Ex 2-1

Correction: $\frac{\partial g(\vec{x})}{\partial \vec{x}} = [\frac{\partial g(\vec{x})}{\partial x_1}, \dots, \frac{\partial g(\vec{x})}{\partial x_n}]^T$.

Ex 1-1:

a) $\vec{1}^T \to \vec{1}$

b) $2\vec{x}^T \to 2\vec{x}$

c) $2(\vec{x}-\vec{\mu})^T \to 2(\vec{x}-\vec{\mu})$

a)

cost function:

$\frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

gradient descent-based method

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \frac{\partial cost}{\partial \vec{w}} = \vec{w} - \eta \cdot \frac{\partial}{\partial \vec{w}}\left[\sum_{i=1}^{N}\frac{1}{2}(y_i - \hat{y}_i)^2\right]$$

$$= \vec{w} - \eta \cdot \sum_{i=1}^{N}\frac{\partial}{\partial \vec{w}}\left[\frac{1}{2}(y_i - \hat{y}_i)^2\right]$$

$$= \vec{w} - \eta \cdot \sum_{i=1}^{N}(y_i - \hat{y}_i) \cdot \frac{\partial \hat{y}_i}{\partial \vec{w}} = \vec{w} - \eta \cdot \sum_{i=1}^{N}(y_i - \hat{y}_i) \cdot \vec{x}_i.$$

Thus: $\vec{w} \leftarrow \vec{w} - \eta \sum_{i=1}^{N}[(y_i - \hat{y}_i)\vec{x}_i]$ ← a batch of sample.

Perceptron learning rule: $\vec{w} = \vec{w} - \eta \cdot y_i \cdot \vec{x}_i$. (delta-rule) (sample-based rule)

↑ one sample

b) sample-based rule for gradient-based learning rule:

$\vec{w} \leftarrow \vec{w} - \eta \cdot (y_i - \hat{y}_i)\vec{x}_i$. ← stochastic gradient descent

c) SGD can be learned on the fly. The model can be update sample by sample. It's unessary to recompute the whole model. Essentially better for large datased.

d) Striking difference: objective function.

- perceptron : binary outputs $\longrightarrow$ classification
- ADALINE : real outputs $\longrightarrow$ regression
- If data is separable, perceptron converge faster than ADALINE.

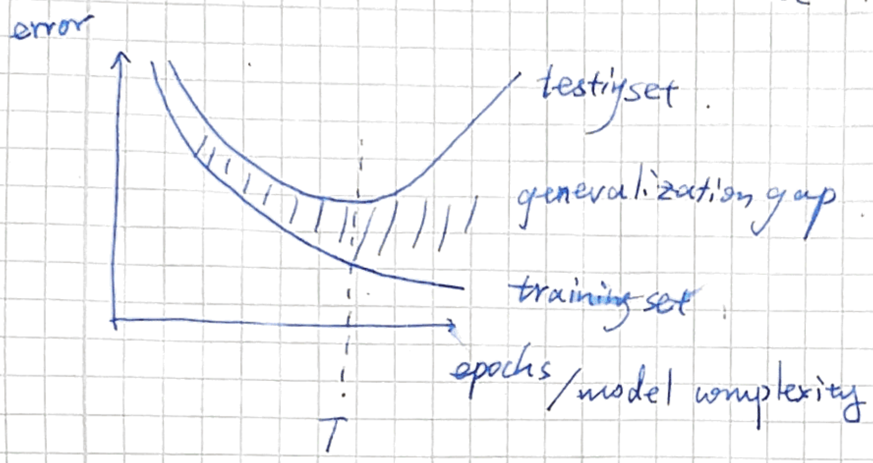
error / loss / cost / objective function

Ex 2-2.

a) overfitting: trained model over adapt to a given dataset.

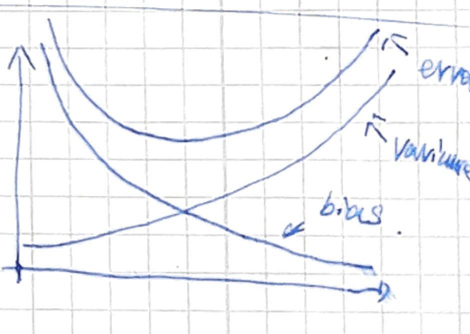
Reason: ① generalization reason (too many features)

② bias - variance decomposition



when #features >> #samples. the model over adapts to ~~the~~ sample while training.

Bias - Variation decomposition:



$$error = \mathbb{E}[(y-\hat{y})^2] \quad \text{(mean square error)}$$

$$= \mathbb{E}[(y-\mathbb{E}[\hat{y}]+\mathbb{E}[\hat{y}]-\hat{y})^2]$$

$$= \mathbb{E}[(y-\mathbb{E}[\hat{y}])^2] + \mathbb{E}[(\mathbb{E}[\hat{y}]-\hat{y})^2] + \mathbb{E}[2(y-\mathbb{E}[\hat{y}])(\mathbb{E}[\hat{y}]-\hat{y})]$$

ground truth ~~prediction~~ exp. of $\hat{y}$ prediction

$$= (y-\mathbb{E}[\hat{y}])^2 + \mathbb{E}[(\mathbb{E}[\hat{y}]-\hat{y}]^2 + 2(y-\mathbb{E}[\hat{y}])(\mathbb{E}[\hat{y}]-\mathbb{E}[\hat{y}])$$

$$= (bias)^2 + variance$$

b) ① very good prediction, but very bad on test dataset.

② model complexity measure: (typically, in traditional machine learning is the # parameters.

A model has high complexity may cause a overfitting problem. The generalization gap:

$$\text{Test error} \leq \text{training error} + \text{model complexity penalty}.$$

$$= O\left(\sqrt{\frac{\#\text{parameter}}{\#\text{samples}}}\right).$$

$\Rightarrow$ Testerror − Trainingerror << model complexity term

---

c) – Regularization. (weight decay)

– #features << # samples

Ex 2-3

a) continues / differentiable almost every where.

b) $$f(\vec{x}_i) = \sum_{h=0}^{M_\phi-1} w_h \phi_h(\vec{x}_i, \vec{v}_h) = \sum_{h=0}^{M_\phi-1}\left[\sum_{j=0}^{M} w_{h,j} \cdot x_{i,j}\right] \cdot w_h.$$

$$= \sum_{j=0}^{M}\left[\sum_{h=0}^{M_\phi-1} w_{h,j} \cdot w_h\right] \cdot x_{i,j} = \sum_{j=0}^{M} \alpha_j \cdot x_{i,j}.$$

which is equivalent to linear model.

c) No. Radial basis function (RBF): $\phi(x_{i,j}, \mu_j, \sigma_j) = \exp\{\frac{\|x_{i,j} - \mu_j\|}{2\sigma_j^2}$

which are typically used as a first layer in a 2-layer RBF network.

The second layer is a linear combination.

It is similar to a gaussian kernel density estimator.