Lecture October 1

Gradient descent methods

- cost function
  $C(\beta) / \hat{\beta} = \min_{\beta \in \mathbb{R}^p} C(\beta)$
- gradient $g(\beta) = \nabla_\beta C(\beta)$
- Hessian $H(\beta) = \frac{\partial^2 C}{\partial\beta \partial\beta^T}$

- iterative scheme

$$\beta^{(n+1)} = \beta^{(n)} - \gamma^{(n)} \nabla_\beta C(\beta^{(n)})$$
$$= \beta^{(n)} - \gamma^{(n)} g(\beta^{(n)})$$

Log-Reg and linear regression we have often purely convex functions. Iteration stops when $\nabla_\beta C = g = 0$

Linear Regression

$$g \propto \underline{X}^T (X\beta - \underline{y})$$

$$\beta \in \mathbb{R}^p \quad X \in \mathbb{R}^{n \times p}$$

$$n = 10^5 \quad p = 10^3$$

Flops for $X'X \sim p^2 \cdot n$

$10^6 \cdot 10^5 \sim 10^{11}$

every iteration $X^TX \in \mathbb{R}^{p \times p}$

$(X^TX)\beta \sim p^2 \sim 10^6$

Standard SGD: calculate gradient as expectation value

Recipe

- subselect, at random, $n_B \le N$ of training point

|| without replacement

- fix a learning rate $\gamma^{(0)} = \gamma_0$
- place them in a mini-batch B_1
- compute the gradient

$$\beta_1^{(1)} = \beta_0^{(0)} - \gamma_0 \sum_{i \in B_1} \nabla_\beta (C(\beta^{(0)}))$$

- Select randomly another n_B mini-batch B_2
- continue till we have

a ·

$\sim N/n_B$

start with $\beta^{(0)}$, in B_1

update B_1 (β_1 updated)

Then ~~so~~ update B_2 using β_1

$$\beta_2 = \beta_1 - \gamma_1 \sum_{i \in B_2} \nabla_\beta C((\beta_1))$$

- continue till last mini-batch,

Defines an epoch. Repeat for a given number of epochs.

Two more parameters;

1) # mini-batches

2) # epochs

Learning rate; decay γ linearly:

$$\gamma_k = (1-\alpha)\underline{\gamma_0} + \alpha \underline{\gamma_\tau}$$

$$\alpha = \frac{k}{\tau} \qquad \gamma_\tau \sim \frac{1}{100}\gamma_0$$

pick values for $\gamma_0 \in [10^{-5}, 10^{-4}, \ldots 10^{-3} \ldots 1]$

- 2nd-order Taylor approx to a function $f(x)$

$$f(x^{(0)} - \gamma g) \cong f(x^{(0)}) - \gamma g^T g + \frac{1}{2}\gamma^2 g^T H g$$

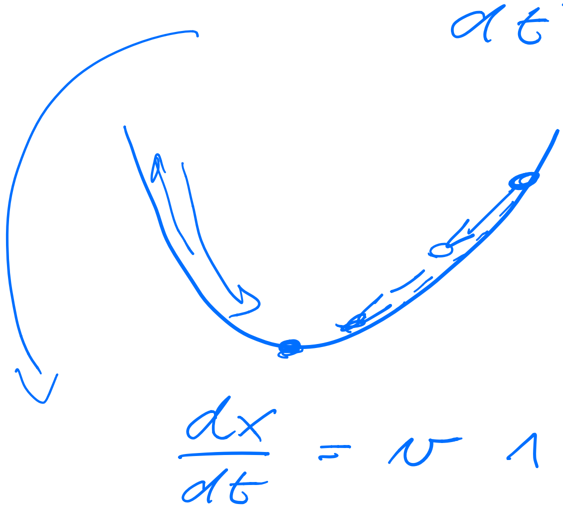
$$\gamma^* = \frac{g^T g}{g^T H g}$$

$$\frac{1}{2}\gamma^2 g^T H g - \boxed{\gamma g^T g}$$

- **Momentum SGD:**

analogy with Newtonian mechanics:

$$F(t) = \frac{d^2 x(t)}{dt^2} \quad \Big| \quad \beta^{(n)} \longrightarrow x(t)$$



$$\frac{dx}{dt} = v \wedge \frac{dv}{dt} = F(t)$$

Euler Forward method
$$\begin{cases} v_{i+1} = v_i + h\,\underline{F(t_i)} = v_i + hF_i \\ x_{i+1} = x_i + h\,v_i \end{cases}$$

$F(t_i)$ plays the role of $\nabla_\beta C(\beta)$

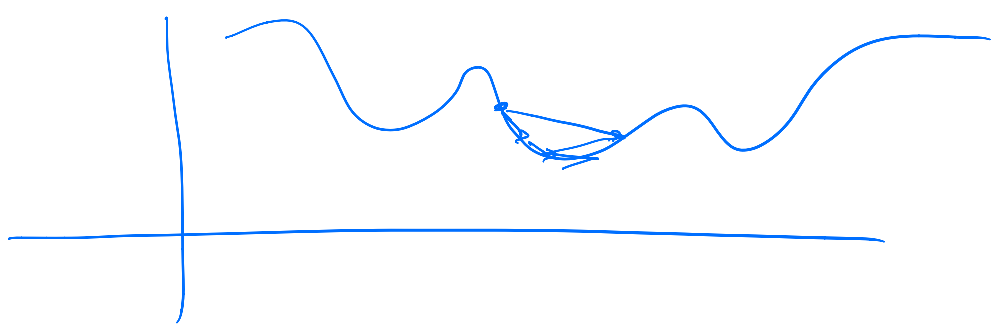
$$v_{i+1} = \underline{\alpha v_i} + hF_i$$

$$\beta^{(n+1)} = \beta^{(n)} - \gamma^{(n)} \nabla_\beta C(\underline{\beta}^{(n)})$$

Momentum SGD

$$v^{(n)} = \boxed{\alpha v^{(n-1)} - \gamma^{(n-1)} \times \nabla_\beta C(\beta^{(n-1)})}$$

$$\beta^{(n+1)} = \beta^{(n)} - v^{(n)}$$



<u>Adagrad</u>

SGD: accumulate the square gradient

$$r^{(n+1)} = r^{(n)} + g^T g$$

$$\beta^{(m+1)} = \beta^{(m)} - \frac{\gamma^{(m)}}{\delta + \sqrt{r}} g$$

$\delta \sim 10^{-8}$

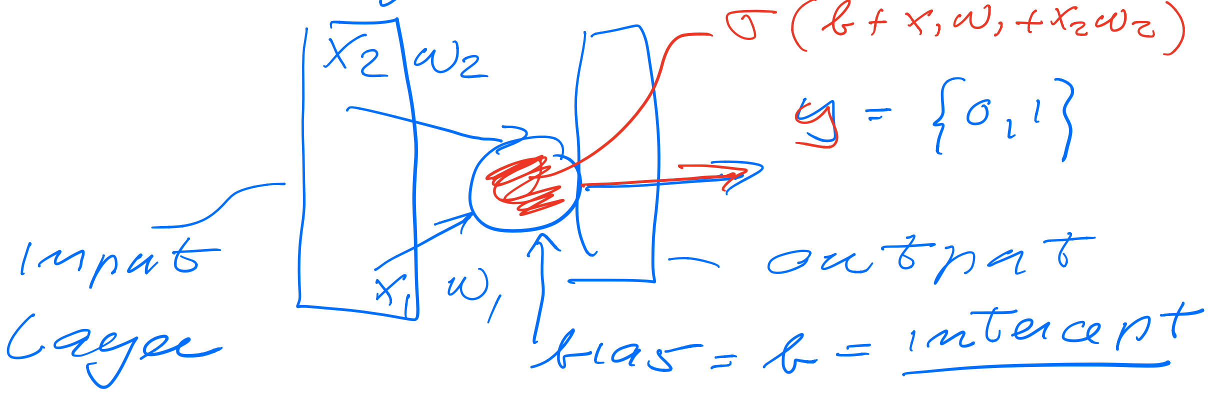
RMSprop

$$r^{(m+1)} = \rho r^{(m)} + (1-\rho) g^T g$$

$$\beta^{(m+1)} = \beta^{(m)} - \frac{\gamma^{(m)}}{\sqrt{\delta + r}} g$$

Neural networks

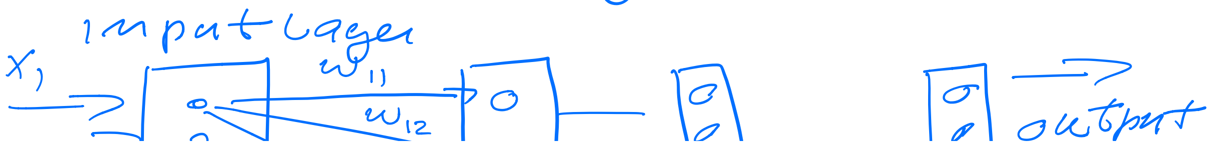
- single perceptron model

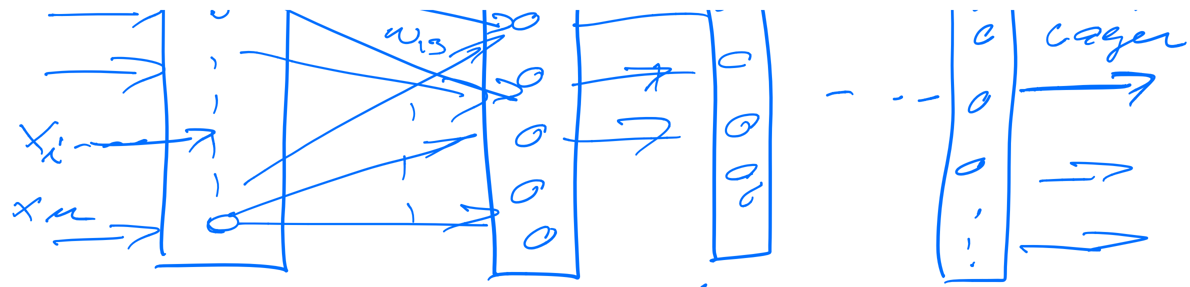


$$y = x_1 w_1 + x_2 w_2 + b$$

$$(\beta_0 + \beta_1 x_1 + \beta_2 x_2)$$

- multi-layer perceptron





hidden layer 1   layer i

XOR - model