Machine Learning

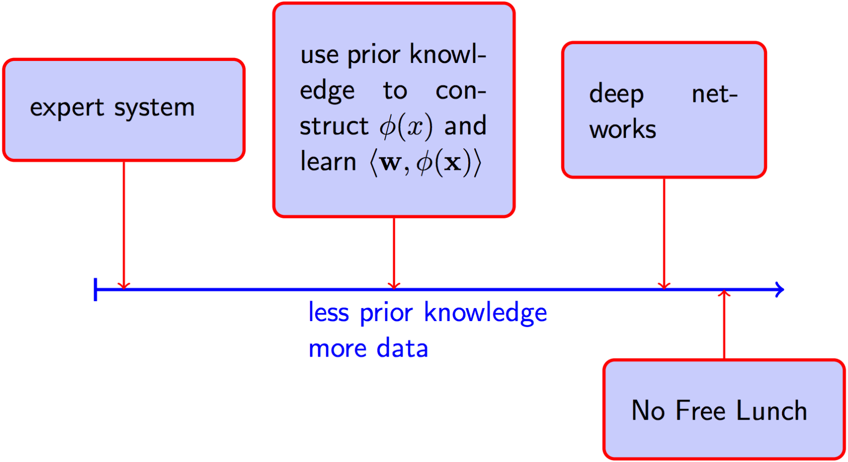
Neural Networks

11 Dec.: NN ; VC dimension
15 Dec.: VC dimension ; clustering
18 Dec : LAB (clustering)
22 Dec : Deep learning

Fabio Vandin December 11^{th}, 2023

8 Jan : LAB (NN ; other topics)
12 Jan : Additional topics ; exercises
15 Jan : Exercises – Q&A

An Extremely Powerful Hypothesis Class...



Runtime of Learning NNs

Informally: applying the ERM rule with respect to $\mathcal{H}_{V,E,\text{sign}}$ is *computationally difficult*, even for small NN...

Proposition

Let $k \geq 3$. For every d, let (V, E) be a layered graph with d input nodes, $k+1$ nodes at the (only) hidden layer, where one of them is the constant neuron, and a single output node. Then, it is NP-hard to implement the ERM rule with respect to $\mathcal{H}_{V,E,\text{sign}}$.

Well maybe the above is only for very specific cases...

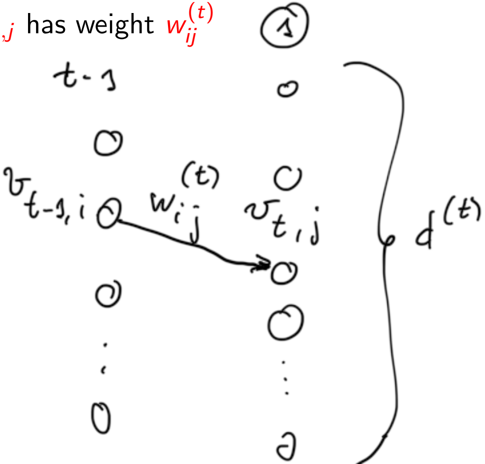
- instead of ERM rule, find h close to ERM? **Computationally infeasible!** (probably)
- other activation functions (e.g., sigmoid)? **Computationally infeasible!** (probably)
- smart embedding in larger network? **Computationally infeasible!** (probably)

So? *Heuristic* for training NNs $\Rightarrow$ SGD algorithm and its improved versions are used: gives good results in practice!

Matrix Notation

Consider layer t, $0 < t < T$:

- let $d^{(t)} + 1$ the number of nodes:
  - constant node 1
  - values of nodes for (hidden) variables: $v_{t,1}, \ldots, v_{t,d^{(t)}}$
- arc from $v_{t-1,i}$ to $v_{t,j}$ has weight $w_{ij}^{(t)}$



Matrix Notation

Consider layer t, $0 < t < T$:

- let $d^{(t)} + 1$ the number of nodes:
  - constant node 1
  - values of nodes for (hidden) variables: $v_{t,1}, \ldots, v_{t,d^{(t)}}$
- arc from $v_{t-1,i}$ to $v_{t,j}$ has weight $w_{ij}^{(t)}$

Let

$$\mathbf{v}^{(t)} = \left(1, v_{t,1}, \ldots, v_{t,d^{(t)}}\right)^T$$

$$\mathbf{w}_j^{(t)} = \left(w_{0j}^{(t)}, w_{1j}^{(t)}, \ldots, w_{d^{(t-1)}j}^{(t)}\right)^T$$

Then

$$v_{t,j} = \sigma\left(\langle \mathbf{w}_j^{(t)}, \mathbf{v}^{(t-1)} \rangle\right)$$

Note:

$$\mathbf{v}^{(t)} = \begin{bmatrix} 1 \\ v_{t,1} \\ \vdots \\ v_{t,d^{(t)}} \end{bmatrix} = \begin{bmatrix} 1 \\ \sigma\left(\langle \mathbf{w}_1^{(t)}, \mathbf{v}^{(t-1)} \rangle\right) \\ \vdots \\ \sigma\left(\langle \mathbf{w}_{d^{(t)}}^{(t)}, \mathbf{v}^{(t-1)} \rangle\right) \end{bmatrix}$$

Let

$$a_{t,j} := \langle \mathbf{w}_j^{(t)}, \mathbf{v}^{(t-1)} \rangle$$

and

$$\mathbf{a}^{(t)} = \begin{bmatrix} a_{t,1} \\ \vdots \\ a_{t,d^{(t)}} \end{bmatrix} \qquad \sigma\left(\mathbf{a}^{(t)}\right) = \begin{bmatrix} \sigma\left(a_{t,1}\right) \\ \vdots \\ \sigma\left(a_{t,d^{(t)}}\right) \end{bmatrix}$$

Then

$$\mathbf{v}^{(t)} = \begin{bmatrix} 1 \\ \sigma\left(\mathbf{a}^{(t)}\right) \end{bmatrix}$$

weights incoming edges for $v_{t,1}$

weights incoming edges for $v_{t,2}$

Let

$$\mathbf{w}^{(t)} = \begin{bmatrix} w_{01}^{(t)} & w_{02}^{(t)} & \cdots & w_{0d^{(t)}}^{(t)} \\ w_{11}^{(t)} & w_{12}^{(t)} & \cdots & w_{1d^{(t)}}^{(t)} \\ \vdots & \vdots & \cdots & \vdots \\ w_{d^{(t-1)}1}^{(t)} & w_{d^{(t-1)}2}^{(t)} & \cdots & w_{d^{(t-1)}d^{(t)}}^{(t)} \end{bmatrix}$$

($\mathbf{w}^{(t)}$ describes the weights of edges from layer $t-1$ to layer t)

Let

$$\mathbf{w}^{(t)} = \begin{bmatrix} w_{01}^{(t)} & w_{02}^{(t)} & \cdots & w_{0d^{(t)}}^{(t)} \\ w_{11}^{(t)} & w_{12}^{(t)} & \cdots & w_{1d^{(t)}}^{(t)} \\ \vdots & \vdots & \cdots & \vdots \\ w_{d^{(t-1)}1}^{(t)} & w_{d^{(t-1)}2}^{(t)} & \cdots & w_{d^{(t-1)}d^{(t)}}^{(t)} \end{bmatrix}$$

($\mathbf{w}^{(t)}$ describes the weights of edges from layer $t-1$ to layer t)

Then

$$\mathbf{a}^{(t)} = \left(\mathbf{w}^{(t)}\right)^T \mathbf{v}^{(t-1)}$$

Using Matrix Notation Warm-Up: Forward Propagation Algorithm

Input: $\mathbf{x} = (x_1, \ldots, x_d)^T$; NN with 1 output node
Output: prediction y of NN;

$\mathbf{v}^{(0)} \leftarrow (1, x_1, \ldots, x_d)^T$;
for $t \leftarrow 1$ **to** T **do**
  $\mathbf{a}^{(t)} \leftarrow \left(\mathbf{w}^{(t)}\right)^T \mathbf{v}^{(t-1)}$;
  $\mathbf{v}^{(t)} \leftarrow \left(1, \sigma\left(\mathbf{a}^{(t)}\right)^T\right)^T$;
$y \leftarrow \sigma\left(\mathbf{a}^{(T)}\right)$;
return y;

Learning NN parameters

How do we compute the weights $w_{ij}^{(t)}$?

ERM: given training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ pick $w_{ij}^{(t)}, \forall i, j, t$ (defining a specific model h) minimizing the training error:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, (\mathbf{x}_i, y_i))$$

How?

Not easy!

Learning NN parameters (2)

We use GD seeing $L_S(h)$ as a function of $\mathbf{w}^{(t)}, \forall 1 \leq t \leq T$:

GD Update rule:

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla L_S(\mathbf{w}^{(t)})$$

where $\nabla L_S(\mathbf{w}^{(t)})$ is the gradient of L_S (and η is the learning parameter). To compute it we need $\forall t, 1 \leq t \leq T$:

$$\frac{\partial L_S}{\partial \mathbf{w}^{(t)}} = \frac{\partial}{\partial \mathbf{w}^{(t)}} \left(\frac{1}{m} \sum_{i=1}^{m} \ell(h, (\mathbf{x}_i, y_i)) \right) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \ell(h, (\mathbf{x}_i, y_i))}{\partial \mathbf{w}^{(t)}}$$

$\Rightarrow$ need $\frac{\partial \ell}{\partial \mathbf{w}^{(t)}}$

Learning NN parameters (3)

Definition: Sensitivity vector for layer t

$$\delta^{(t)} = \frac{\partial \ell}{\partial \mathbf{a}^{(t)}} = \begin{bmatrix} \frac{\partial \ell}{\partial a_{t,1}} \\ \vdots \\ \frac{\partial \ell}{\partial a_{t,d^{(t)}}} \end{bmatrix} = \begin{bmatrix} \delta_1^{(t)} \\ \vdots \\ \delta_{d^{(t)}}^{(t)} \end{bmatrix}$$

Learning NN parameters (3)

Definition: Sensitivity vector for layer t

$$\delta^{(t)} = \frac{\partial \ell}{\partial \mathbf{a}^{(t)}} = \begin{bmatrix} \frac{\partial \ell}{\partial a_{t,1}} \\ \vdots \\ \frac{\partial \ell}{\partial a_{t,d^{(t)}}} \end{bmatrix} = \begin{bmatrix} \delta_1^{(t)} \\ \vdots \\ \delta_{d^{(t)}}^{(t)} \end{bmatrix}$$

$\delta^{(t)}$ quantifies how the training error changes with $\mathbf{a}^{(t)}$ (the inputs to the t layer - before the nonlinear transformation)

Learning NN parameters (4)

Consider a weight $w_{ij}^{(t)}$: a change in $w_{ij}^{(t)}$ changes only $a_{t,j}$ therefore by chain rule we have

$$
\begin{aligned}
\frac{\partial \ell}{\partial w_{ij}^{(t)}} &= \frac{\partial \ell}{\partial a_{t,j}} \cdot \frac{\partial a_{t,j}}{\partial w_{ij}^{(t)}} \\
&= \delta_j^{(t)} \cdot \frac{\partial}{\partial w_{ij}^{(t)}} \left(\sum_{k=0}^{d^{(t-1)}} w_{kj}^{(t)} v_{t-1,k} \right) \\
&= \delta_j^{(t)} \cdot v_{t-1,i}
\end{aligned}
$$

compute the prediction for the given input $\vec{x}, y$

Learning NN parameters (4)

Consider a weight $w_{ij}^{(t)}$: a change in $w_{ij}^{(t)}$ changes only $a_{t,j}$ therefore by chain rule we have

$$
\begin{aligned}
\frac{\partial \ell}{\partial w_{ij}^{(t)}} &= \frac{\partial \ell}{\partial a_{t,j}} \cdot \frac{\partial a_{t,j}}{\partial w_{ij}^{(t)}} \\
&= \delta_j^{(t)} \cdot \frac{\partial}{\partial w_{ij}^{(t)}} \left(\sum_{k=0}^{d^{(t-1)}} w_{kj}^{(t)} v_{t-1,k} \right) \\
&= \delta_j^{(t)} \cdot v_{t-1,i}
\end{aligned}
$$

fixed by the input

Therefore to compute the gradient we only need $\delta^{(t)} = \frac{\partial \ell}{\partial \mathbf{a}^{(t)}} \quad \forall t$.
How can we compute it?

Learning NN parameters (5)

Since ℓ depends from $a_{t,j}$ only through $v_{t,j}$, then from chain rule:

$$\begin{aligned}\delta_j^{(t)} &= \frac{\partial \ell}{\partial a_{t,j}} \\ &= \frac{\partial \ell}{\partial v_{t,j}} \cdot \frac{\partial v_{t,j}}{\partial a_{t,j}} \\ &= \frac{\partial \ell}{\partial v_{t,j}} \cdot \sigma'(a_{t,j})\end{aligned}$$

$v_{t,j} = \sigma(a_{t,j})$

(the last equality derives from the definition of $v_{t,j}$)

Learning NN parameters (6)

Consider $\frac{\partial \ell}{\partial v_{t,j}}$: we need to understand how loss ℓ changes due to changes in $v_{t,j}$

- change in $\mathbf{v}^{(t)}$ affects only $\mathbf{a}^{(t+1)}$ (and then ℓ)
- changes in $v_{t,j}$ can affect every $a_{t+1,k}$

$\Rightarrow$ sum chain rule contributions

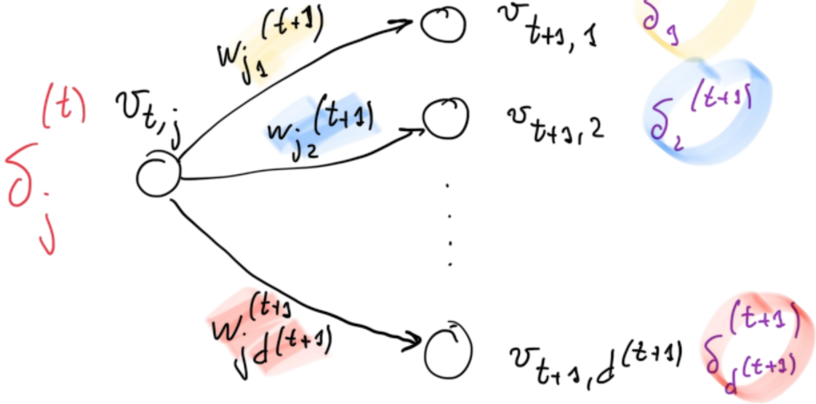
Then

$$\frac{\partial \ell}{\partial v_{t,j}} = \sum_{k=1}^{d^{(t+1)}} \frac{\partial a_{t+1,k}}{\partial v_{t,j}} \cdot \frac{\partial \ell}{\partial a_{t+1,k}}$$

$$= \sum_{k=1}^{d^{(t+1)}} w_{jk}^{(t+1)} \cdot \delta_k^{(t+1)}$$

Putting everything together:

$$\delta_j^{(t)} = \sigma'(a_{t,j}) \cdot \sum_{k=1}^{d^{(t+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)}$$



Learning NN parameters (7)

Putting everything together:

$$\delta_j^{(t)} = \sigma'(a_{t,j}) \cdot \sum_{k=1}^{d^{(t+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)}$$

Notes:

- $\sigma'(a_{t,j})$ depends on the function σ chosen
- To compute $\delta_j^{(t)}$ need $\delta_k^{(t+1)}$, $1 \leq k \leq d^{(t+1)}$
  $\Rightarrow$ *backpropagation algorithm*
- To start: need $\delta^{(L)} = \frac{\partial \ell}{\partial \mathbf{a}^{(L)}}$ (sensitivity of final layer): depends on the loss ℓ used

Algorithm to compute sensitivities $\delta^{(t)}, \forall t$, for a given data point $(\mathbf{x}_i, y_i)$.

Input: data point $(\mathbf{x}_i, y_i)$, NN (with weights $w_{ij}^{(t)}$, for $1 \leq t \leq T$)
Output: $\delta^{(t)}$ for $t = 1, \ldots, T$
compute $\mathbf{a}^{(t)}$ and $\mathbf{v}^{(t)}$ for $t = 1, \ldots T$; // forward propagation
$\delta^{(T)} \leftarrow \frac{\partial \ell}{\partial a^{(T)}}$;
for $t = T - 1$ **downto** 1 **do**
  $\delta_j^{(t)} \leftarrow \sigma'(a_{t,j}) \cdot \sum_{k=1}^{d^{(\ell+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)}$ for all $j = 1, \ldots, d^{(t)}$;
return $\delta^{(1)}, \ldots, \delta^{(T)}$;

Backpropagation Algorithm

This is the final backpropagation algorithm, based on SGD, to train a NN

Input: training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, NN (no weights $w_{ij}^{(t)}$)

Output: NN with weights $w_{ij}^{(t)}$

initialize $w_{ij}^{(t)}$ for all i, j, t;

for $s \leftarrow 0, 1, 2, \ldots$ **do** `/* until convergence */`

  pick $(\mathbf{x}_k, y_k)$ at random from training data;

  `/* forward propagation */`

  compute $v_{t,j}$ for all j, t from $(\mathbf{x}_k, y_k)$;

  `/* backward propagation */`

  compute $\delta_j^{(t)}$ for all j, t from $(\mathbf{x}_k, y_k)$;

  $w_{ij}^{(t)} \leftarrow w_{ij}^{(t)} - \eta v_{t-1,i} \delta_j^{(t)}$ for all i, j, t; `/* update weights */`

  if *converged* **then return** $w_{ij}^{(t)}$ for all i, j, t;

Notes on Backpropagation Algorithm

- preprocessing: all inputs are normalized and centered
- initialization of $w_{ij}^{(t)}$?
  Random values around 0 - regime where model is $\approx$ linear
  - $w_{ij}^{(t)} \sim U(-0.7, 0.7)$ (uniform distribution)
  - $w_{ij}^{(t)} \sim N(0, \sigma^2)$ with small σ^2
  - if all weights set to 0 $\Rightarrow$ all neurons get the same weights
- when to stop?
  Usually combination of:
  - "small" (training) error;
  - "small" marginal improvement in error;
  - upper bound on number of iterations
- $L_S(h)$ usually has multiple local minima
  $\Rightarrow$ run stochastic gradient descent for different (random) initial weights

Regularized NN

Instead of training a NN by minimizing $L_S(h)$, find h that minimizes:

$$L_S(h) + \frac{\lambda}{2} \sum_{i,j,t} (w_{ij}^{(t)})^2$$

where $\lambda =$ *regularization parameter*

How do we find h? SGD or improved algorithms.
Note: for layer t, gradient is $\nabla(L_S(h)) + \lambda \mathbf{w}^{(t)}$

compute it with back propagation

Regularized NN

Instead of training a NN by minimizing $L_S(h)$, find h that minimizes:

$$L_S(h) + \frac{\lambda}{2} \sum_{i,j,t} (w_{ij}^{(t)})^2$$

where $\lambda =$ *regularization parameter*

How do we find h? SGD or improved algorithms.
Note: for layer t, gradient is $\nabla(L_S(h)) + \lambda \mathbf{w}^{(t)}$

This is called *squared weight decay regularizer*

Other regularizations are possible.