

COURSE5 – W1

Vid 1

Why Sequence Models?

Sequence models'in nerelerde gerekli olabileceğine bakalım.

Örneğin speech recognition için, input olarak bir audio clip olur, bunu bir text ile map etmemiz istenir. Bu örnekte input da output da sequence data'dır. X is an audio clip plays out over time, and Y is sequence of words.


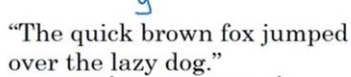




Bir başka örnek, ise music generation burada sadece output sequence data. Input empty set veya genre'yı temsil eden bir integer veya ilk birkaç notayı temsil eden integers olabilir.

Bir diğer örnek ise Sentiment Classification. Bu örnek için X bir sequence data: yani bir düşünceyi temsil eden bir cümle. Output ise bu cümlenin classify edilmiş hali mesela 5 üzerinden kaç puana denk geldiği, bu yüzden output sequence değil.

Bir başka örnek ise DNA sequence analysis, bir DNA sequence verildiğinde bu DNA sequence'in hangi kısmı bir proteine karşılık geliyor bunu bulmamız istenebilir.

Bunun yanında machine translation(language translation), ve video activity recognition(given a video clip identify activity), veya name entity recognition'da(given a sentence identify people) da sequence data gözlemlenir.

Examples of sequence data

Speech recognition		→	
Music generation		→	
Sentiment classification	"There is nothing to like in this movie."	→	
DNA sequence analysis	AGCCCCTGTGAGGAAGTAG	→	AGCCCCTGTGAGGAAGTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger . Andrew Ng

Sonuçta sequence problem'in de bir sürü çeşidi var, kimisinde X ve Y both sequence data iken kimisinde sadece biri sequence data. Bunun yanında kimisinde input ve output length aynı iken kimisinde farklı vesaire.

Vid 2

Notation

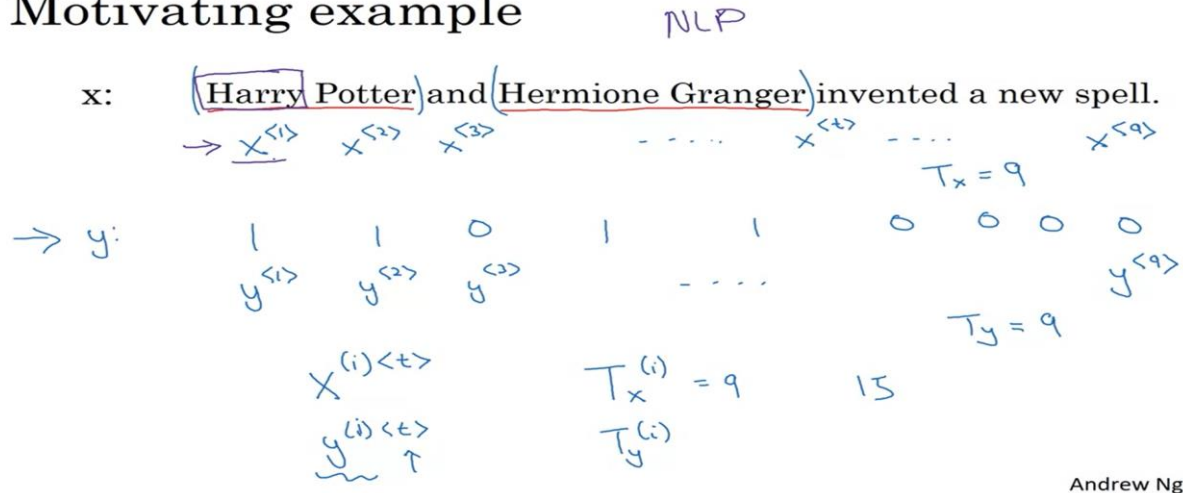
Önceki kısımda sequence models'in kullanılabileceği applications'ı gördük. Bu kısımda bazı notasyonları oluşturalım ki gelecekte bize bir temel oluştursun.

Motivating example olarak şunu kullanalım: Diyelim ki sequence model'e aşağıda görüldüğü gibi bir cümleyi input olarak veriyorum ve modelden bu cümle içindeki kişi isimlerin nerede geçtiğini bulmasını istiyorum. Bu probleme "Name Entity Recognition" denir.

Böyle bir model arama motorlarıyla birlikte kullanılarak, son 24 saatte tüm haberlerde bahsedilen kişilerin isimlerini belirleyebilir.

Ayrıca name entity systems insan isimlerini bulmak için kullanılabileceği gibi, şirket, ülke, zaman, lokasyon, currency isimlerini bulmak için de kullanılabilir.

Motivating example



Diyelim ki yukarıdaki örnek için outputumuz her kelimeye karşılık 0 veya 1 versin böylece eğer ilgili kelime isim ise 1 olarak output verilsin değilse 0 olarak verilsin. Bu yöntem çok iyi bir yöntem sayılmaz daha kompleks outputlardan söz edilebilir ama bu örnek için böyle kabul edelim.

Input'umuz sequence of 9 words. Bu input içindeki her kelimeyi yukarıda görülen $x < t >$ notasyon'u ile gösterelim. Benzer şekilde her bir output elemanını da yine yukarıda görülen $y < t >$ notasyonu ile gösterelim.

T_x ve T_y de length of input and output sequences'ı temsil etsin.

Yani i. training example X burada 9 elemanlı bir sequence oluyor. Buna karşılık da yine 9 elemanlı bir output söz konusu. i.th training example X sequence'inin 3. Sequence elemanını gösterme istersem yukarıda görüldüğü gibi $x^{(i)} < 3 >$ şeklinde gösteririm.

Bir diğer önemli nokta ise şu, her bir training example farklı bir sequence length'e sahip olabilir yani $T_x(i)$ her örnek yani her i için farklı olabilir ayrıca bu örnek için $T_x(i)$ ile $T_y(i)$ her durumda birbirine eşittir.

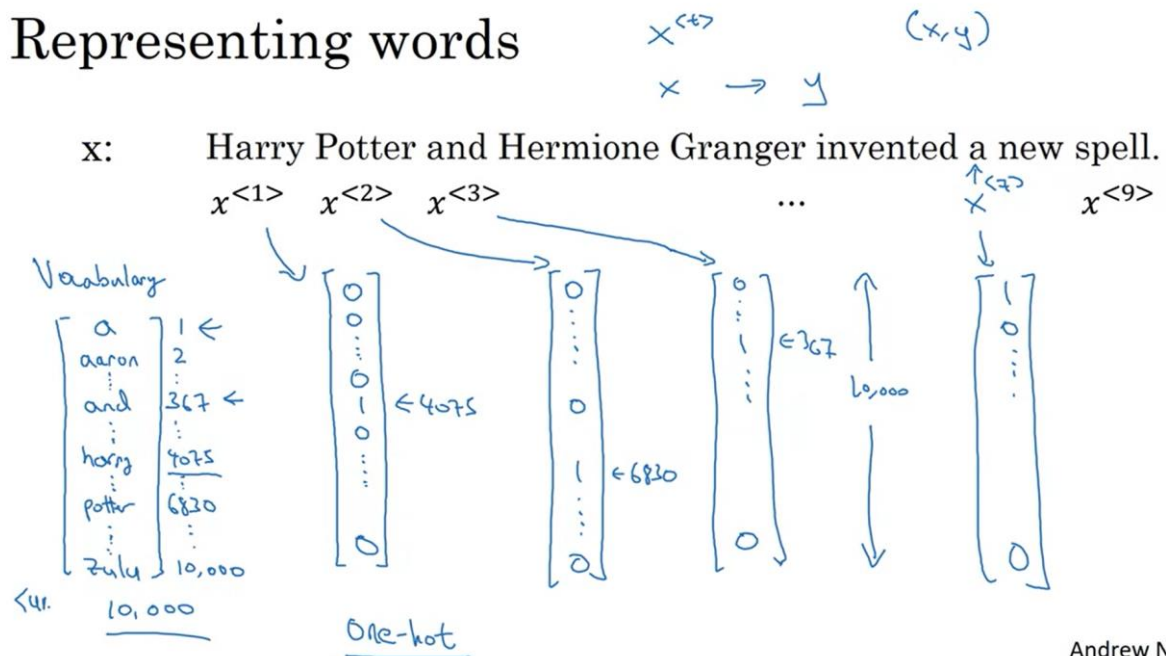
Yukarıda örnekte kafama takılan büyük bir soru var, tamam output'u 0 1 olarak represente edebiliyoruz çünkü output'un taşınması gereken tek bilgi ilgili kelimenin isim olup olmadığı bu bir boolean bilgi tek bir 0-1 ile represente edilebilir.

Peki ya input sequence elemanları nasıl represente edilecek? Her bir $x^{(t)}$ bir string'e karşılık geliyor, modelin inputları ayırt etmesi için bunu sayısal veriye çevirmeliyiz ama nasıl? Bunu tek bir boolean gibi temsil etmek mümkün değil.

Böyle bir representasyonu yapmak için ilk önce bir vocabulary list veya dictionary oluşturmalıyız. Yani aşağıda görüldüğü gibi kendi representasyonumuzda kullanacağımız kelimeleri içeren bir liste oluşturmalıyız.

Bu listenin ilk kelimesi "a" olsun sonra "aaron" şeklinde devam etsin içinde "harry" de olacak "potter" da mesela "zulu" ile bitecek. Aşağıda görüldüğü gibi her kelime bir index'e karşılık gelecek diyelim ki dictionary'imizde 10 000 sözcük olsun. Aslında bu sayı modern NLP projeleri için oldukça küçük. Genelde 30k – 100k kullanılırken 1M'e kadar gidebilir.

Representing words



Andrew Ng

Bu dictionary'i oluşturmak için training sette en çok kullanılan 10k kelimeyi seçebiliriz veya daha önce yapılmış online dictionaries kullanabiliriz, İngilizcede en çok kullanılan 10k kelimeyi seçebiliriz.

Daha sonra da artık training set'imizdeki bir sequence element'i one-hot representation ile yukarıda görüldüğü gibi represente edebiliriz. Yani her bir sequence elemanı bir one-hot vector olarak represente edilecek one-hot diyoruz çünkü vektörün sadece tek bir elemanı 1 gerisi 0, bu hot eleman da sözlükte karşılık gelen kelimeyi gösteriyor.

Bu noktadan sonra artık input ve output'umuzu numerik olarak represente edebiliyoruz, olay bir supervised model eğitmeye indirgenmiş olacak. İlgili X'ler ile Y'leri map eden bir fonksiyon öğrenmemiz gerek.

Toparlamak gerekirse bu dictionary yöntemi ile, ben artık bir kelimeyi 10k dimensional bir uzayda nokta olarak temsil edebilirken, bir cümleyi de aynı uzayda bir noktalar kümesi olarak temsil edebilirim. Bu numerik temsil benim bir model eğitmem için gerekli ilk adımdı.

Bir başka önemli soru işareti şu? Ya cümlede kullanılan kelime bizim vocabularyimizde yoksa? Bu durumda vocabulary'e bir "unknown word" ekleriz ve ilgili kelime bu kelime olarak kabul edilir, bundan daha sonra daha detaylı bahsedeceğiz.

Vid 3

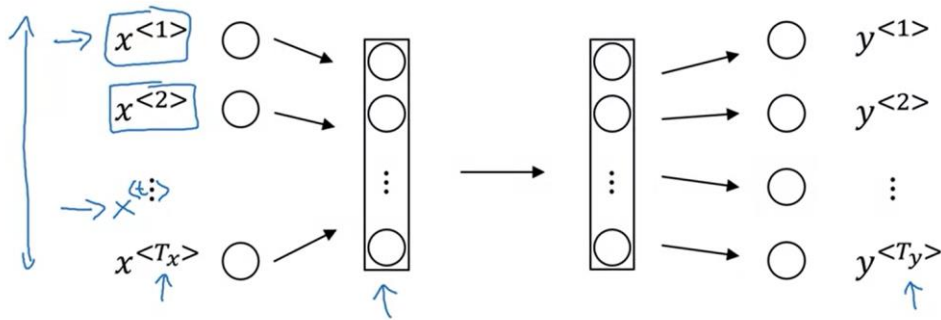
Recurrent Neural Network Model

Geçtiğimiz kısımda sequence learning problems için kullanılan notation'dan bahsettik. Şimdi ise X ile Y 'i map edebilecek bir fonksiyonu öğrenecek olan neural network modelinden bahsedelim.

Akla gelen ilk yöntem, bahsedilen task için standard neural network yapısını kullanmak.

- Yani, mesela önceki örnekte 9 input words vardı, her birini 10k uzunluğunda one hot vectors ile temsil etmiştik, bu vektörleri 9 adet input neuron'a besleyebiliriz, daha sonra birkaç hidden layer ekleyip yine 9 adet output neuron kullanarak modelin gerekli $X \rightarrow Y$ mapping function'ı öğrenmesini umabiliriz.

Why not a standard network?



Problems:

- - Inputs, outputs can be different lengths in different examples.
- - Doesn't share features learned across different positions of text.

Andrew Ng

Ama bu yaklaşım iyi bir yaklaşım değil. Bunun iki temel sebebi var:

- Her örnek için input ve output length aynı değil, bu sebeple sabit boyutlu bir NN kullanamayız, yani belki ith training example için X 9 kelimeden oluşuyor ancak $i+1$ th example için 20 kelimeden oluşuyor bu durumda, standard neural network yapısını kullanmak imkansız!
 - Bu problemi engellemek için akla gelen bir yöntem, maximum length'i belirleyip daha kısa örnekleri padding ile düzenleyebiliriz, ama yine de çok iyi bir yaklaşıma benzemiyor.

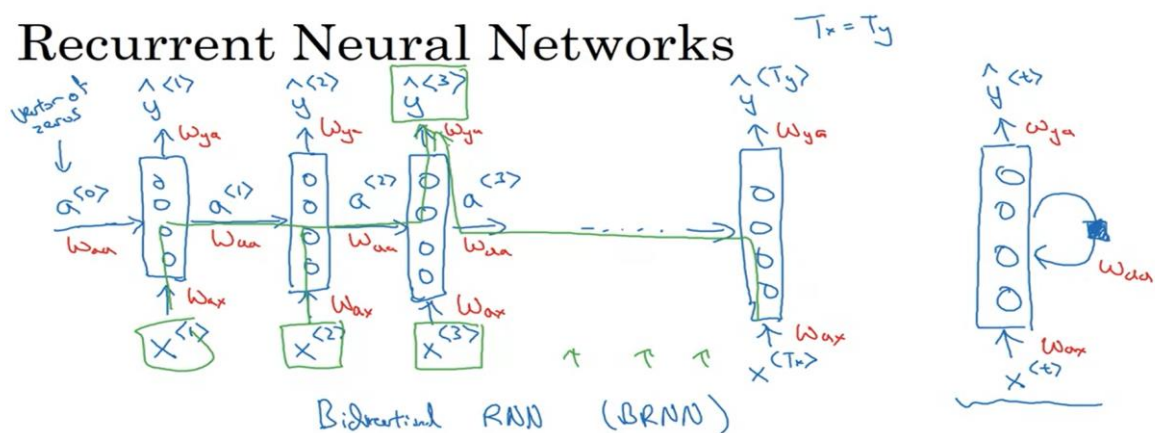
- 2. ve daha önemli problem ise şu, yukarıdaki gibi bir NN architecture text'in farklı pozisyonlarında öğrendiği feature'ları paylaşmıyor.
 - In particular, diyelim ki NN ilk neuron'a input olarak "harry" gelirse bunun bir isim olduğunu öğrendi, yani bu şu demek ilk neuron'a "harry" inputu gelirse bir şekilde, $y_{<1>}$ çıkışı 1 olacak şekilde bir fonksiyon öğrenildi, ancak aynı input bir başka neuron'a gelirse model bunun isim olabileceğini düşünmüyor, bunu ayrıca öğrenmesi lazım.
- Ayrıca böyle bir modelde input layer'ın parameter sayısı çok fazla olacaktır, çünkü diyelim ki max length training example 20 kelimedenden oluşuyor, her bir kelime 10k'lık bir vektörle temsil ediliyor 200k parameter tek bir layer için gerekecek, çok fazla!

İşte Recurrent Neural Networks bu noktada devreye giriyor ve bu problemlerin tamamını çözüyor!

Nedir bu Recurrent Neural Network (RNN)? Let's build one up.

- Timestep 1 için ilk kelime $x_{<1>}$ 'i alıyorum ve bir NN layer'a besliyorum, daha sonra bu input ile $y_{<1>}$ 'i tahmin ediyorum, yani sadece ilk kelimeye bakarak, bu kelimenin isim olup olmadığını tahmin ediyorum.
- Benzer şekilde timestep 2 için, $x_{<2>}$ aynı layer'a input olarak veriliyor fakat bunun yanında $a_{<1>}$ yani bir önceki input'un aktivasyonu da input olarak kullanılıyor. Bu şekilde $y_{<2>}$ hesaplanıyor.
- Bu noktada timestep1 için de bir $a_{<0>}$ aktivasyonunun kullanıldığına dikkat çekelim, ancak bu aktivasyon, temelde bir zeros vector'den fazlası değil.
- Benzer şekilde, sıradaki timestep için de aynı network $x_{<3>}$ 'ü ve $a_{<2>}$ 'yi input olarak alarak $y_{<3>}$ 'ü hesaplar. Bu bu şekilde T_x 'e kadar devam eder.

Recurrent Neural Networks



He said, "Teddy" Roosevelt was a great President."

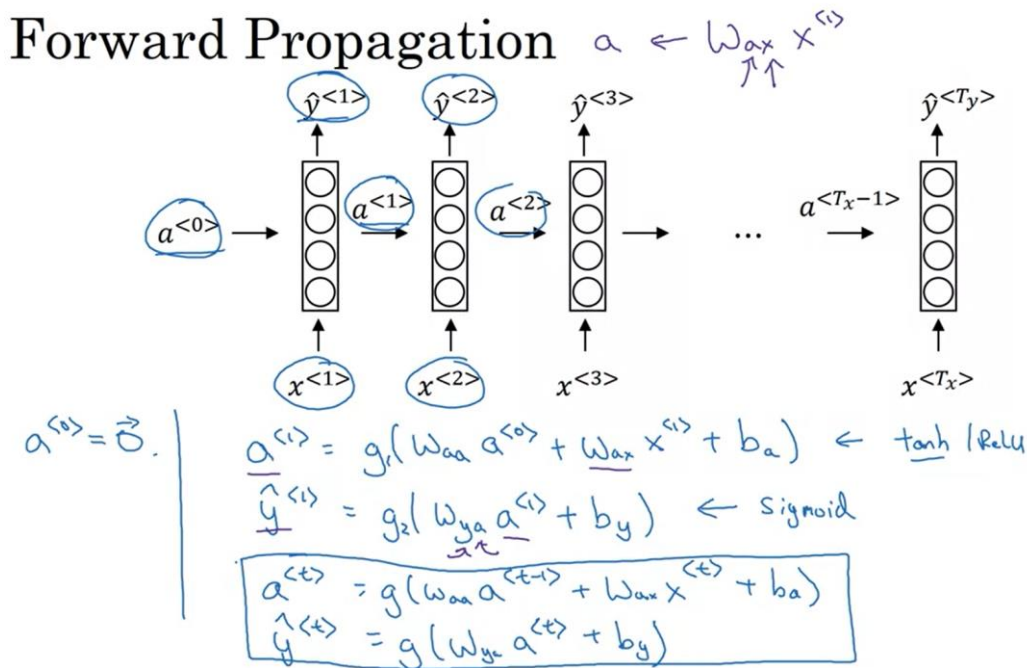
He said, "Teddy" bears are on sale!"

- Burada görünenin tek bir training example için olduğunu unutma yani tek bir örnek için önce ilk kelime girdi olarak veriliyor, ilk kelimeye denk düşen çıktı hesaplanıyor ve daha sonra 2. Kelime'nin çıktısını hesaplamak için hem ilk kelimenin aktivasyonu hem de ikinci kelime kullanılıyor, bu olay bu şekilde son kelimeye kadar devam ediyor ve sonucunda her kelimeye karşın, o kelimenin isim olup olmadığı çıktısı tahmin edilmiş oluyor.
- Bazı yerlerde yukarıda açık şekilde verilen recurrent neural network'ün yine yukarıda sağda olduğu gibi tek bir layer ve loop işareti ile gösterildiğini de görebiliriz, bu da her time step için aynı network'e $x_{<t>}$ 'nin ve $a_{<t-1>}$ 'in beslendiğini ve sonucunda $y_{<t>}$ 'nin hesaplandığını gösterir, ancak soldaki unrolled diagram modeli daha anlaşılır olduğu için ders boyunca bu gösterim kullanılacak.
- Yukarıdan anlaşılabilirliği gibi, RNN data'yı soldan sağa tarar ve her timestep için kullanılan parametreler aynıdır yani parameter paylaşımı söz konusudur. Bu özelliği sayesinde, daha önce bahsedilen 200k parameter problemini çözmüş oluyoruz, her input için bir başka parameter eğitmiyoruz, aynı parameter setini tüm sequence elementlere uygulanacak şekilde eğitiyoruz, bu yaklaşım bir nevi CNN'i orada da verimliliği artıran bir etken, her input pixel için bir parameter atamak yerine aynı parametreleri kullanıp input üzerinde gezdirmek, böylece aynı parametreleri resmin farklı yerlerinde feature detect etmesi için kullanıyorduk, burada da aynı parametreleri cümlelerin farklı yerlerinde feature detect etmesi için kullanmış oluyoruz.
- Kullanılan parametreleri aşağıda detaylı anlatacaz ama yukarıda bakarsak, her timestep için input $x_{<t>}$ bir W_{ax} parametresi ile işlem görürken her activation da W_{aa} parametresi ile işlem görür.
- Son olarak output prediction için de her timestep'de W_{ya} parameter vector kullanılır.

Ancak farketmiş olduğumuz gibi böyle bir RNN yapısı tahminlerini sadece geçmiş data'ya bakarak yapıyor, ve aslında cümlelerin tamamını göz önünde bulunduramıyor. Yani ilk kelimenin kişi ismi olup olmadığını anlamak için sadece ve sadece ilk kelimeyi kullanıyor, benzer şekilde üçüncü kelimeyi tespit ederken sadece ilk 3 kelimeyi göz önünde bulunduruyor, bu durum pek verimli değil ve problemlere yol açabilir, bu durumdan kurtulmak için bi-directional RNN yapısını ilerleyen derslerde göreceğiz, şimdilik temeli anlamak için bu yapıyı kullanmaya devam edelim.

Şimdi yukarıda mantığı açıklanan uni-directional RNN için yapılan calculation'ları anlayalım.

- Tipik olarak $a^{<0>}$ aktivasyonu zero vector olarak alınır ve işe bununla başlanır.
- $a^{<1>}$ aktivasyonunu hesaplamak için aşağıda görüldüğü gibi, W_{aa} parameter vector ile $a^{<0>}$ çarpımı ve W_{ax} parameter vector ile $x^{<1>}$ çarpımına ek olarak ba constant vector'ü g_1 aktivasyon fonksiyonu ile kullanılır.
- Ardından ilgili timestep çıkışı hesaplamak için de $a^{<1>}$ aktivasyonu ile W_{ay} parametresinin yanında by constant vector'ü kullanılır.
- Bu noktada g_1 ve g_2 'nin farklı aktivasyonlar olabileceğini unutma, output hesabı için problemin türüne göre sigmoid veya softmax yaygın olarak kullanılabilirken ara aktivasyonların hesabında genelde tanh veya relu kullanılır.
- Bu hesaplamayı genelleştirirsek, aşağıda kutu içine alınan genel denklem elde edilebilir.



Andrew Ng

Yukarıda kutu içine alınan denklemi daha iyi anlamak ve daha basit bir şekilde göstermek için açalım:

Simplified RNN notation

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

$$y^{<t>} = g(W_y a^{<t>} + b_y)$$

$$a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$

$$[W_{aa}; W_{ax}] = W_a \quad (100, 10,100)$$

$$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} \quad (10,100)$$

$$[W_{aa}; W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa}a^{<t-1>} + W_{ax}x^{<t>}$$

Andrew Ng

Burada W_{aa} , W_{ax} ve W_{ya} 'yı ayrı ayrı layer parametreleri olarak düşünebilirsin, yani eğer 100 elemanlı bir $a^{<1>}$ vektörü elde etmek istiyorsam, $a^{<0>}$ da 100 elemanlı olacak bu durumda W_{aa} 100x100 boyutunda bir matris olacak, eğer $x^{<1>}$ 10k boyutunda ise W_{ax} 100x10k boyutunda olacak, son olarak W_{ya} da 1x100 boyutunda olacak.

Sonuçta W_{aa} ve W_{ax} 'ı ayrı ayrı göstermek yerine sağdaki gibi W_a olarak gösterebiliriz bu W_a matrisi W_{aa} ve W_{ax} 'in horizontal olarak concatenate edilmiş haline denk gelirken, $[a^{<t-1>}, x^{<t>}]$ gösterimi ise bu iki vektörün vertical olarak concatenate edilmiş haline denk gelir.

Böylece $W_{aa}a^{<t-1>} + W_{ax}x^{<t>}$ işleminin sonucu ile $W_a[a^{<t-1>}, x^{<t>}]$ işleminin sonucu birebir aynı olur.

Benzer şekilde W_{ya} yerine de W_y kullanılabilir ve böylece notasyonu sadeleştirmiş oluruz.

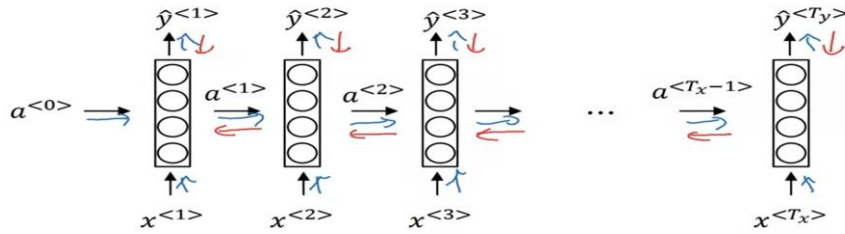
Vid 4

Backpropagation Through Time

Basic RNN structure'ı öğrendik, şimdi RNN için backpropagation'ı anlamaya çalışalım. Aslında programming frameworks otomatik olarak backpropagation kısmını hallediyor ama yine de bir anlayış geliştirmiş olalım.

Forward prop. için mavi oklarla görüldüğü gibi soldan sağa doğru aktivasyonları ve daha sonra çıkışları hesaplıyorduk. Her bir output farklı bir timestep için hesaplanıyordu. Bu durumda aynı daha önce olduğu gibi, backpropagation operasyonunun yönünü terse çevirir ver, weighlerin değişiminin loss'u nasıl değiştirdiğini yani gradients'i hesaplar.

Forward propagation and backpropagation

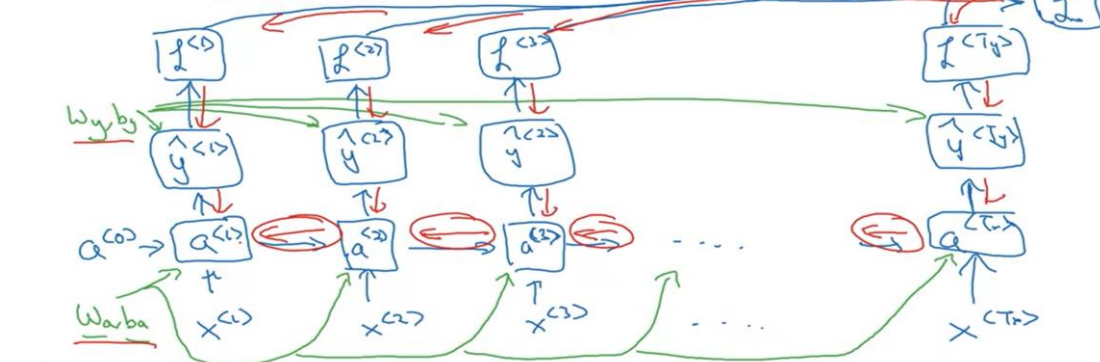


Andrew Ng

Aşağıda bu işlemlerin computation graph'ini anlayalım a0 ve x1 inputları ile Wa,ba parametreleri kullanılarak a1 hesaplanıyor daha sonra bu a1 ile Wyby parametreleri kullanılarak y1 hesaplanıyordu bu işlem farklı timestepler için sağa doğru tekrarlayarak devam ediyordu.

Her timestep için aynı Wa,ba ve Wy,by parametrelerinin kullanıldığına dikkat et.

Forward propagation and backpropagation



$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log (1 - \hat{y}^{<t>})$$
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) \leftarrow$$

Andrew Ng

Tek bir sequence element için loss function yukarıda gibi tanımlanmış olabilir bu standard binary cross-entropy loss'dan fazlası değil.

Tek bir training example için loss hesaplamamız gerekirse ise yukarıdaki gibi bu loss'ları T_x boyunca toplamamız yeterli olacaktır.

Sonuçta böyle bir loss function hesaplanırken, zamanda toplama yapılmış oluyor, yani timesteps için loss'lar ayrı ayrı bulunup toplandı, e bu durumda ben backpropagation yaparsam da Loss'un W_a, b_a ya göre türevini bulabilmek için zamanda geriye doğru gitmiş oluyorum.

İşte bu sebeple RNN için kullanılan backpropagation'a havalı olsun diye "Backpropagation Through Time" diyorlar, bir numarası yok sonuçta aynı işlem yapılıyor.

Vid 5

Different Types of RNNs

Şuana kadar RNN yapısını tek bir örnek üzerinde inceledik bu örnekte yapılan işlem bir cümleyi input olarak alıp, ilgili cümledeki hangi kelimelerin kişi ismi olduğunu output olarak veriyordu.

Bu örnek için Input Sequence Length, T_x ile Output Sequence Length, T_y 'nin hep eşit olduğunu kabul ettik. Bu durum diğer uygulamalar için böyle olmak zorunda değil.

Bu kısımda daha geniş yelpazede RNN mimarilerine değineceğiz.

Aşağıdaki slaytı ilk kısımdan hatırlayabiliriz, farklı sequence model problemlerden bahsedilmişti burada şunu söyleyebiliriz ki T_x ile T_y her zaman aynı olmak zorunda değil.

- Örneğin music generation için $T_x=1$ veya empty set olabilir.
- Benzer şekilde sentiment classification için output 1-5 arasında bir integer iken, input ise sequence data olabilir.
- Machine translation için de input ve output length farklı olabilir.

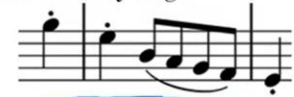
Examples of sequence data

Speech recognition



"The quick brown fox jumped over the lazy dog."

Music generation



Sentiment classification

"There is nothing to like in this movie."



DNA sequence analysis

AGCCCCTGTGAGGAACTAG



AGCCCCTGTGAGGAACTAG

Machine translation

Voulez-vous chanter avec moi?



Do you want to sing with me?

Video activity recognition



Running

Name entity recognition

Yesterday, Harry Potter met Hermione Granger.



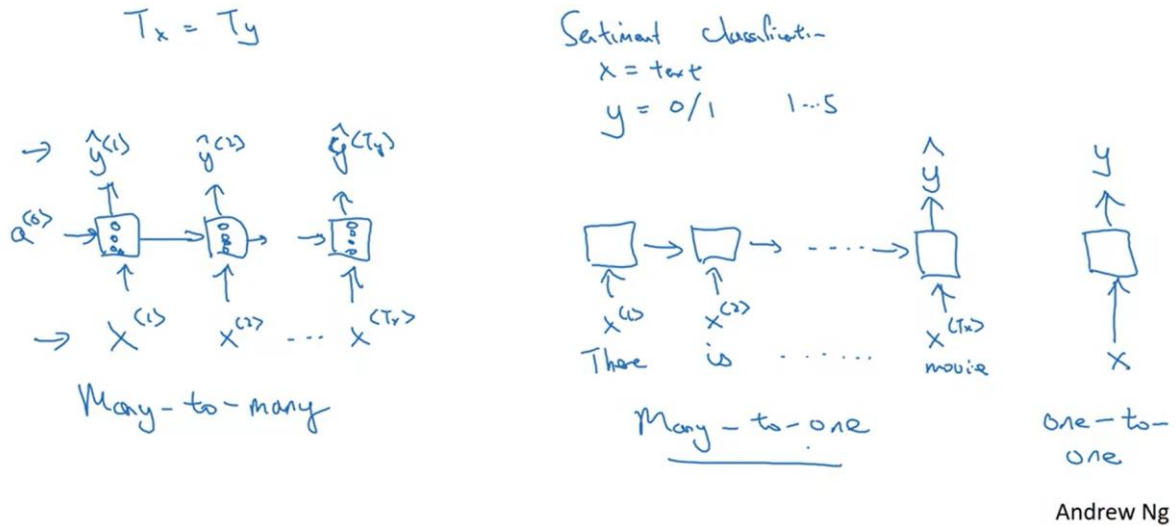
Yesterday, Harry Potter met Hermione Granger.

Andrew Ng

Bazı örnekleri inceleyelim:

- Şimdiye kadar incelediğimiz problem bir name entity recognition problemi idi, input size T_x iken output size hep ona eşit oluyordu yani $T_x = T_y$.
- Aşağıda görüldüğü gibi bu yapıya Many-to-Many architecture denir. Çünkü hem input hem de output sequence has many examples.

Examples of RNN architectures



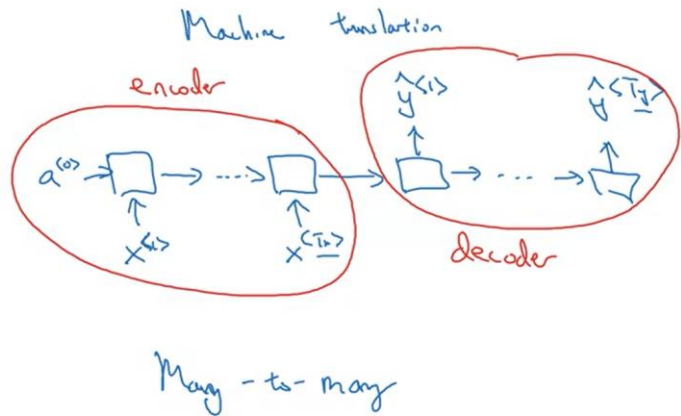
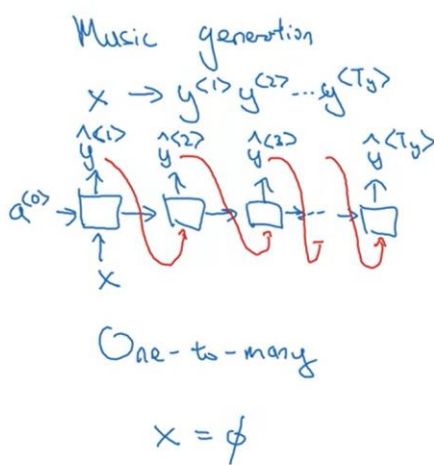
- Bir diğer architecture ise Many-to-One olarak sınıflandırılabilir, örneğin Sentiment Classification problemi için, T_x elemanlı bir sequence cümle input olarak alınır output olarak ise 0/1 kullanılabilir yani sadece yorumun pozitif veya negatif olup olmadığına bakılabilir veya 1-5 arasında bir integer ile puanlandırma yapılabilir.
- Örneğin "There is nothing to like in this movie" gibi bir input alınıyor ve sonucunda 1-5 arası bir puan döndürülüyor, bu durumda her timestep için bir output hesaplanmıyor, tüm cümle interpret edildikten sonra bir output çıkarılıyor.
- Bu noktada aklıma aynı işlemin, name entity recognition içinde yapılabileceği geldi, yani tüm cümle okunduktan sonra T_x boyunda bir output ile çıkış sağlanamaz mı acaba? Sadece bir düşünce.

Ayrıca One-to-One bir yapıdan da söz etmek mümkün ama böyle bir problem için RNN kullanmak pek gerekli değil, daha önce gördüğümüz yapılar iş görecektir.

Architecture'ları incelemeye devam edelim:

- Bir başka örnek One-to-Many architecture olabilir, örneğin music generation için input olarak ya hiçbir şey vermeyiz veya integer olarak bir genre veririz, output olarak T_y elemanlı bir sequence bekleriz.
- Böyle bir örnekte timestep1 için input verilir ve daha sonra hiç input verilmeden RNN yapısı sadece bir önceki timestep'ın aktivasyonunu ve outputunu kullanarak yeni bir nota üretir. Bu şekilde bir müzik oluşturmuş olur.

Examples of RNN architectures



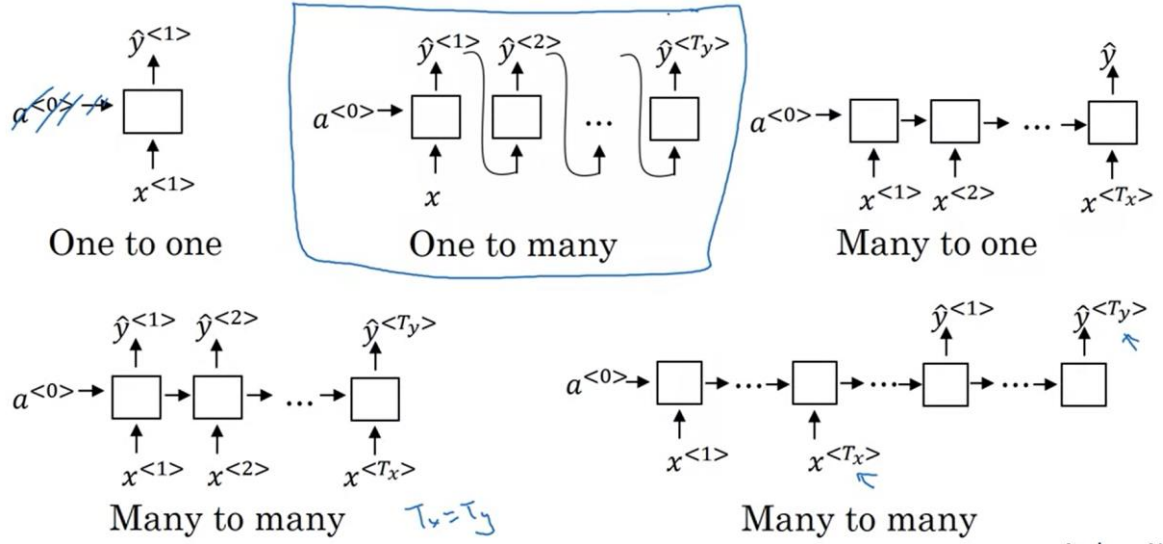
Andrew Ng

Son olarak T_x ve T_y 'nin farklı olduğu bir Many-to-Many architecture'ından da bahsedilebilir:

- Bunun için Machine Translation düşünülebilir, input olarak "Selam, naber?" verdiğimizde output olarak "Hey, how are you?" alıyoruz $T_x=2$, $T_y=4$ oluyor, daha da farklı olabilir bu durumda da yukarıda sağda görüldüğü gibi bir encoder decoder yapısı kullanılabilir.
- RNN yapısı önce T_x elemanlı sequence input'u okur, ve bu inputu encode eder yani tek bir aktivasyon oluşturur, daha sonra kalan timestepler için bu aktivasyon kullanılarak T_y timestep ile translation sequence output decode edilir.

Aşağıda temize çekilmiş RNN architectures görünüyor.

Summary of RNN types



Andrew Ng

Kutu içine alınan One to Many yapısı music generation için kullanılabiliyordu, bu yapı için bahsedilmesi gereken bazı subtleties (kurnazlıklar) söz konusu.

Önümüzdeki kısımda bundan bahsedilecek.

Vid 6

Language Model and Sequence Generation

Language Modeling, NLP'nin en temel ve önemli tasklerinden biridir. RNN ile gayet iyi bir şekilde language modeling yapılabilir.

Bu kısımda RNN kullanarak nasıl Language Modeling yapabileceğimize bakacağız.

İlk olarak language modelling'in ne olduğundan bahsedelim:

- Diyelim ki bir speech recognition sistemi oluşturuyoruz ve ses inputu olarak "The apple and pear salad was delicious" cümlesi verildi, burada sistemin pear sözcüğü ile pair sözcüğünün telaffuzunu karıştırmaması muhtemelen, bunu contextten anlaması gerek.
- İşte language modelling bu noktada devreye girer. İki cümlenin söylenmiş olma ihtimalini karşılaştırır ve yüksek olasılıklı olanın söylendiğini kabul eder.

What is language modelling?

Speech recognition

The apple and pair salad.

→ The apple and pear salad.

$$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$$

$$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$$

$$P(\text{Sentence}) = ? \quad P(y^{(1)}, y^{(2)}, \dots, y^{(T)})$$

Andrew Ng

Sonuçta bir language model'in oluşturulduktan sonra yaptığı iş: input olarak bir sentence alır ve sonucunda bu cümlenin olasılığını döndürür.

Ne demek cümlenin olasılığı derken kastedilen şu: Diyelim ki rastgele bir kitabı açtık, veya televizyonu açtık vesaire ilk duyduğumuz cümlenin bu cümle olma olasılığı nedir? Bu yüzden yukarıda yazılan olasılık değerleri son derece düşüktür.

Bu işlem speech recognition veya machine translation için kullanılır ve hangi cümlenin söylenmesinin daha muhtemel olduğunu hesaplayarak, arada kaldıklarında o cümleyi seçerler.

Peki bir cümle verildiğinde bunun olasılığı döndüren bir Language Model'i nasıl build ederiz?

- Böyle bir modeli RNN yapısı ile oluşturabiliriz.
- İhtiyacımız olan şey Large corpus of english text'ten bir training set. Yani training set olarak çok fazla sayıda cümleye ihtiyaç duyuyoruz. Böyle bir training setimizin olduğunu varsayalım.
- Daha sonra kendimize daha önce bahsettiğimiz gibi bir vocabulary/dictionary oluşturabiliriz, bunun amacı her kelimeyi sayısal olarak represente edebilmek.
- Daha sonra bu training set içinden rastgele bir cümle aldık diyelim ki: "Cats average 15 hours of sleep a day." Böyle bir cümle dictionary'i kullanarak tokenize edilebilir yani sayısal olarak ifade edilebilir. Yani her bir kelimeyi bir one-hot vector olarak ifade edebiliriz.
- Yapılacak bir başka şey de cümle sonlarını represente etmesi için ekstra bir token eklemek, yani cümle sonunda koyulan noktayı bir kelime gibi represente ediyoruz. <EndOfSentence> token'i ile.

Language modelling with an RNN

Training set: large corpus of english text.

Tokenize

Cats average 15 hours of sleep a day. ↓ <EOS>

$y^{(1)}$ $y^{(2)}$ $y^{(3)}$... $y^{(n)}$ $y^{(n+1)}$
 $x^{(t)} = y^{(t-1)}$

The Egyptian Mau is a bread of cat. <EOS>

10,000

<UNK>

Andrew Ng

- Diyelim ki bir başka cümle de yukarıdaki gibi The Egyptian Mau is a bread of cat. Bu cümledeki bazı kelimeler vocabulary'imde yer almayabilir mesela Mau vocabularyim de yok diyelim, bu durumda bu kelime yerine UNKnown tokenini kullanırız yani bilinmeyen kelimeleri de sayısal olarak ifade ederiz.

Özetle şuan elimizde çok fazla sayıda cümleden oluşan bir dataset var, bunun yanında bir dictionaryimiz var ve training set'in tamamını tokenize edebiliyoruz yani sayısal olarak represente edebiliyoruz buna cümle sonları ve bilinmeyen kelimeler de dahil.

Artık bir RNN ile Language Modelling yapmaya hazırız, bir cümle verince bu cümlenin olasılığını output verecek bir sistem oluşturmaktan bahsediyoruz.

Diyelim ki dictionary size'imızı 10k.

Bir training example olarak "Cats averga 15 hours of sleep a day." sequence'ı alınmış olsun.

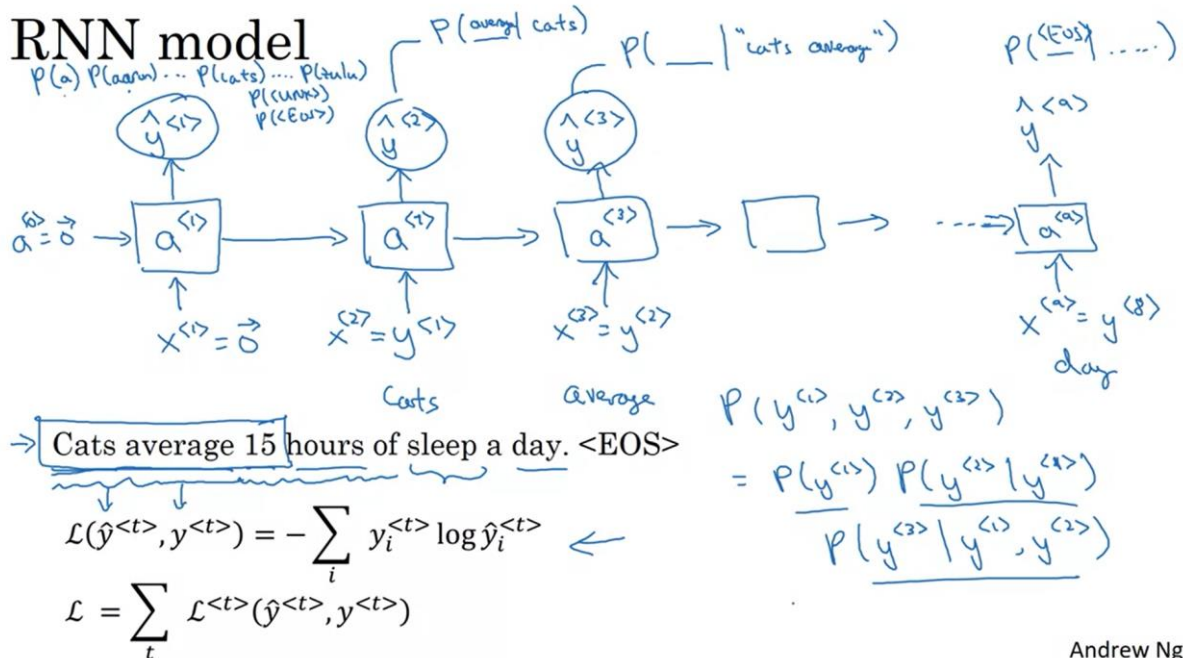
Modelin eğitimi şöyle oluyor, öncelikle a_0 ve x_1 olarak 0 vektörleri timestep 1 için RNN'e veriliyor ve output olarak model 10k unitli bir softmax output'u veriyor her biri sıradaki kelimenin output'unu temsil ediyor, yani timestep1 için modelin doğru cevabı $P(\text{cats})$ 'in diğerlerinden daha yüksek olması, tabi ilk kelime olduğu için bu pek de anlamlı değil, bu durumda yüksek olan olasılıklar, çok kullanılan kelimeler olacaktır.

Herneyse olay burada bitmiyor devam edelim, timestep 2 için modele a_1 ve x_2 veriliyor burada $x_2=y_1$ yani modele bir önceki kelime verilmiş oluyor bu durumda modelden beklenen, cats kelimesinden sonra gelebilecek kelimelere yüksek olasılık vermesi, bu örnek için doğru cevabın "average" kelimesi olması gerekiyor.

Yani backpropagation yaptığımızda bu spesifik "örneğin sisteme etkisi şu olacaktır, bundan sonra bir cümlede "cats" geldiyse sistem "average" kelimesinin olasılığını artıracaktır.

Devam ediyoruz, sisteme sıradaki timestep için önceki kelimelerin yani "cats average" bilgisi iletilir ve sistemin output olarak 10k kelime arasından "15" e ağırlık vermesi beklenir.

Elbette tek bir cümle ile pek bir şey değişmeyecektir ancak düşün ki ben sistemi milyonlarca cümle için eğittim ve bu cümlelerin belki 500 tanesinde "Hey man, whats up?" phrase'i yer alıyordu bu durumda ben eğitilmiş sisteme "Hey man whats" kelimelerini verdiğim zaman, sistem outputunda "up" kelimesinin olasılığının diğerlerine göre bariz yüksek olmasını beklerim!



Tabii ki sistemi nasıl eğitiyoruz? Cost function ile eğitiyoruz, tek bir sequence data için sistem'in doğru output'u belli yani cats verildiğinde sistem "average"'ın olasılığını yüksek verirse, ceza kesilmeyecek, diğerlerinin verirse ceza kesilecek, bildiğimiz softmax cost function kullanılabilir.

Sonuçta tek bir training example için tüm timesteplerin loss'u toplanır.

Sonuta byle oluřturulan bir RNN’i ok byk bir dataset zerinde eęittięimizi varsayarsak, biz artık sisteme “Hey my name” inputunu verdięimizde sistemin “is” kelimesi iin yksek olasılık verirken “apple” kelimesi iin dřk olasılık verdięini grrz.

Ayrıca byle bir modeli kullanarak da tm bir cmlenin olasılıęını hesaplayabiliriz. Diyelim ki “Selam dostum nasılsın?” cmlenin olasılıęını hesaplamak istiyoruz, eęitilen RNN’e nce 0 input veriz ve “Selam” kelimesinin olasılıęını hesaplatırız, daha sonra “Selam” kelimesini veriz ve “dostum” kelimesinin olasılıęını hesaplatırız, son olarak “Selam dostum”u vererek, “nasılsın?” kelimesinin olasılıęını hesaplatırız ve bu  olasılıęın arpımıyla ilgili cmlenin rastgele karřımıza ıkma olasılıęını hesaplamıř oluruz.

Bylece birka farklı cmlle arasında kaldıęımız zaman hepsinin olasılıklarını hesaplayabiliriz ve yksek olasılıklı olanı seebiliriz, temelde modelin ęrendięi řey, belirli kelimeler verildięinde bir sonraki kelime olarak ilgili kelimenin gelme olasılıęını ęrenmek. Byle bir model ile cmlelerin olasılıęı da hesaplanabiliyor.

Vid 7

Sampling Novel Sequences

Geçen kısımda bir Sequence Model'i nasıl eğitebileceğimizi anlamıştık. Modelin nasıl şeyler öğrendiğini anlamamanın bir yolu: "Sampling Novel Sequences".

Önceki kısımda gördüğümüz gibi bir sequence model aşağıdaki gibi bir yapıydı, yani a_0 ve x_1 olarak zeros vector veriyorduk daha sonra y_1 olarak dictionary içindeki her kelime için bir olasılık outputu alıyorduk, max olasılıklı kelimeyi timestep 2'ye geçirip aynı işlemi tekrarlıyorduk.

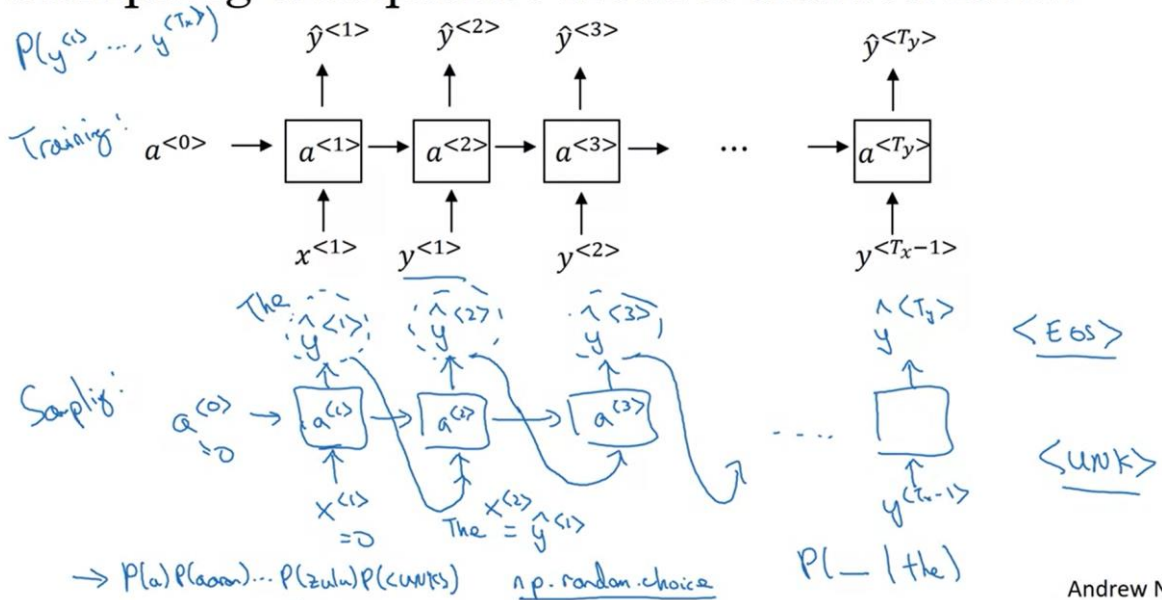
Aslında bu yaklaşımla, eğitilen modelden sampling yapabiliriz. Yani input olarak a_0 ve x_1 'i zero vector olarak veririz. Sonra üretilen output distribution'a bakıp en yüksek olasılıklı kelimeyi seçeriz, mesela "The" kelimesi seçildi daha sonra bu kelimeyi input olarak timestep 2'ye geçiririz, aynı işlemi tekrarlarız ve 2. Kelimeyi elde ederiz.

Eğer EOS token'i kullanıyorsak, modelimiz bu token'i output olarak verdiği zaman sistemi durdurabiliriz, yani bir cümle tamamlanmış olur. Sistem bize en yüksek olasılıklı cümlesini vermiş olacak. (İstersek bu işe randomness da katabiliriz ve en yüksek olasılıklı cümlelerinden birini vermiş olur.)

EOS tokeni kullanmıyorsak da, mesela 20 kelime üret ve daha sonra dur diyebiliriz.

Benzer şekilde UNK tokeninin kullanılmasını istemezsek, bu seçildiğinde reject ederiz.

Sampling a sequence from a trained RNN



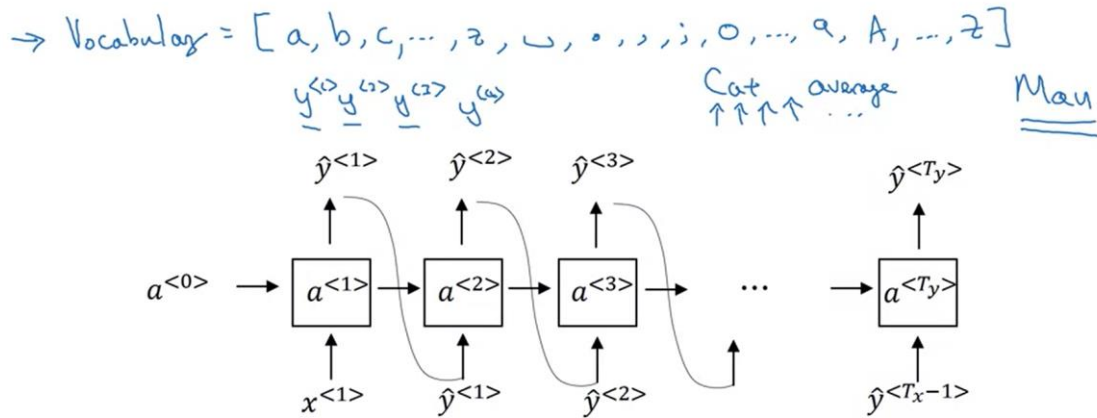
Şimdiye kadar bahsedilen RNN için oluşturulan vocabulary kelimelerden oluşuyordu, uygulamamıza göre buna bir alternatif yaklaşım ise vocabulary'yi kelimelerden değil de harflerden oluşturmak, yani aşağıda gördüğümüz gibi vocabularyimiz $a, b, c, \dots, z, \dots, 0, 1, \dots, 9, A, B, \dots, Z$ şeklinde karakterleri içerebilir.

Böyle bir vocabulary kullanmanın avantajları ve dezavantajları olacaktır.

- Bu durumda her input bir kelime değil karakter olacak ve output da geçmiş karakterlere bakarak bir sonraki adımda hangi karakterin kullanılma olasılığının yüksek olduğunu gösterecek yani output da bir karakter olacak.
- Character level dictionary kullanırsaki, UNKnown word tokenine ihtiyaç kalmaz, çünkü artık kelimeler için değil harfler için eğitim yapıyoruz.
- Ancak bu yaklaşımın dezavantajı ise çok daha uzun sequence'ları ihtiyaç duymamız, yani bir cümlede ortalama 10 kelime varsa belki yüzlerce karakter var.
- Character-level language models are not as good as word level language models at capturing long range dependencies, yani cümlelerin başında olan şey bazen sonunu etkileyebiliriz mesela cats ile başlarsa sonda were kullanılacak, cat derse was bu durumu character-level language model ile öğrenmek imkansızca yakın.
- Ayrıca bu yaklaşımı train etmek much more computationally expensive.

Character-level language model

→ Vocabulary = [a, aaron, ..., zulu, <UNK>] ←



Andrew Ng

Yani sonuçta hala çoğu uygulamada word-level yaklaşım kullanılıyor ancak bilgisayarlar geliştikçe ve bazı spesifik uygulamalarda word-level yaklaşım da kullanılabiliyor.

Özetlersek, çok fazla sayıda cümle içeren bir training set kullanarak, bir RNN'i language model olarak eğitebiliriz(word level veya character level farketmez) ve daha sonra sequence sampling'i modelimize sıfırdan cümlelere ürettirebiliriz, bu cümleler modelin dataset'ine göre sık kullanılan kelimelerin biraraya gelmesinden oluşan cümlelerdir, aşağıda haber datası ve Shakespeare datası üzerinde eğitilen modellerin ürettiği sequence örnekleri yer alıyor.

Sequence generation

News

President enrique peña nieto, announced
sench's sulk former coming football langston
paring.

"I was not at all surprised," said hich langston.

"Concussion epidemic", to be examined. ←

The gray football the told some and this has on
the uefa icon, should money as.

Shakespeare

The mortal moon hath her eclipse in love.

And subject of this thou art another this fold.

When besser be my love to me see sabl's.

For whose are ruse of mine eyes heaves.

Andrew Ng

Vid 8

Vanishing Gradients with RNNs

RNN'in genel yapısını anladık ve, name entity recognition veya language modeling işlemleri için nasıl kullanıldığını kavradık.

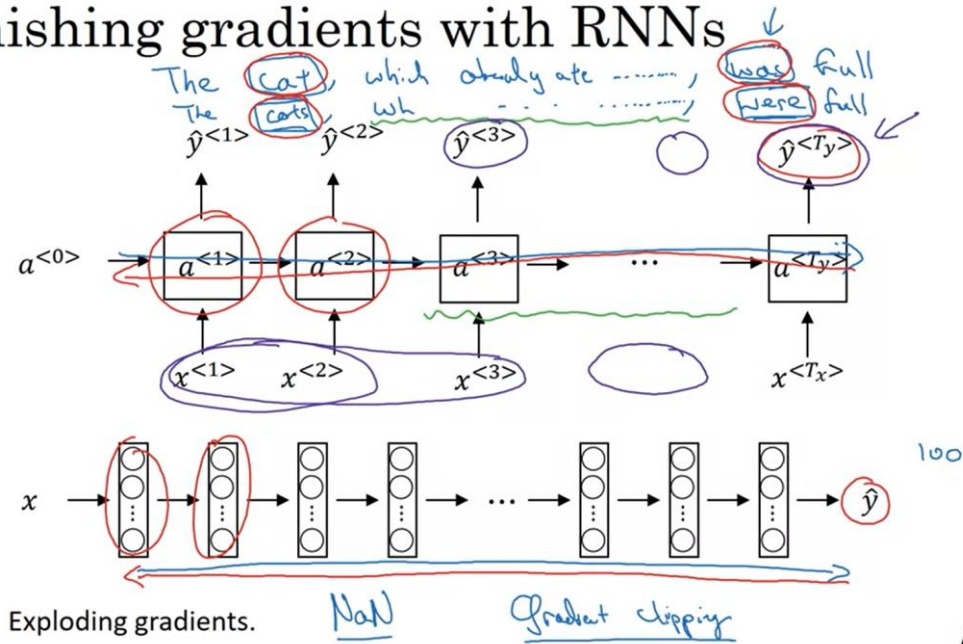
Ancak basic RNN algoritmasının önemli bir problemi daha önce de deep NNs için değindiğimiz "Vanishing/Exploding Gradients Problem".

Bu kısımda bu problemi masaya yatıracağız ve önümüzdeki kısımlarda da, bu ve başka problemleri çözecek bazı daha gelişmiş RNN yapılarından bahsedeceğiz.

RNN yapısının aşağıdaki gibi olduğunu biliyoruz, problemi kavramak için bir "Language Modelling" işlemi için bir RNN eğittiğimizi düşünelim.

- Aşağıda 2 ingilizce cümle görünüyor birisi The cat was full iken diğeri The cats were full şeklinde bir cümle.
- Bu iki cümle bir dilde LONG-TERM DEPENDENCY nedir bunu açıklıyor, cümlelerin en başında söylenen bir kelimedeki bir harf aslında cümlelerin sonunda söylenecek bir başka kelimeyi belirliyor, bu durumda ideal bir modelin cümlelerin başında söylenen kelimeyi unutmamasını ve bu bilgiyi cümlelerin sonuna kadar saklamasını isteriz ki cümle boyunca kelime seçimlerini doğru yapsın.
- Ancak bizim şimdiye kadar gördüğümüz RNN yapısı bu long-term dependencies'i yakalamak konusunda pek başarılı değil.

Vanishing gradients with RNNs



Bu durumu vanishing gradients problemi ile açıklayabiliriz.

- Diyelim ki elimizde yukarıda görüldüğü gibi 100 layerlı bir deep NN olsun.
- Eğer bu network yeterince derinse, backpropagation baştaki layerlara ulaşmakta güçlük çekiyordu, yani backpropagation ile o kadar fazla layerdan geçiyor ki ilk baştaki weightlerin gradient'i çok çok çok küçük çıkıyordu.

Aynı durum RNN yapısı için de geçerli, çok fazla timestep olduğu zaman bu aslında çok derin bir network ile aynı kapağı çıkıyor, backpropagation ile baştaki layerların gradients'i hesaplanmaya çalışıldığında vanishing/exploding gradients problemi başgösteriyor.

Şöyle düşünmek benim aklıma yatıyor: yukarıdaki gibi bir RNN yapısı için herhangi bir output aslında geçmişteki tüm verileri kullanıyor gibi görünse de, en çok etkilendiği veriler local veriler, yani örneğin 10. output kararını en çok bir önceki timestep'in outputuna ve aktivasyonuna göre veriyor elbette ondan öncekilerden de etkileniyor ancak, asıl etki bir öncekinden ve belki birkaç öncekinden geliyor.

Bunu exponentially weighted averages yöntemine benzetiyorum, sonucu en çok etkileyen veriler output'a yakın olan verilerdir. Mesela 10 günlük ewa uyguluyorsak aslında 1. Değer de 100. Değeri etkiler ancak bu çok çok çok küçük bir etkidir, aynen buna benzer şekilde deep networks için ilk layerlardaki parametrelerin son layerlara etkisi çok çok çok çok küçüktür.

Bunun da sebebinin aslında vanishing gradients problemi olduğunu söylüyor veya şöyle diyelim iki probleme de yol açan etken aynı etken, bir outputu belirleyen etkenler yalnızca ona yakın olan inputlar olduğu için, backpropagation yaparken de bir outputun hatasının bedeli yalnızca ona yakın olan unitlere kesiliyor.

Yani bu vanishing gradients problemine yol açan etken aslında, modelin başı ile sonu arasındaki bir kopukluk, model o kadar uzun ki başından sonuna veya sonundan başına veri aktarılarken veri yok oluyor gidiyor. Bu yüzden de başta cat veya cats denildiği bilgisi cümle sonuna gelindiğinde çoktan kaybolmuş oluyor. E doğal olarak backpropagation yaparken de, bir hatanın bedeli baştaki unitlere kesilemiyor yalnızca sona yakın olanlara kesilebiliyor.

Sonuçta bu vanishing gradients problemi basic RNN modelinin bir zayıflığı. Bu zayıflık ilerleyen kısımlarda address edilecek. Bu problem çözülmediği sürece şunu söyleyebiliriz ki basic RNN yapısı long-range dependencies'i tespit etmekte başarılı değil!

Bu noktada daha önce de bahsedilen bir başka problem ise Exploding Gradients problemi yani backpropagation yaparken gradients üssel olarak azalmak yerine üssel olarak artadabilir ancak bu problemi çözmek Vanishing Gradients problemini çözmekten daha kolaydır, ayrıca bu problemi spot etmek de daha kolaydır, tüm parametrelerin gradientleri inanılmaz derecede büyür ve sonucunda NaN şeklinde değerler görmeye başlarız. Bu problem Gradient Clipping ile büyük ölçüde çözülebilir.

- Gradient clipping'i uygulamak çok basittir, gradient vectors'a bakılır ve eğer değerler belli bir threshold'un üzerindeyse clipping ile küçültülür. Bu yöntem ile exploding gradients problemi çözülebilir.

Vanishing gradients'i çözmek ise bu kadar kolay değildir.

Vid 9

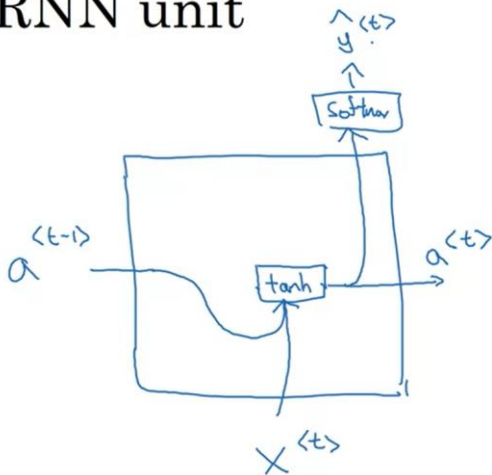
Gated Recurrent Unit (GRU)

Bu videoda RNN'i biraz modifiye edip long range connections capturing konusunda çok daha başarılı bir model ortaya koyacağız, long range connections'ı daha iyi yakalayabilmesi demek aslında timestep 1'deki inputun timestep 1000'i etkileyebilmesi demek yani baştaki weight değişimi sonu etkiler bunu çözdüğümüz zaman zaten vanishing gradients problemi tanımı gereği çözülmüş olur.

Basic RNN unit'in timestep t için aktivasyonu nasıl hesapladığını zaten biliyoruz, aşağıda bunun denklemini ve diyagramını tekrar görelim.

- Temelde yapılan şey, bir önceki timestep'ın aktivasyonunu ve ilgili timestep'ın inputunu almak ve karşılık gelen weightlar ile linear bir operasyondan sonra bir aktivasyonu sokarak timestep t için aktivasyonu hesaplamak.
- Daha sonra bu aktivasyon bir sonraki timestep'e aktarılırken yanında aynı aktivasyon timestep t için output hesaplamaktadır kullanılabilir, bunun için basitçe $a^{<t>}$ 'yi bir softmax layer'a input olarak veriyoruz ve $y^{<t>}$ elde ediyoruz.

RNN unit



$$a^{<t>} = \tanh(g(W_a[a^{<t-1>}, x^{<t>}] + b_a))$$

Andrew Ng

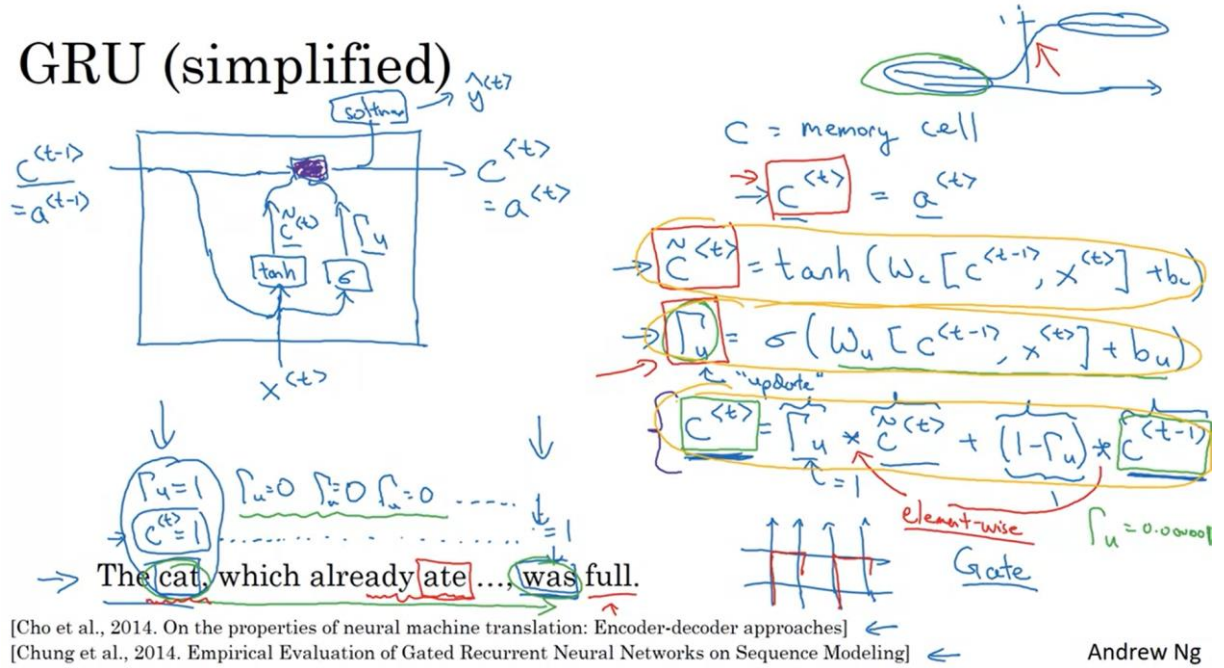
Bu fikri tekrar ettik ki GRU unit'i daha iyi anlayabilelim, GRU'nun denklemlerini ve çizimini de göreceğiz.

GRU unit'i anlamak için daha önce bahsedilen "The cat whicl already ate ... was full" cümlesini motivating example olarak da kullanacağız, daha önceki RNN yapısının problemi böyle bir cümlede "cat" kelimesini gördükten sonra cümlenin sonunda was mı were mi kullanacağına karar veremeydi, yani cümlenin başında "cat" veya "cats" olmasının onun için çok da bir farkı yoktu çünkü cümlenin sonunda "was" veya "were" kelimesini seçmesini etkileyen faktörler genelde ondan bir veya birkaç önceki input outputlardı.

Yani bir unit'in çıktısı ancak local inputlardan etkileniyordu, işte aynı sebeple vanishing gradients problemiyle karşı karşıyaydık.

GRU'nun çözeceği problem bu olacak, en başta yapılan bir değişiklik en sonu etkileyebilirse bu problem çözülmüş olacaktır.

GRU (simplified)



[Cho et al., 2014. On the properties of neural machine translation: Encoder-decoder approaches]

[Chung et al., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling]

Andrew Ng

GRU unit için yeni bir variable tanımlayacağız:

- ➔ C: memory cell bu variable'ın işlevi temelde bazı bilgileri tutmak ve timesteps boyunca taşımak olacak mesela cümlenin başındaki kelimenin tekil mi çoğul mu olduğu bilgisini tutarak sona kadar taşıyabilir böylece model was-were seçiminde bu unitten yararlanabilir.
- ➔ Aslında buradaki $c^{(t)}$ RNN'deki $a^{(t)}$ ile birebir aynı şey, ancak bu notasyonu kullanıyoruz ki LSTM konusunu daha rahat anlayalım orada $c^{(t)}$ ile $a^{(t)}$ farklı olacak.
- ➔ Her timestep için $c^{(t)}$ 'yi overwrite edebileceğimiz bir $c'^{(t)}$ değeri üretiyoruz, bunu yukarıdaki formülle üretiyoruz. Denklem aynı $a^{(t)}$ denklemine benziyor.

GRU'nın önemli fikrine gelirsek: Bir gate tanımlıyoruz: Yukarıda görüldüğü gibi tanımlanan gate γ_u 0 ile 1 arasında bir değer olacak, bunu sigmoid function ile sağlıyoruz, bu gate'in genelde 0'a veya 1'e çok yakın olduğunu düşünebiliriz, çünkü sigmoid'ın tanımı gereği domain'in çoğunda output 0'a veya 1'e çok yakındır.

Yani burada olan olay şu: GRU unit input olarak “The cat” kısmını aldığı zaman diyor ki ben $c_{<t>}$ = 1 yapayım çünkü şuan bahsedilen bir subject var ve bu subject tekil. Daha sonra bu değer cümlede sonunda “was”/“were” kelimesine kadar saklanıyor, bu kelimenin seçimini yapmak için cümle boyunca saklanan $c_{<t>}$ değeri kullanılıyor ve artık bu değer unutulabilir.

Gate’in görevi de bu $c_{<t>}$ değerinin ne zaman update edileceğine karar vermek. Yani “The cat” kısmını gördüğünce gate diyor ki haa bu kısım önemli bunun tekil mi çoğul mu olduğunu memory cell’e atayım. Benzer şekilde “was”/“were” seçimini yaptıktan sonra da yine gate diyor ki tamam bu memory cell’i tekrar boşaltabilirim.

Buna sağlayan denklem yukarıdaki fotoğrafın sağ en altında gösterilmiş, $c_{<t>}$ her time step için hesaplanırken önce $c'_{<t>}$ hesaplanıyor aynı $a_{<t>}$ hesaplanır gibi bu hesap yapılıyor. Fakat bunun yanında bir de her timestep için gate çalışıyor, eğer bu gate 1’e yakın bir output verirse, $c_{<t>}$ almost= $c'_{<t>}$ olarak güncelleniyor, yani bildiğimiz daha önce hesaplanan $a_{<t>}$ gibi aktivasyon hesaplanmış oluyor.

Ancak bazı zamanlarda ise gate 0’a yakın bir değer hesaplar işte bu durumda da bu timestep’in aktivasyonu bir öncekine almost = olarak atılıyor.

İstersek her step için $c_{<t>}$ ’yi daha önce olduğu gibi bir softmax layer’a bağlayabilir ve output elde edebiliriz.

Yani benim yukarıda anlatılandan yaptığım intuitive çıkarım şu: Bu yapı basic RNN’e benziyor ancak şöyle bir farkı var, her timestep için bir önceki kelimeye ($y_{<t-1>}$ ve $a_{<t>}$) bakarak non-linear işlemler sonucunda vocabulary’deki her kelimeye bir probability atıyor ve en ideal kelimeyi seçiyordu. Bu yeni yapıda ise, yine her timestep için bir önceki kelimeye ($y_{<t-1>}$ ve $a_{<t>}$) bakıyor ve bir aktivasyon oluşturuyor ancak, ilgili outputunu ve diğer birime aktaracağı aktivasyonu sadece bunlar belirlemiyor, bir gate yapısı her timestep için çalışıyor eğer bu gate onay verirse bu timestepteki kelimenin etkisi bir sonraki timestep’e taşınıyor, eğer onay gelmezse, bu timestepte hiçbir işlem yapılmıyor ve bir önceki aktivasyon olduğu gibi aktarılıyor yani bir sonraki timestepte karar verirken bu kelimenin hiçbir önemi kalmıyor, iki önceki kelimeye bakarak karar verilecek.

Bir nevi GRU hafızasının kısıtlı olduğunun bilincinde ve her kelimeyi hesaba katmıyor, sadece daha sonra kelime seçiminde gerçekten önem edecek kelimeleri hesaba katıyor, ama bu benim aklıma tam da yatmıyor çünkü bence bir cümledeki 3. Kelimeye 2. Kelime olmadan karar vermek çok ama çok zor. Muhtemelen burada anlamadığım bir şeyler var.

Önemli bir notka şu: Bahsedilen yapıda $c_{<t>}$ bir vektör, yani diyelim ki 100 elemanlı bir vektör! Bu durumda $c'_{<t>}$ de 100 elemanlı bir vektör olacak ve Gamma u yani gate de 100 elemanlı bir vektör olacak.

İşte bu olayları biraz değiştirdi, bu durumda gate’in yaptığı şey önceki kelimenin etkisini tamamen silmek değil, activation units’in bazılarını bir önceki timestepden doğrudan alırken diğerlerini yenileyebilir. Yani 100 aktivasyonun 50 tanesi bir önceki timestepten gelebilir, fakat 50 tanesi şuan ki kelimeye göre oluşturulabilir böylece, hiçbir kelime tamamen etkisiz olmuş olmuyor, 100 aktivasyon unit’in her birini bir bit gibi düşün belki 1. unit bahsedilen kedinin tekil mi çoğul mu olduğunu tutuyor bu durumda cat gördüğünde bu unit yenilenecek, ancak bir was/were kısmına gelene kadar birdaha hiç yenilenmeyecek, hep bir önceki değeri alacak.

Bu süreçte diğer aktivasyon unitlerinin her biri de bir sonraki adımda seçilecek kelimeyi etkileyen bilgiler taşıyor olacak, yani bu aktivasyon unitleri sıradaki kelimenin olasılığına karar veren feature'lar!

Basic RNN ile bu feature'lar ister istemez her stepte update ediliyordu bu yüzden cümlelerin başında kullanılan cat kelimesinin tekil olduğu bilgisi cümlelerin sonuna kadar taşınamıyordu, yalnızca birkaç timestep boyunca taşınıp daha sonra yerini yeni kelimelerin oluşturduğu feature'lara bırakmak zorunda kalıyordu.

GRU yapısı ile ise, eğer bizim için değerli olduğunu düşünüyorsak, bir bilgiyi daha uzun timestepler boyunca taşıyabiliyoruz, işte bu da basic RNN'in problemi olan long-term dependency capturing'i çözmüş oluyor.

Yukarıda gösterilen ve açıklanan denklemler aslında Simplified GRU'ya ait. Full GRU denklemleri aşağıda görüldüğü gibi biraz daha farklıdır. Altında yatan fikir aynı ancak, güncel araştırmalara göre bu denklemler daha iyi sonuçlar almamızı sağlıyor.

Full GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

LSTM

The cat, which ate already, was full.

Andrew Ng

Vid 10

Long Short Term Memory (LSTM)

Bir önceki kısımda GRU yapısını gördük ve bu yapının, bir sequence içinde long term connections'ı nasıl sağladığını anladık.

Böyle bağlantıları yapmayı sağlayan bir diğer unit ise LSTM unitidir. Bu yapı GRU'dan bile daha güçlüdür.

Bu kısımda buna bakacağız.

- Aşağıda solda bir önceki kısımda gördüğümüz GRU equations yer alıyor.
- LSTM GRU'nın biraz daha güçlü ve genelleştirilmiş hali olarak düşünülebilir.
- Sağda da LSTM denklemleri görülüyor, artık $c^{<t>} = a^{<t>}$ demiyoruz, $a^{<t>}$ ve $c^{<t>}$ başka kavramları ifade edecek.
- Önceki benzer şekilde bir $c'^{<t>}$ ve update gate tanımlı.
- Update Gate'in yanında bir Forget Gate ve Output Gate de aşağıdaki gibi tanımlı.
- Önceki $c^{<t>}$ denkleminde olay şuydu, her timestep için bir unite karşılık gelen eğer gate update'e karar verdiyse update yapılıyordu, eğer vermediyse update yapılmıyordu ve bir önceki değer taşınıyordu, LSTM için ise forget gate işin içine girdi ve artık c' ile update yapılabileceği gibi hem update eski değer üzerine de eklenebilir hale geliyor.
- Benzer şekilde artık $c^{<t>}$ ile $a^{<t>}$ aynı olmaktan çıkıyor ve aşağıda görüldüğü gibi $a^{<t>}$ 'nin hesaplanmasına output gate de dahil oluyor.

GRU and LSTM

GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$a^{<t>} = c^{<t>}$

LSTM

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \quad (\text{update})$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \quad (\text{forget})$$

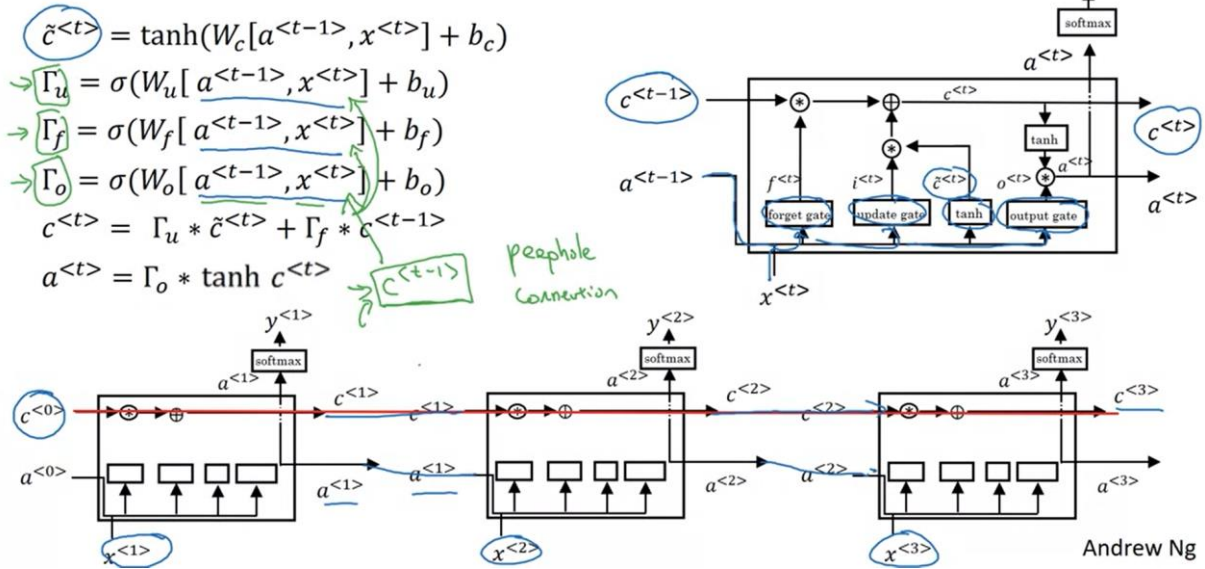
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \quad (\text{output})$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * c^{<t>}$$

Bu denklemleri diyagramla göstermek istersek aşağıdaki ilk diyagrama bakabiliriz.

LSTM in pictures



LSTM yapısı ile certain values can be stored in the memory cell even for many many timesteps.

LSTM ile GRU karşılaştırıldığı zaman biri daha iyi diyemeyiz, bazı problemlerde biri daha iyi çalışırken başka problemlerde diğeri daha iyi performans sergileyebilir.

GRU LSTM'e göre daha basit bir model, bu yüzden daha büyük networkler oluşturulabilir, computationally runs a bit faster. Ancak LSTM'de 2 yerine 3 gate olduğu için daha complex ve daha güçlü diyebiliriz.

Birini rastgele seçeceksen LSTM'i seç, daha iyi sonuç alırsın. Son yıllarda GRU'da daha fazla kullanılmaya başlıyor.

İkisininde temel amacı, long range dependency yakalayabilmek. Yani cümlelerin sonunda gelecek kelimeyi belirlerken veya bir sonraki fiyat değerini belirlerken sadece birkaç timestep önceki değerlere göre değil, tüm zamanları daha iyi gözeterek gerekirse çok eskide kalmış bir değeri de önem arzettiğini düşünüyorsa hesaba katarak işlem yapar.

Vid 11

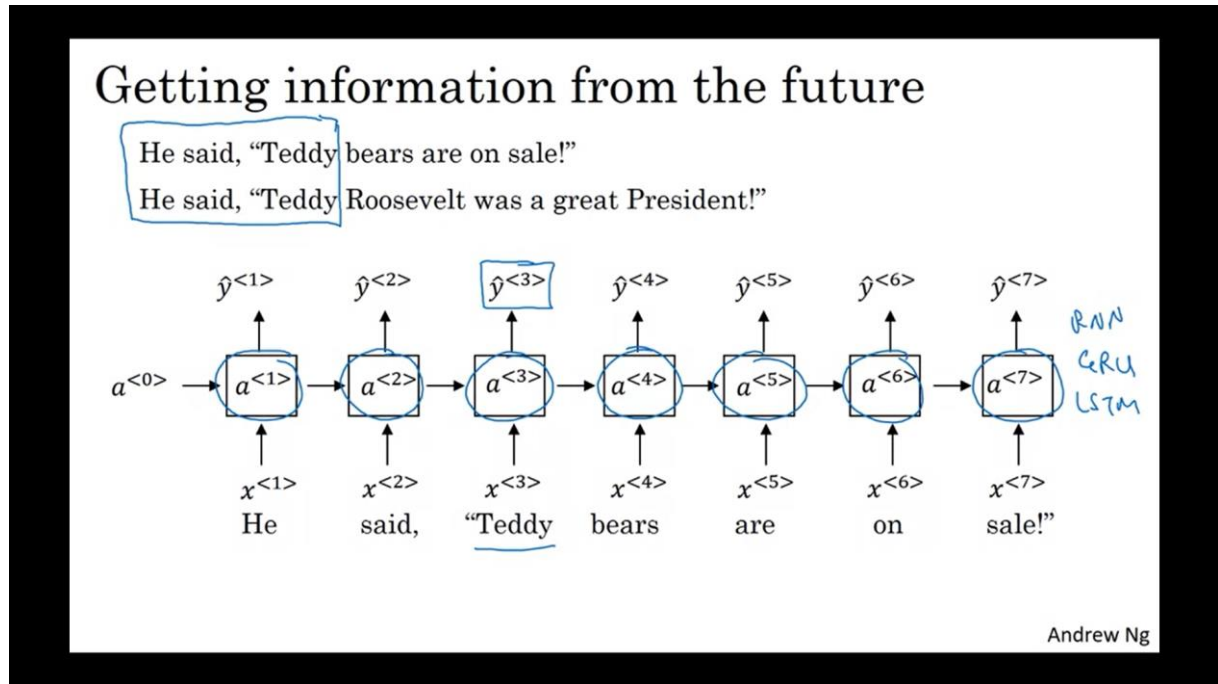
Bidirectional RNN

Şimdiye kadar we've seen most of the cheap building blocks of RNNs. But there are 2 more ideas that let us build much more powerful models.

Bir tanesi Bidirectional RNN, diğeri de Deep RNNs bu başlıkta ilk fikirden bahsedeceğiz.

Motivating problem olarak, daha önce gördüğümüz Name Entity Recognition problemini düşünelim.

- Aşağıdaki gibi bir network ile verilen cümledeki kelimelerin kişi ismi olup olmadığını anlamaya çalışıyoruz diyelim.
- Ancak şimdiye kardığımız modeller, aşağıda görüldüğü gibi iki cümlede Teddy kelimesinin özel isim olup olmadığını söylemekte başarısız olacaklardır çünkü şimdiye kadar gördüğümüz model timestep 3 için sadece ilk 3 timestep verisini kullanıyor, yani model sadece "He said Teddy" kısmını görüyor ve Teddy'nin kişi ismi olup olmadığını anlamaya çalışıyor cümlelerin geri kalanını hesaba katmadan bu iki cümleye aynı yaklaşacak.
- İşte bu problem Uni-directional RNN/GRU/LSTM'in problemi.

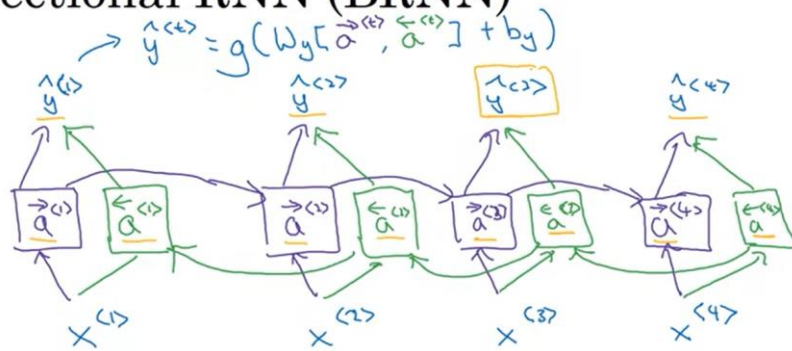


Biz istiyoruz ki modelimiz 3. Kelime'nin outputuna karar verirken cümlelerin kalanından da yararlanabilsin, bu noktada BRNN işin içine giriyor.

Bidirectional RNN'in çalışmasını aşağıda açıklayalım.

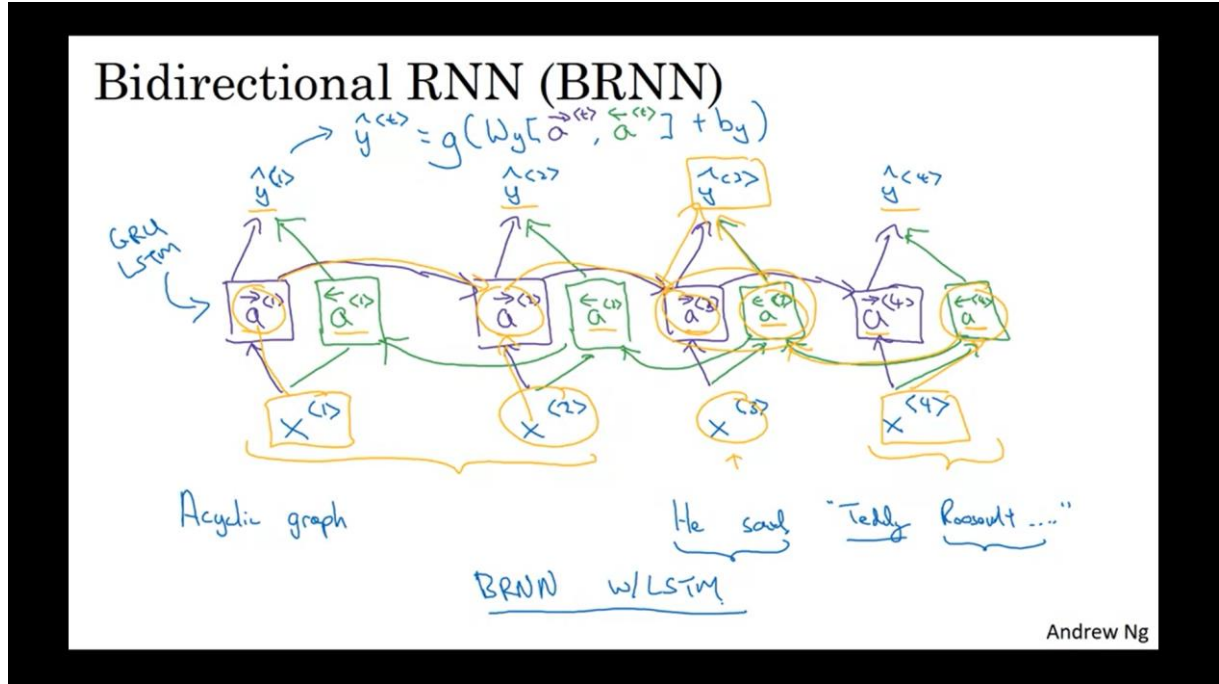
- Örnek olması için 4 kelimeli bir cümlenin input olarak kullanıldığını varsayalım.
- Aşağıda görülen mor yapı, bildiğimiz RNN'in aynısı, her timestepde inputu ve bir önceki timestep aktivasyonu ile output üretiyor.
- Bu yapıya bir de aynı yapının backward çalışanını ekliyoruz, yani 4 kelimelik bir input geldiğinde bildiğimiz yapı önce ilk kelimeyi ve zero inputu alıp a_1 'i hesaplarken bir de backward yapı ile x_4 alınıp a_4 hesaplanıyor biri ileri doğru hesaplama yaparken diğeri de geriye doğru hesaplama yapıyor.
- Daha sonra output'lar hesaplanırken bu iki hesaplama da kullanılıyor. Böylelikle ne sağlanmış oluyor, bir output hesaplanırken aslında inputun tamamı göz önünde bulundurulmuş oluyor, $y_{<t>}$ 'nin hesaplamasına bakarsak hesabın bir kısmı düz hesaplamadan geliyor bir kısmı ise ters hesaplamadan, örneğin 2. Unit için output hesaplanıyorsa hem ilk iki kelime dikkate alınıyor hem de backward hesap ile sondan gelen iki kelime hesaba katılıyor.

Bidirectional RNN (BRNN)



BRNN'in problemi nasıl çözdüğünü anlamak için 4 kelimelik bir cümlede 3. Kelimenin çıkışına nasıl karar verdiğine bakalım:

- Önce x1 ile a1 hesaplanıyor, a1 ve x2 ile a2 hesaplanıyor ardından a2 ve x3 ile a3 hesaplanıyor. Ters taraftanda x4 ile a4 hesaplanıyor ve a4 ile x3 kullanılarak bir a3 daha hesaplanıyor. Sonucunda bu iki a3'te y<3> hesabında devreye giriyor ve karar verilirken tüm input gözetilmiş oluyor.



Yukarıda anlatılan konunun sadece RNN için değil aynı zamanda GRU ve LSTM için de geçerli olduğunu unutma, aynı mantıkla GRU ve LSTM yapılarını da BiDirectional'a çevirebiliriz.

NLP problemlerinin çoğunda BRNN with LSTM blocks are pretty reasonable first thing to try.

Tabi karar vermek için tüm cümleyi almak dezavantajda olabilir, real time speech recognition uygulamalarında cümleyi çevirmek için bitmesini beklemek zaman kaybına yol açabilir.

Vid 12

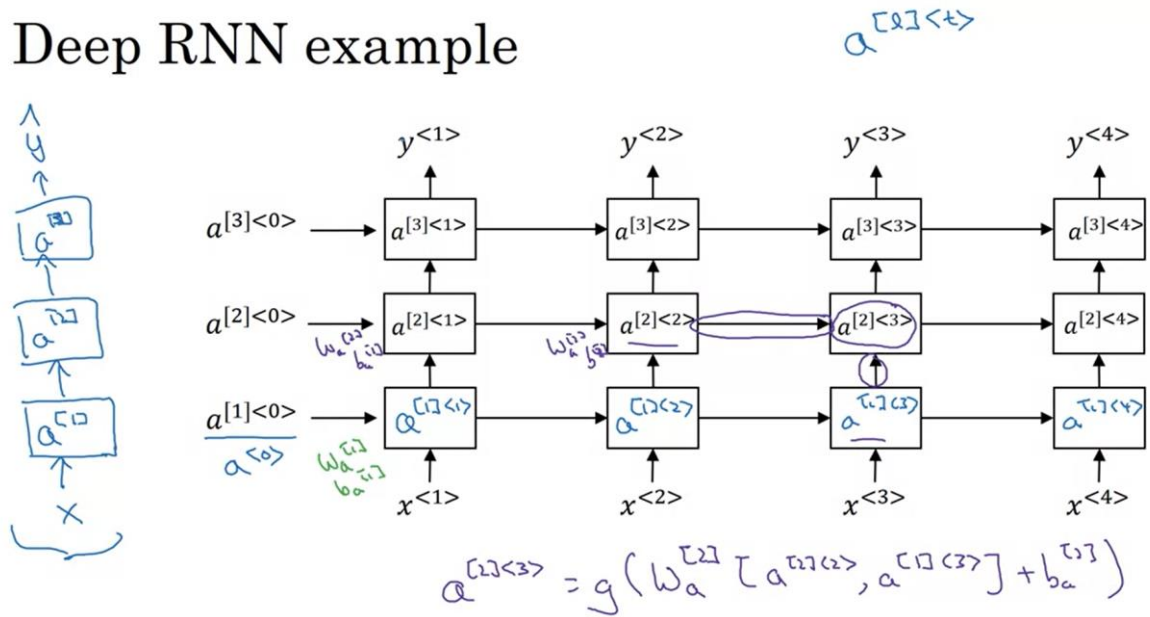
Deep RNNs

Şimdiye kadar gördüğümüz farklı RNN tipleri anlatıldığı şekliyle gayet iyi çalışır. Ancak bazen complex functions öğrenebilmek için birden fazla RNN layer'ını birleştirerek deeper RNNs oluşturmak işe yarar.

Bu son başlıkta deeper RNNs'i nasıl build edebileceğimizi göreceğiz.

- Standard bir NN için sol altta görüldüğü gibi bir aktivasyonun önce layer1 ie bir $a[1]$ aktivasyonuna çevrildiğini daha sonra sıradaki layer ile $a[2]$ aktivasyonu elde edildiğini bu şekilde en sonunda output elde edildiğini biliyoruz.
- Deep RNN yapısı da aynı mantık, aşağıda görüldüğü gibi timestep 1 için artık sadece $a[0]$ yok 3 farklı layer var, önce $a[1]$ hesaplanıyor daha sonra $a[2]$ hesaplanıyor sonra $a[3]$ hesaplanıyor ve ancak bundan sonra $y[1]$ hesaplanıyor bu ne sağlıyor, timestep1 için daha complex fonksiyonlar öğrenebilmeyi sağlıyor.
- Daha sonra timestep 2'ye geçerken de bağlantıları görüyorsun, hem vertical hem horizontal bağlantılar söz konusu.
- Bu yapı şimdilik sadece birkaç katman oluyor, zaten RNNs'in horizontal time bağlantısı yeterince deep bir RNN oluşturuyordu bir de bunu ekleyince işler iyice derinleşiyor.

Deep RNN example

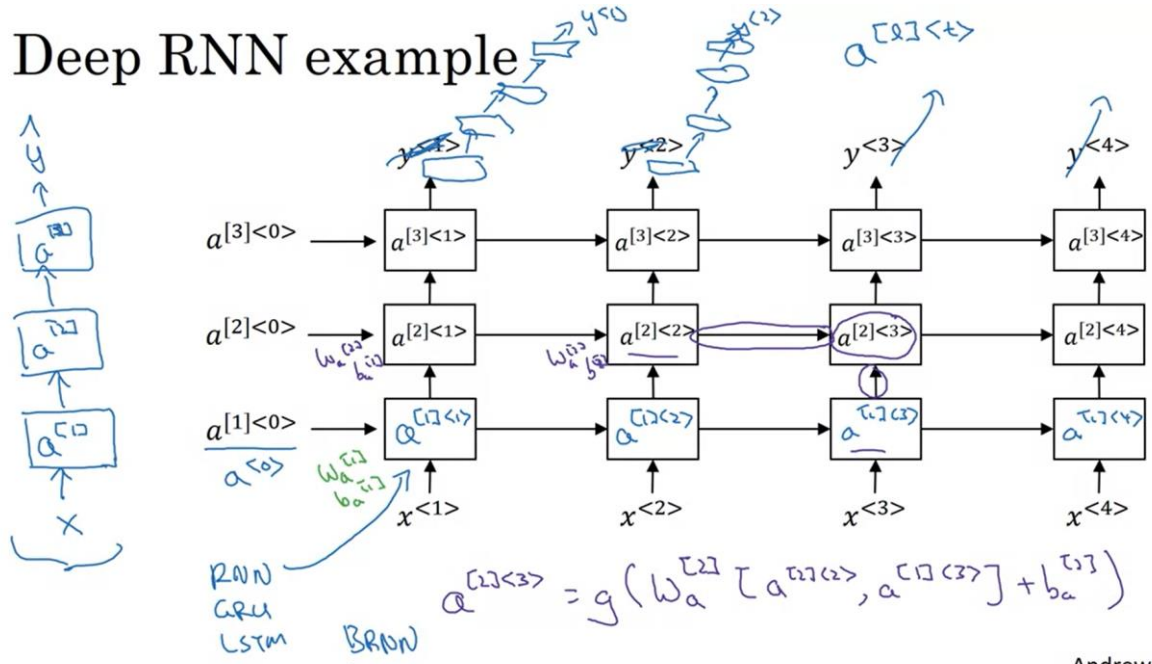


Andrew Ng

Her horizontal layer'ın weightlerinin birbirinden farklı olduğuna dikkat et.

Son olarak bazı yapılarda aşağıda görüldüğü gibi output hesaplanmadan önce sadece vertical networkler kullanılır, yani bir nevi output hesaplanmadan önce bir standard deep NN uygulamış oluyoruz bu da kullanılabilen bi yapı.

Deep RNN example



Andrew Ng

Böylelikle RNNs konusunu tamamlamış olduk!