Neural Nets, Deep-Learning and applications (NLP)

A. Allauzen

Université Paris-Sud / LIMSI-CNRS

11/04/2016

Menu of the day

Before the lunch break

1. Inference and training of feed-forward NNet
2. Known issues and solutions
3. Advanced architecture

After the lunch break

Lab session with Gaétan Marceau-Caron

Outline - Part 1

1. Introduction

2. Neural Nets : Basics
   - Introduction to multi-layered neural network
   - Optimization via back-propagation

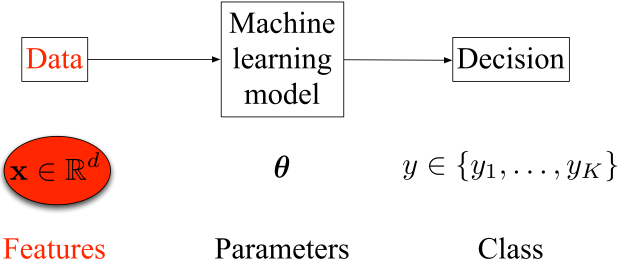
Outline

1. Introduction

2. Neural Nets : Basics
    - Introduction to multi-layered neural network
    - Optimization via back-propagation

Feature engineering

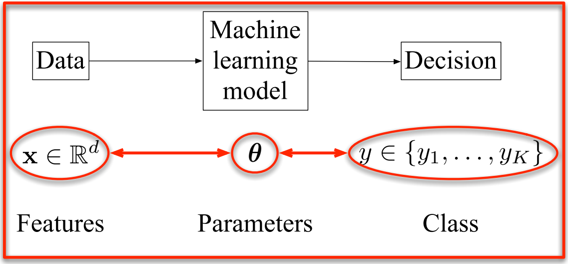
Most current machine learning works well because of human-designed representations and input features



- Time consuming and task/domain dependant
- Features are often both over-specified and incomplete
- Machine learning $\Leftrightarrow$ optimizing parameters to make the best prediction

Representation learning and Deep networks

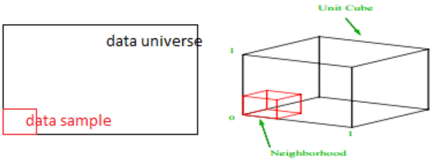
Representation learning attempts to automatically learn useful features



- Learning a hierarchical and abstract representation
- That can be shared among tasks
- Almost all data is unlabeled $\Rightarrow$ unsupervised learning

The curse of dimensionality

In high-dimensional space, training data becomes sparse



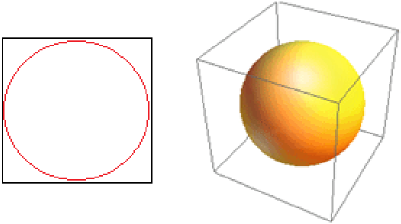
http ://www.edupristine.com/blog/curse-dimensionality

To generalize :

- Use the distance to define some sort of “near-ness”
- Spread the probability mass around training examples (smooth the empirical distribution)

The curse of dimensionality - 2

In 2-dimensions, two points are near if one falls within a certain radius of another.



http://www.edupristine.com/blog/curse-dimensionality

In 2-d, which proportion of uniformly spaced points within black square fall inside the red circle ?

$$\frac{\pi r^2}{4r^2} = \frac{\pi}{4} \approx 78\%$$

This proportion drops to 52% in 3-d, and to 0.24% in 10-d.

Consequence

In high-dimensional space, the distance does not define a useful similarity.

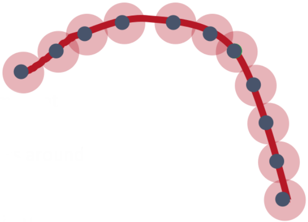
The curse of dimensionality - 3

Smoothing distribution

- The mass is spread around the examples.
- While plausible in this 2-dimensional case, in higher dimensions, the balls will leave holes or be too large in high probability regions.

Manifold

- If we can discover a representation of the probability concentration,
- a lower dimensional (non-linear) manifold,
- we can "flatten" it by changing the representation
- for which the distance is useful for density estimation, interpolation, ...



Y. Bengio, 2015

Neural Networks

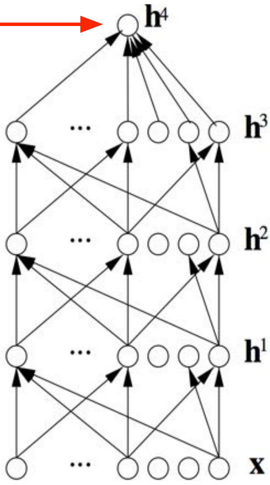
Output layer

prediction of the supervised target

Hidden Layers

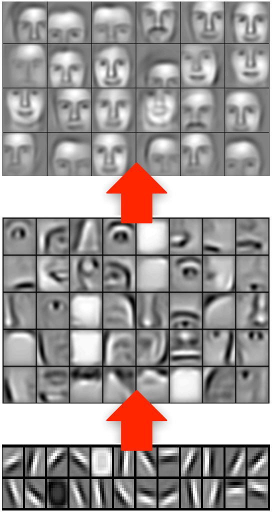
learn more abstract features as you head up

Raw inputs



(from Bengio, 2015)

Illustration

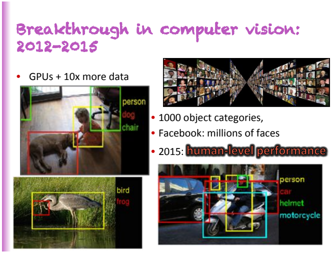


(Lee et al.2009)

Deep learning in neural networks : a success story

Since 2009, deep learning approaches won several challenges

- ImageNet since 2012 (Krizhevsky et al.2012)
- Traffic signs recognition : superhuman performance in 2011 (Ciresan et al.2012) based on (LeCun et al.1989)
- Handwritting recognition since 2009 (Graves and Schmidhuber2009) based on (Hochreiter and Schmidhuber1997)
- Automatic Speech recognition (Hinton et al.2012)



Deep learning in neural networks : a success story

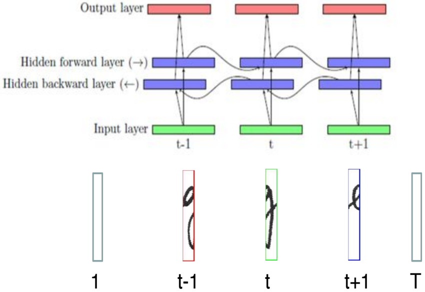
Since 2009, deep learning approaches won several challenges

- ImageNet since 2012 (Krizhevsky et al.2012)
- Traffic signs recognition : superhuman performance in 2011 (Ciresan et al.2012) based on (LeCun et al.1989)
- Handwritting recognition since 2009 (Graves and Schmidhuber2009) based on (Hochreiter and Schmidhuber1997)
- Automatic Speech recognition (Hinton et al.2012)

Deep learning in neural networks : a success story

Since 2009, deep learning approaches won several challenges

- ImageNet since 2012 (Krizhevsky et al.2012)
- Traffic signs recognition : superhuman performance in 2011 (Ciresan et al.2012) based on (LeCun et al.1989)
- Handwritting recognition since 2009 (Graves and Schmidhuber2009) based on (Hochreiter and Schmidhuber1997)
- Automatic Speech recognition (Hinton et al.2012)



Deep learning in neural networks : a long story

The breakthrough of 2006

The expression *Deep Learning* was coined around 2006 with papers on unsupervised pre-training of neural nets (Hinton et al.2006; Hinton and Salakhutdinov2006; Bengio et al.2007)

And before ? (just a few dates)

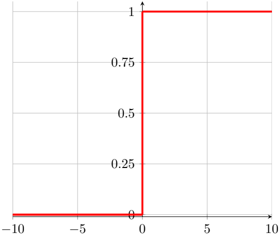
- 1958 Rosenblatt proposed the perceptron (Rosenblatt1958), following the work of McCulloch and Pitts in 1943 and Hebb in 1949.
- 1980 Neocognitron (Fukushima1980) or the multilayered NNets
- 1982 Hopfield network with memory and recurrence (Hopfield1982), the unsupervised SOM (Kohonen1982), Neural PCA (Oja1982)
- 1986 Multilayer perceptrons and backpropagation (Rumelhart et al.1986)
- 1989 Autoencoders (Baldi and Hornik1989), Convolutional network (LeCun et al.1989)
- 1993 Sparse coding (Field1993)

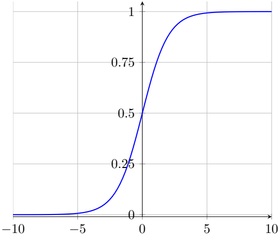
What is new ?

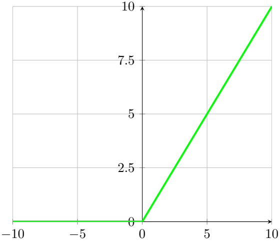
From Chris Bishop's slides (2015)

Why today ?

- The huge amount of data and the growth of computational power.
- Regularization
- and ...







What is new ?

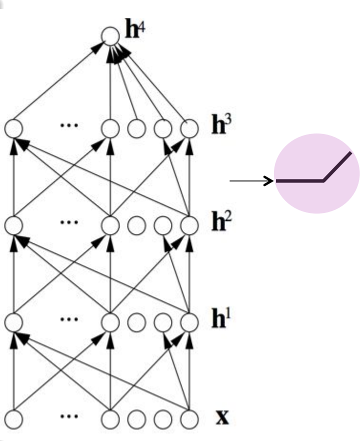
From Kyunghyun Cho's slides (2015)

Why today ?

- We have connected the dots, e.g. (Probabilistic) PCA / Neural PCA / Autoencoder
- We understand learning better (regularization, architecture)
- No need to be scared of non-convex optimization (initialization)
- The huge amount of data and the growth of computational power.

What is the difference between a NNet and a Deep Network ?

An intensive empirical exploration of different issues



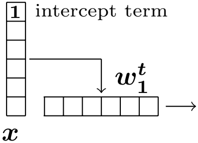
Y. Bengio, 2015

Outline

Outline

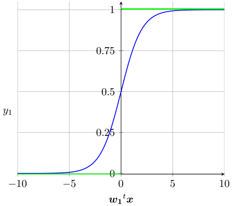
A choice of terminology

Logistic regression (binary classification)



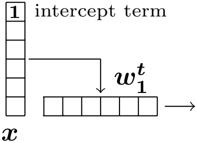
$$y_1 = f(\boldsymbol{w_1}^t \boldsymbol{x})$$

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



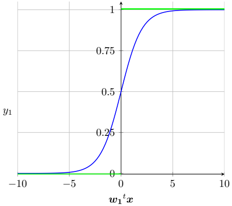
A choice of terminology

Logistic regression (binary classification)

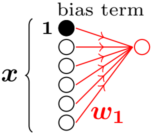


$\longrightarrow \; y_1 = f(\boldsymbol{w_1}^t\boldsymbol{x})$

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



A single artificial neuron

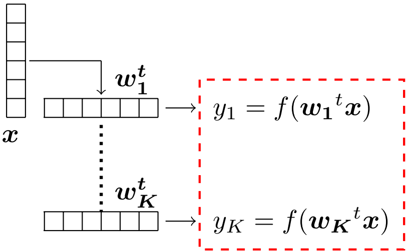


pre-activation : $a_1 = \boldsymbol{w_1}^t\boldsymbol{x}$

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

A choice of terminology - 2

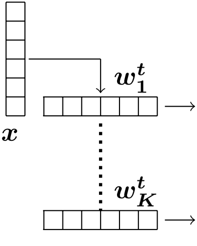
From binary classification to K classes (Maxent)



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

A choice of terminology - 2

From binary classification to K classes (Maxent)

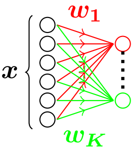


$$y_1 = f(\boldsymbol{w_1}^t\boldsymbol{x})$$

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

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

A simple neural network

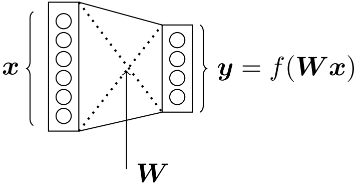


$$y_1 = f(\boldsymbol{w_1^t x})$$

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

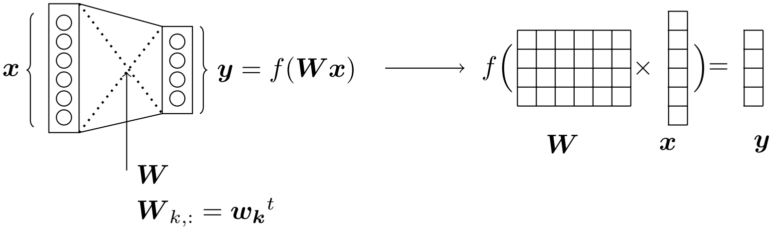
- $\boldsymbol{x}$: *input layer*
- $\boldsymbol{y}$: *output layer*
- each y_k has its parameters $\boldsymbol{w}_k$
- f is the softmax function

Two layers fully connected

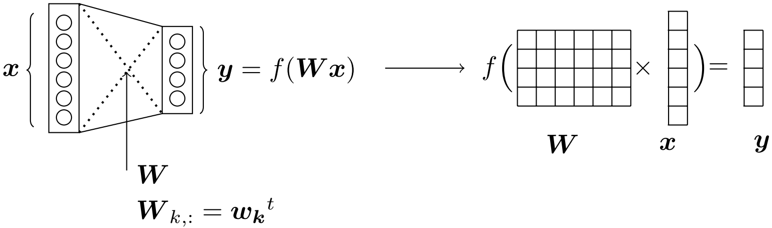


$$\boldsymbol{W}_{k,:} = \boldsymbol{w_k}^t$$

Two layers fully connected



Two layers fully connected



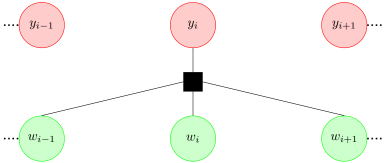
- f is usually a non-linear function
- f is a component wise function
- $e.g$ the softmax function :

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

- tanh, sigmoid, relu, ...

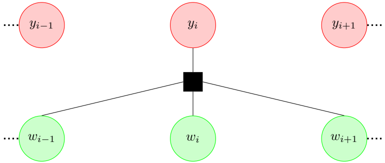
Feature engineering for Maxent classifiers

Ex. POS tagging



Feature engineering for Maxent classifiers

Ex. POS tagging



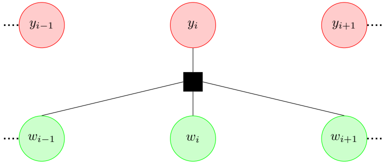
Word representation

For each word in the context

- surface form (one-hot vector)

Feature engineering for Maxent classifiers

Ex. POS tagging



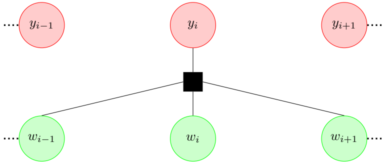
Word representation

For each word in the context

- surface form (one-hot vector)
- prefix
- suffix
- ...

Feature engineering for Maxent classifiers

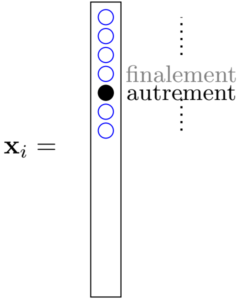
Ex. POS tagging



Word representation

For each word in the context

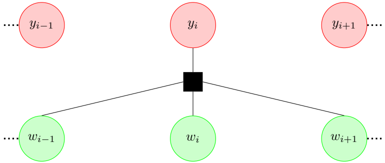
- surface form (one-hot vector)
- prefix
- suffix
- ...



A rich representation of the input for a better generalization.

Feature engineering for Maxent classifiers

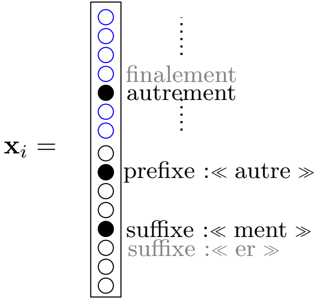
Ex. POS tagging



Word representation

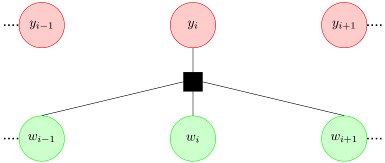
For each word in the context

- surface form (one-hot vector)
- prefix
- suffix
- ...



Feature engineering for Maxent classifiers

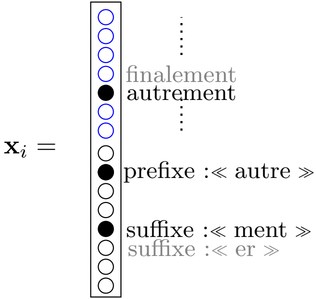
Ex. POS tagging



Word representation

For each word in the context

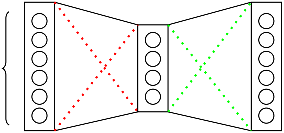
- surface form (one-hot vector)
- prefix
- suffix
- ...



A rich representation of the input for a better generalization.

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 :

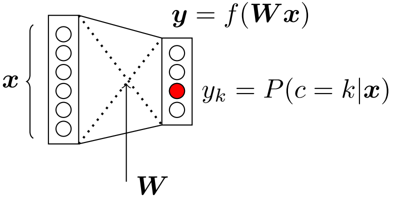
- Calculate the gradient of the objective function and use it to iteratively update the parameters.
- Conjugate gradient, L-BFGS, ...
- In practice : **Stochastic gradient descent (SGD)**

With one hidden layer

- The internal ("hidden") units make the function non-convex ... just like other models with hidden variables :
  - hidden CRFs (Quattoni et al.2007), ...
- But we can use the same ideas and techniques
- Just without guarantees ⇒ **backpropagation** (Rumelhart et al.1986)

Outline

Ex. 1 : A single layer network for classification



The set of parameters is denoted $\boldsymbol{\theta}$, in this case :

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

The log-loss (conditional log-likelihood)

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

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

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

Ex. 1 : optimization method

Stochastic Gradient Descent (Bottou2010)

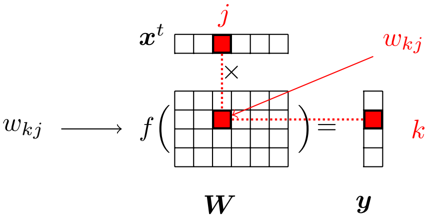
For ($t = 1$; until convergence ; $t++$) :

- 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 (Duchi et al.2011), Down scheduling, ... see (LeCun et al.2012)

Ex. 1 : compute the gradient - 1



Inference chain :

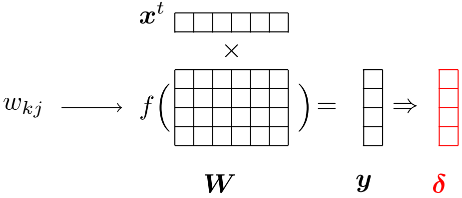
$$\boldsymbol{x}_{(i)} \longrightarrow (\boldsymbol{a} = \boldsymbol{W}\boldsymbol{x}_{(i)}) \longrightarrow (\boldsymbol{y} = f(\boldsymbol{a})) \longrightarrow l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})$$

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



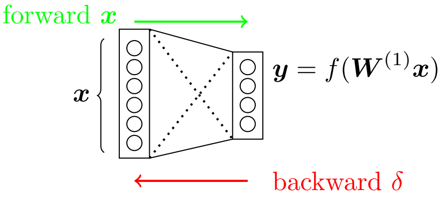
Generalization

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

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

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

Ex. 1 : Summary



Inference : a forward step

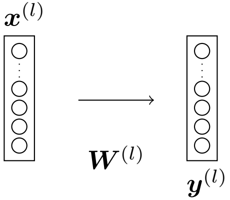
- matrice multiplication with the input $\boldsymbol{x}$
- Application of the activation function

One training step : forward and backward steps

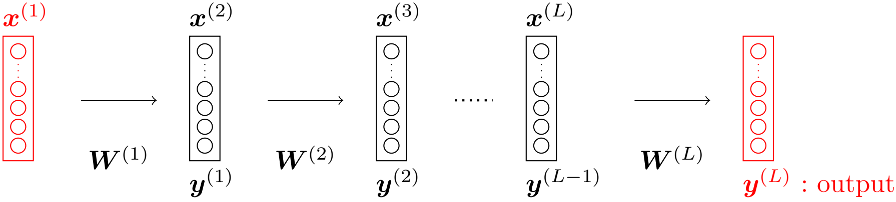
- Pick randomly a sample $(\boldsymbol{x}_{(i)}, c_{(i)})$
- Compute $\boldsymbol{\delta}$
- Update the parameters : $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta_t \boldsymbol{\delta}\boldsymbol{x}^t$

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

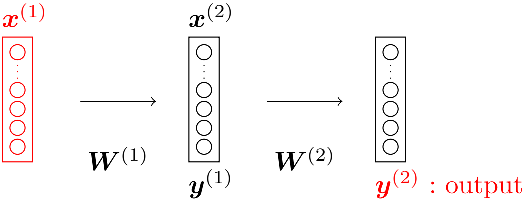
One layer, indexed by l



- $\boldsymbol{x}^{(l)}$: 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



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

Gradient for the output layer

As in the Ex. 1 :

$$\boldsymbol{y} \rightarrow \boldsymbol{y}^{(2)}$$

$$\boldsymbol{W} \rightarrow \boldsymbol{W}^{(2)}$$

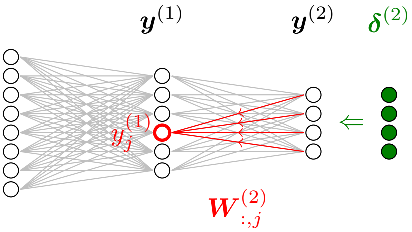
$$\boldsymbol{x} \rightarrow \boldsymbol{x}^{(2)} = \boldsymbol{y}^{(1)}$$

$$\nabla_{\boldsymbol{W}^{(2)}} = \boldsymbol{\delta}^{(2)} {\boldsymbol{x}^{(2)}}^{t}, \text{ with}$$

$$\delta_k^{(2)} = -\mathbb{I}\left\{k = c_{(i)}\right\} - y_k$$

Back-propagation of the loss gradient

For the hidden layer - 1



Inference chain part 1 :

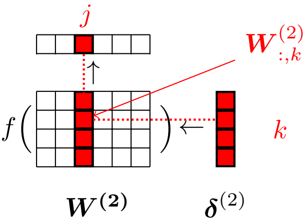
$$\boldsymbol{y}^{(1)} = f^{(1)}(\boldsymbol{a}^{(1)}) \rightarrow \left(\boldsymbol{a}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{y}^{(1)}\right) \rightarrow \left(\boldsymbol{y}^{(2)} = f^{(2)}(\boldsymbol{a}^{(2)})\right) \rightarrow l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})$$

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

$$= \sum_k (\mathbb{I}\{k = c_{(i)}\} - y_k^{(2)}) w_{kj}^{(2)} f'^{(1)}(a_j) = f'^{(1)}(a_j) \left(\boldsymbol{W}^{(2)}_{:,j} {\boldsymbol{\delta}^{(2)}}^t\right)$$

Back-propagation of the loss gradient

For the hidden layer - 2

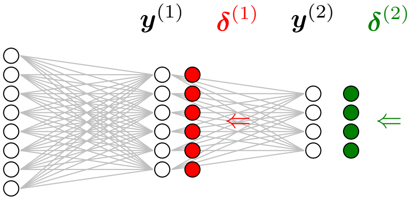


$$\nabla_{\boldsymbol{y}^{(1)}} = {\boldsymbol{W}^{(2)}}^t \boldsymbol{\delta}^{(2)}, \text{ then}$$

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

Back-propagation of the loss gradient

For the hidden layer - 3



As for the output layer, the gradient is :

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

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

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

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

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

P. Baldi and K. Hornik.
1989.
Neural networks and principal component analysis : Learning from examples without local minima.
Neural Networks, 2(1) :53–58, January.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle.
2007.
Greedy layer-wise training of deep networks.
In B. Schölkopf, J.C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press.

Léon Bottou.
2010.
Large-scale machine learning with stochastic gradient descent.
In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186. Physica-Verlag HD.

Dan C. Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber.
2012.
Multi-column deep neural network for traffic sign classification.
Neural Networks, 32 :333–338.

John Duchi, Elad Hazan, and Yoram Singer.
2011.

Adaptive subgradient methods for online learning and stochastic optimization.

J. Mach. Learn. Res., 12 :2121–2159, July.

Dj Field, 1993.

Scale-invariance and self-similar 'wavelet' transforms : an analysis of natural scenes and mammalian visual systems, pages 151–193.

Oxford University press.

Kunihiko Fukushima.

1980.

Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.

Biological Cybernetics, 36 :193–202.

Alex Graves and Juergen Schmidhuber.

2009.

Offline handwriting recognition with multidimensional recurrent neural networks.

In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc.

G. E. Hinton and R. R. Salakhutdinov.

2006.

Reducing the dimensionality of data with neural networks.

Science, 313(5786) :504–507, JUL 28.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh.
2006.
A fast learning algorithm for deep belief nets.
Neural Computation, 18(7) :1527–1554, JUL.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury.
2012.
Deep neural networks for acoustic modeling in speech recognition.
Signal Processing Magazine.

Sepp Hochreiter and Jürgen Schmidhuber.
1997.
Long short-term memory.
Neural Comput., 9(8) :1735–1780, November.

J. J. Hopfield.
1982.
Neural networks and physical systems with emergent collective computational abilities.
Proceedings of the National Academy of Sciences of the United States of America, 79(8) :2554–2558.

Teuvo Kohonen.

1982.
Self-organized formation of topologically correct feature maps.
Biological Cybernetics, 43(1) :59–69.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton.
2012.
Imagenet classification with deep convolutional neural networks.
In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel.
1989.
Backpropagation applied to handwritten zip code recognition.
Neural Comput., 1(4) :541–551, December.

Yann LeCun, Léon Bottou, Genevieve Orr, and Klaus-Robert Müller.
2012.
Efficient backprop.
In Grégoire Montavon, GenevièveB. Orr, and Klaus-Robert Müller, editors, *Neural Networks : Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer Berlin Heidelberg.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng.
2009.

Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.
In *Proceedings of the International Conference of Machine Learning (ICML)*, pages 609–616.

Erkki Oja.
1982.
Simplified neuron model as a principal component analyzer.
Journal of Mathematical Biology, 15(3) :267–273.

Ariadna Quattoni, Sybor Wang, Louis-Philippe Morency, Michael Collins, and Trevor Darrell.
2007.
Hidden conditional random fields.
IEEE Trans. Pattern Anal. Mach. Intell., 29(10) :1848–1852, October.

F. Rosenblatt.
1958.
The perceptron : A probabilistic model for information storage and organization in the brain.
Psychological Review, 65(6) :386–408.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams.
1986.
Learning representations by back-propagating errors.
Nature, 323(6088) :533–536, 10.