Statistical Learning VIII (21-12-2017)

0. Exam
1. Stochastic Optimization
2. Deep Learning / Neural Networks
   a) Neural Networks
   b) Size
   c) Overfitting and Generalization
   d) Online Gradient Descent and Backpropagation
   e) Hardware and Software

0. Exam

- Open book (bring a paper copy!)
- Bring anything on paper you like (notes, prints, etc.)

1. Stochastic Optimization

a) Non-stochastic Optimization

$$\beta^* = \underset{\beta}{\operatorname{argmin}}\, F(\beta)$$

~~$\beta = (\beta_1, \ldots, \beta_p)$~~

$\beta = (b_1, \ldots, b_p)$

Examples:

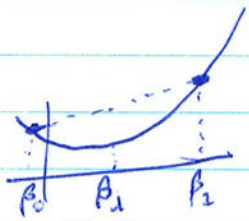
$$F(\beta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i^T\beta)^2 \quad \text{(least squares)}$$

$$F(\beta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=2}^{p}\beta_j^2 \quad \text{(ridge regression)}$$

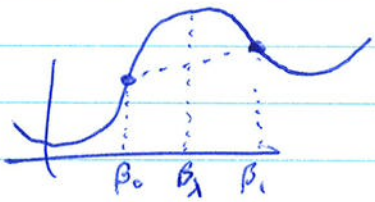
$$F(\beta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=2}^{p}|\beta_j| \quad \text{(lasso)}$$

$$F(\beta) = \frac{1}{N}\sum_{i=1}^{N}\log(1+e^{-y_i x_i^T\beta}) \quad \text{(logistic regression)}$$

All these examples convex functions $F(\beta)$:



convex



not convex

$\beta_\lambda = (1-\lambda)\beta_0 + \lambda\beta_1$

see Fig1 handout

$$F(\beta_\lambda) \leq (1-\lambda)F(\beta_0) + \lambda F(\beta_1) \quad \text{for all } \lambda \in [0,1]$$

Convexity ⇒ Can find minimum by making small improvements to the parameters:

$$\beta_{t+1} = \beta_t - \eta_t \nabla F(\beta_t) \quad \text{(gradient descent)}$$

η_t: step size

β_{t+1}: new parameters

β_t: previous parameters

$\nabla F(\beta_t)$: gradient

$$\nabla F(\beta) = \begin{pmatrix} \frac{\partial}{\partial b_1} F(\beta) \\ \vdots \\ \frac{\partial}{\partial b_p} F(\beta) \end{pmatrix}$$

- vector pointing in direction that is the most up from β
- so move in direction of negative gradient to go the most down

Best choice of η_t depends on type of function.

For instance:

~~If F is differentiable and two times and~~

If F is γ-smooth, then for $\eta_t = \frac{1}{\gamma}$:

$$F(\beta_t) - F(\beta^*) \leq \frac{2\gamma \|\beta_1 - \beta^*\|^2}{t-1}$$

($\|\beta_1 - \beta^*\|^2$ ← starting distance from optimum)

more steps ⇒ closer to optimum

(γ-Smooth means that second derivative in any direction is at most γ.)

b) Stochastic Optimization

Suppose $F(\beta) = \frac{1}{N} \sum_{i=1}^{N} f_i(\beta)$ for some (convex) functions $f_1, \dots, f_N$.

Then computing

$$\nabla F(\beta) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\beta)$$

takes a lot of computation when N is large

All examples are of this form!

(For ridge and lasso take

$$f_i(\beta) = (y_i - X_i^T \beta)^2 + \lambda \text{pen}(\beta).$$)

Stochastic Gradient Descent:

Choose i_t randomly from $\{1, 2, \ldots, N\}$

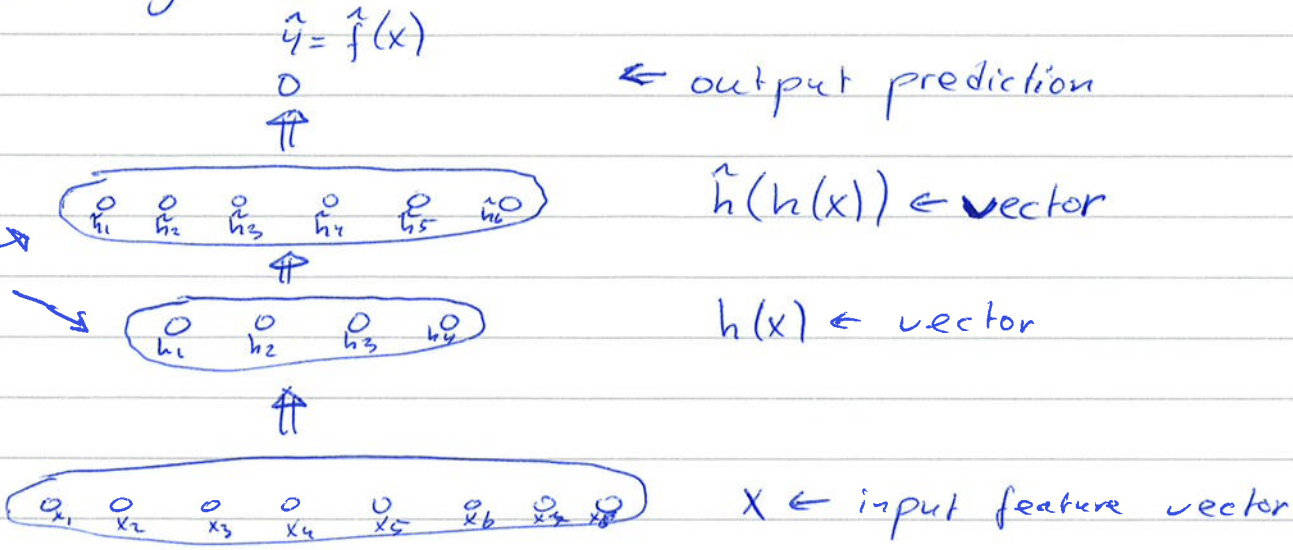
$$\beta_{t+1} = \beta_t - \eta_t \nabla f_{i_t}(\beta_t)$$

- Can do N steps in same time as one update of regular gradient descent.

$$\mathop{\mathbb{E}}_{i_t}[\nabla f_{i_t}(\beta_t)] = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\beta_t) = \nabla F(\beta_t)$$

is unbiased estimator for $\nabla F(\beta_t)$

Deep learning = Big neural networks



Learn feature transformations from the data!

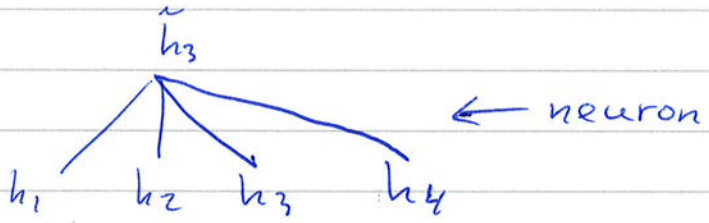
└ completely replaced manually designed SWIFT features in computer vision ← took 10 years to design these

* Can have >3 layers
* Intermediate vectors often very large

} "deep" learning

* Loss $L(y, \hat{y})$ can be ~~easy~~ anything. E.g. quadratic for regression, logistic for classification, etc

How do we learn transformations between layers? Zoom in.



$$\tilde{a}_3 = \sum_{i=1}^{4} w_{h_i \to \tilde{h}_3} \cdot h_i + \text{intercept}_{\tilde{h}_3}$$

linear "activation"

parameters

$$\tilde{h}_3 = \sigma(\tilde{a}_3)$$

σ ← activation function (nonlinear)

- sigmoid: $\sigma(a) = \frac{1}{1+e^{-a}}$

- tanh: $\sigma(a) = \frac{1-e^{-2a}}{1+e^{-2a}} = 2 \cdot \text{sigmoid}(2a) - 1$

- rectified linear unit (ReLu): $\sigma(a) = \max\{a, 0\}$ ← (piece-wise linear; helped a lot in performance)
  recently popular

- ...

Size: At least 2 or 3 hidden layers

- every neuron has its own parameters
- modern network may have
  - inputs: 65536 = 256 x 256 color pixels
  - layers: 500
    500
    2000

rough estimate nr of parameters:

$65536 \times 2000 + 2000 \times 500 + 500 \times 500 + 500 \times 1$

$\approx$ 130 million parameters

So will overfit like crazy...

c) Overfitting and Generalization

- $p \gg N$, so typically get training error for classification to (approximately) 0
- Which solution with 0 training error is chosen? Depends on optimization method, so optimizer acts as a regularizer
- How does this regularizer behave? Active research topic!

How to reduce overfitting

1. Very large data sets (15 million examples in ImageNet data set)
2. Exploit domain knowledge to reduce nr of parameters: images have locally similar structures, so tie parameters together in first layer(s). (convolutional layers)
3. Regularize ($L_1, L_2, \ldots$)
4. Early stopping: monitor generalization error on validation set during optimization, and stop when it starts to increase.

d) Online Gradient Descent and Backpropagation

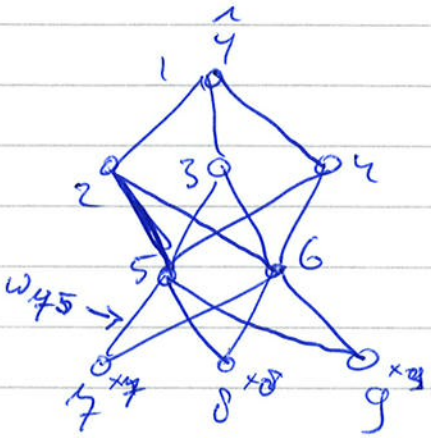
- θ is vector with all weights (e.g. 130 million parameters)
- minimize $E(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_\theta(x_i))$ using stochastic gradient descent.
- $L(y_i, f_\theta(x_i))$ is not convex in θ, but we accept to find a local minimum.
- Big open question: why does it find good local minima in practice? ~~&~~ appears to be better for larger networks

How to compute gradients $\nabla_\theta L(y_t, \hat{f}_\theta(x_t))$?

130 million parameters: only have time for fixed nr. of computations per parameter

Solution: Backpropagation ← use chain rule for derivatives many times:

$$\frac{\delta a}{\delta b} = \frac{\delta a}{\delta c} \cdot \frac{\delta c}{\delta b}$$



For each neuron in o:

$o_m = \sigma(a_m)$ (o: output, a: activation) $\quad m = 2, \dots, 9$

$o_1 = \hat{y}$

1. $\frac{\delta L(y_t, \hat{f}_\theta(x_t))}{\delta o_1}$ is $\frac{\delta L(y_t, \hat{y})}{\delta \hat{y}}$ is derivative of the loss

2. chain rule:

Suppose we have $\frac{\delta L(y_t, \hat{f}_\theta(x_t))}{\delta o_5}$, then

$$\frac{\delta L(y_t, \hat{f}_\theta(x_t))}{\delta w_{45}} = \frac{\delta L(y_t, \hat{f}_\theta(x_t))}{\delta o_5} \cdot \frac{\delta o_5}{\delta w_{45}}$$

$\uparrow$ 9

$$\frac{\delta o_5}{\delta w_{45}} = \frac{\delta o_5}{\delta a_5} \cdot \frac{\delta a_5}{\delta w_{45}}$$

$$\frac{\delta o_5}{\delta a_5} = \frac{\delta \sigma(a_5)}{\delta a_5} = \sigma'(a_5)$$

$$\frac{\delta a_5}{\delta w_{45}} = \frac{\delta \sum_{i=4}^{9} w_{i5} \cdot x_i}{\delta w_{45}} = x_4$$

3. Work backwards from output to inputs to compute partial derivatives w.r.t. each o_m if we already have these partial derivatives higher up in the network.

already computed

vector chain rule:

$$\frac{\delta L(y, f_\theta(x))}{\delta o_5} = \frac{\delta L(y, f_\theta(x))}{\delta o_2} \cdot \frac{\delta o_2}{\delta o_5} + \frac{\delta L(y, f_\theta(x))}{\delta o_3} \cdot \frac{\delta o_3}{\delta o_5} + \frac{\delta L(y, f_\theta(x))}{\delta o_4} \cdot \frac{\delta o_4}{\delta o_5}$$

$$\frac{\delta o_2}{\delta o_5} = \frac{\delta o_2}{\delta a_2} \cdot \frac{\delta a_2}{\delta o_5}$$

$\sigma'(a_2)$

$a_2 = w_{52} \cdot o_5 + w_{62} \cdot o_6$

$$\frac{\delta(a_2)}{\delta o_5} = w_{52}$$

Conclusion: only need to do fixed (nr of) operations for each element of the network.

↳ compute gradient vector quickly