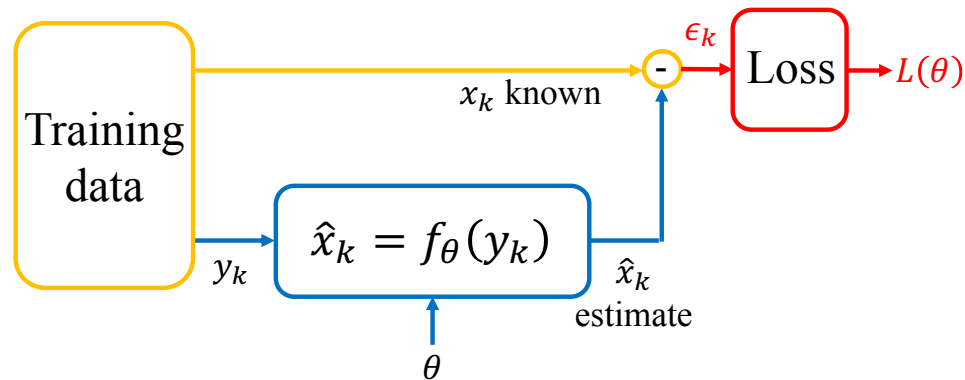


# Gradient Descent Optimization

- 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?

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

$$d \quad 1 \times P$$

gradient is a row vector

- Gradient descent algorithm

Repeat until converged {

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

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

}

*Gradient Descent (GD) Algorithm*

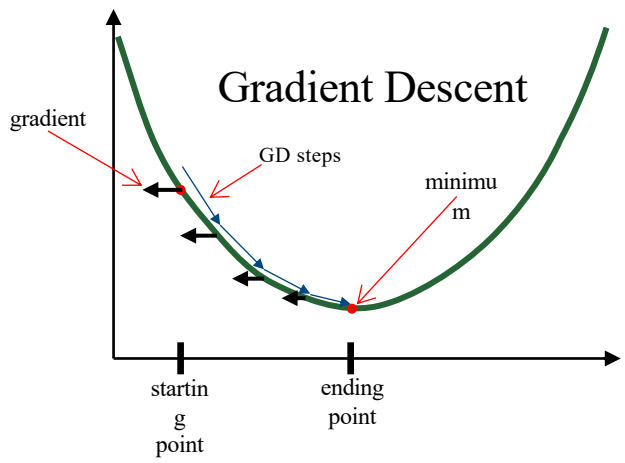
*transpose*

# Gradient Descent Picture

- The GD update step:

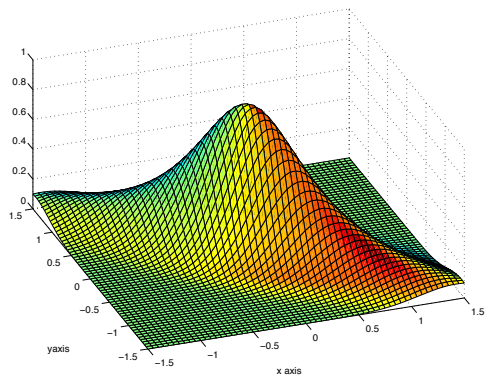
$$d \leftarrow -\nabla L(\theta)$$
$$\theta \leftarrow \theta + \alpha d^t$$

1D case

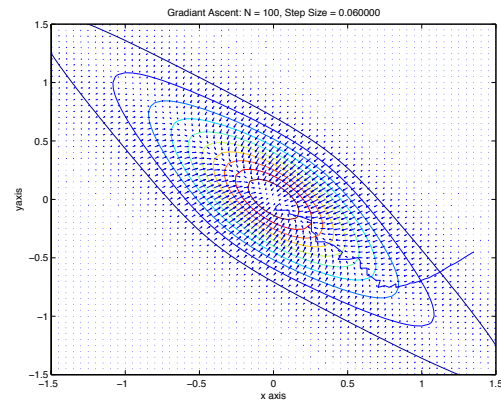


2D case

Function

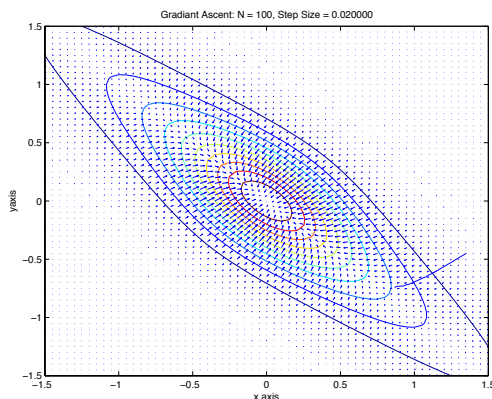


Updates

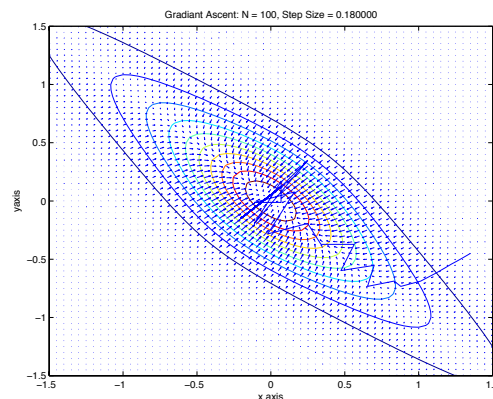


# Gradient Step Size

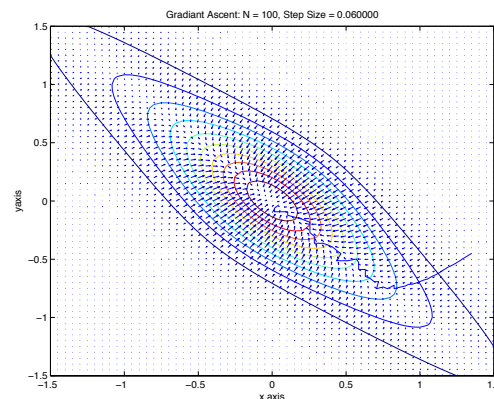
- How large should  $\alpha$  be?
  - $\alpha$  too small  $\Rightarrow$  slow convergence
  - $\alpha$  too large  $\Rightarrow$  unstable
  - Often there is no good choice!



Too small  $\Rightarrow$  slow  
convergence



Too large  $\Rightarrow$  oscillate



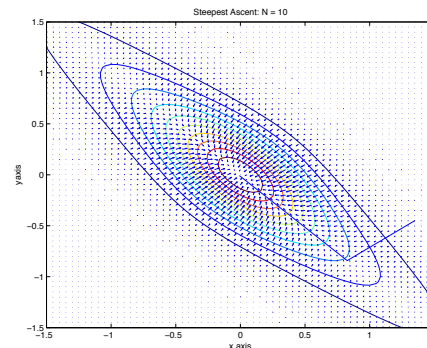
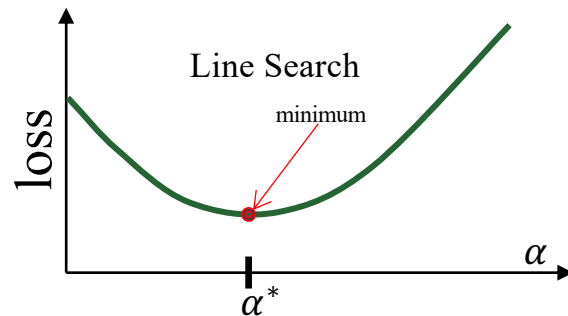
Goldilocks?  $\Rightarrow$  good enough

# Steepest Descent

- Use line search to compute the best  $\alpha$

Repeat until converged {  
 $d \leftarrow -\nabla L(\theta)$   
 $\alpha^* \leftarrow \arg \min_{\alpha} \{L(\theta + \alpha d^t)\}$   
 $\theta \leftarrow \theta + \alpha^* d^t$   
}

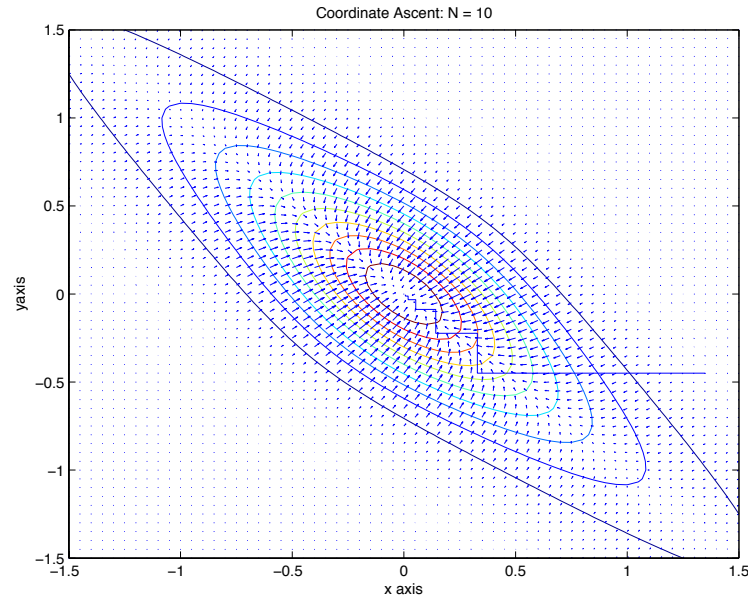
*Steepest Descent Algorithm*



Steepest Descent  $\Rightarrow$  Too Expensive

# Coordinate Descent

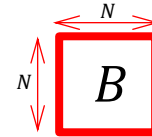
- Update one parameter at a time
  - Reduces problem of selecting step size
  - Each update can be very fast, but lots of updates



# SVD and Eigen Decomposition

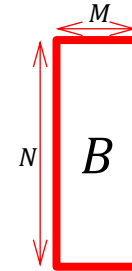
- A matrix  $B$  can have the following properties:

- $B$  is symmetric  $\Leftrightarrow B = B^t$
- $B$  is positive definite  $\Leftrightarrow \forall x \in \mathfrak{R}^N, x \neq 0, x^t B x > 0$
- $B$  is non-negative definite  $\Leftrightarrow \forall x \in \mathfrak{R}^N, x^t B x \geq 0$



- For any  $B \in \mathfrak{R}^{N \times M}$  where  $N \geq M$ :

- Singular Value Decomposition (SVD) is given by
$$B = U \Sigma V^t$$
- where  $\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{M-1})$  are the singular values;
- $U \in \mathfrak{R}^{N \times M}$  is unitary;  $V \in \mathfrak{R}^{M \times M}$  is unitary.



- For any symmetric  $B \in \mathfrak{R}^{N \times N}$ :

- Eigen Decomposition is given by
$$B = U \Sigma U^t$$
- where  $\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{M-1})$  are the eigenvalues;
- $U \in \mathfrak{R}^{N \times N}$  is unitary.

# Quadratic Forms

- Quadratic form:

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

- where  $B$  is positive definite.

- Facts:

- Without loss of generality,  $B = B^t$  is symmetric.
- The function is strictly positive, i.e.,  $\forall \theta, l(\theta) > 0$ .
- $l(\theta)$  takes on its minimum at  $\theta = \theta^*$ . So, we have that  $\theta^* = \arg \min_{\theta} l(\theta)$
- The singular/eigen decomposition is  $B = U \Sigma U^t$  where  $\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{p-1})$  are the singular/eigen values of  $B$ .
- $B$  positive definite  $\Leftrightarrow \sigma_i > 0$  for all  $i \in \{0, \dots, p - 1\}$

# The Condition Number

- Consider a quadratic loss function with the form:

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

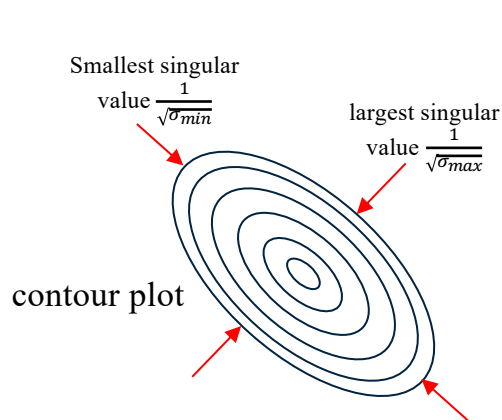
- where  $B = U\Sigma U^t$  and  $\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{p-1})$  are the singular/eigen values of  $B$ .

- Then

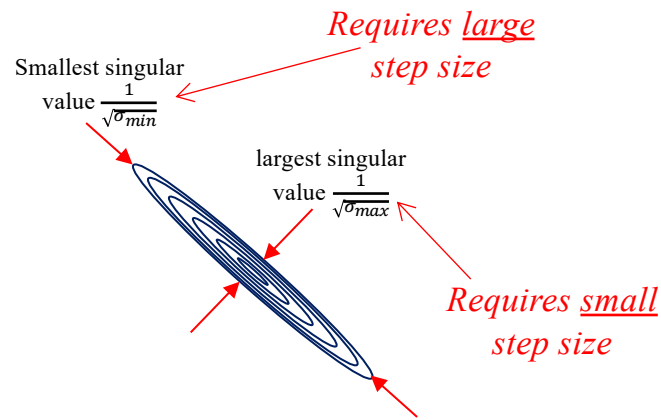
- Condition number =  $\frac{\sigma_{max}}{\sigma_{min}}$

- $\theta^* = \arg \min_{\theta} \text{loss}(\theta)$

- Without loss of generality, we assume  $\theta^* = \mathbf{0}$ , so  $l(\theta) = \theta^t B \theta$

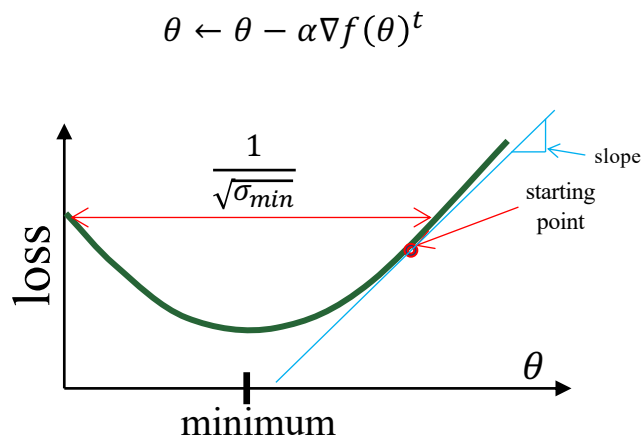


Condition Number  $\frac{\sigma_{max}}{\sigma_{min}} = 4$

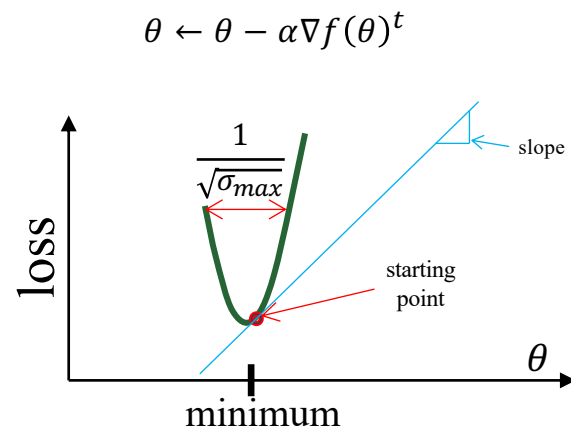


Condition Number  $\frac{\sigma_{max}}{\sigma_{min}} = 100$

# 1D Step Size Choice



Requires large step size  $\alpha$

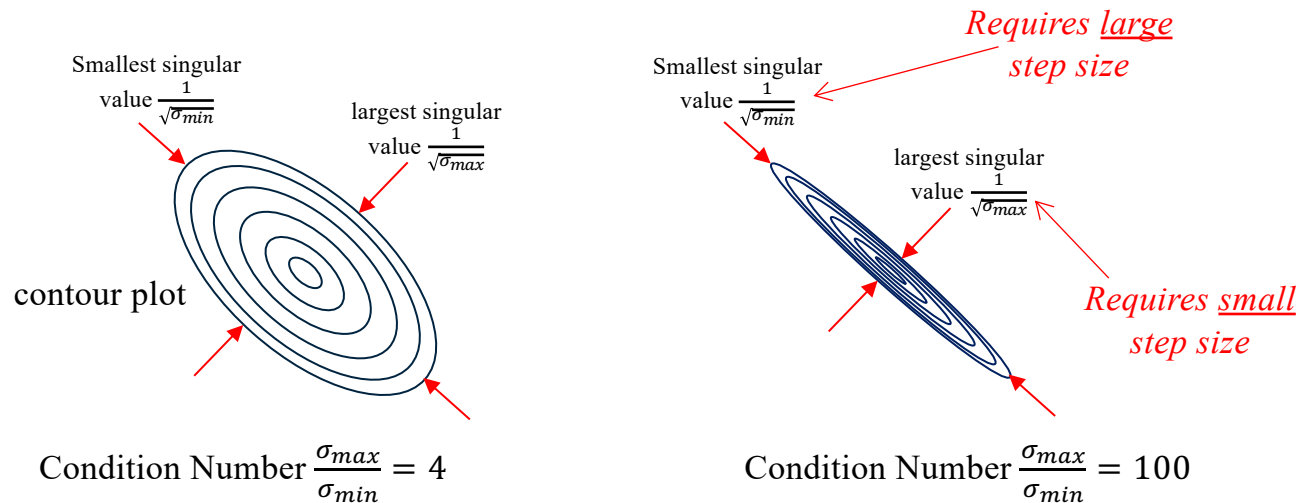


Requires small step size  $\alpha$

- Step size depends on second derivative
  - Small second derivative  $\Rightarrow$  large step size
  - Large second derivative  $\Rightarrow$  small step size

# Slow Convergence of Gradient Descent

- Very sensitive to condition number 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



# MSE Loss Function

- Interpretation of MSE Loss Function

$$L_{MSE}(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \|x_k - f_{\theta}(y_k)\|^2$$

*sum over training data*

*inference function*

# Computing the MSE Loss Gradient

- Use chain rule to compute the loss gradient

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

*What does this mean?*

# Interpretation of Loss Gradient

$$-\nabla_{\theta} L_{MSE}(\theta) = \frac{2}{K} \sum_{k=0}^{K-1} (x_k - f_{\theta}(y_k))^t \nabla_{\theta} f_{\theta}(y_k)$$

*sum over training data* (points to the sum)

*prediction error* (points to  $(x_k - f_{\theta}(y_k))^t$ )

*gradient of function* (points to  $\nabla_{\theta} f_{\theta}(y_k)$ )

- Loss Gradient computation requires:
  - Sum over training data: Big sum, but straight forward.
  - Prediction error: Easy to compute.
  - Gradient of inference function: This is the difficult part.
    - 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  $x_k = f_\theta(y_k) + \text{error}$ , we have that

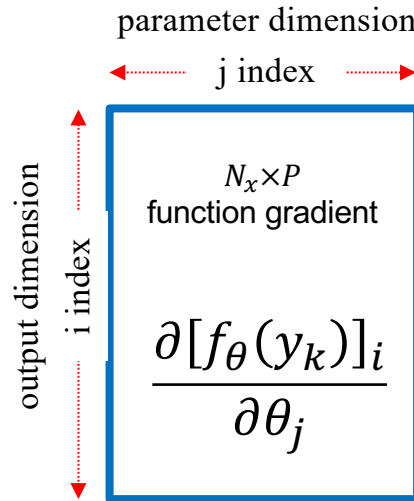
$$\boxed{\begin{array}{c} 1 \times N_x \\ \text{error vector} \end{array}} = \epsilon_k^t = (x_k - f_\theta(y_k))^t = \text{error vector}$$

- Then the parameter vector is given by

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

# Function Gradient

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

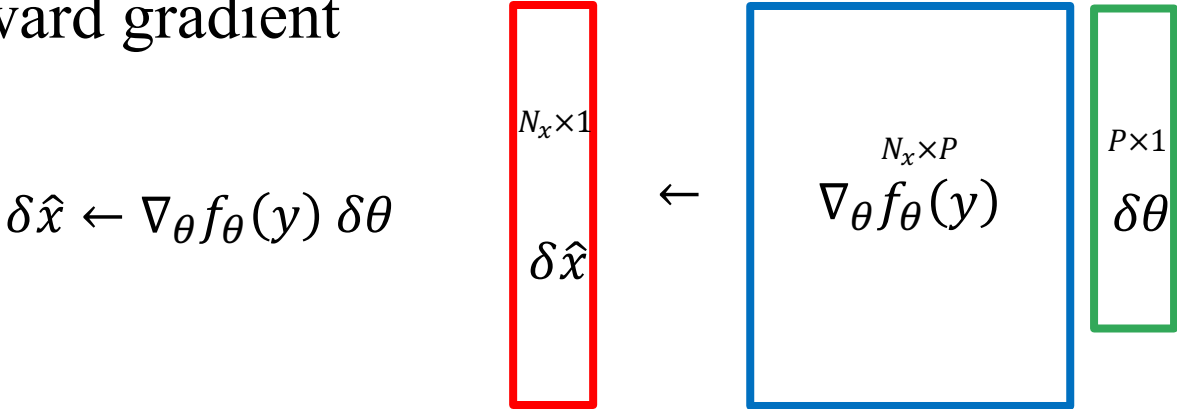


A diagram showing a blue-bordered square representing a matrix. Above the square, the text "parameter dimension" is centered, with a red double-headed arrow below it labeled "j index". To the left of the square, the text "output dimension" is written vertically, with a red double-headed arrow to its right labeled "i index". Inside the square, the text " $N_x \times P$ " is centered above the text "function gradient". Below this, the partial derivative  $\frac{\partial [f_{\theta}(y_k)]_i}{\partial \theta_j}$  is written.

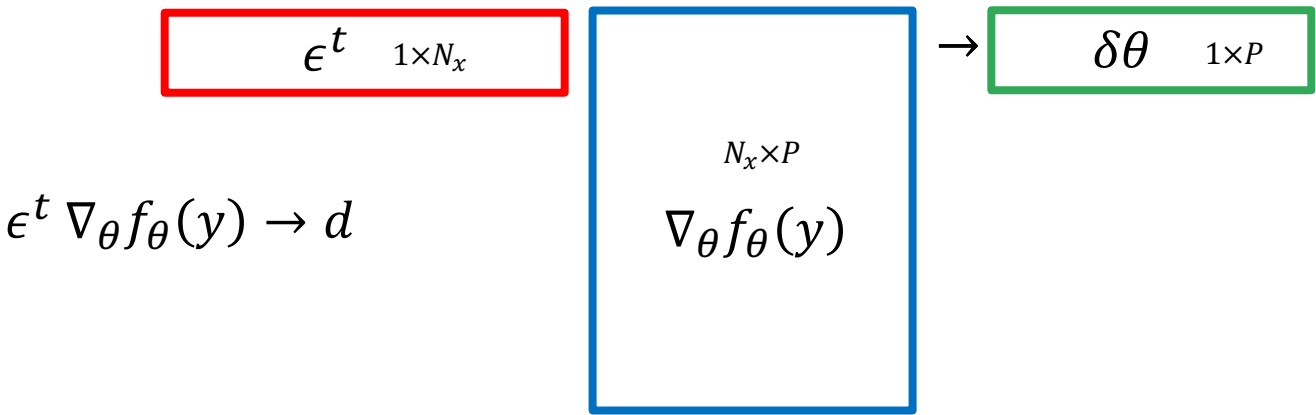
$$= \nabla_{\theta} f_{\theta}(y_k) = \text{gradient of function}$$

# Forward vs Backward Propagation

- Forward gradient



- Backward (adjoint) gradient



# Loss Gradient Computation

- Equation is

$$-\nabla_{\theta} L_{MSE}(\theta) = \frac{2}{K} \sum_{k=0}^{N-1} (x_k - f_{\theta}(y_k))^t \nabla_{\theta} f_{\theta}(y_k)$$

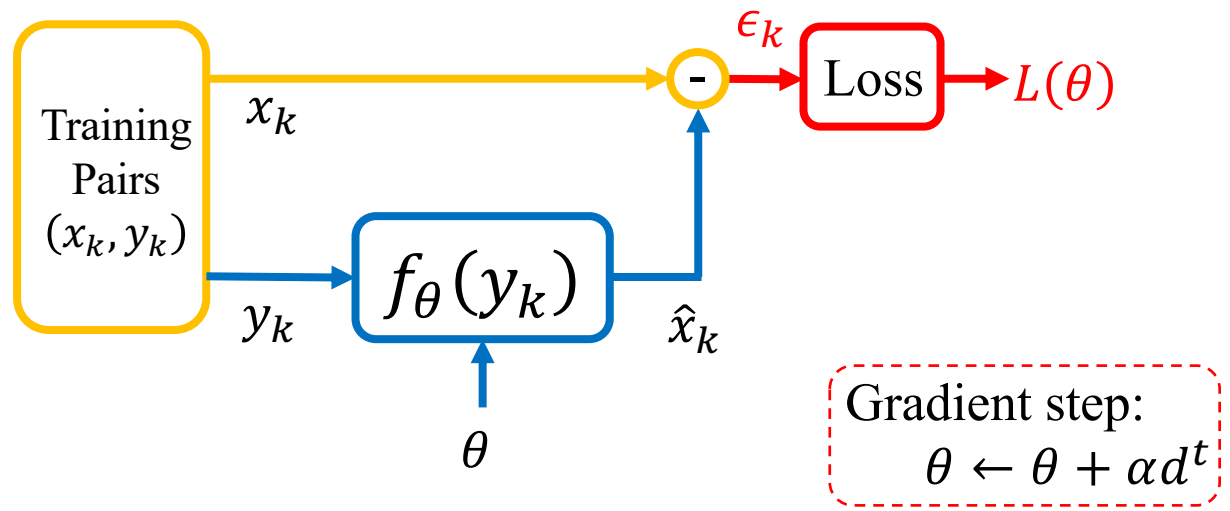
*prediction error*

*gradient of function*

- Looks like

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

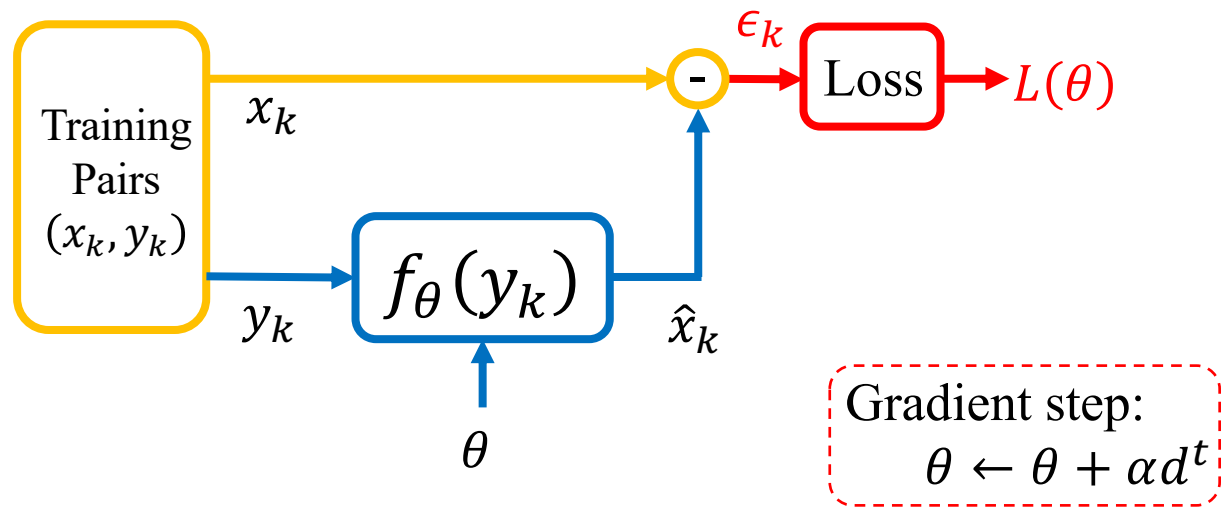
# Update Direction for Supervised Training



- $d$  is given by

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

# Update Direction for Supervised Training



- $d^t$  is given by

$$\begin{array}{|c|} \hline d^t \\ \hline P \times 1 \\ \hline \end{array} = \frac{2}{K} \sum_{k=0}^{K-1} \begin{array}{|c|} \hline [\nabla_{\theta} f_{\theta}(y_k)]^t \\ \hline P \times N_x \\ \hline \end{array} \begin{array}{|c|} \hline \epsilon_k \\ \hline N_x \times 1 \\ \hline \end{array}$$

*Multiply error by the adjoint gradient*