Lecture October 7

In principle we compute sample expectation values

Example $MSE = C(\beta)$

$$= \mathbb{E}\left[(y - X\beta)^2\right]$$

$$= \mathbb{E}\left[(y - \tilde{y}(x,\beta))^2\right]$$

$$\nabla_\beta C(\beta) = \mathbb{E}\left[\left(\frac{\partial C}{\partial \beta}\right)\right]$$

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2$$

$$\frac{\partial C}{\partial \beta_j} = -\frac{2}{n}\sum_{i=0}^{n-1} x_{ij}(y_i - x_{ij}\beta_j)$$

SGD (mini-batches)

Algo

initialize η_k (learning

— ι — β

while stopping criterion not true

DO one epoch

- sample a mini-batch of examples from the data set

  $D = \{(x_0, y_0), \ldots, (x_{m-1}, y_{m-1})\}$

  in total m-mini batches

- compute gradients δ-g-

- update

  $\beta \leftarrow \beta - \eta_k g$

end while

it is common to "decay" the learning rate until a saturation τ

$$\eta_k = (1-\alpha)\eta_0 + \alpha \eta_\tau$$

$\alpha = \frac{k}{\tau}$ τ is chosen to

correspond to the # of iterations needed to make a few hundred passes through the training set

$n_y \sim 1\%$ of n_0

Momentum SGD

$$C(\hat{\beta}) \approx C(\beta^{(m)}) - (\underbrace{\beta - \beta^{(m)}}_{b})^T g + \frac{1}{2} \underbrace{b^T H b}_{n g^T}$$

physics inspired:

Newton's law

position $x(t)$ (1-dim)

Force $f(t)$, $m = 1$

$$f(t) = \frac{d^2x}{dt^2} \Rightarrow$$

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

$$v(t+\Delta t) \cong v(t) + \Delta t\, f(t)$$

$$x(t+\Delta t) \cong x(f) + \Delta t\, v(t)$$

our force is proportional to the negative gradient of the cost function

$x \Rightarrow \beta$ (diag space)

$$v \leftarrow \alpha v - \Delta t \nabla_\beta C(\beta) \quad (\eta)$$

$$\beta \leftarrow \beta + v$$

Momentum algo

Define η and momentum parameter α

initialize β and v

— , — epochs and batches

while stopping criterion not met

DO

- sample m — mini batch
- compute $g \leftarrow \frac{1}{m} \nabla_\beta C(\beta)$
- compute v

$v \leftarrow \alpha v - \eta g$

- update

$\beta \leftarrow \beta + v$

end DO

- Adagrad, RMS-prop

Adam

Newton-Raphson

$$\beta \leftarrow \beta - \underline{\underline{H}}^{-1} \underline{g}$$

Hessian — gradient

MSE example

$$C(\beta) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2$$

$$= \frac{1}{n} \sum_{i=0}^{n-1} \left(y_i - \hat{y}_i(x_i, \beta) \right)^2$$

$$\frac{\partial C(\beta)}{\partial \beta_j} = -\frac{2}{n} \sum_{i=0}^{n-1} \left(y_i - \hat{y}_i \right) \times \frac{\partial \hat{y}_i}{\partial \beta_j}$$

$$\frac{\partial^2 C}{\partial \beta_j \partial \beta_k} = \frac{2}{n} \sum_{i=0}^{n-1} \frac{\partial \hat{y}_i}{\partial \beta_j} \frac{\partial \hat{y}_i}{\partial \beta_k} - \frac{2}{n} \sum_{i=0}^{n-1} \left(y_i - \hat{y}_i \right) \frac{\partial^2 \hat{y}_i}{\partial \beta_j \partial \beta_k}$$

$$\approx 0$$

$$\approx \frac{2}{n} \sum_{i=0}^{n-1} \frac{\partial \hat{y}_i}{\partial \beta_j} \frac{\partial \hat{y}_i}{\partial \beta_k}$$

$$\frac{\partial \hat{y}_i}{\partial \beta_j} \frac{\partial \hat{y}_i}{\partial \beta_j} \approx \eta_j^{-1}$$

Algo Adagrad

Define η

initialize β

— , — $\delta \sim 10^{-7}$

while stopping not met

DO

- sample mini batches
- compute $g \leftarrow \frac{1}{m} \nabla_\beta C(\beta)$
- accumulate squared gradients

  $r \leftarrow r + g \odot g$
- compute $\Delta\beta$

  $\Delta\beta \leftarrow -\frac{\eta}{\delta + \sqrt{r}} \odot g$
- update $\beta \leftarrow \beta + \Delta\beta$

end DO

In RMS-prop :

$r \leftarrow \rho r + (1-\rho) g \odot g$

$$\Delta p = -\frac{\eta}{\sqrt{S+\varepsilon}} \odot g$$

Go-to ones

SGD with momentum

RMS-prop (in deep learning for non-convex $C(p)$)

Adagrad + ADAM

Automatic differentiation = Autograd

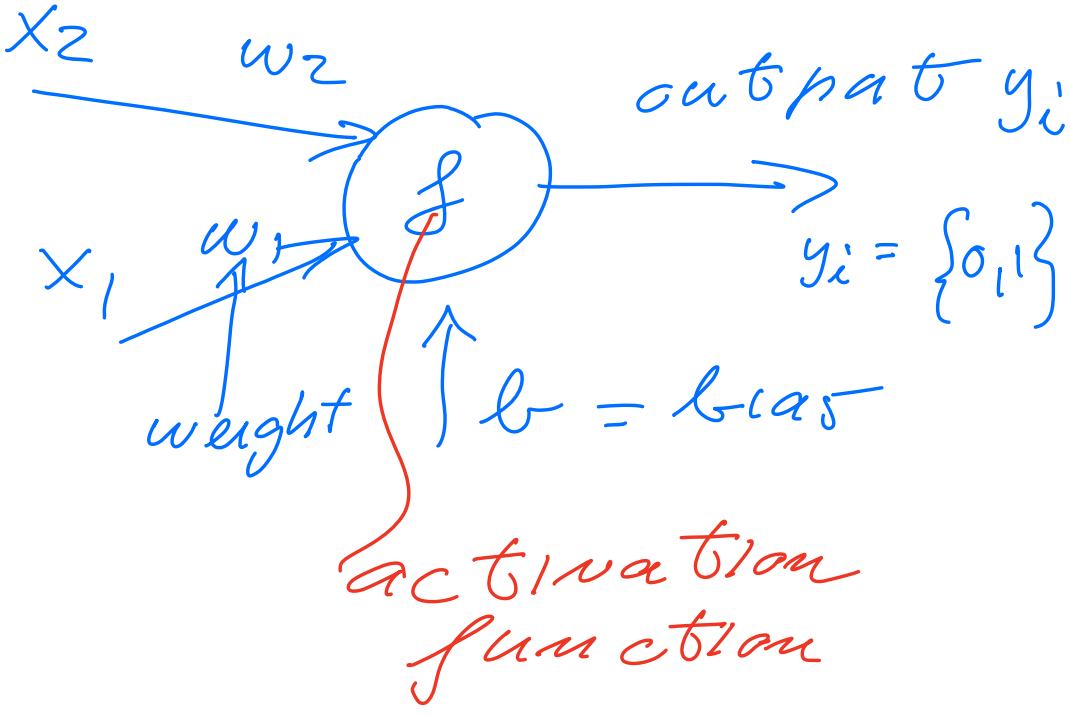
$$f' \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$

$$f'' \approx \frac{f(x+\Delta x) + f(x-\Delta x) - 2f(x)}{\Delta x^2}$$

Deep-learning and Feed Forward Neural network (FFNN)

Basic definition :

- single perception model with one neuron / node / unit



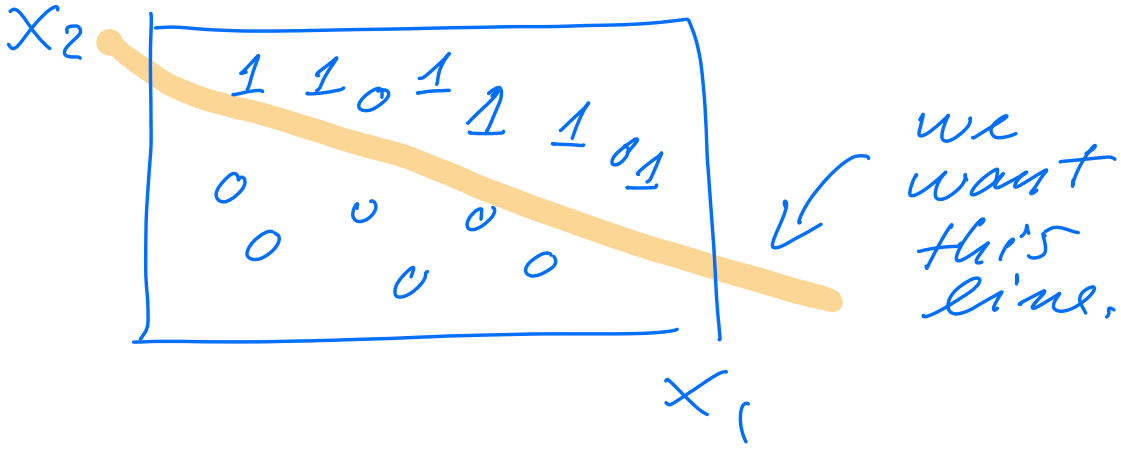
$$f(x_1 w_1 + x_2 w_2 + b) = y_i$$

Example data set:

x_1 = # hours slept

x_2 = # hours studied

$y_i = \begin{cases} 0 = \text{grade below average} \\ 1 = \text{grade above average} \end{cases}$



Universal approximation theorem: with one or more hidden layers we can have a networks which can fit non-linear functions.

Single perceptron model can only fit a linear model.