Neural Networks (Part III)

Machine Learning, A.Y. 2022/23, Padova

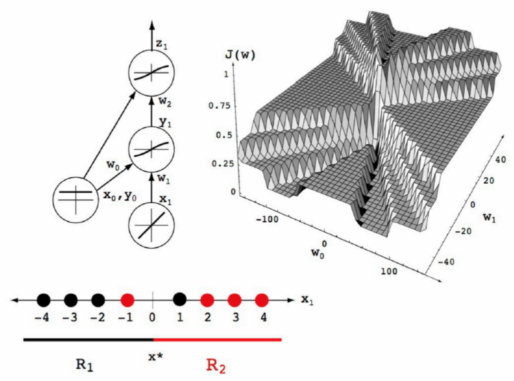
Fabio Aiolli

November 2nd, 2022

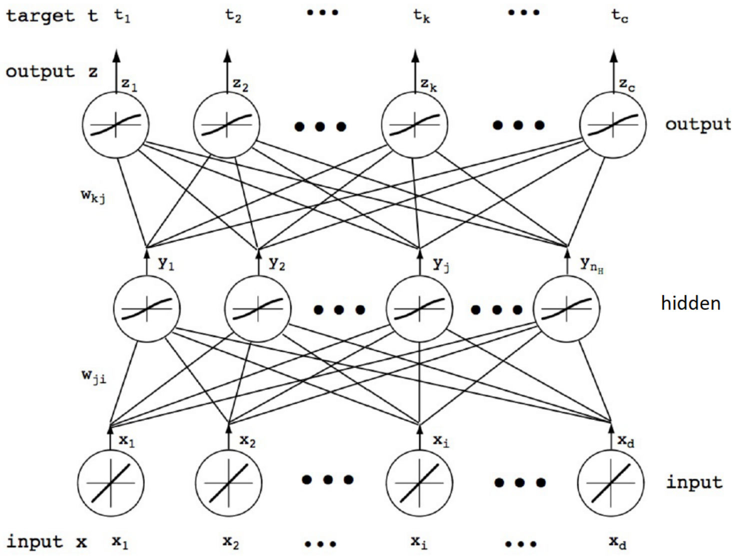
Multilayer Neural Networks: activation

Multiple layers of cascaded units makes a NN to implement arbitrarily complex non linear functions.

- Why not linear activations?
- Why not step activations?



Multilayer Neural Networks: topology

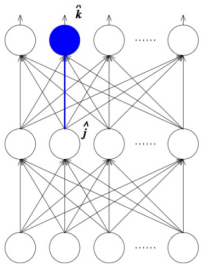


Multilayer Neural Networks: terminology

- d input units: d input data size $\mathbf{x} \equiv (x_1, \ldots, x_d)$, $d+1$ when including the bias in the weight vector $\mathbf{x}' \equiv (x_0, x_1, \ldots, x_d)$
- n_H hidden units (with output $\mathbf{y} \equiv (y_1, \ldots, y_{n_H})$)
- c output units: c size of data output $\mathbf{z} = (z_1, \ldots, z_c)$ and number of desired output $\mathbf{t} = (t_1, \ldots, t_c)$
- w_{ji} weight from the ith input unit to the jth hidden unit ($\mathbf{w}_j$ is the weight vector of the jth hidden unit)
- w_{kj} weight from the jth hidden unit to the kth output unit ($\mathbf{w}_k$ is the weight vector of the kth output unit)

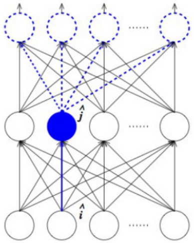
Computing gradient for the weights of a output unit

$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} &= \frac{\partial}{\partial w_{\hat{k}\hat{j}}}\left[\frac{1}{2cN}\sum_{s=1}^{N}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})^2\right] \\
&= \frac{1}{2cN}\sum_{s=1}^{N}\frac{\partial}{\partial w_{\hat{k}\hat{j}}}\left[\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})^2\right] \\
&= \frac{1}{2cN}\sum_{s=1}^{N}2(t_{\hat{k}}^{(s)} - z_{\hat{k}}^{(s)})\frac{\partial}{\partial w_{\hat{k}\hat{j}}}\left[(t_{\hat{k}}^{(s)} - z_{\hat{k}}^{(s)})\right] \\
&= \frac{1}{cN}\sum_{s=1}^{N}(t_{\hat{k}}^{(s)} - z_{\hat{k}}^{(s)})\frac{\partial}{\partial w_{\hat{k}\hat{j}}}\left[t_{\hat{k}}^{(s)} - \sigma(\mathbf{w}_{\hat{k}}\cdot\mathbf{y}^{(s)})\right] \\
&= -\frac{1}{cN}\sum_{s=1}^{N}(t_{\hat{k}}^{(s)} - z_{\hat{k}}^{(s)})\sigma'(\mathbf{w}_{\hat{k}}\cdot\mathbf{y}^{(s)})y_{\hat{j}}^{(s)} \\
&= -\frac{1}{cN}\sum_{s=1}^{N}(t_{\hat{k}}^{(s)} - z_{\hat{k}}^{(s)})z_{\hat{k}}(1 - z_{\hat{k}})y_{\hat{j}}^{(s)}
\end{aligned}
$$



Computing gradient for the weights of a hidden unit

$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} &= \frac{\partial}{\partial w_{\hat{j}\hat{i}}}\left[\frac{1}{2cN}\sum_{s=1}^{N}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})^2\right] \\
&= \frac{1}{2cN}\sum_{s=1}^{N}\sum_{k=1}^{c}\frac{\partial}{\partial w_{\hat{j}\hat{i}}}\left[(t_k^{(s)} - z_k^{(s)})^2\right] \\
&= \frac{1}{2cN}\sum_{s=1}^{N}\sum_{k=1}^{c}2(t_k^{(s)} - z_k^{(s)})\frac{\partial}{\partial w_{\hat{j}\hat{i}}}\left[-z_k^{(s)}\right] \\
&= -\frac{1}{cN}\sum_{s=1}^{N}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})\sigma'(\mathbf{w}_k \cdot \mathbf{y}^{(s)})\frac{\partial}{\partial w_{\hat{j}\hat{i}}}\left[\sum_{j=1}^{N_H} w_{kj}y_j^{(s)}\right] \\
&= -\frac{1}{cN}\sum_{s=1}^{N}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})\sigma'(\mathbf{w}_k \cdot \mathbf{y}^{(s)})w_{k\hat{j}}\frac{\partial}{\partial w_{\hat{j}\hat{i}}}\left[y_{\hat{j}}^{(s)}\right]
\end{aligned}
$$



Computing gradient for the weights of a hidden unit

$$
\begin{aligned}
&= -\frac{1}{cN}\sum_{s=1}^{N}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})\sigma'(\mathbf{w}_k \cdot \mathbf{y}^{(s)})w_{k\hat{j}}\frac{\partial}{\partial w_{\hat{j}\hat{i}}}\left[y_{\hat{j}}^{(s)}\right] \\
&= -\frac{1}{cN}\sum_{s=1}^{N}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})\sigma'(\mathbf{w}_k \cdot \mathbf{y}^{(s)})w_{k\hat{j}}\sigma'(\mathbf{w}_{\hat{j}} \cdot \mathbf{x}^{(s)})\frac{\partial}{\partial w_{\hat{j}\hat{i}}}\left[\mathbf{w}_{\hat{j}} \cdot \mathbf{x}^{(s)}\right] \\
&= -\frac{1}{cN}\sum_{s=1}^{N}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})\sigma'(\mathbf{w}_k \cdot \mathbf{y}^{(s)})w_{k\hat{j}}\sigma'(\mathbf{w}_{\hat{j}} \cdot \mathbf{x}^{(s)})x_{\hat{i}}^{(s)} \\
&= -\frac{1}{cN}\sum_{s=1}^{N}\sigma'(\mathbf{w}_{\hat{j}} \cdot \mathbf{x}^{(s)})x_{\hat{i}}^{(s)}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})w_{k\hat{j}}\sigma'(\mathbf{w}_k \cdot \mathbf{y}^{(s)}) \\
&= -\frac{1}{cN}\sum_{s=1}^{N}y_{\hat{j}}^{(s)}(1 - y_{\hat{j}}^{(s)})x_{\hat{i}}^{(s)}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})z_k^{(s)}(1 - z_k^{(s)})w_{k\hat{j}}
\end{aligned}
$$

Batch vs. stochastic gradient descent

- **Batch**. Until the termination condition is satisfied:
  1. Compute $\nabla E_S[\mathbf{w}]$
  2. $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_S[\mathbf{w}]$
- **Stochastic**. Until the termination condition is satisfied, for each training example s:
  1. Compute $\nabla E_s[\mathbf{w}]$
  2. $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_s[\mathbf{w}]$
- **Mini-batch**. Until the termination condition is satisfied, consider a subset of examples $Q \subseteq S$:
  1. Compute $\nabla E_Q[\mathbf{w}]$
  2. $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_Q[\mathbf{w}]$

where, in general, $E_Q[\mathbf{w}] = \frac{1}{2cN} \sum_{s \in Q} \sum_{k=1}^{c} (t_k^{(s)} - z_k^{(s)})^2$, $Q \subseteq S$.

Backprop algorithm with a hidden layer

Back-Propagation-1hl-stochastic(S, η):

- Initialize all weights with small random values (e.g. between -0.05 and +0.05)
- Until the termination condition is satisfied:
  - For each $(\mathbf{x}, \mathbf{t}) \in S$,
    1. Present $\mathbf{x}$ to the net and compute the vectors $\mathbf{y}$ e $\mathbf{z}$
    2. For each output unit k,

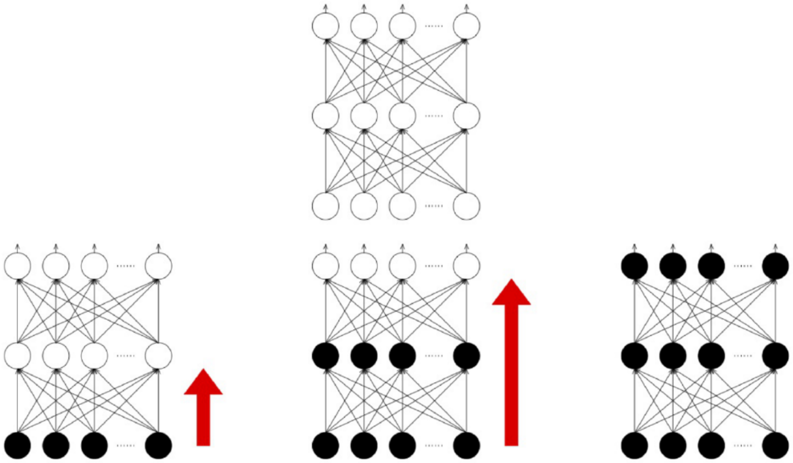
$$\begin{aligned} \delta_k &= z_k(1 - z_k)(t_k - z_k) \\ \Delta w_{kj} &= \delta_k y_j \end{aligned}$$

    3. For each hidden unit j,

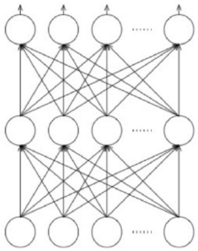
$$\begin{aligned} \delta_j &= y_j(1 - y_j) \sum_{k=1}^{c} w_{kj} \delta_k \\ \Delta w_{ji} &= \delta_j x_i \end{aligned}$$

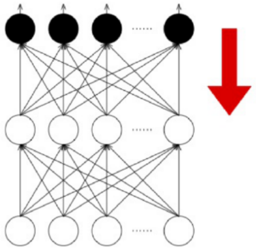
    4. Update all weights: $w_{sq} \leftarrow w_{sq} + \eta \Delta w_{sq}$

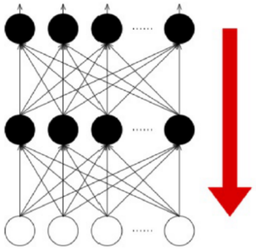
Forward phase



Backward phase







Computational power of neural networks

Theorem (Pinkus, 1996) simplified

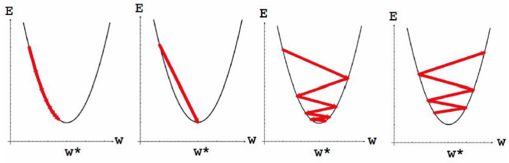
Given a feed-forward NN with just one hidden layer, any continuous function $f : \mathbb{R}^n \to \mathbb{R}$, an arbitrarily small $\epsilon > 0$, then, for a large class of activation functions, there always exists an integer M such that, the function $g : \mathbb{R}^n \to \mathbb{R}$ computed by the net using atleast M hidden units approximates the funcion f with tolerance ϵ, that is:

$$\max_{\mathbf{x}\in\Omega} |f(\mathbf{x}) - g(\mathbf{x})| < \epsilon$$

Note that the theorem attests the existence of a NN with M hidden units that approximates the target function with the desired tolerance but it says nothing about how M can be computed!

Some issues..

- The choice of the net typology determines the hypothesis space actually used. With three-layers architectures (input, hidden, output), the number of hidden units determines the complexity of the hypothesis space;
- The choice of the descent step (value of η) can be crucial for the convergence:

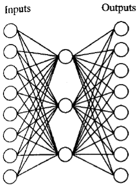


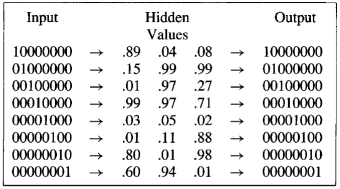
- Training is generally slow (but the output computation is fast);
- Local minima (even if effective in practice).

Avoiding local minima

- Adding a term to the weight update rule (called moment). Practically, a contribution deriving from the previous step is added, imposing a kind of inertia on the system.
- Using stochastic training (randomizing on the examples) in place of batch training can facilitate exit from local minima;
- Training multiple NNs on the same data with different initialization and selection of the most effective net (for instance by evaluation on a validation set). Alternatively, one can consider a "committee" of NNs where the prediction is the average (possibly weighted) of the prediction of individual NNs.

Hidden layers representation



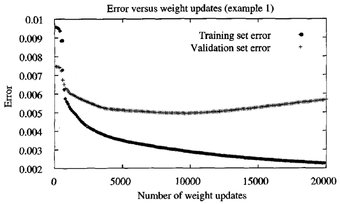


Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

FIGURE 4.7
Learned Hidden Layer Representation. This $8 \times 3 \times 8$ network was trained to learn the identity function, using the eight training examples shown. After 5000 training epochs, the three hidden unit values encode the eight distinct inputs using the encoding shown on the right. Notice if the encoded values are rounded to zero or one, the result is the standard binary encoding for eight distinct values.

An important feature of multi-layer NNs is that they can uncover useful representations of input data (alternative to the input representation). Specifically, the output of the hidden units is an effective (new) input representation that permits a simpler separation of the data in output. In the example, we note how the learned representation (encoding) looks like a 3 bit binary encoding ($[100, 011, 010, \dots]$).

Overfitting



- As the number of weight updates increases, the error on the validation set first decreases then increases. Why?
- At the beginning, with small absolute values of the weights (similar each other) the decision surfaces are "smoother". When the absolute values of the weights increases the complexity of the decision surfaces increases and with it the possibility to suffer overfitting.
- Solutions: Monitor the error on a dataset of validation as the training proceeds, or use an additional term in the error function dependent on the norm of the weights (regularization).

Recap

Notions

- Multi-layer neural networks
- Derivation of the Backpropagation algorithm
- Batch, stochastic, and mini-batch training
- Universality of NNs
- Issues of NNs
- Repprentation of the hidden layers

Exercizes

- Extend the derivation of Backprop to more than one hidden layers;
- Implement the network given on Slide 15 and verify the activations of hidden units.