# 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,

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\log\Big(0, 1+\exp(-y_i(\mathbf{w}^\top \quad \mathbf{x}_i \ +b))\Big).$$

# Recap

Artificial neural networks (ANN)

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

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \log\Big(0, 1+\exp(-y_i(\mathbf{w}^\top \phi\big(\mathbf{x}_i\big)+b))\Big).$$

# Recap

Artificial neural networks (ANN)

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

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d,\phi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \log\Big(0, 1+\exp(-y_i(\mathbf{w}^\top \phi\big(\mathbf{x}_i\big)+b))\Big).$$

# Recap

Artificial neural networks (ANN)

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

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d,\phi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \log\Big(0, 1+\exp(-y_i(\mathbf{w}^\top\phi\big(\mathbf{x}_i\big)+b))\Big).$$

Need to restrict search space of $\phi$!

# Recap

Artificial neural networks (ANN)

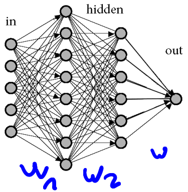
- Key advantage over SVM, logistic regression, and friends: can **learn a good representation** of the data,

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d,\phi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\log\Big(0, 1+\exp(-y_i(\mathbf{w}^\top\phi\big(\mathbf{x}_i\big)+b))\Big).$$

**Need to restrict search space of $\phi$!**

Idea: design $\phi$ similar to our brain

- multiple neurons in multiple layers with feed-forward connections
- $\phi_W(\mathbf{x}_i) := \sigma\Big(W_{L-1}^\top \dots \sigma(W_1^\top \cdot \mathbf{x}_i)\dots\Big)$
- optimize over $W = (W_1, \dots, W_L)$!



# Recap

Artificial neural networks (ANN)

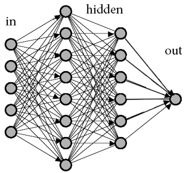
- Key advantage over SVM, logistic regression, and friends: can **learn a good representation** of the data,

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d,\phi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\log\Big(0, 1+\exp(-y_i(\mathbf{w}^\top\phi\big(\mathbf{x}_i\big)+b))\Big).$$

Need to restrict search space of $\phi$!

Idea: design $\phi$ similar to our brain

- multiple neurons in multiple layers with feed-forward connections
- $\phi_W(\mathbf{x}_i) := \sigma\Big(W_{L-1}^\top \dots \sigma(W_1^\top \cdot \mathbf{x}_i)\dots\Big)$
- optimize over $W = (W_1, \dots, W_L)$!




How to train ANNs?

# Contents of this Class

# How to Train (Deep) ANNs?

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

# How to Train (Deep) ANNs?

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

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

# How to Train (Deep) ANNs?

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

Recall the ANN optimization problem:

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

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

# How to Train (Deep) ANNs?

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

Recall the ANN optimization problem:

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

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:

$$\nabla_{\mathbf{w}} F(\mathbf{w}, W) \;=\; \mathbf{w} + C \sum_{i=1}^{n} \nabla_{\mathbf{w}} \log\Big(1 + \exp\big(-y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i)\big)\Big)$$

$$\overset{(*)}{=} 1 + C \sum_{i=1}^{n} \frac{-y_i \phi_w(x_i)}{1+\exp(y_i w^T \phi_w(x_i))}$$

$$= 1 - C \sum_{i=1}^{n} \frac{y_i \phi_w(x_i)}{1+\exp(y_i w^T \phi_w(x_i))}$$

$(*)$ $\log(1+\exp(t))'$

$$= \frac{\exp(t)}{1+\exp(t)} = \frac{1}{1+\exp(-t)}$$

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

$$\nabla_{\mathbf{w}} F(\mathbf{w}, W) \;=\; \mathbf{w} + C \sum_{i=1}^{n} \nabla_{\mathbf{w}} \log\Big(1 + \exp\big(-y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i)\big)\Big)$$

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 $l = 1, \ldots, L$:

$$\nabla_{W_l} F(\mathbf{w}, W) \;=\; W_l + C \sum_{i=1}^{n} \nabla_{W_l} \log \Big( 1 + \exp\big( -y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i) \big) \Big)$$

$$= \qquad\qquad =$$

$$= w_l + C \sum_{i=1}^{n} \frac{-y_i w^\top \nabla_w \phi_w(x_i)}{1 + \exp(y_i w^\top \phi_w(x_i))}$$

From now on, denote the $ij$th entry of $W_l$ by $w_{ijl}$.

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

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

$$\nabla_{W_l} F(\mathbf{w}, W) \;=\; W_l + C \sum_{i=1}^{n} \underbrace{\nabla_{W_l} \log\Big(1 + \exp\big(-y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i)\big)\Big)}$$

From now on, denote the $ij$th 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:

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

## 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!**

# 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

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

① ② ① ② ① ③

# 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:

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

$$= \begin{cases} 1 & \text{if } u_e \geq 0 \\ 0 & \text{else wise} \end{cases}$$

$$= \Theta(u_e)$$

# Second Term

We compute the second term as:

$$② \qquad \frac{\partial \mathbf{u}_l}{\partial \mathbf{v}_{l-1}} = \frac{\delta W_l^T v_{l-1}}{\delta v_{l-1}} = W_l^T$$

# Third Term

We compute the third term as:

$$③ \quad \frac{\partial \mathbf{u}_l}{\partial w_{ijl}} = \frac{\delta\, W_l^T v_{l-1}}{\delta w_{ijl}} = \frac{\delta \left( \sum_k w_{kk'l} v_{k,l-1} \right)_{k'}}{\delta w_{ijl}}$$

$$= v_{i,l-1}\, e_j$$

Notation:

- $v_{k,l-1}$ .... kth entry of $v_{l-1}$
- $e_j$ .... unit vector with 1 in jth component

# Putting Things Together

Our chain rule formula from Slide 9 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

$$
\begin{array}{l}
\text{initialize } \mathbf{v}_0 := \mathbf{x} \\
\textbf{for } l = 1 : (L-1) \textbf{ do} \\
\quad \mathbf{u}_l := W_l^\top \mathbf{v}_{l-1} \\
\quad \mathbf{v}_l := \Theta(\mathbf{u}_l) \\
\textbf{end for}
\end{array}
$$

$x \to u_1 \to v_1 \to u_2 \to v_2 \to \ldots$

$\to v_L$

Then, we compute the gradient via the chain rule:

## Backward propagation

$$
\begin{array}{ll}
\text{initialize } \delta_L := \Theta(\mathbf{u}_L) & \\
\nabla_{w_{ijL}} \phi_W(\mathbf{x}) := \delta_L v_{i,L-1} \mathbf{e}_j & \forall i, j \\
\textbf{for } l = (L-1) : 1 \textbf{ do} & \\
\quad \delta_l := \delta_{l+1} W_{l+1}^\top \Theta(\mathbf{u}_l) & \\
\quad \nabla_{w_{ijl}} \phi_W(\mathbf{x}) := \delta_l v_{i,l-1} \mathbf{e}_j & \forall i, j \\
\textbf{end for} &
\end{array}
$$

$\cdot\, W_{l+1}^\top \leftarrow \Theta(u_l) \leftarrow u_L$

# 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