Evrişimli Sinir Ağları: Adım Adım

Kurs 4'ün ilk görevine hoş geldiniz! Bu ödevde, hem ileri yayılım hem de (isteğe bağlı olarak) geriye yayılım dahil olmak üzere, Numpy'de evrişimsel (CONV) ve havuzlama (POOL) katmanlarını uygulayacaksınız.

Gösterim:

- Üst simge [l), l' nin bir nesnesini belirtir bu katman.
  - Örnek: a [4) 4'tür bu katman aktivasyonu. 5 tarafı W dahil [5) ve b [5) bu katman parametreleri.
- Üst simge (i), i'den bir nesneyi belirtir (i) , i'dir bu örnek.
  - Örnek: x bu eğitim örneği girişi.
- Alt simge i , i [l) 'yi belirtir bu Bir vektörün girişi.
  - Örnek: ai , i tam bağlı (FC) katmanını belirtir . bu katman l'deki aktivasyonların girişi , bunun bir olduğunu varsayarak nH, nW ve nC, belirli bir [l) [l) [l)
- katmanının sırasıyla yüksekliğini, genişliğini ve kanal sayısını belirtir . Belirli bir l katmanına referans vermek istiyorsanız nH , nW , nC de yazabilirsiniz.
- nHp devri, Kuzeybatı pr ev ve nCp r ev sırasıyla [l 1) [l 1) [l 1) önceki katmanın yüksekliğini, genişliğini ve kanal sayısını belirtir . Belirli bir l katmanına atıfta bulunuluyorsa , bu aynı zamanda nH , nW , nC olarak da gösterilebilir.

Numpy'ye zaten aşina olduğunuzu ve/veya uzmanlığın önceki kurslarını tamamladığınızı varsayıyoruz. Başlayalım!

1 - Paketler

Öncelikle bu atama sırasında ihtiyaç duyacağınız tüm paketleri import edelim.

- dizi Python ile bilimsel hesaplama için temel pakettir. matplotlib Python'da grafikleri
- çizmek için bir kütüphanedir. np.random.seed(1)
- tüm rastgele işlev çağrılarının tutarlı olmasını sağlamak için kullanılır. Çalışmanızı notlandırmamıza yardımcı olacaktır.

```
numpy'yi np olarak içe
aktar h5py'yi içe
aktar matplotlib.pyplot'u plt olarak içe aktar

%matplotlib inline
plt.rcParams['figure.figsize'] = (5.0, 4.0) # grafiklerin varsayılan boyutunu ayarlayın

plt.rcParams['image.interpolation'] = 'en yakın' plt.rcParams['image.cmap']
= 'gri'

%load_ext otomatik yeniden yükleme
%otomatik yeniden yükleme 2
```

```
np.random.seed(1)
```

2 - Ödevin Ana Hatları

Evrişimli bir sinir ağının yapı taşlarını uygulayacaksınız! Uygulayacağınız her işlev, gerekli adımlarda size yol gösterecek ayrıntılı talimatlara sahip olacaktır:

- Evrişim fonksiyonları şunları içerir:
  - Sıfır Dolgu
  - Evrişim penceresi
  - İleriye doğru evrişim
  - Geriye doğru evrişim (isteğe bağlı)
- Aşağıdakileri içeren havuzlama işlevleri:
  - İleriye doğru havuzlama
  - Maske oluştur
  - Değeri dağıt
  - Geriye doğru havuzlama (isteğe bağlı)

Bu not defteri sizden bu işlevleri numpy'de sıfırdan uygulamanızı isteyecek. Bir sonraki not defterinde, aşağıdaki modeli oluşturmak için bu işlevlerin TensorFlow eşdeğerlerini kullanacaksınız:

Her ileri fonksiyon için karşılık gelen geriye doğru eşdeğerinin bulunduğunu unutmayın . Bu nedenle, ileri modülünüzün her adımında bazı parametreleri önbellekte saklayacaksınız. Bu parametreler
Geri yayılma sırasındaki gradyanları hesaplamak için kullanılır.

3 - Evrişimsel Sinir Ağları

Programlama çerçeveleri evrişimlerin kullanımını kolaylaştırsa da Derin Öğrenmede anlaşılması en zor kavramlardan biri olmaya devam etmektedir. Evrişim katmanı, aşağıda gösterildiği gibi bir giriş hacmini farklı boyuttaki bir çıkış hacmine dönüştürür.

Bu bölümde evrişim katmanının her adımını oluşturacaksınız. İlk önce iki yardımcıyı uygulayacaksınız fonksiyonlar: biri sıfır doldurma için, diğeri ise evrişim fonksiyonunun kendisini hesaplamak için.

3.1 - Sıfır Doldurma

Sıfır doldurma, görüntünün kenarlarına sıfırlar ekler:

Şekil 1 : 2 dolgulu Sıfır Dolgulu Görüntü (3 kanal, RGB).

Dolgunun başlıca faydaları şunlardır:

- Birimlerin yüksekliğini ve genişliğini mutlaka küçültmeden bir CONV katmanını kullanmanıza olanak tanır. Bu, daha derin ağlar oluşturmak için önemlidir, aksi takdirde daha derin katmanlara gidildikçe yükseklik/genişlik küçülür. Önemli bir özel durum, yüksekliğin/genişliğin bir katmandan sonra tam olarak korunduğu "aynı" evrişimdir.

- Daha fazla bilgiyi görüntünün kenarında tutmamıza yardımcı olur. Dolgu olmadan, Bir sonraki katmandaki çok az değer, görüntünün kenarları olarak piksellerden etkilenecektir.

Alıştırma: Bir grup örnek X'in tüm görüntülerini sıfırlarla dolduran aşağıdaki işlevi uygulayın. Np.pad'i kullanın. (5, 5 ,5 ,5 , 5) şeklindeki "a" dizisini 2. boyut için pad = 1, 4. boyut için pad = 3 ve geri kalanı için pad = 0 ile doldurmak isterseniz, şunu yapmanız gerektiğini unutmayın: Yapmak:

```
a = np.pad(a, ((0,0), (1,1), (0,0), (3,3), (0,0))), 'sabit', sabit_değerler = (.., ..))


# DERECELENDİRİLMİŞ FONKSİYON: sıfır_pad

def sıfır_pad(X, ped):

        Pad with the dataset X'in tüm görsellerini sıfırlar. Dolgu uygulanır
Şekil 1'de gösterildiği gibi bir görüntünün yüksekliğine ve
        genişliğine göre.

        Argüman:
        X -- a'yı temsil eden python numpy şekil dizisi (m, n_H, n_W, n_C)
m adet görsel
        pad -- tamsayı, dikeydeki her görüntünün etrafındaki dolgu miktarı
ve yatay boyutlar

        İadeler:
        X_pad -- şeklin dolgulu görüntüsü (m, n_H + 2*pad, n_W + 2*pad, n_C)


        ### KODU BURADAN BAŞLAYIN ### (     1 satır)
        X_pad = np.pad(X, ((0, 0), (pad, pad), (pad, pad), (0, 0))
 'sabit', sabit_değerler=0)
        ### SON KODU BURAYA ###

        X_pad'i döndür

np.random.seed(1)
x = np.random.randn(4, 3, 3, 2)
x_pad = sıfır_pad(x, 2)
yazdır ("x.shape =", x.shape)
yazdır ("x_pad.shape =", x_pad.shape)
yazdır ("x[1,1] =", x[1,1])
yazdır ("x_pad[1,1] =", x_pad[1,1])
```

```
fig, axarr = plt.subplots(1, 2)
axarr[0].set_title('x')
axarr[0].imshow(x[0,:,:,0])
axarr[1].set_title('x_pad) ')
axarr[1].imshow(x_pad[0,:,:,0])
```

```
x.shape = (4, 3, 3, 2) x_pad.shape
= (4, 7, 7, 2) x[1,1] = [[ 0,90085595
-0,68372786] [-0,12289023 -0,93576943]
   [-0,26788808 0,53035547] ]

x_pad[1,1] = [[ 0.0.] [ 0.0.] [ 0.0.]
   [ 0.0.] [ 0.0.]
   [ 0.0.] [ 0.0.] ]
```

```
<matplotlib.image.AxesImage at 0x7f6c048f31d0>
```

Beklenen çıktı:

3.2 - Tek adımlı evrişim

Bu bölümde, filtreyi girişin tek bir konumuna uygulayacağınız tek bir evrişim adımı uygulayın.
Bu, aşağıdaki özelliklere sahip bir evrişimli birim oluşturmak için kullanılacaktır:

- Bir giriş ses seviyesi alır
- Girişin her konumuna bir filtre uygular • Başka bir birimin çıktısını alır (genellikle farklı boyuttadır)

Şekil 2 : 2x2 filtre ve 1 adımla evrişim işlemi (adım = attığınız miktar)
her kaydırdığınızda pencereyi hareket ettirin)

Bir bilgisayarlı görme uygulamasında soldaki matristeki her değer tek bir piksele karşılık gelir değerini alırız ve 3x3'lük bir filtreyi, değerlerini orijinal matrisle öğe bazında çarparak, ardından toplayıp bir önyargı ekleyerek görüntüyle evriştiririz. Alıştırmanın bu ilk adımında, tek bir gerçek değerli çıktı elde etmek için konumlardan yalnızca birine filtre uygulamaya karşılık gelen tek bir evrişim adımını uygulayacaksınız.

Bu not defterinin ilerleyen kısımlarında, tam evrişimli işlemi uygulamak için bu işlevi girişin birden fazla konumuna uygulayacaksınız.

Alıştırma: conv_single_step() işlevini uygulayın. İpucu.

```
# DERECELENDİRİLMİŞ FONKSİYON: conv_single_step

def conv_single_step(a_slice_prev, W, b):

    Önceki katmanın çıktı aktivasyonunun tek bir dilimine (a_slice_prev) W parametreleriyle
tanımlanan bir filtre uygulayın.


    Argümanlar:
    a_slice_prev -- (f, f, n_C_prev) şeklindeki giriş verilerinin dilimi
    W -- Bir pencerede yer alan ağırlık parametreleri - şekil matrisi (f,
f, n_C_prev)
    b -- Bir pencerede yer alan sapma parametreleri - şekil matrisi (1, 1, 1)


    İadeler:
    Z - kayan pencerenin evrilmesinin sonucu olan skaler bir değer (W,
b) giriş verilerinin bir x diliminde


    ### KODU BURADAN BAŞLAYIN ### ( 2 satır kod)
    # a_slice ve W arasındaki eleman bazında çarpım. Sapmayı henüz eklemeyin.

    s = np.multiply(a_slice_prev, W)
    # Birimlerin tüm girişlerini toplayın.
    Z = np.toplam(lar)
    # Z'ye b sapmasını ekleyin. Z'nin skaler bir değerle sonuçlanması için b'yi bir float()'a dönüştürün.

    Z = Z + kayan nokta(b)
    ### SON KODU BURAYA ###

    Z'ye dönüş

np.random.seed(1)
a_slice_prev = np.random.randn(4, 4, 3)
W = np.random.randn(4, 4, 3)
b = np.random.randn(1, 1, 1)

Z = conv_single_step(a_slice_prev, W, b)
yazdır("Z =", Z)
```

```
Z = -6,99908945068
```

Beklenen Çıktı: Z -6,99908945068

3.3 - Evrişimli Sinir Ağları - İleri geçiş

İleri geçişte birçok filtre alıp bunları girişte birleştireceksiniz. Her 'evrişim' size bir 2D matris çıktısı verir.
Daha sonra 3 boyutlu bir hacim elde etmek için bu çıktıları istifleyeceksiniz:

Alıştırma: W filtrelerini bir A_prev giriş aktivasyonunda evriştirmek için aşağıdaki işlevi uygulayın.
Bu fonksiyon A_prev girdisi olarak önceki katmanın çıktısını (m girdi kümesi için), W ile gösterilen F filtrelerini/ağırlıklarını ve her filtrenin kendi (tek) sapmasına sahip olduğu b ile gösterilen bir sapma vektörünü alır. Son olarak adım ve dolguyu içeren hiperparametreler sözlüğüne de erişebilirsiniz.

İpucu:

1. "a_prev" matrisinin (şekil (5,5,3)) sol üst köşesinde 2x2'lik bir dilim seçmek için, yapardım:

```
a_slice_prev = a_prev[0:2,0:2,:]
```

Aşağıda a_slice_prev'i tanımlayacağınız başlangıç/bitiş indekslerini kullanarak tanımlayacağınızda bu işinize yarayacaktır.

1. Bir_slice'ı tanımlamak için öncelikle köşelerini tanımlamanız gerekir: vert_start, vert_end, horiz_start ve horiz_end. Bu şekil, aşağıdaki kodda her bir köşenin h, w, f ve s kullanılarak nasıl tanımlanabileceğini bulmanızda yardımcı olabilir.

Şekil 3 : Dikey ve yatay başlangıç/bitiş kullanılarak dilim tanımı (2x2 filtre ile) Bu şekil yalnızca tek bir kanalı göstermektedir.

Hatırlatma: Evrişimin çıktı şeklini giriş şekline bağlayan formüller şöyledir:

$$nH\ nH = \frac{_{pr\ ev}\ f + 2\times \text{tuş takımı}}{\text{s ri de}} + 1$$

$$nW = \frac{_{nWp\ rev}\ f + 2\times \text{p reklam}}{\text{uzun adım}} + 1$$

nC=evrişimde kullanılan filtrelerin sayısı

Bu alıştırmada vektörleştirme konusunda endişelenmeyeceğiz ve her şeyi for-döngülerle uygulayacağız.

```
# DERECELENDİRİLMİŞ FONKSİYON: conv_forward

def conv_forward(A_prev, W, b, hparametreleri):

        Bir evrişim fonksiyonu için ileri yayılımı uygular
```

```
    Argümanlar:
    A_prev - önceki katmanın çıktı aktivasyonları, numpy dizisi
şekil (m, n_H_prev, n_W_prev, n_C_prev)
    W -- Ağırlıklar, şeklin numpy dizisi (f, f, n_C_prev, n_C)
    b -- Önyargılar, numpy şekil dizisi (1, 1, 1, n_C)
    hparameters - "stride" ve "pad" kelimelerini içeren python sözlüğü

    İadeler:
    Z -- dönüşüm çıktısı, şeklin numpy dizisi (m, n_H, n_W, n_C)
    önbellek - conv_backward() işlevi için gereken değerlerin önbelleği


    ### KODU BURADAN BAŞLAYIN ###
    # A_prev'in şeklinden boyutları alın (    1 satır) (m, n_H_prev, n_W_prev, n_C_prev) =
     A_prev.shape

    # W'nin şeklinden boyutları al (    1 çizgisi)
     (f, f, n_C_prev, n_C) = W.shape

    # "hparameters"dan bilgi alın (    2 satır)
    adım = hparametreler['adım']
    ped = hparametreler['pad']

    # Aşağıdakileri kullanarak CONV çıkış hacminin boyutlarını hesaplayın:
yukarıda verilen formül. İpucu: yere int() kullanın. (    2 satır)
    n_H = int((n_H_prev - f + 2 * ped) / adım) + 1
    n_W = int((n_W_prev - f + 2 * ped) / adım) + 1

    # Çıkış hacmi Z'yi sıfırlarla başlatın. (    1 satır)
    Z = np.sıfır((m, n_H, n_W, n_C))

    # A_prev'i doldurarak A_prev_pad'i oluşturun
    A_prev_pad = sıfır_pad(A_prev, ped)

    for i in range(m): toplu                                    #üzerinde döngü
eğitim örnekleri
            a_prev_pad = A_prev_pad[i]                                          #
Eğitim örneğinin yastıklı aktivasyonunu seçin
            h in range(n_H) için : çıkış                        #döngü
hacminin dikey ekseni
                w in range(n_W) için : çıkış                    #döngü
hacminin yatay ekseni
                    aralıktaki c için (n_C): çıkış              #döngü
hacminin kanalları (= #filtreler)

                        # Geçerli "dilim"in köşelerini bulun (    4
çizgiler)
                        vert_start = h * adım
                        vert_end = vert_başlangıç + f
```

```
                                    horiz_start = w * adım horiz_end =
                                    yatay_başlangıç + f

                                    # (3B) dilimini tanımlamak için köşeleri kullanın
a_prev_pad (Hücrenin üzerindeki İpucuna bakın). (    1 satır)
                                    a_slice_prev = a_prev_pad[vert_start:vert_end,
horiz_start:horiz_end, :]

                                    # (3D) dilimini doğru filtreyle evriştirin
Bir çıkış nöronunu geri almak için W ve önyargı b. (    1 satır)
                                    Z[i, h, w, c] = conv_single_step(a_slice_prev,
W[...,c], b[...,c])

        ### SON KODU BURAYA ###

        # Çıktı şeklinizin doğru olduğundan emin olun iddia(Z.shape == (m,
        n_H, n_W, n_C))

        # Backprop önbelleği için bilgileri "önbelleğe" kaydedin = (A_prev, W, b,
        hparameters)

        Z'yi döndür , önbellek

np.random.seed(1)
A_prev = np.random.randn(10,4,4,3)
W = np.random.randn(2,2,3,8) b =
np.random.randn(1,1,1,8) hparametreler =
{"pad" : 2, "stride": 2}


Z, önbellek_conv = conv_forward(A_prev, W, b, hparametreler) print("Z'nin ortalaması =",
np.mean(Z)) print("Z[3,2,1] =", Z[3,2,1 ])
print("cache_conv[0][1][2][3] =", önbellek_conv[0]
[1][2][3])

Z'nin ortalaması = 0,0489952035289 Z[3,2,1]
= [-0,61490741 -6,7439236 -2,55153897 1,75698377 3,56208902 0,53036437

    5.18531798 8.75898442]
önbellek_conv[0][1][2][3] = [-0,20075807 0,18656139 0,41005165]
```

Beklenen çıktı:

Son olarak, CONV katmanının da bir aktivasyon içermesi gerekir; bu durumda aşağıdaki kod satırını ekleyeceğiz:

```
# Bir çıkış nöronunu geri almak için pencereyi çevirin
Z[i, h, w, c] =                    ...
# Etkinleştirmeyi uygula
A[i, h, w, c] = aktivasyon(Z[i, h, w, c])
```

Bunu burada yapmanıza gerek yok.

4 - Havuzlama katmanı

Havuzlama (HAVUZ) katmanı, girişin yüksekliğini ve genişliğini azaltır. Hesaplamayı azaltmaya yardımcı olmanın yanı sıra, özellik algılayıcıların girişteki konumuna göre daha değişmez olmasına yardımcı olur. İki tür havuzlama katmanı şunlardır:

- Maksimum havuzlama katmanı: bir (f ,f) penceresini girişin üzerine kaydırır ve maksimum değeri saklar çıktıdaki pencerenin.
- Ortalama havuzlama katmanı: bir (f ,f) penceresini girişin üzerine kaydırır ve ortalamayı saklar çıktıdaki pencerenin değeri.

Bu havuzlama katmanlarının eğitilecek geri yayılım için hiçbir parametresi yoktur. Ancak f pencere boyutu gibi hiper parametrelere sahiptirler . Bu, maksimum veya ortalamayı hesaplayacağınız fxf penceresinin yüksekliğini ve genişliğini belirtir.

4.1 - İleri Havuzlama

Şimdi MAX-POOL ve AVG-POOL'u aynı fonksiyonda uygulayacaksınız.

Alıştırma: Havuzlama katmanının ileri geçişini uygulayın. Aşağıdaki yorumlardaki ipuçlarını takip edin.

Hatırlatma: Dolgu olmadığından, havuzlamanın çıktı şeklini giriş şekline bağlayan formüller şöyledir:

$$nH\ nH = \frac{f}{s\ ri\ de}_{pr\ ev} + 1$$

$$nW = \frac{nWp\ rev\quad f}{uzun\ adım} + 1$$

$$nC = nC_{pr\ ev}$$

```
# DERECELENDİRİLMİŞ FONKSİYON: havuz_ileri

defpool_forward (A_prev, hparameters, mode = "max"):

        Havuzlama katmanının ileri geçişini uygular

        Argümanlar:
        A_prev -- Giriş verileri, şeklin numpy dizisi (m, n_H_prev, n_W_prev,
n_C_prev)
        hparameters -- "f" ve "stride" kelimelerini içeren python sözlüğü
        mod -- kullanmak istediğiniz havuzlama modu, bir dize ("maks" veya "ortalama") olarak tanımlanır


        İadeler:
```

```
        A -- havuz katmanının çıktısı, şekilsiz bir dizi (m, n_H, n_W, n_C)

        önbellek - havuzlama katmanının geri geçişinde kullanılan önbellek, girişi ve hparametrelerini içerir
        """


        # Giriş şeklinden boyutları al
        (m, n_H_prev, n_W_prev, n_C_prev) = A_prev.shape

        # "hparameters"dan hiperparametreleri al
        f = hparametreler["f"]
        adım = hparametreler["adım"]

        # Çıktının boyutlarını tanımlayın
        n_H = int(1 + (n_H_prev - f) / adım)
        n_W = int(1 + (n_W_prev - f) / adım)
        n_C = n_C_prev

        # Çıkış matrisi A'yı başlat
        A = np.sıfır((m, n_H, n_W, n_C))

        ### KODU BURADAN BAŞLAYIN ###
        for i in range(m): eğitim                                   #üzerinde döngü
örnekleri
                    h in range(n_H) için : çıkış                    # döngü açık
hacminin dikey ekseni
                            w in range(n_W) için : çıkış            # döngü açık
hacminin yatay ekseni
                                    aralıktaki c için ( n_C ): çıkış    #üzerinde döngü
hacminin kanalları

                                        # Geçerli "dilim"in köşelerini bulun (    4
çizgiler)
                                        vert_start = h * adım
                                        vert_end = vert_başlangıç + f
                                        horiz_start = w * adım
                                        yatay_son = yatay_başlangıç + f

                                        # A_prev'in i'inci eğitim örneği, kanal c'de geçerli dilimi tanımlamak için köşeleri
kullanın. (    1 satır)
                                        a_prev_slice = A_prev[i, vert_start:vert_end,
horiz_start:horiz_end, c]

                                        # Dilimdeki havuzlama işlemini hesaplayın. Kullanmak
Modları ayırt etmek için bir if ifadesi. np.max/np.mean'ı kullanın.
                                        eğer mod == "maks":
                                                A[i, h, w, c] = np.max(a_prev_slice)
                                        elif modu == "ortalama":
                                                A[i, h, w, c] = np.mean(a_prev_slice)
```

```
        ### SON KODU BURAYA ###

        # Pool_backward() önbellek = (A_prev, hparameters) için girdiyi ve hparametrelerini "cache"de saklayın


        # Çıktı şeklinizin doğru olduğundan emin olun iddia(A.shape == (m,
        n_H, n_W, n_C))

        A'yı döndür , önbellek

np.random.seed(1)
A_prev = np.random.randn(2, 4, 4, 3) hparameters = {"adım" :
2, "f": 3}

A, önbellek =pool_forward(A_prev, hparameters) print("mode = max")
print("A =", A) print()


A, önbellek =pool_forward(A_prev, hparameters, mode = "ortalama") print("mode = ortalama") print("A
=", A)


mod = maksimum
A = [[[[ 1,74481176 0,86540763 1,13376944]]]


  [[[ 1.13162939 1.51981682 2.18557541]]]]

mod = ortalama A =
[[[[ 0,02105773 -0,20328806 -0,40389855]]]


  [[[-0,22154621 0,51716526 0,48155844]]]]
```

Beklenen çıktı:

```
<tr>
<td>
A = </
td> <td>

        [[[[ 1.74481176 0.86540763 1.13376944]]]

[[[ 1.13162939 1.51981682 2.18557541]]]]

        </td> </
tr> <tr>
<td>
```

```
bir =
</td>
        <td>
          [[[[ 0.02105773 -0.20328806 -0.40389855]]]

[[[-0,22154621 0,51716526 0,48155844]]]]
```

```
        </td>
</tr>
```

Tebrikler! Artık evrişimli bir ağın tüm katmanlarının ileri geçişlerini uyguladınız.

Bu not defterinin geri kalanı isteğe bağlıdır ve notlandırılmayacaktır.

5 - Evrişimli sinir ağlarında geri yayılım
(İSTEĞE BAĞLI / NOTLANMAMIŞ)

Modern derin öğrenme çerçevelerinde yalnızca ileri geçişi uygulamanız gerekir ve çerçeve geri geçişi halleder, böylece çoğu derin öğrenme mühendisinin geri geçişin ayrıntılarıyla uğraşmasına gerek kalmaz. Evrişimsel ağlar için geriye doğru geçiş karmaşıktır. Ancak dilerseniz not defterinin bu isteğe bağlı kısmı üzerinde çalışabilirsiniz.
Evrişimli bir ağdaki backprop'un neye benzediğine dair bir fikir edinin.

Daha önceki bir kursta basit (tamamen bağlantılı) bir sinir ağı uyguladığınızda, parametreleri güncelleme maliyetine göre türevleri hesaplamak için geri yayılımı kullandınız.
Benzer şekilde evrişimli sinir ağlarında parametreleri güncellemek için maliyete göre türevleri hesaplayabilirsiniz. Backprop denklemleri önemsiz değildir ve bunları derste türetmedik ancak aşağıda kısaca sunduk.

5.1 - Evrişimsel katmanın geriye doğru geçişi

Bir CONV katmanı için geri geçişi uygulayarak başlayalım.

5.1.1 - dA'nın Hesaplanması:

Bu, belirli bir filtre W eğitimi örneğinin maliyetine göre d A'yı hesaplama formülüdür : c ve verilen

$$d\,A+\text{¿}_{sa=0}^{nH}\ {}_{w=0}^{\text{Kuzeybatı}}\ W\ c\times d\ Zhw$$

W'den c bir filtredir ve d Zhw, maliyetin gradyanına karşılık gelen bir skalerdir.
h'inci satır ve sütundaki Z dönüşüm katmanının çıktısına (soldaki i'inci adım ve aşağı doğru j'inci adımda alınan nokta çarpımına karşılık gelir) . Her seferinde dA'yı güncellerken aynı Wc filtresini farklı bir dZ ile çarptığımızı unutmayın. Bunu yapmamızın temel nedeni ileri yayılımı hesaplarken her filtrenin farklı bir a_slice tarafından noktalanması ve toplanmasıdır.

Bu nedenle dA için backprop'u hesaplarken, sadece tüm a_slice'ların gradyanlarını ekliyoruz.

Kodda, uygun for-döngüleri içinde bu formül şu anlama gelir:

```
da_prev_pad[vert_start:vert_end, horiz_start:horiz_end, :] +=
W[:,:,:,c] * dZ[i, h, w, c]
```

5.1.2 - dW'nin Hesaplanması:

Bu, kayıp açısından d Wc'yi (d Wc bir filtrenin türevidir) hesaplama formülüdür :

$$d\,W_c + \sum_{sa=0}^{nH} \sum_{w=0} \text{asl ic e} \times d\,Z_{hw}$$

Burada bir dilim Zi j aktivasyonunu oluşturmak için kullanılan dilime karşılık gelir . Dolayısıyla bu bize W'nin o dilime göre gradyanını verir. Aynı W olduğundan , d W'yi elde etmek için tüm bu gradyanları toplayacağız .

Kodda, uygun for-döngüleri içinde bu formül şu anlama gelir:

```
dW[:,:,:,c] += a_slice * dZ[i, h, w, c]
```

5.1.3 - Hesaplama veri tabanı:

Bu, belirli bir W filtresinin maliyetine göre db'yi hesaplama formülüdür . $_c$:

$$db \mp \pi \; w\, d\, Z_{hw}$$

Daha önce temel sinir ağlarında gördüğünüz gibi db, d Z'nin toplanmasıyla hesaplanır. Bu durumda, maliyete göre yalnızca dönüşüm çıktısının (Z) tüm gradyanlarını topluyorsunuz.

Kodda, uygun for-döngüleri içinde bu formül şu anlama gelir:

```
db[:,:,:,c] += dZ[i, h, w, c]
```

Alıştırma: Aşağıdaki conv_backward işlevini uygulayın. Tüm eğitim örneklerini, filtreleri, yükseklikleri ve genişlikleri özetlemelisiniz. Daha sonra yukarıdaki 1, 2 ve 3 formüllerini kullanarak türevleri hesaplamanız gerekir.

```
def conv_backward(dZ, önbellek):

        Bir evrişim fonksiyonu için geriye yayılımı uygulama

        Bağımsız
        Değişkenler: dZ -- dönüşüm katmanının çıktısına (Z) göre maliyetin eğimi, numpy şekil dizisi (m,
n_H, n_W, n_C) önbellek -- conv_backward() için gereken değerlerin
        önbelleği, çıktısı
```

```
conv_forward()

    Döndürür:
    dA_prev -- dönüşüm katmanının girişine göre maliyetin gradyanı (A_prev), şeklin numpy dizisi (m, n_H_prev,
n_W_prev, n_C_prev)

    dW - dönüşüm katmanının ağırlıklarına göre maliyetin gradyanı (W)

            numpy şekil dizisi (f, f, n_C_prev, n_C) db -- dönüşüm
    katmanının önyargılarına göre maliyetin gradyanı (b)

    """         numpy şekil dizisi (1, 1, 1, n_C)


    ### KODU BURADAN BAŞLAYIN ###
    # "Önbellek"ten bilgi alın
     (A_prev, W, b, hparametreleri) = önbellek

    # A_prev'in şeklinden boyutları alın (m, n_H_prev,
     n_W_prev, n_C_prev) = A_prev.shape

    # W'nin şeklinden boyutları alın (f, f, n_C_prev,
     n_C) = W.shape

    # "hparameters"dan bilgi alır stride =
    hparameters["stride"] pad =
    hparameters["pad"]

    # dZ'nin şeklinden boyutları alın (m, n_H, n_W,
     n_C) = dZ.shape

    # dA_prev, dW, db'yi doğru şekillerle başlatın dA_prev = np.zeros((m,
    n_H_prev, n_W_prev, n_C_prev))

    dW = np.zeros((f, f, n_C_prev, n_C)) db =
    np.zeros((1, 1, 1, n_C))

    # Pad A_prev ve dA_prev
    A_prev_pad = sıfır_pad(A_prev, ped)
    dA_prev_pad = sıfır_pad(dA_prev, ped)

    (m) aralığındaki i için :                                    #eğitim üzerinde döngü
örnekler

            # A_prev_pad ve dA_prev_pad'den eğitim örneğini seçin a_prev_pad = A_prev_pad[i]
            da_prev_pad = dA_prev_pad[i]


            aralıktaki h için (n_H):                                  # dikey döngü
```

```
çıkış hacminin ekseni
                    w in range(n_W) için : çıkış                    # yatay döngü
hacminin ekseni
                        aralıktaki c için (n_C): çıkış              #üzerinde döngü
hacminin kanalları

                            # Geçerli "dilim"in köşelerini bulun
                            vert_start = h
                            vert_end = vert_başlangıç + f
                            horiz_start = w
                            yatay_son = yatay_başlangıç + f

                            # Dilimi tanımlamak için köşeleri kullanın
a_prev_pad
                            a_slice = a_prev_pad[vert_start:vert_end,
horiz_start:horiz_end, :]

                            # Pencerenin ve filtrenin degradelerini güncelleyin
yukarıda verilen kod formüllerini kullanarak parametreler
                            da_prev_pad[vert_start:vert_end,
horiz_start:horiz_end, :] += W[:,:,:,c] * dZ[i, h, w, c]
                            dW[:,:,:,c] += a_slice * dZ[i, h, w, c]
                            db[:,:,:,c] += dZ[i, h, w, c]

            # i'inci eğitim örneğinin dA_prev'ini dolgusuz da_prev_pad'e ayarlayın (İpucu:
X[pad:-pad, pad:-pad, :] kullanın)
            dA_prev[i, :, :, :] = da_prev_pad[pad:-pad, pad:-pad, :]
    ### SON KODU BURAYA ###

    # Çıktı şeklinizin doğru olduğundan emin olmak
    iddia(dA_prev.shape == (m, n_H_prev, n_W_prev, n_C_prev))

    dA_prev, dW, db'yi döndür

np.random.seed(1)
dA, dW, db = conv_backward(Z, önbellek_conv)
print("dA_mean =", np.mean(dA))
print("dW_mean =", np.mean(dW))
print("db_mean =", np.mean(db))
```

```
dA_ortalama = 0,634770447265
dW_mean = 1,55726574285
db_mean = 7,83923256462
```

** Beklenen Çıkış: dA_mean 1,45243777754 dW_mean 1,72699145831 db_mean** 7,83923256462

5.2 Havuzlama katmanı - geri geçiş

Daha sonra, MAX-POOL katmanından başlayarak havuzlama katmanı için geri geçişi uygulayalım. Havuzlama katmanının güncellenecek backprop için hiçbir parametresi olmasa da, havuzlama katmanından önce gelen katmanların degradelerini hesaplamak için yine de degradeyi havuzlama katmanı boyunca geri yaymanız gerekir.

5.2.1 Maksimum havuzlama - geri geçiş

Havuzlama katmanının geri yayılımına geçmeden önce, aşağıdakileri yapan create_mask_from_window() adında bir yardımcı işlev oluşturacaksınız:

$$X = \begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix} \quad M = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

Gördüğünüz gibi bu fonksiyon, matrisin maksimumunun nerede olduğunu takip eden bir "maske" matrisi oluşturur. Doğru (1), X'teki maksimumun konumunu gösterir, diğer girişler Yanlıştır (0). Daha sonra ortalama havuzlama için geri geçişin buna benzer olacağını ancak farklı bir maske kullanacağını göreceksiniz.

Alıştırma: create_mask_from_window() işlevini uygulayın. Bu işlev geriye doğru havuzlama için yararlı olacaktır. İpuçları:

- np.max() yardımcı olabilir. Bir dizinin maksimumunu hesaplar.
- Eğer bir X matrisiniz ve bir skaler x'iniz varsa: A = (X == x), X ile aynı boyutta bir A matrisi döndürecektir, öyle ki:

```
A[i,j] = X[i,j] = x ise doğrudur
A[i,j] = Yanlış eğer X[i,j] != x
```

- Burada bir matriste birden fazla maksimumun olduğu durumları dikkate almanıza gerek yoktur.

```
def create_mask_from_window(x):
    """
    X'in maksimum girişini tanımlamak için x giriş matrisinden bir maske oluşturur.

    Argümanlar:
    x -- Şekil dizisi (f, f)

    İadeler:
    maske -- Pencere ile aynı şekle sahip dizi, aynı zamanda bir True içerir
x'in maksimum girişine karşılık gelen konum.
    """
    ### KODU BURADAN BAŞLAYIN ### (1 satır)
    maske = x == np.max(x)
    ### SON KODU BURAYA ###

    dönüş maskesi
```

```
np.random.seed(1)
x = np.random.randn(2,3)
maske = create_mask_from_window(x)
yazdır('x = ', x)
print("maske = ", maske)
```

```
x = [[ 1,62434536 -0,61175641 -0,52817175]
  [-1,07296862 0,86540763 -2,3015387 ]]
maske = [[ Doğru Yanlış Yanlış]
  [Yanlış Yanlış Yanlış]]
```

Beklenen çıktı:

Neden maksimumun konumunu takip ediyoruz? Çünkü sonuçta çıktıyı ve dolayısıyla maliyeti etkileyen girdi değeri budur. Backprop, gradyanları maliyete göre hesaplıyor, dolayısıyla nihai maliyeti etkileyen herhangi bir şeyin sıfır olmayan bir gradyanı olmalıdır. Böylece backprop, eğimi maliyeti etkileyen bu özel girdi değerine geri "yayacaktır".

5.2.2 - Ortalama havuzlama - geri pas

Maksimum havuzlamada, her giriş penceresi için çıkış üzerindeki tüm "etki" tek bir girişten geldi değer-maks. Ortalama havuzlamada, girdi penceresinin her elemanı çıktı üzerinde eşit etkiye sahiptir. Yani backprop'u uygulamak için artık bunu yansıtan bir yardımcı fonksiyon uygulayacaksınız.

Örneğin, 2x2 filtre kullanarak ileri geçişte ortalama havuzlama yaparsak, geri geçiş için kullanacağınız maske şöyle görünecektir:

d Z=1 d Z=[1⁄4¿1⁄4)

Bu, dZ matrisindeki her konumun çıktıya eşit katkıda bulunduğu anlamına gelir çünkü ileri geçişte bir ortalama aldık.

Alıştırma: Bir dz değerini boyut şekli matrisi boyunca eşit olarak dağıtmak için aşağıdaki işlevi uygulayın.
İpucu

```
def dağıtım_değeri(dz, şekil):
    """
    Giriş değerini boyut şekli matrisinde dağıtır

    Argümanlar:
    dz -- giriş skaler
    şekil -- bizim için kullandığımız çıktı matrisinin şekli (n_H, n_W)
dz değerini dağıtmak istiyorum

    İadeler:
    a -- dz değerini dağıttığımız boyut dizisi (n_H, n_W)
```

```
        """

        ### KODU BURADAN BAŞLAYIN ###
        # Şekilden boyutları al (    1 çizgi)
        (n_H, n_W) = şekil

        # Matrise dağıtılacak değeri hesaplayın (    1 satırı)
        ortalama = dz / (n_H * n_W)

        # Her girişin "ortalama" değer olduğu bir matris oluşturun (    1 satır)

        a = np.ones(şekil) * ortalama
        ### SON KODU BURAYA ###

        bir dönüş

a = dağıtım_değeri(2, (2,2))
print('dağıtılmış değer =', a)
```

```
dağıtılan değer = [[ 0,5 0,5]
  [ 0,5 0,5]]
```

Beklenen çıktı:

5.2.3 Bir araya getirme: Geriye doğru havuzlama

Artık bir havuz katmanında geriye doğru yayılımı hesaplamak için ihtiyacınız olan her şeye sahipsiniz.

Alıştırma: Pool_backward işlevini her iki modda da ("maks" ve "ortalama") uygulayın.
Bir kez daha 4 for-döngüsü kullanacaksınız (eğitim örnekleri, yükseklik, genişlik ve kanallar üzerinde yinelemeler). Modun 'max' ya da 'average'a eşit olup olmadığını görmek için if/elif ifadesini kullanmalısınız. Eğer 'ortalama'ya eşitse, a_slice ile aynı şekle sahip bir matris oluşturmak için yukarıda uyguladığınız distribut_value() fonksiyonunu kullanmalısınız. Aksi takdirde, mod 'max'a eşittir ve create_mask_from_window() ile bir maske oluşturup bunu karşılık gelen dZ değeriyle çarpacaksınız.

```
defpool_backward (dA, önbellek, mod = "max"):

        Havuzlama katmanının geriye doğru geçişini uygular

        Argümanlar:
        dA - havuzlamanın çıktısına göre maliyet gradyanı
katman, A ile aynı şekilde
        önbellek - havuzlama katmanının ileri geçişinden önbellek çıkışı, katmanın girişini ve hparametrelerini içerir

        mod -- kullanmak istediğiniz havuzlama modu, bir dize ("maks" veya "ortalama") olarak tanımlanır


        İadeler:
```

```
        dA_prev -- havuzlama katmanının girişine göre maliyet gradyanı, A_prev ile aynı şekil
        """

        ### KODU BURADAN BAŞLAYIN ###

        # Önbellekten bilgi al (    1 satır)
        (A_prev, hparametreler) = önbellek

        # "hparameters"dan hiperparametreleri al (    2 satır)
        adım = hparametreler["adım"]
        f = hparametreler["f"]

        # A_prev'in şeklinden ve dA'nın şeklinden boyutları alın (    2 çizgi)

        m, n_H_prev, n_W_prev, n_C_prev = A_prev.shape
        m, n_H, n_W, n_C = dA.şekli

        # dA_prev'i sıfırlarla başlat (    1 satır)
        dA_prev = np.zeros(A_prev.shape)

        (m) aralığındaki i için : örnekler                                  #eğitim üzerinde döngü

                # A_prev'den (    1 satırı) eğitim örneğini seçin
                a_prev = A_prev[i]
                aralıktaki h için (n_H):                                    # dikeyde döngü
eksen
                        aralıktaki w için (n_W):                            # döngü açık
yatay eksen
                                aralıktaki c için (n_C): kanallar           #üzerinde döngü
(derinlik)
                                        # Geçerli "dilim"in köşelerini bulun (    4
çizgiler)
                                        vert_start = h
                                        vert_end = vert_başlangıç + f
                                        horiz_start = w
                                        yatay_son = yatay_başlangıç + f

                                        # Her iki modda da geriye doğru yayılımı hesaplayın.
                                        eğer mod == "maks":
                                                # Tanımlamak için köşeleri ve "c"yi kullanın
a_prev'den geçerli dilim (    1 satır)
                                                a_prev_slice = a_prev[vert_start:vert_end, horiz_start:horiz_end, c]

                                                # a_prev_slice'dan (    1 satır) maskeyi oluşturun
                                                maske = create_mask_from_window(a_prev_slice)
                                                # dA_prev'i dA_prev + olarak ayarlayın (maske doğru dA girişiyle
çarpılır) (    1 satır)
                                                dA_prev[i, vert_start:vert_end,
```

```
horiz_start:horiz_end, c] += np.multiply(mask, dA[i, h, w, c])

                                    elif modu == "ortalama":
                                        # dA'dan a değerini al (    1 satır)
                                        da = dA[i, h, w, c]
                                        # Filtrenin şeklini fxf (    1) olarak tanımlayın
astar)
                                        şekil = (f, f)
                                        # Doğru dA_prev dilimini elde etmek için dağıtın. yani da'nın
dağıtılmış değerini ekleyin. (    1 satır)
                                        dA_prev[i, vert_start:vert_end,
yatay_başlangıç:yatay_son, c] += dağıtma_değeri(da, şekil)

        ### BİTİŞ KODU ###

        # Çıktı şeklinizin doğru olduğundan emin olmak
        iddia(dA_prev.shape == A_prev.shape)

        dA_prev'i döndür

np.random.seed(1)
A_prev = np.random.randn(5, 5, 3, 2)
hparametreler = {"adım" : 1, "f": 2}
A, önbellek = havuz_forward(A_prev, hparametreler)
dA = np.random.randn(5, 4, 2, 2)

dA_prev =pool_backward(dA, önbellek, mod = "maks")
print("mod = maksimum")
print(' dA'nın ortalaması = ', np.mean(dA))
print('dA_prev[1,1] = ', dA_prev[1,1]) print()

dA_prev =pool_backward(dA, önbellek, mod = "ortalama")
print("mod = ortalama")
print(' dA'nın ortalaması = ', np.mean(dA))
print('dA_prev[1,1] = ', dA_prev[1,1])
```

```
mod = maksimum
dA'nın ortalaması = 0,145713902729
dA_prev[1,1] = [[ 0. [ 5,05844394                    0.                  ]
  -1,68282702]
  [ 0. ]]                    0.

mod = ortalama
dA'nın ortalaması = 0,145713902729
dA_prev[1,1] = [[ 0,08485462 0,2787552 ]
  [ 1,26461098 -0,25749373]
  [ 1,17975636 -0,53624893]]
```

Beklenen çıktı:

mod = maksimum:

dA'nın ortalaması =

mod = ortalama

dA'nın ortalaması =

Tebrikler !

Bu görevi tamamladığınız için tebrikler. Artık evrişimli sinir ağlarının nasıl çalıştığını anlıyorsunuz. Bir sinir ağının tüm yapı taşlarını uyguladınız. Bir sonraki ödevde TensorFlow'u kullanarak bir ConvNet uygulayacaksınız.