Lezione 4 11/10/2024

Nell'ultima lezione abbiamo visto loss functions e output functions. Abbiamo visto che l'ouput function è l'input della loss, e deve essere buono per misurare l'errore e per il gradient.

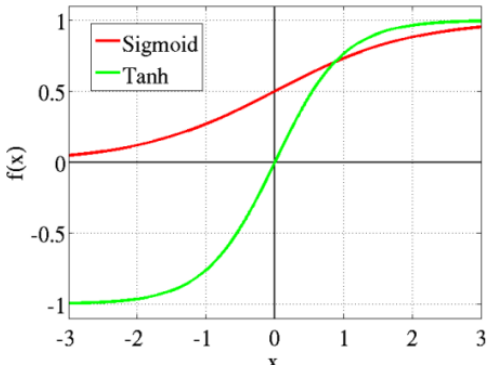
Hidden units

La maggior parte delle unità di output possono essere usate negli hidden layer.

- Most hidden units consist of an affine transformation $z = Wx + b$ of the input x, to which an element-wise non linear function $g(z)$ is applied

La differenza negli hidden units consiste nelle diverse funzioni di attivazione g(x).

La funzione sigmoide può essere usata come activation delle hidden unit, oltre che output unit. C'è anche la funzione iperbolica tangente Tanh che è simile ma migliore.



The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

As the sigmoid:

The function is **differentiable**.

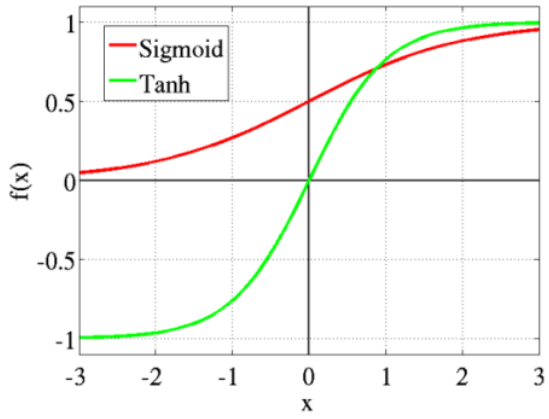
The function is **monotonic** but its **derivative is not monotonic**.

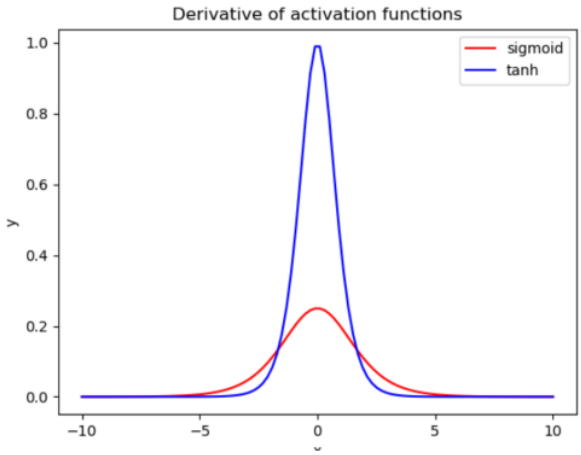
The tanh function is mainly used classification between two classes.

L'input zero sarà mappato vicino allo 0 ...

Questa funzione è principalmente usata per problemi di classificazione.

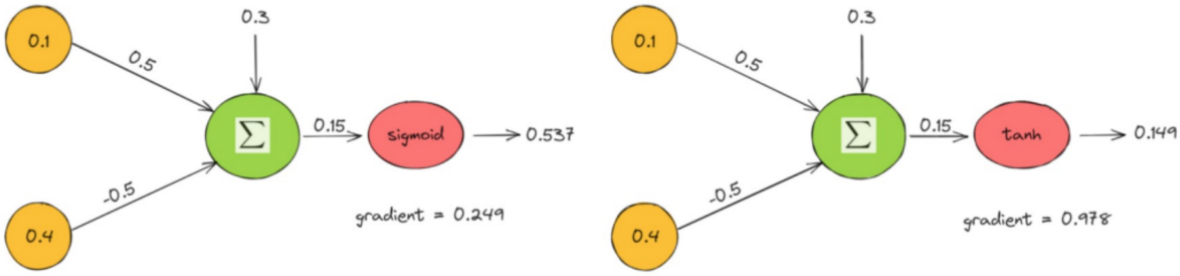
A destra vediamo le derivate delle due funzioni sigmoide e tanh.





Se abbiamo una alta confidenza nell'output sono uguali. Ma in quei casi dove ci sono divisioni vicine allo 0, con tanh abbiamo un gradiente più alto che velocizza il processo di apprendimento.

- This example compares sigmoid (left) with tanh (right) activation functions for a simple network with input equal to 0.1 and 0.4 and weights 0.5 and -0.5.



0.3 è il bias. Vediamo che l'output per il tanh è più vicino a 0 e il gradiente è 4 volte più alto, quindi è più sensibile, migliora l'apprendimento più velocemente.

Ci sono molte altre funzioni di attivazione possibili che non vediamo. è impossibile predire quale sarà quella migliore da usare. Ci sono delle prove sperimentali che dicono che nei layer nascosti i **rectified linear units** hanno performance migliori (è la scelta standard da cui iniziare).

g(z)=max[0,z] is not differrentiable in 0 !

Right derivative is 0, left derivative is 1

Due to numerical approximation (a 0 is unlikely to truly be 0), software implementation usually return one of the two values rather than undefined.

La funzione non è lineare. Abbiamo il problema che nello 0 la funzione non è differenziabile.

errore nella slide sopra: penso che 0 e 1 sono opposti.

La probabilità di ottenere 0 è sostanzialmente 0 se stiamo lavorando nello spazio continuo, quindi di solito possiamo dire al software che se accade, torna un numero di default.

Rectified Linear Units (RLU) are usually the default choice:

$$g(z) = max[0, z]$$

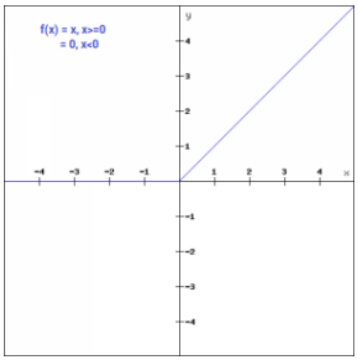
The derivatives are large whenever the unit is active

- Typically used on top on linear transformation
- nondifferentiability can be overcame
- gradient based algorithm still working
- Allow for sparse representation, accelerate learning, simplify the model

Dato che tutti i valori negativi, il gradiente è 0, queste unità permettono una rappresentazione sparsa, cioè molti neuroni diventeranno inattivi, il che è molto utile per semplificare il modello.

L'errore può essere backpropagato

- ReLU (Rectified linear unit):
  - it is non linear
  - We can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
  - Main advantage over other activation functions:
    - it does not activate all the neurons at the same time.



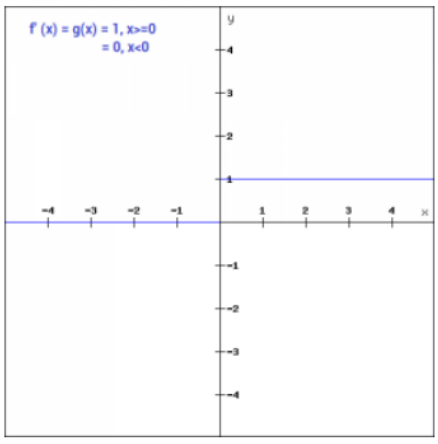
LEARNING - ENZA MESSINA

If the input is negative it will convert it to zero and the neuron does not get activated.

- This means that at a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

Il vantaggio è che non attiva tutti i neuroni contemporaneamente perchè alcuni sono 0.

- If we look at the gradient it is ...very nice
  - The gradient is zero for x<0, which means the weights are not updated during the back-propagation

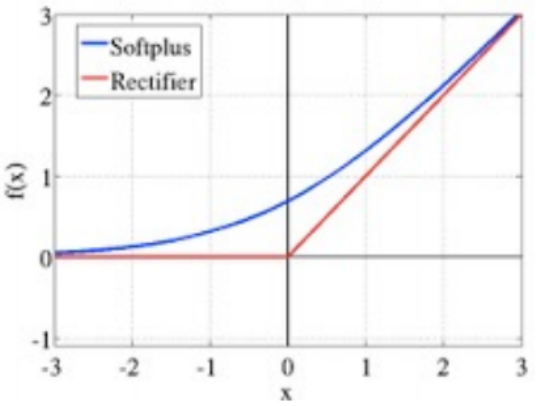


Quindi quando il gradiente è meno di 0, i pesi non sono aggiornati.

Ci sono alcune generalizzazioni della funzione RLU.

Generalization of RLU

A smooth approximation of the ReLU is "softplus" function



$$f(x) = \ln(1 + e^x)$$

One might expect it to have advantage over the rectifier due to being differentiable everywhere or due to saturating less completely, but empirically it does not.

Per esempio questa è un'approssimazione smooth intorno allo 0, è chiamata "**softplus**".

Funziona in alcuni casi specifici, quindi non è una buona scelta da cui iniziare.

Generalization of RLU $h_i = g(z,\alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$

Absolute value rectification: fixes $\alpha_i = -1$ to obtain $g(z) = |z|$

***Leaky** RLU:* fixes $\alpha_i = 0.01$

***Parametric** RLU (PReLU):* is a learnable parameter

Maxout units : It learns the activation function itself

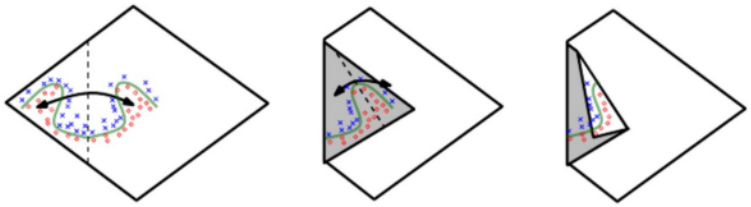
Queste sono altre generalizzazioni di RLU. La prima è la forma generalizzata di RLU.

Absolute value rectification è utile per i problemi di object recognition

- Absolute Value Rectification: it is the version of the ReLU function that

  fixes $\alpha_i = -1$ to obtain $g(z) = |z|$

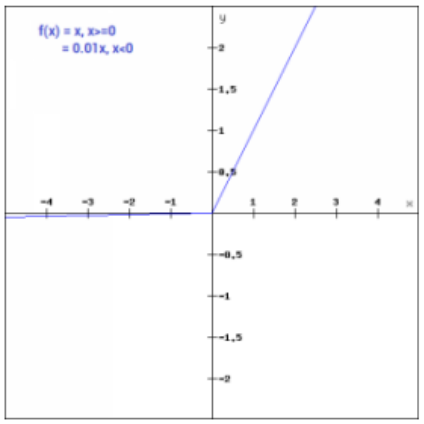
  It is used, for example, for object recognition from images where it makes sense to seek features that are invariant with respect to polarity reversal of input illumination



Per esempio la persona nella foto potrebbe avere una situazione di luce più chiara o più scura. Quindi sarebbero due versioni della stessa foto, e quindi l'output dovrebbe essere lo stesso. Quindi questo metodo può funzionare in situazioni di simmetria, ma di solito non funziona bene.

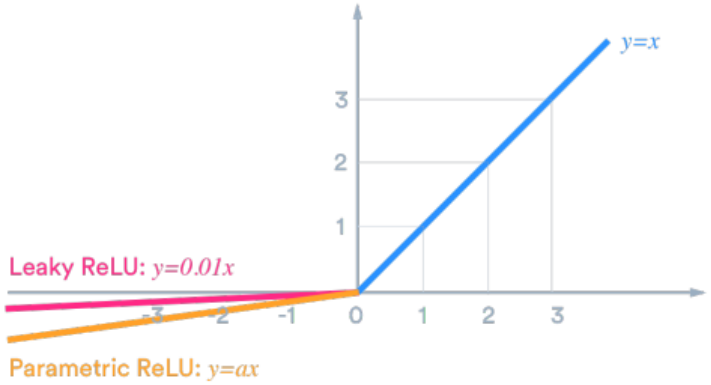
Con **leaky ReLU** il gradient non è mai zero, c'è una piccola pendenza a sinistra. Questo significa che se l'output è negativo viene comunque fatto un pochino di learning.

- It will have a negative slope (0.01)



Con **parametric ReLU** viene aggiunto un parametro che viene trovato durante l'apprendimento, che rappresenta la pendenza a sinistra.

- Instead of imposing a value for α this is considered as a learnable parameter



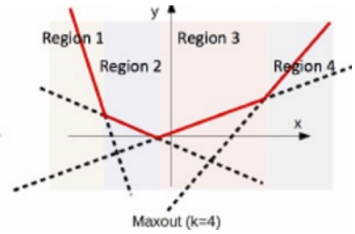
Con **maxout units** ...

Maxout units : It learns the activation function itself

It divides $\boldsymbol{z}$ in groups of k values and each maxout unit outputs the max element of one of these groups

$$g(z)_i = \max_{j \in G^{(i)}} z_j$$

$G^{(i)}$ = Set of indices of the group i



Scelta della funzione di attivazione

Come scegliere la funzione di attivazione? ReLU è lo standard.

- Sigmoid functions and their combinations generally work better in the case of classifiers
- Sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem
- ReLU function is a general activation function and is used in most cases these days
- If we encounter a case of dead neurons in our networks the leaky ReLU function is the best choice
- Always keep in mind that ReLU function should only be used in the hidden layers
- As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case ReLU doesn't provide with optimum results

tanh può sistemare il problema dove si arriva ad un punto dove non c'è più learning.

ReLU può essere usato solo negli hidden layers, non per l'output.

Di solito si inizia usando ReLU e nel caso cambiarla dopo se non funziona bene. Bisogna però capire perchè non funziona, per capire come muoversi.

Initialization Hints

- For bias b
  - Initialize all to 0
- For weights W
  - Can't initialize all weights to the same value
    - need to break symmetry
  - Idea:
    - sample around 0 but break symmetry
    - maintain the same variance across every layer

Lo xavier method dice di inizializzare i pesi con una distribuzione normale che ha media 0, mantenendo la varianza tra i layers uguale. In questo modo tutti i layer hanno le stesse opportunità, non ci sono layers che imparano di più o di meno.

simulazioni: https://www.deeplearning.ai/ai-notes/initialization/index.html