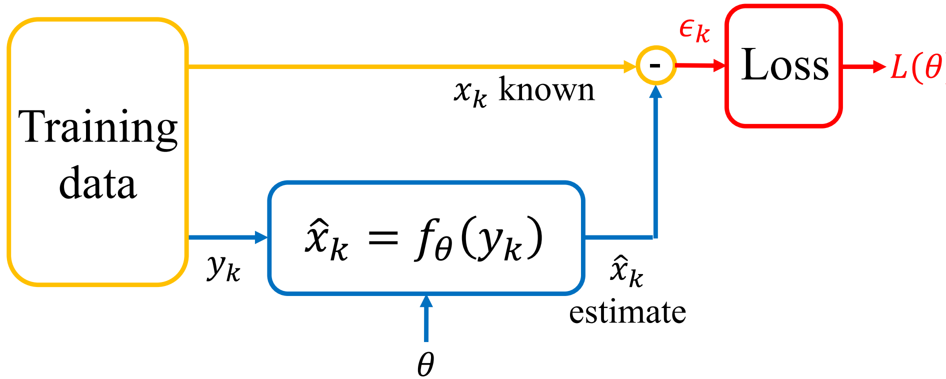
Gradient Descent Optimization

- Definition
- Mathematical calculation of gradient
- Matrix interpretation of gradient computation

Minimizing Loss



- In order to train, we need to minimize loss

$$\theta^* = \arg\min_\theta \{L(\theta)\}$$

  - How do we do this?

- Key ideas:
  - Use gradient descent
  - Computing gradient using chain rule, adjoint gradient, back propagation.

What is Gradient Descent?

- Gradient descent:
  - The simplest (but surprisingly effective) approach
  - Move directly down hill
- What is the down hill direction?

$$d = -\nabla L(\theta)$$

$\boxed{d \quad 1 \times P}$

gradient is a row vector

- Gradient descent algorithm

Repeat until converged {

$$d \leftarrow -\nabla L(\theta)$$

$$\theta \leftarrow \theta + \alpha d^{\boxed{t}}$$

}

transpose

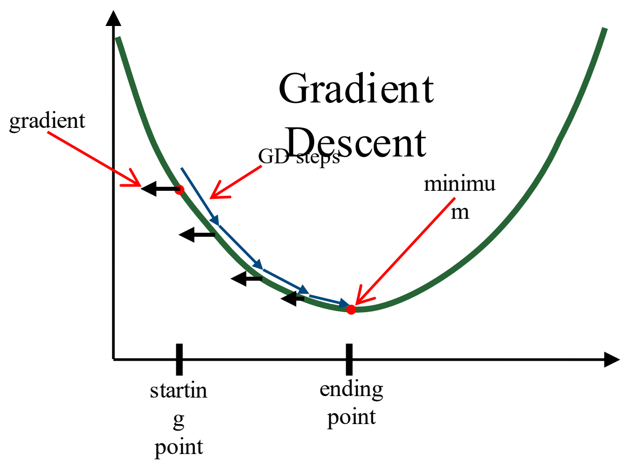
Gradient Descent (GD) Algorithm

Gradient Descent Picture

- The GD update step:

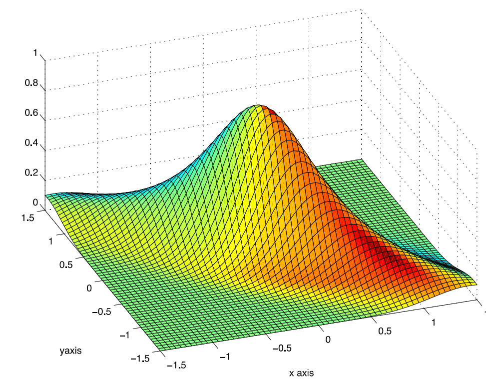
$$d \leftarrow -\nabla L(\theta)$$
$$\theta \leftarrow \theta + \alpha d^t$$

<u>1D case</u>

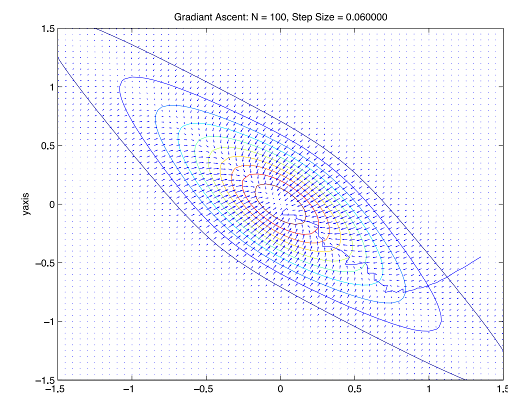


<u>2D case</u>

Function

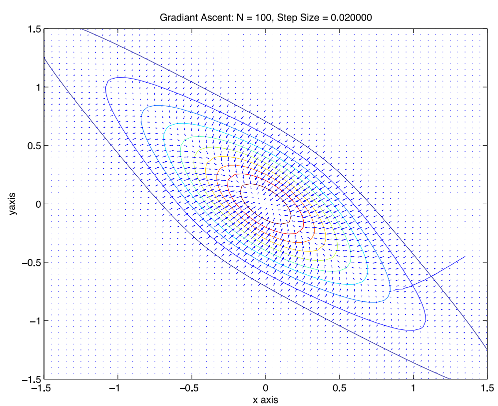


Updates

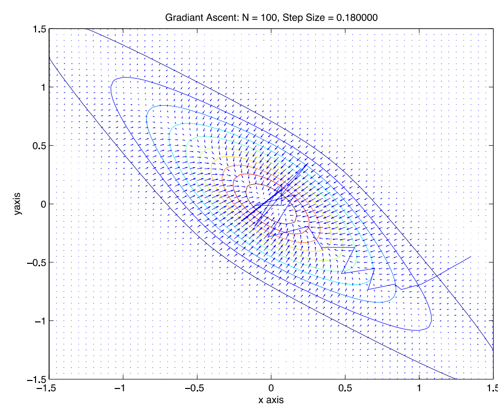


Gradient Step Size

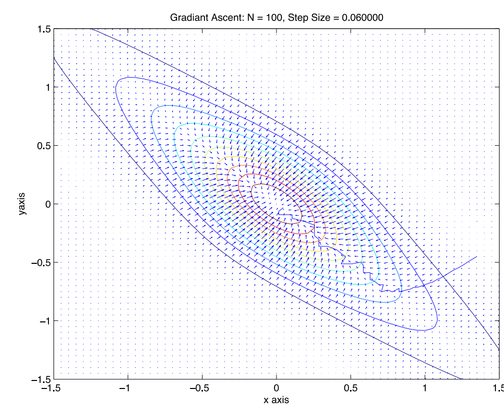
- How large should α be?
  - α too small $\Rightarrow$ slow convergence
  - α too large $\Rightarrow$ unstable
  - Often there is no good choice!



Too small $\Rightarrow$ slow convergence



Too large $\Rightarrow$ oscilate



Goldilocks? $\Rightarrow$ good enough

Steepest Descent

- Use line search to compute the best α

Repeat until converged {

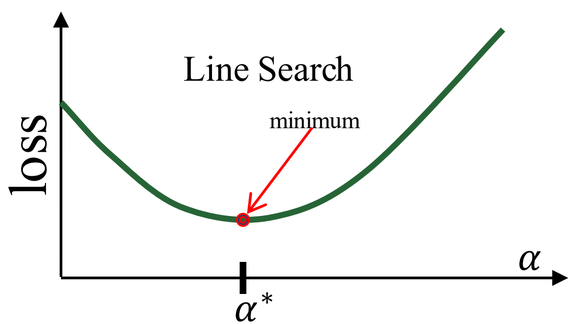
$$d \leftarrow -\nabla L(\theta)$$

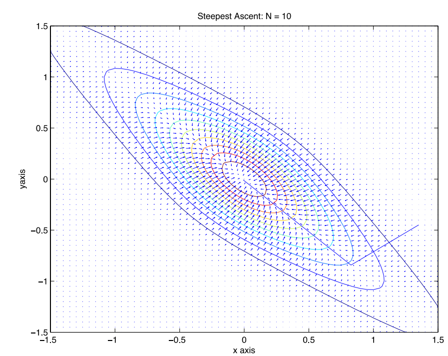
$$\alpha^* \leftarrow \arg\min_{\alpha}\{L(\theta + \alpha d^t)\}$$

$$\theta \leftarrow \theta + \alpha^* d^t$$

}

Steepest Descent Algorithm

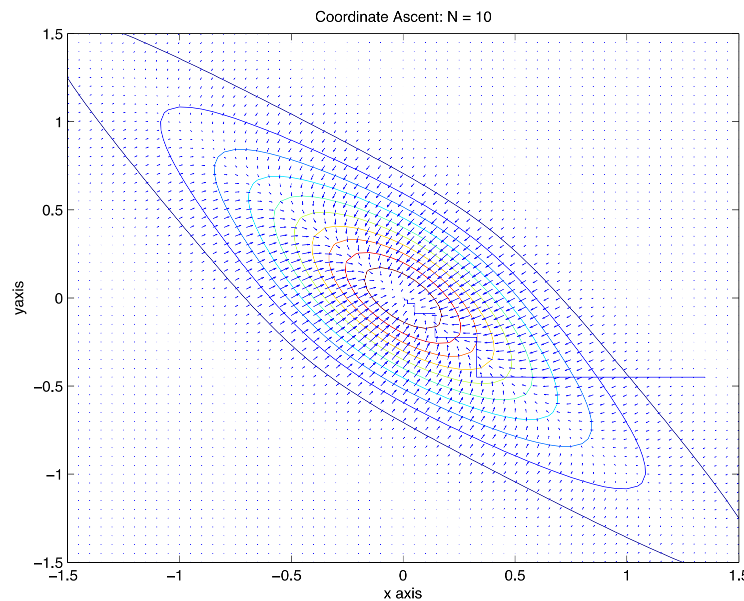




Steepest Descent ⇒ Too Expensive

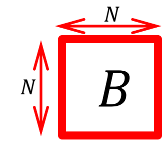
Coordinate Descent

- Update one parameter at a time
  - Reduces problem of selecting step size
  - Each update can be very fast, but lots of updates



SVD and Eigen Decomposition

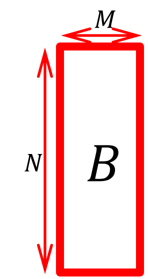
- A matrix B can have the following properties:
  - B is symmetric $\Leftrightarrow B = B^t$
  - B is positive definite $\Leftrightarrow \forall x \in \Re^N \; x \neq 0, \; x^t B x > 0$
  - B is non-negative definite $\Leftrightarrow \forall x \in \Re^N, \; x^t B x \geq 0$



- For any $B \in \Re^{N \times M}$ where $N \geq M$:
  - Singular Value Decomposition (SVD) is given by

$$B = U \Sigma V^t$$

  - where $\Sigma = \mathrm{diag}(\sigma_0, \cdots, \sigma_{M-1})$ are the singular values;
  - $U \in \Re^{N \times M}$ is unitary; $V \in \Re^{M \times M}$ is unitary.



- For any <u>symmetric</u> $B \in \Re^{N \times N}$:
  - Eigen Decomposition is given by

$$B = U \Sigma U^t$$

  - where $\Sigma = \mathrm{diag}(\sigma_0, \cdots, \sigma_{M-1})$ are the eigenvalues;
  - $U \in \Re^{N \times N}$ is unitary.

The Condition Number

- Consider a quadratic loss function with the form:

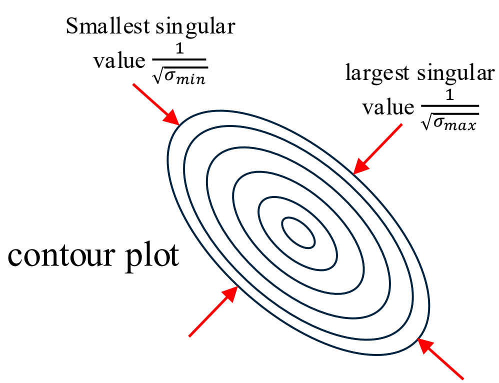
$$loss(\theta) = (\theta - \theta^*)^t B(\theta - \theta^*)$$

  - where $B = U\Sigma V^t$ and $\Sigma = diag(\sigma_0, \cdots, \sigma_{p-1})$ are the singular values of B.

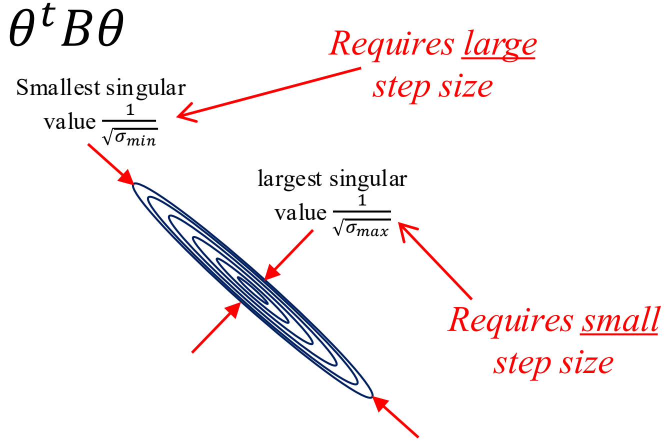
- Then
  - Condition number $= \frac{\sigma_{max}}{\sigma_{min}}$
  - $\theta^* = \arg\min_\theta loss(\theta)$

- Without loss of generality, we assume $\theta^* = \mathbf{0}$, so

$$loss(\theta) = \theta^t B\theta$$

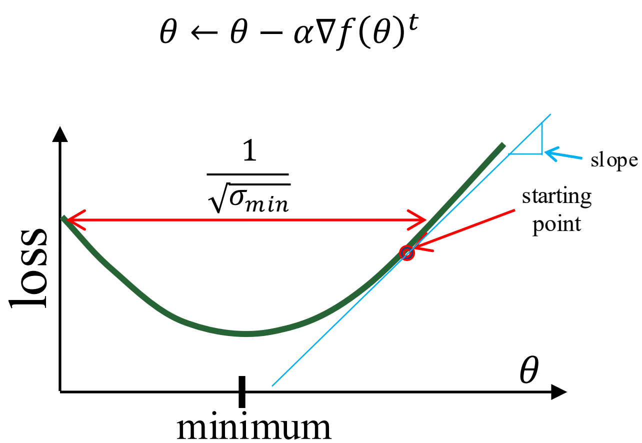


Condition Number $\frac{\sigma_{max}}{\sigma_{min}} = 4$

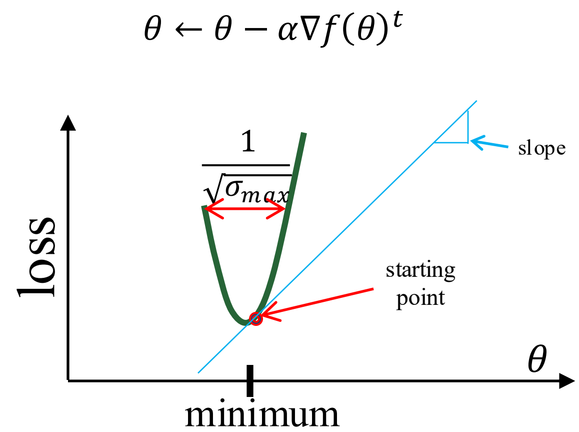


Condition Number $\frac{\sigma_{max}}{\sigma_{min}} = 100$

1D Step Size Choice



Requires <u>large</u> step size α

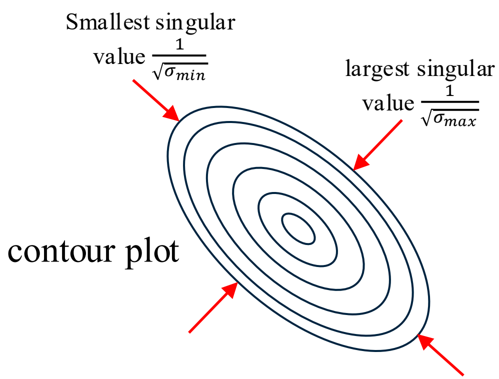


Requires <u>small</u> step size α

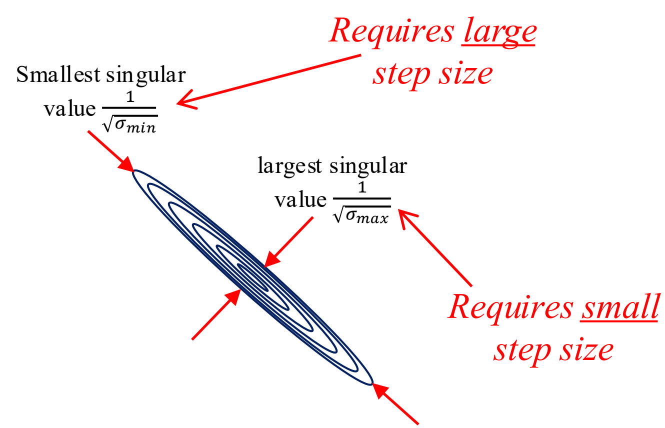
- Step size depends on second derivative
  - Small second derivative ⇒ large step size
  - Large second derivative ⇒ small step size

Slow Convergence of Gradient Descent

- Very sensitive to <u>condition number</u> of problem
  - No good choice of step size
- Newton's method: Correct for local second derivative
  - "Sphere" the ellipse
  - Too much computation; Too difficult to implement
- Alternative methods
  - Preconditioning: Easy, but tends to be ad-hoc, not so robust
  - Momentum: More latter



Condition Number $\frac{\sigma_{max}}{\sigma_{min}} = 4$



Condition Number $\frac{\sigma_{max}}{\sigma_{min}} = 100$

MSE Loss Function

- Interpretation of MSE Loss Function

$$L_{MSE}(\theta) = \underbrace{\frac{1}{K}\sum_{k=0}^{K-1}}_{\textit{sum over training data}} \|x_k - \underbrace{f_\theta(y_k)}_{\textit{inference function}}\|^2$$

Computing the MSE Loss Gradient

- Use chain rule to compute the loss gradient

$$
\begin{aligned}
\nabla_\theta L_{MSE}(\theta) &= \nabla_\theta \left\{ \frac{1}{K} \sum_{k=0}^{K-1} \| x_k - f_\theta(y_k) \|^2 \right\} \\
&= \frac{1}{K} \sum_{k=0}^{K-1} \nabla_\theta \{ \| x_k - f_\theta(y_k) \|^2 \} \\
&= \frac{2}{K} \sum_{k=0}^{K-1} \big(x_k - f_\theta(y_k) \big)^t \, \nabla_\theta \big(x_k - f_\theta(y_k) \big) \\
&= \frac{-2}{K} \sum_{k=0}^{K-1} \big(x_k - f_\theta(y_k) \big)^t \, \nabla_\theta f_\theta(y_k)
\end{aligned}
$$

What does this mean?

Interpretation of Loss Gradient

$$-\nabla_\theta L_{MSE}(\theta) = \underbrace{\frac{2}{K}\sum_{k=0}^{K-1}}_{\text{sum over training data}} \underbrace{\left(x_k - f_\theta(y_k)\right)^t}_{\text{prediction error}} \underbrace{\nabla_\theta f_\theta(y_k)}_{\text{gradient of function}}$$

- Loss Gradient computation requires:
  - Sum over training data: Big sum, but straight forward.
  - Prediction error: Easy to compute.
  - Gradient of inference function: <u>This is the difficult part</u>.
    - Most challenging part to compute.
    - Enabled by automatic differentiation built into modern domain specific languages (DSL) such as Pytorch, Tensorflow, and others.
    - For NN this is known as back propagation.

Matrix Interpretation

- Since $x_k = f_\theta(y_k) + error$, we have that

$$\boxed{\begin{matrix} 1 \times N_x \\ \text{error vector} \end{matrix}} = \epsilon_k^t = \left(x_k - f_\theta(y_k)\right)^t = \text{error vector}$$

- Then the parameter vector is given by

$$\boxed{\begin{matrix} P \times 1 \\ \text{parameter vector} \end{matrix}} = [-\nabla_\theta L_{MSE}]^t = \text{dimension of parameter vector}$$

Function Gradient

- Inference function gradient, $\nabla f_\theta(y_k)$, is given by

parameter dimension

j
index

output dimension

i
index

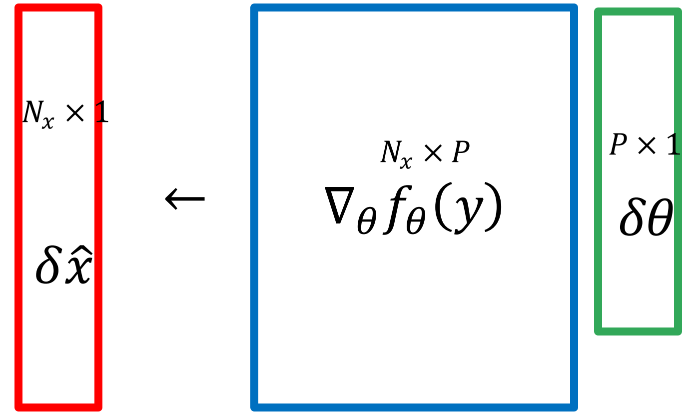
$N_x \times P$
function gradient

$$\frac{\partial[f_\theta(y_k)]_i}{\partial \theta_j} = \nabla_\theta f_\theta(y_k) = \text{gradient of function}$$

Forward vs Backward Propogation

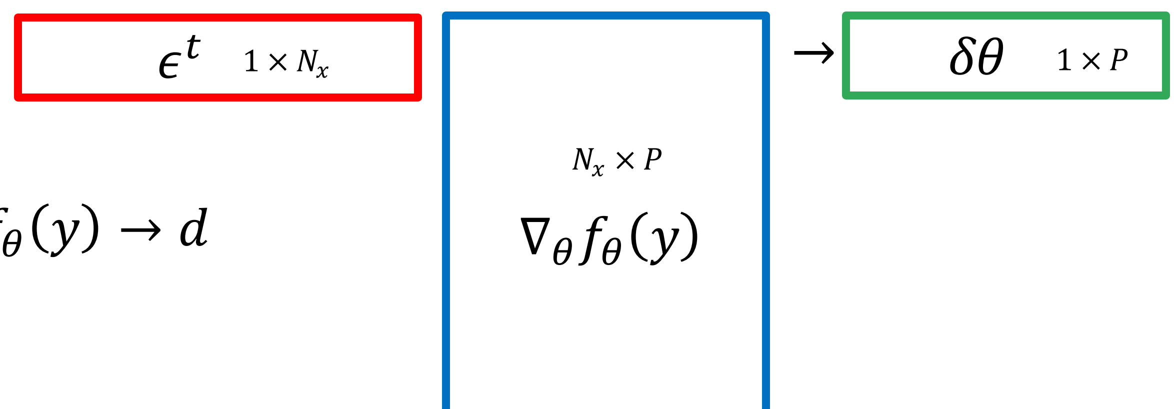
- Forward gradient

$$\delta\hat{x} \leftarrow \nabla_\theta f_\theta(y)\ \delta\theta$$



- Backward (adjoint) gradient

$$\epsilon^t\ \nabla_\theta f_\theta(y) \rightarrow d$$



Loss Gradient Computation

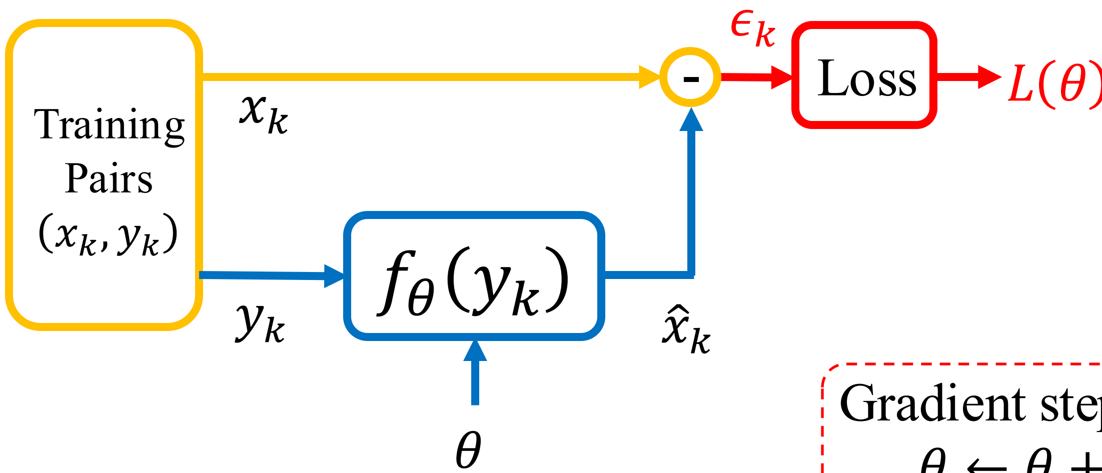
- Equation is

$$-\nabla_\theta L_{MSE}(\theta) = \frac{2}{K}\sum_{k=0}^{N-1} \overbrace{\boxed{\left(x_k - f_\theta(y_k)\right)^t}}^{\textit{prediction error}} \underbrace{\boxed{\nabla_\theta f_\theta(y_k)}}_{\textit{gradient of function}}$$

- Looks like

$$\boxed{d \quad {\scriptstyle 1\times P}} = \frac{2}{K}\sum_{k=0}^{K-1} \boxed{\epsilon_k^t \quad {\scriptstyle 1\times N_x}} \boxed{\begin{matrix} {\scriptstyle N_x \times P} \\ \nabla_\theta f_\theta(y_k) \end{matrix}}$$

Update Direction for Supervised Training

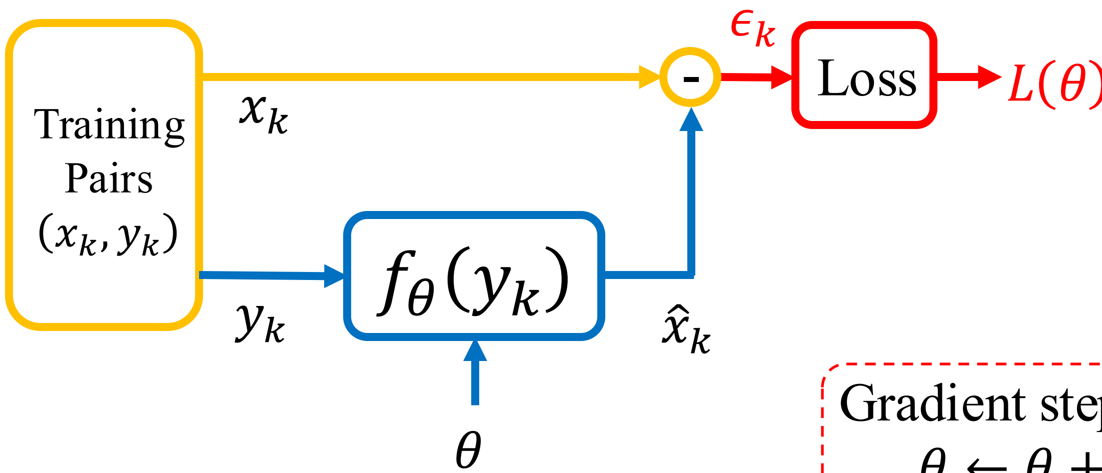


Gradient step:
$$\theta \leftarrow \theta + \alpha d^t$$

- d is given by

$$\boxed{\underset{1\times P}{d}} = \frac{2}{K}\sum_{k=0}^{K-1} \boxed{\underset{1\times N_x}{\epsilon_k^t}}\ \boxed{\underset{N_x\times P}{\nabla_\theta f_\theta(y_k)}}$$

Update Direction for Supervised Training



Gradient step:

$$\theta \leftarrow \theta + \alpha d^t$$

- d^t is given by

$$\underbrace{d^t}_{P \times 1} = \frac{2}{K} \sum_{k=0}^{K-1} \underbrace{[\nabla_\theta f_\theta(y_k)]^t}_{P \times N_x} \underbrace{\epsilon_k}_{N_x \times 1}$$

Multiply error by the adjoint gradient