Neural Net: feed-forward architecture and back-propagation

A. Allauzen

Université Paris-Sud / LIMSI-CNRS

14 novembre 2017

Outline

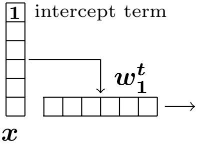
1. Neural Nets : Basics
   - Terminology
   - Training by back-propagation

Outline

1. Neural Nets : Basics
   - Terminology
   - Training by back-propagation

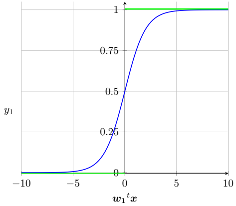
A choice of terminology

Logistic regression (binary classification)



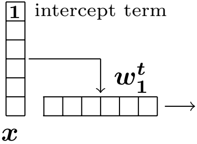
$$\longrightarrow \; y_1 = f(\boldsymbol{w_1}^t \boldsymbol{x})$$

$$f(a = \boldsymbol{w_1}^t \boldsymbol{x}) = \frac{1}{1 + e^{-a}}$$



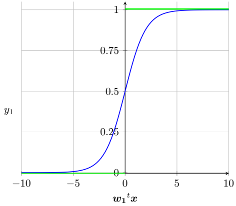
A choice of terminology

Logistic regression (binary classification)

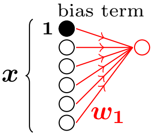


$$y_1 = f(\boldsymbol{w_1}^t \boldsymbol{x})$$

$$f(a = \boldsymbol{w_1}^t \boldsymbol{x}) = \frac{1}{1 + e^{-a}}$$



A single artificial neuron

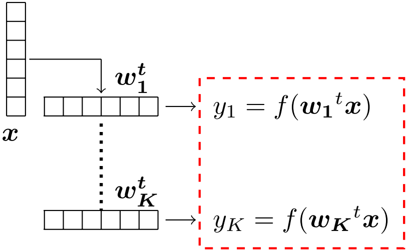


pre-activation : $a_1 = \boldsymbol{w_1}^t \boldsymbol{x}$

$y_1 = f(\boldsymbol{w_1}^t \boldsymbol{x})$, f is the activation function of the neuron

A choice of terminology - 2

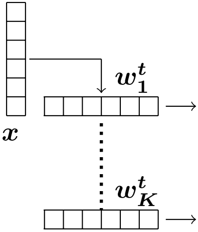
From binary classification to K classes (Maxent)



$$f(a_k = \boldsymbol{w_k}^t\boldsymbol{x}) = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}} = \frac{e^{a_k}}{Z(\boldsymbol{x})}$$

A choice of terminology - 2

From binary classification to K classes (Maxent)

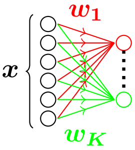


$$y_1 = f(\boldsymbol{w_1}^t \boldsymbol{x})$$

$$y_K = f(\boldsymbol{w_K}^t \boldsymbol{x})$$

$$f(a_k = \boldsymbol{w_k}^t \boldsymbol{x}) = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}} = \frac{e^{a_k}}{Z(\boldsymbol{x})}$$

A simple neural network

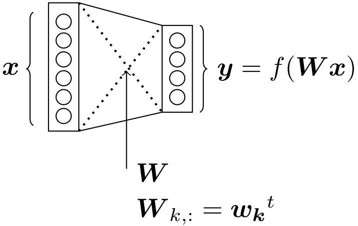


$$y_1 = f(\boldsymbol{w_1^t x})$$

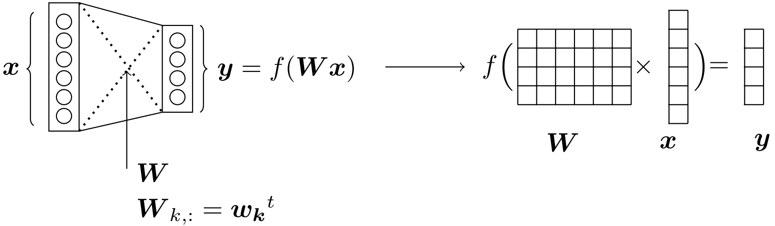
$$y_K = f(\boldsymbol{w_K^t x})$$

- $\boldsymbol{x}$: *input layer*
- $\boldsymbol{y}$: *output layer*
- each y_k has its parameters $\boldsymbol{w}_k$
- f is the **softmax** function

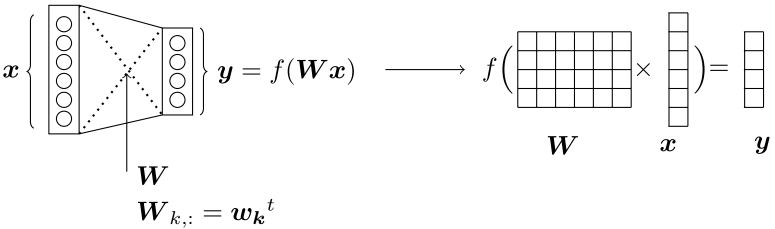
Two layers fully connected



Two layers fully connected



Two layers fully connected



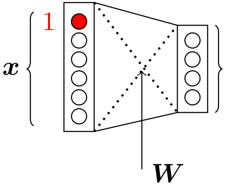
- f is usually a non-linear function
- f is a component wise function
- $e.g$ the softmax function :

$$y_k = P(c = k|\boldsymbol{x}) = \frac{e^{\boldsymbol{w_k}^t \boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{w_{k'}}^t \boldsymbol{x}}} = \frac{e^{\boldsymbol{W}_{k,:}\boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{W}_{k',:}\boldsymbol{x}}}$$

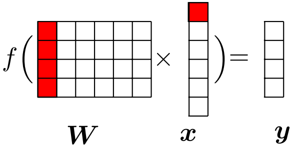
- tanh, sigmoid, relu, ...

Bias or not bias

Implicit Bias

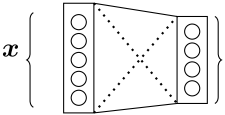


$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x}) \longrightarrow$$

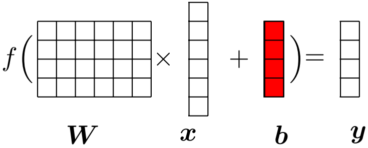


$$\boldsymbol{W}_{k,:} = \boldsymbol{w_k}^t$$

Explicit bias

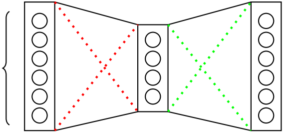


$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \longrightarrow$$



With neural network : add a hidden layer

$\boldsymbol{x}$: raw input representation



$\boldsymbol{y} = f(\boldsymbol{W}^{(2)}\boldsymbol{h})$

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

the internal and tailored representation

Intuitions

- Learn an internal representation of the raw input
- Apply a non-linear transformation
- The input representation $\boldsymbol{x}$ is transformed/compressed in a new representation $\boldsymbol{h}$
- Adding more layers to obtain a more and more abstract representation

How do we learn the parameters ?

For a supervised single layer neural net

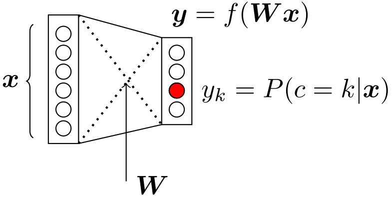
Just like a maxent model :

- Calculate the gradient of the objective function and use it to iteratively update the parameters.
- Conjugate gradient, L-BFGS, ...
- In practice : **Stochastic gradient descent (SGD)**

With one hidden layer

- The internal (“hidden”) units make the function non-convex ... just like other models with hidden variables :
  - hidden CRFs (**?**), ...
- But we can use the same ideas and techniques
- Just without guarantees ⇒ **backpropagation** (**?**)

Ex. 1 : A single layer network for classification



$\boldsymbol{\theta}$= the set of parameters, in this case :

$$\boldsymbol{\theta} = (\boldsymbol{W})$$

The log-loss (conditional log-likelihood)

Assume the dataset $\mathcal{D} = (\boldsymbol{x}_{(i)}, c_{(i)})_{i=1}^{N}$, $c_{(i)} \in \{1, 2, \ldots, C\}$

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)}) = \sum_{i=1}^{N} \Big(- \sum_{c=1}^{C} \mathbb{I}\big\{c = c_{(i)}\big\} \log(P(c|\boldsymbol{x}_{(i)})) \Big) \quad (1)$$

$$l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)}) = - \sum_{k=1}^{C} \mathbb{I}\big\{k = c_{(i)}\big\} \log(y_k) \quad (2)$$

Ex. 1 : optimization method

Stochastic Gradient Descent (?)

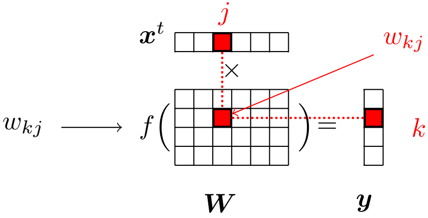
For ($t = 1$; until convergence ; $t++$) :

- Pick randomly a sample $(\boldsymbol{x}_{(i)}, c_{(i)})$
- Compute the gradient of the loss function w.r.t the parameters ($\nabla_{\boldsymbol{\theta}}$)
- Update the parameters : $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta_t \nabla_{\boldsymbol{\theta}}$

Questions

- convergence : what does it mean ?
- what do you mean by η_t ?
  - convergence if $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$
  - $\eta_t \propto t^{-1}$
  - and lot of variants like Adagrad (?), Down scheduling, ... see (?)

Ex. 1 : compute the gradient - 1



Inference chain :

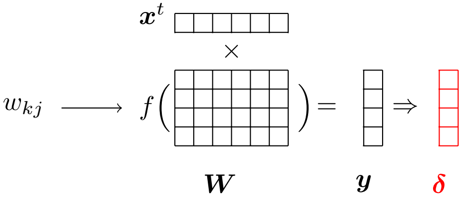
$$\boldsymbol{x}_{(i)} \longrightarrow (\boldsymbol{a} = \boldsymbol{W}\boldsymbol{x}_{(i)}) \longrightarrow (\boldsymbol{y} = f(\boldsymbol{a})) \longrightarrow l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})$$

The gradient for w_{kj}

$$\nabla_{w_{kj}} = \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial w_{kj}} = \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial \boldsymbol{y}} \times \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}} \times \frac{\partial \boldsymbol{a}}{\partial w_{kj}}$$

$$= -(\mathbb{I}\left\{k = c_{(i)}\right\} - y_k)x_j = \delta_k x_j$$

Ex. 1 : compute the gradient - 2



Generalization

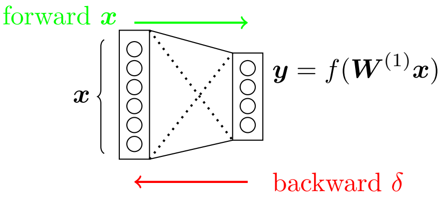
$$\nabla_{\boldsymbol{W}} = \boldsymbol{\delta}\boldsymbol{x}^t$$

$$\boldsymbol{\delta} = \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial \boldsymbol{a}}$$

$$\delta_k = -(\mathbb{I}\left\{k = c_{(i)}\right\} - y_k)$$

with $\boldsymbol{\delta}$ the gradient at the pre-activation level.

Ex. 1 : Summary



Inference : a forward step

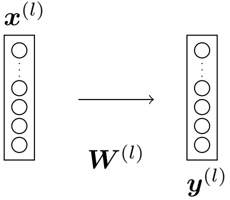
- matrice multiplication with the input $\boldsymbol{x}$
- Application of the activation function

One training step : forward and backward steps

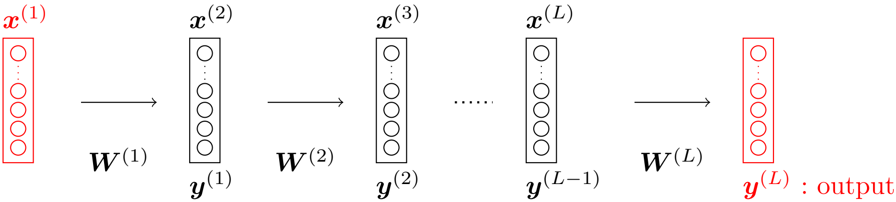
- Pick randomly a sample $(\boldsymbol{x}_{(i)}, c_{(i)})$
- Compute $\boldsymbol{\delta}$
- Update the parameters : $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta_t \boldsymbol{\delta} \boldsymbol{x}^t$

Notations for a multi-layer neural network (feed-forward)

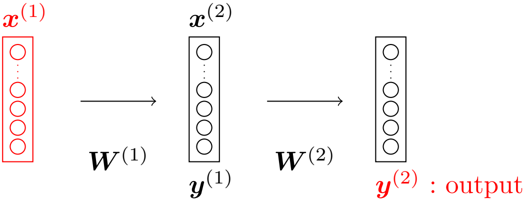
One layer, indexed by l



- $\boldsymbol{x}^{(l)}$: input of the layer l
- $\boldsymbol{y}^{(l)} = f^{(l)}(\boldsymbol{W}^{(l)}\ \boldsymbol{x}^{(l)})$
- stacking layers : $\boldsymbol{y}^{(l)} = \boldsymbol{x}^{(l+1)}$
- $\boldsymbol{x}^{(1)}$ = a data example



Ex. 2 : with one hidden layer



$$\boldsymbol{\theta} = (\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)})$$

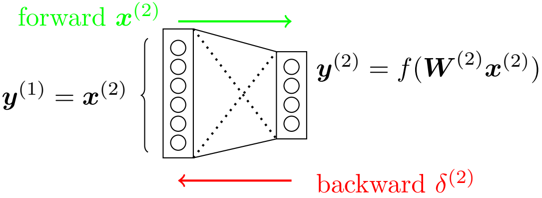
To learn, we need the gradients for :

- the output layer : $\nabla_{\boldsymbol{W}^{(2)}}$
- the hidden layer : $\nabla_{\boldsymbol{W}^{(1)}}$

Back-propagation of the loss gradient

For the output layer

As in the Ex. 1 :



$$
\begin{aligned}
\nabla_{\boldsymbol{W}^{(2)}} &= \boldsymbol{\delta}^{(2)}{\boldsymbol{x}^{(2)}}^t, \text{ with}\\
\delta_k^{(2)} &= -(\mathbb{I}\left\{k = c_{(i)}\right\} - y_k)\\
\boldsymbol{y} &\rightarrow \boldsymbol{y}^{(2)}\\
\boldsymbol{W} &\rightarrow \boldsymbol{W}^{(2)}\\
\boldsymbol{x} &\rightarrow \boldsymbol{x}^{(2)} = \boldsymbol{y}^{(1)}
\end{aligned}
$$

Back-propagation of the loss gradient

For the hidden layer - 1

The goal : compute $\boldsymbol{\delta}^{(1)}$

Inference (/forward) chain from $\boldsymbol{a}^{(1)}$ to the output :

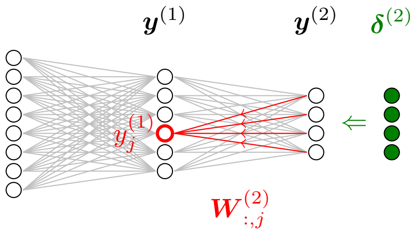
$$\boldsymbol{y}^{(1)} = f^{(1)}(\boldsymbol{a}^{(1)}) \rightarrow \left(\boldsymbol{a}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{y}^{(1)}\right) \rightarrow \left(\boldsymbol{y}^{(2)} = f^{(2)}(\boldsymbol{a}^{(2)})\right) \rightarrow l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})$$

Backward / Back-propagation :

$$\delta_j^{(1)} = \nabla_{a_j^{(1)}} = \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial a_j^{(1)}} = \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial \boldsymbol{y}^{(2)}} \times \frac{\partial \boldsymbol{y}^{(2)}}{\partial \boldsymbol{a}^{(2)}} \times \frac{\partial \boldsymbol{a}^{(2)}}{\partial y_j^{(1)}} \times \frac{\partial y_j^{(1)}}{\partial a_j^{(1)}}$$

Back-propagation of the loss gradient

For the hidden layer - 2



Backward / Back-propagation :

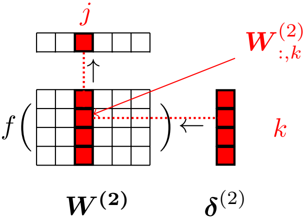
$$\delta_j^{(1)} = \nabla_{a_j^{(1)}} = \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial a_j^{(1)}} = \frac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial \boldsymbol{y}^{(2)}} \times \frac{\partial \boldsymbol{y}^{(2)}}{\partial \boldsymbol{a}^{(2)}} \times \frac{\partial \boldsymbol{a}^{(2)}}{\partial y_j^{(1)}} \times \frac{\partial y_j^{(1)}}{\partial a_j^{(1)}}$$

$$= f'^{(1)}(a_j)\left(\boldsymbol{W}_{:,j}^{(2)\,t} \boldsymbol{\delta}^{(2)}\right)$$

Back-propagation of the loss gradient

For the hidden layer - 3

the back-propagated signal $\nabla_{\boldsymbol{y}^{(1)}} \leftarrow$

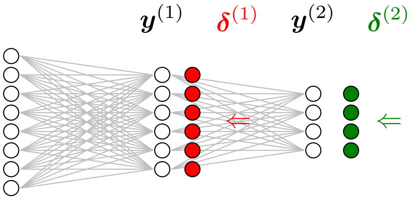


$$\nabla_{\boldsymbol{y}^{(1)}} = \boldsymbol{W}^{(2)^t}\boldsymbol{\delta}^{(2)}, \text{ then}$$

$$\boldsymbol{\delta}^{(1)} = \nabla_{\boldsymbol{a}^{(1)}} = f^{(1)'}(\boldsymbol{a}^{(1)}) \circ \left(\boldsymbol{W}^{(2)^t}\boldsymbol{\delta}^{(2)}\right)$$

Back-propagation of the loss gradient

For the hidden layer - 4



As for the output layer, the gradient is :

$$\nabla_{\boldsymbol{W}^{(1)}} = \boldsymbol{\delta}^{(1)} {\boldsymbol{x}^{(1)}}^t, \text{ with}$$
$$\delta_j^{(1)} = \nabla_{a_j^{(1)}}$$
$$\boldsymbol{\delta}^{(1)} = {f'}^{(1)}(\boldsymbol{a}^{(1)}) \circ ({\boldsymbol{W}^{(2)}}^t \boldsymbol{\delta}^{(2)})$$

The term $({\boldsymbol{W}^{(2)}}^t \boldsymbol{\delta}^{(2)})$ comes from the upper layer.

Back-propagation : generalization

For a hidden layer l :

- The gradient at the pre-activation level :

$$\boldsymbol{\delta}^{(l)} = {f'}^{(l)}(\boldsymbol{a}^{(l)}) \circ \left(\boldsymbol{W}^{(l+1)^t} \boldsymbol{\delta}^{(l+1)}\right)$$

- The update is as follows :

$$\boldsymbol{W}^{(l)} = \boldsymbol{W}^{(l)} - \eta_t \boldsymbol{\delta}^{(l)} \boldsymbol{x}^{(l)^t}$$

The layer should keep :

- $\boldsymbol{W}^{(l)}$: the parameters
- $f^{(l)}$: its activation function
- $\boldsymbol{x}^{(l)}$: its input
- $\boldsymbol{a}^{(l)}$: its pre-activation associated to the input
- $\boldsymbol{\delta}^{(l)}$: for the update and the back-propagation to the layer $l-1$

Back-propagation : one training step

Pick a training example : $\boldsymbol{x}^{(1)} = \boldsymbol{x}_{(i)}$

Forward pass

For $l = 1$ to $(L-1)$

- Compute $\boldsymbol{y}^{(l)} = f^{(l)}(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)})$
- $\boldsymbol{x}^{(l+1)} = \boldsymbol{y}^{(l)}$

$\boldsymbol{y}^{(L)} = f^{(L)}(\boldsymbol{W}^{(L)}\boldsymbol{x}^{(L)})$

Backward pass

Init : $\boldsymbol{\delta}^{(L)} = \nabla_{\boldsymbol{a}^{(L)}}$
For $l = L$ to 2 // all hidden units

- $\boldsymbol{\delta}^{(l-1)} = f'^{(l-1)}(\boldsymbol{a}^{(l-1)}) \circ \left({\boldsymbol{W}^{(l)}}^t \boldsymbol{\delta}^{(l)}\right)$
- $\boldsymbol{W}^{(l)} = \boldsymbol{W}^{(l)} - \eta_t \boldsymbol{\delta}^{(l)} {\boldsymbol{x}^{(l)}}^t$

$\boldsymbol{W}^{(1)} = \boldsymbol{W}^{(1)} - \eta_t \boldsymbol{\delta}^{(1)} {\boldsymbol{x}^{(1)}}^t$

Conclusion on back-propagation for one layer l

Training a NNet relies on forward-backward propagation.

Forward :

- get $\boldsymbol{x}^{(l)}$ for the previous layer ;
- compute and send $\boldsymbol{y}^{(l)} = f^{(l)}(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)})$.

Backward :

- get $\boldsymbol{\delta}^{(l)}$ as input from the up-comming layer ;
- compute and send $\boldsymbol{\delta}^{(l-1)}$ to the previous layer ;
- update parameters $\boldsymbol{W}^{(l)}$

Initialization recipes

A difficult question with several empirical answers.

One standard trick

$$\boldsymbol{W} \sim \mathcal{N}(0, \frac{1}{\sqrt{n_{in}}})$$

with n_{in} is the number of inputs

A more recent one

$$\boldsymbol{W} \sim \mathcal{U}\Big[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\Big]$$

with n_{in} is the number of inputs