başlık altında toplanabilir: Kural tabanlı, hibrid ve istatistiksel kök modelleri. En bilinen kural tabanlı kök bulma modelleri Lovins [4], Porter [5], Krovetz [6]'dir. Bu modeller sondan eklemeli ve kök bulma kurallarının el ile oluşturulabildiği farklı dillerde uygulanmıştır. Bunlardan en fazla kullanılan algoritma Porter algoritması olmuştur. Porter algoritması bir dizi kuralları işleterek her sözcüğün köküne ulaşmayı hedefler. Kurallar şu şekilde ifade edilir:

(koşul) S1 → S2

Verilen koşulun sağlanması durumunda ve eğer sözcük S1 ekiyle bitiyorsa S2 ile yer değiştirilecektir. Örneğin İngilizce için;

(m > 1) EMENT ->

kuralı sözcükteki karakter sayısı 1'den fazlaysa ve *ement* ekiyle bitiyorsa ek silinir. Örneğin *replacement* sözcüğü bu kural ile *replace* şeklinde değiştirilir.

Porter algoritması daha sonra resmi bir dile dönüştürülerek Snowball dili ortaya atılmıştır [7]. Yeni bir dil olarak önerilen Snowball dili gövdeleme amaçlı olarak önerilmiş olup İngilizce ve Almanca gibi farklı dillerde uygulanmıştır. Snowball, Türkçe için de uygulanmış ve Türkçedeki tüm kurallar tanımlanarak bir Snowball gövdeleme algoritması oluşturulmuştur [8]. Ancak belirtilen çalışma sadece isim gövdeleri için geliştirilmiş olup, fiiller için yeni kuralların oluşturulması gerekmektedir. Türkçede fiillere eklenebilecek ek sayısı 50'den fazla olduğundan [9] ötürü fiiller için tanımlanacak

kuralların çok daha geniş olması beklenmektedir. Tunçelli [10] fiiller için de eklerin bir kısmını ekleyerek yine Snowball ile bir kural tabanlı Türkçe kök bulma algoritması oluşturmuştur.

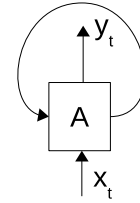
Bunun haricinde Türkçe için önerilen kök bulma algoritmalarından bir başkası Resha'dır [11]. Bu modelde Nuve [12] adlı Doğal Dil İşleme Kütüphanesi tarafından gerçekleştirilen istatistiksel dil modeli kullanılarak öncelikle bir kök sözlüğü oluşturulmuştur. Snowball ile gerçekleştirilen diğer Türkçe kök bulma algoritmalarına göre daha doğru çalıştığı iddia edilmektedir. Diğer Türkçe kök bulma algoritmaları gibi bu çalışmada da kökü bulunmak istenen sözcüğün bağlamına bakılmamaktadır. Bağlamın dikkate alınarak, sözcük türüne göre kökün bulunmasına lemmalaştırma (lemmatization) adı verilmektedir. Örneğin, Türkçede eşsesli olarak ifade edilen sözcüklerin verilen bağlama göre gövdesinin bulunması gerekmektedir. Verilen çalışmaların hiçbirinde sözcüklerin bağlamı ele alınmamaktadır. Onun yerine sözcük olarak ele alınmakta ve olası gövdelerinden biri bulunmaktadır. Biz de bu çalışmada bağlamı ele almadan bir kök bulma modeli oluşturmaya çalıştık.

Bunların dışında Türkçe için önerilen morfolojik analiz sistemleri de kök bulma modeli olarak kullanılabilir. Türkçe için önerilen morfolojik analiz sistemlerinin başında PC-Kimmo [13] adı verilen iki katmanlı dil analiz modelinin önerdiği sonlu durum kuralları ile tanımlanmış iki katmanlı sistemdir [14]. Eryiğit ve Adalı [9] tarafından önerilen model de morfolojik analiz modeli olarak önerilmiş olup benzer şekilde sonlu durumlu özdevinirler (finite state machines) tarafından morfolojik kuralların sözcük türüne göre (isim, fiil vs) farklı sınıflara ayrıştırılarak tanımlanmasıyla gerçekleştirilmiştir. Önceki çalışmalardan [14] farklı olarak bu çalışmada kökten eke doğru değil, sözcük sonundaki eklerden köke doğru eklerin tek tek bulunmasıyla kök bulma işlemi gerçekleştirilmektedir. Sonlu durumlu özdevinirler kullanan bir başka çalışma ise Çöltekin [15] tarafından benzer şekilde kurallar tanımlanarak geliştirilmiştir. Zemberek [16] adlı açık kaynak kodlu kütüphane ise morfolojik analiz, kök bulma gibi farklı araçları içermektedir ve diğer modeller gibi kural tabanlıdır.

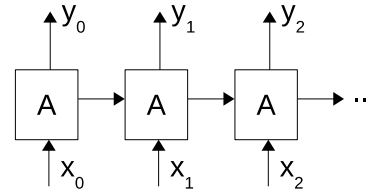
Biz bu çalışmada önceki sistemlerden farklı olarak kural tabanlı olmayan ve tamamen kendi kurallarını kendi öğrenebilen bir model geliştirmeyi amaçladık. Böylece sonlu durumlu özdevinirlerin oluşturulması, dile bağımlı olan kuralların tanımlanması gibi aşamaların olmadığı ve dolayısıyla sistemin eğitiminin daha pratik hale geldiği bir sistem geliştirdik. Daha önce Türkçe için derin öğrenme ile gerçekleştirilmiş kök bulma yöntemi önerilmediği için bu çalışma diğerlerinden yöntem olarak farklı bir yere oturmaktadır.

3. KODLAYICI-KOD ÇÖZÜCÜ MODELİ İLE KÖK BULMA (STEMMING WITH AN ENCODER-DECODER MODEL)

Klasik yapay sinir ağları, gelen her bilgiyi önceki bilgilerden bağımsız olarak işlemektedir. Diğer bir ifadeyle her gelen bilgi bir sonraki zaman biriminde unutulmaktadır. Bu da sıralı yapının önemli olduğu veri türlerinde sorun yaratmaktadır. Örneğin, bir cümlenin anlamını çıkaracak bir model geliştirilmek istendiğinde, her sözcüğün cümleye katkısı olduğundan hepsinin ayrı ayrı anlamını çıkarmak yerine bütün sözcüklerin birlikte meydana getirdiği anlamın ortak olarak oluşturulması gerekir. İnsan beynindeki ilgili mekanizmanın da genellikle bu şekilde olduğu varsayılmaktadır. Bir metni okurken karşılaştığımız her sözcüğü öncekilerle bağlantılandırarak anlamlandırmaya çalışırız ve her seferinde her sözcük için baştan yeni bir anlam yaratmaya çalışmayız.



Şekil 1. Tekrarlayan Sinir Ağı (Recurrent Neural Network)

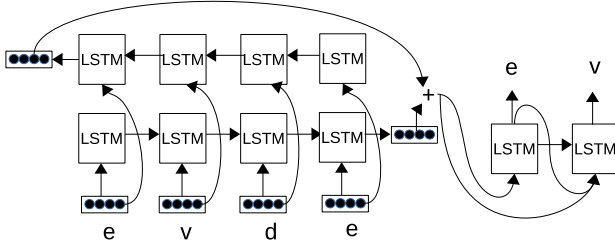


Şekil 2. Tekrarlayan sinir ağının açılmış hali (Unfolded Recurrent Neural Network)

Bu amaçla önerilen Tekrarlayan Sinir Ağları (Recurrent Neural Networks) kendi içinde sahip oldukları döngülerle hem yeni gelen bilgiyi işler, hem de önceden gelen bilgileri kendi içinde tutabilirler (bkz. Şekil 1). Aslında normal bir sinir ağı ile yapısal olarak benzerdir ancak bir yapay sinir ağının kopyaları oluşturulmuş hali gibi düşünülebilir (bkz. Şekil 2). Böylece aynı sinir ağı her yeni gelen bilgiyi tekrar tekrar işlemekte ve böylece önceki bilgileri de unutmamaktadır.

Geçmiş bilginin miktarının artması, tekrarlayan sinir ağlarında problem yaratmaktadır. Yapay sinir ağlarında kullanılan geriye yayılım (backpropagation) gibi algoritmalarda gradyanların en son zaman biriminden en önceki zaman birimine doğru yapılmasından ötürü gradyanların geçmiş bilginin artmasıyla küçülmesi ve yavaş yavaş kaybolması problemi ortaya çıkmaktadır [17]. Bu nedenle tekrarlayan sinir ağları çok fazla geriye dönük bilgi olması durumunda çalışmamaktadır. Örneğin, çok uzun cümlelerde en sondaki sözcükten en başa doğru bir geriye yayılım yapıldığında gradyanlar kaybolmakta ve uzun aralıklı bağımlılık ilişkilerinin (long-term dependency) bulunması zorlaşmaktadır.

Bahsedilen problemi ortadan kaldırmak için Uzun Kısa Süreli Bellek Ağları (Long-short Term Memory Networks - LSTM) önerilmiştir. Tekrarlayan Sinir Ağlarının bir türü olan Uzun Kısa Süreli Bellek Ağlarında özel bir hafıza hücresi kullanılarak uzun geçmişlerin de öğrenilmesi mümkün olmaktadır. Özellikle doğal dil işleme kapsamında oluşturulan dil modellerinde geçmiş bilginin unutulmaması gerektiğinden tekrarlayan sinir ağlarını kullanmak oldukça avantajlı olabilmektedir.



Şekil 3. Temel Kodlayıcı-Kod Çözücü modeline ait mimarinin gösterimi
(The illustration of the Baseline Encoder Decoder Model)

Biz bu çalışmada Uzun Kısa Süreli Bellek Ağı yapısını kullanarak bir kök bulma modeli oluşturduk. Bu modelde iki farklı Uzun Kısa Süreli Bellek Ağından oluşan ve Kodlayıcı-Çözücü (Encoder-Decoder) olarak anılan bir mimari kullandık. Bu mimari Sutskever ve arkadaşları [1] tarafından makine tercümesi için önerilen bir modeldir ve sonrasında da çoğunlukla makine tercümesi için kullanılmıştır. Bunun haricinde diğer sıradan sıraya (sequence-to-sequence) tahminleme problemlerinde de kullanılmaktadır.

Oluşturduğumuz temel mimari Şekil 3'te verilmiştir. Bu mimaride bir çift yönlü Uzun Kısa Süreli Bellek Ağı (LSTM) sözcüklerin ham hallerini kodlamak için kullanılmıştır. Çift yönlü LSTM'den oluşan yapıda, bir LSTM sözcüklerin harflerini gelen sırada işlemekte, diğer LSTM ise sözcükleri sondan başa doğru işlemektedir. Böylece her bir karakter için hem ondan sonra gelebilecek karakterler öğrenilmekte, hem de ondan önce gelen olası karakterler öğrenilerek aslında iki yönlü bir dil modeli oluşturulmaktadır ve böylece uzun aralıklı bağıllık ilişkileri de öğrenilebilmektedir. Her iki LSTM'e de verilen girdi her bir karaktere ait olan ve onun özellik vektörü (feature vector) diyebileceğimiz bir gösterim vektöründen (representation veya embedding) oluşmaktadır. Böylece her bir karakterin de aynı zamanda bir vektörü eğitim süresince oluşturulmaktadır. Sözcüğün kodlanmış hali, ileri LSTM'den ve geri LSTM'den alınan vektörlerin uç uca eklenmiş vektörüdür. Bu vektör, kod çözücüye verilmekte ve kod çözücü bu koda karşılık gelen sözcük kökünü tahmin etmeye çalışmaktadır. Kodlayıcı yapısında bazı mimari farklılıkları önererek iki farklı modeli bu çalışma kapsamında geliştirdik. Bu iki farklı modelin ayrıntıları aşağıda verilmiştir.

3.1. Temel Kodlayıcı-Kod Çözücü Modeli (Baseline Encoder-Decoder Model)

Verilen bir w sözcüğü $w = \{ \langle \text{İLK} \rangle, c_0, c_1, \dots, c_n, \langle \text{SON} \rangle \}$ olarak tanımlanır. Sözcüğün başladığını ve bittiğini ifade etmek için başlangıç sembolü olarak $\langle \text{İLK} \rangle$, bitiş sembolü olarak da $\langle \text{SON} \rangle$ kullanılmaktadır. Gösterimi basitleştirmek için Şekil 3'te başlangıç ve bitiş sembolleri gösterilmemiştir. Burada c_j olarak ifade edilen, sözcüğe ait j . karakterdir. Girdi olarak verilen her bir karakter bir vektör uzayında tanımlanmaktadır. Bu uzaya ait her bir karakter vektörü $e_j \in \mathcal{R}^{1 \times k}$ olarak tanımlanır ve k olarak ifade edilen karakter vektörünün boyutudur. Girdileri işleyecek olan ileri kodlayıcı $BiLSTM_{ileri}$ ile ifade edilmektedir. Her bir girdi karakterine ait olan vektör sırayla işlenmektedir:

$$e_{w_i}^{ileri} = BiLSTM_{ileri}(e_{1:n})$$

Burada w_i i . sözcüğü ifade, n ise sözcüğün karakter sayısını ifade etmektedir. Sonuç olarak sözcüğü ileri doğru kodlayan $e_{w_i}^{ileri}$ vektörü elde edilir. Sözcük karakterlerini sondan başa doğru işleyen geri kodlayıcı ise $BiLSTM_{geri}$ ile ifade edilmektedir. Her bir girdi karakteri sözcüğün sonundan başına doğru sırayla geri kodlayıcı tarafından işlenmektedir:

$$e_{w_i}^{geri} = BiLSTM_{geri}(e_{1:n})$$

Buradan da sonuç olarak $e_{w_i}^{geri}$ ile ifade edilen ve sözcüğü sondan başa doğru kodlayan vektör elde edilir. Sözcüğün toplam vektörü e_{w_i} bu iki vektörün uç uca eklenmesiyle oluşturulur:

$$e_{w_i} = e_{w_i}^{ileri} \oplus e_{w_i}^{geri}$$

Böylece, geliştirdiğimiz temel kodlayıcı-kod çözücü mimarisinde kodlayıcı ile kodlanan her sözcüğün ileri ve geri LSTM'lerden alınan kodlanmış vektörleri uç uca eklenerek kod çözücü LSTM'ine her zaman biriminde girdi olarak verilerek sözcüğün kökü karakter karakter oluşturulur. Bunun için $BiLSTM_{çözücü}$ tanımlanmıştır. Oluşturulacak kök $s = \{ \langle \text{İLK} \rangle, c_0, c_1, \dots, c_k, \langle \text{SON} \rangle \}$ ile ifade edilmekte olup yine aynı şekilde başlangıç ve bitiş sembolleri ile kökün başı ve sonu belirtilmektedir. Böylece kod çözücü ne zaman işleme başlayacağını ve ne zaman karakter üretmeyi sonlandıracağını öğrenebilmektedir.

Her t zaman biriminde kod çözücüye verilecek girdi, sözcüğe ait vektör e_{w_i} ile bir önceki zaman birimindeki sözcüğe ait karaktere ait çıktı vektör uzayına ait vektörün uç uca eklenmiş halidir. Burada, $f_j \in \mathcal{R}^{1 \times k}$ çıktı uzayına ait her bir çıktı karakteri için tanımlanan vektördür. Böylece kod çözücüye verilecek toplam girdi vektörü t zamanında

$$f_t = e_{w_i} \oplus f_{t-1}$$

şeklinde olur. Kod çözücü ise bu durumda k kök uzunluğu olmak üzere $BiLSTM_{\text{çözücü}}(f_{1:k})$ olarak tanımlanır. Dolayısıyla kod çözücü, hem bir önceki zamandaki karakteri, hem de sözcüğün kendisini kullanarak bir sonraki karakteri tahmin etmeye çalışır. Burada dikkat edilmesi gereken, girdi karakterlerine ait vektör uzayı ile, çıktı karakterlerine ait olan vektör uzayının birbirinden farklı olmasıdır. Biri sözcüklere ait olan bir vektör uzayı iken, diğeri gövdelere ait olan karakter vektör uzayını temsil etmektedir. Bu yüzden iki farklı karakter gösterim uzayı öğrenilmektedir. Bir başka dikkat edilmesi gereken nokta ise, eğitim ve test aşamalarında kod çözücünün farklı çalışıyor olmasıdır.

Eğitim aşamasında, kod çözücüye her zaman biriminde verilen karakter, bir önceki zaman biriminde oluşturulan karakter değil, eğitim kümesinde bir önceki zaman biriminde oluşturulması gereken karakterdir. Böylece her zaman biriminde kodlayıcıya bir önceki zamanda alınması gereken doğru karakterin çıktı uzayındaki vektörel gösterimi bir sonraki zaman birimindeki karakteri üretmek için kullanılır. Bunun nedeni, kod çözücünün ürettiği karakterlerin henüz doğru olmaması ve bir önceki zamanda ürettiği karaktere göre sonraki karakter oluşturulursa öğrenmenin gerçekleşmeyecek olmasıdır.

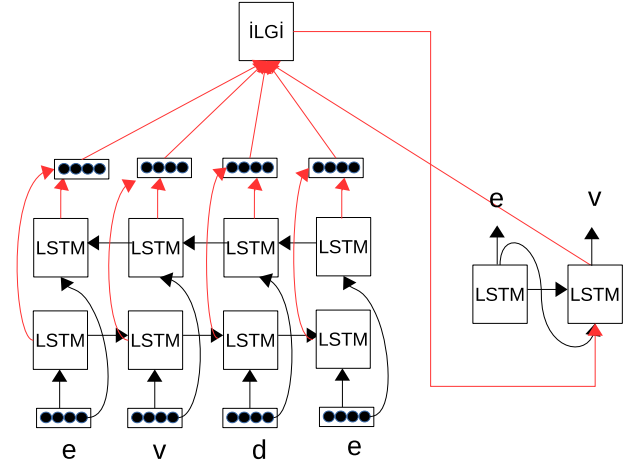
Test aşamasında ise, kod çözücüye verilen girdi, bir önceki zaman biriminde kod çözücünün oluşturduğu karakterin çıktı uzayındaki vektörel gösterimidir. Kod çözücünün artık eğitimi tamamlandığı için öğrenilen dil modeli üstünden her zaman biriminde yeni bir karakter oluşturulmakta ve bu karakter bir önceki zaman biriminde oluşturulan karaktere göre seçilmektedir.

3.2. İlgi Mekanizması Kullanan Kodlayıcı-Kod Çözücü Modeli (Encoder-Decoder Model With Attention Mechanism)

Temel modelde kullanılan sözcüğün kodlanmış hali her sözcükteki her karaktere eşit önem vermektedir. Özellikle tercüme sistemlerinde, bir dildeki bir cümleyi herhangi bir dildeki karşılığına çevirirken kaynak dildeki her sözcüğün hedef dildeki her sözcük için eşit anlam katkısı olmaz. Hedef dildeki bir sözcüğü çevirirken kaynak dilde verilen cümledeki bazı sözcükler daha önemli olmaktadır. Bu önemi öğrenmek üzere ilgi mekanizması (attention mechanism) [1] olarak adlandırılan ve genelde bir gizli katmandan oluşan bir sinir ağı tanımlanabilmektedir. Bu sinir ağı aracılığıyla cümleyi başka bir dile çevirirken cümledeki farklı yerlerine odaklanabilmektedir. Böylece her hedef sözcük hedef dilde kod çözücü tarafından oluşturulurken kaynak dildeki sözcüklerin ağırlıklı toplamı kullanılmaktadır.

Bir önceki bölümde verilen temel modelde, kod çözücü her karakteri oluştururken sözcüğün kodlanmış halini her karakter için eşit olarak kullanmaktaydı. İlgi mekanizmasını kullanarak geliştirilen yeni kodlayıcı-kod çözücü modeli Şekil 4'te verilmiştir. Buna göre 'v' karakterini oluştururken, 'e', 'v', 'd', 'e' karakterlerine ait

vektörel gösterimlerin de ağırlıklı toplamı kullanılacaktır.



Şekil 4. İlgi ağı kullanarak oluşturulan Kodlayıcı-Kod Çözücü Modeli (Encoder-Decoder Model with Attention Mechanism)

Ağırlıklar bulunurken normalize edilmesi gerekmektedir:

$$\alpha_{ts} = \frac{\exp(\text{skor}(e_s, f_t))}{\sum_{s'=1}^S \exp(\text{skor}(e_{s'}, f_t))}$$

Burada α_{ts} ile verilen kod çözücüdeki t . karakter için kodlayıcıdaki s . karakterin skorudur, yani t . karakter oluşturulurken s . karaktere ne kadar önem verileceğini belirler. $\text{skor}(e_s, f_t)$ sözcükteki s . karakterin, kökteki t . karakter üzerindeki ağırlığını yani önemini ifade eder. Örneğin, *kitabı* sözcüğünün kökü *kitap* olarak bulunurken 'p' karakterinin oluşturulması esnasında b sözcüğüne sözcükteki diğer karakterlere nispeten daha fazla ağırlık verilir ve böylece 'b' karakterinin 'p' karakterine dönüştürüleceği tespit edilir. $\text{skor}(e_s, f_t)$ normalize edilmemiş bir ağırlıktır ve sözcükteki diğer bütün karakterler için de benzer şekilde bir skor hesaplanarak bu toplama bölünerek normalize edilir. Böylece kod çözücüde her bir karakter için kodlayıcıdaki her bir karakterin vektörünün ağırlıklandırılmış toplamı esas alınır:

$$c_t = \sum_s \alpha_{ts} e_s$$

Eğitim aşamasında, kod çözücünün her bir zaman biriminde verilen girdi, kodlayıcıdan alınan ağırlıklı toplam c_t 'ye bir önceki aşamadaki kodlayıcıdan alınması gereken doğru karakterin vektörünün eklenmiş halidir:

$$f_t = c_t \oplus f_{t-1}$$

İlgi mekanizması kodlayıcıya verilen her bir karakterin kod çözücüdeki her bir karakter için ne kadar önemli olduğunu bulur. İlgili skor şu şekilde hesaplanır:

$$\text{skor}(e_s, f_t) = v \tanh([W_1 e_s]; [W_2 f_t])$$

Burada v , W_1 ve W_2 öğrenilecek ağırlık matrislerine karşılık gelmektedir. Yani aslında ilgi mekanizması çıktı fonksiyonu tanh olan tek katmanlı bir yapay sinir ağına karşılık gelmektedir.

Test aşamasında yine bir önceki modelde anlatıldığı gibi kod çözücünün bir önceki zaman biriminde ürettiği karakterin çıktığı uzayındaki vektörel gösterimi bir sonraki zaman biriminde kod çözücüye girdi olarak verilmek için kullanılır. Bunun için kod çözücünün her üretilen çıktısı softmax işleminden geçirilerek karakterler üzerinde olasılıksal bir dağılım elde edilir. En yüksek olasılığa karşılık gelen karakter seçilerek, kod çözücünün o karakteri üretmesi sağlanır.

4. DENEYLER VE SONUÇLAR (THE EXPERIMENTS AND RESULTS)

Önerilen iki farklı model Türkçe kök bulma problemi için uygulanmış ve deney sonuçları diğer Türkçe kök bulma algoritmaları ile karşılaştırılmıştır.

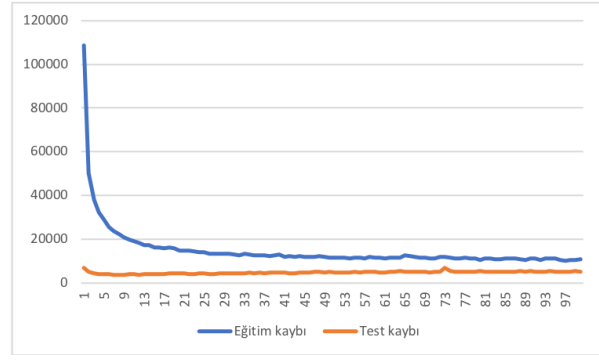
4.1. Uygulama Detayları (Implementation Details)

Önerilen modeller Dynet [18] kütüphanesi kullanılarak kodlanmıştır ve makalenin kabul edilmesi durumunda kaynak kodu paylaşılacaktır.

Önerilen mimarilerde, karakter tabanlı kök bulma problemi için en uygun vektör boyutlarını bulabilmek için farklı vektör boyutlarıyla deneyler gerçekleştirilmiştir. Girdi ve çıktı uzayındaki karakter vektörleri e_j ve f_j için $k=32, 64, 128$ boyutları denenmiştir. Hem temel modelde, hem de ilgi mekanizması içeren modelde kodlayıcı ve kod çözücü LSTM'ler, $BiLSTM_{ileri}$ ve $BiLSTM_{geri}$ için, çıktı boyutu (state size) olarak aynı şekilde $l=32, 64$ ve 128 boyutları ile farklı deneyler gerçekleştirilmiştir. İlgi mekanizması için kullanılan W_1 ve W_2 matrislerinin boyutları $32 \times (2k)$ olarak ve v vektörünün boyutu 1×32 olarak belirlenmiştir.

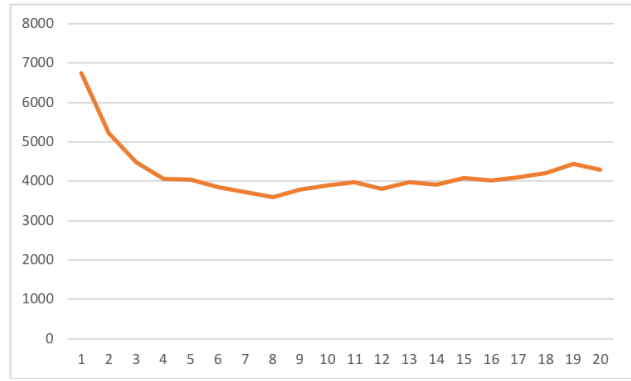
Eğitim, her iki model için de 100 epoch süresince sürdürülmüştür. Eğitim algoritması olarak Olasılıksal Dereceli Azalma (Stochastic Gradient Descent) algoritması kullanılmıştır. Kayıp fonksiyonu olarak ise (loss function) negatif log olasılığı (negative log likelihood) kullanılmıştır. Temel model için eğitim ve test kayıp değerleri Şekil 5'te verilmiştir. Grafikte de görüldüğü üzere hem eğitim, hem de test kaybı süreç içerisinde düşmeye devam etmekte ve bir süre sonra dengeye ulaşmaktadır.

Sadece test kaybı ilk 20 epoch için incelendiğinde ise test kaybının ilk 10 iterasyonda hızlıca düştüğü, sonrasında ise bir miktar yükseldiği gözlenmiştir (bkz. Şekil 6). Bu da, 10. iterasyondan sonraki aşırı uyum (overfitting) ihtimalini göstermektedir.



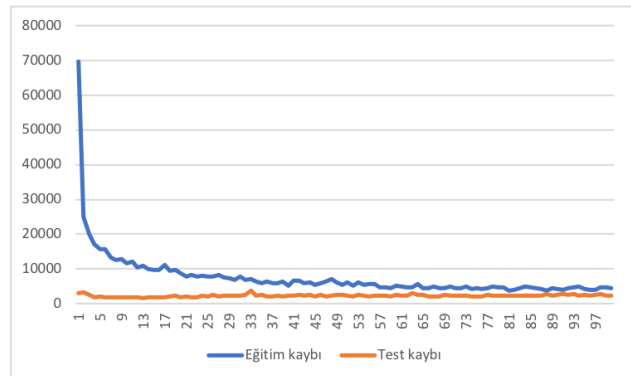
Şekil 5. Temel kodlayıcı-kod çözücü modelin eğitimi esnasında eğitim kayıp değerlerinin ve test kayıp değerlerinin değişimi

(The change of the training loss value during the training of the baseline encoder-decoder model)



Şekil 6. Temel kodlayıcı-kod çözücü modelin eğitimi esnasında test kayıp verilerinin 20 iterasyon için değişimi (The change of the test loss during the training of the baseline encoder-decoder model during the first 20 epochs.)

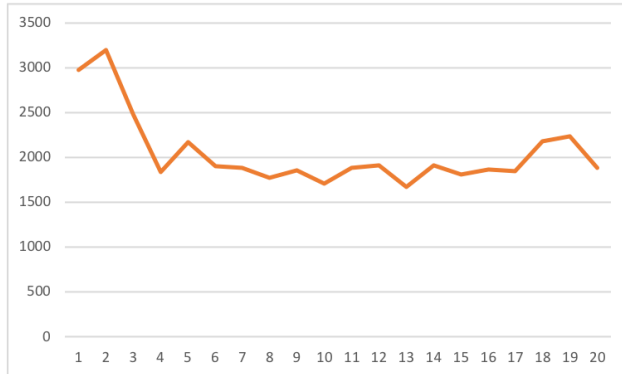
İlgi mekanizması içeren model için eğitim ve test kayıp değerleri ise Şekil 7'de verilmiştir. Diğer modele benzer şekilde, bu modelin eğitimi esnasında da eğitim ve teste ait kayıp değeri zaman içinde düşmekte ve bir süre sonra dengeye ulaşmaktadır.



Şekil 7. İlgi mekanizması içeren kodlayıcı-kod çözücü tabanlı modelin eğitimi esnasında kayıp (loss) değerinin değişimi

(The change of the loss value during the training of the encoder-decoder model with attention mechanism)

Test kaybını daha net inceleyebilmek için aynı şekilde ilk 20 iterasyondaki test kaybı değişimini gösteren grafik Şekil 8'de verilmiştir. En düşük değer 13. iterasyonda elde edildiği ve sonrasında aşırı uyuma bağlı olarak kayıp değerinin az da olsa arttığı gözlenmiştir.



Şekil 8. İlgili mekanizması içeren kodlayıcı-kod çözücü tabanlı modelin eğitimi sırasındaki test kaybı değerlerinin ilk 20 iterasyondaki değişimi
(The change of the test loss value during the training of the encoder-decoder model with the attention mechanism)

4.2. Veri Kümesi (Dataset)

Türkçe için sadece kök bilgisi içeren bir veri kümesi olmadığından ötürü sözcüklerin morfolojik analizlerini de içeren METU-Sabancı Treebank [19] [20] kullanılmıştır. Bu derlemde 4997 cümle bulunmaktadır. Toplam 45141 sözcük içermektedir. Sadece sözcük türleri (word types) dahil edildiğinde bu sayı 17025'e inmektedir. Dolayısıyla, bazı sözcükler derlemde birden fazla yer almaktadır. Bu sayının çoğu durak sözcüklerdir. Bazı sözcüklerin ise birden fazla kök analizi olabilmektedir. Örneğin, *yedi*, *işte*, *beyin* gibi sözcüklerin cümledeki sözdizimsel ve anlamsal kullanımına göre kök değişebilmektedir. Birden fazla kök analizine sahip olan sözcüklerin her bir kök analizi ayrı olarak dahil edildiğinde toplam 17243 sözcük-kök çifti elde edilmiştir. Diğer bir deyişle, 218 sözcüğün birden fazla kök analizi bulunmaktadır. 17243 sözcükten 15519'u eğitim kümesi olarak, 1724'ü ise test kümesi olarak belirlenmiştir (test kümesi \cong toplam veri kümesi/10). Kümelerin dağılımı rasgele belirlendiği için, k katlamalı çapraz doğrulama uygulanmıştır. Burada k=10 olarak belirlenmiştir. Belirtilen sonuçlar, her katlamada elde edilen doğruluk oranlarının ortalamasıdır. Diğer modellerle karşılaştırmak için de benzer şekilde her katlamada kullanılan test kümeleri üzerinde ayrı ayrı modellerin doğrulukları hesaplanıp ortalaması alınmıştır.

Diğer karşılaştırılan modeller zaten kural tabanlı olduğu için ayrı bir eğitim kümesine ihtiyaç duymamaktadırlar. Dolayısıyla diğer algoritmalar için sadece oluşturulan test kümesi kullanılmıştır.

4.3. Deneyler ve Sonuçlar (Experiments And Results)

Algoritma ve modellerin performansı doğruluk (accuracy) cinsinden ölçülmüştür. Buna göre;

$$\text{doğruluk} = 100 * \frac{\text{doğru kök sayısı}}{\text{toplam sözcük sayısı}}$$

ile her algoritmanın doğru kök bulma oranı hesaplanmıştır.

İlk olarak, kök bulma problemi için optimum mimari ve parametrelerin bulunması amacıyla hiperparametrelerin ayarlanmasına (hyperparameter tuning) yönelik deneyler gerçekleştirilmiştir. Bu amaçla, girdi vektör boyutları ve kullanılan LSTM'lerdeki katman sayıları da değiştirilerek hem temel modelde, hem de ilgi mekanizması tabanlı model üzerinde farklı mimariler denenmiştir. Bu amaçla eğitim kümesi (training set) olarak %80, geçiş kümesi (validation set) olarak %10 ve test kümesi (test set) için %10'luk bir bölüm ayrılmış ve hiperparametreler geçiş kümesi üzerindeki kaybı minimize edecek şekilde sabitlenerek eğitim kümesi üzerinde eğitim gerçekleştirilmiş ve son olarak test kümesinin doğruluğu hesaplanmıştır. Hiperparametre seçiminde ızgara araması (grid search) yöntemi uygulanarak tüm olası hiperparametre kombinasyonları ile test edilmiştir. LSTM'lerin girdi vektörlerinin boyutu, katman sayıları ve çıktı vektörlerinin boyutları üç temel hiperparametreyi oluşturmaktadır. Üçünün bütün kombinasyonları için arama uzayı çok büyük olduğundan, parametre uzayı kademeli olarak artırılmış ve öncelikle LSTM katman sayısı ve girdi vektör boyutu ızgara biçiminde farklı kombinasyonlarla çalıştırılmış, sonrasında ise en iyi sonuç alınan katman sayısı-girdi vektörü çifti kombinasyonu ile, LSTM çıktı vektörünün boyutu değiştirilerek farklı değerler ile yine ızgara yöntemi ile deneyler gerçekleştirilmiştir.

Temel model üzerinde 2 farklı LSTM katman sayısı ve 3 farklı girdi vektör boyutu (32, 64, 128) ile ızgara yöntemi uygulanarak elde edilen sonuçlar *Tablo 1*'de verilmiştir. En iyi doğruluk oranı 2 LSTM katmanı ve 32 girdi vektör boyutu ile 70.01% doğruluk oranı ile elde edilmiştir. Sonuçlar, katman sayısının mimari üzerinde olumlu bir etkisinin olduğunu göstermektedir, ancak vektör boyutunun artırılması sonuçlar üzerinde olumlu bir etki yaratmamıştır. Bu nedenle çıktı vektör boyutunun farklı değerleri 2 katmanlı LSTM ve 32 girdi vektör boyutu ile test edilmiştir.

Tablo 1. Temel modelde LSTM'lere ait katman sayısı ve karakterlerin vektörel gösterimlerinin boyutlarının sonuçlar üzerindeki etkisi

(The effect of the number of layers in the LSTMs and the dimension of the vector representations of the characters on the results in the baseline model)

| | 32 girdi v. | 64 girdi v. | 128 girdi v. |
|----------|--------------|-------------|--------------|
| 1 katman | 67.51 | 66.47 | 68.03 |
| 2 katman | 70.01 | 68.15 | 69.19 |

2 LSTM katmanı ve 32 girdi vektör boyutu ile, çıktı vektör boyutunun 32, 64 ve 128 boyutları ile üç farklı deney gerçekleştirilmiş ve en iyi sonuç 78.88% doğruluk oranı ile 128 çıktı vektör boyutundan elde edilmiştir (bkz. *Tablo 2*). Bu nedenle, hiperparametreler temel model için 2 katman, 32 girdi vektör boyutu ve 128 çıktı vektör boyutu ile sonraki deneylerde kullanılmak üzere sabitlenmiştir.

Tablo 2. Temel modelde LSTM'lere ait çıktı karakterlerinin vektörel gösterimlerinin boyutlarının sonuçlar üzerindeki etkisi

(The effect of the number of layers in the LSTMs and the dimension of the output vector representations of the characters on the results in the baseline model)

| | 32 çıktı v. | 64 çıktı v. | 128 çıktı v. |
|---------------------------|-------------|-------------|--------------|
| 2 katman-32 girdi vektörü | 70.01 | 75.40 | 78.88 |

İlgi mekanizması tabanlı model üzerinde de benzer deneyler gerçekleştirilmiştir. LSTM katman sayısı ve girdi vektör boyutunun farklı değerleri için izgara yöntemi sonucu elde edilen sonuçlar *Tablo 3*'te verilmiştir. Sonuçlara göre, vektör boyutunun sonuçlar üzerinde olumlu bir etki yarattığı gözlenmiştir. Vektör boyutunun artırılması, hem 1 katmanlı LSTM kullanıldığında, hem de 2 katmanlı LSTM kullanıldığında sonuçları genel olarak iyileştirmiştir. Katman sayısının artırılmasının ise sonuçlar üzerinde pozitif bir etkisinin olmadığı gözlenmiştir. En yüksek doğruluk oranı 82.59% ile 1 katmanlı LSTM ve 128 boyutlu girdi vektörü kullanıldığında alınmıştır.

Bu kombinasyon ile LSTM durumlarının farklı çıktı vektör boyutlarıyla farklı kombinasyonlar denenmiş ve en yüksek sonuç 84.91% doğruluk oranı ile 128 çıktı vektör boyutundan elde edilmiştir. İlgi mekanizması tabanlı model için de 1 LSTM katmanı, 128 girdi vektör boyutu ve 128 çıktı vektör boyutu ile mimari, sonraki deneylerde kullanılmak üzere sabitlenmiştir.

Belirlenen mimariler ile yapılan deneyler sonucunda, temel model (DeepStemmer-1) ve ilgi mekanizması üzerine kurulu olan modelden (DeepStemmer-2) alınan doğruluk oranlarına ait sonuçlar *Tablo 5*'te verilmiştir. İlk iki satırda verilen sonuçlar 15519 sözcüklük eğitim kümesi ve 1724 sözcüklük test kümesinden elde edilen 10 katlamalı çapraz doğrulama sonuçlarına aittir. İlgi mekanizması ile birlikte sonuçlar 79.62%'den 85.16%'ya yükselmiştir. İlgi mekanizmasının eklenmesi yaklaşık olarak %6'lık bir yükselme sağlamıştır. Bu iki deneyde test kümesi, eğitim kümesinden tamamen farklı sözcükler içermektedir. Diğer bir deyişle, verilen doğruluk oranları daha önce eğitim esnasında hiç karşılaşılmayan sözcükler için verilen doğruluk oranlarıdır.

Tablo 3. İlgi mekanizması tabanlı modelde LSTM'lere ait katman sayısı ve karakterlerin vektörel gösterimlerinin boyutlarının sonuçlar üzerindeki etkisi

(The effect of the number of layers in the LSTMs and the dimension of the vector representations of the characters on the results)

| | 32 girdi v. | 64 girdi v. | 128 girdi v. |
|----------|-------------|-------------|--------------|
| 1 katman | 81.43 | 82.01 | 82.59 |
| 2 katman | 81.43 | 80.68 | 82.19 |

Tablo 4. İlgi mekanizması tabanlı modelde LSTM'lere ait çıktı karakterlerinin vektörel gösterimlerinin boyutlarının sonuçlar üzerindeki etkisi

(The effect of the number of layers in the LSTMs and the dimension of the output vector representations of the characters on the results)

| | 32 çıktı v. | 64 çıktı v. | 128 çıktı v. |
|----------------------------|-------------|-------------|--------------|
| 1 katman-128 girdi vektörü | 82.59 | 84.51 | 84.91 |

10 katlamalı çapraz doğrulama deneyine ek olarak, eğitim kümesinin boyutunu nispeten artırabilmek için 100 katlamalı çapraz deneyler de yapılmış, böylece tüm veri kümesinin 1/100'ü test kümesi olarak, geri kalanı ise eğitim veri kümesi olarak ayrılarak bir deney gerçekleştirilmiştir. Sonuçlar k=100 olarak aynı tabloda verilmiştir. Verilen sonuçlar aynı şekilde tüm katlamalarda elde edilen doğruluk değerlerinin ortalaması alınarak elde edilmiştir. Temel modelde 81.12% doğruluk oranı elde edilirken, ilgi mekanizması tabanlı modelde 86.77% doğruluk oranı elde edilmiştir. 10 katlamalı sonuçlara göre her iki modelde de yaklaşık olarak 1.5% oranında bir yükselme gerçekleşmiştir.

Tablo 5. Önerilen modellerden 10 ve 100 katlamalı çapraz doğrulama sonucu elde edilen kök bulma doğruluk oranları

(The stemming accuracies obtained from the proposed models with 10 and 100 fold cross validation)

| Model | Doğruluk |
|-----------------------|----------|
| DeepStemmer-1 (k=10) | 79.62 |
| DeepStemmer-2 (k=10) | 85.16 |
| DeepStemmer-1 (k=100) | 81.12 |
| DeepStemmer-2 (k=100) | 86.77 |

10 katlamalı çapraz doğrulama sırasında kullanılan test kümeleri karşılaştırma amaçlı olarak diğer kök bulma algoritmalarının testi için kullanılmıştır. 10 farklı test kümesinden elde edilen doğruluk oranlarının ortalaması tüm modeller için *Tablo 6*'da verilmiştir. Karşılaştırılan diğer algoritmalar Snowball [10], Resha [11], PC-Kimmo [14] ve Zemberek [16]'tir. Oflazer tarafından önerilen iki katmanlı morfolojik analiz modelinin iki farklı versiyonuyla ayrı ayrı karşılaştırma yapılmıştır. PC-Kimmo – unk olarak belirtilen model bilinmeyen sözcükler için analiz önermek için önceki versiyona göre iyileştirilmiş modeldir. Bu modellerden hiçbiri istatistiksel olmadığından ötürü ayrı bir eğitim sürecine gerek duymamaktadır. Dolayısıyla doğrudan test işlemi gerçekleştirilmiştir.

Tablo 6. Elde edilen kök bulma doğruluk oranlarının diğer modellerle karşılaştırması
(The comparison of the stemming accuracy with the other models)

| Model | Doğruluk |
|---------------------|--------------|
| PC-Kimmo - unk [14] | 94.66 |
| PC-Kimmo [14] | 92.97 |
| Zemberek [16] | 91.01 |
| DeepStemmer-2 | 85.16 |
| DeepStemmer-1 | 79.62 |
| Resha [11] | 65.38 |
| Snowball [10] | 48.34 |

Elde edilen sonuçlara göre, PC-Kimmo [14]'nin bilinmeyen sözcükler için iyileştirilmiş versiyonu en iyi sonuçları vermektedir. Büyük bir sözlük ve tüm Türkçe kurallarının sistemde hazır bulunması nedeniyle çoğu sözcüğün analizini doğru olarak yapabilmektedir. Zemberek [16] ve PC-Kimmo [14] sözcükler için birden fazla morfolojik analiz sunabilmektedir. Doğruluk hesabı yapılırken, önerilen analizler arasında doğru kökün olup olmamasına göre analiz yapılmıştır. Doğru kök analizinin, önerilen analiz listesinde olması durumunda algoritmanın doğru kökü önerdiği varsayılmıştır. Önerdiğimiz Deep Stemmer modeli için de bağlamı dikkate aldığımız herhangi bir belirsizleştirme (disambiguation) işlemi yapılmadığı için diğer algoritmalar için de bağlama göre doğru analizin bulunup bulunmaması durumu göz ardı edilmiştir.

Diğer algoritmalarda test kümesinde verilen sözcükler sözlüklerinde zaten var olduğundan dolayı, test kümesinin tamamının eğitim kümesine dahil edildiği durumda önerdiğimiz modelin vereceği sonuçlarla karşılaştırmak daha adil olacaktır. Ancak derin öğrenme modelinin verilen test kümesini ezberleme ihtimali de göz önünde bulundurularak, burada test kümesinin tamamının hiç görülmeyen sözcüklerden oluştuğu durumla karşılaştırılmıştır. Şunu da belirtmek gerekir ki, Metu-Sabancı derlemi PC-Kimmo modeli ile otomatik olarak analiz edilmiş ve sonrasında elle kontrolü yapılmıştır. Dolayısıyla PC-Kimmo sonuçları derlemde belirtilen sonuçlarla neredeyse bire bir aynıdır ve sonuçlarının yüksek olması beklenen bir durumdur. Diğer otomatik olarak analiz yapan sistemlerle karşılaştırıldığında ise, diğer modeller sözlük kullanmasına rağmen bizim modelimiz diğerlerine göre daha yüksek doğruluk oranına sahiptir.

Eğitim veri kümesinin boyutunu artırmanın bir etkisinin olup olmayacağını test etmek amacıyla, tamamen doğru kökleri içermese bile, doğruluk oranı yüksek olan başka bir kök bulma algoritması kullanılarak büyük bir veri kümesinin kök analizi yapılmış ve eğitim kümesi genişletilmiştir. Bu amaçla, morfolojik analiz sistemi olarak Zemberek [16] ve veri kümesi olarak herkese açık bir şekilde paylaşılan Morpho Challenge 2010 [21] ham sözcük listesi kullanılmıştır. Morpho Challenge 2019 [21] tarafından Türkçe için verilen ham sözcük listesi 381,098 sözcük içermektedir. Verideki gürültüyü azaltmak için bu sözcük listesinden frekansı 5'ten küçük olanlar elenerek

toplam 100,412 sözcük içeren bir veri kümesi elde edilmiştir. Bu sözcüklerin Zemberek ile kök analizleri yapılmış ve önerilen kök analizlerinin hepsi eğitim kümesine dahil edilmiştir. Metu-Sabancı derleminden alınan 17243 sözcüğün 12,000'i test amaçlı olarak ayrılmış ve geri kalan 5243 sözcük Morpho Challenge verisine eklenerek toplam 93,268 sözcüklük bir eğitim kümesi elde edilmiştir. Test kümesindeki tüm sözcükler eğitim kümesinden çıkarılmıştır. Son olarak, veri kümesinin büyüklüğünün etkisini ölçmek amacıyla Zemberek [16] tarafından analiz edilen sözcükler de eğitim kümesine dahil edilmiştir. Veri kümeleriyle ilgili ayrıntılı bilgi 5.2. Veri Kümesi bölümünde verilmiştir. Hem kod çözücü için, hem de kodlayıcı için 1 katmanlı LSTM ve girdi-çıkıta uzayına ait karakterler için vektör boyutu 32 olarak verilmiştir. İlgi mekanizması kullanan modelden 73.65% doğruluk oranı elde edilmiştir. Farklı veri kümesi boyutlarında deneyler yapılsa da sonuç hep benzer olmuştur. Bu durumda, veri kümesinin artırılmasının sonuçlar üzerinde olumlu bir etki yaratmadığı söylenebilir. Bunun bir nedeni de eğitim kümesinin gürültülü olması olabilir.

İlgi mekanizması tabanlı modelin 10 çapraz doğrulamalı çalışma süresi toplam 100 epoch için 5 saat sürmektedir. Temel modelde ise aynı parametrelerle bir eğitim yaklaşık olarak 4.5 saat sürmektedir. Kullanılan makine Intel Xeon 2.4 GHz, RAM 192 Gb özelliklerine sahiptir.

Sonuçlar incelenerek yapılan hata analizinde hataların çoğunlukla sık karşılaşılmayan sözcüklerde meydana geldiği gözlenmiştir. Sık karşılaşılan sözcükler çoğunlukla doğru analiz edilmiştir. Diğer algoritmalarından farklı olarak, yanlış bulunan kökler kod çözücünün sıfırdan oluşturduğu karakterlerden oluştuğu için sözcükten oldukça uzak ve geçersiz kökler de oluşturulabilmektedir.

Doğru ve yanlış bulunan köklere örnekler Tablo 7'de verilmiştir. Örneğin, *referanduma*, *sekizyüzelliüç*, *petrolleri* gibi frekansı düşük olan sözcükler doğru analiz edilememiştir. Ancak, *görünüyordu*, *uygulanabileceğini*, *söylemeyecektim* gibi frekansı düşük de olsa köklerinin frekansı yüksek olan sözcükler doğru analiz edilebilmiştir.

İlgi mekanizması içeren model için de benzer bir hata analizi yapıldığında *referanduma*, *sekizyüzelliüç*, *petrolleri* gibi frekansı düşük olan sözcüklerin doğru analiz edildiği, buna rağmen çok daha frekansı düşük olan *elektrozayıf* gibi sözcüklerin doğru analiz edilemediği gözlenmiştir (bkz. Tablo 8). Temel modele göre ilgi mekanizması dahil edildiğinde seyreklik problemiyle başa çıkılabilmektedir.

Tablo 7. Temel modelden elde edilen sonuçlarda doğru ve yanlış kök örnekleri

(Examples to correct and incorrect stems obtained from the baseline model)

| Sözcük | Doğru kök | Tahmin edilen kök |
|--------------------|-------------|-------------------|
| duruyorsunuz | dur | dur |
| görünüyordu | görün | görün |
| referandumu | referandum | referaned |
| petrolleri | petrol | pet |
| sekizyüzeği | sekizyüzeği | sekizyen |
| uygulanabileceğini | uygula | uygula |
| söylemeyecektim | söyle | söyle |
| varsayımıyla | varsayım | var |
| dönüşmesi | dön | dönüş |

Tablo 8. İlgili mekanizması içeren modelden elde edilen sonuçlarda doğru ve yanlış kök örnekleri

(Examples to correct and incorrect stems obtained from the attention model)

| Sözcük | Doğru kök | Tahmin edilen kök |
|---------------|--------------|-------------------|
| referandumu | referandum | referandum |
| sekizyüzeği | sekizyüzeği | sekizyüzeği |
| varsayımıyla | varsayım | varsayım |
| dönüşmesi | dön | dönüş |
| petrolleri | petrol | petrol |
| elektrozayıf | elektrozayıf | elektrozayıf |
| haberliyim | haber | haberli |
| onunla | o | o |
| hissedemezler | hisset | hisset |
| modellerini | model | model |

5. SONUÇ, TARTIŞMA VE GELECEK ÇALIŞMALAR (CONCLUSION, DISCUSSION AND FUTURE WORK)

Bu çalışmada, Türkçede kök bulma problemi için derin öğrenme tabanlı iki farklı model önerilmiştir. Önerilen modeller diğer çoğu kök bulma algoritmasında olduğu gibi el ile oluşturulan bir kural listesine ihtiyaç duymamakta, sadece sözcük ve kök çiftlerinden oluşan bir eğitim kümesi verilerek eğitilebilmektedir. Böylece, diğer algoritmalarda kullanılan kural listelerine gerek kalmadan daha hızlı bir şekilde kök bulma modeli oluşturulabilmektedir.

Önerilen modellerden ilki kodlayıcı-kod çözücü prensibine dayanmakta ve her iki bileşen için de LSTM kullanılmaktadır. Önerilen modellerden ikincisi ise, ilkinin göre farklı olarak, kök bulurken sözcüğün hangi karakterlerine odaklanacağını bir ilgi mekanizması yardımıyla öğrenmekte ve öğrendiği ağırlıklar sayesinde sözcüğü içeren karakterlerden farklı anlamlar çıkabilmektedir. Bu özellik, seyrek sözcüklerde oluşan kök hataları da nispeten giderilebilmiştir.

İki model için gerçekleştirilen farklı mimari özelliklerin farklı sonuçlar verdiği gözlemlenmiştir. Temel modelde 2 katmanlı LSTM'in daha iyi öğrendiği gözlenirken, ilgi

mekanizması eklendiğinde tek katmanlı LSTM en iyi sonucu vermiştir. Modelin, farklı karakterler için öğrendiği farklı ağırlıklar sayesinde daha basit bir mimari yapıyla çok daha iyi sonuçlar verdiği gözlemlenmiştir. Karakterlerin ağırlıklarının da modele dahil edilmesiyle birlikte, karakterler için temel modelde kullanılan 32'lik girdi vektörü, ilgi mekanizmasında yeterli gelmemiş, 128'lik girdi vektörüyle çok daha iyi sonuç alınmıştır. Bu da, mimarinin parametrelerin üzerinde önemli bir etkisi olduğunu göstermektedir.

Belirlenen hiperparametrelerle gerçekleştirilen deneyler sonucunda, ilgi mekanizması tabanlı modelin temel modele göre daha iyi sonuç verdiği gözlemlenmiştir. Temel modelden 79.62% doğruluk oranı elde edilirken, bu oran ilgi mekanizmasının eklenmesiyle birlikte 85.16%'ya çıkmıştır. Hatta 100 katlamalı çapraz doğrulama ile eğitim kümesinin artırılması durumunda bu oran 86.77%'ye kadar yükselmektedir. Bu sonuçlar, PC-Kimmo [14] ve Zemberek [16]'e göre nispeten düşük olsa da, diğer kök bulma algoritmalarına göre daha yüksektir.

Önerilen modeller, Türkçe üzerinde test edilmiş de olsa, dile bağımlı herhangi bir kural listesi kullanılmadığı için başka dilde verilecek herhangi bir eğitim kümesi ile de diğer diller için kullanılabilir. Gelecek çalışmalarımızın arasında, önerilen kök bulma modelini başka diller için de test etmek bulunmaktadır.

Yapılan çalışma, bağlamı dikkate almadan sözcük tabanlı çalışmaktadır. Gelecekte, cümle analizinin mümkün olabileceği ve sesteş sözcüklerin bağlama göre köklerinin bulunabileceği şekilde modelin revize edilmesi planlanmaktadır. Bir başka gelecek çalışma olarak, diğer Türkçe kök bulma algoritmalarının ek bir sözlük kullanma özelliğini derin öğrenme ile birleştirmeyi hedefliyoruz. Böylece, modelin dilde olmayan kökler oluşturmasının da önüne geçilebileceğini düşünüyoruz.

KAYNAKLAR (REFERENCES)

- [1] I. Sutskever, O. Vinyals, Q. V. Le, "Sequence to Sequence Learning with Neural Networks", *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems*, Montreal, 2014.
- [2] M.-T. Luong, H. Pham, C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation", *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, 2015.
- [3] K. Cho, B. v. Merriënboer, D. Bahdanau, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches", *Proceedings of SST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, Doha, 2014.
- [4] J. B. Lovins, "Development of a Stemming Algorithm", *Defense Technical Information Center*, 31, 1968.
- [5] M. F. Porter, "An Algorithm for Suffix Stripping", *Readings in Information Retrieval*, 313-316, 1997.
- [6] R. Krovetz, "Viewing Morphology as an Inference Process", *Proceedings of the 16th Annual International ACM SIGIR*

- Conference on Research and Development in Information Retrieval**, 1993.
- [7] Internet: M. F. Porter, Snowball, <http://www.snowball.tartarus.org/texts/introduction.html>, 2001.
- [8] E. K. Çilden, *Stemming Turkish Words Using Snowball*, 2006.
- [9] G. Eryiğit, E. Adalı, “An Affix Stripping Morphological Analyzer for Turkish”, **Proceedings of the IASTED International Conference Artificial Intelligence and Applications**, Innsbruck, 2004.
- [10] Internet: O. Tunçelli, Github, <https://github.com/otuncelli/turkish-stemmer-python>, 09.11.2018.
- [11] Internet: H. R. Zafer, Github, <https://github.com/hrzafer/resha-turkish-stemmer>, Kasım 2018.
- [12] Internet: H. R. Zafer, Github, <https://github.com/hrzafer/nuve>, 09.11.2018.
- [13] K. Koskenniemi, P. Tapanainen, A. Voutilainen, “Compiling and Using Finite-State Syntactic Rules”, **Proceedings of the COLING-92, the 14th International Conference on Computational Linguistics**, Nantes.
- [14] K. Oflazer, “Two-level Description of Turkish Morphology”, **In Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics**, 1994.
- [15] C. Çöltekin, “A Freely Available Morphological Analyzer for Turkish”, **Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)**, 2010.
- [16] A. A. Akın, M. D. Akın, “Zemberek, An Open Source NLP Framework for Turkic Languages”, *Structure*, 10, 1--5, 2007.
- [17] S. Hochreiter, “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions”, *International Journal of Uncertainty, Fuzziness, Knowledge-Based Systems*, 6(2), 107-116, 1998.
- [18] G. Neubig, C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Ballesteros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, “DyNet: The Dynamic Neural Network Toolkit”, *ArXiv*, 2017.
- [19] N. B. Atalay, K. Oflazer, B. Say, “The Annotation Process in the Turkish Treebank”, **Proceedings of the EACL Workshop on Linguistically Interpreted Corpora - LINC**, Budapeşte, 2003.
- [20] K. Oflazer, D. Say, D. Z. Hakkani-Tür, G. Tür, “Building a Turkish Treebank”, **Building and Exploiting Syntactically-annotated Corpora**, Kluwer Academic Publishers, 2003.
- [21] Internet: M. Kurimo, K. Lagus, S. Virpioja, V. Turunen, Morpho Challenge 2010, Aalto University, <http://morpho.aalto.fi/events/morphochallenge2010/>, November 2018].