Statistical Machine Learning from Data

Other Artificial Neural Networks

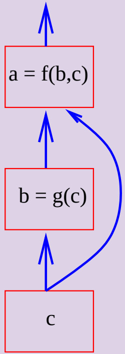
Samy Bengio

IDIAP Research Institute, Martigny, Switzerland, and
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
bengio@idiap.ch
http://www.idiap.ch/~bengio

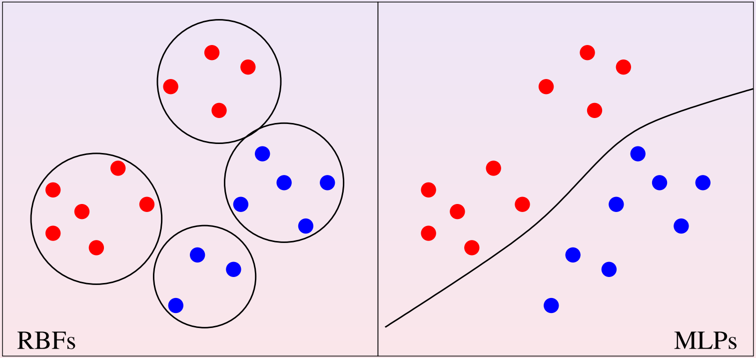
January 17, 2006

Generic Gradient Descent Mechanism

- if $a = f(b, c; \theta_f)$ is differentiable and $b = g(c; \theta_g)$ is differentiable
- then you should be able to compute the gradient with respect to $a, b, c, \theta_f, \theta_g \ldots$
- Hence only your imagination prevents you from inventing another neural network machine!



Difference Between RBFs and MLPs



Radial Basis Function (RBF) Models

- Normal MLP but the hidden layer l is encoded as follows:
  - $s_i^l = -\frac{1}{2} \sum_j (\gamma_{i,j}^l)^2 \cdot (y_j^{l-1} - \mu_{i,j}^l)^2$
  - $y_i^l = \exp(s_i^l)$
- The parameters of such layer l are $\theta_l = \{\gamma_{i,j}^l, \mu_{i,j}^l \quad : \quad \forall i, j\}$
- These layers are useful to extract local features (whereas tanh layers extract global features)

Gradient Descent for RBFs

- $s_i^l = -\frac{1}{2}\sum_j (\gamma_{i,j}^l)^2 \cdot (y_j^{l-1} - \mu_{i,j}^l)^2$
- $y_i^l = \exp(s_i^l)$

$\longrightarrow$

- $\frac{\partial y_i^l}{\partial s_i^l} = \exp(s_i^l) = y_i^l$
- $\frac{\partial s_i^l}{\partial y_j^{l-1}} = -(\gamma_{i,j}^l)^2 \cdot (y_j^{l-1} - \mu_{i,j}^l)$
- $\frac{\partial s_i^l}{\partial \mu_{i,j}^l} = (\gamma_{i,j}^l)^2 \cdot (y_j^{l-1} - \mu_{i,j}^l)$
- $\frac{\partial s_i^l}{\partial \gamma_{i,j}^l} = -\gamma_{i,j}^l \cdot (y_j^{l-1} - \mu_{i,j}^l)^2$

Warning with Variances

- Initialization: use K-Means for instance
- One has to be very careful with learning γ by gradient descent
- Remember:

$$y_i^l = \exp\left(-\frac{1}{2}\sum_j (\gamma_{i,j}^l)^2 \cdot (y_j^{l-1} - \mu_{i,j}^l)^2\right)$$

- If γ becomes too high, the RBF output can explode!
- One solution: constrain them in a reasonable range
- Otherwise, do not train them
  (keep the K-Means estimate for γ)

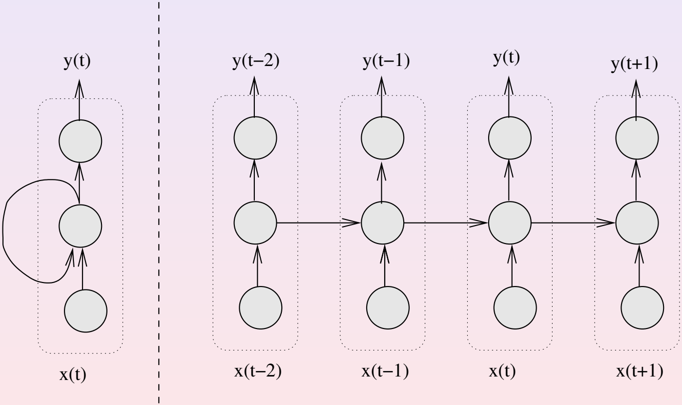
Recurrent Neural Networks

- Such models admit layers l with integration functions $s_i^l = f(y_j^{l+k})$ where $k \geq 0$, hence loops, or recurrences
- Such layers l encode the notion of a temporal state
- Useful to search for relations in temporal data
- Do not need to specify the exact delay in the relation
- In order to compute the gradient, one must enfold in time all the relations between the data:

$$s_i^l(t) = f(y_j^{l+k}(t-1)) \text{ where } k \geq 0$$

- Hence, need to exhibit the whole time-dependent graph between input sequence and output sequence
- Caveat: it can be shown that the gradient vanishes exponentially fast through time

Recurrent NNs (Graphical View)



Recurrent NNs - Example of Derivation

- Consider the following simple recurrent neural network:

$$h_t = \tanh(d \cdot h_{t-1} + w \cdot x_t + b)$$

$$\hat{y}_t = v \cdot h_t + c$$

with $\{d, w, b, v, c\}$ the set of parameters

- Cost to minimize (for one sequence):

$$C = \sum_{t=1}^{T} C_t = \sum_{t=1}^{T} \frac{1}{2}(y_t - \hat{y}_t)^2$$

Derivation of the Gradient

- We need to derive the following:

$$\frac{\partial C}{\partial d}, \frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}, \frac{\partial C}{\partial v}, \frac{\partial C}{\partial c}$$

- Let us do it for, say, $\frac{\partial C}{\partial w}$.

$$\begin{aligned}
\frac{\partial C}{\partial w} &= \sum_{t=1}^{T} \frac{\partial C_t}{\partial w} \\
&= \sum_{t=1}^{T} \frac{\partial C_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial w} \\
&= \sum_{t=1}^{T} \frac{\partial C_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial w}
\end{aligned}$$

Derivation of the Gradient (2)

$$\frac{\partial C}{\partial w} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial w} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \sum_{s=1}^{t} \frac{\partial h_t}{\partial h_s} \cdot \frac{\partial h_s}{\partial w}$$

$$\frac{\partial C_t}{\partial \hat{y}_t} = \hat{y}_t - y_t$$

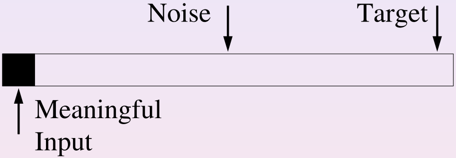
$$\frac{\partial \hat{y}_t}{\partial h_t} = v$$

$$\frac{\partial h_t}{\partial h_s} = \prod_{i=s+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=s+1}^{t} (1 - h_i^2) \cdot d$$

$$\frac{\partial h_s}{\partial w} = (1 - h_s^2) \cdot x_s$$

Long Term Dependencies

- Suppose we want to classify a sequence according to its first frame but the target is known at the end only:



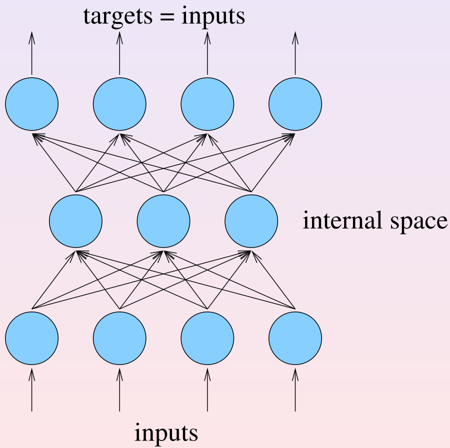
- Unfortunately, the gradient vanishes:

$$\frac{\partial C_T}{\partial w} = \sum_{t=1}^{T} \frac{\partial C_T}{\partial h_t} \frac{\partial h_t}{\partial w} \longrightarrow 0$$

- This is because for $t \ll T$

$$\frac{\partial C_T}{\partial h_t} = \frac{\partial C_T}{\partial h_T} \prod_{\tau=t+1}^{T} \frac{\partial h_\tau}{\partial h_{\tau-1}} \text{ and } \left| \frac{\partial h_\tau}{\partial h_{\tau-1}} \right| < 1$$

Auto Associative Nets (Graphical View)



Auto Associative Networks

- Apparent objective: learn to reconstruct the input
- In such models, the target vector is the same as the input vector!
- Real objective: learn an internal representation of the data
- If there is one hidden layer of linear units, then after learning, the model implements a principal component analysis with the first N principal components (N is the number of hidden units).
- If there are non-linearities and more hidden layers, then the system implements a kind of non-linear principal component analysis.

Mixture of Experts

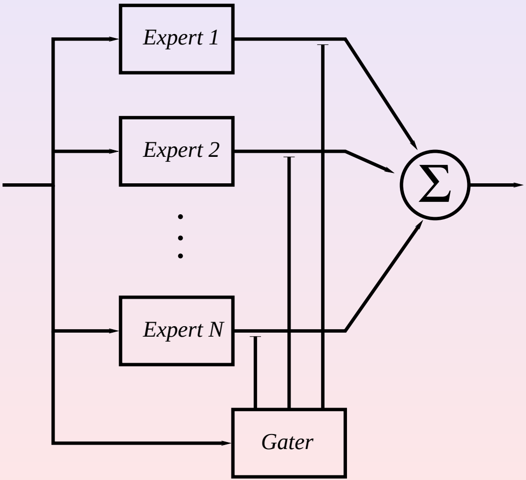
- Let $f_i(x; \theta_{f_i})$ be a differentiable parametric function
- Let there be N such functions f_i.
- Let $g(x; \theta_g)$ be a gater: a differentiable function with N positive outputs such that

$$\sum_{i=1}^{N} g(x; \theta_g)[i] = 1$$

- Then a mixture of experts is a function $h(x; \theta)$:

$$h(x; \theta) = \sum_{i=1}^{N} g(x; \theta_g)[i] \cdot f_i(x; \theta_{f_i})$$

Mixture of Experts - (Graphical View)



Mixture of Experts - Training

- We can compute the gradient with respect to every parameters:
    - parameters in the expert f_i:

$$\begin{aligned}\frac{\partial h(x;\theta)}{\partial \theta_{f_i}} &= \frac{\partial h(x;\theta)}{\partial f_i(x;\theta_{f_i})} \cdot \frac{\partial f_i(x;\theta_{f_i})}{\partial \theta_{f_i}} \\ &= g(x;\theta_g)[i] \cdot \frac{\partial f_i(x;\theta_{f_i})}{\partial \theta_{f_i}}\end{aligned}$$

    - parameters in the gater g:

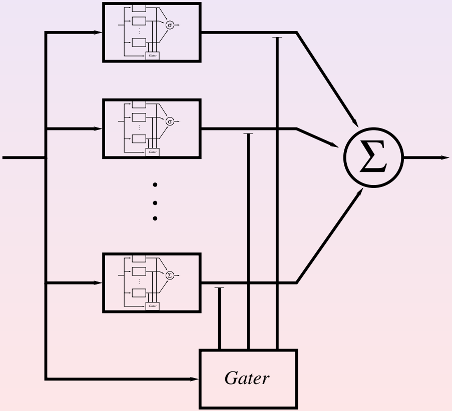
$$\begin{aligned}\frac{\partial h(x;\theta)}{\partial \theta_g} &= \sum_{i=1}^{N} \frac{\partial h(x;\theta)}{\partial g(x;\theta_g)[i]} \cdot \frac{\partial g(x;\theta_g)[i]}{\partial \theta_g} \\ &= \sum_{i=1}^{N} f_i(x;\theta_{f_i}) \cdot \frac{\partial g(x;\theta_g)[i]}{\partial \theta_g}\end{aligned}$$

Mixture of Experts - Discussion

- The gater implements a soft partition of the input space (to be compared with, say, K-Means $\rightarrow$ hard partition)
- Useful when there might be regimes in the data
- Special case: when the experts can be trained by EM, the mixture can also be trained by EM.

Hierarchical Mixture of Experts

When the experts are themselves represented as mixtures of experts:

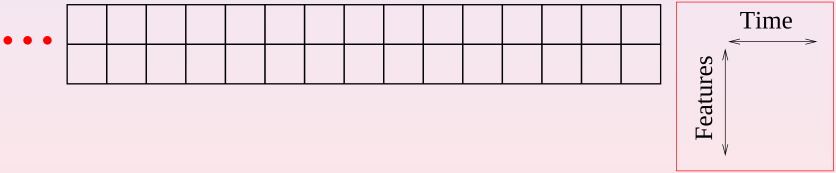


Time Delay Neural Networks (TDNNs)

- TDNNs are models to analyze sequences or time series.
- Hypothesis: some regularities exist over time.
- The same pattern can be seen many times during the same time series (or even over many times series).
- First idea: attribute one hidden unit to model each pattern
  - These hidden units should have associated parameters which are the same over time
  - Hence, the hidden unit associated to a given pattern p_i at time t will share the same parameters as the hidden unit associated to the same pattern p_i at time $t + k$.
- Note that we are also going to learn what are the patterns!

TDNNs: Convolutions

- How to formalize this first idea? using a convolution operator.
- This operator can be used not only between the input and the first hidden layer, but between any hidden layers.



Convolutions: Equations

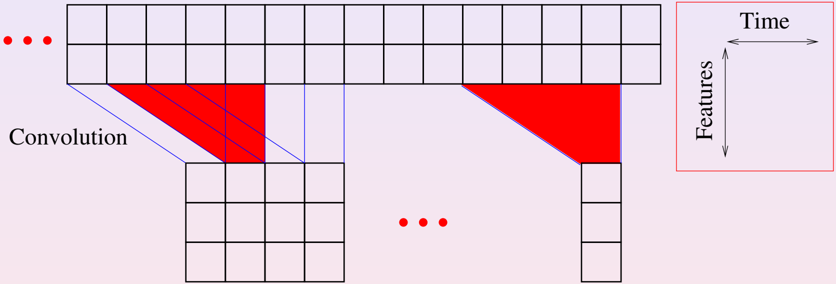
- Let $s_{t,i}^{l}$ be the input value at time t of unit i of layer l.
  Let $y_{t,i}^{l}$ be the output value at time t of unit i of layer l.
  (inputs values: $y_{t,i}^{0} = x_{t,i}$).
  Let $w_{i,j,k}^{l}$ be the weight between unit i of layer l at any time t and unit j of layer l at time $t-k$.
  Let b_{i}^{l} be the bias of unit i at layer l.
- Convolution operator for windows of size K:

$$s_{t,i}^{l} = \sum_{k=0}^{K-1} \sum_{j} w_{i,j,k}^{l} \cdot y_{t-k,j}^{l-1} + b_{i}^{l}$$

- Transfer:

$$y_{t,i}^{l} = \tanh(s_{t,i}^{l})$$

Convolutions (Graphical View)



- Note: weights $w^l_{i,j,k}$ and biases b^l_i do not depend on time.
- Hence the number of parameters of such model is independent of the length of the time series.
- Each unit $s^l_{t,i}$ represents the value of the same function at each time step.

TDNNs: Subsampling

- The convolution functions always work with a fixed size window (K in our case, which can be different for each unit/layer).
- Some regularities might exist at different granularities.
- Hence, second idea: subsampling (it is more a kind of smoothing operator in fact).
  - In between each convolution layer, let us add a subsampling layer.
  - This subsampling layer provides a way to analyze the time series at a coarser level.

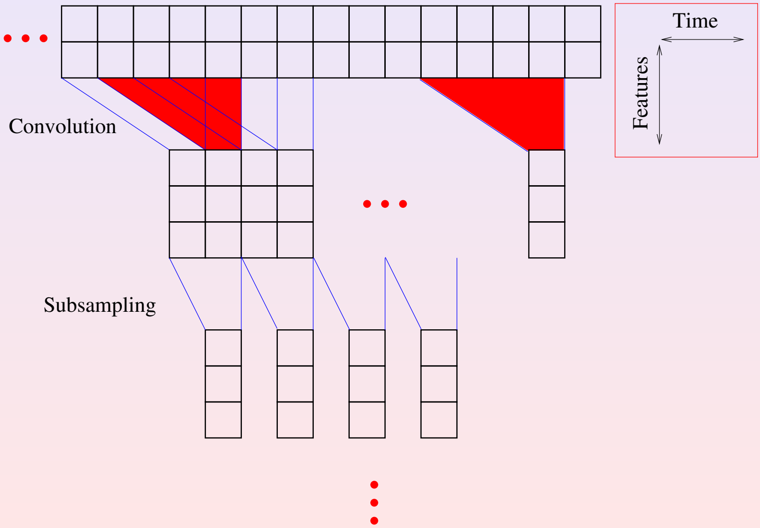
Subsampling: Equations

- How to formalize this second idea?
- Let $y_{t,i}^l$ be the output value at time t of unit i of layer l. (inputs values: $y_{t,i}^0 = x_{t,i}$).
- Let r be the ratio of subsampling. This is often set to values such as 2 to 4.
- Subsampling operator:

$$y_{t,i}^l = \frac{1}{r} \sum_{k=0}^{r-1} y_{rt-k,i}^{l-1}$$

- Only compute values $y_{t,i}^l$ such that $(t \mod r) = 0$.
- Note: there are no parameter in the subsampling layer (but it is possible to add some, replacing for intance $\frac{1}{r}$ by a parameter and adding a bias term).

TDNNs (Graphical View)



Learning in TDNNs

- TDNNs can be trained by normal gradient descent techniques.
- Note that, as for MLPs, each layer is a differentiable function.
- We just need to compute the local gradient:
- Convolution layers:

$$\frac{\partial C}{\partial w^l_{i,j,k}} = \sum_t \frac{\partial C}{\partial s^l_{t,i}} \cdot \frac{\partial s^l_{t,i}}{\partial w^l_{i,j,k}} = \sum_t \frac{\partial C}{\partial s^l_{t,i}} \cdot y^{l-1}_{t-k,j}$$

- Subsampling layers:

$$\frac{\partial C}{\partial y^{l-1}_{t-k,i}} = \frac{\partial C}{\partial y^l_{t,i}} \cdot \frac{\partial y^l_{t,i}}{\partial y^{l-1}_{rt-k,i}} = \frac{\partial C}{\partial y^l_{t,i}} \cdot \frac{1}{r}$$

LeNet for Images

- TDNNs are useful to handle regularities of time series ($\rightarrow$ 1D data)
- Could we use the same trick for images ($\rightarrow$ 2D data)?
- After all, regularities are often visible on images.
- It has indeed been proposed as well, under the name LeNet.

LeNet (Graphical View)