Machine Learning from Data

Lecture 22: Spring 2021

Today’s Lecture

- Neural Networks and Overfitting
  - Approximation Vs Generalization
  - Regularization and Early Stopping
  - Minimizing in-sample error more efficiently

Neural Networks and Fitting the Data

Forward Propagation:

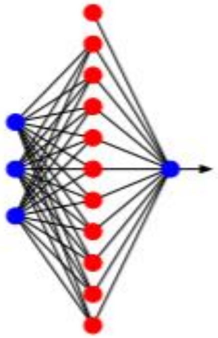
$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathrm{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{\mathrm{W}^{(2)}} \mathbf{s}^{(2)} \cdots \xrightarrow{\mathrm{W}^{(L)}} \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x})$$

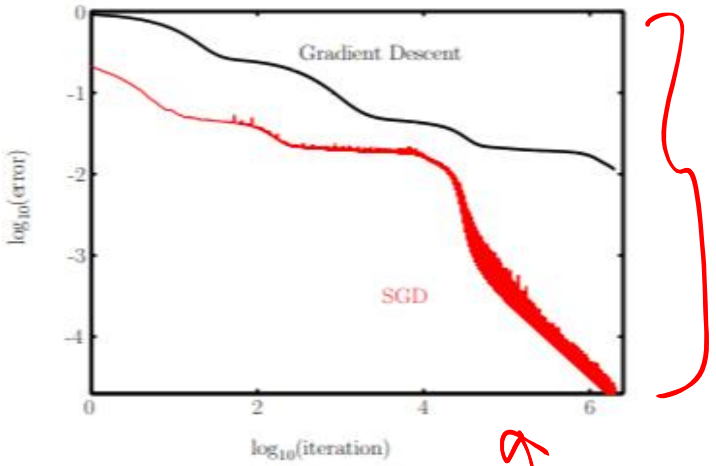
$$\mathbf{s}^{(\ell)} = (\mathrm{W}^{(\ell)})^{\mathrm{T}}\mathbf{x}^{(\ell-1)} \qquad \mathbf{x}^{(\ell)} = \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(\ell)}) \end{bmatrix}$$

$h(x)$

(Compute h and E_{in})

Choose $\mathrm{W} = \{\mathrm{W}^{(1)}, \mathrm{W}^{(1)}, \ldots, \mathrm{W}^{(L)}\}$ to minimize E_{in}





Gradient descent:

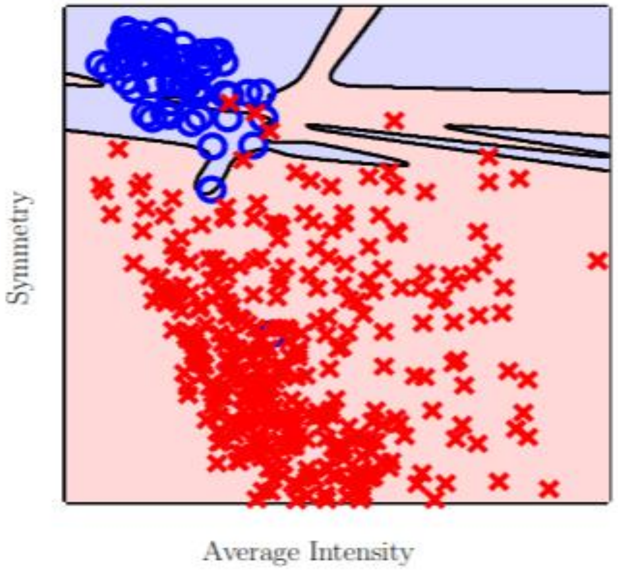
$$\mathrm{W}(t+1) \leftarrow \mathrm{W}(t) - \eta \nabla E_{\text{in}}(\mathrm{W}(t))$$

Compute gradient $\longrightarrow$ need $\dfrac{\partial \mathrm{e}}{\partial \mathrm{W}^{(\ell)}}$ $\longrightarrow$ need $\boldsymbol{\delta}^{(\ell)} = \dfrac{\partial \mathrm{e}}{\partial \mathbf{s}^{(\ell)}}$

$$\frac{\partial \mathrm{e}}{\partial \mathrm{W}^{(\ell)}} = \mathbf{x}^{(\ell-1)}(\boldsymbol{\delta}^{(\ell)})^{\mathrm{T}}$$

δ^{L}

δ^{L-1}



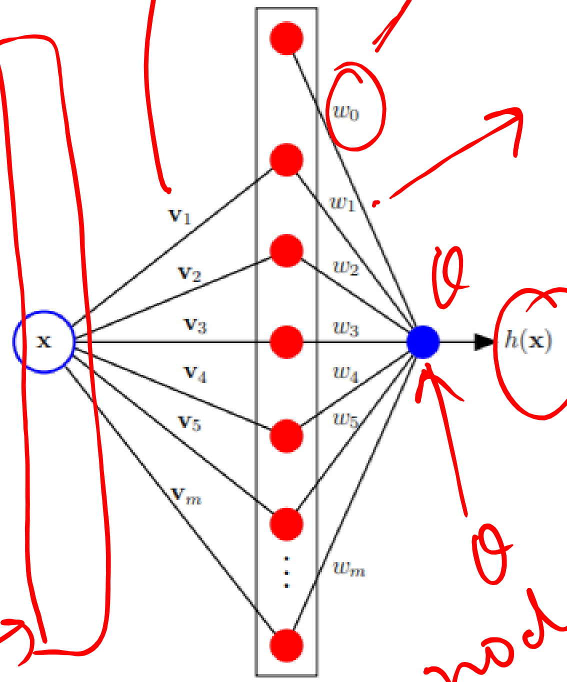
Backpropagation:

$$\boldsymbol{\delta}^{(1)} \longleftarrow \boldsymbol{\delta}^{(2)} \cdots \longleftarrow \boldsymbol{\delta}^{(L-1)} \longleftarrow \boldsymbol{\delta}^{(L)}$$

$$\boldsymbol{\delta}^{(\ell)} = \theta'(\mathbf{s}^{(\ell)}) \otimes \left[\mathrm{W}^{(\ell+1)}\boldsymbol{\delta}^{(\ell+1)}\right]_1^{d^{(\ell)}}$$

2-Layer Neural Network

perceptrons



θ

d

θ

m-nodes

$$w_1\,\theta(v_1^T x)$$
$$w_2\,\theta(v_2^T x)$$
$$\vdots$$
$$w_m\,\theta(v_m^T x)$$

$$h(\mathbf{x}) = \theta\left(w_0 + \sum_{j=1}^{m} w_j \theta\left(\mathbf{v}_j{}^{\mathrm{T}}\mathbf{x}\right)\right)$$

The Neural Network has a Tunable Transform

Linear Models.

$\Phi(x) = [1, \phi_1(x), \phi_2(x) \dots \phi_m(x)]$

$k = m$

Neural Network

$$h(\mathbf{x}) = \theta\left(w_0 + \sum_{j=1}^{m} w_j \theta\left(\mathbf{v}_j^{\mathrm{T}} \mathbf{x}\right)\right)$$

feature transform fits the data.

Nonlinear Transform

$\tilde{d} = m$

$$h(\mathbf{x}) = \theta\left(w_0 + \sum_{j=1}^{\tilde{d}} w_j \Phi_j(\mathbf{x})\right)$$

Z $E_{in} \times E_{out}$

k-RBF-Network

kernel

non-linear

$$h(\mathbf{x}) = \theta\left(w_0 + \sum_{j=1}^{k} w_j \phi\left(\| \mathbf{x} - \boldsymbol{\mu}_j \|\right)\right)$$

Similarity

μ_j's

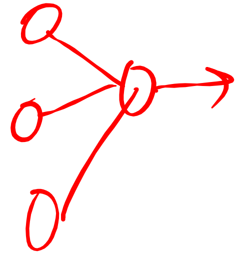
$$E_{\text{in}} = O\left(\frac{1}{m}\right)$$

approximation

Regression

$E_{in} = O\left(\frac{1}{\sqrt{m}}\right)$

$\theta = \text{sign}$



Generalization

$h(x) = \theta\left(w_0 + \sum_{j=1}^{m} w_j \, \theta(v_j^T x)\right)$

$O(md)$

MLP:

$$\boxed{d_{\text{VC}} = O(md\log(md))}$$

$$\boxed{m = \sqrt{N}}$$

(convergence to optimal for MLP, just like k-NN)

semi-parametric because you still have to learn parameters.

tanh :

$$d_{\text{VC}} = O(md(m+d))$$

$$E_{in} \in O\left(\frac{1}{m}\right)$$

$$E_{in} \in O\left(\frac{1}{\sqrt{m}}\right)$$

$$E_{out} \le E_{in} + O\left(\sqrt{\frac{d_{vc}}{N}\log N}\right)^{\frac{1}{4}}\ \text{-}\ \frac{1}{N^{1/4}}, \quad m = \sqrt{N}$$

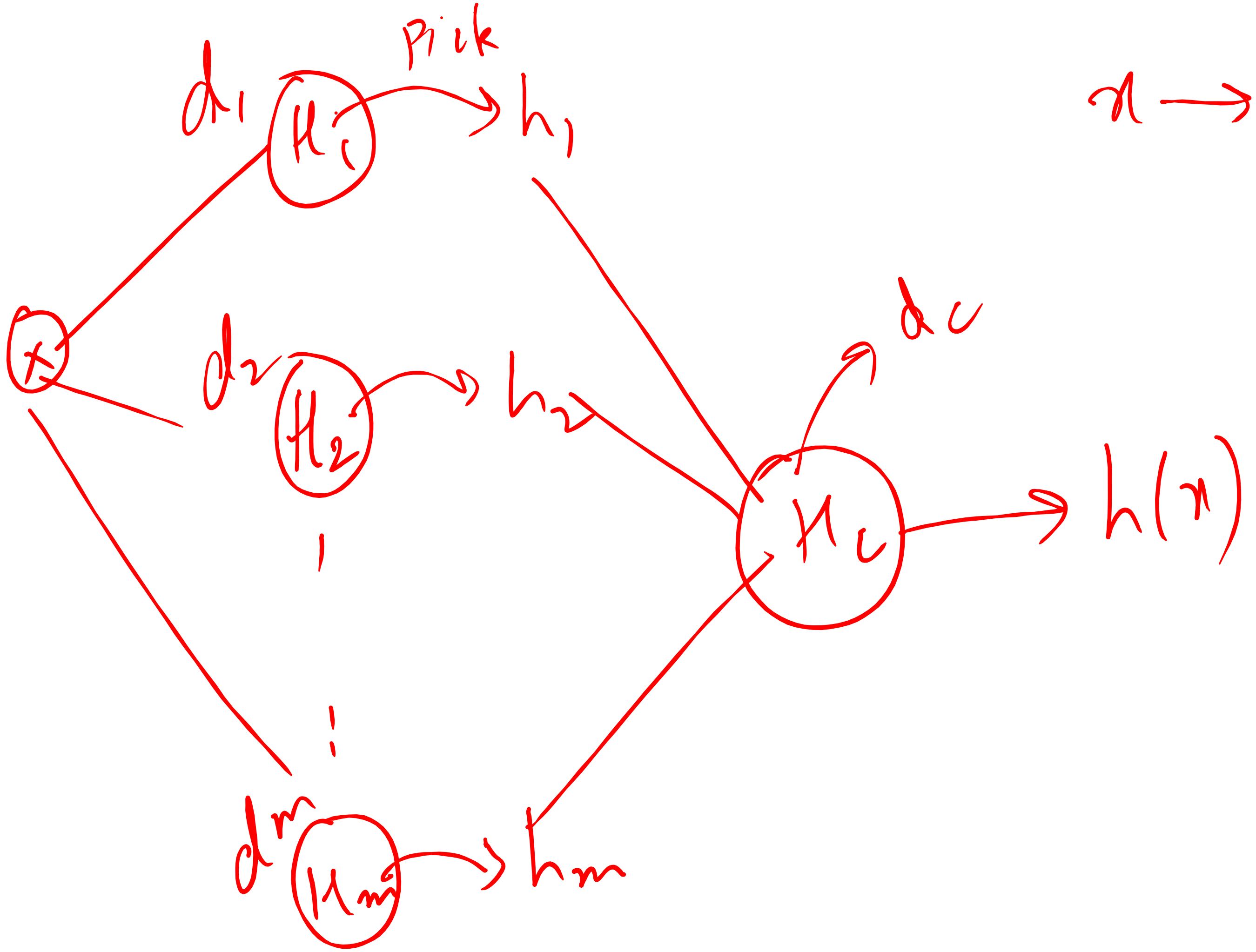
$$E_{in} \simeq \frac{1}{N^{1/4}}$$

$$E_{out}^{*} \qquad m = \sqrt{N}$$

$$E_{out} \simeq 0 \qquad \boxed{m}$$

Semi-parametric method.

<u>Validation</u>



$$x \to (h_1(x) \; h_2(x) \; \ldots \; h_m(x))$$

$$h(x) = h_c(h_1(x) \; h_2(x) \; \cdots \; h_m(x))$$

# of dichotomies

$$
\begin{array}{lcl}
x_1 \rightarrow & \big(h_1(x_1) \quad h_2(x_1) \;\cdots\; h_m(x_1)\big) & \rightarrow z_1 \\
x_2 \rightarrow & \big(h_1(x_2) \quad h_2(x_2) \;\cdots\; h_m(x_2)\big) & \rightarrow z_2 \\
\vdots & & \\
x_N \rightarrow & \big(h_1(x_N) \quad h_2(x_N) \;\cdots\; h_m(x_N)\big) & \rightarrow z_m \\
 & N^{d_1} \quad N^{d_2} \;\cdots\; N^{d_m} &
\end{array}
$$

# of configurations of z's $\leq N^{d_1 + d_2 \cdots + d_m}$

# of dichotomies $\leq N^{\underbrace{d_c + \sum_{i=1}^{m} d_i}_{D}}$

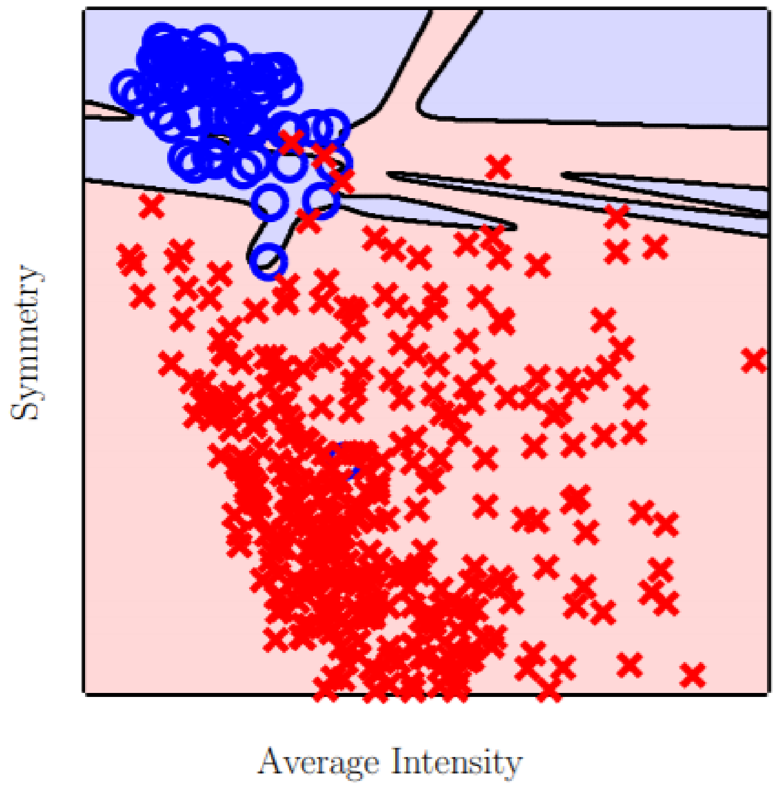
Lemma: $N^D \le 2^N$, $N \in \Omega(D \log D)$

$N > 2D \log_2 D$ then $m(N) < 2^N$ & $d_{VC} \le 2D \log D$

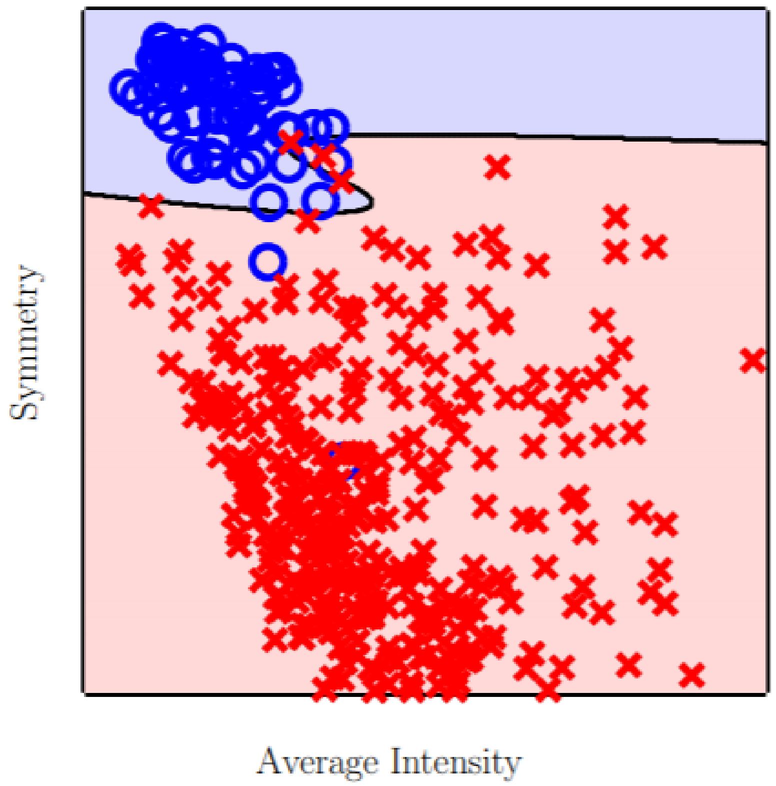
$O\left(\sum d_i + d_c\right) \log\left(\sum d_i + d_c\right)$

Weight Decay with Digits Data

No Weight Decay



Weight Decay, $\lambda = 0.01$



$$E_{aug} = E_{in} + \frac{\lambda}{N} w^T w$$

$$W = \{W^{(1)}, W^{(2)} \ldots W^{(l)}\}$$

$$E_{aug} = E_{in} + \frac{\lambda}{N} \sum_{l_{ij}} \left(w_{ij}^{(l)}\right)^2$$

$$\frac{\partial E_{aug}}{\partial W_{ij}^{(l)}} = \frac{\partial E_{in}}{\partial w_{ij}^{(l)}} + \frac{\lambda}{N} 2 \cdot w_{ij}^{(l)}$$

$$\rightarrow \frac{\partial E_{aug}}{\partial W^{(l)}} = \frac{\partial E_{in}}{\partial W^{(l)}} + \frac{2\lambda}{N} W^{(l)}$$

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial E_{in}}{\partial W^{(l)}} - \frac{2\lambda}{N} W^{(l)}$$

Back propagation

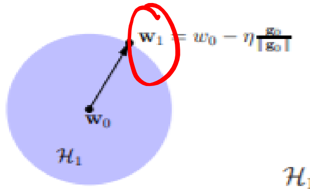
Hypothesis set

$$H_1 \subseteq H_2 \subseteq H_3 \cdots \subseteq H_m$$

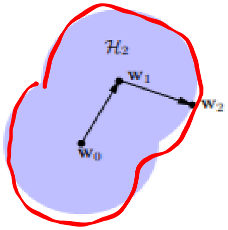
$$d_1 \leq d_2 \leq d_3 \cdots \leq d_m .$$

Early Stopping

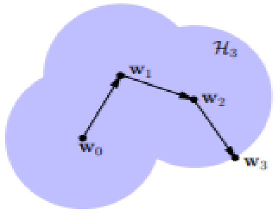
Gradient Descent



$\mathcal{H}_1 = \{\mathbf{w} : \ \|\mathbf{w} - \mathbf{w}_0\| \leq \eta\}$



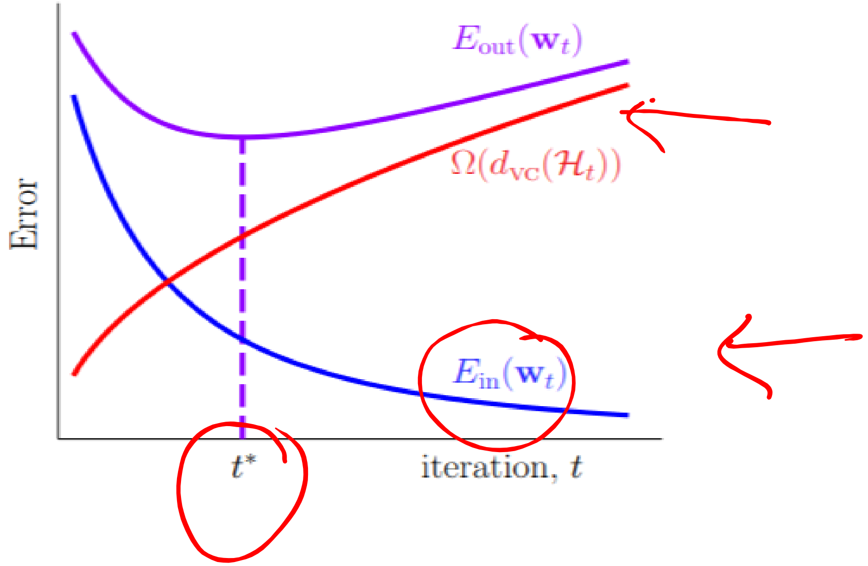
$\mathcal{H}_2 = \mathcal{H}_1 \cup \{\mathbf{w} : \ \|\mathbf{w} - \mathbf{w}_1\| \leq \eta\}$

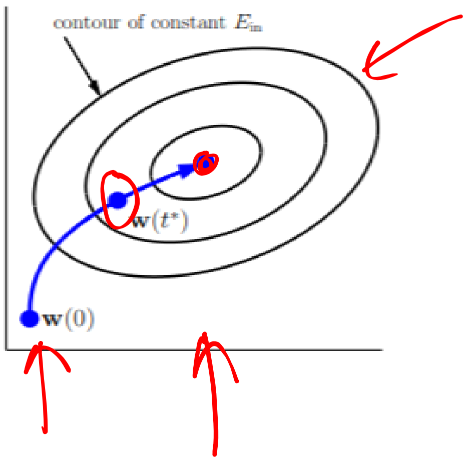


$\mathcal{H}_3 = \mathcal{H}_2 \cup \{\mathbf{w} : \ \|\mathbf{w} - \mathbf{w}_2\| \leq \eta\}$

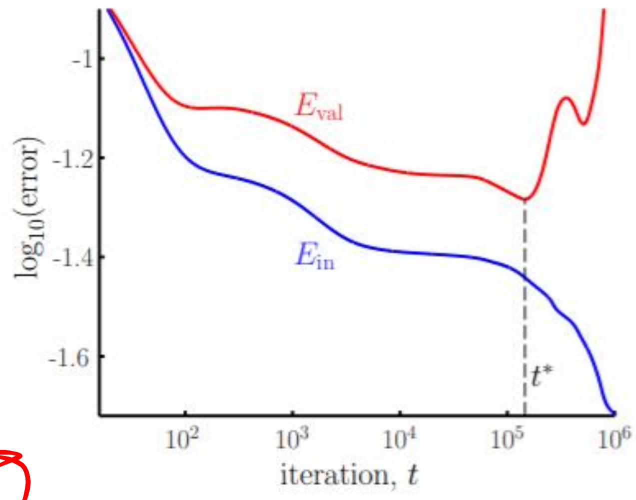
Each iteration explores a larger $\mathcal{H}$

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3 \subset \mathcal{H}_4 \subset \cdots$$

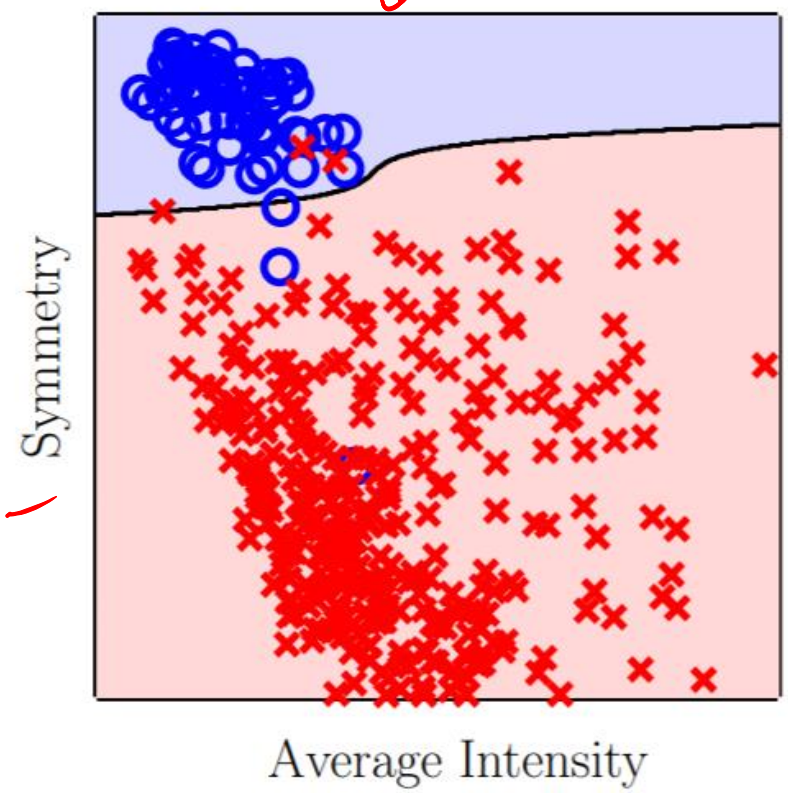




Early Stopping on Digits Data

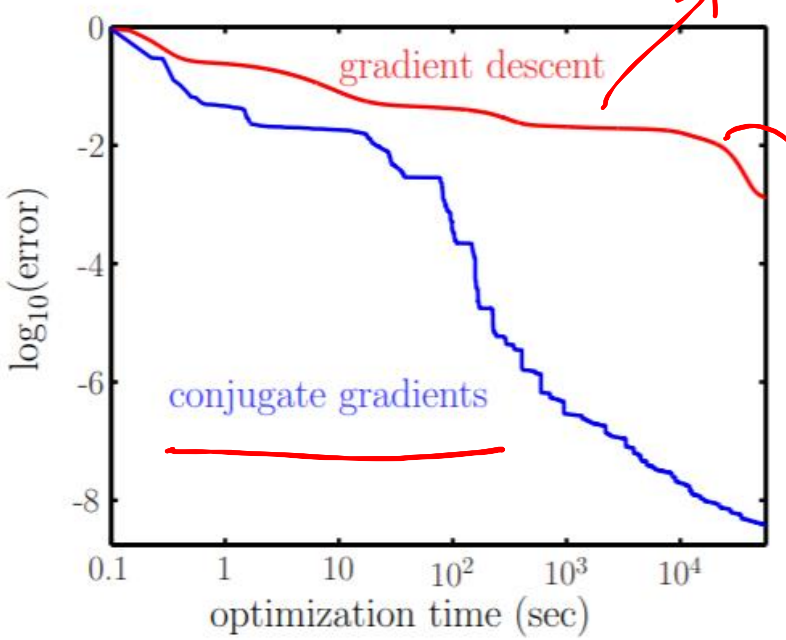


Use a validation set to determine t^*
Output $\mathbf{w}^*$, do not retrain with all the data till t^*.



Minimizing E_{in}

1. Use regression for classification
2. Use better algorithms than gradient descent



$s^L \to \theta \to h(x)$

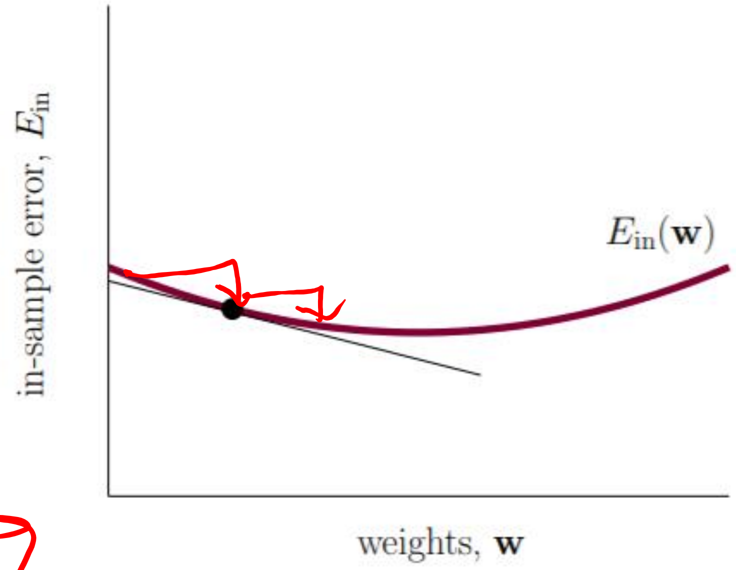
$= \theta(s^L)$

sign or tanh

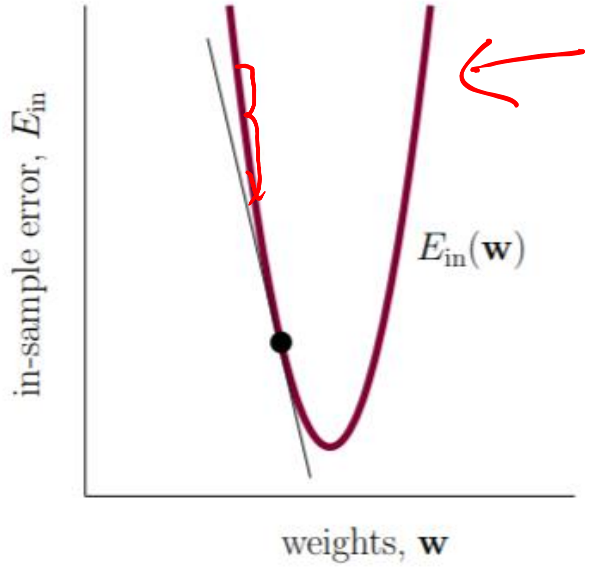
fast

i) Direction (-ve of the grad.)
ii) step size (η)

Determine the gradient $\mathbf{g}$



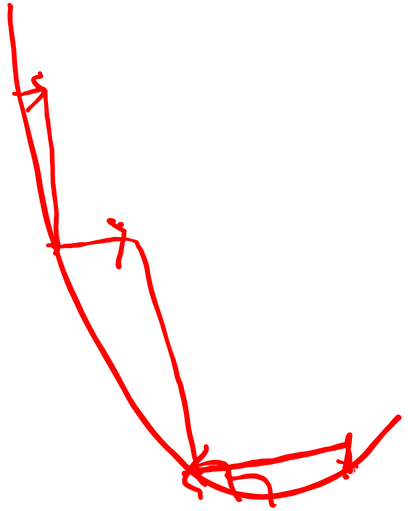
Shallow: use large η.



Deep: use small η.

1) Start, take a step, check if E_{in} improves.

$\eta \rightarrow \eta \times$ some factor $(\beta = 1.03)$



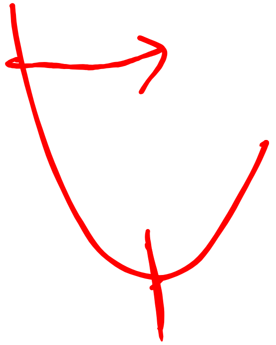
2) Take a bigger step & improve E_{in}.

3) If you worsen E_{in} then

$\eta \rightarrow \eta \times \alpha \; (< 1)$

You may backtrack.

Variable $\eta \rightarrow$ GD



Steepest Descent - Line Search

1: Initialize $w(0)$ and set $t = 0$;

2: **while** stopping criterion has not been met **do**

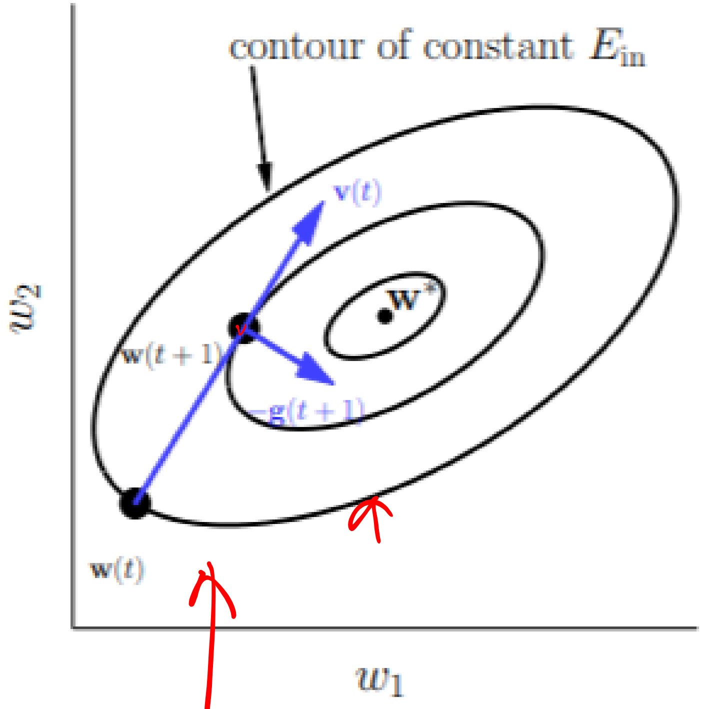
3: Let $\mathbf{g}(t) = \nabla E_{\text{in}}(\mathbf{w}(t))$, and set $\mathbf{v}(t) = -\mathbf{g}(t)$.

4: Let $\eta^* = \text{argmin}_\eta\, E_{\text{in}}(\mathbf{w}(t) + \eta\mathbf{v}(t))$.

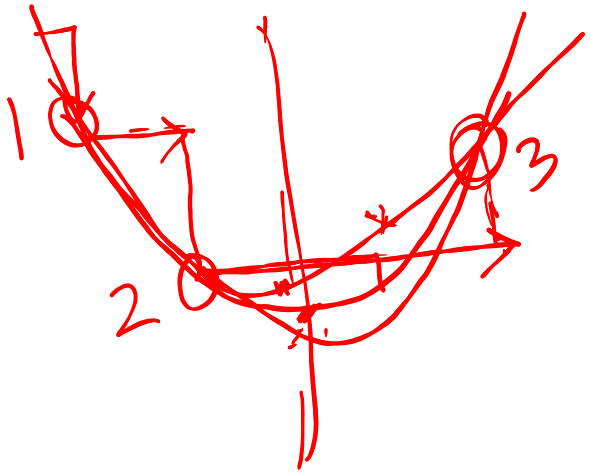
5: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta^*\mathbf{v}(t)$.

6: Iterate to the next step, $t \leftarrow t + 1$.

7: **end while**



How to accomplish the line search (step 4)?
Simple bisection (binary search) suffices in practice



- U shaped

Binary search

In practice 5-10 (ε)

Method	Optimization Time		
	10 sec	1,000 sec	50,000 sec
Gradient Descent	0.079	0.0206	0.00144
Stochastic Gradient Descent	**0.0213**	**0.00278**	0.000022
Variable Learning Rate	0.039	0.014	0.00010
Steepest Descent	0.043	0.0189	**0.000012**

SGD → x_n
→ small steps.

Steepest Descent → line search

grad for E_{in}

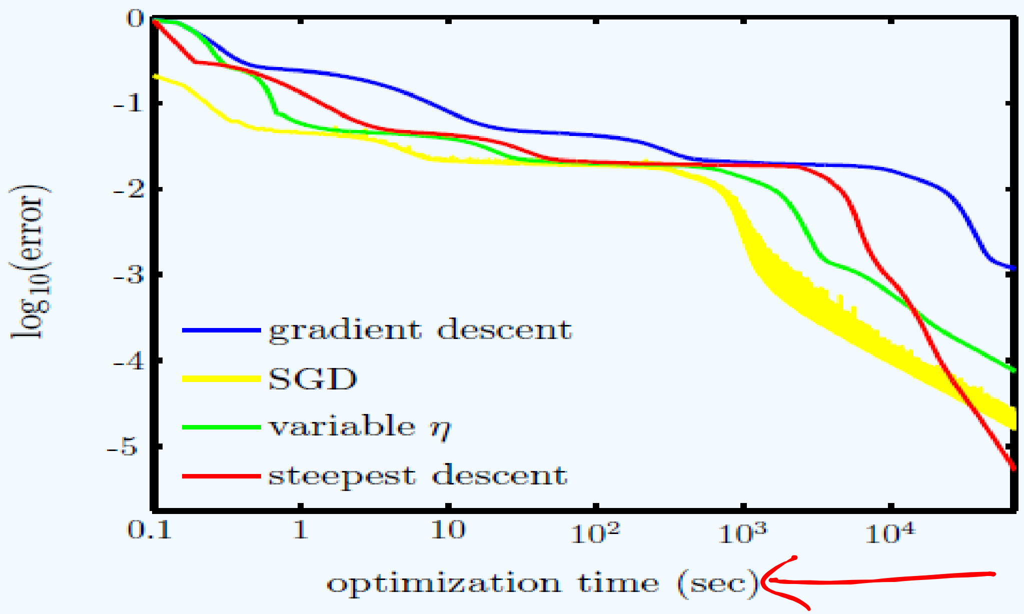
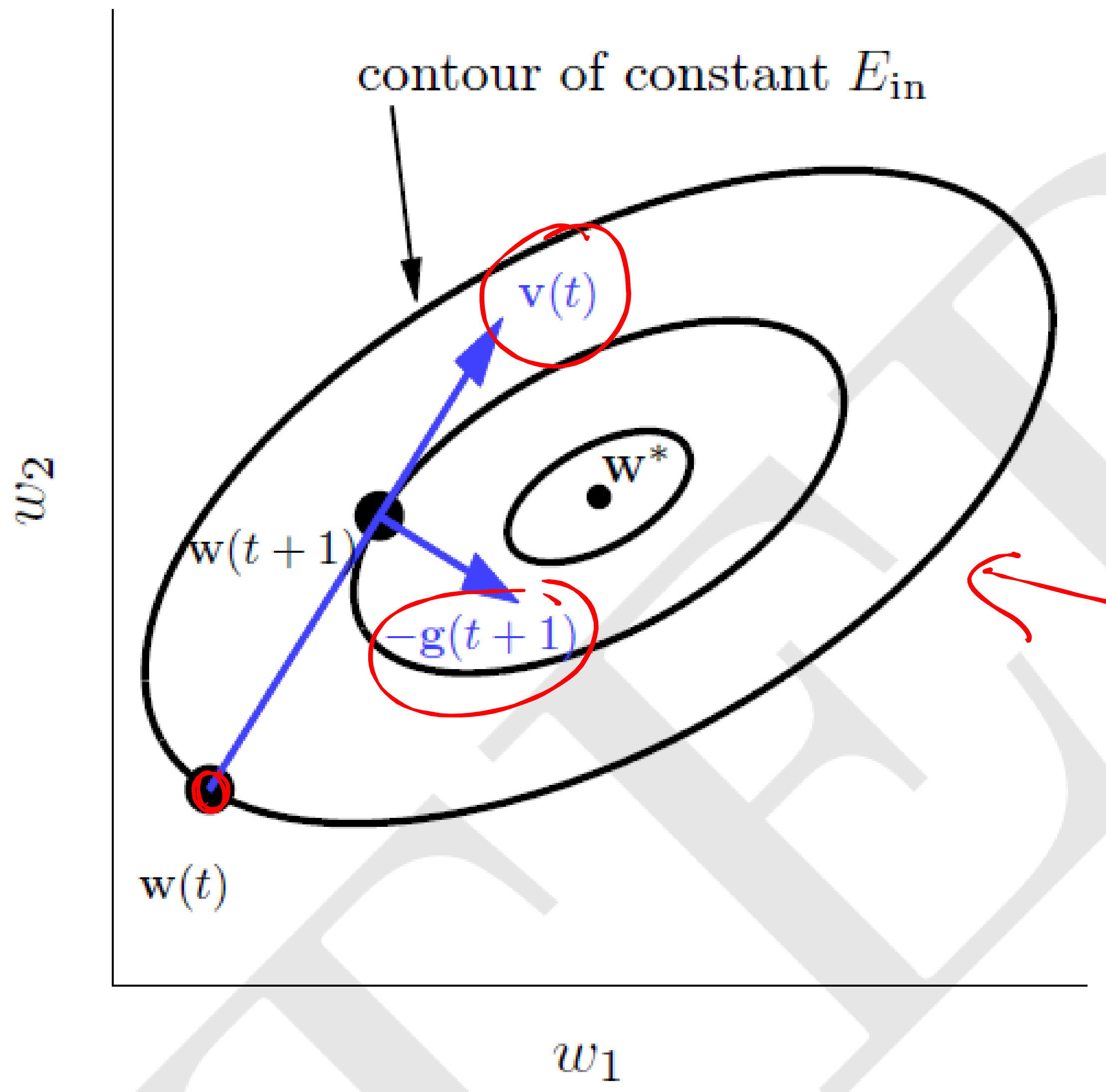
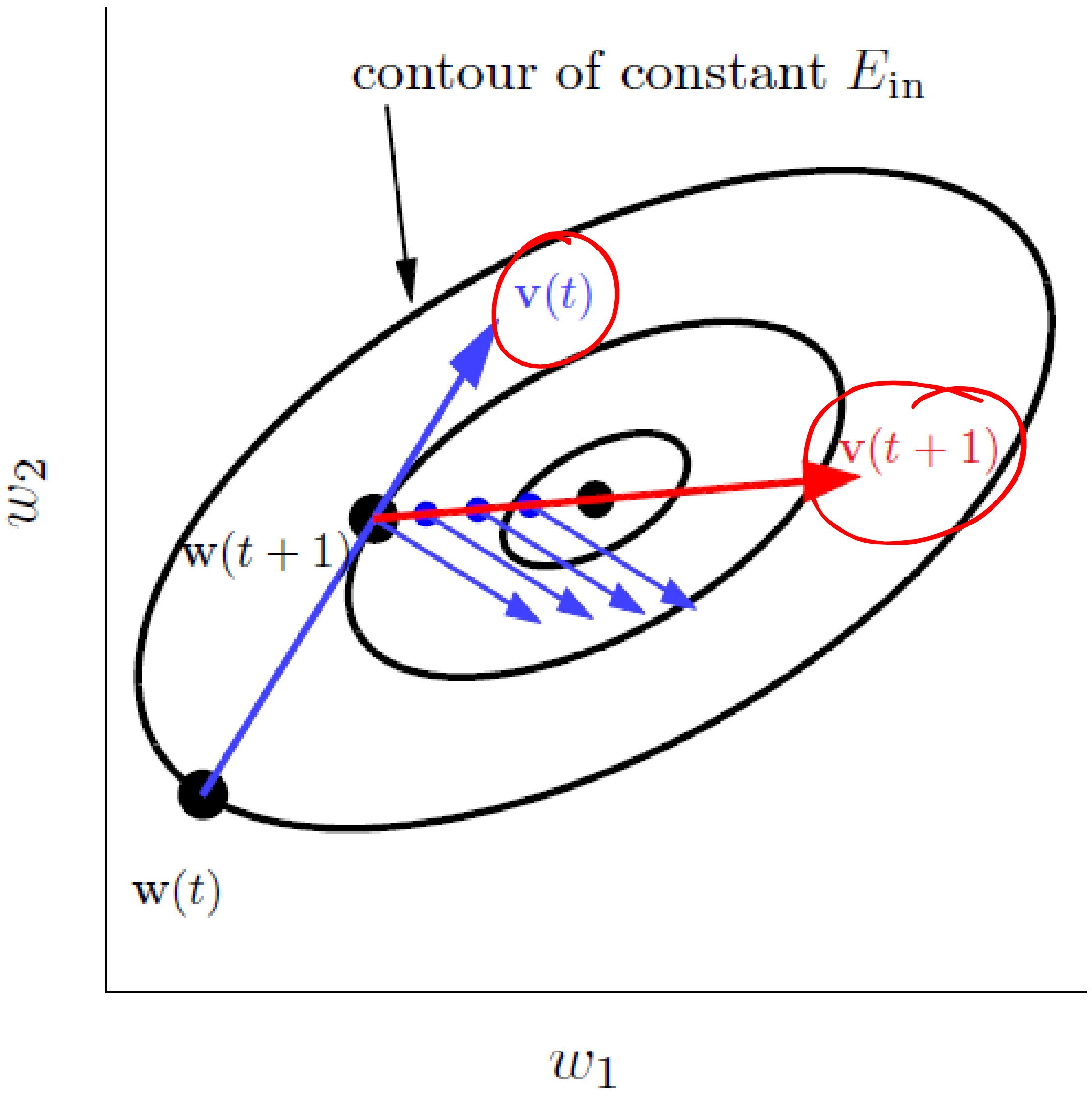


Figure 7.4: Gradient descent, variable learning rate and steepest descent using digits data and a 5 hidden unit 2-layer neural network with linear output. For variable learning rate, $\alpha = 1.1$ and $\beta = 0.8$.



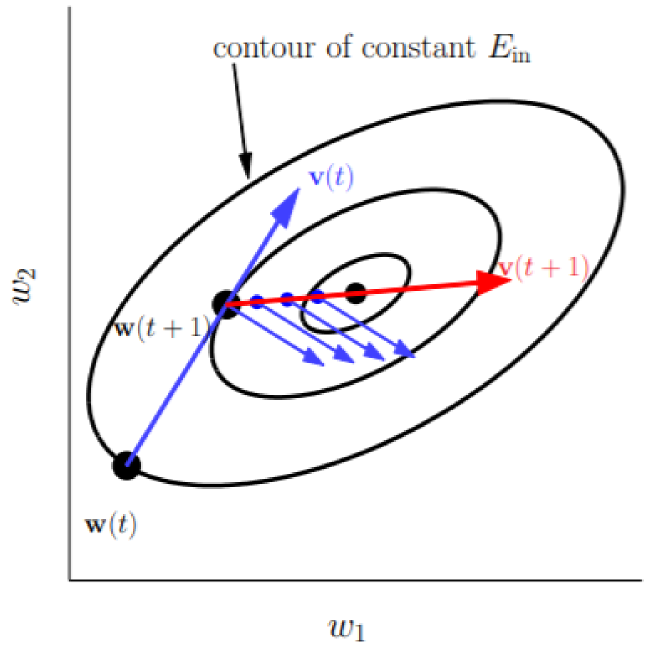
new grad ⊥ prev. line search dir

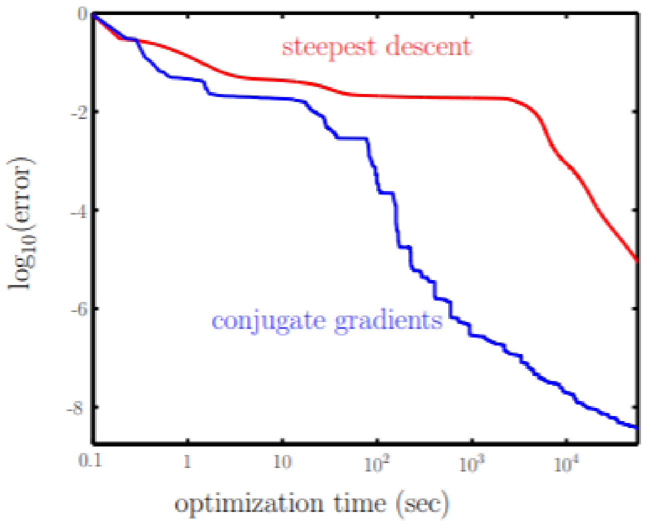

contour of constant E_{in}
$\mathbf{v}(t)$
$\mathbf{v}(t+1)$
$\mathbf{w}(t+1)$
$\mathbf{w}(t)$
w_2
w_1

d-steps.

Conjugate Gradients

1. Line search just like steepest descent.
2. Choose a better direction than $-\mathbf{g}$





Method	Optimization Time		
	10 sec	1,000 sec	50,000 sec
Stochastic Gradient Descent	0.0203	0.000447	1.6310×10^{-5}
Steepest Descent	0.0497	0.0194	0.000140
Conjugate Gradients	**0.0200**	$\mathbf{1.13 \times 10^{-6}}$	$\mathbf{2.73 \times 10^{-9}}$

Thanks!