6.1 Training Neural Networks

Machine Learning 1: Foundations

Marius Kloft *(TUK)*

Recap

Artificial neural networks (ANN)

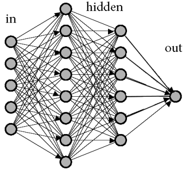
- Key advantage over SVM, logistic regression, and friends: can **learn a good representation** of the data,

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d,\phi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ln\Big(1+\exp(-y_i(\mathbf{w}^\top\phi\big(\mathbf{x}_i\big)+b))\Big).$$

Need to restrict search space of ϕ!

Idea: design ϕ similar to our brain

- multiple neurons in multiple layers with feed-forward connections
- $\phi_W(\mathbf{x}_i) := \sigma\Big(W_L^\top \ldots \sigma(W_1^\top \cdot \mathbf{x}_i)\ldots\Big)$
- optimize over $W = \big(W_1,\ldots,W_L\big)$!



How to train ANNs?

Contents of this Class

1. Training Neural Networks
2. Deep Learning

How to Train (Deep) ANNs?

In the same way as we trained the SVM:
(stochastic) gradient descent!

Recall the ANN optimization problem:

$$\min_{\mathbf{w},W} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2 + C\sum_{i=1}^{n}\ln\Big(1+\exp\big(-y_i\mathbf{w}^\top\phi_W(\mathbf{x}_i)\big)\Big)}_{=:F(\mathbf{w},W)}$$

How to compute the gradient of F?

For the sake of simplicity, we focus on discussing how to train *fully connected ANNs* (not CNNs).

The Gradient of F With Respect to $\boxed{\mathbf{w}}$ is Simple:

The function

$$g(x) = \ln(1 + \exp(x))$$

has the derivative

$$g'(x) = \frac{\exp(x)}{1 + \exp(x)} = \frac{1}{1 + \exp(-x)}.$$

Thus, by the chain rule:

$$\begin{aligned}
\nabla_{\mathbf{w}} F(\mathbf{w}, W) &= \mathbf{w} + C \sum_{i=1}^{n} \nabla_{\mathbf{w}} \ln\Big(1 + \exp\left(-y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i)\right)\Big) \\
&= \mathbf{w} - C \sum_{i=1}^{n} \frac{y_i \phi_W(\mathbf{x}_i)}{1 + \exp(y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i))}
\end{aligned}$$

But how to compute the gradient of F with respect to $\boxed{W}$?

Gradient of F With Respect to $\boxed{W = (W_1, \ldots, W_L)}$

Analogously, we have, for all $l = 1, \ldots, L$:

$$\begin{aligned}\nabla_{W_l} F(\mathbf{w}, W) &= W_l + C \sum_{i=1}^{n} \nabla_{W_l} \ln\Big(1 + \exp\big(-y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i)\big)\Big) \\ &= W_l - C \sum_{i=1}^{n} \frac{y_i \mathbf{w}^\top \boldsymbol{\nabla}_{\boldsymbol{W_l}} \boldsymbol{\phi}_{\boldsymbol{W}}(\mathbf{x}_{\boldsymbol{i}})}{1 + \exp(y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i))},\end{aligned}$$

where we applied the chain rule.

From now on, denote the ijth entry of W_l by w_{ijl}.

Given a data point $\mathbf{x}$, how to compute $\boxed{\nabla_{w_{ijl}} \phi_W(\mathbf{x})}$?

Computing $\nabla_{w_{ijl}} \phi_W(\mathbf{x})$

We have:

$$\nabla_{w_{ijl}} \phi_W(\mathbf{x}) = \nabla_{w_{ijl}} \underbrace{\sigma\Big(\underbrace{W_L^\top \sigma\big(\ldots \underbrace{\sigma(\underbrace{W_1^\top \mathbf{v}_0}_{=\mathbf{u}_1})}_{=\mathbf{v}_1} \ldots\big)}_{\mathbf{u}_L} \Big)}_{\mathbf{v}_L}.$$

Need to compute a gradient of a nested function!

Idea: Chain rule

$$\nabla_{w_{ijl}} \phi_W(\mathbf{x}) = \frac{\partial \mathbf{v}_L}{\partial w_{ijl}} = \underset{①}{\frac{\partial \mathbf{v}_L}{\partial \mathbf{u}_L}} \cdot \underset{②}{\frac{\partial \mathbf{u}_L}{\partial \mathbf{v}_{L-1}}} \cdot \underset{①}{\frac{\partial \mathbf{v}_{L-1}}{\partial \mathbf{u}_{L-1}}} \cdots \underset{②}{\frac{\partial \mathbf{u}_{l+1}}{\partial \mathbf{v}_l}} \cdot \underset{①}{\frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l}} \cdot \underset{③}{\frac{\partial \mathbf{u}_l}{\partial w_{ijl}}}$$

Three Terms Occur by the Chain Rule:

For all $l = 1, \ldots, L$:

1. $\frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l}$
2. $\frac{\partial \mathbf{u}_l}{\partial \mathbf{v}_{l-1}}$
3. $\frac{\partial \mathbf{u}_l}{\partial w_{ijl}}$

We need to compute all of them!

First Term

We compute the first term as:

$$① \qquad \frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l} = \frac{\partial \sigma(\mathbf{u}_l)}{\partial \mathbf{u}_l} = \frac{\partial \max(0, \mathbf{u}_l)}{\partial \mathbf{u}_l} = \Theta(\mathbf{u}_l),$$

where

$$\Theta : \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R} \\ x & \mapsto & \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases} \end{array}$$

is the **heavyside function**, which, for a vector $\mathbf{x} = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$, is defined elementwise:

$$\Theta(\mathbf{x}) := \begin{pmatrix} \Theta(x_1) \\ \vdots \\ \Theta(x_d) \end{pmatrix}.$$

Second Term

We compute the second term as:

$$\textcircled{2} \qquad \frac{\partial \mathbf{u}_l}{\partial \mathbf{v}_{l-1}} = \frac{\partial (W_l^\top \mathbf{v}_{l-1})}{\partial \mathbf{v}_{l-1}} = W_l^\top$$

Third Term

We compute the third term as:

$$③ \qquad \frac{\partial \mathbf{u}_l}{\partial w_{ijl}} = \frac{\partial (W_l^\top \mathbf{v}_{l-1})}{\partial w_{ijl}} = \left(\frac{\partial \sum_k w_{kk'l} v_{k,l-1}}{\partial w_{ijl}} \right)_{k'} = v_{i,l-1} \mathbf{e}_j$$

where

- $v_{i,l-1}$ denotes the ith entry of $\mathbf{v}_{l-1}$
- $\mathbf{e}_j$ is a unit vector with entries zero everywhere except in the jth component.

Putting Things Together

Our chain rule formula from Slide 7 thus translates into:

$$
\begin{aligned}
\nabla_{w_{ijl}} \phi_W(\mathbf{x}) &= \frac{\partial \mathbf{v}_L}{\partial \mathbf{u}_L} \cdot \frac{\partial \mathbf{u}_L}{\partial \mathbf{v}_{L-1}} \cdot \frac{\partial \mathbf{v}_{L-1}}{\partial \mathbf{u}_{L-1}} \cdots \frac{\partial \mathbf{u}_{l+1}}{\partial \mathbf{v}_l} \cdot \frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l} \cdot \frac{\partial \mathbf{u}_l}{\partial w_{ijl}} \\
&= \Theta(\mathbf{u}_L) W_L^\top \Theta(\mathbf{u}_{L-1}) \cdots W_{l+1}^\top \Theta(\mathbf{u}_l) v_{i,l-1} \mathbf{e}_j
\end{aligned}
$$

How to code up the computation of

$$\nabla_{w_{ijl}} \phi_W(\mathbf{x}) \qquad \forall i, j, l$$

in an efficient algorithm?

Backpropagation Algorithm

Given an input $\mathbf{x}$, we first compute all variables $\mathbf{u}_l$ and $\mathbf{v}_l$:

Forward propagation

initialize $\mathbf{v}_0 := \mathbf{x}$
for $l = 1 : (L-1)$ **do**
$\quad \mathbf{u}_l \;:=\; W_l^\top \mathbf{v}_{l-1}$
$\quad \mathbf{v}_l \;:=\; \sigma(\mathbf{u}_l)$
end for

Then, we compute the gradient via the chain rule:

Backward propagation

initialize $\boldsymbol{\delta}_L \;:=\; \Theta(\mathbf{u}_L)$
$\nabla_{w_{ijL}} \phi_W(\mathbf{x}) \;:=\; \boldsymbol{\delta}_L v_{i,L-1} \mathbf{e}_j \qquad \forall i,j$
for $l = (L-1) : 1$ **do**
$\quad \boldsymbol{\delta}_l \;:=\; \boldsymbol{\delta}_{l+1} W_{l+1}^\top \Theta(\mathbf{u}_l)$
$\quad \nabla_{w_{ijl}} \phi_W(\mathbf{x}) \;:=\; \boldsymbol{\delta}_l v_{i,l-1} \mathbf{e}_j \qquad \forall i,j$
end for

Conclusion

How to train ANNs?

- Stochastic gradient descent

How to compute gradient?

- ANN is a nested function
- Thus we compute the gradient via the chain rule
- Lead to a recursive algorithm: backpropagation

Outlook

Advanced training algorithms:

- Adagrad
- Adam
- Nesterov momentum