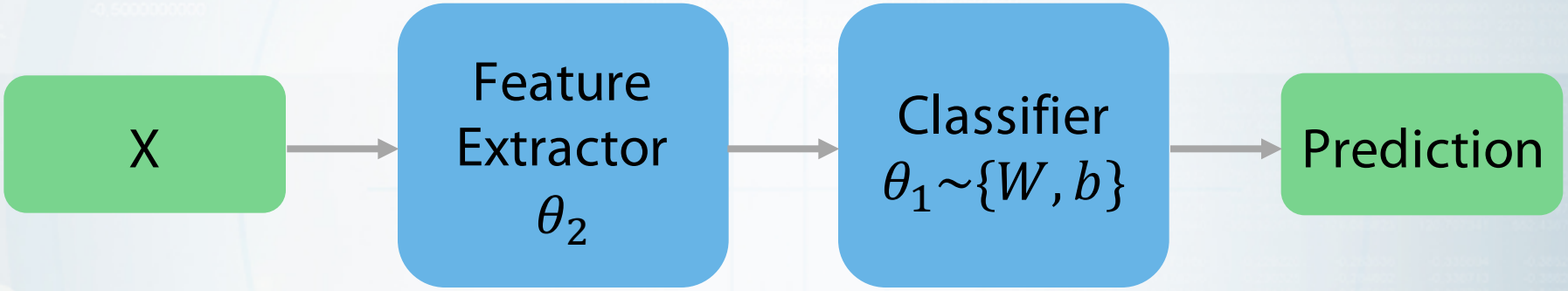
Training a neural network

In this lecture

Learn how to train a deep learning model:

- Computing the gradient
- Backpropagation

Training?



Training: $\underset{\theta_1, \theta_2}{\operatorname{argmin}} L\big(y, P(y|x)\big)$

Optimization!

Deep model is just a (composite!) function

- $f_1(f_2(f_3(x, w_3), w_2), w_1) \rightarrow loss$
- x – features vector
- $w_{1,2,3}$ – model parameters

Remember those optimization methods from the last week?

But they need gradient…

Differentiation recap

$$f: \mathbb{R} \to \mathbb{R}$$

You know what to do

Differentiation recap

$$f: \mathbb{R}^n \rightarrow \mathbb{R}$$

X is now a vector $[x_1, x_2, \ldots, x_n]$

f' is now a vector of partial derivatives $\frac{df}{dx} = \left[\frac{df}{dx_1}, \frac{df}{x_2}, \ldots, \frac{df}{x_n}\right]$

Differentiation recap

$$f: \mathbb{R}^n \to \mathbb{R}^m$$

X is a vector $[x_1, x_2, \ldots, x_n]$

f is also vector $[f_1, f_2, \ldots, f_m]$

f' is now a Jacobian matrix

$$\frac{df}{dx} = \left[\frac{df}{dx_1} \cdots \frac{df}{dx_n}\right] = \begin{bmatrix} \frac{df_1}{dx_1} & \cdots & \frac{df_1}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{df_m}{dx_1} & \cdots & \frac{df_m}{dx_n} \end{bmatrix}$$

Matrix derivative

$$F: \mathbb{R}^{m \cdot n} \rightarrow \mathbb{R}^{p \cdot q}$$

X is a matrix

F is also a matrix

F' is now
- a matrix of matrices
- fourth-rank tensor
- four-dimensional array

$$\frac{\partial F}{\partial X} = \begin{bmatrix} \frac{\partial F}{\partial X_{11}} & \cdots & \frac{\partial F}{\partial X_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{dF}{\partial X_{m1}} & \cdots & \frac{dF}{\partial X_{mn}} \end{bmatrix}$$

$$\frac{\partial F}{\partial X_{ij}} = \begin{bmatrix} \frac{\partial F_{11}}{\partial X_{ij}} & \cdots & \frac{\partial F_{1q}}{\partial X_{ij}} \\ \vdots & & \vdots \\ \frac{\partial F_{p1}}{\partial X_{ij}} & \cdots & \frac{\partial F_{pq}}{\partial X_{ij}} \end{bmatrix}$$

is a $p \times q$ matrix

Matrix derivative example

$$W^2 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}^2 = \begin{bmatrix} w_{11}^2 + w_{12}w_{21} & w_{11}w_{12} + w_{12}w_{22} \\ w_{21}w_{11} + w_{22}w_{21} & w_{21}w_{12} + w_{22}^2 \end{bmatrix}$$

$$\frac{dW^2}{dW} = \begin{bmatrix} \begin{bmatrix} 2w_{11} & w_{12} \\ w_{21} & 0 \end{bmatrix} & \begin{bmatrix} w_{21} & w_{11} + w_{22} \\ 0 & w_{21} \end{bmatrix} \\ \begin{bmatrix} w_{12} & 0 \\ w_{11} + w_{22} & w_{12} \end{bmatrix} & \begin{bmatrix} 0 & w_{12} \\ w_{21} & 2w_{22} \end{bmatrix} \end{bmatrix}$$

4 Dimensions

Chain rule

$$L(x) = g\big(f(x)\big)$$

For $f, g : \mathbb{R} \to \mathbb{R}$ $\quad \dfrac{dL}{dx} = \dfrac{dg}{df} \cdot \dfrac{df}{dx}$

$$\left.\frac{dL}{dx}\right|_{x=x_0} = \left.\frac{dg}{df}\right|_{u=f(x_0)} \cdot \left.\frac{df}{dx}\right|_{x=x_0}$$

Chain rule example

$$\frac{d}{d\vec{x}}\sin(|\vec{x}|)\Big|_{\vec{x}=\vec{a}} = \frac{d}{dy}\sin(y)\Big|_{y=\vec{a}} \cdot \frac{d}{d\vec{x}}|\vec{x}|\Big|_{\vec{x}=\vec{a}} =$$

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\vec{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

$$= \cos(|\vec{a}|) \cdot \frac{d\sqrt{x_1^2 + x_2^2}}{d\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}\Bigg|_{\vec{x}=\vec{a}} =$$

$$= \cos(|\vec{a}|) \cdot \frac{1}{\sqrt{a_1^2 + a_2^2}} \cdot \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

Chain rule quiz (технический слайд)

Backpropagation

Chain rule can be evaluated numerically!

1. Compute network output and the loss value
2. Compute $\frac{\text{dLoss}}{\text{dActivation\_of\_the\_last\_layer}}$
3. For each layer, starting from the last:
    1. Compute $\frac{\text{dActivation}}{\text{dLayer\_parameters}}$, $\frac{\text{dActivation}}{\text{dLayer\_input}}$
    2. Multiply it by $\frac{\text{dLoss}}{\text{dActivation}}$, get $\frac{\text{dLoss}}{\text{...}}$
    3. Make optimization step for the parameters

Summary

- Neural network training is an optimization problem
- Commonly solved via gradient-based methods
- Gradients are computed via backpropogation