COMP755-Lect21

November 13, 2018

1 COMP 755

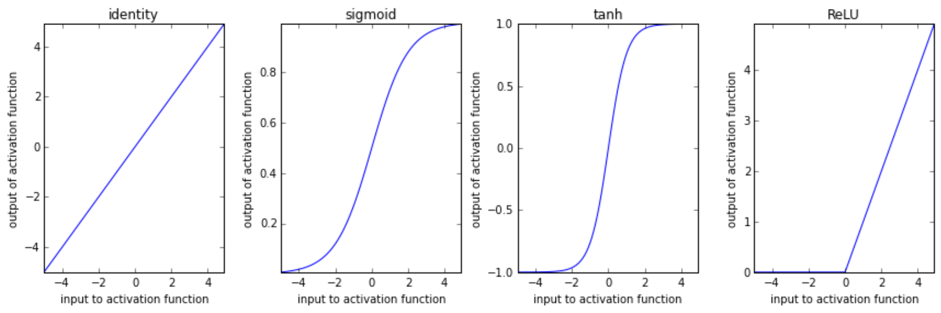
Plan for today

1. More on filters -- sizes, strides, pooling
2. Viewing neural network computation as composition
3. Computing the simplest of gradients for neural networks

2 Activation functions

Typical activation functions are

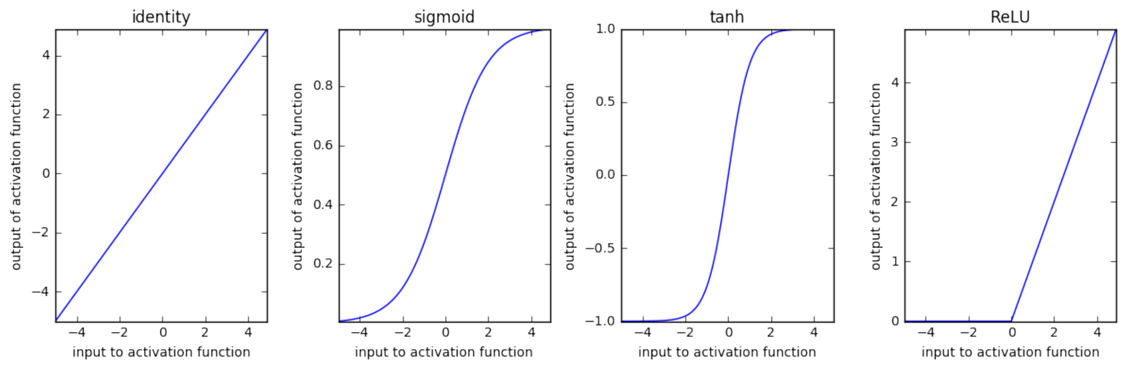
1. Identity $f(x) = x$
2. Sigmoid $f(x) = \frac{1}{1+e^{-x}}$
3. Hyperbolic tangent $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
4. Rectified linear unit (ReLU) $f(x) = \max(x, 0) = (x)_+ = [x > 0]x$



```
In [19]: import numpy as np
         import matplotlib.pyplot as plt
         from __future__ import print_function
         %matplotlib inline
         plt.figure(figsize=(12,4))
         x = np.arange(-5.0,5.0,0.1)
         activations = [('identity',lambda x:x),
                        ('sigmoid' ,lambda x:1./(1. + np.exp(-x))),
                        ('tanh'    ,lambda x:(np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))),
                        ('ReLU'    ,lambda x: (x>0)*x)]
```

```
for (i,act) in enumerate(activations):
    name = act[0]
    f = act[1]
    plt.subplot(1,len(activations),i+1)
    plt.title(name)
    plt.plot(x,f(x))
    plt.xlim([np.min(x),np.max(x)])
    plt.ylim([np.min(f(x)),np.max(f(x))])
    plt.xlabel('input to activation function')
    plt.ylabel('output of activation function')

plt.tight_layout()
```



3 Deep networks are composed of layers

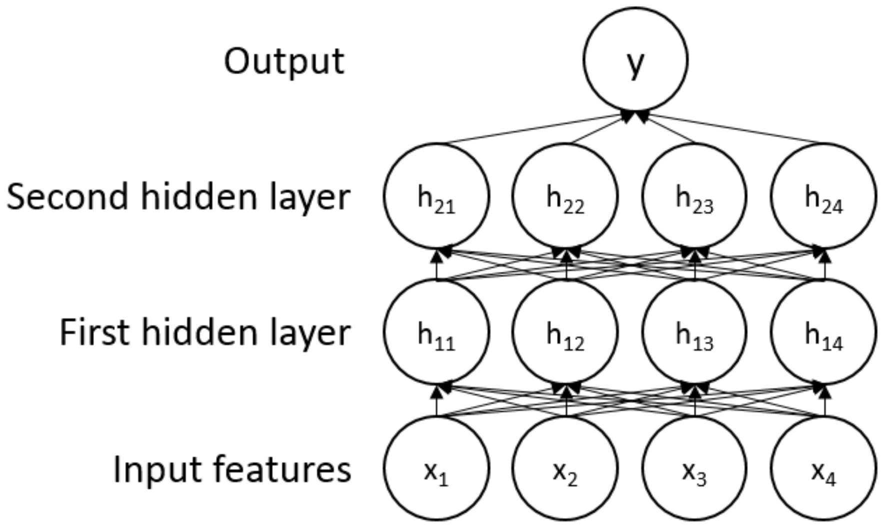
Typically the deep networks are built up out of layers.

4 Convolution illustration

5 Convolutional output size

Suppose we have a matrix of size 5×5 and we have a filter of size 2×2. What is the response matrix size?

```
In [22]: im = misc.imread('barbara.png').astype('float32')
         response1 = np.zeros(im.shape)
         response2 = np.zeros(im.shape)
         for i in range(im.shape[0]-2):
             for j in range(im.shape[1]-2):
                 patch = im[i:i+3,j:j+3]
                 patchvec = patch.reshape(9,1)
                 response1[i,j] = np.dot(filter1vec.T,patchvec)
                 response2[i,j] = np.dot(filter2vec.T,patchvec)
```



X=

1	2	0
1	0	2
0	2	1

w=

1	-1
2	-1

Filter

H=

1	0
-1	1

Response

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

H(0,0) = 1*1 + 2*(-1) + 1*2 + 0*(-1) = 1

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

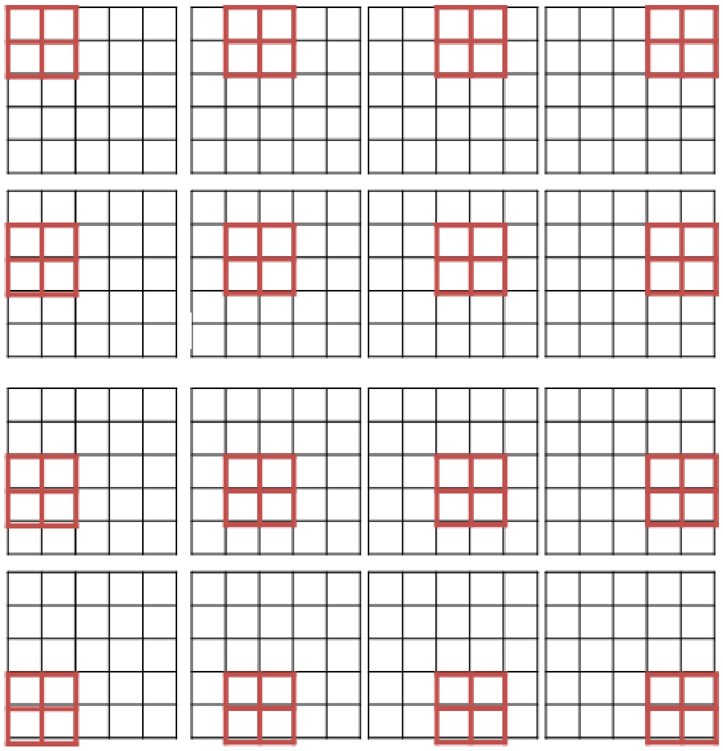
H(0,1) = 2*1 + 0*(-1) + 0*2 + 2*(-1) = 0

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

H(1,0) = 1*1 + 0*(-1) + 0*2 + 2*(-1) = -1

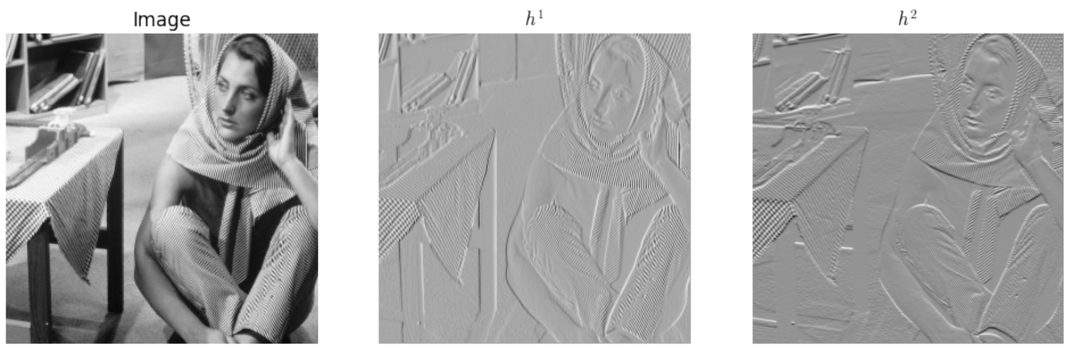
1	2	0
1	0*1	2*-1
0	2*2	1*-1

H(1,1) = 0*1 + 2*(-1) + 2*2 + 1*(-1) = 1



```
plt.figure(figsize=(12,4))
plt.subplot(1,3,1)
plt.imshow(im,cmap='Greys_r')
plt.title('Image')
plt.axis('off')
plt.subplot(1,3,2)
plt.imshow(response1,cmap='Greys_r')
plt.axis('off')
plt.title('$h^1$')
plt.subplot(1,3,3)
plt.imshow(response2,cmap='Greys_r')
plt.axis('off')
plt.title('$h^2$')
```

```
Out[22]: <matplotlib.text.Text at 0xb33d278>
```



6 Filters and Convolution

A little bit more formally, discrete (1D) convolution is computed by

$$\text{conv}(\mathbf{y},\mathbf{x})_i = \sum_j \mathbf{y}_{i-j}\mathbf{x}_j$$

Discrete 2D convolution is computed by

$$\text{conv}(\mathbf{Y},\mathbf{X})_{i,j} = \sum_k \sum_l \mathbf{y}_{i-k,j-l}\mathbf{x}_{j,k}$$

Note that the formal definition is a little bit different (indices are decreasing in $\mathbf{y}$).

7 Convolution can be made fast

Suppose both $\mathbf{y}$ and $\mathbf{x}$ are n long

$$\text{conv}(\mathbf{y},\mathbf{x})_i = \sum_j \mathbf{y}_{i-j}\mathbf{x}_j.$$

Naive computation can take $O(N^2)$, since both i and j range from 1 to N.

This can be sped up to $O(N\log(N))$ using discrete fourier transform -- details of this are beyond our course, but the idea is to regress both $\mathbf{y}$ and $\mathbf{x}$ onto an orthonormal basis in which convolution becomes elementwise addition.

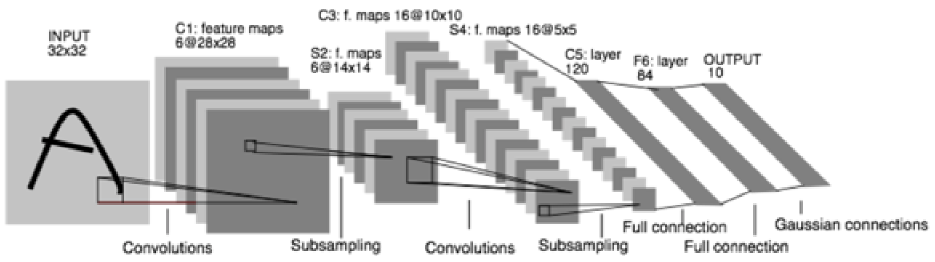
```
scipy.signal.convolve2d(y,np.flipud(np.fliplr(x)),mode='valid')
```

8 Convolutional layers in neural nets

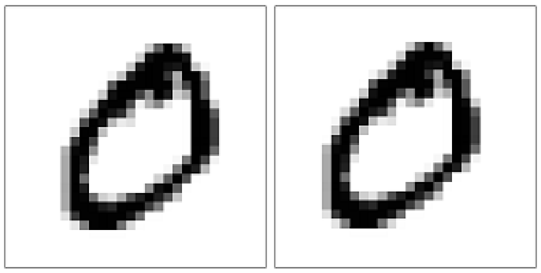
Convolving an image with multiple filters gives responses for each.

This is typically the first step in deep learning nets.

This example is from 1989! In the illustration 6 filters are applied in the first layer of the network.



9 Pooling responses

Here are two images of the same digit but shifted 

10 Other types of layers: Pooling

In order to make the network's output invariant to small shifts, we can agregate nearby values.

11 Other types of layers: Subsampling

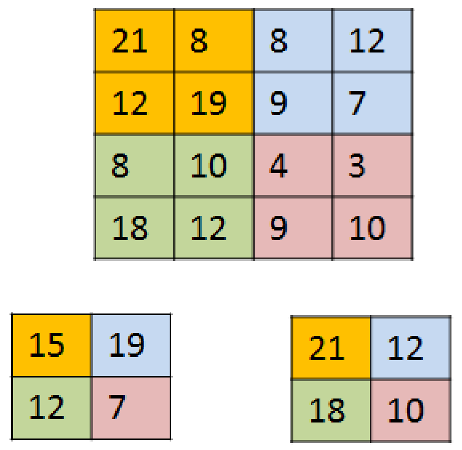
12 The concept of stride

You can compute output of a filter for all offsets, or a subset. Stride determines how much you skip.

13 Formal definition of filter responses in deep nets

Given filter size $d \times d$ and stride s responses $h_{i,j}$ for different filter types are: * Convolutional $h_{i,j} = \sum_{k=0}^{d} \sum_{l=0}^{d-1} x_{is+k,js+l} w_{k,l}$ * Max-pool $h_{i,j} = \max_{k,l \in \{0,d-1\}} x_{is+k,js+l}$ * Average-pool $h_{i,j} = \sum_{k=0}^{d} \sum_{l=0}^{d-1} \frac{1}{d^2} x_{is+k,js+l}$ * Subsample $h_{i,j} = x_{is,js}$

Note that only convolutional filter depends has parameters, rest are fixed.



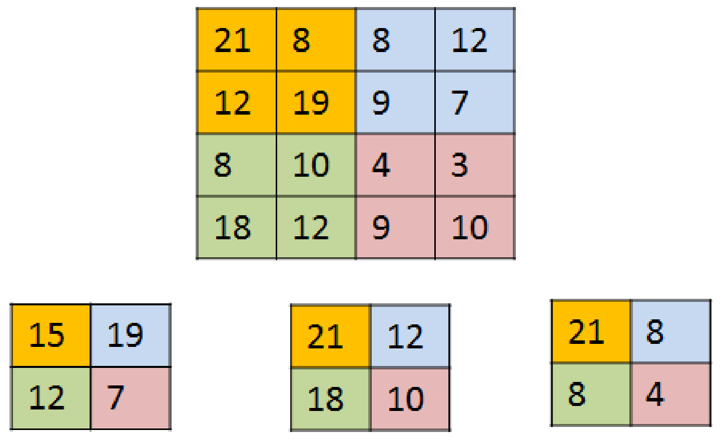
21	8	8	12
12	19	9	7
8	10	4	3
18	12	9	10

15	19
12	7

Output of
Average pooling
2 x 2 filter

21	12
18	10

Output of
Max-pooling
2 x 2 filter



21	8	8	12
12	19	9	7
8	10	4	3
18	12	9	10

15	19
12	7

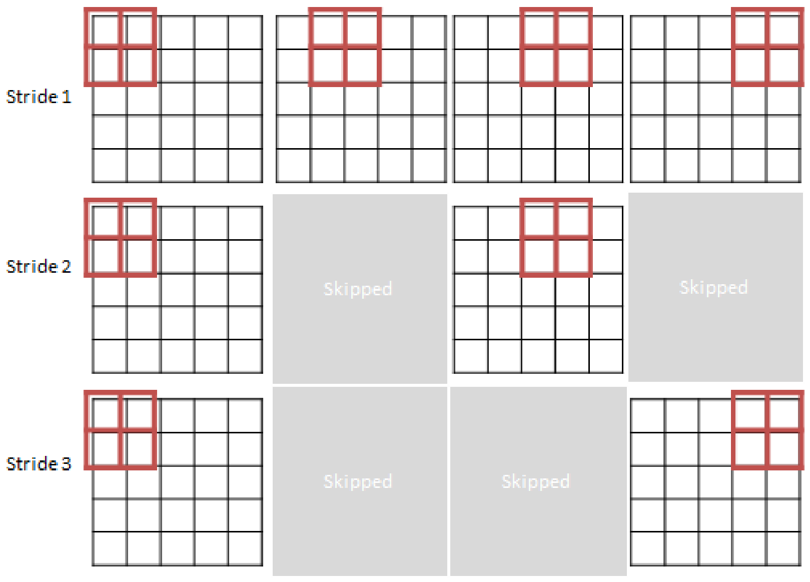
Output of
Average pooling
2 x 2 filter

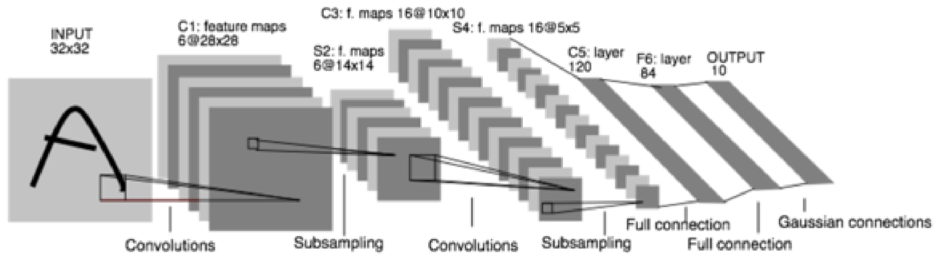
21	12
18	10

Output of
Max-pooling
2 x 2 filter

21	8
8	4

Output of
Subsampling
2 x 2 filter





14 How do we train a neural net?

Let's take a bird's eye view of the neural nets.

Given an input $\mathbf{x} \in \mathbb{R}^d$, a neural net takes outputs of a layer and feeds them to the next layer.

Let f be the function that computes all outputs of the first layer ($f : \mathbb{R}^d \to \mathbb{R}^{n_1}$)

$$\mathbf{h}^1 = f(\mathbf{x})$$

Let g be the functiont ath computes outputs of the second layer using inputs from the first layer ($g : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$)

$$\mathbf{h}^2 = g(\mathbf{h}^1)$$

Hence, second layer's output is given by composition of the two functions

$$\mathbf{h}^2 = g(f(\mathbf{x})).$$

15 Neural net forward propagation is function composition

Let f be an activation function and

$$\mathbf{f}(\mathbf{z})_i = f(z_i)$$

In words, $\mathbf{f}$ just applies activation function to each entry in vector $\mathbf{z}$ and give back a vector of outputs.

This notation will just help us consider the whole layer at once.

Propagation through a single layer l

$$\mathbf{h}^l = \mathbf{f}(\mathbf{b}^l + \mathbf{W}^l \mathbf{h}^{l-1})$$

where $\mathbf{W}^l$ is matrix of weights, one row per unit, and $\mathbf{b}$ is a bias matrix.

Hence a two layer network's output is given by

$$\mathbf{f}(\mathbf{b}^2 + \mathbf{W}^2 \mathbf{f}(\mathbf{b}^1 + \mathbf{W}^1 \mathbf{x}))$$

and more generally:

$$\mathbf{f}(\mathbf{b}^l + \mathbf{W}^l \mathbf{f}(\mathbf{b}^{l-1} \mathbf{W}^{l-1} \mathbf{f}(...\mathbf{f}(\mathbf{b}^1 + \mathbf{W}^1 \mathbf{x}))...))$$

16 Intermission: deep linear neural networks are trivial

Let activation function be identity $f(z) = z$ and $\mathbf{f}(\mathbf{z}) = \mathbf{z}$.

Output of the first layer is

$$\mathbf{f}(\mathbf{b}^1 + \mathbf{W}^1 \mathbf{x}) = \mathbf{b}^1 + \mathbf{W}^1 \mathbf{x}$$

Output of the second layer is

$$\begin{aligned} \mathbf{f}(\mathbf{b}^2 + \mathbf{W}^2 \mathbf{f}(\mathbf{b}^1 + \mathbf{W}^1 \mathbf{x})) &= \mathbf{b}^2 + \mathbf{W}^2(\mathbf{b}^1 + \mathbf{W}^1 \mathbf{x}) & (1) \\ &= \underbrace{\mathbf{b}^2 + \mathbf{W}^2 \mathbf{b}^1}_{\mathbf{b}} + \underbrace{\mathbf{W}^2 \mathbf{W}^1}_{\mathbf{W}} \mathbf{x} & (2) \end{aligned}$$

is equivalent to output of a single layer network with bias $\mathbf{b}$ and weights $\mathbf{W}$.

Hence two layer linear network "collapses" into a single layer linear neural network.

17 Intermission: deep linear neural networks are trivial

Conclusion: Networks using **solely** identity activations are trivial.

However, some of the recent results in deep learning argue for a sum of linear and nonlinear activations

$$f(z) = z + g(z)$$

giving rise to **residual networks** -- residual because they model residual, $z - f(z)$

18 Multivariate calculus refresher -- total derivative

Total derivative

$$f(\mathbf{z}(t)) = \sum_i \frac{\partial f}{\partial z_i}\frac{\partial z_i}{\partial t}$$

Let's say

$$f(x,y,z) = x^2 + y^2 + z^2$$

and let's say

$$g(t) = f(t,t,1)$$

Q: Using formula for total derivative

$$\frac{\partial g(t)}{\partial t} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial t}$$

compute $\frac{\partial g(t)}{\partial t}$.

19 Objectives for fitting the neural networks

Typical applications of neural networks are in supervised learning (data $\{y^t, \mathbf{x}^t : t = 1, ..., N\}$)

The output of the network

$$\hat{y}(\mathbf{x}) = f(\mathbf{b}^l + \mathbf{W}^l \mathbf{f}(\mathbf{b}^{l-1}\mathbf{W}^{l-1}\mathbf{f}(...\mathbf{f}(\mathbf{b}^1 + \mathbf{W}^1\mathbf{x}))...))$$

In order to asses how the network is performing we need an objective.

$$E = \sum_t \text{loss}(\hat{y}(\mathbf{x}^t), y^t)$$

Typical losses * square loss $(\hat{y}(\mathbf{x}^t) - y^t)^2$ * cross-entropy loss $-y^t \log(\hat{y}(\mathbf{x}^t)) + (1 - y^t)(1 - \hat{y}(\mathbf{x}^t))$

We note that objectives, E, depend on $\hat{y}(\mathbf{x}^t)$, which in turn depends on parameters $\mathbf{b}$ and $\mathbf{W}$.

20 Toy network

For simplicity we will first look at a toy network with two layers

$$\hat{y}(\mathbf{x}) = f(\mathbf{b}^2 + \mathbf{W}^2\mathbf{f}(\mathbf{b}^1 + \mathbf{W}^1\mathbf{x}))$$

and some objective E

Our goal is to solve the optimization problem

$$\underset{\mathbf{b}^1,\mathbf{b}^2,\mathbf{W}^2,\mathbf{W}^2}{\text{minimize}} \sum_t \text{loss}(\hat{y}(\mathbf{x}^t), y^t)$$

and we will use gradient descent for this. Hence, we need $\frac{\partial E}{\mathbf{b}^l}$ and $\frac{\partial E}{\mathbf{W}^l}$

This seems tricky.

21 Looking at the objective and forward propagation

In our simple toy network the objective is a sum of compositions of loss, second layer function, and first layer function.

$$E = \sum_t \text{loss}(\hat{y}(\mathbf{x}^t), y^t)$$

where

$$\hat{y}(\mathbf{x}) = f(\mathbf{b}^2 + \mathbf{W}^2 \mathbf{f}(\mathbf{b}^1 + \mathbf{W}^1 \mathbf{x}))$$

Let's say we wanted to compute derivative with respect to a second layer weight $\frac{\partial E}{\partial w_{ij}^2}$.

We can use chain rule:

$$\frac{\partial E}{\partial w_{ij}^2} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{ij}^2}$$

but this is still a bit tricky -- that second partial derivative is complicated.

22 Looking at the objective and forward propagation

Given

$$\hat{y}(\mathbf{x}) = f(\mathbf{b}^2 + \mathbf{W}^2 \mathbf{f}(\mathbf{b}^1 + \mathbf{W}^1 \mathbf{x}))$$

we got to

$$\frac{\partial E}{\partial w_{ij}^2} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{ij}^2}$$

However that second derivative still looks tricky. Recall that even though this is composition we can take advantage of the intermediate computation in layers.

$$\hat{y}(\mathbf{x}) = f(\mathbf{z}^2) \tag{3}$$
$$\mathbf{z}^2 = \mathbf{b}^2 + \mathbf{W}^2 \mathbf{h}^2 \tag{4}$$
$$\mathbf{h}^2 = \mathbf{f}(\mathbf{z}^1) \tag{5}$$
$$\mathbf{z}^1 = \mathbf{b}^1 + \mathbf{W}^1 \mathbf{x} \tag{6}$$

and

$$\frac{\partial E}{\partial w_{ij}^2} = \frac{\partial E}{\partial \hat{y}} \sum_k \frac{\partial \hat{y}}{\partial z_k^2} \frac{\partial z_k^2}{\partial w_{ij}^2} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^2} h_j^2$$

Q: Why does the sum

$$\sum_k \frac{\partial \hat{y}}{\partial z_k^2} \frac{\partial z_k^2}{\partial w_{ij}^2}$$

disappear and only term $\frac{\partial \hat{y}}{\partial z_i^2} h_j^2$ is left?

23 Looking at the objective and forward propagation

$$\frac{\partial E}{\partial w_{ij}^2} = \frac{\partial E}{\partial \hat{y}} \sum_k \frac{\partial \hat{y}}{\partial z_k^2} \frac{\partial z_k^2}{\partial w_{ij}^2} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^2} h_j^2$$

In order to compute this derivative we need * derivative of loss $\frac{\partial E}{\partial \hat{y}}$ * for sum of squares $2(\hat{y} - y)$ * for cross entropy $-y\frac{1}{\hat{y}} - (1-y)\frac{1}{1-\hat{y}}$ * derivative of activation $\frac{\partial \hat{y}}{\partial z_i^2} = \frac{\partial f(\mathbf{z}^2)}{\partial z_i^2}$ * for ReLU $[z_i^2 > 0]$ * for sigmoid $f(z_i^2)f(1 - z_i^2)$ * h_j^2

24 Recap

We computed the simplest of the derivatives. It depended on 1. contribution from the loss 2. contribution from the activation function 3. state of the input nodes

Next time we will generalize this for the first layer and the deeper networks.