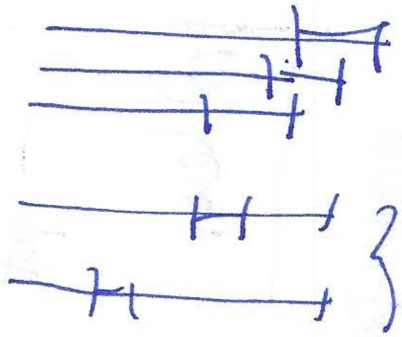
- cross validation for time series:
- or split data across the time series.



<u>linear regression</u> data dim $d >> n$ #observation.

(1) unique best fit planes run exactly through data! MSE = 0 !
or many!
caution this - need CV.

<u>examples</u> - blood pressure from DNA ~ 1 million data points dimensions.
~ 100 patients
- online shopping habits from browsing history as word cloud/something else.
~ 100 customers

(2) interpretability - want to know what outcome depends on.

basic methods: · subset selection
· shrinkage.
· dimension reduction.

<u>subset selection</u> Q: how many possible subsets? 2^d ! .
- better: consider d 1-var models & pick best.
then consider $d-1$ 2-var models & pick best
⋮
stop when CV gets bad.

<u>Shrinkage</u>: least squares: linear mod $\beta_0 + \beta_1 x_1 + \cdots \beta_d x_d = f(x)$

minimize $\sum_{i=1}^{n} (y_i - f(x_i))^2 \leftarrow$ LSE.
RSS

ridge regression minimizes $\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \sum_{i=1}^{d} \beta_i^2$

RSS ↑ tuning parameter.

($\lambda = 0$ get RSS LSE)

large λ — nearly all β_i zero ∈ close close/large β_i

Lasso min $\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \sum_{i=1}^{d} |\beta_i|$ ← ℓ^1 not ℓ^2.

large λ → forces many β_i to zero.

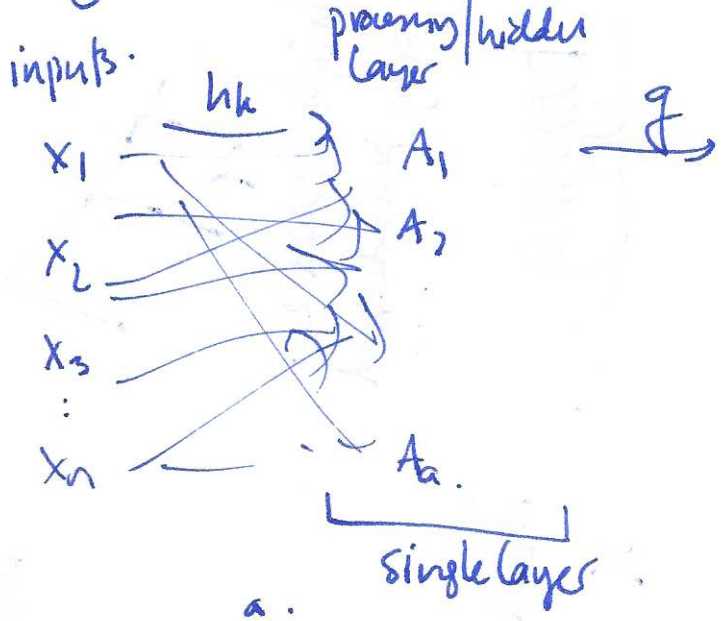
Dimension reduction:

— PCA.

warning: collinearity — in large dimensions often many things weakly correlated, and everything is a linear combination of everything else. so if you build blood pressure model on ~20 vars. ← will be able to build similarly good model on 20 different vars!

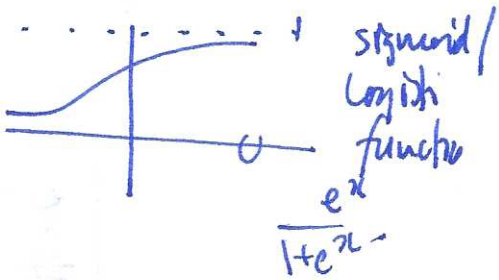
deep learning / neural nets.

motivation : neurons in the brain

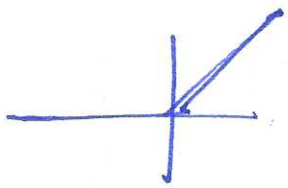


$$f(x) = \beta_0 + \sum_{k=1}^{a} \beta_k h_k(x)$$

activation functions: $A_k = h_k(x) = g\left(w_{k0} + \sum_{j=1}^{d} w_{kj} x_j\right)$.



ReLU



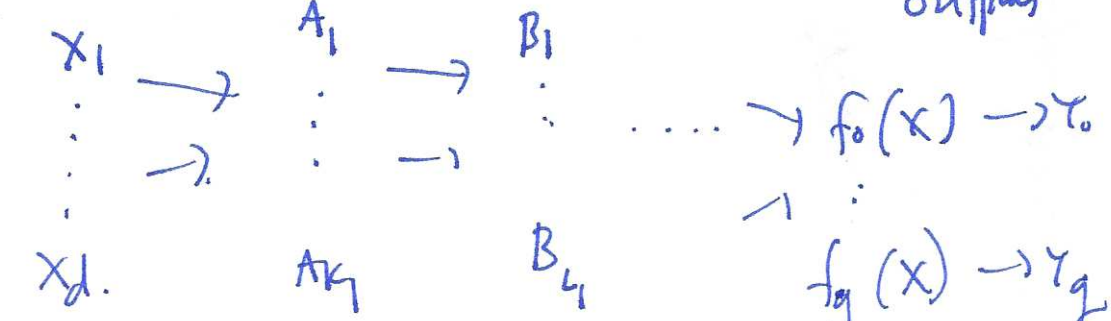
$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0. \end{cases}$$

note all of these parameters need to be estimated! w_{ij} β_i.

(note: if activation function was linear, would just get linear model)

choose parameters to ~~estimate~~ minimize $\sum (y_i - f(x_i))^2$ RSS.

multilayer.



note input $d = 784$ two layers $K_1 = 256$, $K_2 = 128$ give $\sim 200{,}000$ parameters.

Q: how to find $\beta = (\beta_0, \dots \beta_K)$. w_{kj}

$w_k = (w_{k0}\ w_{k1} \dots w_{kd})$.

$$\min_{w_k\ \beta} \sum (y_i - f(x_i))^2 \qquad f(x_i) = \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\right).$$

— function non-convex in parameters.

— (possibly) multiple solutions.

— slow learning / gradient descent

— regularization (lasso / ridge).

$$R(\theta) = \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \quad (*)$$

$\uparrow$ $w_{ij}\ \beta_i$.

gradient descent: 1. start with a guess θ^0, $t = 0$.

2. repeat until error (*) stops decreasing:

a) find a vector δ giving a small change in θ s.t.

$\theta^{t+1} = \theta^t + \delta$ has smaller error. $(R(\theta^{t+1}) < R(\theta_t))$.

b) $t \mapsto t+1$.

Back propagation. gradient: vector of partial derivatives.

$$\nabla R(\theta) = \left.\frac{\partial R(\theta)}{\partial \theta}\right|_{\theta = \theta_t} \quad \leftarrow \text{gives direction of fastest increase}$$

$$\theta_{t+1} = \theta_t - \rho \nabla R(\theta_t). \qquad \rho \text{ learning rate.}$$

$$R(\theta) = \sum_{i=1}^{n} R_i(\theta) = \sum_{i=1}^{n} \left(y_i - f_\theta(x_i)\right)^2$$

$$R_i(\theta) = \left(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k \, g\Big(\underbrace{w_{k0} + \sum_{j=1}^{d} w_{kj} x_{ij}}_{z_{ik}}\Big)\right)^2$$

$$\frac{\partial R_i(\theta)}{\partial \beta_k} = \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial \beta_k}$$

$$= -\underbrace{\left(y_i - f_\theta(x_i)\right)}_{\text{residual}} g(z_{ik})$$

$$\frac{\partial R_i(\theta)}{\partial w_{kj}} = \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{kj}}$$

$$= -\underbrace{\left(y_i - f_\theta(x_i)\right)}_{\text{residual}} \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}$$

stochastic gradient descent : subsample observations $1, \dots, n$ as $i_1, \dots, i_m$.

dropout learning: set some unit to ~~zero~~. ← can do fraction of at random or something else..

~~training~~: regularization: RSS + $\sum \beta_j^2$

Tuning:
- number of layers / number of units per layer
- regularization: dropout rate ϕ. Lasso parameter λ.
- stochastic GD: batch size / # iterations / data augmentation.