

①

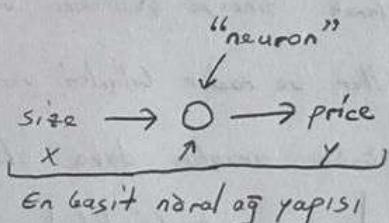
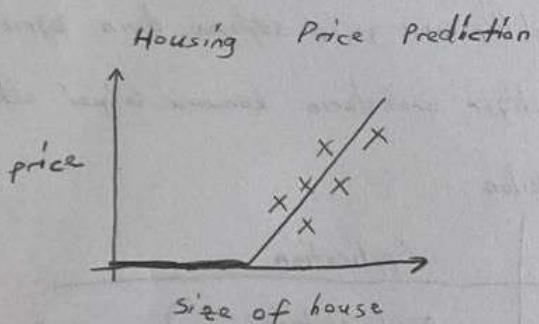
## Deep Learning Specializations

### I. Neural Networks and Deep Learning

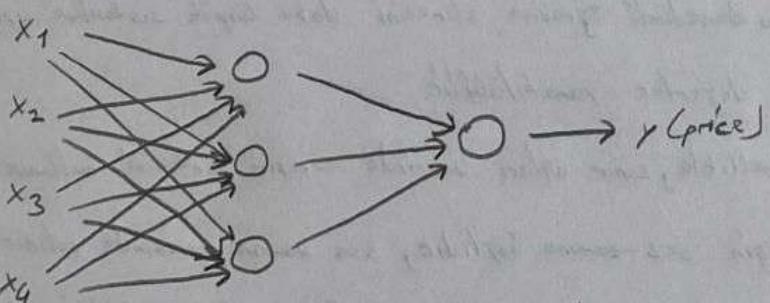
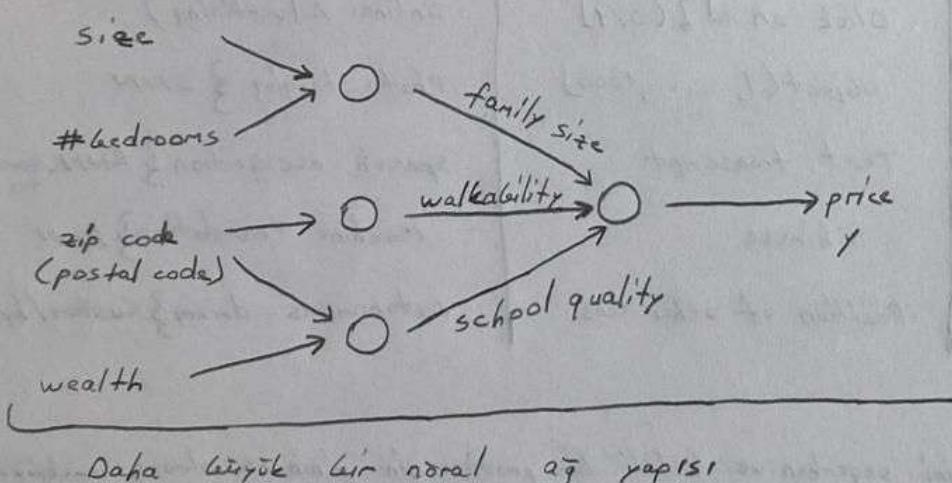
Week 1:

Özürin öğrenme internette arama ve reklamcılık gibi geleneksel internet işlerini değiştirdi. Sağlık hizmetlerinin geliştirilmesi, x-ray okuma, hassas tarama, sürücüsüz araçlar ve binzeri girişek alana yayılabilen potansiyel sahip bir alan haline geldi.

AI'ye yeni elektrik diberildi. Elektriğin dünyaya girmesi ile girişek alanda ciddi değişiklikler yaşandı. Yapay zeka da bu potansiyel var diperabiliti ve özürin öğrenme de yapay zekanın önemli parçalarından biridir.



ReLU  
Rectified Linear Unit



Bu yapıda her input bütün nöronlara ulaşır. Fakat hepsini etkilemez. Etkileşimlerini de aynı derecede etkilemek zorunda değildir.

(2)

$x$  ve  $y$  ile ilgili yeterli verinin olması ve  $x$  ile  $y$ 'nin birlikte olduğu eğitim örnekleri verildiğinde sinir ağları  $x$ 'den  $y$ 'ye doğru gelebilde eşleştirilen işlevleri bulmakta olduğuna basarılıdır.

### Supervised Learning with Neural Networks:

Denetimli öğrenme önceliği bir kism verilen  $x$  ve  $y$ lere göre uygulamaların geliştirilmesinde kullanılır. Birçok alanda çok önceliği olan reklamcılık alanında çok önemlidir ve çok kazançlıdır. Kullanıcı profillerinin özelliklerine göre tüketicinin hangi reklamlara tıklayıp tıklamayacağı, antar ve bu sayede şirketlerin reklam gelirlerini düzenlemek ve harcamalarına net bir etkisi olur. Bilgisayarlı gören çok geliştiği için herhangi bir resim seçip 1000 farklı resinden hangisi olduğunu bilmesi istenildi. Input olarak ses veya çıktı olarak text alan bir yapı geliştirilebilir. İngilizce bir cümle seçip çıktı olarak içindeki kelimesini ve en diller arası geçişini sağlayan bir öğrenme uygulaması oluşturulur. Görüşeller ve radar bilgileri verilip diğer arabaların konumunu bilgisini edebilir ve bu sonucunda aragari daha oturan tıkla.

Input( $x$ )	Output( $y$ )	Application
Home features	Price	real estate
Ad, user info	Click on ad? (0/1)	online Advertising
Image	object(1, ..., 1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine Translation
Image, Radar Info	Position of other cars	Autonomous driving

$x$ 'in ne olması gerektiğini sezerken ve belirli bir problem için  $y$ 'nın ne olması gerektiğini akıllıca sezerken, ardından bu denetimli öğrenme sürecini daha büyük sistemlere uygulayarak sinir ağları ile bağlantılu değerler产生abilir.

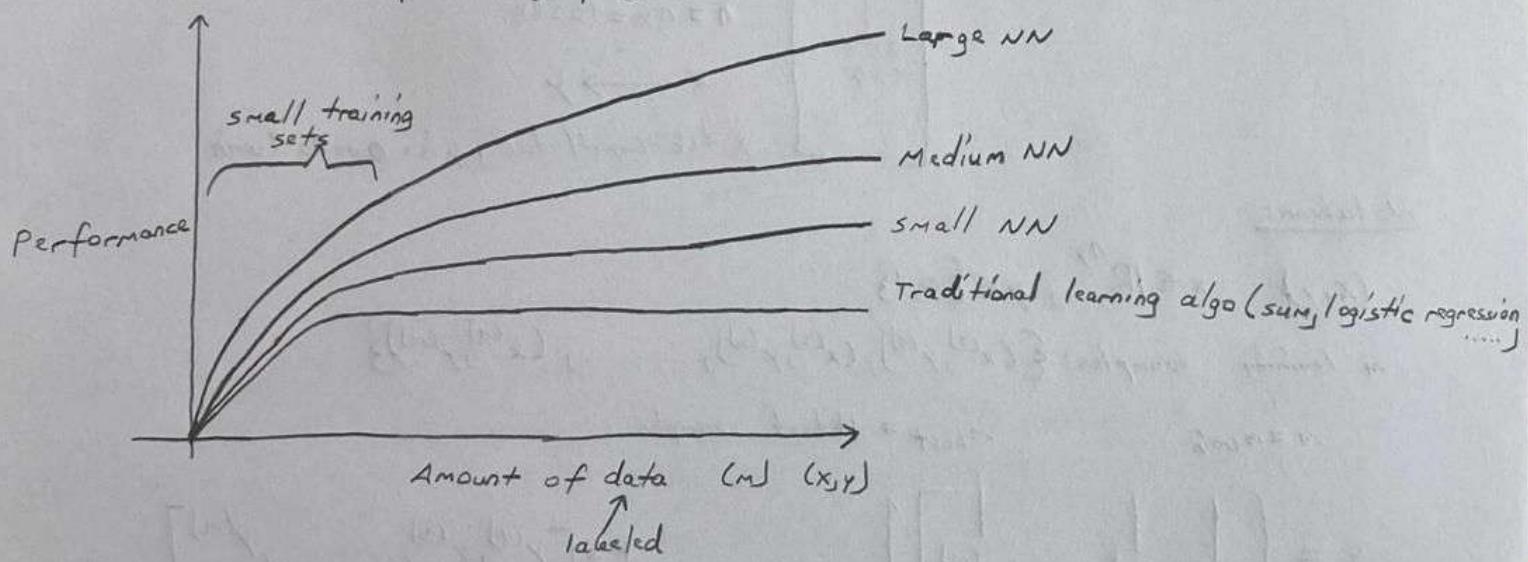
Görüntü uygulamaları için genellikle sinir ağları üzerinde endüstriyel panterini kullanır (CNN). Sıralı veri tipleri için önceden ses-zamana bağlıdır, ses zaman serisinde solunır bu yüzden de ses 1 boyutlu zaman serisi olarak (time / temporal) gösterilir. Sıralı veri tipleri için RNN kullanılır. Dilden dile geçtiğinde, RNN alfabeler ve kelimeler ard arda sıralanır.

(3)

Structured Data veritabanı sistemlerine benzer. Verilecek x inputları ve inputlara göre y sonucu net bir şekilde belli'dir.

Unstructured Data ise daha çok audio, image, text gibi bilgisayarlar için her zaman structured datayı anlamak ve işlem yapmak unstructured dataya göre çok daha kolay olmuştur. İnsanlık tarihi boyunca insanlar da görseli, yazılımları anlamada ve yorumlamada sürekli bir gelişim yaşamıştır. Bilgisayarlar da nöral ağlar sayesinde bu forte verileri anlamada ve yorumlamada hızla bir aşama katediyor. Bu da günümüz dünyası için büyük bir hedefendir. Fakat günümüzde unstructured data tabanlı bilgisayarın kendi kendisi gidiş önelebilsese structured data tabanlı, rehberlik, şirket kararları gibi hem ekonomi tabanlı hem de insanların karar almadada, anlamada zorlandığı seyler data çok önemseniyor.

\*\* Scale drives deep learning progress



Internet'in yaygınlaşması ile birlikte veri çok fazla arttı ve geleneksel yolların fazla veriyi çok da iyi anlamadığı anlaşıldı. Bu süreçten sonra fazla veride data iyi sonuç veren modeller geliştirildi ve CNN'de kadar uzandı. Kırık verisetlerinde ise hangi modelin data iyi sonucunu vereceğini bilinmez (görsel yanıltıcı) modelleri veriseti için test etmemiz gerekiyor. Bu sırada boyunca Data dışında Computation (İşlem, hesap) ve Algorithms önecli bir hal aldı. İşlem hızını artırmak için sigmoid function yerine ReLU kullanılmaya başlandı. Sigmoid function negatif ve pozitifde yataş eksenle geliyor ve bu optimizm problemesi demek ReLU'da ise her pozitif veri için +1 fayda anlamına gelir.

(4)

Algoritmaların daha hızlı çalışması data büyük sınırları aşabilemenize ya da sınırları kabul edilebilir zaman içerisinde çalıştırmanızı sağlar.

### ~~Week 2:~~

Bu kısımda nöral ağların nasıl programlanacağı, üzerinde bilgiler verilir. Genellikle 2 tür vardır forward propagation ve backward propagation olmak üzere incelenir.

Logistic regression ikili sınıflandırma için kullanılır (kedi ya da kedi değil gibi)

Bir görseli döşenirsek bilgisayarında 3 katmanlı şekilde sınıflandırılır. (RGB)

64x64 bir görselinin olduğunu düşünelim. Buunla işlem yapmak için bir matris kullanırız ve buin değerlerini gireriz, ardından verilen inputa göre y değerleri elde edilir.

Örnek matris

$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

RGB  
↑  
 $64 \times 64 \times 3 = 12288$   
 $n = n_x = 12288$   
 $x \rightarrow y$   
 $x$  tek boyutluur,  $y$  ise oxada 1 varır

### Notation:

$$(x, y) \quad x \in \mathbb{R}^{n_x}, y \in \{0, 1\}$$

$$m \text{ training examples: } \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$$m = m_{\text{train}}$$

$$m_{\text{test}} = \# \text{test examples}$$

$$x = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | \end{bmatrix} \begin{matrix} \uparrow \\ n_x \\ \downarrow \end{matrix}$$

$y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$   
 $y \in \mathbb{R}^{1 \times m}$   
 $y.\text{shape} = (1, m)$

$$x \in \mathbb{R}^{n_x \times m}$$

$$x.\text{shape} = (n_x, m)$$

(5)

## Logistic Regression

Logistic regression ibili sınıflandırma için kullanılır. Verilen bir kedi resmi için muhtemel iki sonuc söylemek (kedi ya da değil)

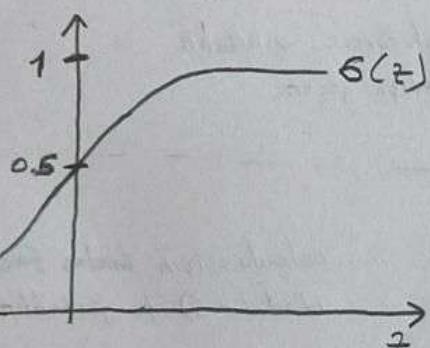
Given  $x$ , want  $\hat{y} = P(y=1|x)$

$$x \in R^n, \quad 0 \leq \hat{y} \leq 1$$

Parameters:  $w \in R^{n \times 1}$ ,  $b \in R$

Output  $\hat{y} = w^T x + b \rightarrow$  Bu formül genelde linear regression için kullanılır sadece 1'den büyük ya da negatif değerler verebilir. Fakat  $w$  0,1 aralığında değer istiyoruz.

Bunun için  $\hat{y} = \sigma(w^T x + b)$



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

- z büyük pozitif bir sayı ise cevap 1'e yaklaşır.
- z büyük negatif bir sayı ise cevap 0'a yaklaşır.

Alternatif notasyon: (kullanılmayacak sadice f'tır)

$$x_0 = 1, \quad x \in R^{n+1}$$

$$\hat{y} = \sigma(\theta^T x)$$

Bu şekilde notasyonlar da kullanılabilir. Fakat üstteki ile devam edilecektir.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_{n+1} \end{bmatrix} \quad w$$

Logistic Regression cost function:

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}} \quad z^{(i)} = w^T x^{(i)} + b$$

Given  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ , want  $\hat{y}^{(i)} \approx y^{(i)}$ .

Loss (error) function:  $L(\hat{y}, y) = \frac{1}{2} (\hat{y} - y)^2$

$$L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$$

If  $y=1$ :  $L(\hat{y}, y) = -\log \hat{y} \leftarrow$  want  $\log \hat{y}$  large, want  $\hat{y}$  large

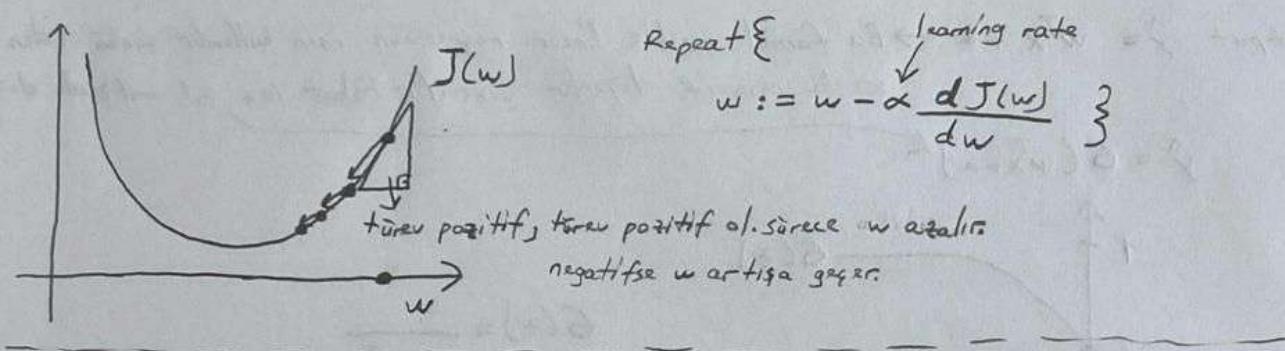
If  $y=0$ :  $L(\hat{y}, y) = -\log(1-\hat{y}) \leftarrow$  want  $\log(1-\hat{y})$  large, want  $\hat{y}$  small

(6)

$$\text{Cost function} = J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m [\hat{y}^{(i)} \log \hat{y}^{(i)} + (1 - \hat{y}^{(i)}) \log (1 - \hat{y}^{(i)})]$$

### Gradient Descent:

Bu metod cost function'ın 3 boyutlu düzlemdeki değerini minimize etmeye çalışır ve de minimum değerini bulmaya çalışır. 3 boyutlu çözüm zor olduğu için öncelikle biraz daha basit hali tip 2 boyutlu düşünülmeli.



3 boyutlu

$$J(w, b) = w - \alpha \frac{d J(w, b)}{d w}$$

$$b - \alpha \frac{d J(w, b)}{d b}$$

calculus için minden fazla değişkenle sahip ifadelerde  $\partial/\partial$  gösterilir. Parçalı türev anlamına gelir.

### Computation Graph:

$$J(a, b, c) = 3(a + b c)$$

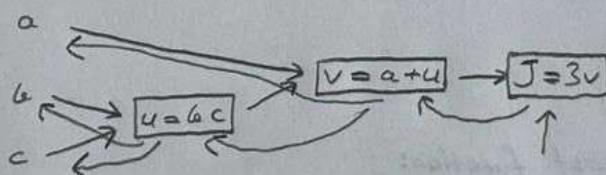
$$\underbrace{\quad\quad\quad}_{v} \quad \underbrace{u = b c}_{u} \quad \underbrace{v = a + u}_{v}$$

$$\underbrace{\quad\quad\quad}_{J} \quad \quad \quad \quad J = 3v$$

$$u = b c$$

$$v = a + u$$

$$J = 3v$$



Computation graph'ları doğrudan ve geriye doğru işlem için iyi bir fikirdir. İleri doğru yaptığıınız işlemlerini kenarlar bir şekilde tersten yapınız. Tersten türler hesaplanır. Bundan sonra kısında daha detaylı bir şekilde computation graph ile türler hesabı var.

(7)

### Computing Derivatives:

$$a = 5 \rightarrow v = a + u \rightarrow J = 3v$$

$$b = 3 \rightarrow u = bu \rightarrow J = 3v$$

$$c = 2 \rightarrow J = 3v$$

$$\left. \begin{array}{l} J = 3v \\ v = 11 \rightarrow 11.001 \\ J = 33 \rightarrow 33.003 \end{array} \right\} \frac{dJ}{dv} = 3$$

$$\left. \begin{array}{l} a = 5 \rightarrow 5.001 \\ v = 11 \rightarrow 11.001 \\ J = 33 \rightarrow 33.003 \end{array} \right\} \frac{dJ}{da} = 3$$

a → v → J       $\frac{dJ}{dv} \cdot \frac{dv}{da} = 3 \cdot 1 = 3$

same

$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{du}$$

$$\frac{dI}{db} = \frac{dI}{du} \cdot \frac{du}{db} = 6$$

$$b = 3 \rightarrow 3.001$$

$$u = b \cdot c = 6 \rightarrow 6.002 \quad c = 2$$

$$\frac{dI}{dc} = \frac{dI}{du} \cdot \frac{du}{dc} = 9$$

Bu gösterim hem ileri hem geri işlem için taraç almayı öneklemdir.

### Logistic Regression Gradient Descent:

$$z = w^T x + b \quad , \quad \hat{y} = a = \sigma(z) \quad , \quad L(a, y) = -[y \log a + (1-y) \log(1-a)]$$

$$\begin{aligned} x_1 & \rightarrow z = w_1 x_1 + w_2 x_2 + b \rightarrow \hat{y} = a = \sigma(z) \rightarrow L(a, y) \\ w_1 & \\ x_2 & \\ w_2 & \\ b & \end{aligned}$$

$$\begin{aligned} dz &= \frac{dL}{dz} = \frac{dL(a, y)}{dz} \\ &= a - y \\ &= \frac{dL}{da} \cdot \frac{da}{dz} \\ &= -\frac{y}{a} + \frac{1-y}{1-a} \end{aligned}$$

$$\frac{\partial L}{\partial w_1} = "dw_1" = x_1 \cdot dz \quad , \quad dw_2 = x_2 \cdot dz \quad , \quad db = dz$$

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

Bu islemler bir öncegi göre bir derecelik adım, tensil eder.

8

### Gradient descent on m examples:

Bir önceki kısımda tek bir örnek için yapılabilecek işlemlerini gördük. Bu kısımda ise birden çok örnekte nasıl işlem yapılacağını göreceğiz.

$$\begin{aligned} J(w, b) &= \frac{1}{m} \sum_{i=1}^m L(a^{(i)}, y^{(i)}) \\ \rightarrow a^{(i)} &= \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b) \\ \frac{\partial}{\partial w_1} J(w, b) &= \frac{1}{m} \sum_{i=1}^m \underbrace{\frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})}_{d w_1^{(i)} = (x^{(i)}, y^{(i)})} \end{aligned}$$

Programlama kısmı:

$$J = 0; dw_1 = 0; dw_2 = 0; db = 0$$

for  $i := 1$  to  $m$

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 += x_1^{(i)} \cdot dz^{(i)} \quad \uparrow n=2 \text{ için}$$

$$dw_2 += x_2^{(i)} \cdot dz^{(i)} \quad \downarrow$$

$$db += dz^{(i)}$$

$$dw_1 = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

$$J / m$$

$$dw_1 / m; dw_2 / m; db / m$$

Denim öğrenme uygulamalarında for loop kullanmak "şunu" söylemek isterse for loop kullanmak ise faktikten denetimde örnekte 2 tane  $w$  değeri vardı, daha fazlası oldugunda for içinde 2. bir for döngüsü denilebilir. Bu durumdan kurtulmak için vectorization işlemleri kullanılır ve bu işlemler for kullanmaya gerek bile kalmadan hizlica işlem yapabilmek.

Bundan sonrası kısımda vec for ile işlem hızlanacaktır.

### Vectorization:

$$z = w^T x + b$$

$$w = \begin{bmatrix} | \\ | \\ | \end{bmatrix} \quad x = \begin{bmatrix} | \\ | \\ | \end{bmatrix}$$

$$\begin{array}{l} w \in \mathbb{R}^{n \times 1} \\ x \in \mathbb{R}^{n \times 1} \end{array}$$

non-vectorized:

$$z = 0$$

for  $i$  in range( $n-x$ ):

$$z += w[i] * x[i]$$

$$z += b$$

return z

$$z = np.dot(w, x) + b$$

hızlı

CPU } SIMD-single instruction  
GPU } multiple data

(9)

### Vectorizing Logistic Regression:

$$z^{(1)} = w^T x^{(1)} + b$$

$$\alpha^{(1)} = \sigma(z^{(1)})$$

$$z^{(2)} = w^T x^{(2)} + b$$

$$\alpha^{(2)} = \sigma(z^{(2)})$$

$$z^{(3)} = w^T x^{(3)} + b$$

$$\alpha^{(3)} = \sigma(z^{(3)})$$

$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | \end{bmatrix} \quad R^{n \times m}$$

$$Z = [z^{(1)} z^{(2)} \dots z^{(m)}] = w^T X + [b \ b \ \dots \ b] = [w^T x^{(1)} + b, w^T x^{(2)} + b, \dots, w^T x^{(m)} + b]$$

$\rightarrow z = np.dot(w, T, x) + b$

 $\uparrow (1,1)$ Burdaklı  $b$   $(1,1)$  olmasında rağmen python otomatik  $(1,m)$ 

matrix olarak alımlarla ve hata vermez. Bu da "broadcasting" denir.

$$A = [\alpha^{(1)} \ \alpha^{(2)} \ \dots \ \alpha^{(m)}] = \sigma(z)$$

$$dz^{(1)} = \alpha^{(1)} - y^{(1)}, \quad dz^{(2)} = \alpha^{(2)} - y^{(2)}$$

$$dz = [dz^{(1)} \ dz^{(2)} \ \dots \ dz^{(m)}]$$

$$A = [\alpha^{(1)} \ \dots \ \alpha^{(m)}]$$

$$y = [y^{(1)} \ \dots \ y^{(m)}]$$

$$dz = A - y = [\alpha^{(1)} - y^{(1)}, \alpha^{(2)} - y^{(2)}, \dots]$$

$$dw = 0$$

$$dw += x^{(1)} dz^{(1)}$$

$$db = 0$$

$$db += dz^{(1)}$$

$$dw += x^{(m)} dz^{(m)}$$

$$db += dz^{(m)}$$

$$dw / m$$

$$db / m$$

non-vectorization

$$dw = \frac{1}{m} \sum_{i=1}^m dz^{(i)}$$

$$= \frac{1}{m} np.sum(dz)$$

$$dw = \frac{1}{m} \times dz$$

$$= \frac{1}{m} \left[ \begin{bmatrix} | & | & | \\ x^{(1)} & \dots & x^{(m)} \\ | & | & | \end{bmatrix} \right] \left[ \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix} \right]$$

$$= \frac{1}{m} \left[ x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)} \right]$$

vectorization

Şimdi kader öğrenme kriterini kullanarak for loop olmadan ilk tane güncellemeyi yapınız.

$$z = np.dot(w, T, x) + b$$

$$A = \sigma(z)$$

$$dz = A - y$$

$$dw = \frac{1}{m} \times dz^T$$

$$db = \frac{1}{m} np.sum(dz)$$

$$w := w - \alpha dw$$

$$b := b - \alpha db$$

(10)

## Broadcasting in Python:

Broadcasting python kodunun daha hızlı çalışması için kullanlan bir başka tekniktir.

### Example

Calories from Carbs, Proteins, Fats in 100g of different foods:

	Carbs	Protein	Fat	
Apple	56.0	1.2	1.8	
Beef	0.0	104.0	135.0	
Eggs	4.4	52.0	99.0	
Potatoes	68.0	8.0	0.9	
				= A (3,4)

59 cal

$$\frac{56}{59} \approx 94.9\%$$

Carb, protein ve yağ için kalori değerini hesaplamak istiyorsunuz. Bunu for-loop kullanmadan nasıl yaparsınız?

ilk yapacağınız şey bu sütunları nasıl 4 parçaya ayıracığınızı öğrenmek daha sonra da python kodıyla işlerini doldurmak.

```
import numpy as np
```

```
A = np.array([[56.0, 0.0, 4.4, 68.0],
              [1.2, 104.0, 52.0, 8.0],
              [1.8, 135.0, 99.0, 0.9]])
```

```
print(A)
```

cal = A.sum(axis=0)  $\rightarrow$  axis=0 column olarak topla (dileyde)  
axis=0 row olarak topla (patayda)

```
print(cal)
```

output  $\rightarrow$  [ 59. 239. 155.4 76.9 ]

percentage = 100 \* A / cal.reshape(1,4)

```
print(percentage)
```

output  $\rightarrow$  [[ 94.91525424 0. 2.83140283 88.42652796 ]
 $\quad$  [ 2.03389831 43.51464435 33.46203346 10.40312096 ]
 $\quad$  [ 3.05084746 56.48535565 63.70656371 1.17035111 ] ]

## Farklı broadcasting örnekleri:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 100 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} = \begin{bmatrix} 101 \\ 102 \\ 103 \\ 104 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 \\ 200 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix}$$

(11)

$$\begin{matrix} a & b & c \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

### A note on Python / Numpy Vectors:

$a = np.random.randn(5)$   
 $a.shape(5,)$   
 "rank 1 array"

} Don't use

Bu ömrekti oluşturur. Bu kismin kullanılması önerilmez.

$a = np.random.randn(5, 1) \rightarrow a.shape = (5, 1)$  column vector  
 $a = np.random.randn(1, 5) \rightarrow a.shape = (1, 5)$  row vector  
 $\text{assert}(a.shape == (5, 1)) \rightarrow$  bu fonksiyon vektörün yapısını kontrol eder.

Eğer bir şekilde bir dizi elde etmek zorunda kaldığak su işlemi yapabiliriz:  
 $a = a.reshape((5, 1))$

Soru öğrencisi ya da kod yazan dizi ya da 1.derece bir yapı kullandığını işin çok sık hata yapar.  
 Diğer taraftan direkt 2 boyutlu yapı kullanmak sık sık assert fonksiyonu kullanmak ve daima göre reshape kullanmak çok önemlidir.

### Explanation of Logistic Regression Cost Function:

$$\hat{y} = P(y=1|x)$$

$$\text{if } y=1 : P(y|x) = \hat{y}$$

$$\text{if } y=0 : P(y|x) = 1-\hat{y}$$

$$\text{if } y=1 : P(y|x) = \hat{y} \quad \left. \begin{array}{l} \text{if } y=0 : P(y|x) = 1-\hat{y} \end{array} \right\} P(y|x)$$

$$P(y|x) = \hat{y}^y (1-\hat{y})^{(1-y)}$$

$$\text{if } y=1 : P(y|x) = \hat{y} \quad (1-\hat{y})^0$$

$$\text{if } y=0 : P(y|x) = \hat{y}^0 (1-\hat{y})^1 = 1 \times (1-\hat{y}) = 1-\hat{y}$$

$$\uparrow \log P(y|x) = \log \hat{y}^y (1-\hat{y})^{(1-y)} = y \log \hat{y} + (1-y) \log (1-\hat{y}) \\ = -\mathcal{L}(\hat{y}, y) \downarrow$$

### cost on m examples:

$$\log P(\text{labels in training set}) = \log \prod_{i=1}^m P(y^{(i)}|x^{(i)})$$

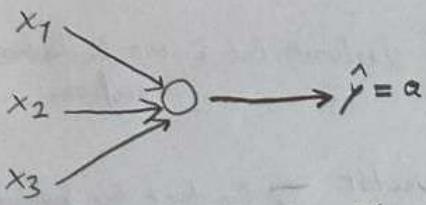
$$\log P(\dots) = \sum_{i=1}^m \underbrace{\log P(y^{(i)}|x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)}, y^{(i)})}$$

$$= -\sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

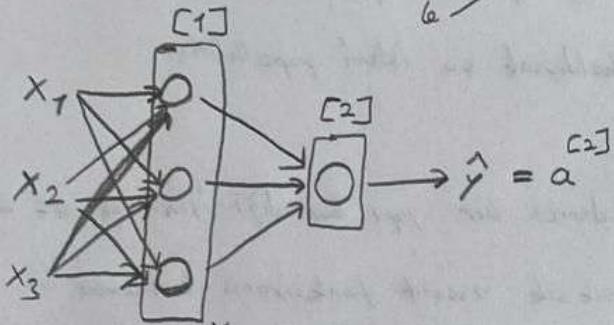
$$\text{Cost: } J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

12

### Neural Ağlara Genel Bakış:



$$x \rightarrow z = w^T x + b \rightarrow a = \sigma(z) \rightarrow L(a, y)$$



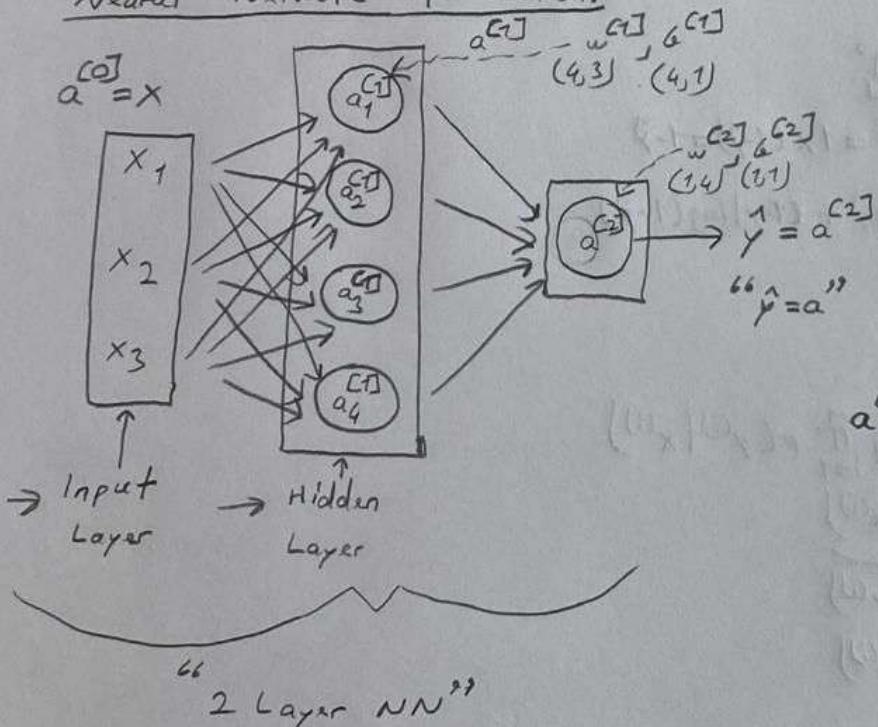
$$dz \leftarrow da \leftarrow$$

Back propagation

$$\begin{aligned} &w^{[1]} \rightarrow z^{[1]} = w^{[1]} x + b^{[1]} \rightarrow a^{[1]} = \sigma(z^{[1]}) \rightarrow z^{[2]} = w^{[2]} a^{[1]} + b^{[2]} \rightarrow a^{[2]} = \sigma(z^{[2]}) \\ &\frac{dw^{[2]}}{da^{[2]}} \quad \frac{w^{[2]}}{b^{[2]}} \quad \frac{d z^{[2]}}{d a^{[2]}} \quad \frac{d a^{[2]}}{d L(a^{[2]}, y)} \end{aligned}$$

Back propagation kismı

### Neural Network Representation:



$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$

Giriş değerleri layer sapılmaz

(13)

Computing a one hidden layer neural network's output:

Bir önceki görevdeki özetten hidden layerları önce tek tek sonra vektörlüştürme yaparak hesaplayacağımızda:

$$\begin{aligned} z_1 &= \left( w_1^T x + b_1 \right)^T, \quad a_1 = \sigma(z_1) \\ z_2 &= \left( w_2^T x + b_2 \right)^T, \quad a_2 = \sigma(z_2) \\ z_3 &= \left( w_3^T x + b_3 \right)^T, \quad a_3 = \sigma(z_3) \\ z_4 &= \left( w_4^T x + b_4 \right)^T, \quad a_4 = \sigma(z_4) \end{aligned}$$

$$z^{[1]} = \underbrace{\begin{bmatrix} (w_1^T x + b_1)^T \\ (w_2^T x + b_2)^T \\ (w_3^T x + b_3)^T \\ (w_4^T x + b_4)^T \end{bmatrix}}_{w^{[1]}} + \underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}}_{b^{[1]}} = \begin{bmatrix} (w_1^T x + b_1)^T \\ \vdots \\ (w_4^T x + b_4)^T \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$

Given input  $x: [x_0]$

$$\rightarrow z^{[1]} = w^{[1]} x + b^{[1]}$$

$(4,1) \quad (4,3) \quad (3,1) \quad (4,1)$

$$\rightarrow a^{[1]} = \sigma(z^{[1]})$$

$(4,1) \quad (4,1)$

$$\rightarrow z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$

$(1,1) \quad (1,4) \quad (4,1) \quad (1,1)$

$$\rightarrow a^{[2]} = \sigma(z^{[2]})$$

$(1,1) \quad (1,1)$

Vectorizing across multiple examples:

$$\begin{aligned} x \rightarrow a^{[2]} &= \hat{y} \\ x^{(1)} \rightarrow a^{[2](1)} &= \hat{y}^{(1)} \\ x^{(2)} \rightarrow a^{[2](2)} &= \hat{y}^{(2)} \end{aligned}$$

$$\rightarrow \text{for } i \text{ to } m: \quad \begin{aligned} z^{[1](i)} &= w^{[1]} x^{(i)} + b^{[1]} \\ a^{[1](i)} &= \sigma(z^{[1](i)}) \\ z^{[2](i)} &= w^{[2]} a^{[1](i)} + b^{[2]} \\ a^{[2](i)} &= \sigma(z^{[2](i)}) \end{aligned}$$

$$x^{(m)} \rightarrow a^{[2](m)} = \hat{y}^{(m)}$$

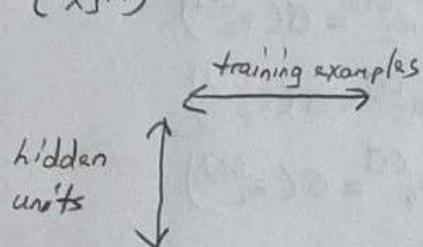
$a^{[2]}$   $\nwarrow$   $\nearrow$  example  $i$   
layer

(14)

$$X = \begin{bmatrix} | & | & | & \dots & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(m)} \\ | & | & | & \dots & | \end{bmatrix}$$

$(n \times m)$

$$z^{[1]} = \begin{bmatrix} | & | & | & \dots & | \\ z^{[1]}(1) & z^{[1]}(2) & z^{[1]}(3) & \dots & z^{[1]}(m) \\ | & | & | & \dots & | \end{bmatrix}$$



$$A^{[1]} = \begin{bmatrix} | & | & | & \dots & | \\ a^{[1]}(1) & a^{[1]}(2) & a^{[1]}(3) & \dots & a^{[1]}(m) \\ | & | & | & \dots & | \end{bmatrix}$$

Justification for vectorized implementation:

$$z^{[1]}(1) = w^{[1]} x^{(1)} + b^{[1]}, \quad z^{[1]}(2) = w^{[1]} x^{(2)} + b^{[1]}, \quad z^{[1]}(3) = w^{[1]} x^{(3)} + b^{[1]}$$

$$w^{[1]} = \begin{bmatrix} | \\ | \\ | \\ | \end{bmatrix}$$

$$w^{[1]} x^{(1)} = \begin{bmatrix} | \\ | \\ | \\ | \end{bmatrix}$$

$$w^{[1]} x^{(2)} = \begin{bmatrix} | \\ | \\ | \\ | \end{bmatrix}$$

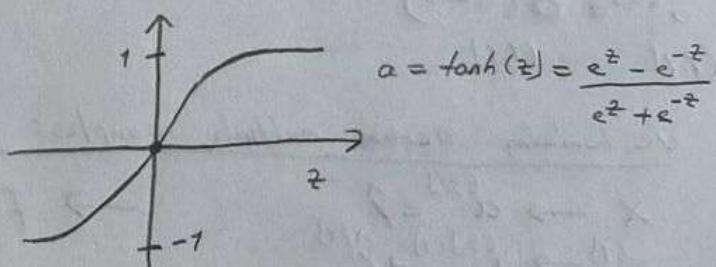
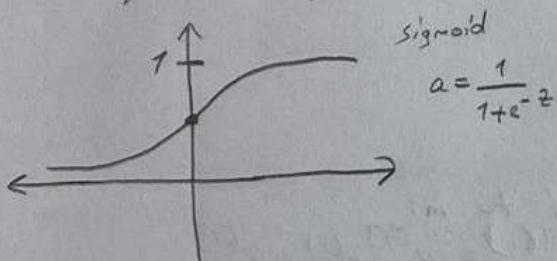
$$w^{[1]} x^{(3)} = \begin{bmatrix} | \\ | \\ | \\ | \end{bmatrix}$$

$$w^{[1]} \begin{bmatrix} | & | & | & \dots \\ x^{(1)} & x^{(2)} & x^{(3)} & \dots \\ | & | & | & \dots \end{bmatrix} = \begin{bmatrix} | & | & | & \dots \\ \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \dots \end{bmatrix} = \begin{bmatrix} | & | & | & \dots \\ z^{[1]}(1) & z^{[1]}(2) & z^{[1]}(3) & \dots \\ | & | & | & \dots \end{bmatrix} = z^{[1]}$$

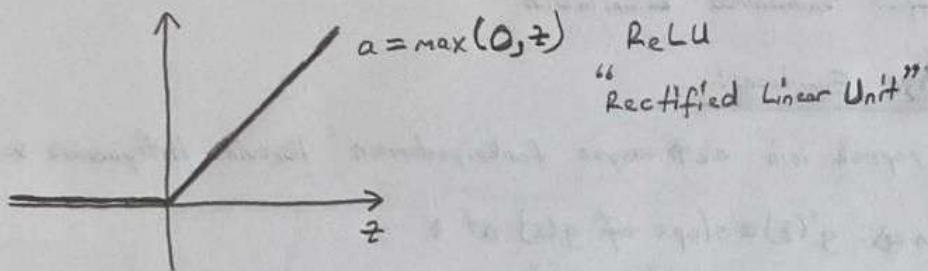
$w^{[1]} x^{(1)} = z^{[1]}(1)$

### Activation Functions:

Aktivasyon fonksiyonlarına baktığımızda genelde sigmoid function kullanılır ama daha iyi aktivasyon fonksiyonları var.



Tanjant fonksiyonu hidden layer için her zaman sigmoid fonksiyondan daha iyi işlevlilik. Sigmoid function genelde binary classification yaparken output layerda tercih edilebilir.



ReLU functionin bir çok iyi özelliğine vardır. Özellikle hidden layerda ne kullanacağını bilmiyorsan ReLU iyi seçeneklerden birisi ve son zamanların en çok tercih edilen fonksiyonudur. ReLU functionin en avantajlı kısmı pozitif kismidir çünkü modelin öğrenme hızını çok fazla artırır. Ayrıca leaky ReLU da tercih edilebilir.

\*\* Sigmoid function: Bu fonksiyon neredeyse hiç kullanılmamış gerken bir fonksiyondur. Sadece ikili sınıflandırma örneklarında output layer için tercih edilebilir. Tanh fonksiyonu çok daha avantajlidir.

\*\* ReLU or Leaky ReLU:  $\text{ReLU} \rightarrow \max(0, z)$ , Leaky ReLU  $\rightarrow \max(0.01z, z)$

ReLU aktivasyon fonksiyonu en çok tercih edilenlerden wśród özellikle probleminiz için hangi fonksiyonu kullanacağınızı bilmeyorsanız direkt ReLU tercih edilebilir.

- Kararsız kalınan durumlarda farklı farklı fonksiyonlar denenebilir. Bu yazının rutauna uygun olduğu gibi fonksiyonların karakteristik özellikleri de anlamazsa yardımcı olur.

#### Why do you need Non-Linear Activation Functions:

Neden linear olmayan aktivasyon fonksiyonlarına ihtiyaç duyuyoruz? Sigmoid ya da diğerini kullanmak ne olurdu?

Eğer aktivasyon fonksiyonunu bakiasesle mesela  $a^{[1]} = z^{[1]}$  olsun ya da identity function olsun  $g(z) = z$  ne veriset onu sıktı veriyor. Bu şarttada hidden layerin bir önemi olmazdı. Koş tane hidden layer olduğunu bilmezdi hatta hidden layer var mı yok mu onu bile bilenezdile. Bu durumu matematiksel islamla kanıtlayalım.

$$\begin{aligned} a^{[1]} &= z^{[1]} = w^{[1]}x + b^{[1]} \\ a^{[2]} &= z^{[2]} = w^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= w^{[2]}(w^{[1]}x + b^{[1]}) + b^{[2]} \\ &= (w^{[2]}w^{[1]})x + (w^{[2]}b^{[1]} + b^{[2]}) \\ &= w'x + b' \end{aligned}$$

Gördüğü gibi input his degirmiyor.

(16)

Linear function sadece output katmanında kullanılır.

### Derivatives of Activation Functions:

Backpropagation işlemi yapmak için aktivasyon fonksiyonlarının türerine ihtiyaç duyulur.

$$\text{Sigmoid activation function} \Rightarrow g'(z) = \text{slope of } g(x) \text{ at } z \\ = g(z)(1-g(z))$$

$$\text{Tanh activation function} \Rightarrow g(z) = \tanh(z)$$

$$g'(z) = \text{slope of } g(x) \text{ at } z$$

$$g'(z) = 1 - (\tanh(z))^2$$

$$\text{ReLU and Leaky ReLU} \Rightarrow g(z) = \max(0, z)$$

$$\Rightarrow g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

$z=0$  technically is undefined but doesn't matter

$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

### Gradient descent for neural networks:

Parameters:  $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$        $n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$

$$\text{Cost function: } J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}_i, y_i)$$

Gradient descent:

Repeat { compute predicts( $\hat{y}^{(i)}$ ,  $i=1 \dots m$ ) }

$$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}, \quad dB^{[1]} = \frac{\partial J}{\partial B^{[1]}}, \dots$$

$$W^{[1]} := W^{[1]} - \alpha dW^{[1]}$$

$$B^{[1]} := B^{[1]} - \alpha dB^{[1]}$$

### Formulas for computing derivatives:

#### Forward propagation:

$$z^{[1]} = w^{[1]}x + b^{[1]}$$

$$A^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[2]} = w^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(z^{[2]}) = g(z^{[2]})$$

#### Backward propagation:

$$dZ^{[2]} = A^{[2]} - Y \quad Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$dB^{[2]} = \frac{1}{m} \text{np.sum}(dZ^{[2]}), \text{axis}=1, \text{keepdims=True}$$

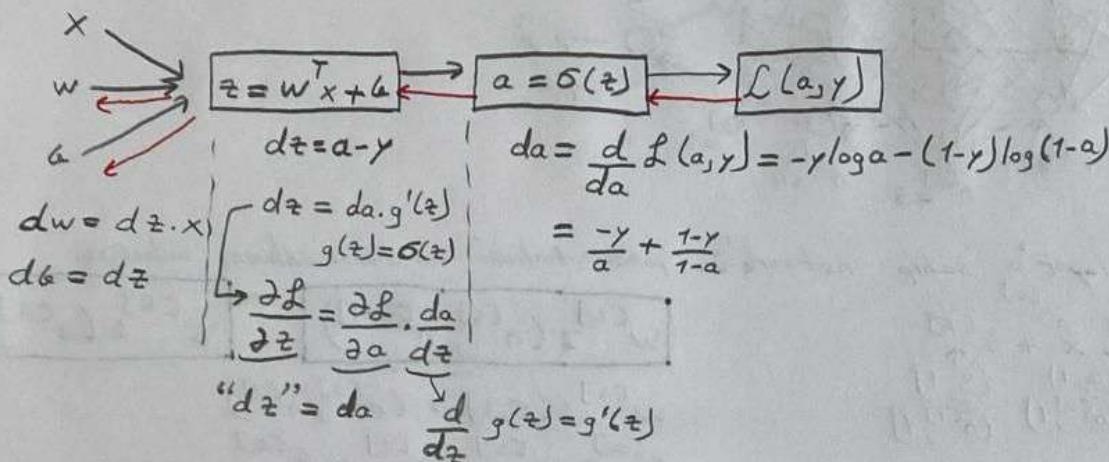
$$dZ^{[1]} = W^{[2]T} dZ^{[2]} \times g^{[1]}'(z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$dB^{[1]} = \frac{1}{m} \text{np.sum}(dZ^{[1]}), \text{axis}=1, \text{keepdims=True}$$

## Computing gradients:

Logistic regression:



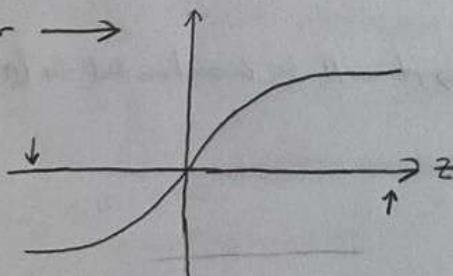
Note: Nöral network yapısında ağırlıkları 0 olarak vermemiz. Eğer tüm ağırlıklar 0 ise ana katmandaki bütün nöronlar aynı işlevi gerçekleştirirler yani simetrik olurlar. Bu da birim nöral ağı kullanma amacımız yok sayı ve hiçbir işe yaramaz. Modeli ne kadar eğittiğimiz önemli degildir.

$$w^{[1]} = np.random.randn(2, 2) * 0.01$$

$$b^{[1]} = np.zeros((2, 1))$$

$w$  değerini farklı değerlerde atamak önemlidir fakat bu işinden sonra  $b$  değerini 0 yapmak lütfen. Bu problem olmaz çünkü fonksiyonlar tamamen farklı şeyler üreticekler. Bu da symmetry breaking problem denir.

Bu işlem ağırlık değerlerini 0.01 gibi çok büyük değerler ile çarpıyoruz. Bunun sebebi şudur →



$$z^{[1]} = w^{[1]} x + b^{[1]}$$

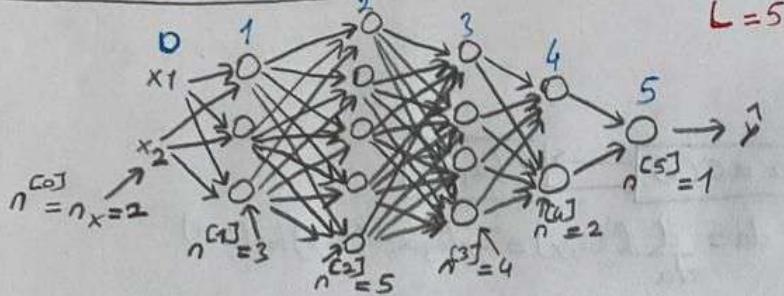
$$a^{[1]} = g^{[1]}(z^{[1]})$$

Eğer  $w$  değerimiz çok büyük ya da çok büyük olursa sonrasında tanh veya sigmoid gibi fonksiyonları kullanırsak gelen değerler de çok büyük ya da küçük olur ve bu değerler işin eğimi çok düşük olur ve eğitimi çok yavaş ilerletir.

Kullanırsak gelen değerler de çok küçük ya da büyük olur ve bu değerler işin eğimi çok düşük olur ve eğitimi çok yavaş ilerletir.

(18)

Parameters  $W^{[L]}$  and  $b^{[L]}$

 $L=5$ 

Yukarıdaki 5 layer'a sahip network'in parametrelerini ve dimensionlerini bulacagiz.

$$z^{[L]} = w^{[L]} \cdot x + b^{[L]}$$

$$\begin{matrix} (3, 1) & (3, 2) & (2, 1) & (3, 1) \\ (n^{[L]}, 1) & (n^{[L]}, n) & (n^{[L]}, 1) & (n^{[L]}, 1) \end{matrix}$$

$$\dim(w \cdot x) = \dim(b)$$

$$w^{[L]} = (n^{[L]}, n^{[L-1]})$$

$$b^{[L]} = (n^{[L]}, 1)$$

$$w^{[2]} = (5, 3) = (n^{[2]}, n^{[1]})$$

$$z^{[2]} = w^{[2]} \cdot a^{[1]} + b^{[2]}$$

$$\begin{matrix} (5, 1) & (5, 2) & (3, 1) & (5, 1) \\ (n^{[2]}, 1) & (n^{[2]}, n^{[1]}) & (n^{[2]}, 1) & (n^{[2]}, 1) \end{matrix}$$

$$w^{[3]} = (4, 5)$$

$$w^{[4]} = (2, 4), w^{[5]} = (1, 2)$$

NOT = Backpropagation işin terev alma işlemi optimizasyonda da dimensionlar aynıdır.

$$dw^{[L]} = (n^{[L]}, n^{[L-1]})$$

$$db^{[L]} = (n^{[L]}, 1)$$

Ayrıca sigmoid ya da diğer aktivasyon fonksiyonları uygulanıktan sonra dimension değişmez.

$$a^{[L]} = g^{[L]}(z^{[L]}) \quad \dim(a^{[L]}) = \dim(z^{[L]})$$

### Vectorized Implementation

Klasik implementation  $\underbrace{z^{[L]}}_{\left[ z^{[1], 1} \ z^{[1], 2} \ \dots \ z^{[1], m} \right]} = \underbrace{w^{[L]}}_{(n^{[L]}, n^{[L-1]})} \underbrace{x}_{(n^{[L-1]}, n^{[L-1]})} + \underbrace{b^{[L]}}_{(n^{[L]}, 1)}$

$$\rightarrow \underbrace{z^{[L]}}_{(n^{[L]}, m)} = \underbrace{w^{[L]}}_{(n^{[L]}, n^{[L-1]})} \underbrace{x}_{(n^{[L-1]}, m)} + \underbrace{b^{[L]}}_{(n^{[L]}, 1)} \rightarrow \text{it will be broadcasted to } (n^{[L]}, m)$$

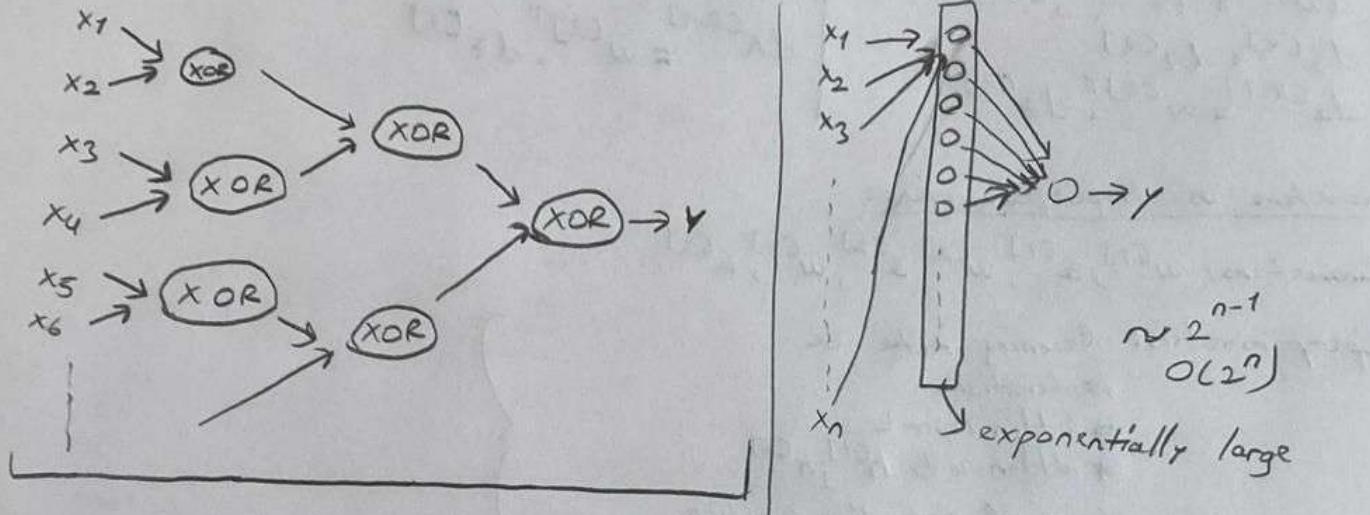
$$\text{classic } \rightarrow z^{[L]}, a^{[L]} : (n^{[L]}, 1)$$

$$\begin{cases} \text{vectorized } z^{[L]}, a^{[L]} : (n^{[L]}, m) \\ \quad l=0 \quad A^{[0]} = x = (n^{[0]}, m) \\ \quad \{ dz^{[L]}, da^{[L]} : (n^{[L]}, m) \end{cases}$$

## Circuit Theory and Deep Learning

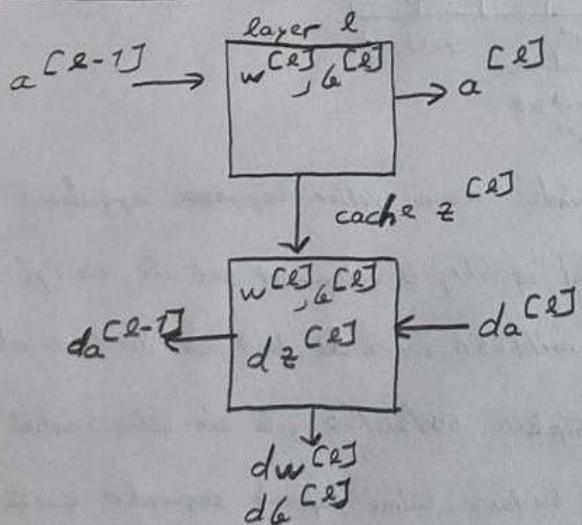
Informally: There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

$$y = x_1 \text{XOR} x_2 \text{XOR} x_3 \text{XOR} x_4 \dots \text{XOR} x_n$$



Burada anlatılmak istenen fazla hidden layer'a sahip ağları tek bir hidden'a sahip normal network ile temsil etmek istenek hidden layerdaki nöron sayısının artması gerektigidir. Bu şekilde tüm detayları model anlayabiliriz.

## Forward and Backward Functions



Forward propagation yapanın bütün z değerleri cache içinde tutulur ve backpropagation işlemi yapanın bu değerler kullanarak foremler hesaplanır sonradan ise parametreleri güncelliriz.

$$\begin{aligned} w^{[l-1]} &:= w^{[l-1]} - \alpha dw^{[l-1]} \\ \theta^{[l-1]} &:= \theta^{[l-1]} - \alpha d\theta^{[l-1]} \end{aligned}$$

Backpropagation kismında z değerlerinin yanında \$w\$ ve \$\theta\$ değerlerini de kullanacağız. Bu yüzden backpropagation işleminde \$z\$, \$w\$ ve \$\theta\$ değerlerinin kopyasını arka planda saklıyoruz diyebiliriz.

## Backward Propagation for Layer L

Input  $da^{[L]}$

Output  $da^{[L-1]}, dw^{[L]}, db^{[L]}$

$$dz^{[L]} = da^{[L]} * g'(z^{[L]})$$

$$dw^{[L]} = dz^{[L]} \cdot a^{[L-1]T}$$

$$db^{[L]} = dz^{[L]}$$

$$da^{[L-1]} = w^{[L]T} \cdot dz^{[L]}$$

$$dz^{[L]} = dA^{[L]} * g'(z^{[L]})$$

$$dw^{[L]} = \frac{1}{m} dz^{[L]} \cdot A^{[L-1]T}$$

$$db^{[L]} = \frac{1}{m} \cdot np.sum(dz^{[L]}, axis=1, keepdims=True)$$

$$dA^{[L-1]} = w^{[L]T} \cdot dz^{[L]}$$

## Parameters vs. Hyperparameters:

Parameters:  $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, w^{[3]}, b^{[3]}, \dots$

Hyperparameters: learning rate  $\alpha$

#iterations

#hidden layer L

#hidden units  $n^{[1]}, n^{[2]}, \dots$

choice of activation function

Momentum, minibatch size, regularizations

## 2. Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization

### Train / dev / test sets:

Data		dev	test
training set		-Hold-out cross validation	testset
		-Development set "dev"	

Bu sistem genel olarak makine öğrenmesinde uzun yıllar boyunca uygulanan veriseti bölmeye işlemi gösteriyor. Training set ile model eğitilir, development set ile en iyi sonucu veren model bulunur ve modele karar verildikten sonra da test seti ile test edilir. 100 - 1000 - 10000 civarlarında verisetine sahipken 60/20/20 gibi bir bölmeye manzıltı olabilir fakat 1.000.000 gibi bir verisetine sahipken bu tanzim yapmak sağlanamaz çünkü çok fazla dev ve test set ayrılmak ve devset ile sadece model deniyoruz. Dolayısıyla probleme bağlı olarak 10.000 test ve dev seti varsa bile yeterli olabilir. Bu da %79.8 train, %0.7 dev, %0.5 test set olarak 60/20/20 dağılımını verir. Bu veri data da artarsa %99.5 train, %0.25 dev %0.25 test set olarak bile dağılım yapılabılır.

Not: Train ve test setlerin dağılımında farklılıklar olabilir.

<u>örneğin</u>	Training Set: Cat pictures from webpages	Dev/test sets: Cat pictures from users using your app
----------------	--	---

örneğin site kendi veritabanı opt'in için kullanıyor. Kullanıcının verilerini dev/test için kullanıyor olabilir.

\* Burada dikkat edilmesi gereken nokta dev ve test set aynı distribution'dan gelmemelidir. Ayrıca sadece train/test olarak ayrılmalarında test set validation olarak kullanılıyor olabilir ve bu durumda test seti olmas faktet bu problem değişdir.

### Bias and variance:

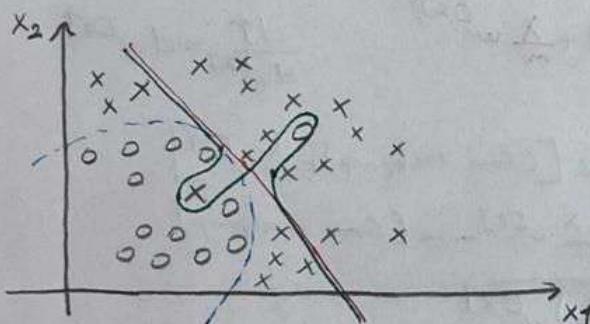
Cost Classification	
Train set error: %0.1	%0.15 high bias high variance
Dev set Error: %0.11	%0.16 high bias high variance

%0.30  
high bias  
high variance

%0.1  
low bias  
low variance

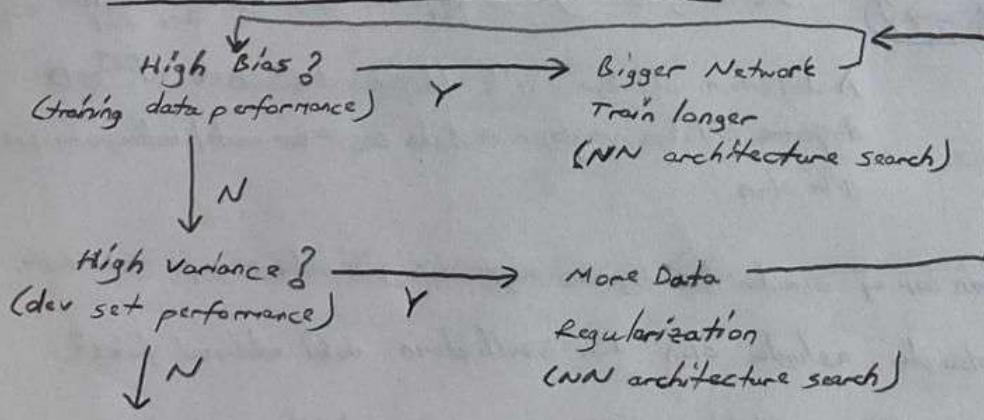
%0.5  
low bias  
low variance

Optimal (Bayes) error:  $\approx 0\%$  verisetine göre degrisi'nden optimal degeri olduguna biseyle dogru olur.



Yukarıdaki görselde mavi renkli çizgi en iyi modellardan biri olabilir. Kırmızı çizgi çok iyi genetikleştiremediği için high bias yorumu yapılabılır. Yeşil görsel ise hala çok iyi genetikleştiremeyecek kadar çok esnek devranmış. Bu durumda high bias ve high variance olarak yorumlanır.

### Basic Recipe for Machine Learning:



“Bias-variance tradeoff” bias ve variance arasında denge kurulmasını fede eder.

Data büyük network variance degerini çok kötü etkileyen, fakat veri ise bias degerini kötü etkileyen faktet bunların seyisinde denge kurulur.

## \*\* Regularization

### Logistic Regression:

$$\min_{w, b} J(w, b) \quad w \in \mathbb{R}^n, b \in \mathbb{R} \quad \lambda = \text{regularization parameter}$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2$$

$$L_2 \text{ regularization } \|w\|_2^2 = \sum_{j=1}^n w_j^2 = w^T w$$

$$L_1 \text{ regularization } \frac{\lambda}{2m} \sum_{j=1}^n |w_j| = \frac{\lambda}{2m} \|w\|_1$$

### Neural Network:

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{[l]}\|_F^2$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^n \sum_{j=1}^{n_{l-1}} (w_{ij}^{[l]})^2 \quad w: (n^{[L]}, n^{[L-1]})$$

↓  
"Frobenius Norm"

$$dw^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} w^{[l]} \quad \frac{dJ}{dw^{[l]}} = dw^{[l]}$$

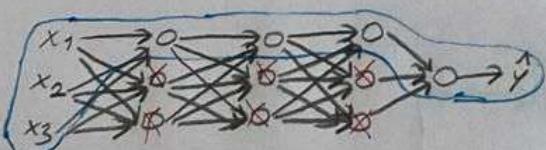
$$w^{[l]} := w^{[l]} - \alpha dw^{[l]}$$

$$w^{[l]} := w^{[l]} - \alpha \left[ (\text{from backprop}) + \frac{\lambda}{m} w^{[l]} \right]$$

$$= w^{[l]} - \underbrace{\alpha \frac{\lambda}{m} w^{[l]}}_{(1-\alpha \frac{\lambda}{m})} - \alpha (\text{from backprop})$$

$L_2$  regularization i̇steminde gelen  $\alpha \frac{\lambda}{m}$  kısmini asıl dēgerden çıkarmışız. Bu duruma  $L_2$  regularization yerine weight decay denir. Yani ağırlık 1'den fazla daha büyük bir seyrə <sup>ekit</sup> olur.

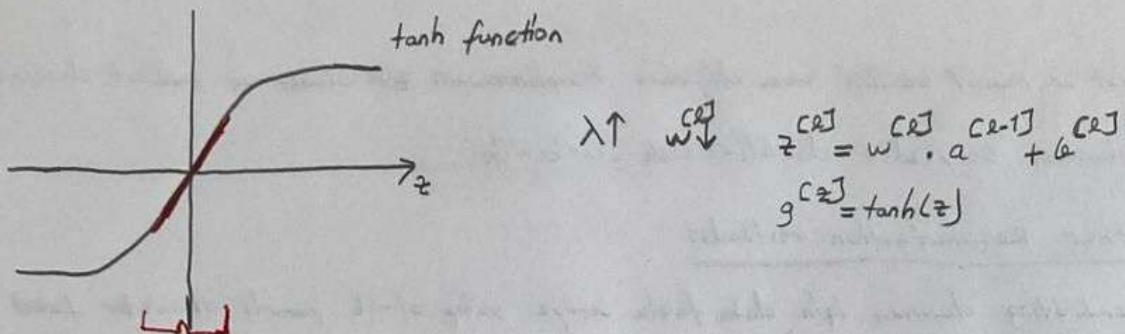
### How Does Regularization Prevent Overfitting?



$$J(w^{[l]}, b^{[l]}) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{[l]}\|_F^2$$

$\lambda$  dēgerinin oldukça büyük olduğunu düşünürsək  $w^{[l]} \approx 0$  dēgerine oldukça yaklaşır ve daha basit bir model kullanılmayağınız olur.

Yukarıdaki şablonlu 'gibi' dērin bir ağı sıradan bir logistic regression gibi oldu ve yapılmıştır. Bu fakir teorik olarak olsa da asılinda olsan tüm özelliliklerin dahi edilmesi fakir özelliklerinin azaltılmasıdır. Bu da modelin esnekliğini azaltır ve generalize edebilir.



Eğer lambda değerini büyütürsek ve bu sayede w dada kışık bir değer alırsa doğrusal olduğumuzda da kışık bir sonuca ulaşır. Bu da tanh fonksiyonunda değişimin çok olduğu nispeten doğrusal (kirmizi) olana denk gelir. Bu da her nöronun tekela linear fonksiyona takın bir yapı sağlıyor gibi ve linear katmanlara sahip nöral ağa sahip oluruz. Bu da yapının basitliğini ve overfitting durumunu azaltmaya çalışır.

Görselleştirmeye yararken de  $J(\dots) = \sum_{i=1}^n \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m} \sum_k \|w^{[k]}\|_F^2$  bu fonksiyon kullanmalıdır yoksa monoton analizi yapamayız.

### Dropout Regularization:

Dropout verilen olasılığa bağlı olarak bazı nöronları yok sayar ve girdi ve çıktı ile olan bağlantılarını keser. Bu da network'in basitliğini ve overfitting'i azaltır.

“Inverted dropout” yöntemi en çok uygulan yöntemlerden biridir ve aşağıdaki gibi implement edilir:

Illustrate with layer  $l=3$  keep-prob=0.8

$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\_prob$

$a3 = np.multiply(a3, d3)$

$a3 /= keep\_prob$

Bu yapı %20 olasılıkle nöronusları ve sonda bu nöronların değerini korumak için tekrar prob degerine bölüyor.

\* Test sırasında dropout kullanılmaz çünkü elde ettiğimiz modeli değerlendirmek istiyoruz. Eğer yaparsak söyletilmiş bir modelde test etmiştik oluruz ve tekrardan keep-prob ile bu nöronların değerini değiştirelim.

\* Dropout'un negatif özelliği Loss değerinin hesaplanmasıın zor olması ve gradient descent'e zararlı etkimes, bunun yerine double-check önerilebilir.

### Other Regularization Methods:

Overfitting durumu için data fazla varije sahip olmak yararlı olacaktır fakat veri集de etki etmeyeceğinden data augmentation teknikleri kullanabiliriz. Görüntü lens gevime, rotate etme, bin kesitini alma gibi işleneler uygulanabilir.

Early stopping validation setin en düşük loss değerini bulmaya çalışır ve bu en iyi senaryoya bantanza yardımcı olur. Negatif etkisi ise diğer işlenelerin tüm sete uygulanması ve işlenelerin yanında kesilmesidir.

### Normalizing Training Sets:

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Subtract mean:

$$M = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$X := X - M$$

Normalize variance

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - M)^2$$

$$X := X / \sigma$$

Use same  $M$  and  $\sigma$  to normalize test set

iki değişkenli verilerimizi subtract mean ile  $x$  eksenine sıkıştırmağa çalıştığımızda ise normalize variance ile veriler  $x$  ve  $y$  ekseninde her birimlik aralıklara hapsedilir.

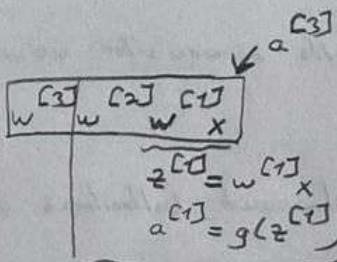
$$x_1: 1 \dots 1000$$

$$x_2: 0 \dots 1$$

Degiskenlerin yukarıdaki değerlere sahip olduğunu varsayırsak ince uzun bir gradient ortaya çıkar. Bu durum optimize etme' ve minimum bulmayı zorlaştırır bunun yerine normalization yapınız.

### Vanishing / Exploding Gradients:

$$y = w^{[L]} w^{[L-1]} w^{[L-2]} \dots$$



$$w^{[L]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

$$y = w^{[L]} \underbrace{\begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}}_{w^{[L-1]}} x$$

$$\begin{bmatrix} 1.5^{L-1} & 0 \\ 0 & 1.5^{L-1} \end{bmatrix} x$$

Network derinligiçe üstel olarak artar ve üstel azalma ya da artışı gergeldegide 1'den büyük değerler için sansızca gitme (exploding), küçük değerler için ise 0'a gitme (vanishing) problemi olur.

Vanishing/Exploding Gradient problem iin kismi gider.

$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

large  $n \rightarrow$  smaller  $w_i$

$$\text{Var}(w_i) = \frac{1}{n} \cdot \frac{2}{n}$$

is better

$$w^{[l]} = np.random.randn(\text{shape}) * np.sqrt(\frac{2}{n^{[l-1]}})$$

↑  
Relu activation iin bulasılıklar

$$\tanh \sqrt{\frac{1}{n^{[l-1]}}}^l$$

↑  
"Xavier initialization"

Degiskenlerde kisit degisiklikler yaparak problemi kismi olarak gider.

\* Hizli artan ve azalan egim probleminin çözümü

### Batch vs. Mini-Batch gradient descent

Vektörizasyon çok veriyi verimli bir şekilde egitmamizi saglar:

$$X = \underbrace{[x^{(1)} \ x^{(2)} \ x^{(3)} \dots \ x^{(1000)}]}_{\substack{x^{(1)} \\ (n_x, 1000)}} \mid \underbrace{x^{(1001)} \dots \ x^{(2000)}}_{\substack{x^{(2)} \\ (n_x, 1000)}} \mid \dots \mid \underbrace{x^{(M)}}_{\substack{x^{(5000)} \\ (n_x, 1000)}}$$

$$Y = \underbrace{[y^{(1)} \ y^{(2)} \ y^{(3)} \dots \ y^{(1000)}]}_{\substack{y^{(1)} \\ (1, 1000)}} \mid \underbrace{y^{(1001)} \dots \ y^{(2000)}}_{\substack{y^{(2)} \\ (1, 1000)}} \mid \dots \mid \underbrace{y^{(M)}}_{\substack{y^{(5000)} \\ (1, 1000)}}$$

Yukarıdaki yapı 5.000.000 veriye sahip olup her biri 1000 örneğe sahip toplamda 5.000 mini batches fikrine göre tasarlanmıştır. Her bir batch  $\frac{1}{3}$  içinde ifade edilen batch tüm veriyi egittiğinden sonra gradient descent uygulanır, mini-batch kisit bir seti eğitip sonra gradient descent uygular, bu sayede daha fazla gradient descent uygulanır ve algoritmanın data iyi performans vereceği düşünülür.

### \* Mini-Batch Gradient Descent:

for  $t = 1, \dots, 5000$

$$\begin{aligned} & \text{forward propagation } X \\ & z^{[t]} = w^{[t]} X^{[t]} + b^{[t]} \\ & A^{[t]} = g^{[t]}(z^{[t]}) \end{aligned}$$

$$A^{[t]} = g^{[t]}(z^{[t]})$$

vectorized implementation  
(1000 example)

Sanki tüm verimiz 1000 örnekligini birlikte dövranarak aynı işlerde yapar. Her döngüre "epoch" denir.

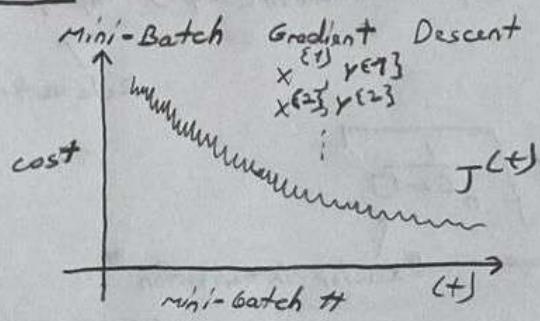
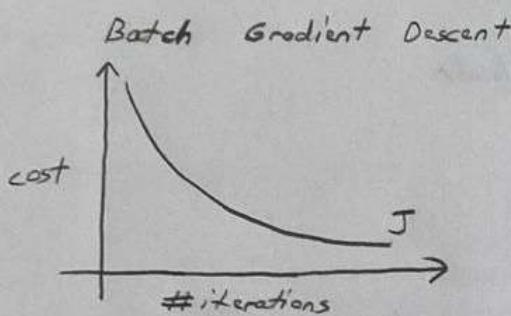
(26)

$$\text{Compute cost } J^{t+3} = \frac{1}{1000} \sum_{i=1}^2 \mathcal{L}(y_i^{(t)}, y_i^{(t)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|w^{[l]}\|_F^2$$

$$w^{[l]} := w^{[l]} - \alpha d w^{[l]}, \quad b^{[l]} := b^{[l]} - \alpha d b^{[l]}$$

Aynı işlemleri her epoch için yapılıyor.

### \* Training with mini-batch gradient descent:



Batch kısmında düzgün bir şekilde düzenli azalış görünen, mini-batch salınımlı olarak azalıyor burada öneşli olan azalış trend'ine sahip olmasıdır. Burada arasında artışı olması normaldir ve böyle ilgili problem olabilir, ya da öğrenilmesi kolay ve zor örnekler perçeve gelmiş olabilir.

### Choosing your mini-batch size

If mini-batch size =  $m$ : Batch gradient descent  $\rightarrow$  tüm veriyi alır.

If mini-batch size = 1: Stochastic gradient descent  $\rightarrow$  tek bir örnek kendi değeri sahip

Batch gradient yolunu bulurken, stochastic gradient hizen yanlış şapkalılar ya da minimum nokta etrafında dolopar. Bütün  $i$  ile  $m$  arasında bir değer seçmeliyiz çünkü her ikiinin de ekib yonları var. Batch-gradient çok fazla hızlı ugrastığı için asırı zaman tüketen bir uygulamadır, stochastic gradient descent ise vectorization işlerinden faydalananız ve hizini kaydeden

Eğer eğitim konusunuz varsa direkt batch-gradient uygunluğunuza. Daha büyük setler içinde  $2^n$  katları 64, 128, 256, 512 gibi değerler uygunluğunuza. Bu memory'in kapasitesine ve dözenlenmeye uygundur. Ayrıca CPU/GPU hafızası ile uyusun bir değer olmalıdır.

### Exponentially Weighted Averages:

Batch ve mini-batch uygulamalarının daha iyisi yapılabılır. Bunun için istatistikte exponentially weighted moving averages da denen eksponansiyel kullanıyoruz.

Exponentially weighted Averages devamlı

Temperature in London

$$\theta_1 = 40^{\circ}\text{F}$$

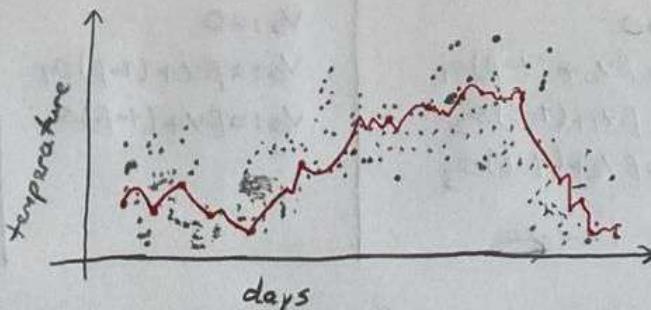
$$\theta_2 = 49^{\circ}\text{F}$$

$$\theta_3 = 45^{\circ}\text{F}$$

$$\vdots$$

$$\theta_{180} = 60^{\circ}\text{F}$$

$$\theta_{181} = 56^{\circ}\text{F}$$



$$V_0 = 0$$

$$V_1 = 0.9 \times V_0 + 0.1 \times \theta_1$$

$$V_2 = 0.9 \times V_1 + 0.1 \times \theta_2$$

$$V_t = 0.9 \times V_{t-1} + 0.1 \times \theta_t$$

Şekilde görüldüğü gibi olursa da tüm verileri düzenleme aktardan sonra EWA değeri her gün için hesaplanır ve türkeli siyah kurşun bir şerit hale gelir.

$$V_t = \beta V_{t-1} + (1-\beta) \theta_t$$

$V_t$  can approximately average over

$$\beta = 0.9 : \approx 10 \text{ days temperature} \rightarrow \frac{1}{1-\beta} \text{ days temperature}$$

$$\beta = 0.99 : \approx 50 \text{ days} \rightarrow \frac{1}{1-0.99} = 50$$

Bu değer daha dağ ve gerçek değerlerden sonra gerçek değerlere en yakın bir yapı gösteren

süntü çok fazla gönnesas alıyor ve önceki güne çok fazla dikkat gösteriyor. ( $\beta=0.99$ ) Yarış uygum

$$\beta = 0.5 : \approx 2 \text{ days} \rightarrow \frac{1}{1-0.5} = 2 \text{ iti gönün ortalamasını alır.}$$

Titresimli sonuçlar var fakat değişimlere sabuk ayak aydunur. (Mm)

Understanding EWA:

$$V_{100} = 0.9 V_{99} + 0.1 \theta_{100}$$

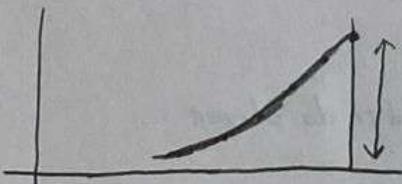
$$V_{99} = 0.9 V_{98} + 0.1 \theta_{99}$$

$$V_{98} = 0.9 V_{97} + 0.1 \theta_{98}$$

...

$$\Rightarrow V_{100} = 0.1 \theta_{100} + 0.9 V_{99}$$

$$= 0.1 \theta_{100} + 0.1 \times 0.9 \times \theta_{99} + 0.1 \times (0.9)^2 \times \theta_{98} + 0.1 \times (0.9)^3 \times \theta_{97} + 0.1 \times (0.9)^4 \times \theta_{96}$$



Üstel olarak azalan bir ağırlık grafiği çiziyor ve da ekstra gittikçe etki azalır anlaşıma gelir.

Implementing EWA:

$$V_0 = 0$$

$$V_1 = \beta V_0 + (1-\beta) \theta_1$$

$$V_2 = \beta V_1 + (1-\beta) \theta_2$$

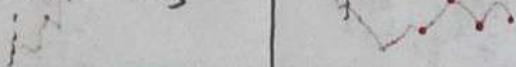
$$V_3 = \beta V_2 + (1-\beta) \theta_3$$

$$V_\theta := 0$$

$$V_\theta := \beta V + (1-\beta) \theta_1$$

$$V_\theta := \beta V + (1-\beta) \theta_2$$

$$V_\theta := \beta V + (1-\beta) \theta_3$$



code:

$$V_0 = 0$$

Repeat  $\mathcal{E}$ Get next  $\theta_t$ 

$$V_\theta := \beta V_\theta + (1-\beta) \theta_t$$

{}

Bu işlemi programlama yaygın olarak kullanılmazının nedeni hafıza verimliliğidir.

Tüm verileri tutmak yerine önceki sonucu tutuyoruz. Bu da varlığı artırıyor.

\*Bias Correction:

$\beta = 0.98$  değeri örnek alırsak ve  $V_0 = 0$  olduğunu düşündürsek ilerleyen tahminler aşırı düşük olacak ve bu bir problem olup düzeltmek için;

$$\frac{V_t}{1-\beta^t} \quad t=2: 1-\beta^t = 1-(0.98)^2 = 0.0396$$

$$\frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$$

$t$  degeni 64'e düşüge gorseller eşittir fakat bu degenin doğrulanması için bu önemlidir.

Gradient Descent with Momentum:

Standart gradient descent dikay ve yatayda minimuma yaklaşırken data arası salınım yapacaktır. Bu da learning rate değerini düşürerek yol açar. Dikayedeki salınımları azaltıp yataydaki salınımları öğrenimi artırırsak, frizitçe bir topun yokuş aşağı yuvarlanmasıne genzer bir yapı oluşturur ve momentum artar.

Implementation Details:

$$Vdw = 0, Vd\theta = 0$$

On iteration  $t$ :Compute  $dw, d\theta$  on the current mini-batch

$$Vdw = \beta Vdw + (1-\beta) dw$$

$$Vd\theta = \beta Vd\theta + (1-\beta) d\theta$$

$$w = w - \alpha Vdw, \theta = \theta - \alpha Vd\theta$$

$$\text{Hyperparameters} = \alpha, \beta \quad \beta = 0.9$$

Akademik kaynaklarda  $(1-\beta)$  kismının kullanılmadığı da oluyor.

## RMSprop (Root Mean Square Prop):

Bu optimizasyon standart gradient descent ifadesine göre daha iyi performans gösterebilir.

On iteration  $t$ :

Compute  $dW, d\theta$  on current mini-batch

$$Sd_w = \beta Sd_w + (1-\beta) dW^2 \leftarrow \text{small}$$

$$Sd_\theta = \beta Sd_\theta + (1-\beta) d\theta^2 \leftarrow \text{large}$$

$$w := w - \alpha \frac{dW}{\sqrt{Sd_w}}, \quad \theta := \theta - \alpha \frac{d\theta}{\sqrt{Sd_\theta}}$$

Burada  $\theta$  değerini dik hâle getirme etkisi eden parametrelər,  $w$  değerini ise yatay hâle getirme etkisi eden parametrelər olaraq düşünürsek  $w$  değerini təqib etmək istəriz qəntərə fərzi kətəcək və yataya az etkili etmək istəriz.  $\theta$  deyəri isə təqib olsun və deyəri təqib etmək istəriz.

## Adam Optimization Algorithm:

Adam optimization RMSprop ve momentum birlikte kullanır.

$$Vdw=0, Sdw=0 \quad Vd\theta=0, Sd\theta=0$$

On iteration  $t$ :

Compute  $dW, d\theta$  using current mini-batch

$$Vdw = \beta_1 Vdw + (1-\beta_1) dW, \quad Vd\theta = \beta_1 Vd\theta + (1-\beta_1) d\theta \leftarrow \text{"momentum"} \beta_1$$

$$Sdw = \beta_2 Sdw + (1-\beta_2) dW^2, \quad Sd\theta = \beta_2 Sd\theta + (1-\beta_2) d\theta^2 \leftarrow \text{"RMSprop"} \beta_2$$

After bias correction

$$Vdw_{\text{corrected}} = Vdw / (1 - \beta_1^t), \quad Vd\theta_{\text{corrected}} = Vd\theta / (1 - \beta_1^t)$$

$$Sdw_{\text{corrected}} = Sdw / (1 - \beta_2^t), \quad Sd\theta_{\text{corrected}} = Sd\theta / (1 - \beta_2^t)$$

$$w := w - \alpha \frac{Vdw_{\text{corrected}}}{\sqrt{Sdw_{\text{corrected}}} + \epsilon}, \quad \theta := \theta - \alpha \frac{Vd\theta_{\text{corrected}}}{\sqrt{Sd\theta_{\text{corrected}}} + \epsilon}$$

Bu yapı, təqib növələri və farklı yapıları isin sətə pozitif sonular verənək təqibinə təsdiq edir.

Hyperparameters choice:

$\alpha$ : needs to be tuned

$$\beta_1: 0.9 \quad (dw)$$

$$\beta_2: 0.999 \quad (dw^2)$$

$$\epsilon: 10^{-8}$$

Adam = Adaptive moment estimation

(30)

### Learning Rate Decay:

Learning Rate Decay aslında learning rate değerini zamanla (mini-batch ile) azaltmaktadır. Bu sure anası minimum değerde yoldaşılığa olusacak noise problemini azaltmaktadır.

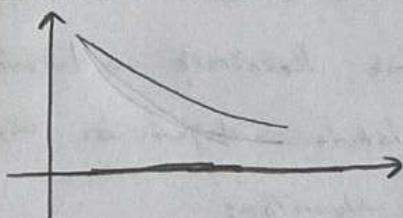
#### Implementation:

1 epoch = 1 pass through the data

$$\alpha = \frac{1}{1 + \text{decayRate} \times \text{epochNumber}} \alpha_0$$

Epoch	$\alpha$
1	0.1
2	0.087
3	0.05
4	0.04
:	:

$$\alpha_0 = 0.2 \\ \text{decay-rate} = 1$$

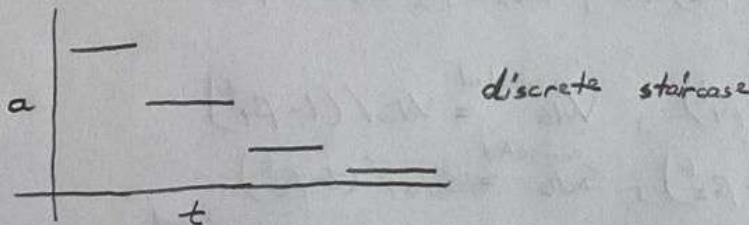


Other learning rate decay methods:

$$\alpha = 0.95^{\text{epoch\_num}} \cdot \alpha_0 - \text{exponentially decay.}$$

$$\alpha = \frac{k}{\sqrt{\text{epoch\_num}}} \cdot \alpha_0 \quad \text{or} \quad \frac{k}{\sqrt{t}} \cdot \alpha_0 \\ \downarrow \text{mini batch}$$

or

or  
"Manual Decay"

### The Problem of Local Optima:

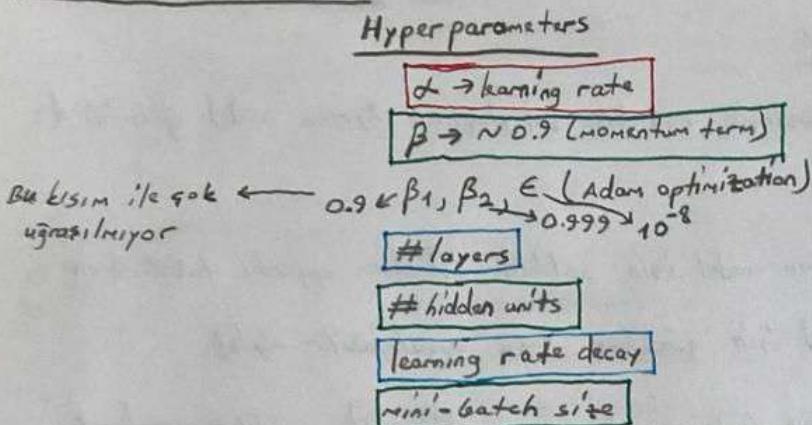
Eski zamanlara göre derin öğrenmedeki backtracking hızı değişti. Az boyutlu görselleri, çok boyutlu problemleri anlamak için kullanamayız belki bazı problemlerde 20.000 parametre olacak ve bunu anlamak kolay olmayacağı gibi. Burada local optiminde takılmak yerine saddle points noktalarında dağ rastlarız. (Bir atın sırtı ömek olabilir.) Bu plateau noktalarından kurtulmak dağ kolay olabilir. Burada not alınabilecek 2 sonuclu şey vardır:

1. Local optiminde sızmak sok düşülebilir ihtimaldir.

2. Plato noktaları öğrenmeyi oldukça yavaşlatıyor fakat bu noktada momentum, RMSprop ve da

Adam gibi algoritmalar öğrenme algoritmasına yardımcı oluyor. Adam gibi algoritmalar platodan aşağı iner ve sonra onu terk etme sürecini hızlandırabilir.

## Tuning Process:



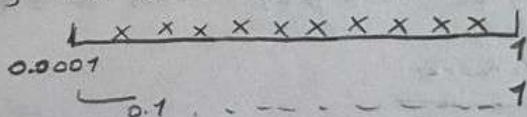
### Importance

Kırmızı çerçeve → en önemli  
Yeşil çerçeve → önemli parametreler  
Mavi çerçeve → daha az önemli

Az sayıda hyperparametreye sahip olsaydık bir grid oluşturmak ve kombinasyonları deneyip hangisi daha iyi sonuc veriyorsa onu seçebilirdik. Fazla parametre olduğu için rastgele olarak değerleri sağlamak daha iyidir çünkü hangi değerlerin modellimize en iyi uyuyacagini bilmemiz. Geniş bir aralığa baktiktan sonra data türküt bir aralığa odaklanabilir. Örneğin 10-100-1000 değerlerine baktık ve en iyi sonucu 100 verdii sonra 80-100-120-150 gibi bir aralığa odaklılıyız ve bu lüzi en iyi değer yaklaştırmış.

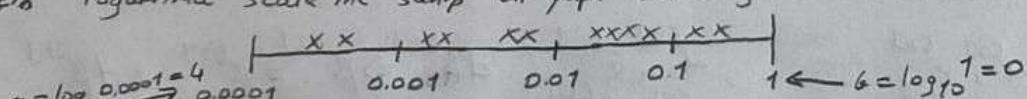
### Using an appropriate scale to pick hyperparameters:

→ learning rate  $\alpha = 0.0001, \dots, 1$



Bu tarz bir doğrulukta örtüklerin çoğu 0.1 ile 1 arasında olacaktır. Bu durumdan kurtulmak için logaritmik scale'ine sahip bir yapı kurmak gerekti.

$a = \log_{10} \frac{0.0001}{0.0001} = 4$



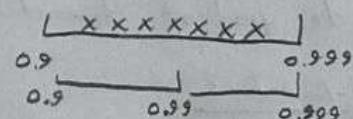
Bu tarz bir yapı dahe çok iyi olacak. Bunu python'da aşağıdaki gibi kodlayabiliriz:

$$r = -4 * np.random.rand() \leftarrow r \in [-4, 0]$$

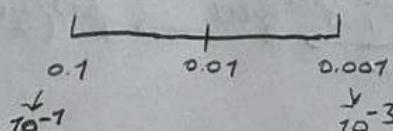
$$\alpha = 10^{-4}$$

→ for  $\beta = 0.9, \dots, 0.999$

↓  
for example      ↓  
1000 example



$$1 - \beta = 0.1, \dots, 0.001$$



Burada exponentially weighted average

kullandığımızda  $\frac{1}{1-\beta}$  değerler 1'e

yaklaştıktan sonra değişebilidiklerine bile asırı

durable hale gelir ve bununla birlikte log scale ile

türküt kesimlara odaklanıyoruz.

$$r \in [-3, -1]$$

$$1 - \beta = 10^r$$

$$\beta = 1 - 10^r$$

## Batch Normalization:

### Normalizing activations in a network:

Batch normalization hipuperametre arası problemini çok daha kolaylaştırır. Ayrıca model eğitiminin de kolaylaştırır.

$$M = \frac{1}{m} \sum_i x^{(i)}$$

Basit bir model için soldaki islemler uygundu fakat dem

$$x = x - M$$

bir model için girdilerde bunu uygulayabilir misiniz?

$$\sigma^2 = \frac{1}{m} \sum_i (x^{(i)} - M)^2$$

Aktivasyon fonksiyonundan önce  $\tilde{z}^{(1)}, \tilde{z}^{(2)} \dots$  girdilere uygulanır

$$x = x / \sigma$$

modeli hızlandırır.

### Implementing Batch Norm:

Given some intermediate values in NN  $\underbrace{z^{(1)}, \dots, z^{(m)}}_{\tilde{z}}$

$$M = \frac{1}{m} \sum_i z^{(i)}$$

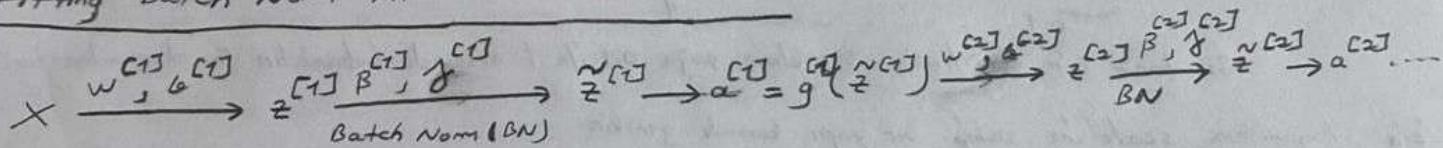
$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - M)^2$$

$$\tilde{z}_{norm}^{(i)} = \frac{z^{(i)} - M}{\sqrt{\sigma^2 + \epsilon}}$$

learnable parameters

$$\tilde{z}^{(i)} = \gamma \tilde{z}_{norm}^{(i)} + \beta \quad \text{if } \gamma = \sqrt{\sigma^2 + \epsilon} \text{ and } \beta = M \text{ then } \tilde{z}^{(i)} = z^{(i)}$$

### Fitting Batch Norm into a Neural Network:



$$\text{Parameters: } w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \dots, w^{[L]}, b^{[L]}$$

$$\beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots, \beta^{[L]}, \gamma^{[L]}$$

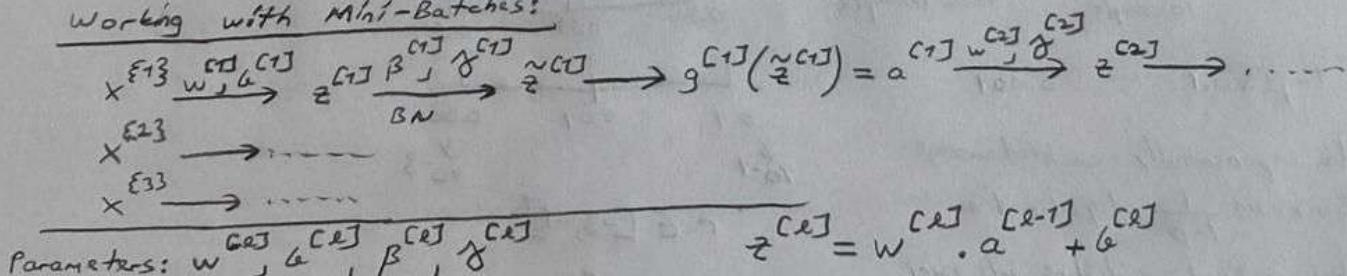
$$d\beta^{[L]} \Rightarrow \beta^{[L]} = \beta^{[L]} - \alpha d\beta^{[L]}$$

[f. nn. batch-normalization]

Buradaki  $\beta$  ifadesi her  $\beta'$ lardan tamamen farklıdır (Adam vs...)

$\beta$  ve  $\gamma$  parametreleri de gradient descent ile güncellenebilir

### Working with Mini-Batches:



Batch normalization  $O$  ortalama ve standart varyans ile düzeltip sonra da  $\beta$  ve  $\gamma$  ile ölçekteştirme

Burada  $b^{[c]}$  geçersiz parametredir çünkü batch norm hesaplamada sırasıkaaktır en yerden  $O$  obrak da atanır.

## Why Does Batch Norm Work?

Tamamen siyah kedilerden ve diğer hayvanlardan oluşan bir verisetini eğitirsek bu model test aşamasında renkli kedileri tespit edemez ve bu tarz bir probleme "covariate shift" denir. Bu tarz bir problemi çözüm için öğrenme algoritmosu tekrar eğitmek zorunda kalırız. Bunu derin nöral ağlara uyguladığımızda ilk katmanlar içeri katmanlar üzerinde çok fazla etkiye sahip olabileceğini için burada mean ve variance değerine girdiğim ayarlanması her katmanda oluşan değişimimi azaltacaktır. Dolayısıyla batch-norm girdi değerlerinin değişim problemini azaltır, bu değerlerin data kıraklı hale gelmesine yol açar. Bu içeri katmanlara daha sağlam bir zemin hazırlar ve girdi değerlerinde değişim olsa bile ilerli katmanlarda daha az değişim olur. Bu ilk katmanlarda öğrenim sık olsa bile sonraki katmanlardaki tepkinin azalması antlamına gelir. Bu da her katmanın girişinden daha da yavaşlaşması ve kendi kendine öğrenmesini antlamına gelir. Bu da witin ağıın öğrenme hızını artttır.

### → Batch Norm as Regularization:

Her mini-batch mean/variance hesabına göre ayarlığında biraz noise içeriği içinde sadece 64-128-256 gibi küçük porsa vary ile hesaplanmıştır  $\hat{z}^{[2]}$  ve  $\hat{z}^{[L]}$  de bu değerler iki hesaplandığı için noise' e sahip olacaklardır. Eğer mini-batch değerini 512-1024 gibi değerlere çıkarırsak noise azalacaktır ve da regularization etkisini azaltır. Batch-norm regularization için ilk önce değil fakat bir etken olabilir.

### Batch Norm at Test Time:

$$\mu = \frac{1}{m} \sum_i z^{[l]i} \rightarrow m \text{ örneğine sahip mini-batch için ortalama hesabı}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z^{[l]i} - \mu)^2 \rightarrow \text{mini-batch için variance hesabı}$$

$$z_{\text{norm}}^{[l]} = \frac{z^{[l]i} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\hat{z}^{[l]} = \gamma z_{\text{norm}}^{[l]} + \beta$$

Bu formüller train kismi için uygunluğunu yaridoydu. Test kismi için yeterince veri olmayalıhatta tek bir vary'ı test etmek isteyebiliriz bunun da ortalama ve varyans hesaplamak mantıklı degildir.

$\mu, \sigma^2$ : estimate using exponentially weighted average (across mini-batch)

$$\frac{\mu_{\text{avg}}}{\sigma_1} = \frac{x^{E13}[C]}{\sigma_1}, \frac{\mu_{\text{avg}}}{\sigma_2} = \frac{x^{E23}[C]}{\sigma_2}, \frac{\mu_{\text{avg}}}{\sigma_3} = \frac{x^{E33}[C]}{\sigma_3} \rightarrow \mu$$

$$\frac{\sigma^2_{\text{norm}}}{\sigma^2} = \frac{\sigma^2_{\text{avg}}}{\sigma^2} = \frac{\sigma^2_{\text{avg}} + \epsilon}{\sigma^2 + \epsilon}$$

$$z_{\text{norm}} = \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad \hat{z} = \gamma z_{\text{norm}} + \beta$$

### Softmax Regression:

Softmax islemde output sınıf sayisi kader olmalıdır. Output layerdeki her nöron her bir sınıfta alt değerler hasaplar tarafla ise şu formül kullanılır

$$\text{Output layer} = \text{layer } L$$

$$z^{[L]} = w^{[L]} \cdot a^{[L-1]} + b^{[L]} \quad \begin{matrix} \text{4 sınıfta bir problem için} \\ (4, 1) \end{matrix}$$

Activation Function:

$$t = e^{z^{[L]}}$$

$$a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{i=1}^4 t_i}, \quad a'_i = \frac{t'_i}{\sum_{i=1}^4 t'_i}$$

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

$$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}, \quad \sum_{i=1}^4 t_i = 176.3$$

$$a^{[L]} = \frac{t}{176.3} \rightarrow \frac{e^5}{176.3} = 0.842, \quad \frac{e^2}{176.3} = 0.042, \quad \frac{e^{-1}}{176.3} = 0.002, \quad \frac{e^3}{176.3} = 0.114$$

ile nöron en yüksek olasılık değerine sahiptir.

Softmax activation ile regression tipinin genişletilmiş hali' gidiyor. Notaları sınıflandırılmış ve işi gruba ayırdığınızda dağın genişletilmiş hali' gidi' deşarj edilir.

### Training a softmax classifier:

Softmax regression generalizes logistic regression to C classes.

#### Loss Function:

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$-r_2 \log \hat{y}_2 = -\log \hat{y}_2$$

$$L(\hat{y}, y) = - \sum_{j=1}^4 r_j \log \hat{y}_j$$

$$a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$$

Loss değerini en yüksek olan tekmin  $\hat{y}$  değerini oldukça sıyrılmış sınıflı olasılıkla yaklaştırır. Log değer 0'a gitmeyen

(35)

$$J(w^{[0]}, b^{[0]}, \dots) = \frac{1}{m} \sum_{i=1}^m L(y^{(i)}, \hat{y}^{(i)})$$

buradan sırda gradient descent

kullanarak azaltmaya çalışıyorsunuz.

$$\text{Tahminler} \rightarrow Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}], \quad \hat{Y} = [\hat{y}^{(1)}, \dots, \hat{y}^{(m)}]$$

$$= \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0.3 & 0.2 & 0.1 & 0.4 \end{bmatrix}$$

$\rightarrow (4, m)$  dimension

$\rightarrow (4, m)$  dimension

$$\text{Backprop} \Rightarrow d_z^{(L)} = \hat{y} - y$$

$(4, 1)$

$$\frac{\partial J}{\partial z^{(L)}}$$

### TensorFlow:

TensorFlow gibi kütüphanelerizi kullanarak ve ögrenildikten sonra aynı yapıları tekrar tekrar yazmamız gereklidir.

Şimdi:  $J(w) = w^2 - 10w + 25$  fonksiyonu için Python implementation yapalım

```

import numpy as np
import tensorflow as tf

w = tf.Variable(0, dtype=tf.float32)
optimizer = tf.keras.optimizers.Adam(0.1)

def train_step():
    with tf.GradientTape() as tape:
        cost = w**2 - 10*w + 25
    trainable_variables = [w]
    grads = tape.gradient(cost, trainable_variables)
    optimizer.apply_gradients(zip(grads, trainable_variables))

for i in range(1000):
    train_step()
    print(w)

```

Aynı işin TensorFlow implementation'i:

```

w = tf.Variable(0, dtype=tf.float32)
x = np.array([1.0, -10.0, 25.0], dtype=np.float32)
optimizer = tf.keras.optimizers.Adam(0.1)

def training(x, w, optimizer):
    def cost_fn():
        return x[0]*w**2 + x[1]*w + x[2]

    for i in range(1000):
        optimizer.minimize(cost_fn, [w])

    return w

w = training(x, w, optimizer)
print(w)

```

## Structuring Machine Learning Projects

Bu kurs makine öğrenmesi projelerini nasıl yapılandırıcağınızda ya da planlayacağınızda ve bunu yaparken nasıl bir strateji itleyeceğinize odaklanmaktadır.

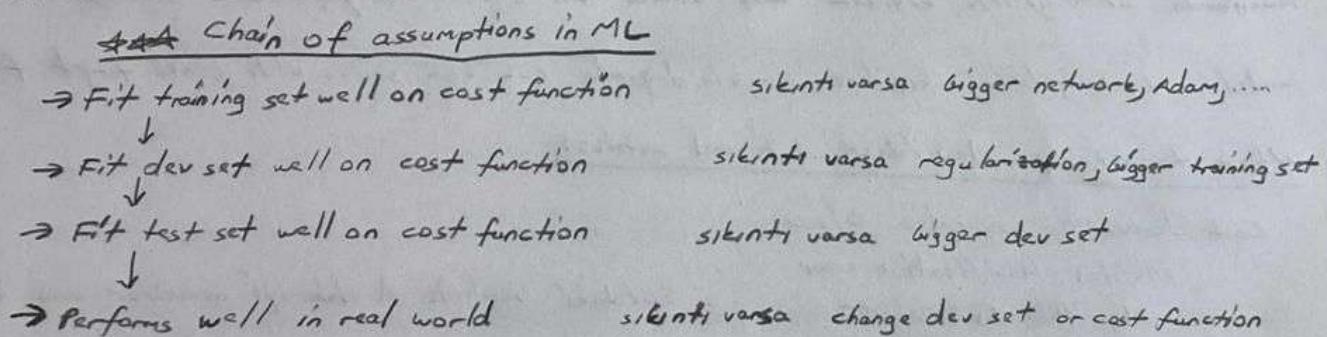
Mesela bir cat classifier projesi %90 accuracy'e sahip olabilmek ama bu yeterli olmayacağı ve yeni filtreler ortaya çıkabilir.

- Ideas:
- Collect more data
  - Collect more diverse training set
  - Train algorithm longer with gradient descent
  - Try Adam instead of gradient descent
  - Try bigger network
  - Try smaller network
  - Try dropout
  - Add L<sub>2</sub> regularization
  - Network Architecture
    - Activation Functions
    - # hidden units
    - ...

Bütün seçenekleri denemek osm zaman tüketici için sonuca gören bazı yöntemler göreciniz.

### Orthogonalization:

Orthogonalizasyon modeli etkileyen özelliklerin birbirinden bağımsız olarak ele alınmasıdır. İki boyutta düzlemede 90°'larak daşınabiliriz. Bu şekilde daha kolay yönetiriz ve ne yapmak istiyorsak modeli onu yapırız.



### Single Number Evaluation Metric:

Model değerlendirmesinde birden fazla metric karışımıza olabılır mesela iki farklı classifier'in birinin precision değerini recall değerini yükselt olsun bu durumda hangi modelin sepeceğiniz noktada kararsız kalğımızız için her ikisini ığıncaan F1-score sonuçlarına bakmak mantıklı olacaktır. Tamamen farklı kılçelende denilen ve errorları olan birden fazla model için tüm hataların ortalamasını almak ve sadexe bu metric'le ızarıdan gittikçe mantıklı olacakları

Satisficing and Optimizing Metrics:

Her zaman metric'lerin tekne indirme hizasını olmayabilir.

classifier	optimizing Accuracy	satisficing Running Time
A	90%	80ms
B	92%	95ms
C	95%	1500ms

maximize accuracy, subject to running time  $\leq 100\text{ms}$

Wakewords / Trigger words

Alexa, ok Google, Hey Siri gibi ifadelerin çalışma zamanını test'ler burada accuracy yanında #false → positive durumunda etkili olabilir. Yani seslenmediginde otomatik kendisine salismasi durumu burada maximize accuracy - optimizing olur.  
 s.t.  $\leq 1$  false positive - satisfying  
 every 24 hours

\* \* \* Dev set ve test set'in tamamen farklı dağılımlardan alınması genel uygulamalı. Dev ve test için alınan veri seti shuffle edilip rastgele bir dağılım alınması data pozitif sonuc ratio Amaçlarda orta geliri, kisiye göre vermek için (geni ödemeyi yapabilecektir) salısan şirket aynı modeli dörtte getirili kişiye uyguladığında sole kötü sonuc aldı çünkü target farklı!

When to change dev/test sets and metrics:

cat dataset examples olsun.

Metric: classification error

Algorithm A: 9.3 error

Algorithm B: 9.5 error

Soldaki verilerde A daha iyi gidiyor ama testlerin dışında pornografik içerikdeki iki veren bir modelse bu test edilebilir.

$$\text{Error: } \frac{1}{m_{\text{dev}}} \sum_{i=1}^{m_{\text{dev}}} \left\{ \begin{array}{ll} 1 & \text{if } y_{\text{pred}}^i \neq y^i \\ 0 & \text{otherwise} \end{array} \right\}$$

Burada pornografia de değer lendiren bir modeli etrafından geçen her bir gözdeki gibi ayarlanabilir.

$$\text{Error: } \frac{1}{\sum w^{ij}} \sum_{i=1}^{m_{\text{dev}}} w^{ij} \left\{ \begin{array}{ll} 1 & \text{if } y_{\text{pred}}^i \neq y^i \\ 0 & \text{otherwise} \end{array} \right\}$$

$$\rightarrow w^{ij} = \begin{cases} 1 & \text{if } x^{ij} \text{ is non-porn} \\ 10 & \text{if } x^{ij} \text{ is porn} \end{cases}$$

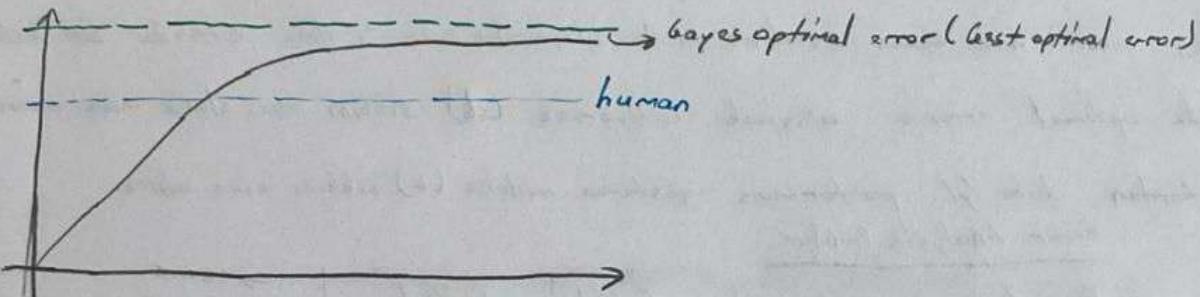
Bu işlem de orthogonalization isminde dahildir. Problemi farklı parçalara ayırp ayrı ayrı yönettiğimiz loss için  $\mathcal{J} = \frac{1}{\sum w^{ij}} \sum_{i=1}^{m_{\text{dev}}} w^{ij} \mathcal{L}(y_{\text{pred}}^{ij}, y^{ij})$

$$\mathcal{L}(y_{\text{pred}}^{ij}, y^{ij}) = \frac{1}{2} (y_{\text{pred}}^{ij} - y^{ij})^2$$

Not: Cat classifier ya da buna benzer dev/test sette iyi per formans gösteren bir modelin olasılıklu smogia yitesek kaliteli gorsellende iyi performansa sahipse fakat uygulanmaya yitlenen diye kaliteli gorselleri tespit ederiyorsa bu durunda metriklerde ya da dev/test seti' genellimelik degistirmek gereklidir.

### Why Human-level Performance?

İnsanların algısı yitesek ve bir model, gelistiştirken bunun temel alınması gerektesinde.



Modelller insanları gazi konularda geçse de en iyi noktaya ulaşmak nedeniyle inkansız gibi görüntülerdeki blurlu image gibi problemler cevapsız kalmıyor.

Model insan seviyesinin altında kalmırsa geliştirmek öneminder fakat insana denk olmak en seçip de insanla karşılaştırmanın bir diğer seçenekidir.

ML is worse than humans, you can:

- Get labeled data from humans
- Gain insight from manual error analysis  
why did a person get this right?
- Better analysis of bias/variance

### Avoidable Bias:

Cat classification Example

Humans (E/Boyes)	%1	odak noktası
Training error	%8	%7.5 ↑ ↓ %8 ↓
Dev. error	%10	Variance ↓ %10 ↓
		Focus on bias Focus on variance

Human-level error as a proxy for Bayes error

İnsanların performansı optimal error (Bayes error)'a yakın olduğu için training kismini birebileceğimiz ve andaki farklı "avoidable bias" diriz. Training error ile dev error arasındaki fark ise variance'dır.

## Understanding Human-level Performance:

Human-level error as a proxy for Bayes error

Medical image classification

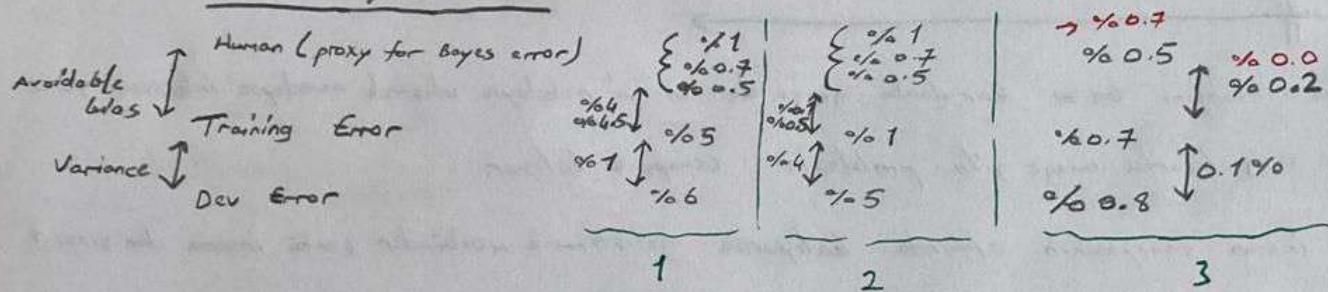
Suppose:

- (a) Typical human ..... %0.3 error
- (b) Typical doctor ..... %1 error
- (c) Experienced doctor ..... %0.7 error
- (d) Team of experienced doctor, ... %0.5 error

$$\text{Bayes error} \leq 0.5\%$$

Bayes error her zaman insanların ya da algoritmaların sistemin daha altında bir hataya sahip olacak durada optimal error'a ulaşmak istiyorsak (d) şıklını esas alırız ama anadımlı bir doktorдан data iyi' performans gösteren modelse (b) şıklını esas alırız.

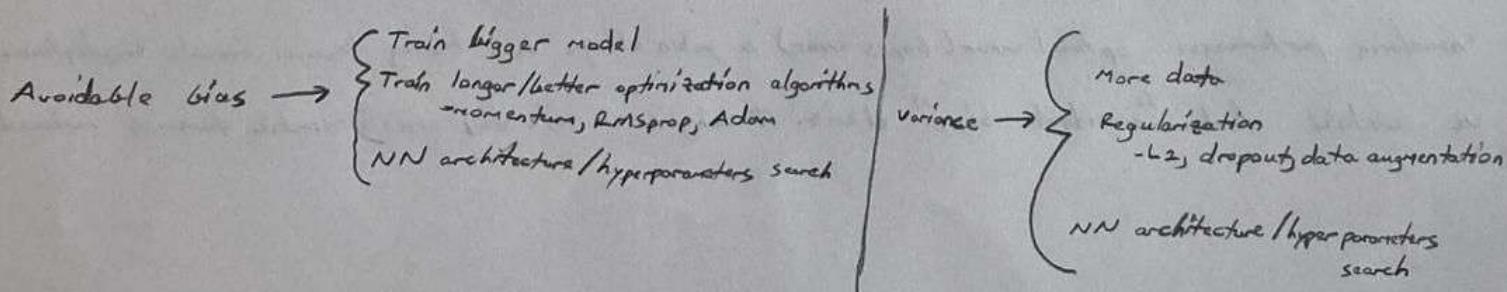
### Error Analysis Problem



1 numaralı örnekte training error ile human performance arasındaki fark çok büyük olduğu için değerini en fazla %7'ni ya da %5'ni olduğuna bakmadan bias ile ilgilenilir.

2 numaralı örnekte yine human performansı fark etmek için variance bakılmalıdır çünkü train ile dev arasında çok fark var.

3 numaralı örnekte değerler çok yakın olduğu için human error %5 seçiliyor bias (%0.2) variance dan (%0.1) 2 kat daha büyük olduğu için bias odaklanırız. Ama human error %7 sessizdir variance odaklanırdık, burada human error %0.5 alıp bias odaklı olan data mantıklı. Model insanların gösterdiği en iyi performansı gestikten sonra problem estis'i kadar az bir olmaz ve çeşitli problemler olabilir. Begin'e kadar insanlardan data iyi' performans gösteren modeller vardır ve çoğu structured data sayesinde geliştirilmiştir.



## Carrying Out Error Analysis:

Dev set için %90 accuracy, %10 error olsun.

Error Analysis:

- Get ~100 mislabeled dev set examples
- Count up how many are dogs

Cat classifier modeli'ne köpekleri kediler olarak sınıflandırabilir. Burada 100 tane total sınıflandırma örneği alıp koc tanesinin köpek olduğunu baktıktan sonra bir köpekle bu %10'luk error rate'ı %95 indirir. 50 tane ise %10'luk error rate'ı %5'e indirebilir. İlk senaryo için köpek resimlerini dizelemeye galmak mantıklı değil, ikinci senaryo için zaman ayırmaya değer.

### Evaluate multiple ideas in parallel

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats
- Fix great cats (lions, panthers, etc.) being misrecognized
- Improve performance on blurry images

Image	Dog	Great Cat	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
⋮					
% of total	1/8	1/43	1/67	1/12	

Birden çok fileri aynı anda düşünürsek yüzdesi en yüksek olan ile (6%) devam etmek sonucu daha fazla etki edecektir.

⚠️ Yanlış etikete sahip verileri deneyselde de önemlidir. Training loss, eğitimdeki durum çok büyük bir etkiye sahip değildir fakat dev/test işin bu durumla ilgilenmemelidir.

⚠️ Build your first system quickly, then iterate

## Training and Testing on Different Distributions:

Cat app example

Data from webpages

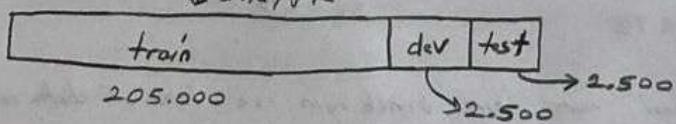
→ ≈ 200.000 example

Data from mobile app

→ ≈ 10.000

210.000  
↓  
shuffle

Option 1:



Shuffle ettiğinde aynı dağılım olusacağı için 2.500 verinin 2381 tanesi web'den

119 tanesi mobile app'dan gelecek bu da güncel olarak güncellenen verilere uygunun daha az

(41)

olacağrı antenine gelir. Bu durumda farklılık bir seyrek denenebilir.

Option 2:

web	mobile	dev	test
train set		mobile	
	2.500 mobile app		2.500 mobile app

Tüm web örneklerini train kisnesi için kullanıp mobile kismedeki de train, dev ve test kismına doğrultusak yeterince fazla fotoğraf örneklerine data fazla odaklanmış oluruz bu kise vadide bozun ise parçalama da uygun vadide ise yarar.

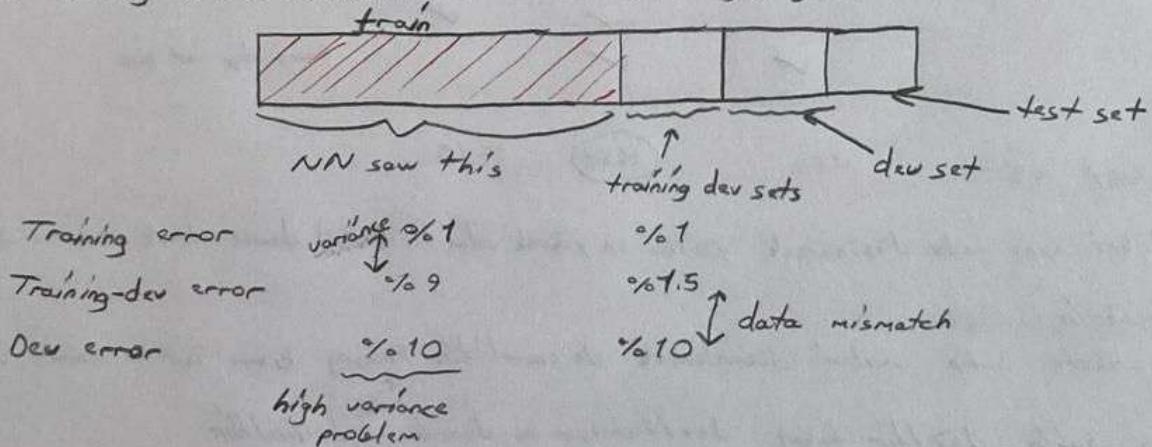
### Bias and Variance with Mismatched Data Distributions:

Cat classifier example Assume humans get  $\approx 60\%$  error

Training error .....  $\approx 61\%$  ↓  $\approx 60\%$   
Dev error .....  $\approx 610\%$

Aynı distribution'dan alınan veriye variance problemi olarak yorumlanabilir fakat diğer durumda aynı yorum yapılmaz.

\*\* Training-dev sets: same distribution as training set, but not used for training.

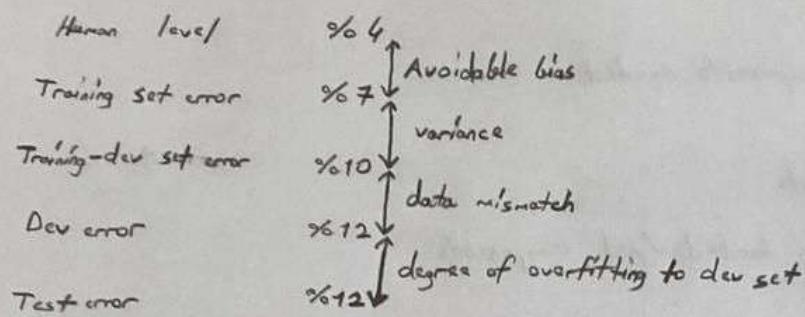


Train verisetinden bir kismını belirle training-dev olarak test etmek için kullanırsanız sonuçta problem varsa training silindilidir. Eğer burada problem yok dev,test kismeda problem varsa veriler aynı distribution'dan olmamış ve bir mismatch durumu oluşturur.

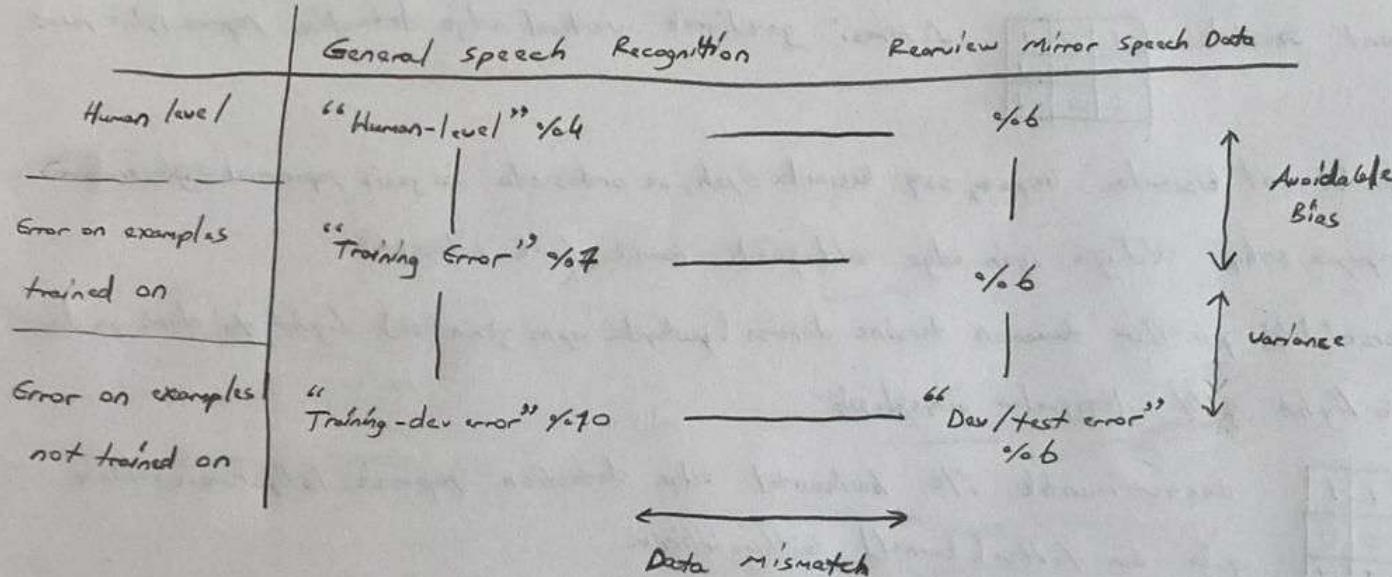
Human error	$\approx 60\%$	Avoidable bias
Training error	$\approx 610\%$	Avoidable bias
Training-dev error	$\approx 611\%$	Data mismatch
Dev error	$\approx 612\%$	

İlk örnek için bias problemi var ikinci örnek için ise bias ve data mismatch problemi var.

Bias and variance on mismatched training and dev/test sets



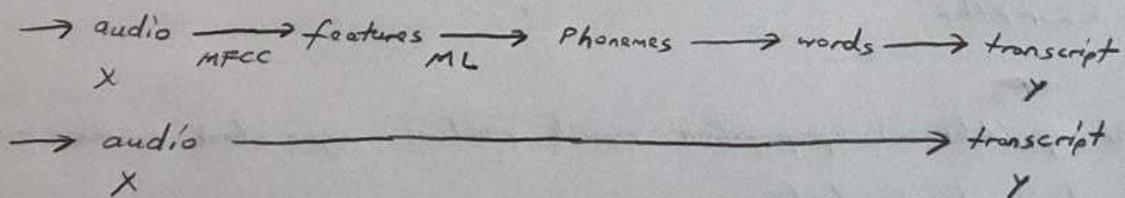
→ More General Formulation



### what is end-to-end Deep Learning?

Daha önceki yapılmış genel bir çok adımı tek isteme indirgeyen yapıdır.

Speech recognition example



ilk yapıdan ikinci yapıya geçirilir

- Face Recognition
- Machine Translation
- Estimating child's age

Yukarıda end-to-end DL örnekleri var ama daha kolay geçiti sağlarsak işin daha fazla veriye ihtiyaçınız vardır. Az veri için eski işlevlerini takip etmek daha yararlıdır.

## Pros:

- Let the data speak
- Less hand-designing of components needed

## Cons:

- May need large amount of data
- Excludes potentially useful hand-designed components

Convolutional Neural Networks:

Görsel üzerinde

1	0	-1
1	0	-1
1	0	-1

filtresi getdirmek vertical edge detection yapma işine yarar.

Bu filtre sol kısmında beyaz, sağ kısmında siyah ve ortasında ise gri & yapraklı sağlanacak bir yaprığa sahip olduğu için edge olduğunda genel fark edecektir.

Bir görseldeki pixeller tamamen tersine dönerse (yaklaşık ayna görisi) light to dark ya da dark to light gibi değişimler gerçekleşsin.

1	1	1
0	0	0
-1	-1	-1

Gençer mantık ile horizontal edge detection yapmak istiyorsa soldaki gibi bir filter (kernel) kullanılır.

1	0	-1
2	0	-2
1	0	-1

3	0	-3
10	0	-10
3	0	-3

Soldaki filtrelerde yukarıdaki mantığın orta katmandan girişinin artırmılmış şeklidir.

Sobel Filter Scharr Filter

Bu filtrelerdeki ağırlıklar backpropagation yaparak probleme uygun olarak ayarlanacaktır. Bu da verilen filtrelerin el ile tasarılananlardır.

Padding:

Bir görselle filtre uyguladiktan sonra olusacak output size'ini  $n-f+1$  formülü ile bulabiliriz.

Filtre istemisinin bazı negatif yönleri vardır. 1. si output size'ı 6x6'tır ve 2. olmak da orta katmandan olan pixeller daha fazla hesaba katılır fakat köşelerdeki pixeller sadece 1 kere hesaplanır. Bu da bu leyligi kaybetme ya da fazla odaklanmaya sebebi olabilir. Bunun için padding yapın extra bir border ekleyiniz.  $(n+2p-f+1) \times (n+2p-f+1)$  formülü ile padding sonrası filtrelenecek olusacak output size', bululuruz.

### \*\*\* "Valid" and "Same" convolutions

Valid convolution padding yapılmamış olsadır  $\rightarrow n-f+1$

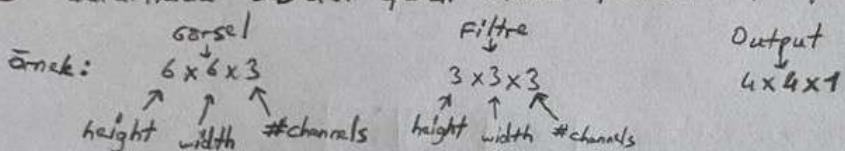
Same convolution padding yapılmış olsadır ve input-output size aynıdır  $\rightarrow n+2p-f+1$

Padding ise  $p = \frac{f-1}{2}$  ile bulunur ve ekstra olarak eklenir ( $f$  is usually odd)

### \*\*\* Strided convolution görselde her adımda attıma yaparak iteratif bir yapıdadır.

Formül:  $\left\lfloor \frac{n+2p-f+1}{s} \right\rfloor$  ile hesaplanır (output size)

3D durumlarında 2D'den farklı olarak filtrede ög boyutlu olur



$3 \times 3 \times 3$ 滤镜 kendisi ile görselin sol üstündeki  $3 \times 3 \times 3$  kisimlik toplam 27 sayıyı birlikte yapar ve sonuc output'un  $1/16$  elemanı olur. Red, green ve blue channel için ayrı ayrı işlem yapabiliriz. Örneğin sadece red channel'ın vertical edge detection filtersine sahipse ve green, blue tamamen 0 ise red channelda işlem yapmış oluruz.

\*\*\*\*\* Aynı görselle hem horizontal edge detection hem de vertical edge detection uygulayıp bunları birbir arkaya eklersek yukarıdaki örnek için  $4 \times 4 \times 2$  output elde etmiş oluruz. Bu işlem filtrenin genel yarısından ve tespit etmek istediğiniz özellikleri tespit ederek birbirine eklenir.

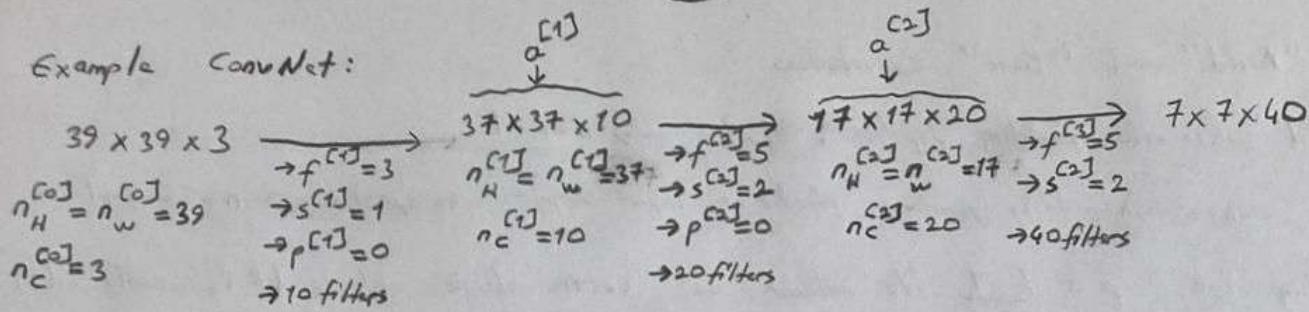
Genel formül:  $n-f+1 \times n-f+1 \times \sum_{\text{filters}}^{\text{#}}$  padding yapılmamış.

Daha önceki 6'da dikkat  $z^{[1]} = w^{[1]} \cdot a^{[0]} + b^{[1]}$  formüllerini kullanarak oluşturulan  $a^{[0]}$  image pixelidir,  $w^{[1]}$  filtrelerdir bu kisinin karşımı ise  $4 \times 4$ 'lik output oluyor. Bu outputa bias ekleyip relu'ya sokuttus sonra ise (her bir feature için ayrı ayrı yap) outputları birleştirip yararı birleştirildikten sonra ise  $a^{[1]}$  oluyor.

Koç parametre olduğunu hesaplayacağsak,  $3 \times 3 \times 3$  filter'in 27 tanesi ve bias ile toplam 28 yapar bunu 10 farklı feature için hesaplaysak toplam 280 parametreye sahip oluruz. Bu sayının gizli yani input image'dan bağımsız parametre sayısıının kis桔e tutulabilmesidir.

(45)

Example ConvNet:



Sonrasında  $7 \times 7 \times 40$ 'lik bir tane  $7 \times 40$  elemanlı sütun matris olarak aktivasyon fonksiyonuna verip tahminleri alırız.

Types of layer in a convolutional network:

- Convolution
- Pooling
- Fully connected

Pooling layer: Max pooling:

1	3	2	1
2	8	7	4
1	3	2	3
5	6	1	2

 $4 \times 4$ 

Hyperparameters:

$f = 2$

$s = 2$

9	2
6	3

 $2 \times 2$ 

No parameters!

Nothing to learn w/ gradient descent

Neural Network Example:

	Activation Shape	Activation Size	#parameters
Input:	(32, 32, 3)	3,072	0
CONV1 ( $f=5, s=1$ )	(28, 28, 6)	4,704	456
POOL1	(14, 14, 6)	1,176	0
CONV2 ( $f=5, s=1$ )	(10, 10, 16)	1,600	2,416
POOL2	(5, 5, 16)	400	0
FC 3	(120, 1)	120	48,120
FC 4	(84, 1)	84	10,156
Softmax	(10, 1)	10	850

~~Convolution kullanmanın sebebi~~ daha az parametre kullanmak istedir.

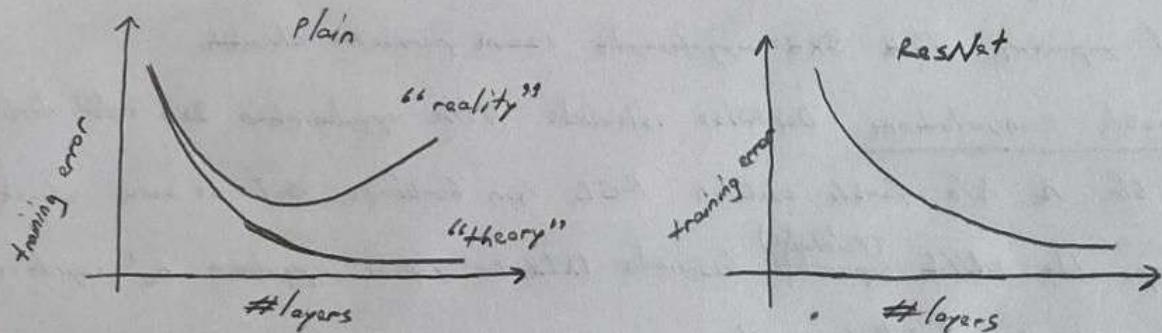
Tek bir filtreyi tüm girdis'e uygulayabiliriz. Bu yüzden:

- Parameter Sharing
- Sparsity of connections

Bu konuya önceli yapısı İki etkendir oljebiliti.

Fiziksel sok zar olduğu için modellerin mimarisini 2.hafıza pdf'inde var.

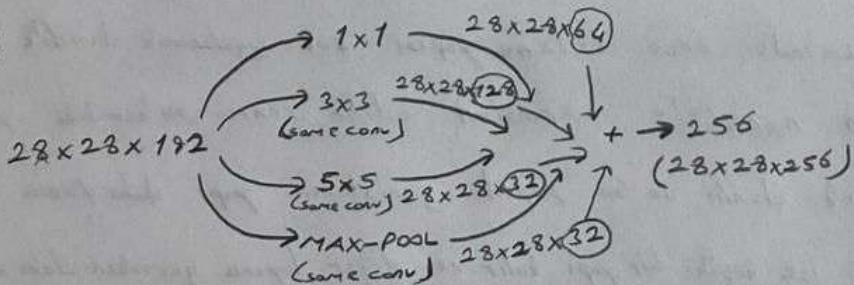
~~Büyük modellerde vanishing / exploding problemlen kurtulmak için her 2 katlıda 2 katlı layerdeki a degeri 2 katlı sonraki katlıda aktivasyona katılır ve bu yapısı daha derin nöral ağ eğitmeye yarar. Bu her 2 katlı yapısına residual blocks denir. Network'a ise Residual Network~~



Theoride model derinleştirme error azalması, gerçekte plain networkler için bu böyle deildir. ResNet bu durumu biraz sağlasa da sok derin ağlar için yine şereflidir.

~~1x1 convolution ya da network in network dediğimiz yapı derinlemesine işlem yapın convolution tipidir. Örneğin  $6 \times 6 \times 32$  bir girdisde  $1 \times 1 \times 32$  bir filtre uygularsa  $6 \times 6 \times \#filters$  şeklinde bir output alırız. Pooling layer height ve width tespitlerde bu yapı ise  $n_c$ , channel değerini azaltır. Örneğin  $28 \times 28 \times 192$  bir girdisde  $1 \times 1 \times 192$  boyutunda 32 filtre uygularsa  $28 \times 28 \times 32$  oluruz.~~

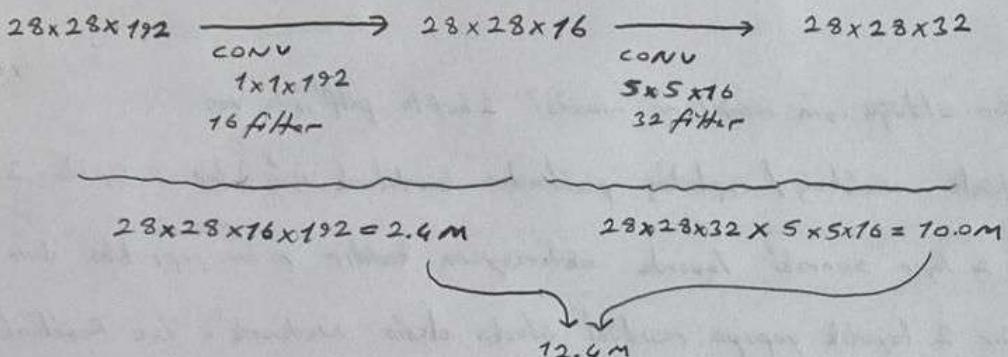
~~Inception Network birden çok farklı yapının kullanılmasına ve çıktıların birleştirilmesine dayanır. Örneğin~~



Model bu stacklenmiş yapının iyi performans sağlayacağı kullanacak.

Inception modellerin computational costları çok yüksek olmak fastla parametre olgun ve bunu azaltmamız gerekiyor.

Direkt  $5 \times 5$  bir filtre uygulanarak yerine önce  $1 \times 1$  uygulayıp sonra  $5 \times 5$  uygulayarak parametre sayesini  $1/10$  degere düşürebiliriz.



Eğer bunu işlemi yapmayıp direkt  $5 \times 5$  uygulayorsak 120M parametre alacaktı.

Depthwise Separable Convolution: Depthwise işleminde filtre uygulanırken  $3 \times 3$  sisteki gibi bu filtre her bir  $n_c$  işin farklı okuluların RGB işin her birine  $n_c$  ve sonuc olarak dene detaylı, yapı elde edilir. Separable isminde  $1 \times 1 \times n_c$  yapısı uygulanır.  $n_c$ ' sayıda bu filtre yapılıarak output derinliği ikiye düşer.

### Cost:

Normal convolution işin 2160 parametre olan bir modeli

Depthwise işin 432 ve pointwise işin 240 toplamda 672 parametre ile sonudsuz oluruz.

$$\frac{672}{2160} = 0.31 \text{ katı kadar maliyeti diğer buna } = \frac{1}{n_c!} + \frac{1}{f^2} \text{ formülü ile de hesaplanır.}$$

$\downarrow n_c$

$$= \frac{1}{5} + \frac{1}{3^2} \text{ (Gizem örneğinde işin)}$$

**[MobileNet]** architecture bunu yapı kullanılarak tasarlandı.

MobileNetV2 seriminde önce  $1 \times 1 \times n_c$  yapısı çok uygulanarak derinlik kazandırılır. 18 tane mesela  $n \times n \times 3$ 'tan  $n \times n \times 18$ 'e expansion edilir sonra yukarıdaki yapı kullanılır. Bu yapıya bottleneck denir ve bunu yapıcı genişleyen yapı daha fazla derinlige ulaşmasını sağlar. Sıktı isminde işe kocası bir yapı kalır ve diğer layera geçerken data az parametre olurken

## Object Localization:

Image classification işlemiinde bir resim araba resmi mi değil mi sadece bunu tespit ediyorduk. Classification with localization problemında ise tespit edilen nesne özneginin bir dikdörtgen isine aitira. Detection dediginiz kisimda ise birden çok objeki tespit etme olayı vardir.

Classification modellerinde öznegin son layerda softmax kullanılıp 1-pedestrian, 2-car, 3-motorcycle, 4-background gibi sınıflar tahmin ediliyordu. Localization'da output olarak class dışında  $b_x, b_y, b_h, b_w$  gibi sıfırlar da alınır.  $b_x, b_y \rightarrow x \text{ ve } y$  değerlerini belirtirken  $b_h$  yüksekliği,  $b_w$  ise yatay kenarın uzunluğunu verir. Bu yapıda gärselin sol üst köşesi  $(0,0)$  ve sağ alt köşesi  $(1,1)$  koordinatı döşenülerek ayarlanır.

Target label  $y =$  need to output  $b_x, b_y, b_h, b_w$ , class label (1-4)

$$y = \begin{cases} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{cases}$$

is there any object?  
1 or 0

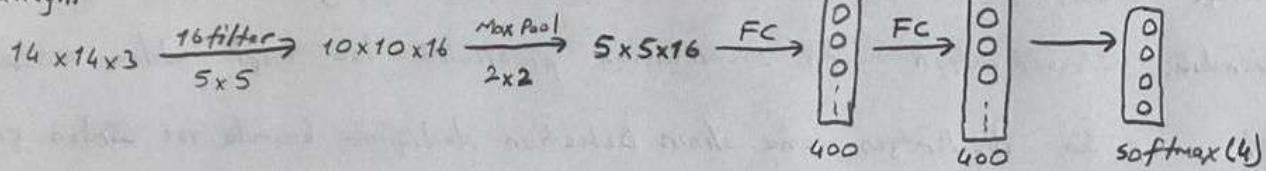
4 - background değeri hantangi bir obje olmadığını söyleyir.  
Eğer background sonucu gelirse  $p_c$ 'den au durumda  
 $c_1, c_2, c_3 \rightarrow$  gerçekte yoktur. Diğer durumda kapsı kullanılır  
 $b_x, b_y, b_h, b_w$

Loss Function:  $L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \downarrow \\ & \text{there is no object} \end{cases}$

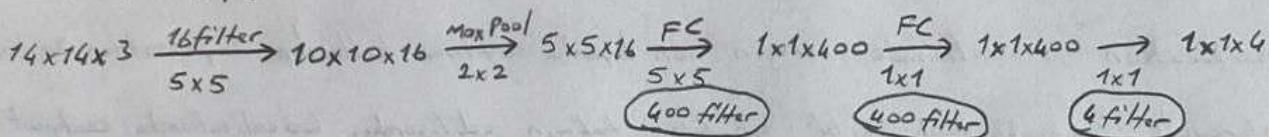
Veriseti oluştururken mesela bir insan gözeni tespit eden bir model geliştirmek istiyorsak gözenin içerisindeki noktalardan  $x \text{ ve } y$  sıfırları alayız. Mesela 64x64 sıfır isin toplam 128 girdi ve bir de insan mi değil mi kontrolü  $p_c$  (1) toplama 128 girdi ile aynı eğitilir. Pose estimation isinde kritik bölgelerin koordinatları alınarak model eğitilebilir. ~~detektör~~ Sliding windows detection farklı boyutluk pencereler ile tüm görselde her biri isin aranır yapar. Bu işlem her farklı pencere isin görselde geçtiği isin iyi bir algoritma denemeli. Fakat bu algoritma daha verimli ve computational costu daha iyi bir şekilde tasarlayabiliriz.

## Turning FC layer into convolutional layers

Örneğin

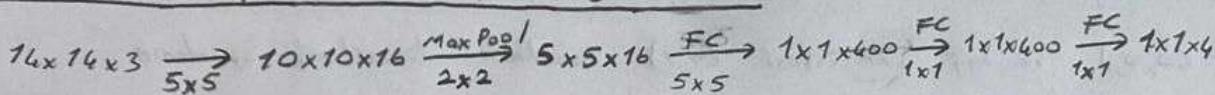


Yukarıdaki yapıyı basit bir işlem ile dönüştirebiliriz.

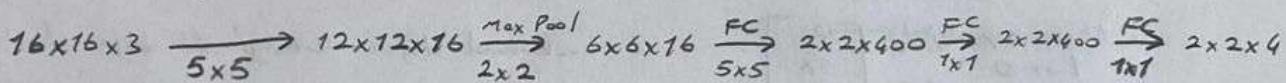


Bu yapı sliding window algoritması için faydalı olabilir.

### Convolution Implementation of Sliding Windows:



Örneğin 16x16x3 görsel için



Yukarıdaki yapıda oluşan  $2 \times 2 \times 4$  yapısına sahip output  $14 \times 14$  çerçevesinden oluşan ve 2 stride ile aşağı ve sağa kaydırılan yapının sonucunu gösterir. Bu örnük tüm farklı boyuttaki örnekler için kullanılabilir.

\* \* \* Burada gezdirilen windowlar bir seilde görseli istenilebilir. Bu problem için YOLO algoritması kullanıyor ve bu algoritma örneğin bir görseli 9 farklı hücre oluşturuyor ve objenin orta noktasıının hangi hücrede kaldığını batarak işlem yapıyor. Bounding box girmek için ise değerler bulunurken her hücre sol üst köşesi ( $l, u$ ) ve sağ alt köşesi ( $r, d$ ) olarak dırırıltır ve ona göre hesaplanır. Eğer görsel boyutu ve ikinci hücreye sınırlıysa değer 1'den büyük olur. YOLO açılımı You Only Look Once olarak bilinir.

Eğer birden fazla hücre içine objenin merkez noktası giriyor ise bu durumda objenin ayrı ayrı hücrelerdeki boyutu bulunur ve oranı alınır. Eğer 0.5-0.6 gibi bir değerden büyükse büyük boyutu içeren hücre esas alınır. Bu işlem Intersection over Union (IoU) denir.

### Non-Max Supression Algorithm

Bir aradıktan hizre boyutuna da bağlı olarak birde fazla bulunabilir. Mesela  $19 \times 19$  grid'e götürürse bir görselde birde çok gridde aradıktan bulunacaktır. Burada grid seginleri bir absolute fonksiyonu ile yapılır.

Each output prediction is:

$$\begin{bmatrix} P_C \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$

Discard all boxes with  $P_C \leq 0.6$

while there are any remaining boxes:

- Pick the box with the largest  $P_C$  output that as a prediction
- Discard any remaining box with  $IoU > 0.5$  with the box output in the previous step.

### Anchor Boxes:

Her grid cell tek bir hizre içermiyordu, eğer her hizre birden fazla obje içermeye gəsənə sahip olursa bu durumda nə olur?

Üst üste gelen objelərdə anchor boxes tullanılabilir.

$$\text{General } y = \begin{bmatrix} P_C \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \\ P_C \\ b_x \\ \vdots \\ C_3 \end{bmatrix} \quad \left. \begin{array}{l} \text{Anchor Box 1} \\ \text{Anchor Box 2} \end{array} \right\} \quad \text{Dimension: } (16, 7) \text{ sənki 2 obje var tək oğide } (3, 7)$$

### Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint.

$$\text{Output } y = 3 \times 3 \times 8$$

### With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

$$\text{Output } y: 3 \times 3 \times (2 \times 8) = 3 \times 3 \times 16$$

Eğer 2 anchor box içinde 3 obje bulunursa bu durumda algoritma problem yaşayır.

## Semantic Segmentation with U-Net:

Görseller tespit edildikten sonra sınırları sınırlayıp ve hangi pixeller objeye ait olduğunu farklı olduğunu anlamak için bu yapı kullanılır.

Bu yapı erişler için sorunlu olabilir yolların tespitinde, hastalıklarda göğüs problemleri ya da boyutları tespitlerinde çok yararlıdır.

### Per-pixel class labels

1. car

2. building

3. road

Bir görselde yol, araba ve bina olduğumu düşünüm bunların her birini detaylı bir şekilde tespit etmek istiyorsak arabaya ait her pixel'e 1 binaya 2 ve yola ait her pixel'e 3 vererek bir segmentation map oluşturabiliriz. Bu diğer işlemlere göre çok daha büyük veri eğitliğimiz anlamına gelir.

## Deep Learning for Semantic Segmentation

Bu yapıyı eğitmek için farklı bir networke ihtiyacımız var. Diğer networklerdeki son katmanları kaldırıp kisiler yapıyı başlangıçtaki boyutlara uygunen yeni bir yapıya ihtiyacımız var sınıflı class prediction vs. yapmayıza aynı görselin segmentation yapılması halini istiyoruz. Bunun için transpose convolutions kullanırız. Bu yapıdaki asıl olay her pixelde uygun label değerini vermek ve modelde girdi sağlanmaktır.

### Transpose Convolutions:

#### Normal Convolution

$$6 \times 6 \times 3 \rightarrow 3 \times 3 \times 3 \rightarrow 4 \times 4 \times 5 \\ (\text{5 filter})$$

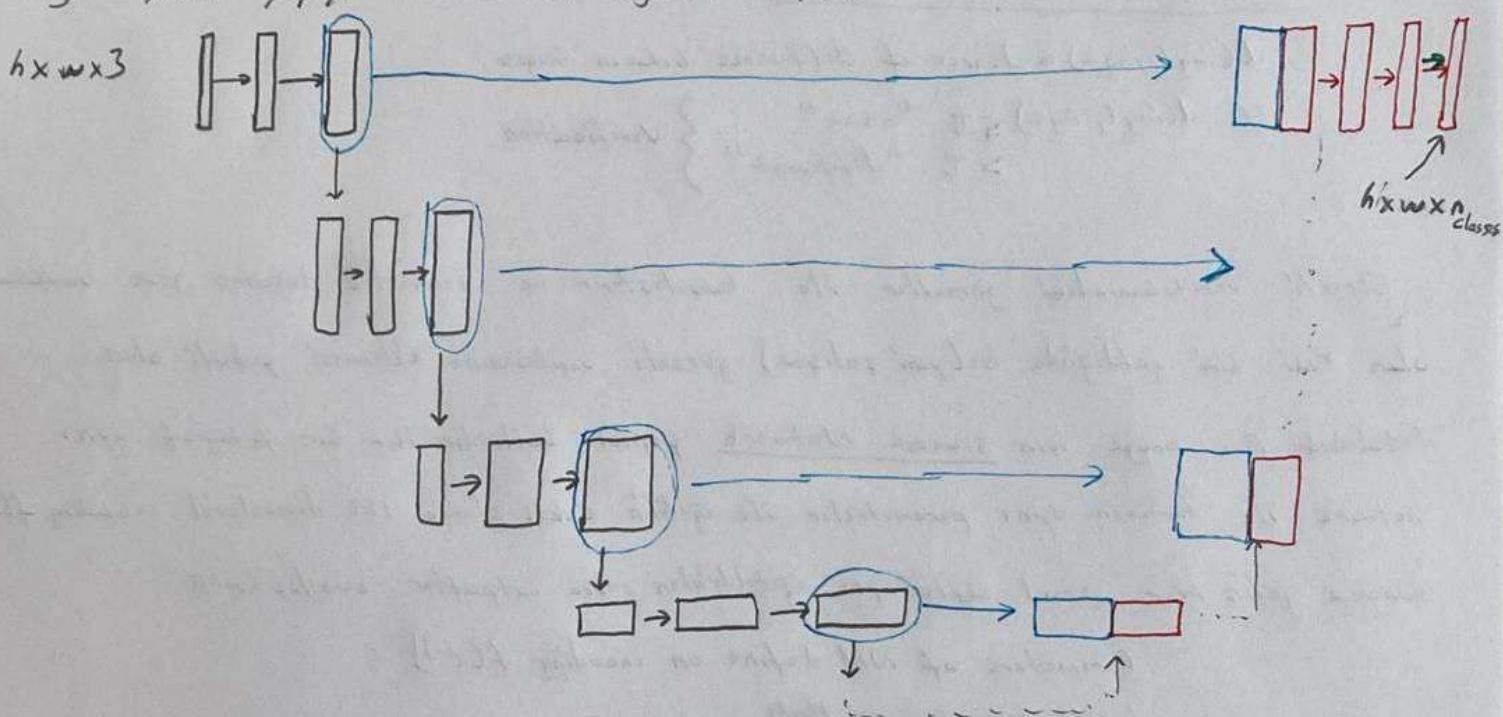
#### Transpose Convolution

$$2 \times 2 \times 3 \times 3 \rightarrow 4 \times 4 \quad (\text{Bigger than original input}) \\ (\text{filter})$$

Transpose convolutionda inputtaki her eleman tüm filtre ile çarpılıp filtre boyu kadar output kismına eklenir. Mesela yukarıdaki Transpose Convolution örneği için padding=1 ve stride=2 kullanırsak sol üsteki input ile filtre çarpılıp output kismında sol üst (padding dahil) kısma yazılır ve sonra stride kadar kayip diğer eleman için uygunca ortak kesişen noktalardaki variler ise toplanır ve bu sisteme yapı köprü.

### U-Net Architecture:

Bu mimaride önce görseli 64x64'ye sonra 32x32'ye getirmek ilk etapta bilgi kaybına sona  
ise daha az bilgi içeren yapıyı 64x64'ye ugrasır bu işe yine sonucta olumsuz duruma yol  
açar. Bu nedenle aşın skip connection kullanıp ilk layerden en son layer'e ait bilgiyi  
bilgi içeren yapıyı korunamızı sağlar. Bu işi yapılı birleştiririz.



Genel yapıda her seferde bir Conv, ReLU yapısıdır, alt katmanlarda genellikle Max Pool, üst katmanlarda  
genellikle Trans Convolution, matris şeklinde olan yapı skip connection veya son layer olan türünden  
yapılır ise Conv(1x1) antrensör gelir. Görüldede .... kısımını sigmoid ile  
yazıldığında genel yapı budur. Son layerda class sayısı kadar output gelir (class number).

### Face Verification vs. Face Recognition:

#### Verification:

- Input image, name / ID
- Output whether the input image is that of the claimed person

99.9 accuracy

#### Recognition:

- Has a database of K persons
- Get an input image
- Output ID if the image is any of the K persons (or "not recognized")

99%

### One-Shot Learning:

Bir yüz tanıma sistemi yapmak istediğimizde örneğin salıncaların çok fazla fotoğrafını bulamayabiliriz. Tek bir fotoğraf da insanların kendilerinde değişebilecek özelliklerinde algılayamaz ya da bu nedenle CNN'İN çok zordur bu durum ise problem oluşturur. Buna one-shot learning problem denir.

#### Learning a "similarity" function:

$d(\text{img}_1, \text{img}_2) = \text{degree of difference between images}$

If  $d(\text{img}_1, \text{img}_2) \leq T$  "same" } Verification  
 $> T$  "different" }

Görseli veritabanındaki görseller ile karşılaştırır ve benzerlik değerine göre verification olur. Yani biri geldiğinde isen (yani eşit) görseli veritabanına etiketlenmesi yetkilidir.

~~But~~ Bu projede isin Siamese Network yapısı kullanılır. Her bir fotoğraf aynı network ile tamamen aynı parametreler ile eğitilir. Çekti olarak 128 dimensional encoding  $f(x^{(i)})$  sonucu gelir. Her görsel ayrı ayrı eğitildikten sonra outputlar karşılaştırılır.

Parameters of NN define an encoding  $f(x^{(i)})$

Learn parameters so that:

If  $x^{(i)}, x^{(j)}$  are the same person,  $\|f(x^{(i)}) - f(x^{(j)})\|^2$  is small

If  $x^{(i)}, x^{(j)}$  are different persons,  $\|f(x^{(i)}) - f(x^{(j)})\|^2$  is large

### Triplet Loss:

Bir Anchor, bir Positive ve bir de Negative örnek düşünlerek tasarlandığı isin bu adı almıştır. Temelinde Anchor'un positive ve negatif olan farklılarından türetilen.

wants

$$\underbrace{\frac{\|f(A) - f(P)\|^2}{d(A, P)}}_{0.2} + \alpha \leq \underbrace{\frac{\|f(A) - f(N)\|^2}{d(A, N)}}_{margin}$$

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha \leq 0$$

↑  
margin (SVM gibi)

Positive ve negatif görsel / ile fark 0 olunca sonuc da 0 oluyordu ve bu istenmeyen bir sonuc daha iyi bir model istiyorsak da bu üçte fark olusturan bir yapıda olmalı, tıpkı farklılıklar göstermek. Bu işi ayarlamanız için de SVM yapısındaki benzer bir margin vardır.

Loss Function:

Given 3 images  $A, P, N$ :

$$\begin{aligned} \text{Loss} \quad L(A, P, N) &= \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \lambda_1, 0) \\ \text{cost} \quad J &= \sum_{i=1}^m L(A^{(i)}, P^{(i)}, N^{(i)}) \end{aligned}$$

Training set = 106 pictures of 1K person

Her çalışanın birden fazla fotoğrafı olmalıdır ve loss gradient descent ile azaltılmalı  
calısır.

~~\*\*~~ Görüllerin rastgele seçimi bir problemdir çünkü birinden farklı görüntülerini  
az çok kolay bulabılır. Bu da modelin eğitime göre bir kayıp katması. Fayda sağlanası  
için eğitilmesi zor triplets segmeti genelde zor eğitim öngö  $d(A, P) \approx d(A, N)$  olurken

$$\hat{y} = \sigma \left( \sum_{k=1}^{128} w_k [f(x^{(i)})_k - f(x^{(j)})_k] + b \right)$$

Supervised Learning ve output 0 veya 1 ( $L_{pos/HF}$ ) gelecektir

Neural Style Transfer:

Bu yapı, iki farklı görseli bir content dilgi style olmak üzere birleştirir ve  
yeni bir görsel oluşturur.

Converts ilk kırımlarda daha ince noktalara odaklanan content yapısını  
fakat daha anlamlı, daha net görseller oluşturmaya çalışır

Cost Function:

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

Generated image

Find the generated image  $G$ :

1. Initiate  $G$  randomly

$G: 100 \times 100 \times 3$

2. Use gradient descent to minimize  $J(G)$

$$G := G - \frac{\partial J(G)}{\partial G}$$

### Content cost function:

$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

- Say you use hidden layers  $l$  to compute content cost.
- Use pre-trained ConvNet
- Let  $a^{[l]}(c)$  and  $a^{[l]}(G)$  be the activation of layer  $l$  on the images
- If  $a^{[l]}(c)$  and  $a^{[l]}(G)$  are similar, both images have similar content

$$J_{\text{content}}(C, G) = \frac{1}{2} \|a^{[l]}(c) - a^{[l]}(G)\|_F^2$$

### Style Cost Function:

Birutular arasında korelasyon olması arkaya gelen bloklardaki featureların aynı parçaya ait olabileceğini ifade eder. Görsel generate ettiğinden sonra da arkaya gelen bloklar arasındaki korelasyona bakınız.

Style Matrix:

Let  $a^{[l]}_{ijk}$  = activation at  $(i, j, k)$ .  $G^{[l]}$  is  $n_h \times n_w \times n_c$

$$\rightarrow G^{[l](s)}_{kk'} = \sum_{i=1}^{n_h} \sum_{j=1}^{n_w} a^{[l](s)}_{ijk} a^{[l](s)}_{ijk'}$$

$$\rightarrow G^{[l](G)}_{kk'} = \sum_{i=1}^{n_h} \sum_{j=1}^{n_w} a^{[l](G)}_{ijk} a^{[l](G)}_{ijk'}$$

$$J_{\text{style}}(S, G) = \frac{1}{(\dots)} \|G^{[l](s)} - G^{[l](G)}\|_F^2$$

$$= \frac{1}{(2n_h n_w n_c)^2} \sum_k \sum_{k'} (G^{[l](s)}_{kk'} - G^{[l](G)}_{kk'})^2$$

$$J_{\text{style}}(S, G) = \sum_l \lambda^{[l]} J_{\text{style}}^{[l]}(S, G)$$

## 5-) Sequence Models:

Examples of sequence data:

Speech Recognition

Music Generation

Sentiment classification

DNA sequence analysis

Machine Translation

Video activity recognition

Name entity recognition

Bir cümlede kelimeler  $x^{(t)}$  notasyunu ile birlikte ille kelime igo  $x^{(1)}$ , sonra  $x^{(2)}$  ...  
 $y$  (label) isimde de aynıdır.  $T_x$  ve  $T_y$  değeri ise cümledeki kelime sayısidır. Her kelime  
 için kelimelerin olduğu Vocabulary array'i içerisinde kendisi yerini işaretleyerek kendini tespit eder.  
 Eğer bilinenen ya da fakat bir kelime varsa bu arrayin son üyesi olan unknown kisimina  
 dahil edilir.

## Recurrent Neural Networks:

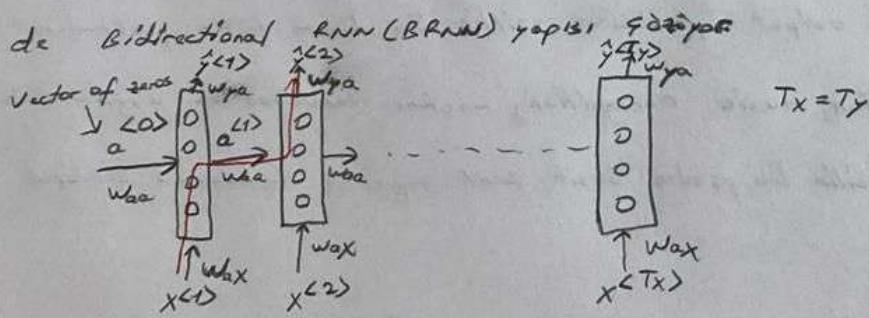
Standart GRU'nun tek farkı sırası verilen işin problemlerini解决

1. Inputs, outputs can be different lengths in different examples

2. Doesn't share features learned across different positions of text

Sıradan GRU'nun yapısı her kelimeyi birbirinden bağımsız değerlendirdir fakat cümlede kelimeler  
 anlamal yapısı konum adına ilişkileri ile bağlantılıları vardır.

Recurrent neural network yapısı daha önceki kelimelerin değerini da hesaba katarak bir yapısın  
 fakat RNN yapısının etkiliği ise sonraki kelimeleri hesaba katmasınaidir. Bu problemi



önceki kelimeler sırası ile tüm layerlara katkı sağlarlar.

## Forward Propagation:

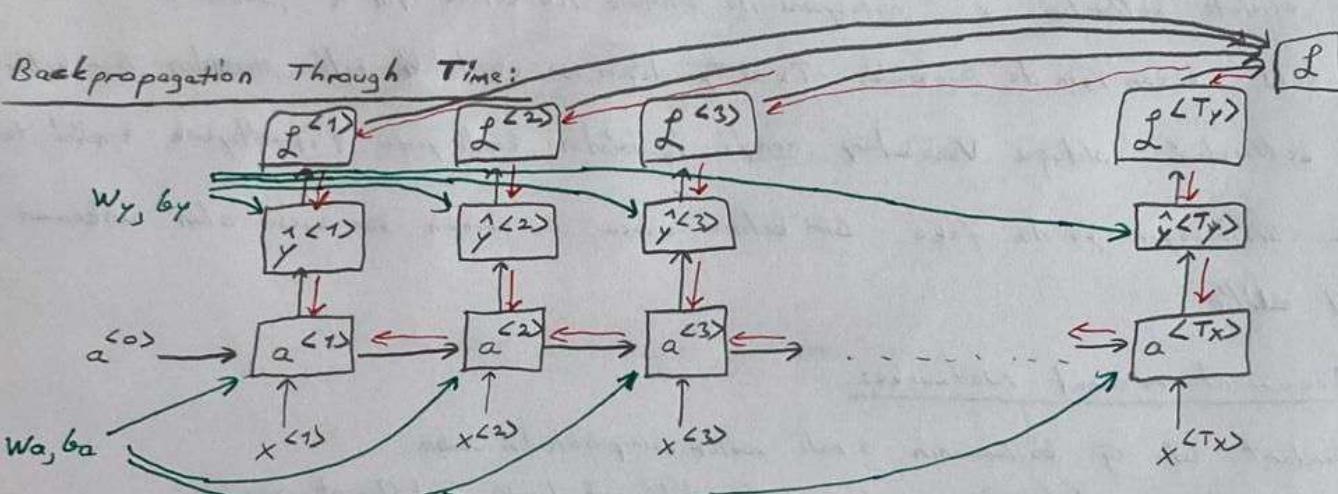
$$\begin{aligned}
 a^{(0)} &= 0 \\
 a^{(1)} &= g_1(w_{aa}a^{(0)} + w_{ax}x^{(1)} + b_a) \leftarrow \text{tanh / arctan ReLU} \\
 y^{(1)} &= g_2(w_{ya}a^{(1)} + b_y) \leftarrow \text{sigmoid} \\
 a^{(2)} &= g_1(w_{aa}a^{(1)} + w_{ax}x^{(2)} + b_a) \\
 y^{(2)} &= g_2(w_{ya}a^{(2)} + b_y)
 \end{aligned}$$

Simplified RNN activation:

$$\begin{aligned} a^{<t>} &= g(W_{aa} a^{<t-1>} + W_{ax} x^{<t>} + b_a) \\ &\quad \downarrow \quad \downarrow \\ &\quad (100, 100) \quad (100, 10,000) \\ \hat{y}^{<t>} &= g(W_{ya} a^{<t>} + b_y) \end{aligned}$$

elaborate computation (-)

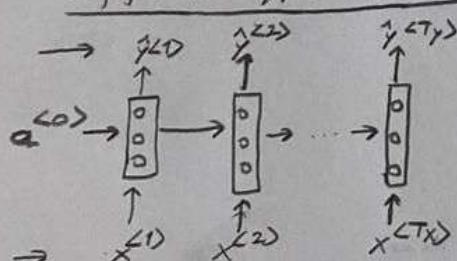
$$\begin{aligned} a^{<t>} &= g(W_a [a^{<t-1>} x^{<t>}] + b_a) \\ &\quad \uparrow \quad \uparrow \\ &\quad 100 \quad 100 \\ &\quad \overbrace{\quad\quad\quad\quad\quad}^{\quad\quad\quad\quad\quad} = W_a \\ &\quad (100, 10,000) \\ \left[ a^{<t-1>} x^{<t>} \right] &= \left[ \frac{a^{<t-1>}}{x^{<t>}} \right] \uparrow 100 \quad \downarrow 10,000 \\ \left[ W_{aa} \mid W_{ax} \right] \left[ \begin{matrix} a^{<t-1>} \\ x^{<t>} \end{matrix} \right] &= W_{aa} a^{<t-1>} + W_{ax} x^{<t>} \end{aligned}$$



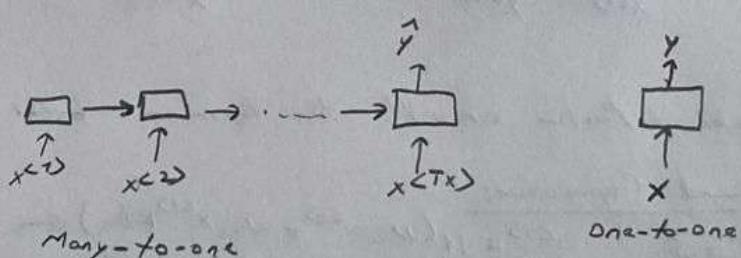
$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1-y^{<t>}) \log (1-\hat{y}^{<t>})$$

$$L(\hat{y}, y) = \sum_{t=1}^{Ty} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

Her zaman analizinde sırayla yeni bir adım işlendiği için "Backpropagation through time" denir. Bu temel yapıda input  $T_x$  ile output  $T_y$  birbirine eşittir. Bu durum bazı problemlere uygulanamaz çünkü speech recognition, music recognition, machine translation vb. uygulamalarda input ve output tamamen farklı olabilir bu yüzden basit RNN yapısı ile ilgilenmemek gerekiyor.

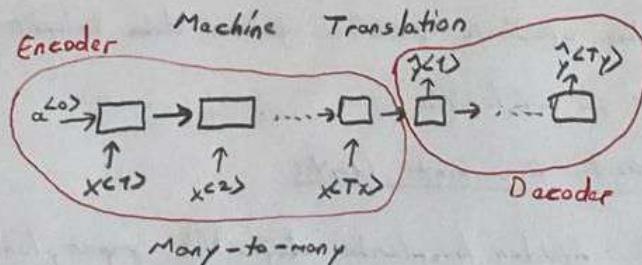
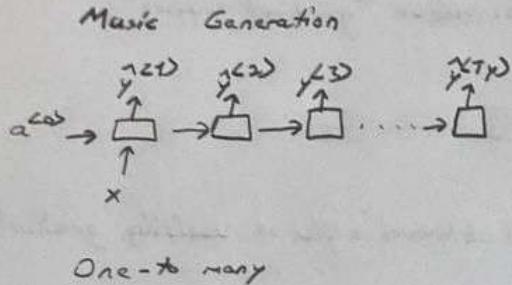
Different Types of RNNs:

Many-to-many



Many-to-one

One-to-one



### Language Model and Sequence Generation:

What is language modelling?

The apple and pair salad.

The apple and pear salad.

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

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

Bir konusmada, kitapta, gazetede geçen bir cümle'nin yapıya uygun olma ihtiyacılığı bulunur. Bu durum dil modellerinin önceliği bir parçasıdır.

Mesela "Cats average 15 hours of sleep a day. <eos>" bu tarz bir yapıyı RNN verdiğimizde ilk önce mutlakelik kelimelerden ilerleyince tahmin ediliyor, sonraki kelimeye dek attenda ileri birde tahmin ediliyor. En son time窗 ile sağlanan koşulu ile <eos> tahmin ediliyor. Her time窗'ta sadece tek bir kelime tahmin ediliyor ve yapıda "softmax" activation kullanılır.

3 kelimeyle cümle örneğinin  $P(y^{(1)}, y^{(2)}, y^{(3)}) = P(y^{(1)}) \cdot P(y^{(2)} | y^{(1)}) \cdot P(y^{(3)} | y^{(1)}, y^{(2)})$  denklemi oluşturur.

Character-level tokenization ile word-level tokenization farklıdır. Word-level tokenizationda bilinmeyen kelimelerin vocabularyda unknown kismına kayıtlı yorum fakat character-level kismında yok. character-level çok computational-cost ister ve ayrıca word-level gibi tüm candidat'ları yakalayamaz.

### Vanishing Gradients with RNNs:

The cat, which actually ate ..... was full

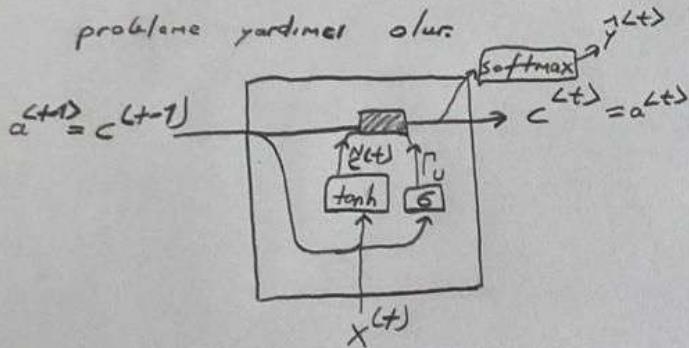
The cats, which ..... were full

RNN kullanarak bu tarz cümle örneklerini eğitirsek ve ara türmler uzunsa bu daha kötü, devrin bir normal ağı antikama gelir. Daha kötü ağıda vanishing gradient problem olur ve singular plural anlam cümle koşulları sonuna kadar sağlanamayabilir. Bu durum ciddi problemlerden biridir.

Exploding gradients problemi görece dəha tələydir. NaN həsi və "gradient clipping" yəntəm işləşəbilir.

### Gated Recurrent Unit:

GRU hidden layerlarda deqisiklik yapar, long-range bilgi aktarımını sağlar ve vanishing gradient probleme yordam olur.



c = memory cell

$$\rightarrow c^{t+1} = a^{t+1}$$

$$\rightarrow c^{t+1} = \tanh(w_c [c^{t+1}, x^{t+1}] + b_c)$$

$$\rightarrow \pi_u = \sigma(w_u [c^{t+1}, x^{t+1}] + b_u)$$

↑ "update"

$$c^{t+1} = \pi_u * c^{t+1} + (1 - \pi_u) * c^{t+1}$$

↓ element-wise

Sigmoid fonksiyonu ilə deyer 1 ni 0 nı, 6alır. Bu bilgi area katmanları boyu etibarənən gəzər və gerək yerde tekrar təndi deyerini alır.  $c^{t+1}$ ,  $c^{t+1}$  bu gəzər deyerlər 100-dimensional vector olub və bit həzində işləm yaparak cümle işləmə kəlləmələrin deyerləri ayarlanabilir.

Yukarıda simplified GRU vardır. Full GRU'da deqisiklik işlək hüdudların əlavə edilmişini ehtiva edir.

$$\begin{aligned} \tilde{c}^{t+1} &= \tanh(w_c [\pi_r * \underbrace{c^{t+1}}_{\text{update}} \times x^{t+1}] + b_c) \\ \pi_r &= \sigma(w_r [c^{t+1}, x^{t+1}] + b_r) \end{aligned}$$

### LSTM:

LSTM, GRU yapısının dəha gəlmiş hallidən və asagadakı dənkəndən əlavə kontrollu dəlliyyətini təqdim edir.

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

$$\xrightarrow{\text{update}} \pi_u = \sigma(w_u [a^{t+1}, x^{t+1}] + b_u)$$

$$\xrightarrow{\text{forget}} \pi_f = \sigma(w_f [a^{t+1}, x^{t+1}] + b_f)$$

$$\xrightarrow{\text{output}} \pi_o = \sigma(w_o [a^{t+1}, x^{t+1}] + b_o)$$

$$c^{t+1} = \pi_u * \tilde{c}^{t+1} + \pi_f * c^{t+1}$$

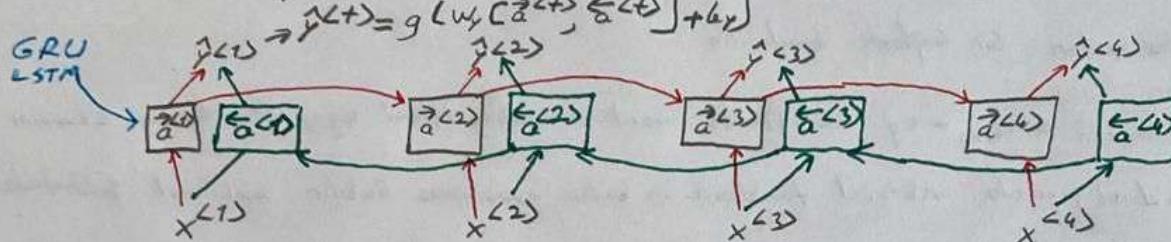
$$a^{t+1} = \pi_o * \tanh(c^{t+1})$$

Bidirectional RNN:

Bu yapı, gelecek kelimeinden de bilgi almaktan için tasarlanmıştır. NLP, text tabanlı problemlerde LSTM ile sıklıkla kullanılır.

He said, "Teddy bears are on sale!"  
He said, "Teddy Roosevelt was a great President!"

Bu cümlelerde Teddy, insan mı, devil mi anlamak için BRNN gereklidir.



Acyclic Graph, BRNN with LSTM

Kelimeleri one-hot encoding ile vocabulary arasında işaretlediğimizde kelimeleri temsil edebiliyoruz fakat kelimeler arasında bir ilişkisi, bağlantı kurmak zorda. Bu yüzden word embeddings ile kelimelerin farklı featurelere olan yakınlığına test ediyoruz ve bu değer 0-1 arasında bir sonuc alıyor. En sonunda oluşturulan sonuçlar arasında ise bir vektörel yakınıklığı oluşturuyor ve yakınlarına sahip kelimeler vektörel olarak birbirine yakın konumlanıyor.

Using word Embeddings:

Named entity recognition example:

1 ↑ Sally	1 ↑ Johnson	0 ↑ is	0 ↑ an	0 ↑ orange	0 ↑ farmer.
Robert Lin is an apple farmer.					
a durian cultivator.					

Word embeddings ile 1. cümlede sahipset, 2. cümledeki Robert Lin isminin ismi olduğunu da makine algılayabilir. Çünkü orange ve apple birbirine yakın olan vektörler ve aynı anda 3. cümlede ise durian nadir bir meyve ve cultivator, farmer ile yakın anlamda sahip olduğu için çok büyük eğitimin verilerinde buna da bağlantı kurulabilir. Makine cultivator'ın bir insan olduğu sevgisini yapabilmek internette belki 1B ya da 100Billion (milyar) kadar user test yapıları unlabeled şekilde bulunabilir ve makine pattern'i anlamaya başlar. Bu yapıda transfer learning'e katkı sağlar. 100.000 ya da 600.000 daha az labeled data'ya sahipset bunun ile gaza sıkışmış bir modelde sahip olursa oluruz.

Transfer Learning and word embeddings:

1. Learn word embeddings from large text corpus. (1-100B words)
2. (or download pre-trained embedding online.)
3. Transfer embedding to new task with smaller training set.  
(say, 100k words) (10,000 dimensional one-hot per one 300 dimensional dense layer)
3. Optional: Continue to finetune the word embeddings with new data.

~~Siamese Network~~ Siamese Network (face encoding) yapısına benzer bir şekilde bir eğitilip sonra farklı sorulara göre cevaplar verme能力和 kumulatif

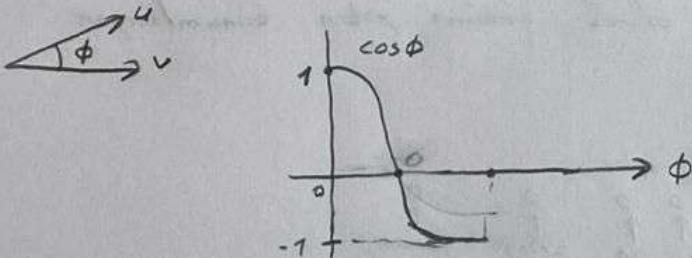
~~Cosine-embedding =  $e_{king} - e_3$~~  Bu şekilde word embeddinglerin  $e_king - e_{queen}$  arasındaki ve  $e_king$  ile  $e_{woman}$  arasındaki açıları ölçülür. Bu açılarla word embeddinglerin benzerlik seviyesi hesaplanır. Bu açılarla word embeddinglerin benzerlik seviyesi hesaplanır. Bu açılarla word embeddinglerin benzerlik seviyesi hesaplanır.

Find word w:  $\operatorname{argmax}_{\text{sim}}(\text{ew}, \text{e}_k - \text{e}_{\text{king}} + \text{e}_{\text{woman}})$

Burada nonlinearity oluşturmak ve 300D yapıyı 2D'ye çevirme "t-SNE" isimli yapı vardır.

Bu yapı görselleştirirken ve diğer yöntemlerde yararlı olabilir. Ayrıca bazı işgilleri korur.

$$\text{cosine similarity} \Rightarrow \text{sim}(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2} \quad \text{setinde formüle edilebilir}$$

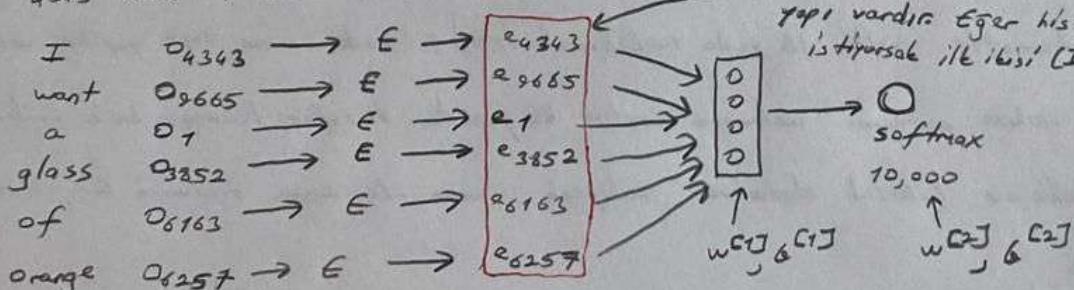


Calculus konusundaki gibi konerlik 1'e yakın olur,  $90^\circ$  sıfır yakını ettilerim ve  $180^\circ$  ise  $-1$  yönü tamamen farklı olduklarını ifade eder.

~~300x10,000~~ bir featurized matrisi ( $10,000, 1$ ) bir one-hot vector ile çarparsak ( $300, 1$ ) bir kelimeli tensör oluşturur.

### Learning Word Embeddings:

I want a glass of orange — .  
4343 9665 1 3852 6163 6257



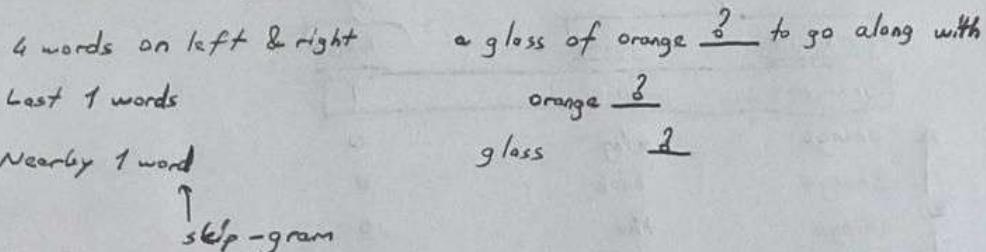
Bu kısımda her kelimeli 300 dimensional toplam 1000 dimensional yapı vardır. Eğer kelimelerin kategorik ismi varsa ilk 1000 (I want) sıralanır ve 1000 dimensiona 10,000'a göre eğitilir.

(62)

other context / target pairs

I want a glass of orange juice to go along with my cereal.

Context: Last 4 words



Word2Vec:

Skip-grams:

I want a glass of orange juice to go along with my cereal.

<u>Context</u>	<u>Target</u>
orange	juice
orange	glass
orange	my

Her bir context içindeki onunla ilişkili rastgele kelimelerin sayısı bir baglantı tarihlenmesi

Model:

Vocab size = 10,000

Context  $c$  ("orange")  $\rightarrow$  Target  $t$  ("juice")  
6257  $\rightarrow$  4834

$o_c \rightarrow e \rightarrow e_c \xrightarrow{\text{softmax}} \hat{o}_t \rightarrow \hat{y}$

$$e_c = E_{oc}$$

$$\text{Softmax} = p(+|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

$$L(\hat{y}, y) = - \sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

$\theta_t$  = parameter associated with output

$$y = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \leftarrow 4834$$

Vocabulary size ile de bağlantılı olacak şekilde bu işlemekin computational speed' fazladır.

Bu süreci azaatlılık için hierarchical softmax gibi yapısı kullanır ve vocabularyyi yarıya bölen sözler ile de hız artıratır.

### Negative Sampling:

Defining a new learning problem:

I want a glass of orange juice to go along with my cereal.

Context	X	word	Y	target
orange		juice		1
orange		king		0
orange		book		0
orange		the		0
orange		of		0

$\ell = 5-20$  smaller datasets

$\ell = 2-5$  larger datasets

Tek bir target positive example ile eğitilen negative örnekler kullanıma yaradıracak.

Bu işlemi olayı örneğin 10.000 kelimeye sahip vocabulary eğitmek yerine 1 tane positive ve 6 tane negatif örneği kullanmak etmektedir ve bu işlem çok daha az computational cost gerektirir, bu teknigue de negatif sampling denir.

\* Negative örnekleri sezerken esitlik, şartlanmalar vardır mesela middle kelimeler sezikteşlik ya da sıktılık faktörlerden kelimeler olabilir fakat the, of, and,... gibi genel işin wordları olayabilir. Heuristic bir algoritma olsun  $P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10000} f(w_j)^{3/4}}$  yapısı sık tenkit edilir.

\* Sentiment Classification probleminde basit bir yapı izleyip tüm kelimelerin temsilinin ortalamasını alıp nöral ağa vererek bu paragrafın payına çok fazla sabıktır ve herhangi bir sırra gözetmez. Bu problemi many-to-one RNN kullanarak çözülebilir.

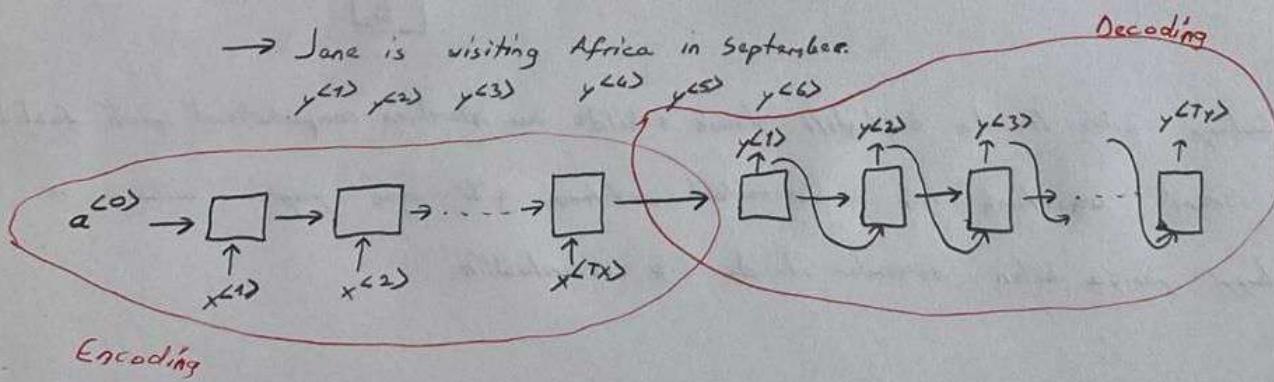
### Basic Models:

#### Sequence to sequence model:

$x^{(1)} \ x^{(2)} \ x^{(3)} \ x^{(4)} \ x^{(5)}$   
Jane visite l'Afrique en Septembre

→ Jane is visiting Africa in September.

$y^{(1)} \ y^{(2)} \ y^{(3)} \ y^{(4)} \ y^{(5)} \ y^{(6)}$



Bu yapıda önceki text encode edilir ve sonra text anlamı çıkarıldıktan sonra decoding ile gecmiş text elde edilir. Bu işlem fotoğraf hakkında yazı üretken modelde de kullanılır. Son layer olan softmax layeri sırasıyla ondan önceki layerda görseli temsil eden encoding hali elinizde kalar ve sonra bunu bir decoding yapısı ile text ürettilir.

### Picking the most likely sentence:

İlk etapta her çevireceğimiz kelime ( $x^{(i)}$ ) verdigimiz ve sonra ona bağlı translated word ( $y^{(i)}$ ) oluşturulan bir modelliniz varsa, bu model bir önceki kelimelere de bağlı olarak tahmin edilecektir. Son incelediğiniz machine translation modelinde ise  $P(y^{(1)}, \dots, y^{(T_y)} | x^{(1)}, \dots, x^{(T_x)})$  tüm bir cümleyi toplu olarak alıp sonra ona bağlı tahminler yapıyor luna "conditional language model" denilebilir. Verilen French sentence için muhtemel olan English translation türlerini döndürür ve bize bunlardan en muhtemel olanını bulacaktır.

### Finding the most likely translation:

Jane visite l'Afrique en septembre.

→ Jane is visiting Africa in September.

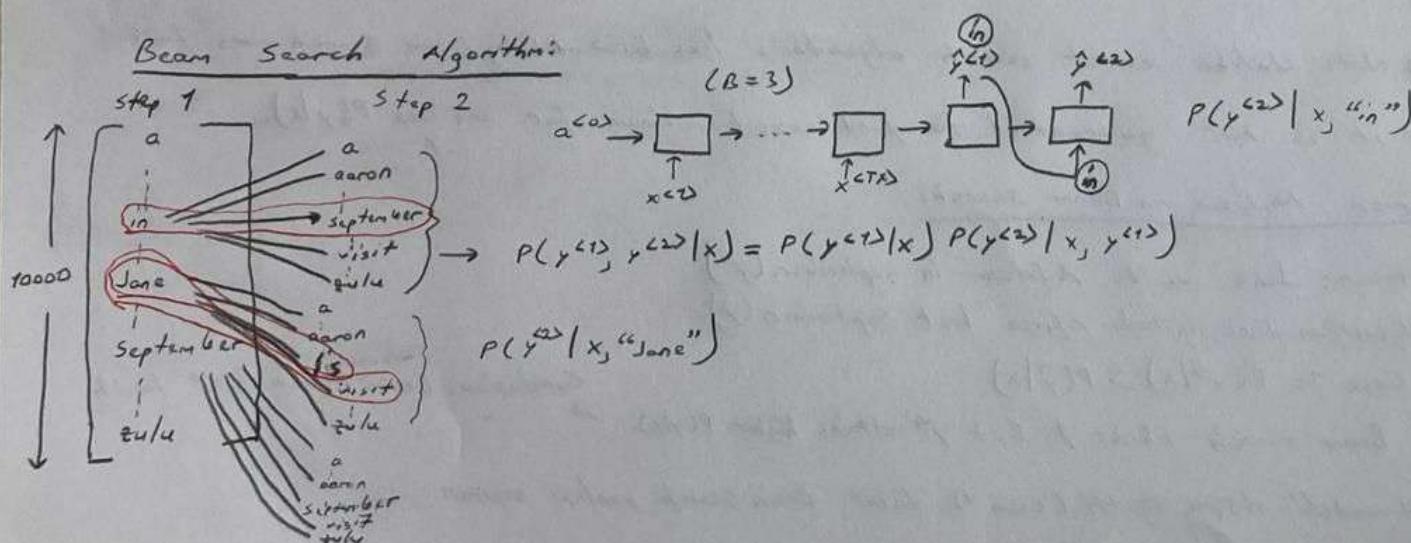
→ Jane is going to be visiting Africa in September.

→ In September, Jane will visit Africa.

→ Her African friend welcomed Jane in September.

$$\underset{y^{(1)}, \dots, y^{(T_y)}}{\arg\max} P(y^{(1)}, \dots, y^{(T_y)} | x)$$

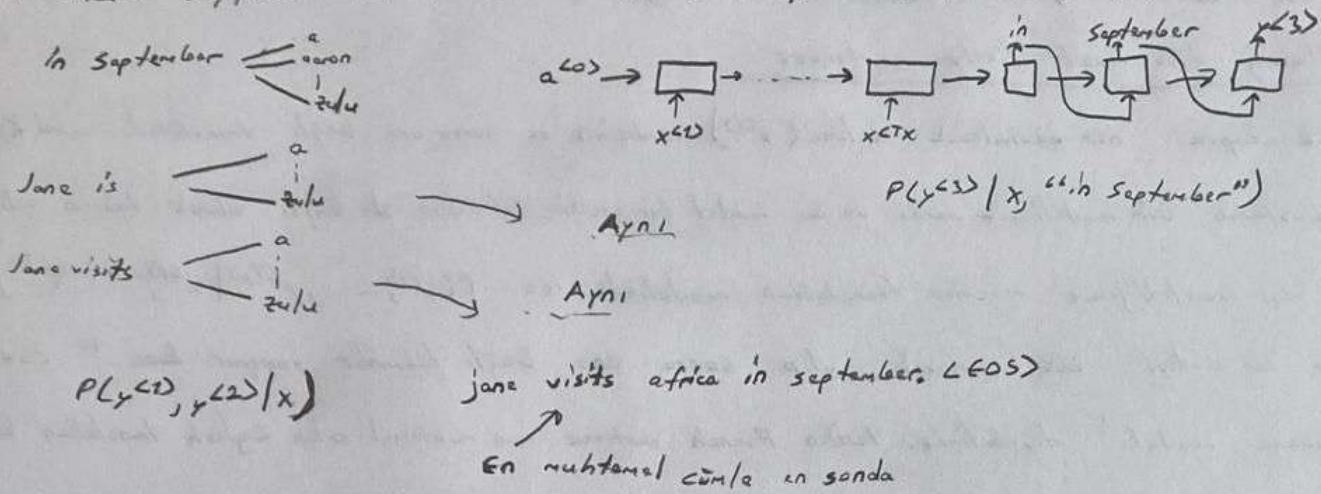
Fransızca cümle için en yüksek olasılığı veren İngilizce çevirisi için "Beam Search" kullanılabılır. Greedy Search gibi önce en iyi ihtimale sahip ilk kelimelerin sonraki kelimeleri ve bu şekilde giden yapıda tahminler yapılırsa bu çok iyi sonuc vermez çünkü zaman yapısını ya da cümledeki diğer detayları anlamasın. Bu yüzden en iyi ihtimale sahip en iyi cümle bulunmalıdır.



(65)

Beam Search en multimedial 3, 5'te da 10 yapıya bağlı kalımı seger. Sonra bu kalime ile en multimedial dğeri arar. Bu yapı 3 tane art arda gelan en iyi seçenek'i buffer ile 3 kalime içinde 10.000 vocabulary ile dener. Toplam 30.000 seçenek arasından en iyi 3'üne bulup onlara devam eder.

Bir önceki soyfada seenen en multimedial 3 iyiyi alarak tekrar işlem yapar.



### Length Normalization:

$$\begin{aligned} & \underset{y}{\operatorname{argmax}} \prod_{t=1}^{T_y} P(y^{ct} | x, y^{ct}, \dots, y^{ct-1}) \\ & \underset{y}{\operatorname{argmax}} \sum_{t=1}^{T_y} \log P(y^{ct} | x, y^{ct}, \dots, y^{ct-1}) \\ & \frac{1}{T_y} \sum_{t=1}^{T_y} \log P(y^{ct} | x, y^{ct}, \dots, y^{ct-1}) \end{aligned}$$

$\log P(y|x)$

Olasılık değerlerin 0-1 arasında olduğu için sıralı sırpırm yapmak çok kusur bir deger verir. Bu yüzden logaritmik yapısı kullanmak daha mantıklıdır. Değerler 1'den kısık olduğu için hep negatif değerler gelir ve bu durumdan dolayı da average alırsınız.

Beam width  $B$  değerini uygulamaya göre degritir  $\rightarrow$  large  $B$ : better result, slower  
 $\rightarrow$  small  $B$ : worse result, faster

~~Note:~~ Unlike exact search algorithms like BFS or DFS, Beam search runs faster but it is not guaranteed to find exact minimum for  $\underset{y}{\operatorname{argmax}} P(y|x)$ .

### Error Analysis on Beam Search:

Human: Jane visits Africa in September. ( $y^*$ )

Algorithm: Jane visited Africa last September. ( $\hat{y}$ )

Case 1:  $P(y^*|x) > P(\hat{y}|x)$

Beam search chose  $\hat{y}$ . But  $y^*$  attains higher  $P(y|x)$ .

Conclusion: Beam Search is at fault

RNN modeli doğru eğitti (Case 1) fakat Beam Search yanlışlığını seviyor.

Case 2:  $P(y^*(x)) \leq P(\hat{y}|x)$

$y^*$  is a better translation than  $\hat{y}$ . But RNN predicted  $P(y^*(x)) \leq P(\hat{y}|x)$

Conclusion: RNN model is at fault.

### Attention Model:

Modeller daha azur cümlelerde iyi performans sergileyemiyordular. Bleu score gibi teknikler de belli bir kelime sapından sonra toto performans sergiliyor. Attention modeller RNN yapısındaki her  $x$  için  $y$  tekniği esasında  $y^*$ 'i buluyor ve onu yerine her kelime tekniğini için bütün kelimelerin bağılı bir ağırlık ile katkı sağlıyor. Yani her kelime tekniğini yaparken sadece önceki kelimelerin etkisi yerine, önceki kelimelerin ve sıradan cümledeki kelimelerin toplam katkıyı alınarak tekniğin editiliyor.

$\alpha^{(t,t')}$  = amount of "attention"  $y^{(t)}$  should pay to  $a^{(t')}$

$$\alpha^{(t,t')} = (\rightarrow^{(t,t')}, \leftarrow^{(t,t')})$$

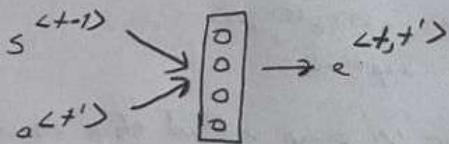
$$\text{önce} \rightarrow c^{(t)} = \sum_{t'} \alpha^{(t,t')} \cdot a^{(t')}$$

Buradaki önceliği kismi ağırlıkların belirlenmesi ve teknin kismidir.

### Computing attention $\alpha^{(t,t')}$

$\alpha^{(t,t')} = \text{amount of attention } y^{(t)} \text{ should pay to } a^{(t')}$

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



Bu modelin negatif yanı çok réaliteli olmasının Tx girdi ve Ty çıktıların attention parametresi Tx multiply Ty ( $Tx \cdot Ty$ ) kadar olur.

### Transformer Network:

Klasik RNN, GRU, LSTM yapıları yine paralel isteme kolaylığı sağlayan bir yapıdır.

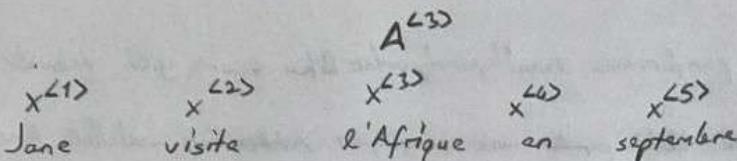
- Attention + CNN
- Self-Attention
- Multi-Head Attention

Yapıları sayısında çok daha etkili ve verimli bir yapı sunar.

Self-Attention:

$A(q, k, v)$  = attention-based vector representation of a word

Calculate for each word  $A^{(1)}, \dots, A^{(5)}$



Bu örnekte mesala 3. kelime l'Afrique döşenirsek bunu bir embedding ile ifade ederken fazla bunu yaparken şurra kelimelere ( $x^{(2)}, x^{(4)}, \dots$ ) bakarak Afrikası tarihi mi yoksa turistlik yer olmak mı, ada olmak mı dahası döşenilmesini anlamaya çalışıp ana göre tensilini oluşturacaktır.

Transformers Attention:

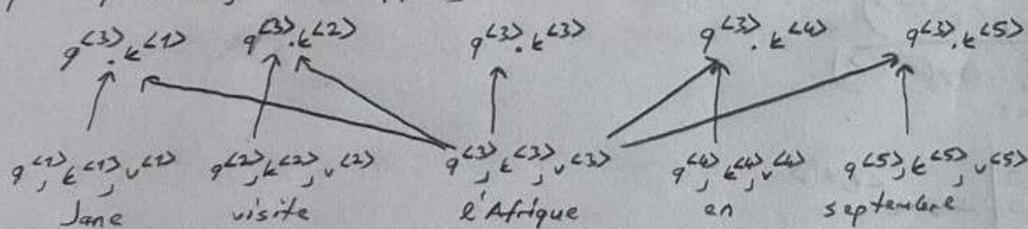
$$A(q, k, v) = \sum_i \frac{\exp(q \cdot k^{(i)})}{\sum_j \exp(q \cdot k^{(j)})} v^{(i)}$$

query      key      value

3. kelime l'Afrique için

$$\begin{aligned} q^{(3)} &= w^Q \cdot x^{(3)} \\ k^{(3)} &= w^K \cdot x^{(3)} \\ v^{(3)} &= w^V \cdot x^{(3)} \end{aligned}$$

$q = \text{question, what's happening there (Topic: destination, history...)}$



Bu işlemler 3. kelimenin sorusuna kelimelerin ne kadar iyi cevap verdığını ölçer. Mesela 1. kelime bir person, 2. action gibi döşenirsek 2. kelimenin durumu açıklayacağı düşünülebilir.

Yukarıdaki  $q \cdot k$  çarpımlarının softmax hesaplandıktan sonra her biri kendi  $v$  (value) değerini alır ve en son elde edilen tüm sonuçlar toplanır. Bu işlem fixed representation yapısı yerine daha aktif useful representation yapısının oluşmasına katkı sağlar.

Formülün vectorized version değerler (1-5) hesabın later son formül ise;

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)V$$

Diger bir ismi ise "scaled dot-product attention" dir.

Multi-Head Attention:

Self-attention yapısının büyük bir for-loop'u olarak düşünelibiliç.  
 $q, k, v$  yapısı, farklı weightler ile sorulur mesala 8 farklı weight değerinin ve günler ile  
 oluşturulan self attention yapısı her weight değerinin düşer yani farklı sorular soruyor gibi düşünelibiliç.

- what's happening?
- when ... ?
- who ... ?

Bu yapılar da farklı tekniklerle anlansal olarak öne çıkarır ayrıca hesaplamalar birbirinden bağımsız olduğu için paralel işlemde yapılabilir.

$$\text{Multihead}(Q, K, V) = \text{concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h) W_0$$

$$\text{head}_i = \text{Attention}(W_i^Q Q, W_i^K K, W_i^V V)$$

→ Transformer Details....