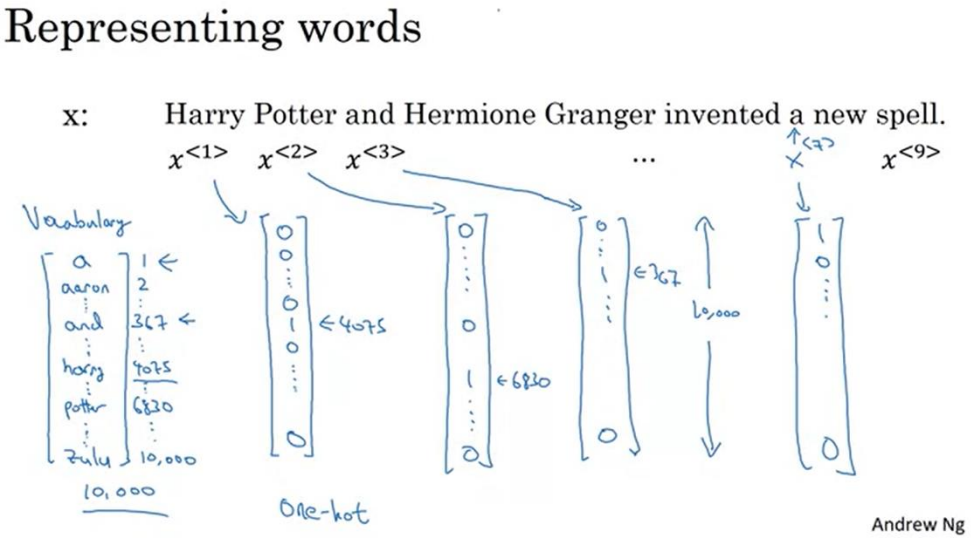
WEEK 1

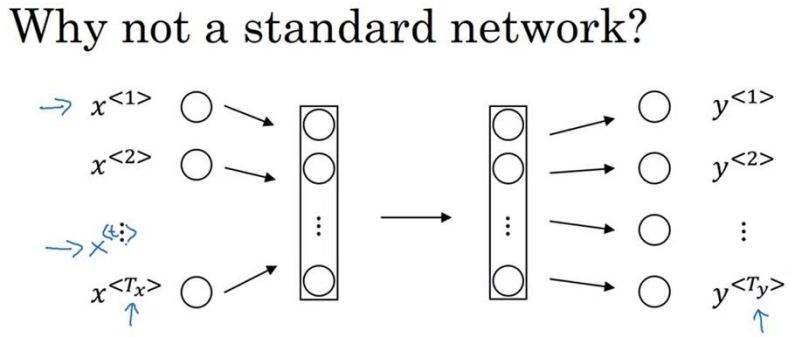
Week 1 – Notation

Burada Andrew kelimelerin nasıl represente edilebileceğini bakmış. Hatırladığım gibi önce bir dictionary/vocabulary oluşturulmuş ve her kelime bu vocab kullanılarak one-hot-encoded vectors olarak represente edilmiş.

Pratik kursta olay böyle değildi, kelimeler sadece 0-10k(vocab size) arasında bir sayı olarak temsil ediliyordu, daha sonra NN'e beslenirken embedding layer ile bu iş kotarılıyordu.



Week 1 – RNN Model



Sequence data için standard model kullanmamızın 2 sebebi var. İlki input ve output size'ın sabit kalmayabilecek olması, her cümlenin uzunluğu aynı olmadığına göre sabit boyutlu bir model eğitmemiz mümkün değil!

Belki bu problemi inputa 0 padding uyguyalarak vesaire çözebiliriz ama bu da iyi bir representation sayılmaz.

İkinci problem ise böyle bir network architecture'ın farklı text pozisyonları için öğrenilen feature'ları paylaşmıyor oluşu. Yani örneğin NN ilk inputa Harry verilirse bunun bir isim olduğunu öğrenmiş olabilir, ancak Harry ismi başka bir input'a verildiğinde NN bunun isim olabileceğini düşünmüyor. Yani her inputu kendi başına özelleşiyor. Biz ise pozisyon farketmeksizin cümlenin bir yerindeki kelimeyi başka bir yerinde görünce de benzer tepkiler versin isteriz.

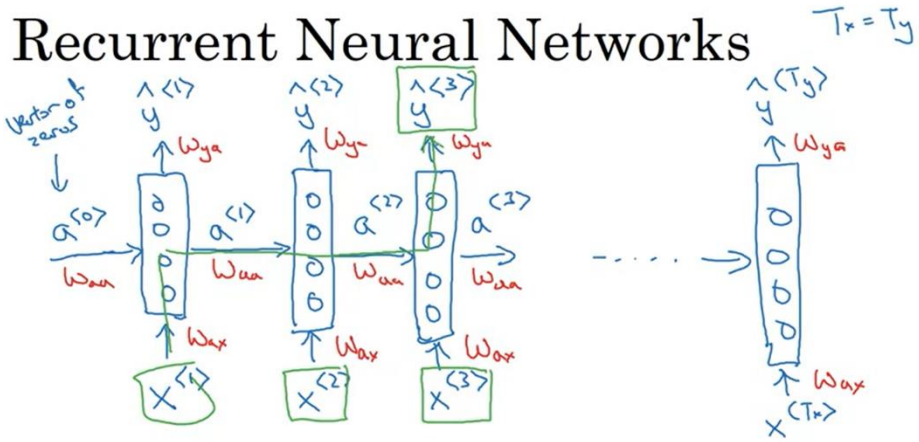
Aslında burada yapılmasını istediğimiz şey CNN gibi bir şey, hatta CNN de burada işe yarıyor zaten. Çünkü image problemi için de aynı şeyler geçerli diyelim ki image'e bakıp face detection yapıyoruz. Eğer ben düz bir NN kullanırsam, ne olacak ilk input unit sadece ilgili pixel için karar vermeyi öğrenecek, yani hep ekranın ortasında kedi fotoğrafları görünce belki bunu öğrenecek ancak kedi kenarda çıkınca model afallayacak. İşte bu yüzden biz CNN kullandık ki, filtereler ile pozisyon farketmeksizin input'un heryerini taramak ve feature detect etmek mümkün olsun.

Benzer mantık text problemlerinde de geçerli ben unitlerin spesifik olarak bir konuma focuslanmalarını istemiyorum, cümlenin geneline baksınlar ve buradan features extract edebilsinler isterim.

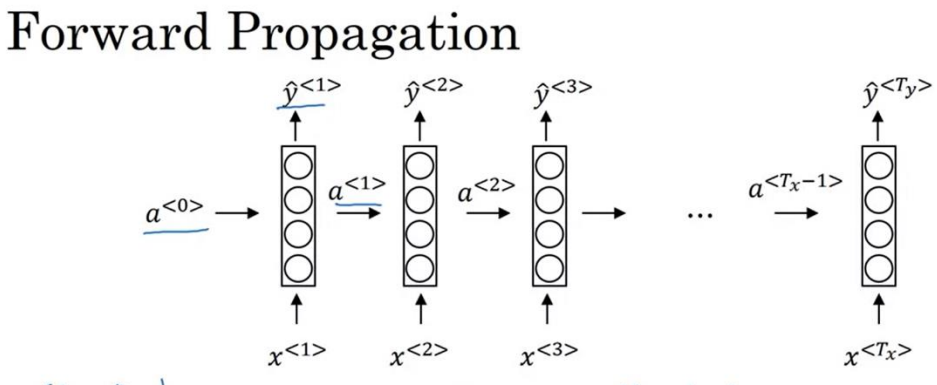
Ayrıca yine CNN'e benzer olarak eğer burada düz bir NN kullanırsak her bir input 10k'lık bir one-hot-encoded vektör olsa, en uzun cümle 20 kelimelik olsa 200k'lık bir input size söz konusu, bu modelin parametre sayısı inanılmaz büyük olacaktır. Image problemi için de aynı sıkıntı söz konusuydu CNN ile çözülmüştü.

Burada da bu problemleri RNN ile çözeceğiz, tüm bu disadvantages RNN kullanırsak ortadan kalkacak.

Aşağıda bir RNN yapısı görülüyor, her bir kelime için o kelimenin isim olup olmadığı belirlenmeye çalışıyor. RNN yapısı sabit 3 farklı parametre söz konusu, input parametreleri, aktivasyon parametreleri ve output parametreleri. Her timestep için aynı parametreler kullanılıyor. Buradaki problem şu, örneğin 3. Kelimenin isim olup olmadığına karar verirken 1. 2. 3. Inputları göz önünde bulundurmuş oluyor. Ancak cümlenin kalanına bakmamış oluyor bunun için bi-directional RNN fikri açıklanacak. Tx cümlenin uzunluğu yani 10 kelimelik bir cümle ise Tx 10 olacak. Ty'de 10 olacak.



Temiz model ve forward propagation aşağıda. Aktivasyonları hesaplarken kullanılan act. Func. Genelde relu veya tanh olur, output hesaplarken ise sigmoid veya softmax.



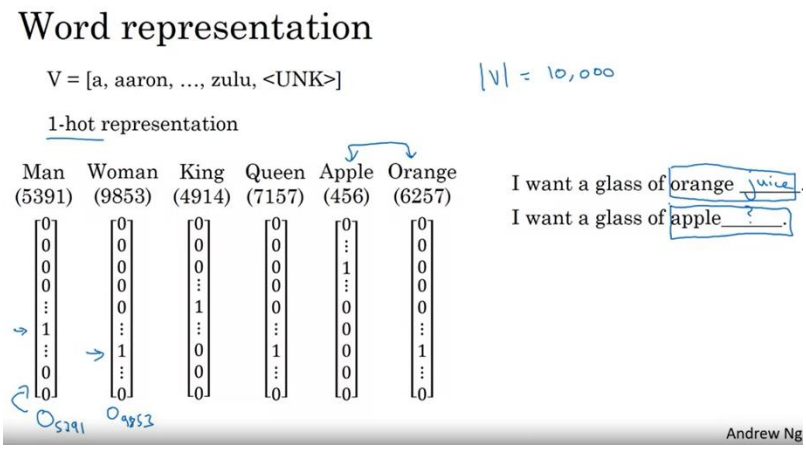
$a^{<0>} = \vec{0}$.

$$a^{<1>} = g(W_{aa} a^{<0>} + W_{ax} x^{<1>} + b_a)$$
$$\hat{y}^{<1>} = g(W_{ya} a^{<1>} + b_y)$$

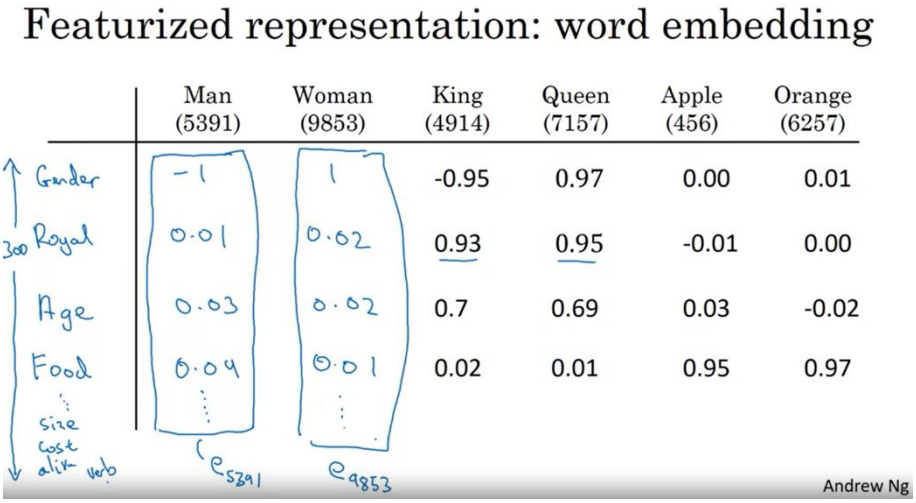
WEEK 2

Week 2 – Word Representation

Word representation için ilk yaklaşımımız aşağıdaki gibi one-hot-encoded vektörler ile idi. Ancak bu yaklaşımın bir dezavantajı şu: her kelime birbirinden tamamen bağımsız şekilde represente ediliyor bu yüzden modelin orange ile apple arasında bir bağlantı olduğundan haberi olması zorlaşıyor. Halbu ki bu representation'lar bu ikisi arasında bir bağlantı olacak şekilde kurulsaydı, böylece modelimiz tam olarak bilmediği cümlelerde bile daha iyi performans sergilerdi çünkü ona benzer kelimelerle kurulmuş cümleleri de bu tahminlerinde kullanabilirdi.

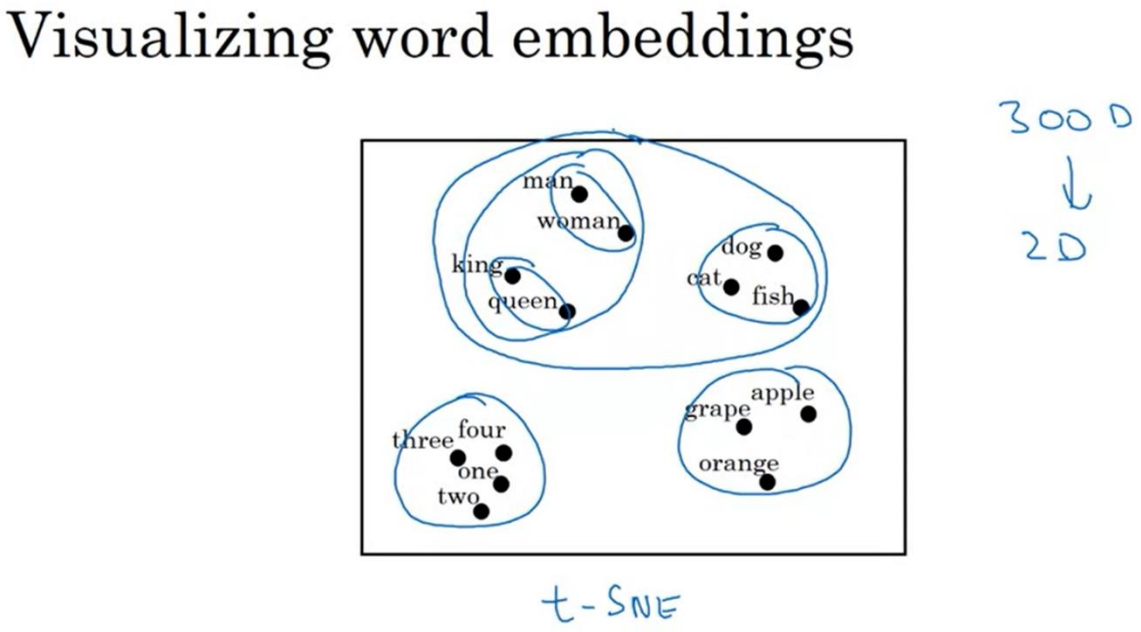


Bunun yerine aşağıdaki gibi featurized representation kullanabiliriz, böylece aslında bir kelimeyi sallıyorum 300 farklı feature ile temsil ederiz ve her feature'ın bir karşılığı olur ve bu gösterimle artık olay sadece kelimeyi numeric olarak represente etmek değil, kelimeyi uzayda bir konuma oturtarak ona anlam yüklemek.



5391. kelimenin yani Man’in one-hot-encoded gösterimini O5391 olarak gösterdik embedding vector ile gösterimini de e5391 olarak gösterdik.

Embedding sayesinde modelimiz Apple ile Orange’ın anlamsal olarak çok benzer şeyler olduğunu öğrenmiş olacak bu da modele büyük bir esneklik kazandıracak.



Embedding vectors’ü 2D represente ettiğimde yukarıdaki gibi kelimeler arasındaki bağlantıları doğrudan görebilirim.

Week 2 – Using Word Embeddings

Embedding kullanımının bir avantajı da şu: Transfer Learning'e izin veriyor. Eğer OneHotEncoding representation kullansak zaten burada öğrenilecek bir şey yok transfer de edemeyiz. Ayrıca dictionary size'ımız kısıtlı oluyor neden? Çünkü dictionary size 10k ise her kelime 10klık vektör ile represente edilecek yok 100B ise her kelime 100B'lik vektör ile temsil edilecek bu öğrenmeyi o kadar yavaşlatır ki!

İşte Embeddings burada inanılmaz bir avantaj sağlıyor, hem artık bir kelimeyi temsil etmek için 100B'lik bir veköre ihtiyacımız kalmıyor hatta 10k'lığa bile kalmıyor 256 gibi vector size ile işi çözüyoruz. Buna ek olarak da Dictionary'imiz aslında 100B kelimelik olabiliyor, ayrıca da her kelime arasındaki bağlantıları tutabiliyoruz bu inanılmaz bir kolaylık!

Ancak word embeddings ile çok büyük corpuslarda diyelim ki 100 Billions word üzeride embeddings öğrenildi bu sonuçta burada elimde 100Billions word'e karşılık bir embedding vector var.

Şimdi ben gidip benim kendi uygulamam için bu embedding'leri kullanabilirim benim uygulamam da sadece 100k word olabilir, muhtemelen benim uygulamamdaki en absurd kelime için bile bir embedding'im olacak ve bunun sağlayacağı performans artışı inanılmaz olacak bence.

Model eğitime neredeyse her kelimenin anlamları arasındaki bağlantıları bilerek başlayacak.

Ayrıca optional olarak, kendi uygulamam için embedding matrix'i fine-tune edebilirim.

Transfer learning and word embeddings

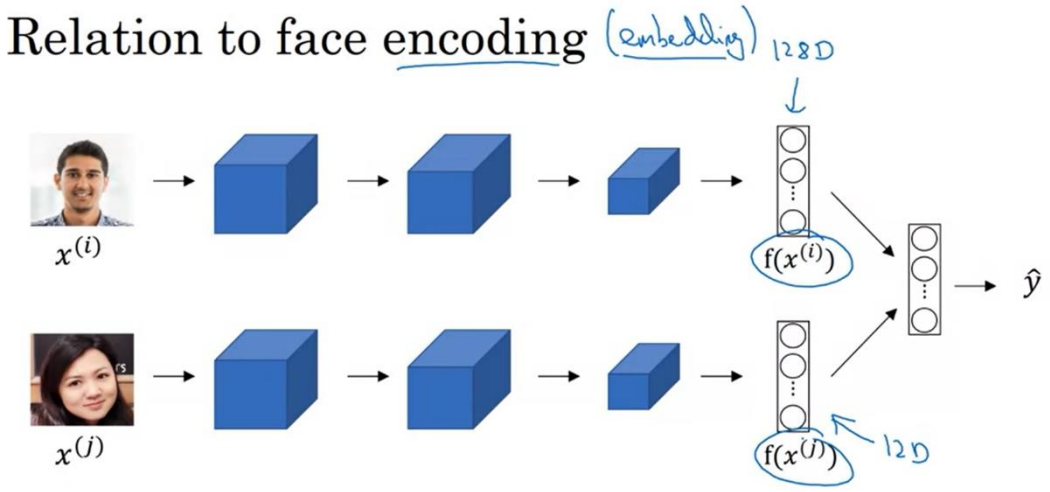
1. Learn word embeddings from large text corpus. (1-100B words)

   (Or download pre-trained embedding online.)

2. Transfer embedding to new task with smaller training set. (say, 100k words) → 10,000 → 300

3. Optional: Continue to finetune the word embeddings with new data.

Burada bir hatırlatma, önceki kurslardan face encoding'i hatırlarsın, yüzleri bir CNN kullanrak 128 feature'a encode ediyorduk daha sonra bu feature'ları kıyaslayıp aynı insan olup olmadığına karar veren bir model eğitiyorduk:



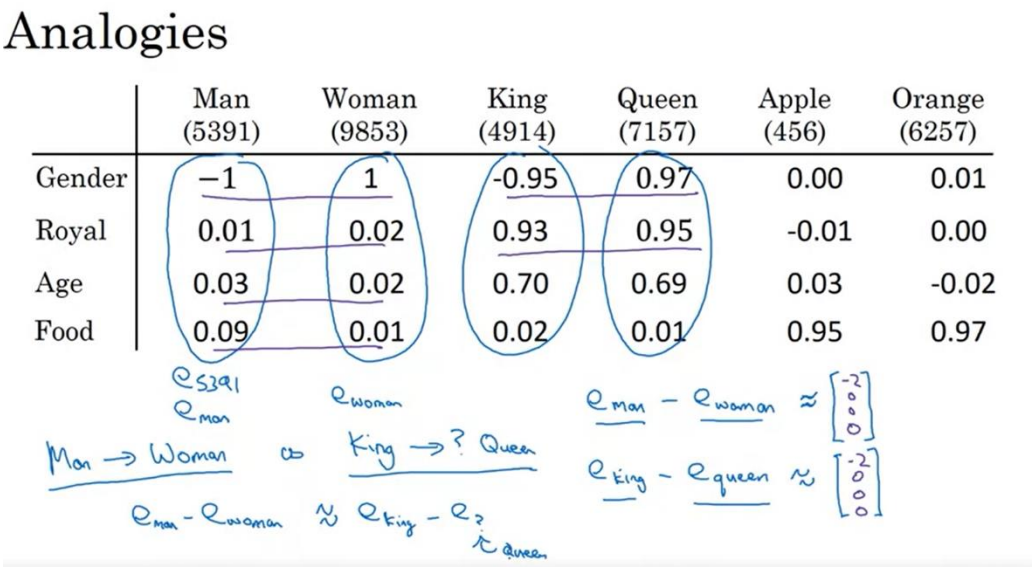
Aslında ENCODING ile EMBEDDING aynı mantık, yani embedding vector'ü bir kelimenin embedding size'lık bir vektöre encode edilmiş hali olarak düşünmek doğru olacaktır. Elbette encode edilme stratejisi farklı olacak, word embedding'I oluşturmak için word'ü bir CNN'den geçirmeyeceğiz bunu göreceğiz.

Week 2 – Properties of Word Embeddings

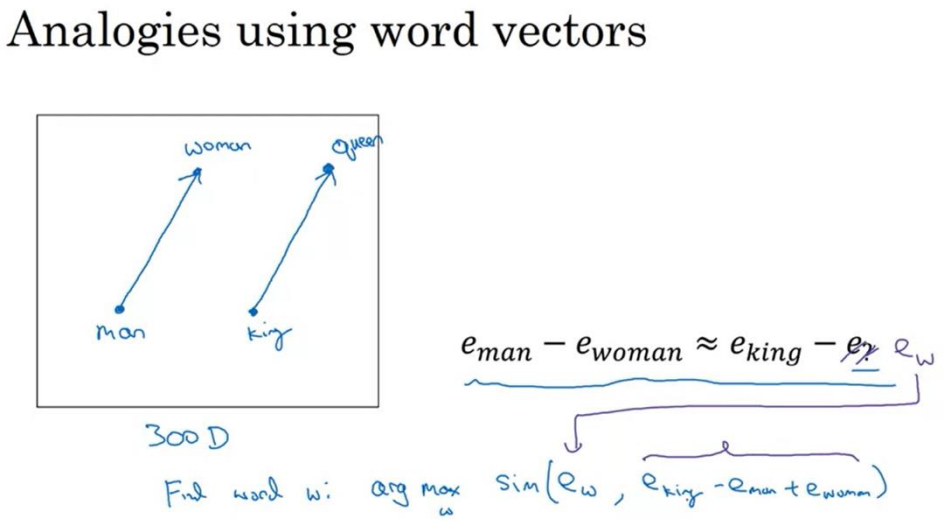
Burada word embeddings ile anolojiler kuralabileceği fikrini irdeleyeceğiz. Diyelim ki aşağıdaki giibi kelimelerim var ve embedding size=4. Ben istiyorum ki Man → Woman ise King → ? Burada ? embeddings kullanarak bulunabilir mi? Evet bulunabilir.

Yani embeddings ile model analojiler kurabiliyor. Peki nasıl bulunacak? eMan – eWoman dersek sonuçta aşağıdaki görünen [-2 0 0 0] gibi bir vektör elde edeceğiz. İki kelimenin farkını bu vektör veriyor. Doğal olarak iki kelime arasındaki en bariz fark cinsiyetleri olacak.

Şimdi biz gidip eKing - ? = [-2 0 0 0] gibi bir eşitlik kurarsak bu eşitliği sağlamaya en yakın kelimenin Queen olacağını bulabiliriz.



Bu fikri daha iyi anlayalım:

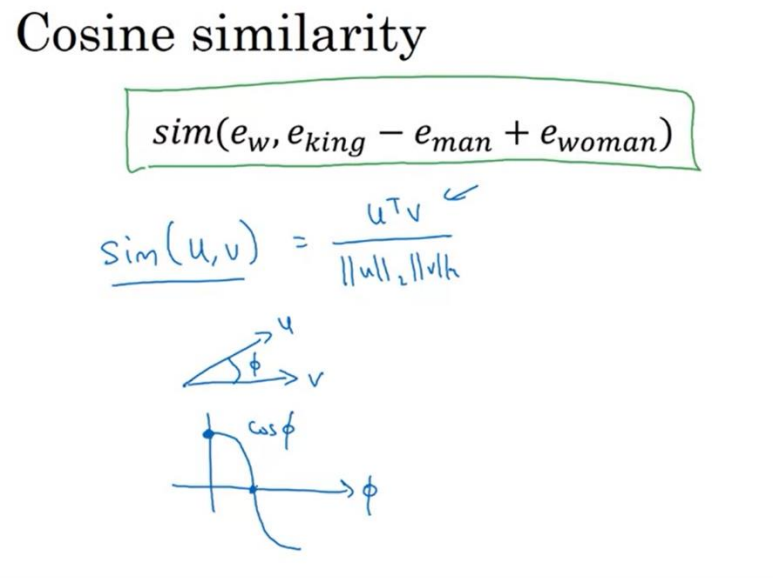


Aslında analoji için sorulacak soru yukarıdaki soru: eman-ewoman = eking-e?

300D'lik embedding space'de eman-ewoman'a ve eking-equeen'e bakarsak yukarıdaki gibi çok benzer veya aynı vektörler olduğunu görebiliriz.

İşte bu fikirle eğer equeen'i arıyorsak bir similarity function tanımlayıp bu similartiy function'I max yapan kelimeye bakabiliriz ki burada da hangi kelime vektörü ile eking-eman+ewoman vektörü çok benzer onu sorguladık. Bunun cevabı equeen çıkmalı.

Similarity function olarak cosine similarity kullanılabilir temelde iki vektör arasındaki açı ne kadar az ise cosine similarity o kadar büyük çıkar. Açı ters ise – sonuç çıkar.



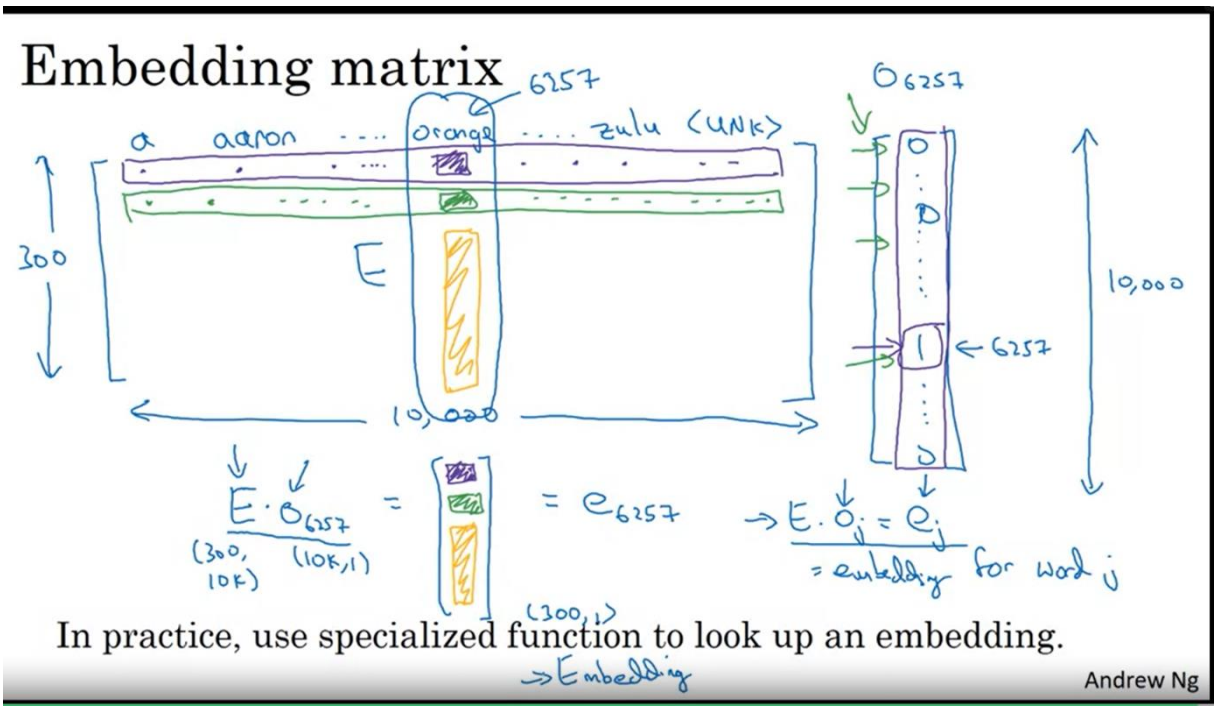
Week 2 – Embedding Matrix

Word embeddings'in ne olduğunu neye yaradığını vesaire anladık. Peki word embeddings nasıl öğreniliyor bunu basitçe anlayalım.

Temelde öğrenme yöntemi şu: Diyelim ki 10k kelimelik bir dictionary'imiz olsun, embedding dimension=300 olsun yani her kelimeyi 300 feature ile temsil edeceğiz.

O halde Embedding Matrix 300x10 000'lik bir matrix olacak, bu matrix random initialize edilecek ve modelimiz task'I gerçekleştirmeye çalışırken aynı zamanda bu matrix weight'lerini gradient descent ile update edecek.

Bir kelimenin embedding vector'ünü bulmak için basitçe bir matrix multiplication yapabiliriz. Örneğin eğitim sırasında 6257. Kelimenin embedding'ini modele beslememiz lazım, E*O6257 ile ki O6257 6257. Indexi 1 olan bir one-hot-encoded vector'den fazlası değil ilgili kelimenin embedding vector'üne ulaşabiliriz. Aslında bu işlemin bir lookup'dan fazlası olmadığına dikkat et. Zaten pratikte bu lookup için matrix multiplication yerine specialized look up function kullanılır.

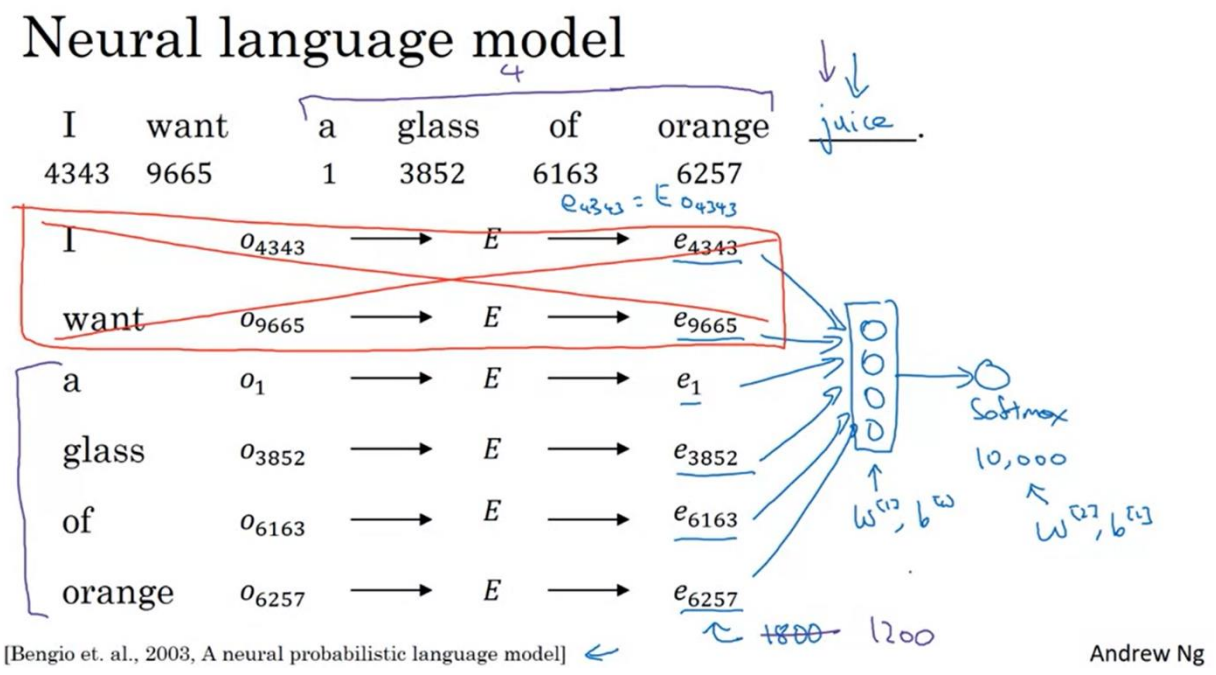


Week 2 – Learning Word Embeddings

Burada embeddings'in nasıl öğrenildiğini formüzüle etmeye çalışacağız. Önce daha komplex algoritmalardan başlayıp sonra basite kayacağız. Kronolojik olarak da böyle olmuş, ilk başta insanlar embeddings'in öğrenilmesi için kompleks algoritmalara başvurmuşlar, daha sonra ise giderek algoritmalar sadeleşmiş ve aynı sonuçlar veya daha iyileri elde edilmiş.

Ancak andrew diyor ki size direkt günce algoritmayı versem dersin ki ya bu kadar basit bir şey nasıl çalışıyor, magical görünür diyor o yüzden zordan başlayacağız.

Standard bir NN ile language model oluşturduğumuzu varsayalım, son 4 kelimeye bakarak sıradaki kelimeyi tahmin etmeye çalışsın: Cümlemiz aşağıdaki gibi başlıyor, her bir kelimenin dictionary index'I verilmiş, her kelimeye karşılık embedding vector lookup ile bulunur, her bir embedding size 300 olsun, sadece son 4 kelimeye baktığımız için aşağıdaki çizimde ilk 2 kelime karalanmış



Son 4 kelime yani 4*300=1200'lük bir input embedding grubu bir fully connected layer'a ve oradan da 10k'lık yani dictionary size'lık bir softmax'e besleniyor ve sonuçta softmax next word'ü tahmin etmeye çalışıyor bu örnek için target word juice olacak, model bilemezse ona göre gradient descent ile parametreleri yenileyecek.

Sonuçta bu modelin parametreleri sadece W ve b değil E'yi yani embedding matrix'i de modele bir parametre olarak veriyoruz, modelimiz gradient descent ile embedding matrix'i de update ederek aslında, verilen kelimelerden next word'ü tahmin edebilecek bir model'I eğitirken aynı anda bu işlevi daha hızlı yapabilmesi için gerekli olan kelime düzenini ona sağlayacak bir embedding matrix de eğitiyor.

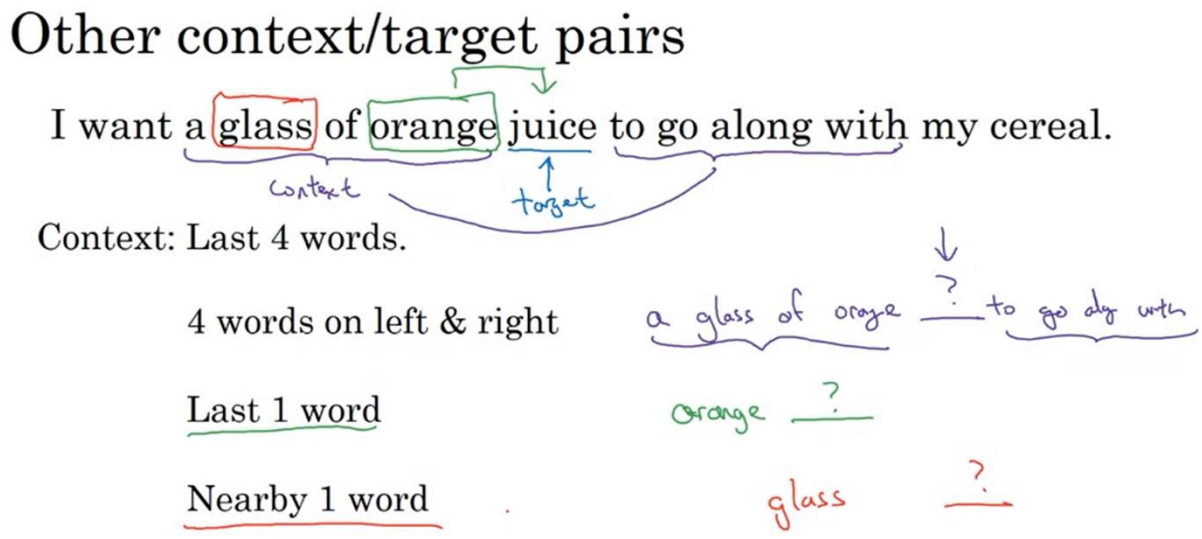
Sonuçta modelden her kelimeyi 300'lük boyutta encode etmesini ve bu encoding'I kullanarak tahmin yapmasını istiyoruz e doğal olarak modelimiz benzer kelimeleri benzer şekilde encode etmeli ki sağlıklı sonuçlar elde edebilsin yani eğer apple ile orange çok alakasız yerlerde ise FC Layer'a orange'ın embedding'I girdiğinde ve apple'ın embedding'I girdiğinde aynı sonucu(mesela sıradaki kelime juice olacak) elde etmeyi bekleyemeyiz çünkü FC Layer ikisini de aynı işlemleri uygulayacak, doğal olarak embedding vectors öğrenilmek zorunda!

Bu yaklaşım görece eski ve ve karmaşık, ama arkasındaki intuiton gayet mantıklı. Modelin başarılı olmak için embeddings'I öğrenmekten başka çaresi yok.

Şimdi görece daha basit ve yeni algoritmalara bakalım:

Önceki örnekte bir language model öğrendiğimiz için, modelimize son 4 kelimeyi beslemiştik ve modelden next word'ü tahmin etmesini beklemiştik, bir language model eğitirken bir yandan da embeddings'i oluşturmuştuk.

Fakat embeddings'I öğrenmek için tek yol bir language model oluşturmak değil. Language model için target word'ü tahmin etmemizi sağlayan context last 4 words idi.



Ancak bizim amacımız sadece embedding eğitmek ise, sadece son 4 kelime yerine target'den sonraki 4 kelimeye de bakabiliriz ve modelden 8 kelimeyi input ederek target'I tahmin etmesini bekleyebiliriz, böyle bir modeli language model olarak kullanamayız ancak bu modelin tahmin işlemin yerine getirmesi daha muhtemeldir, çünkü elinde daha çok veri var.

Bu sayede de aslında biraz önceki yaklaşımdan avantajlı bir konumdayız çünkü şuanda modelimizin tahmin yapan FC layer'I eğitmesi görece daha kolay, artık embeddings'i eğitmek için daha fazla enerji harcayabilir. Önceki örnekte model son 4 kelimeye bakarak zaten sıradaki kelimeye büyük ihtimalle tahmin edemeyecek, e yanlış tahmin yapan bir modelin embeddings'i de çok iyi olamaz. Ancak bu modelde model daha başarılı tahminler yapabilir elinde verisi var, o veriyi de embeddings'I de daha iyileştirmek için de kullanacak.

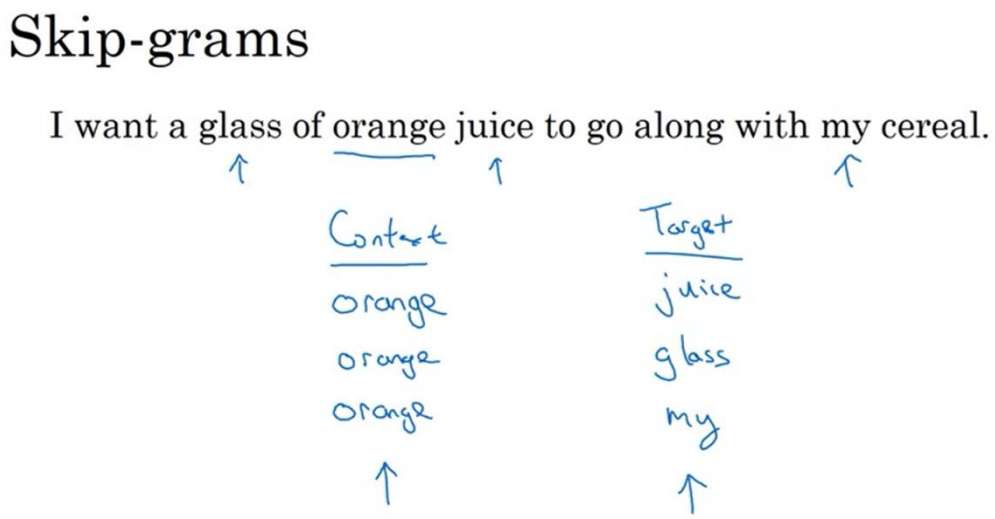
Buna alternatif olarak sadece son kelimeye bakarak, veya target'ın etrafındaki tek bir kelimeye bakarak tahmin eden bir model ile embedding matrix oluşturmaya çalışmak da işe yarayacaktır.

Bunların detaylarını Word2Vec kısmında açıklayacağız.

Week 2 – Word2Vec

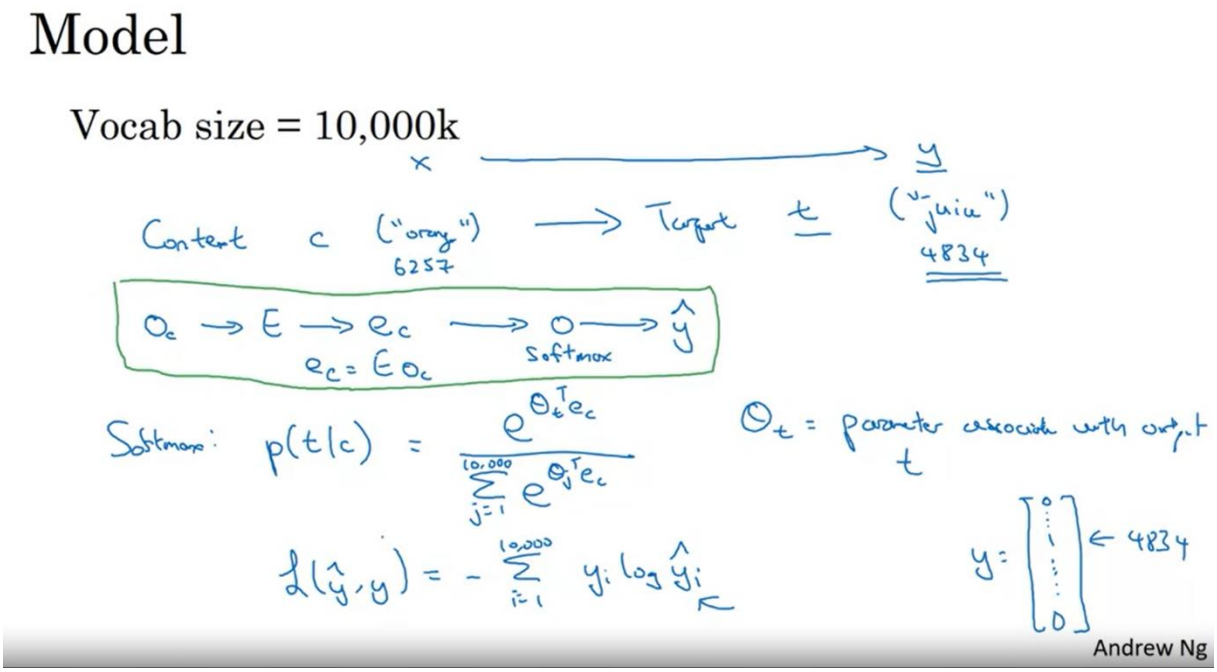
Şimdi language model eğitmeye gerek kalmadan embeddings'i öğrenmek için kullanılan basit, computationally efficient ve etkili bir algoritma olan Word2Vec'I anlayalım.

Olay şu ben bir corpusu alırım ve oradan context-target pairs elde ederim, bu pairs'i elde etmek için rastegele seçilen bir context word ile o context word'e belirli bir range'de bulunan herhangi random bir başka target word'ü seçerim.



Bu durumda yukarıdaki cümle için bazı context-target pairs yukarıdaki gibi olabilir. Elbette ki böyle bir dataset için modelimizin target'ı doğru bilmesi pek mümkün olmayacak yani modele bir kelime veriyoruz ve cümlede buna yakın olan kelime hangisi olabilir bunu söylemesini istiyoruz, kesinlikle modelin hata payı yüksek olacaktır. Ancak modeli bu şekilde eğiterek aslında çok başarılı bir embedding matrix oluşturabiliyoruz yani model hangi kelimeri nasıl encode edersem bu task'de daha başarılı olurum onu anlıyor. Yani orange ile juice'ı apple ile juice'I glass ile wine'ı etc pair target olarak öğrendikçe model bunların arasında bağlantılar olduğunu keşfediyor.

Algoritmayı anladık biraz daha detaylı bakalım, diyelim ki 10k vocab size'ımız var pairs oluşturulmuş, datasetim hazır. Bir context word mesela orange alınıyor, önce bunu O6257 ile çarparak veya gerçekte direk bir lookup function ile embedding vector'ü elde ederiz, bu embeddig vector bir softmax unit'e beslenir ve sonuçta softmax'in 10k'lık çıktısında O4384'ü görmek isteriz ki bu da target word juice'I temsil ediyor.



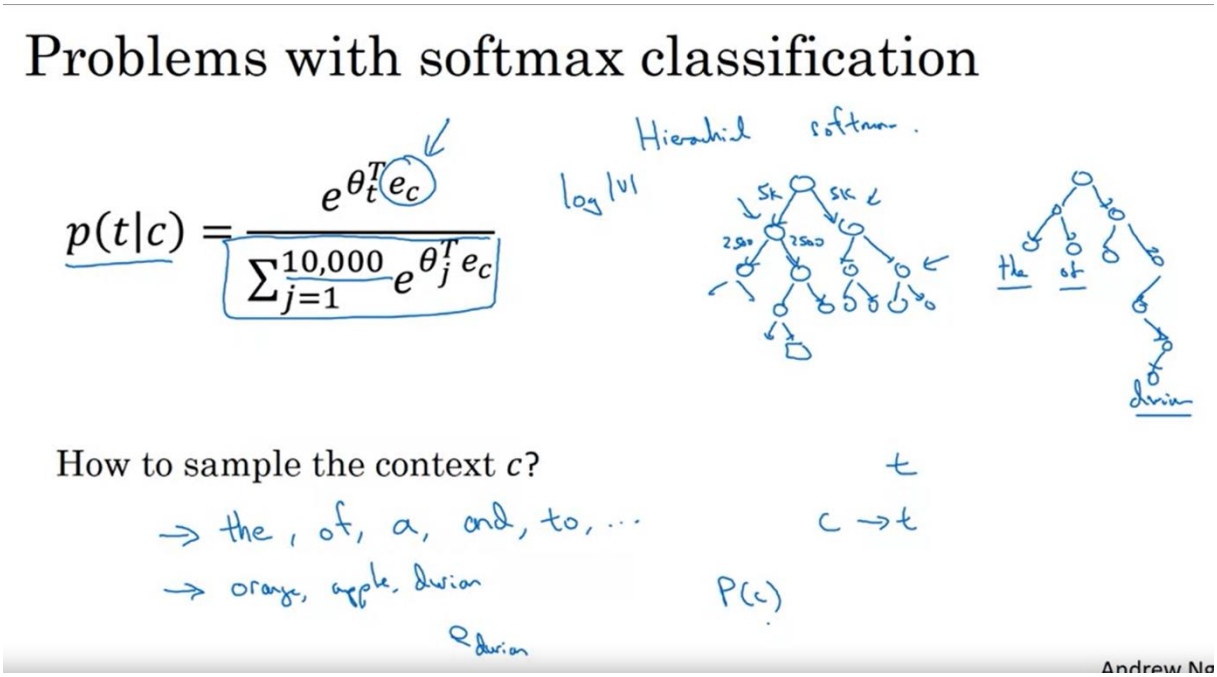
Bu modele Skip Grams deniyor, ve bu sayede güzel embeddings öğrenebiliriz, ancak softmax'in hesaplaması computationally expensive oluyor. Softmax layer'da 10 k unit var her biri gelen embedding vector'ile bir weight'I çarpıp topluyor daha sonra da exponential'ini alıyor. Sonra tüm exponentiel alınmış değerer tek tek toplam değere bölünüyor ki toplamları 1 eden bir probability dağılım output'u elde edebilelim.

Bu hesaplama çok pahalı oluyor bunu daha iyi bir hale getirebiliriz.

Bu pahalı hesabın asıl sebebi her tahmin için 10k adet exponensiyel outputu toplamamızın gerekmesi ve daha sonra tek tek her outputu bu toplama bölmemiz gerekmesi.

Bu problemi çözmek için hierarchial softmax kullanılabilir, bunun mantığı şu target'I tahmin etmek için model 10k'lık tek bir layer kullanmak yerine, ardarda gelen daha derin binary classifier'lardan oluşabilir. Örneğin ilk binary classifier target'in 0-5k mı yoksa 5k-10k aralığında mı olduğuna karar verir. Sıradaki range'I daha da küçültür vesaire en nihayetinde bir unit target'ın 5782. sırada olduğunu output edebilir.

Bu şekilde işleri hızlandırabiliriz ama daha sıradaki başlıkta göreceğimiz negative sampling concept'i bize daha iyi bir sonuç sağlayacak.



Son olrak context-target pairs'I nasıl oluşturacağımıza tekrar değinelim, context word'ü seçtikten sonra target'i belirli bir range'de random atabiliriz, ancak context'I nasıl seçeceğiz?

Uniform olarak random seçebiliriz, ama eğer öyle yaparsak the, of, a, and, to gibi yardımcı kelimeler corpusda daha sık göründüğü için bu kelimeler daha çok seçilecektir, bu da iyi bir sonuç vermeyecektir biz sadece bu kelimeler arasında bağlantı kurmak istemiyoruz olabildiğince geniş range'deki kelimeler arasında bağlantı kurmak istiyoruz, bunu gözeterek seçim yapan bazı yöntemler var. Aklında bulunsun.

Şimdi hiearchical softmax'e alternatif olarak hesaplamayı hızlandırabilecek ve skip-grams'ın eksiğini kapatabilecek bir yöntem olan Negative Samplings'den bahsedeceğiz.

Week 2 – Negative Sampling

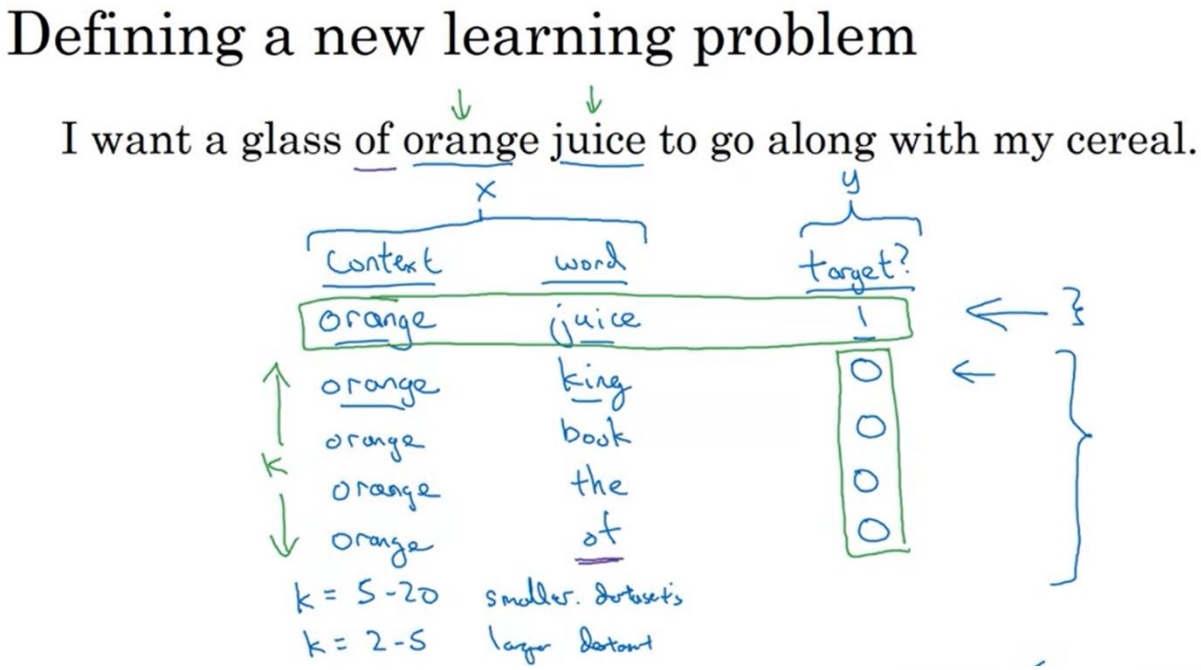
Skip-gram modeli ile bir supervised learning task oluşturduk ve context-target word pairs oluşturarak modeli bu set ile eğitmenin useful word embeddings oluşumunu sağladığını anladık. Bu yaklaşımın problemi softmax unitinin hesaplamasının çok yavaş olmasıydı.

Bu kısımda ise NEGATIVE SAMPLING denilen modified bir learning problem oluşturacağız, skip-gram yaklaşımına benzer olacak ama çok daha verimli bir algoritma ile bunu yapacağız.

Yeni learning algorithm'imiz şöyle olacak: Artık sadece bir context-target pair oluşturmak yerine, positif ve negatif pairs örnekleri oluşturacağım, örneğin orange-juice'ın target'i 1 olurken orange-king'in target'ine 0 diyebiliriz. Sonuçta aslında alakalı ve alakasız kelimelerden labeled bir dataset oluşturacağız.

Positive örneği oluştururken skip-gram'deki mantığı kullanabiliriz, şans bazlı bir context word seçip daha sonra pair'ini bu kelimenin çevresinden bir başka kelime olarak seçip bu yeni kelimenin bu kelime ile bağlantılı olduğunu kabul ederiz.

Daha sonra aynı context word için negative examples üretmeye başlarız, bunun için de pair olarak dictionary'den random olarak kelimeler seçilebilir.

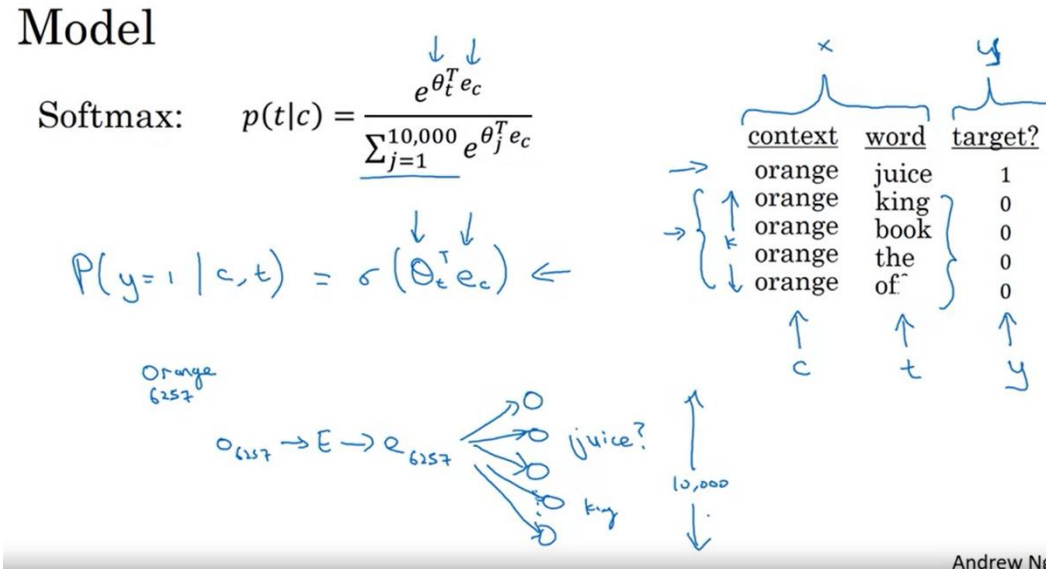


Özetle dataset'I şöyle oluştururuz: önce ratgele bir context word seçilir daha sonra bu context word'ün komşularından biri pair olarak seçilir ve positive example seçilmiş olur. Daha sonra her bir positive example için aynı context

word'e karşılık k tane alakasız random word seçilir bu kelimeler dictionary'den rastgele seçilir ve bu rastgele alakasız kelimelerin aslında alakalı olma olasılığı da vardır mesela "of" kelimesi alakasız olması niyeti ile rastgele seçilmişti ve şans eseri cümlede orange'ın tam solunda yer alır bu olabilir.

Bu yaklaşımla artık X ve Y'lerimi yani dataset'imi ve labels'I oluşturdum artık elimde bir supervised learning problemim var modelin iki kelimeye bakıp bunların alakalı mı alakasız mı olduğunu öğrenmesini istiyorum.

Bu adımdan sonra modeli oluştururken benim beklediğim şey, iki kelimenin input olarak modele verilmesi ve tek bir sigmoid unit ile bu iki kelimenin alakalı mı alakasız mı olduğuna karar verilmesi idi böylece hesaplama işi çok rahatlayacaktı ama sanırım bundan daha iyi çalışan bir yöntem var:



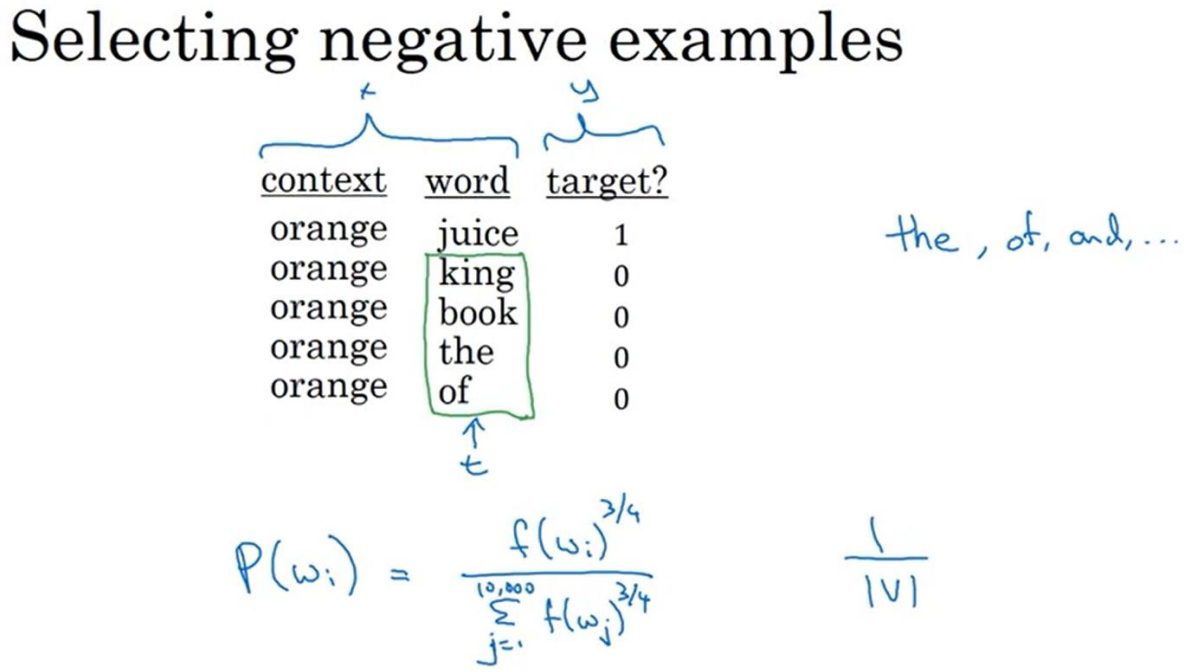
Her bir context word embedding'e çevrilip 10k sigmoid unitli bir layer'a veriliyor. Yani bu softmax layer'a benziyor ama sonucu hesaplamak için her bir output'da tüm unitlerin outputu toplanıp tek tek bölünmüyor.

Bu 10k unitli yani dictionary size ile eşit olan layer'daki her bir sigmoid unit'i tek bir kelime için bir logistic regression unit olarak düşün yani ben orange kelimesini verdiğimde her bir sigmoid unit is orange related to juice? Is orange related to king? gibi 10k farklı kelime için cevap arıyor.

Ayrıca her iterasyonda tüm 10k unit train edilmez sadece k+1 tanesini yani yukarıdaki örnekte sadece 5 tanesini train ederiz is orange related to juice? Is orange related t king? is orange related t book? … bunları soran 5 unit'in outputu alınır, cost hesaplanır ve gradient step atılır.

Sonuçta bu yöntemle 10k binary classifier kullanarak ucuz hesaplama sağlıyoruz, ayrıca her iterasyonda 10k unitten sadece k+1 tanesini train ediyoruz. Bu yaklaşımla çok düşük computational costs ile useful embedding matrixes oluşturabiliriz.

Bu konuyu kapatmadan önce son bir soru soralım how do we choose negative examples?



Context word orange'I ve positive example juice'I seçtik diyelim negative examples'ı dictionary'den nasıl seçeceğiz?

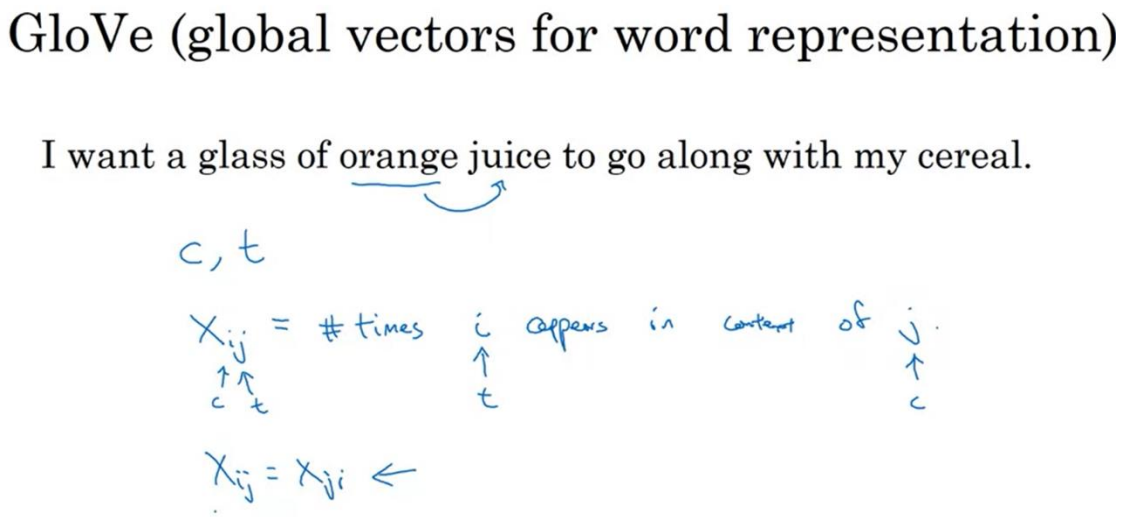
Empirical frequencies'e göre seçebiliriz, yani sık çıkan kelimelere ağırlık veririm, ancak bu durumda the,of, and gibi kelimeler sıklıkla seçilmiş olur bunu istemeyiz.

Uniformly random olarak seçsek yine aynı problem.

Bunun için yazarlar empiric olarak yukarıdaki gibi bir sampling bulmuşlar ve iyi çalıştığına karar vermişler.

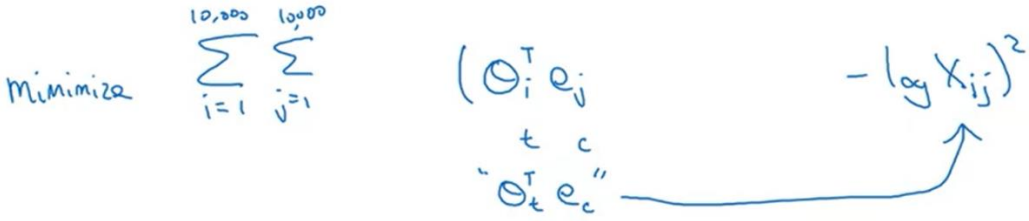
Week 2 – GloVe Word Vectors

Şimdiye kadar gördüklerimizden daha basit bir embedding learning algorithm'i hızlıca anlayalım.



Xij'in temsil ettiği şey i ve j kelimelerinin corpus içerisinde hangi sıklıkla komşu olarak bulundukları. Komşuluğu şöyle tanımlayabiliriz mesela +-10 kelime range'de bulunuyorlarsa bunu sayarız. Yani Xij=Xji=100 ise bu demekki bu iki kelime corpus içerisinde 100 farklı yerde komşu olarak yakın yerlerde bulunmuş.

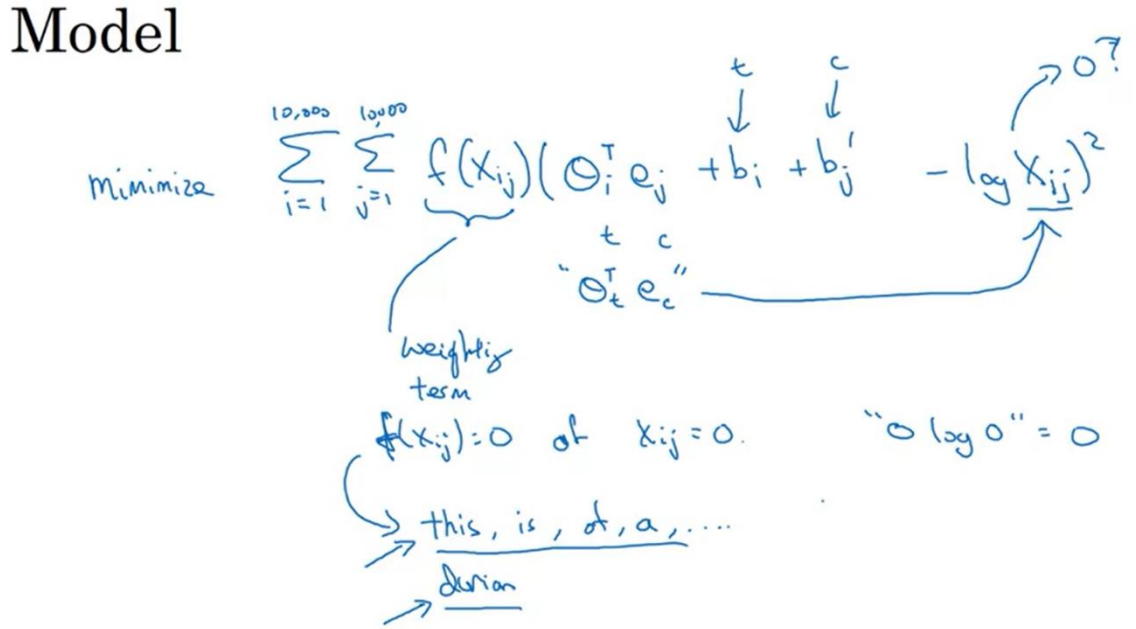
GloVe ile yapılacak şey ise aşağıdaki denklemi minimize etmeye çalışıyoruz, denklemin kabası şu:



Burada Q*e term'ini şöyle düşün negative sampling modelindeki sondaki her bir sigmoid unitin içi gibi düşün. Yani bir dictionary word için bir Qi weight'I var aynı şekilde her bir dictionary için bir embedding e var. Bu ikisinin inner product'ından iki kelimenin görülme sıklığını çıkarıyoruz.

Yani amacımız şu her kelime için öyle bir Q ve e vektörleri bulalım ki her kelime için bu vektörler çarpıldığında bize kelimelerin ne kadar alakalı olduğunu yani kaç kez yanyana görüldüğünü tahmin etsinler ki böylece cost'un içi 0'a yaklaşır ve sonuçta cost'u minimize etmiş oluruz.

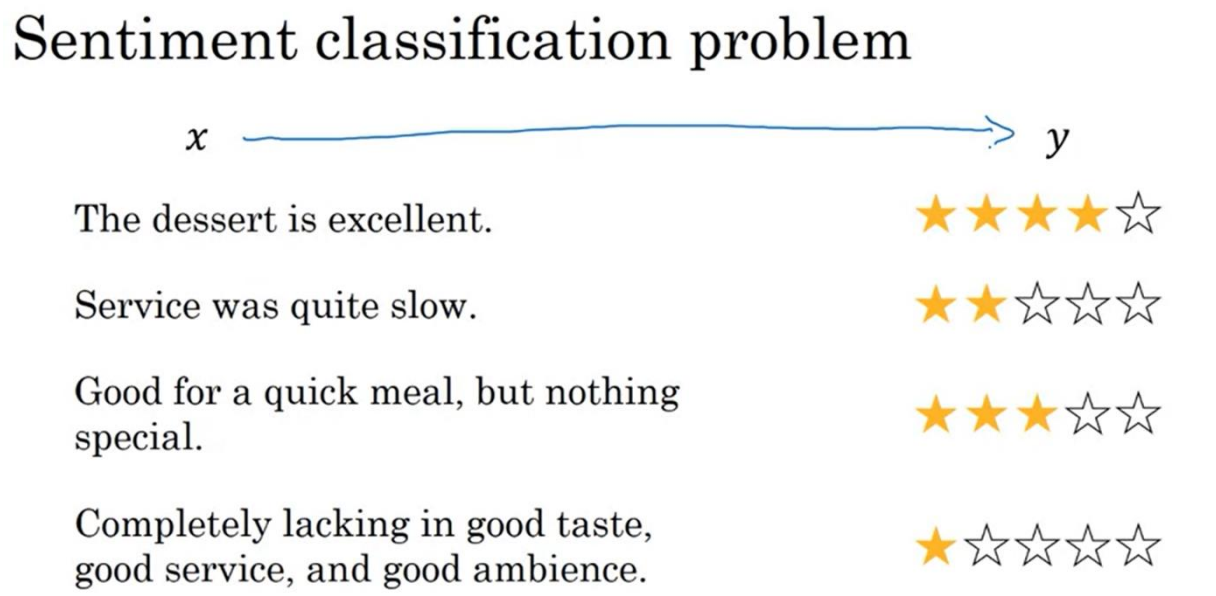
Bu sayede de gerekli embeddings'I öğrenmiş oluruz. Bu denkleme ek olarak başa bir weighting term eklenir bu logXij'in -sonsuz yani Xij'nin 0 olduğu durumda term'i sum'a dahil etmemek için kullanılır. Ayrıca sık çıkan kelimelerle az çıkan kelimeler arasında bir denge sağlamak için de kullanılır.



Burada Qi ve Ej 'in satırları simetriktir. Ayrıca Q ile e aslında aynı işlevi yapan vectorler olacak yani ikisi de kelimeyi temsil eden embeddings olarak kullanılabilir bu yüzden, bazı yaklaşıklarda final embedding için Qw ile ew'nin ortalaması alınarak w kelimesinin embedding'i olarak kullanılabilir.

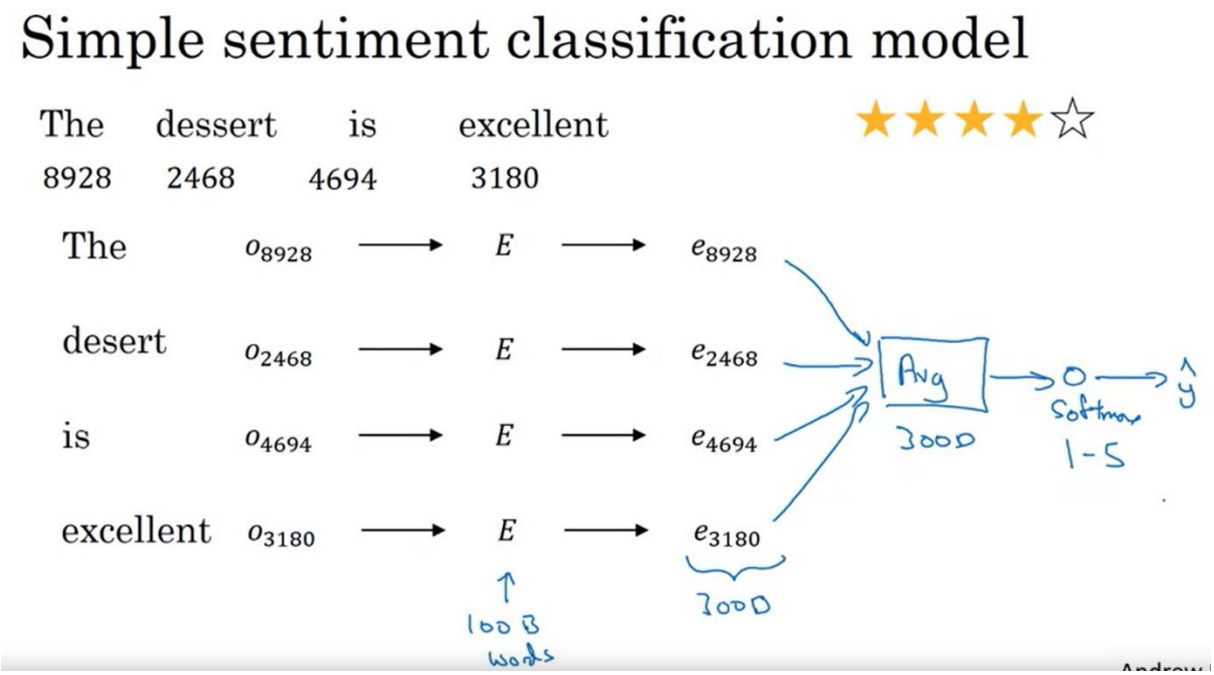
Week 2 – Application – Sentiment Classification

Örneğin bir yoruma bakarak kişinin bu şeyi sevip sevmediğini anlamaya çalışıyorsak bir sentiment classificiation problemi ile uğraşıyoruz demektir.



Cümlelerden kullanıcının kaç star verdiğini anlamaya çalışıyor olalım.

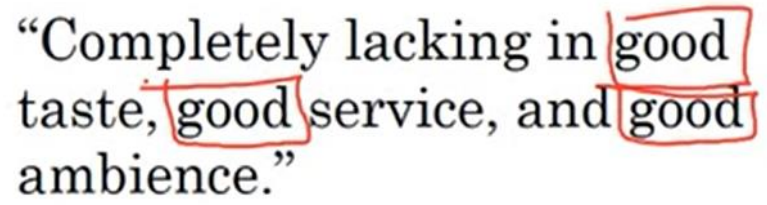
Hazır embedding matrix'ten yararlanarak basit bir model eğitebiliriz:



Örneğin yukarıdaki X cümlesi alınır, 100B word'lük pre-trained embedding'den yararlanarak eğitim setinde unknown kelime kalmadan modele sokabilirim. Sonuçta embedding vectors elde edildikten sonra avg alınarak modele sokuluyor ve softmax ile 1-5 arasında bir classification yapılıyor.

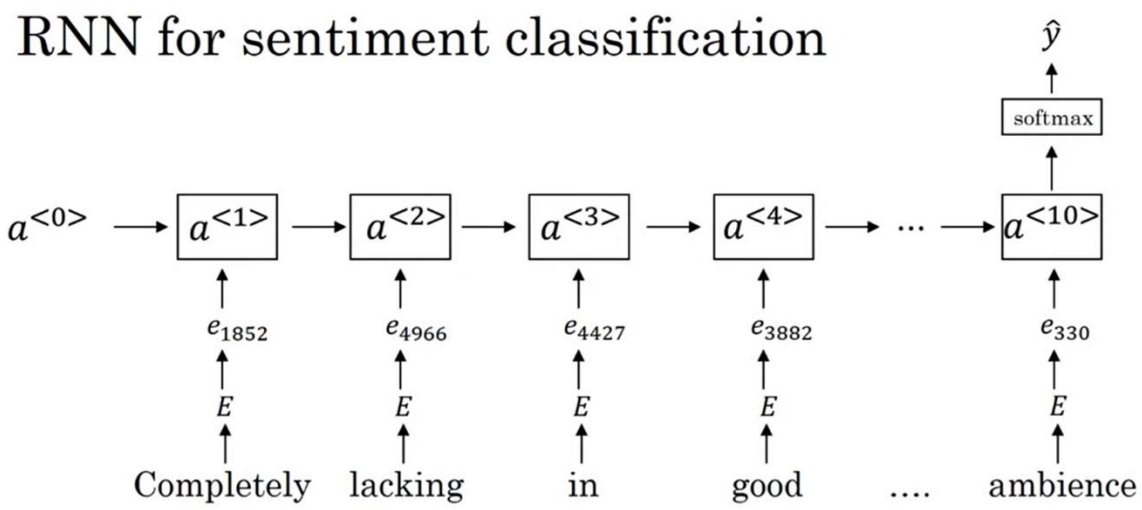
Burada average alınması bana tuhaf geldi bunun avantajı şu cümlenin uzunluğu ne olursa olsun bu NN modeli işe yarayacaktır. Ancak burada kelimelerin sırası vesaire bunlara dikkat edilmiyor sadece hangi kelimelerin var olduğu dikkate alınmış oluyor.

Sonuçta bu basit algoritma genelde iyi çalışır ancak aşağıdaki gibi bir review olduğunu düşünelim:



Burada model kelimelerin embeddings'lerinin ortalamasını veya sum'ını alacağı için ve review'de positive kelimeler sık göründüğü için bu review için model büyük ihtimalle pozitif diyecek ve aslında cümle yapısını, orderını gözardı etmiş olacak.

Bu problemi çözmek için RNN kullanabiliriz:



Sonuçta kelimelerin embeddings'I RNN'e beslenecek ve cümle bitince softmax ile rating tahmini yapılacak böylece, cümlenin order'ını daha iyi hesaba katabiliyoruz. Bunun yanında yine RNN kullandığımız için farklı uzunluklardaki inputs için modelimiz çalışacaktır.

Böylece words sequence'i hesaba katmış olacağız ve "not good" gibi statement'ları anlayabilecek bir model oluşturmuş oluruz.

Week 2 – Application – Debiasing Word Embeddings

Training için kullanılan text corpus'u sonuçta biased word embeddings öğrenilmesine neden olabilir. Çünkü bu text'ler içinde genelde bias içerir. Örneğin embedding ile daha önce yaptığımız analoji örneğini (man→king ise women:queen) hatırlayalım.

Böyle bir analojiyi man:computer_programmer women:? Veya father:doctor mother:? Olarak kurarsak modelimizin homemaker veya nurse gibi gender inequality ile biased olduğunu görebiliriz.

Buna benzer olarak age, ethnicity, age biases da söz konusu olabilir. Bu kısımda bunlardan kurtulabilmek için gerekli bazı fikirlerden bahsediliyor.

The problem of bias in word embeddings

Man:Woman as King:Queen

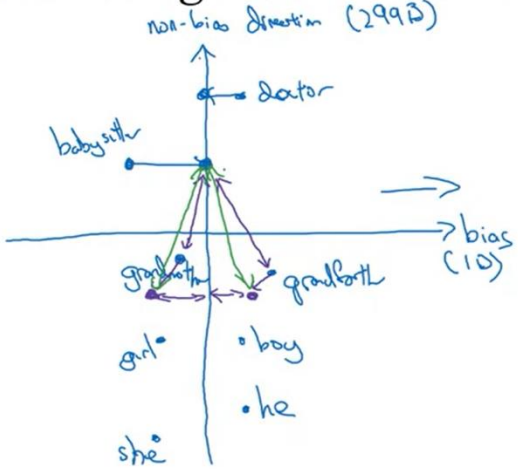
Man:Computer_Programmer as Woman:Homemaker ✗

Father:Doctor as Mother:Nurse ✗

Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model.

Aslında burada olay şu: diyelim ki 300D uzayda embeddings aşağıdaki gibi pozisyon almış. Burada he-she, male-female gibi gender defitinitive word embeddings arasındaki farkın ortalamasını bularak bias direction'I bulabiliriz.

Addressing bias in word embeddings



1. Identify bias direction.

$e_{he} - e_{she}$

$e_{male} - e_{female}$

average

2. Neutralize: For every word that is not definitional, project to get rid of bias.

3. Equalize pairs.

grandmother — grandfather

girl boy

Yani bu direction erkek ile kadın arasındaki farkın direction'ıdır deriz bu 1D de olabilir daha fazla da olabilir, 1D kabul edelim o halde kalan 299D non-bias direction oalrak düşünülebilir.

Bu noktada olay doctor, computer programmer etc gibi aslında gender ile bağlantısız olması gereken kelimeleri iki gender'ın ortasına denk getirmektir, yani neutral space' project etmektir. yani bias var ise yukarıdaki gibi doctorbias direction'a göre erkek kavramına daha yakındır, ancak biz bundan kurtuluruz ve bias direction'a göre bir projection ile ortalama yaparız.

Benzer şekilde babysitter hem grandfather hem de grandmother olabilir bu yüzden bunu da ortalarız, yine de grandmother babystting'e yukarıdaki gibi daha yakın duruyor olabilir o halde bunları da linear algebra ile eşit konumlara getiririz.

Yani özetle olay şu bias olabilecek kelimeler'in embedding'leri birtakım linear algebra yöntemleri ile bias'ten arındırılabiliyor. Bu kelimelerin seçimi vesaire ayrı bir problem onu da yine classification algoritmaları ile yapmak mümkün.

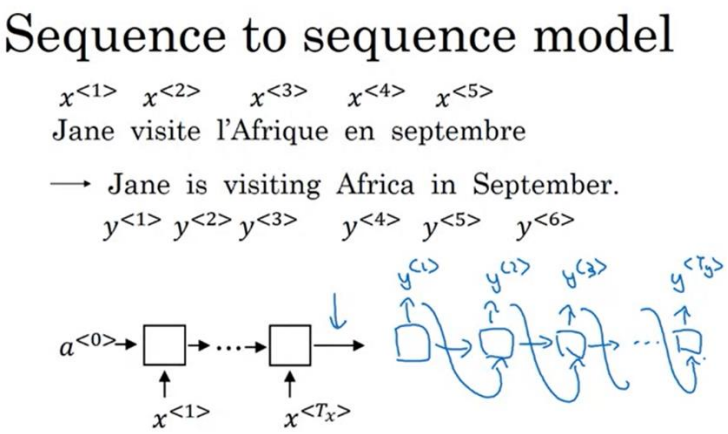
Fikir bu.

WEEK 3

Basic Models

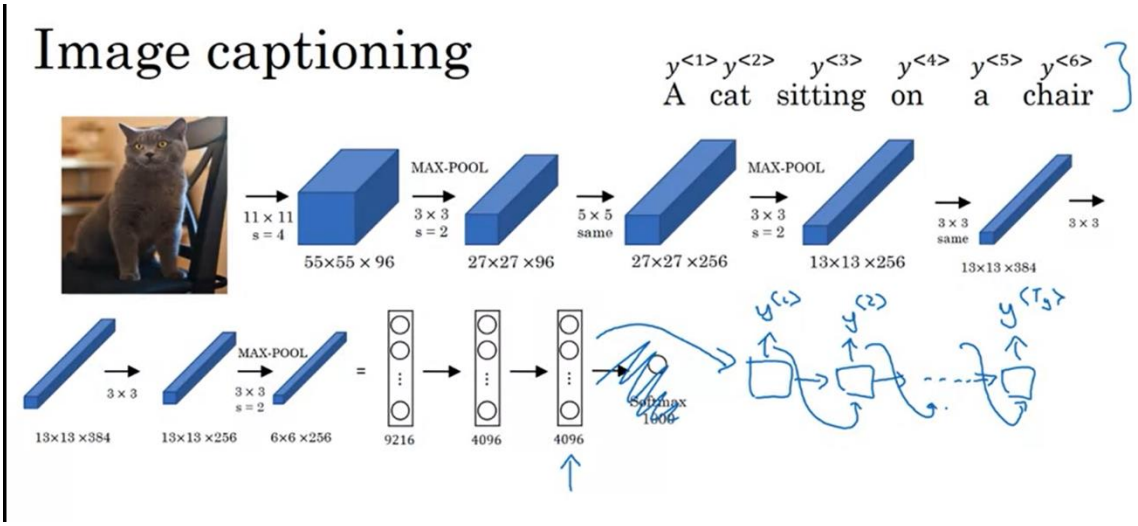
Bu hafta sequence to sequence models'den bahsedilecek. Machine Translation, Speech Recognition gibi taskler için seqeunce to sequence models kullanılır.

Machine translation örneğini ele alalım:



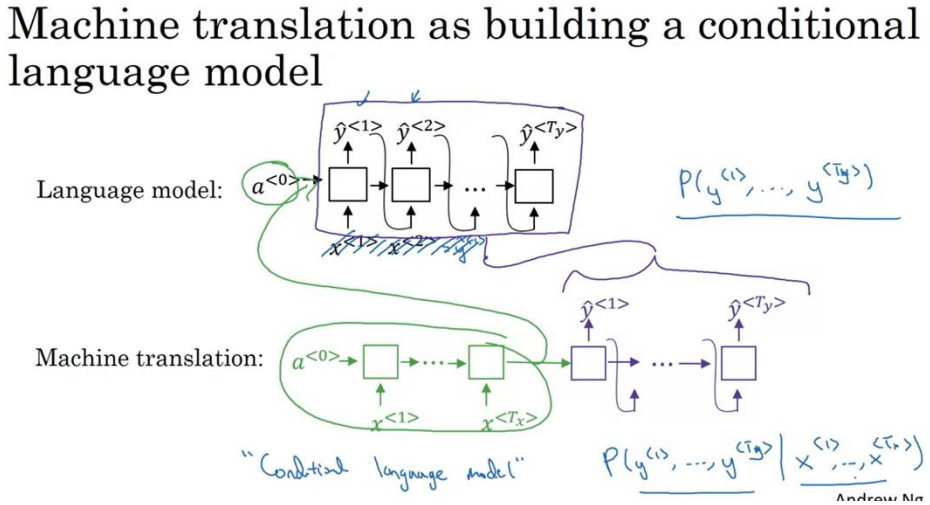
Yukarıdaki fransızca cümleyi ingilizceye çevirmek için yukarıdaki gibi bir RNN/LSTM/GRU yapısı kullanılabilir, ilk kısım da sequence input RNN'e tek tek beslenir ve sonuçta input sentence encode edilir daha sonra bu encode edilen cümle modelin decoder kısmına verilir ve model encoded cümleden bir translation decode eder.

Bir başka problem image captioning, modelimiz image'a bakıp gördüğü şeyi açıklayan bir cümle üretsin isteyebiliriz. Bu durumda aşağıdaki yaklaşım işe yarar pretrained bir CNN'I encoder olarak kullanırız ve encoded image'ı bir RNN'e besleriz, böylece umarız ki model görüdüğünü anlatabilsin.



Picking the Most Likely Sentence

Burada language model ile Machine Translation arasındaki benzerlikten bahsediliyor. Language model ile verilen bir input için bir sonraki en olası kelimeyi bulabiliyorduk. Aynı mantıklı verilen bir cümlenin occurrence probability'sini de bulabiliyorduk.



Aslında machine translation'ın ikinci kısmı language model ile aynı ancak a0 gibi bir zero vector ile işe başlamak yerine başka dildeki bir cümlenin encoded hali ile işe başlıyor ve most probable cümleyi üretiyor.

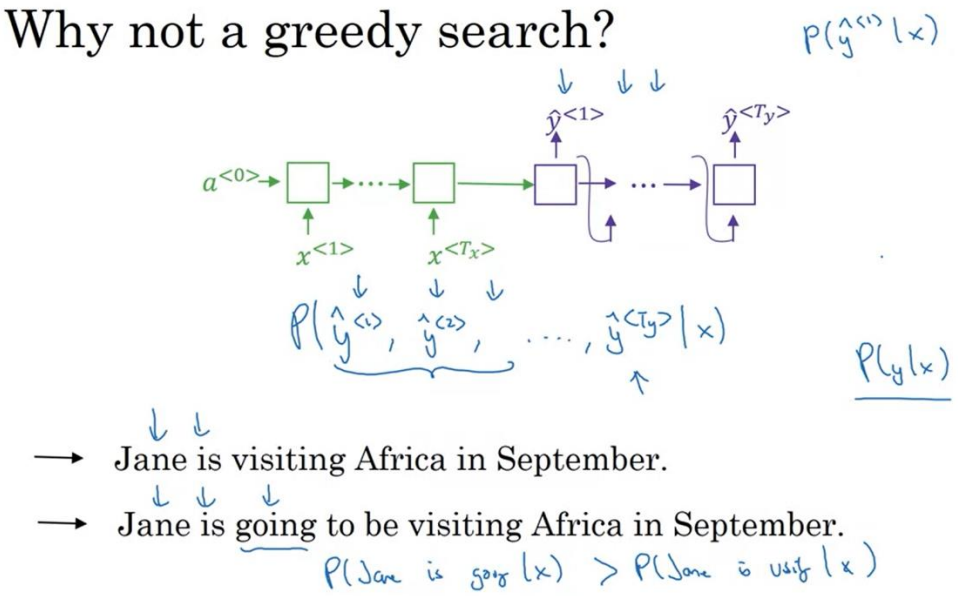
Yani fransızca'dan encode edilmiş cümleyi a0 olarak alıyor ve sonucunda ingilizcedeki most probable cümleyi üretiyor. Bunu da Andrew "conditional language model" olarak isimlendirmiş.

Çünkü fransızcadaki şu cümle gerçekleştiğinde, ingilizcedeki en olası cümle nedir gibi bir output elde ediyoruz. Bu da conditional probability gibi düşünülebilir.

Şimdi olay şu, andrew diyor ki burada model fransızca cümlenine encoded bilgisini aldıktan sonra max probability'li english sentence'I üretmesi gerekiyor ancak bunu daha önceden öğrendiğimiz gibi greedy search ile yapmak kötü sonuçlar veriyor.

Greedy search ile yapılan şey şuydu kelimeleri tek tek seçmek, yani önce encoded sentence için most probable first english word seçiliyor, daha sonra bu english word için most probable second word seçiliyor, so on. Ancak işte böye seçim yapmak hatalı sonuçlar verebiliyor.

Örneğin Jane is visiting Africe yerine Jane is going to be visiting Africe diyebiliyor çünkü Jane is'den sonra model en çok going kelimesini duymuş, hemen going'i yapıştırmak istiyor.



Bu yüzden bizim burada decoder kısmında daha farklı bir yol izlememiz gerek. İlgili french sentence için most probable english words'ü seçmeliyiz.

Bu amaçla BEAM SEARCH kullanacağız.

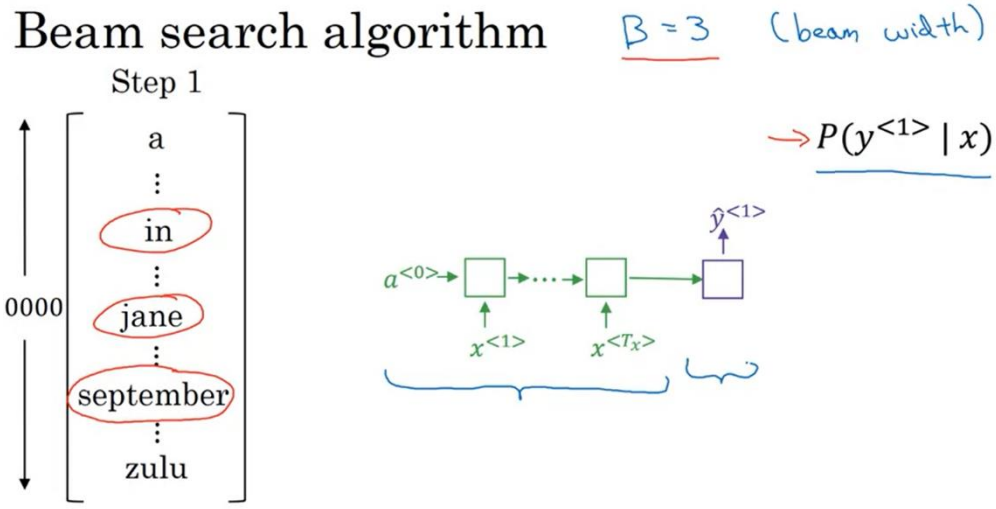
BEAM Search

Amacımız random bir french sentence için most likely english translation'ı elde etmek. Aynı mantık speech recognition problemi için de geçerli.

Bu noktada beam search algoritması çokça kullanılıyor.

Anladığım kadarıyla olay şu, BEAM search ile ilk önce encoded output alınıyor ve buna bakılarak language model most probable B word seçiyor. B burada BEAM SIZE denilen bir parametre mesela B=3 diyelim.

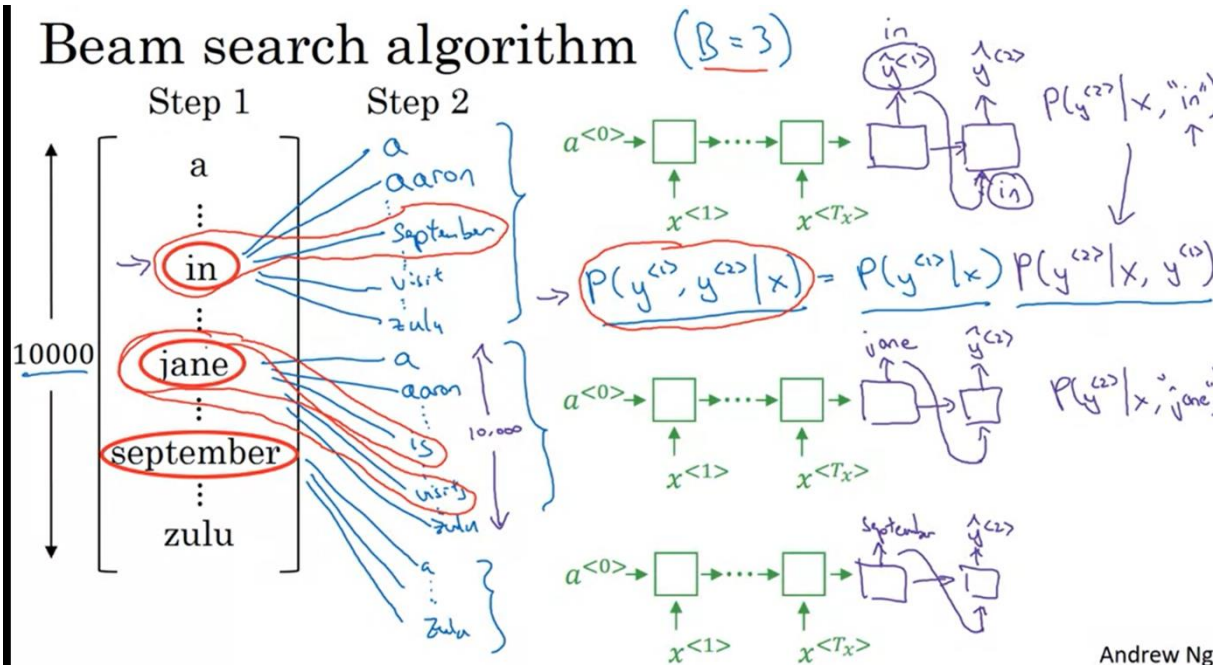
Bu durumda model ilgili french sentence'in encoded datasını alıyor ve most probable 3 english world'ü seçiyor, greedy search ile language model sadece most probable'I alıyordu.



Yani yukarıdaki gibi yeşil encoded modelden alınan çıktı, mor decoder'a veriliyor bu model de en sonunda dictionary size kadar outputlu bir softmax layer kullanılarak mesela 10k output veriyor, buradan most probable 1 yerine most probable 3 word seçiliyor.

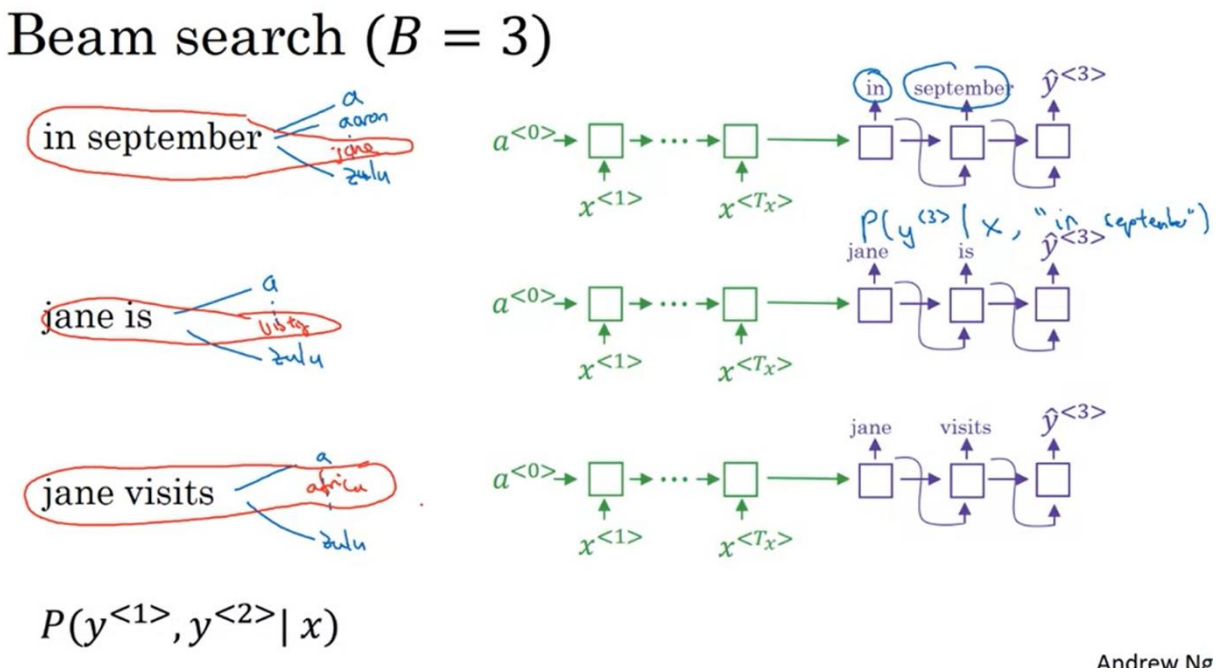
Şimdi ise olay şöyle devam edecek seçilen her üç kelime için most probable next word'ü bulacağız. Yani mesela ilk kelime için "in" seçilirse next word ne olması sorusunun cevabına bakıyoruz, bu kez decoder'a encoded veri ve "in" kelimesi veriliyor ve most probable next word bulunuyor.

Bu noktada aslında verilen x yani encoded data için most probable 2 word pair'I bulmuş oluyoruz. Greedy search ile şunu bulurduk verilen x için most probable first word, sonra verilen x ve first word için second word şeklinde. Ama buradaki yaklaşım, verilen x için most probable pairs of word'ü yani overall da verilen x için most probable sentence'i bulmuş oluyoruz.



Sonuçta ilk etapta x için most probable B word seçilmişti, şimdi her word için most probable second word seçildi. Bu şekilde şunu hesaba katıyoruz, mesela mesela greedy search ile ilk word "in" in olasılığı yüksek olabilir, direkt o seçilecek ama belki 2 word pairs olarak bakınca "Jane is", "in september" dan daha olası olabilir bu yüzden, beam search bizim bunu yakalamamıza olanak sağlıyor.

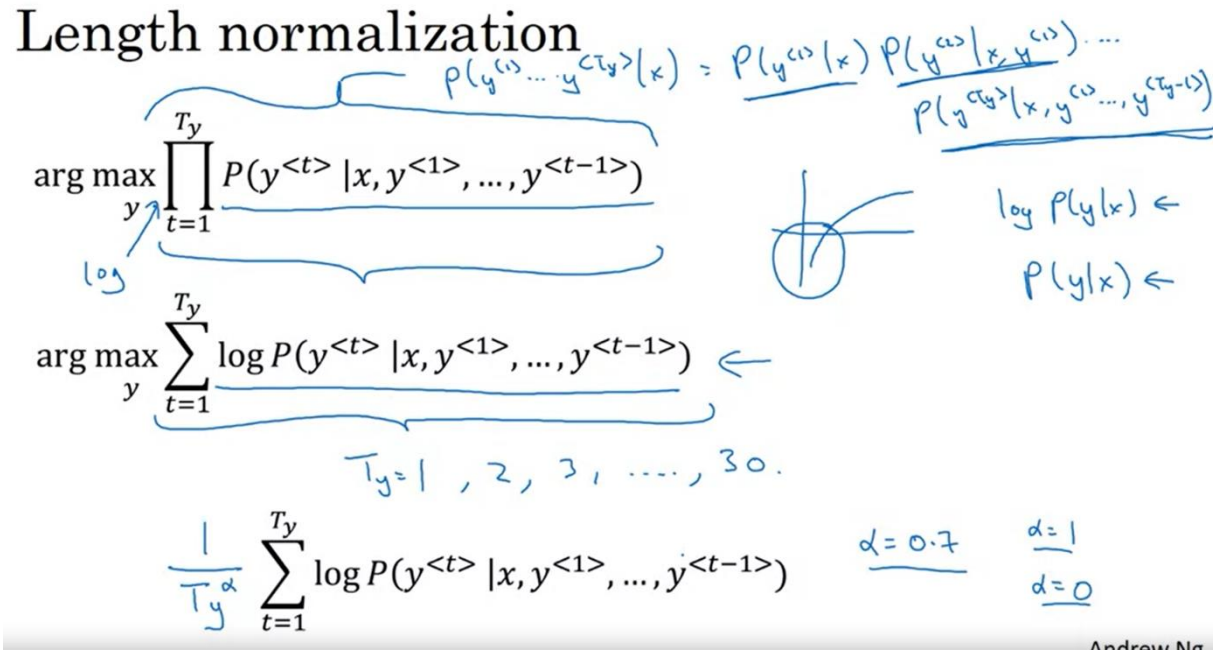
Bu şekilde B tane ki bu durumda B=3 en iyi 2 word pair seçildi, şimdi 3. Kelimeleri bulacağız, mantık aynı, bu kez de her B durum için word pairs'i ve x'I verip modelden most probable 3. Word'ü bekleriz. Sonuçta elimizde B adet most probable 3 words olur.



Ancak farkındaysan burada ilk durumdan sonrakilerde tekrar B'ye bölünme olmuyor, bence mesela in kelimesinden sonra da most probable B word'ü bulsa yani ilk kelime için 3 ihtimal, ilk 2 kelimelik pair için 9 ihtimal, ilk 3 kelimelik pair için 27 ihtimal şeklinde gitse çok daha kapsamlı bir sonuç elde ederiz, ancak muhtelemen bu büyük computation zorlukları çıkarıyor.

Refinements BEAM Search

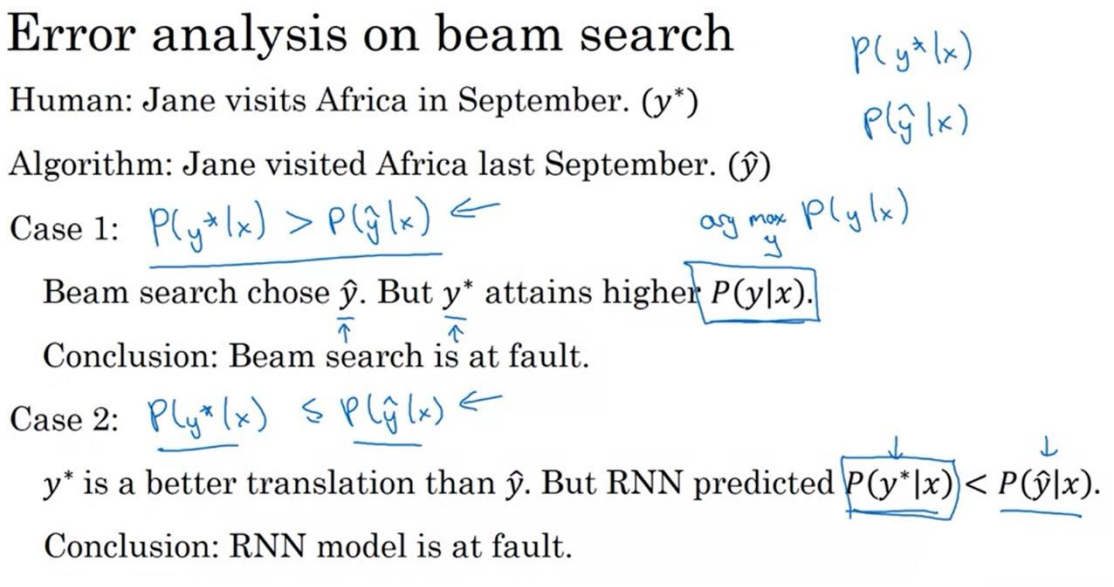
Önceki kısımda basic BEAM Search algoritmasını gördük, bunun üzerinde yapılacak bazı değişikliklerle daha iyi sonuçlar elde edebiliriz.



Unlike exact search algorithms like BFS (Breadth First Search) or DFS (Depth First Search), Beam Search runs faster but is not guaranteed to find exact maximum for $\arg\max_y P(y|x)$.

Error Analysis Process

Beam search kullanarak machine translation yapıyoruz diyelim. Fakat modelimiz istediğimiz gibi çeviri yapmıyor. Örneğin fransızca cümlenin gerçek çevirisi aşağıdaki y* iken modelimiz y_hat şeklinde çeviriyor.



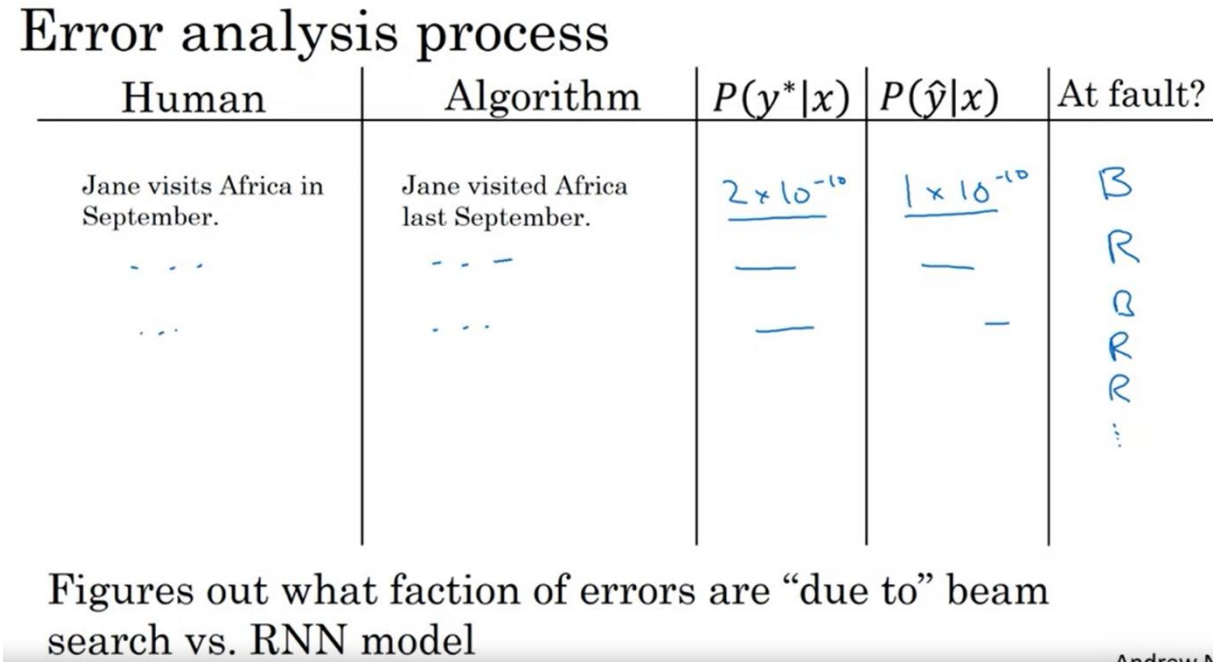
Bunun sebebi, beam search'ün ilgili y*'i bulamamış olması mı yoksa RNN modelimiz yeterince iyi encoding ve decoding yapamıyor bu yüzden hatalar mı yapıyor?

Bunu anlamak için error analysis yaparız ve modelimizin hangisinin probability'sini daha yüksek gördüğünü test ederiz: P(y*|x) mi yoksa P(y_hat>x) mi.

Eeğer case 1 geçerliyse RNN iyi çalışıyor modelimiz aslında doğru kombinasyonun y* olduğunu biliyor demek, ancak beam search yeterince iyi çalışmıyor ve en iyi kombinasyonu veren kelime grubunu bulamadık.

Ancak case 2 durumunda ise RNN zaten y_hat'in y*'dan daha iyi bir translation olduğunu düşünüyor bu durumda modelimizi geliştirmeliyiz modelimizde sıkıntı var demektir.

Aşağıdaki process ile hatalı cümlelerin analizini yapıp, hangi cümlelerin hata sebebinin beam search hangilerinin sbeebinin RNN modeli olduğunu anlayabiliriz.



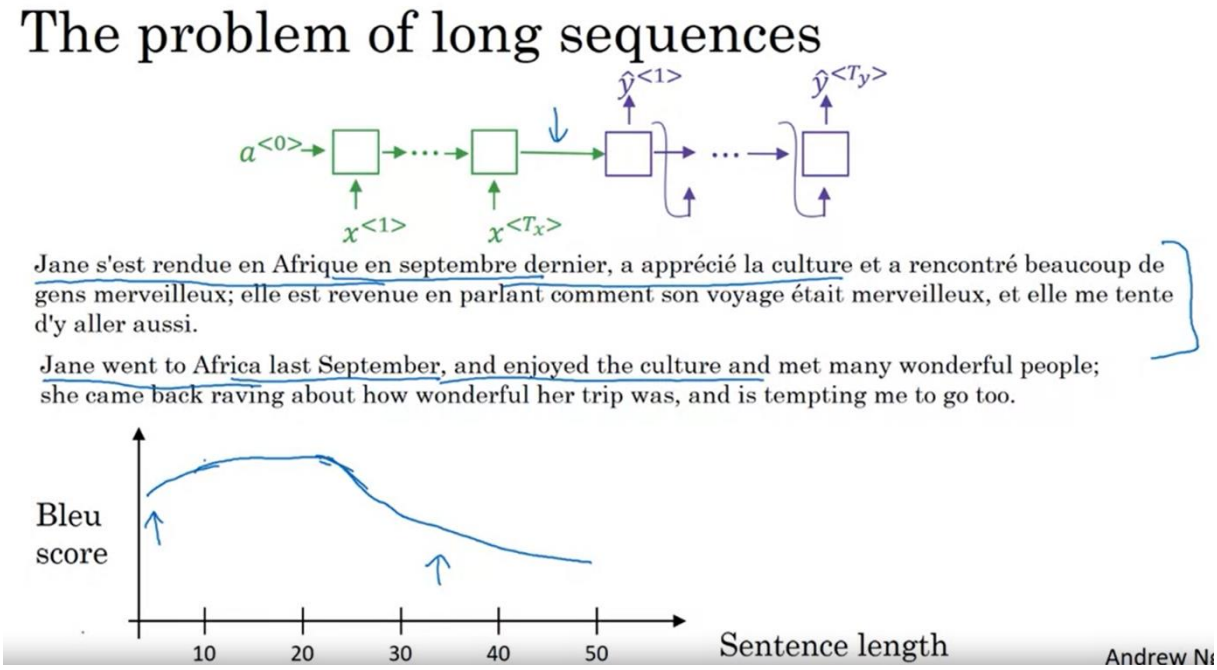
Böylece dev set'in hatalarını sayarak yüzde kaçının beam search'den kaçının ise RNN'den kaynaklandığını anlar ve ona göre geliştirme yaparız.

Attention Model Intuition

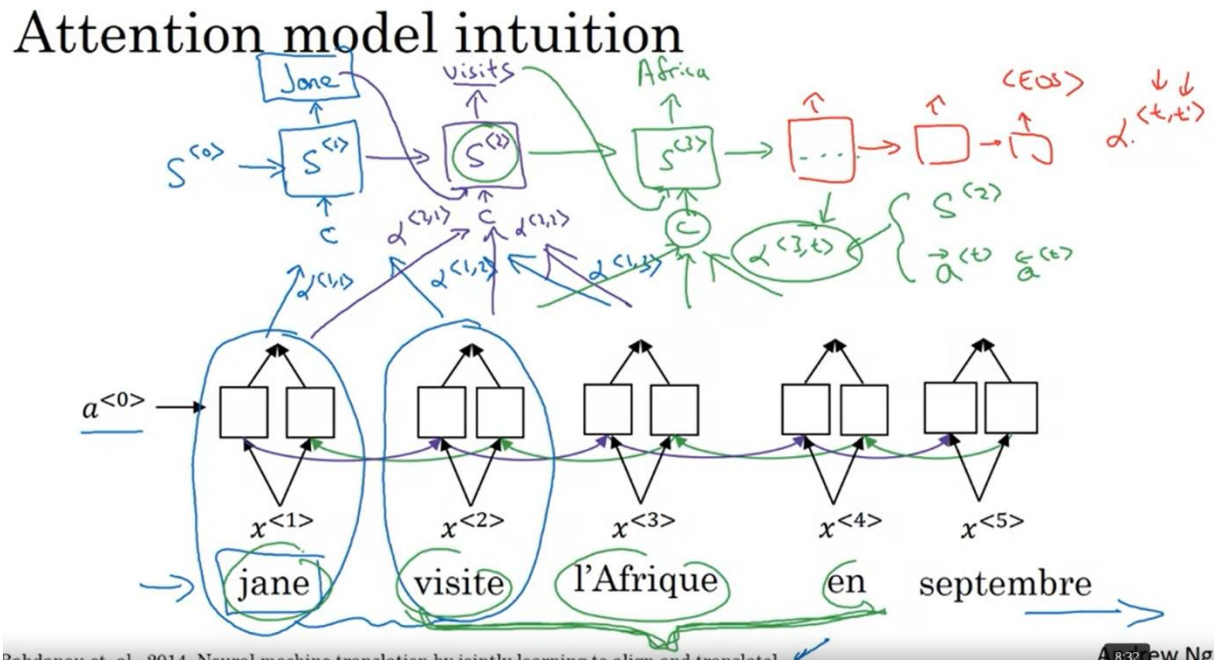
Yukarıdaki kısımlarda 2 RNN yapısı ile encoding ve decoing yapmıştık. Bir tanesi cümleyi okurken diğeri ise translated sentence'i output ediyor.

Bu yapıya bir modifikasyon yaparak Attention Model'I elde ederiz. Böylece daha iyi sonuçlar elde ederiz. Attention fikri deep learning için son derece önemlidir şimdi bunu anlamaya çalışalım.

Modelimizin çalışma mantığı verilen cümlenin tamamını alıp önce encode etmek yani cümlenin tamamını tek seferde activations ile temsil etmek daha sonra da decode etmek ve translation'ı üretmek.



Ancak bu yaklaşım uzun cümlelerde problematic hale geliyor ve kötü performanslar alıyoruz. Bir human translator da uzun cümleleri tek seferde translate etmez onun yerine cümlenin anlamlı küçük parçacıklarını yavaş yavaş translate ederek ilerler.



Kabaca olay şu, fransızca cümle yukarıdaki gibi bi-directional rnn ile encode ediliyor. Decoding sırasında ise kelimeler üretilirken işe attention weights giriyor.

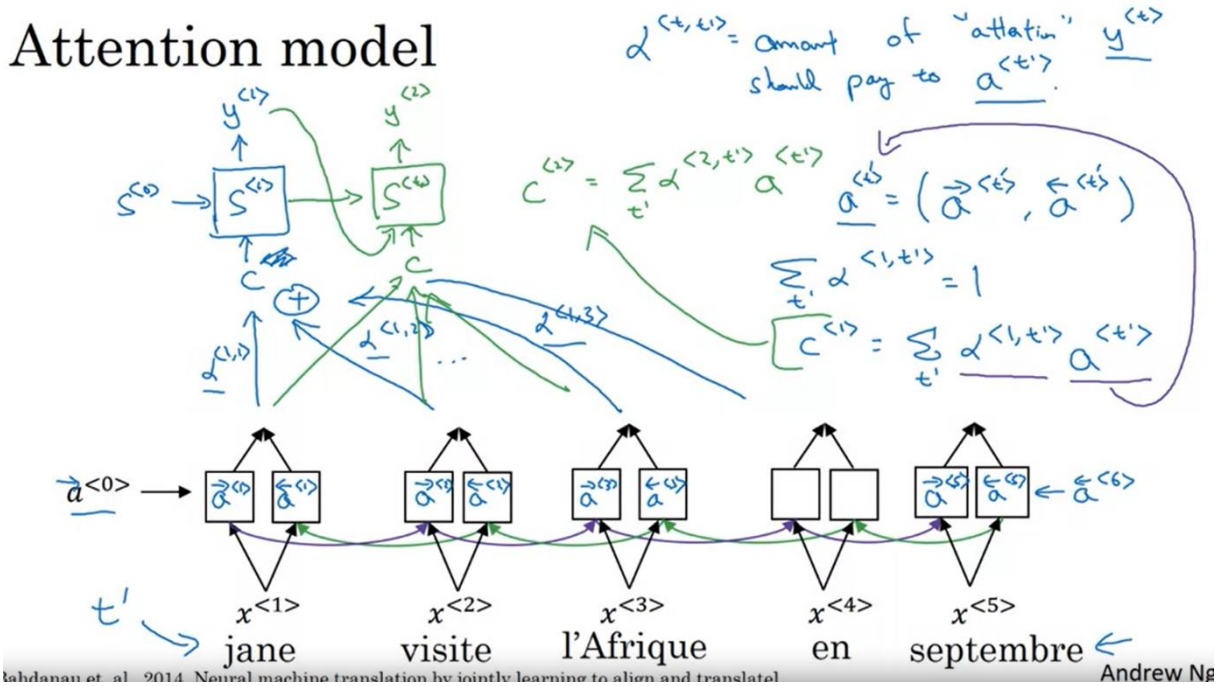
Örneğin s3'ü üretirken model bi-directional RNN'in t için activation'ını alıyor bunun yanında s2'yi alıyor. Bu ikisi modelin ilgili kelimeyi üretirken french cümlenin t. kelimesinin ne kadar dikkate alındığını belirliyor.

Yani attention weights ile cümle translation'I sırasında original cümlenin tamamı yerine bazı kısımları dikkate alınmış oluyor.

Biraz daha detayına inelim:

a<t’> ile herhangi bir t için encoder’ın forward ve backward activationlarını temsil ediyoruz. Decoder modelimiz cümle output ederken, tüm a<t’> aktivasyonlarını kullanmak istemiyor, işte atteintion weights burada devreye giriyor, context ismindeki parametre bu weightler ile aktivasyonların çarpımlarının toplamı olarak tanımlanıyor.

Böylece model translation yaparken her kelime output’u için original sentence’in belirli kısımlarına dikkat ederek çeviriyi yapıyor. İkinci kelime yani y<2> için de bakarsak başka bir context tanımlanıyor bu context için kullanılan attention weights farklı olacak.



Computing attention $\alpha^{<t,t'>}$

$\alpha^{<t,t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$