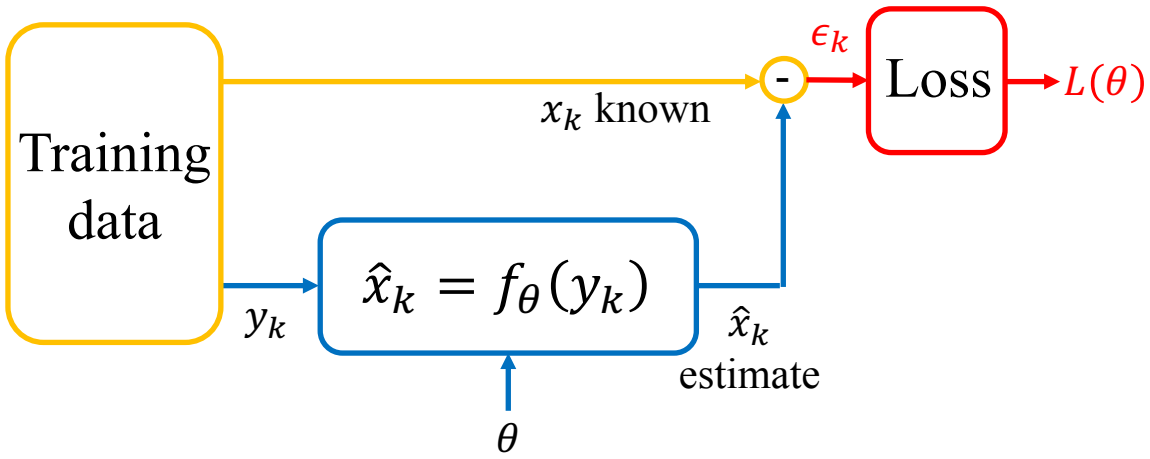
Gradient Descent Optimization

- 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?

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

Repeat until converged {

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

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

}

Gradient Descent (GD) Algorithm

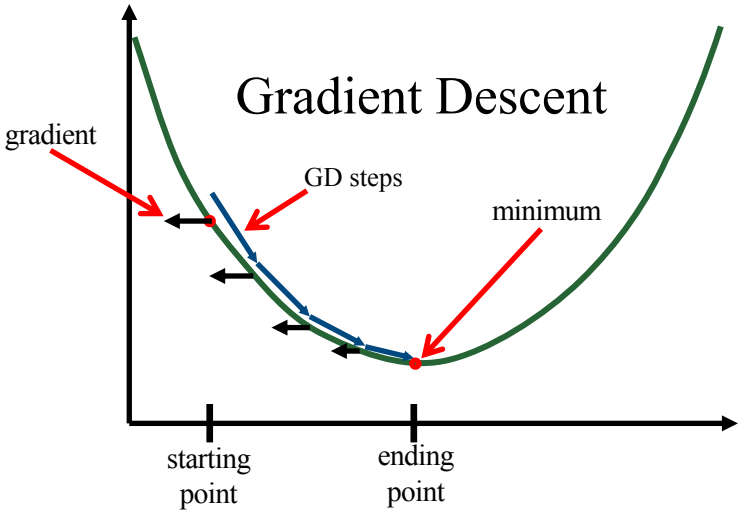
- Selecting α
  - α too small ⇒ slow convergence
  - α too large ⇒ unstable
  - Often there is no good choice!

Gradient Descent Picture

- The GD update step:

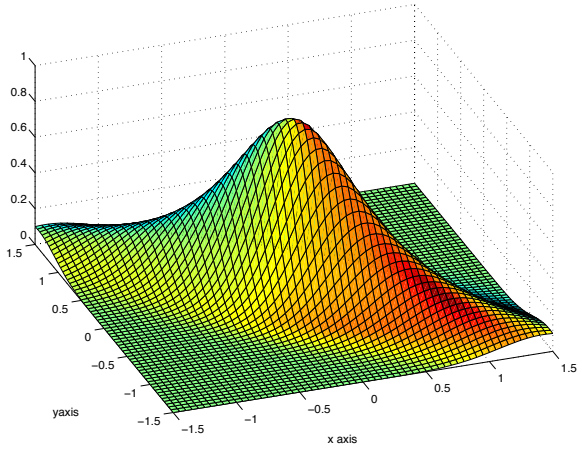
$$d \leftarrow -\nabla L(\theta)$$
$$\theta \leftarrow \theta + \alpha d^t$$

1D case

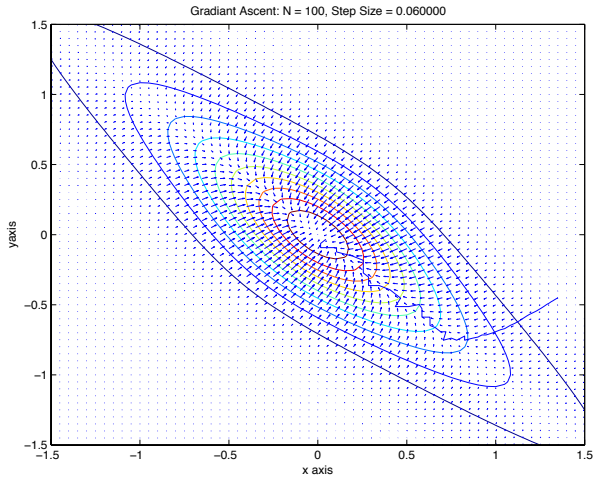


2D case

Function

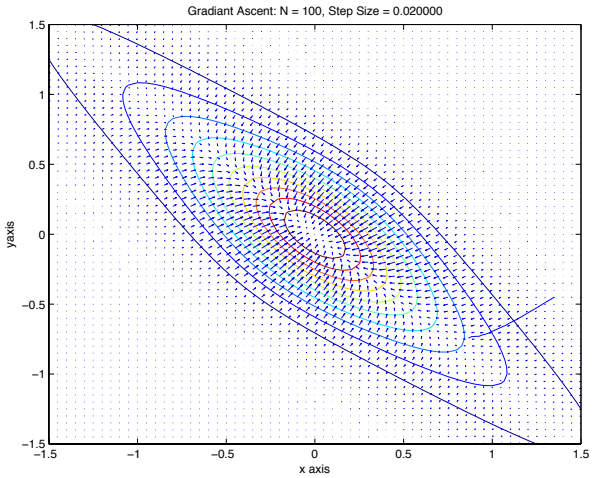


Updates

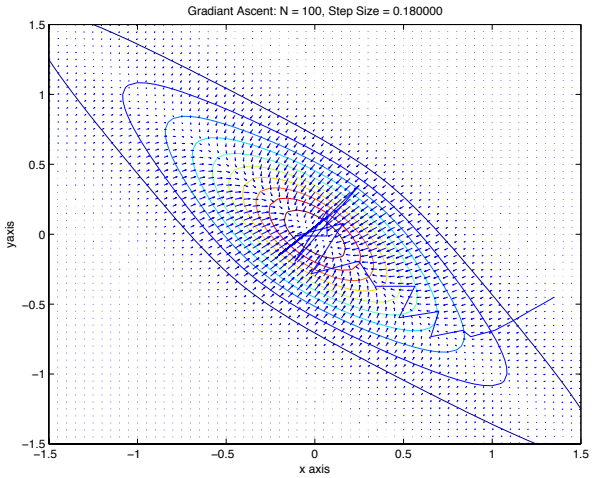


Gradient Step Size

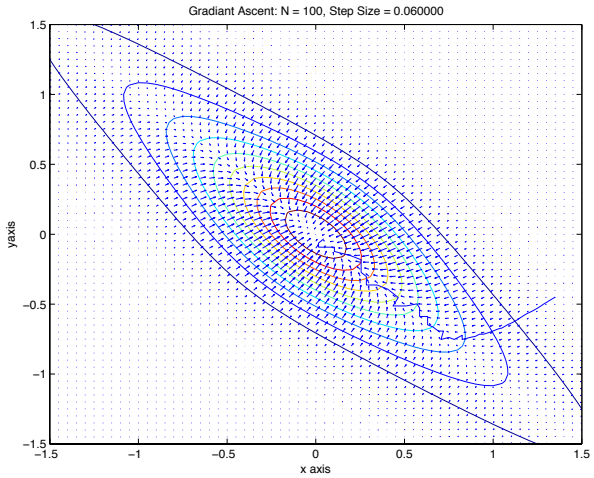
- How large should α be?



Too small ⇒ slow convergence



Too large ⇒ oscilate



Goldilocks? ⇒ good enough

Steepest Descent

- Use line search to compute the best α

Repeat until converged {

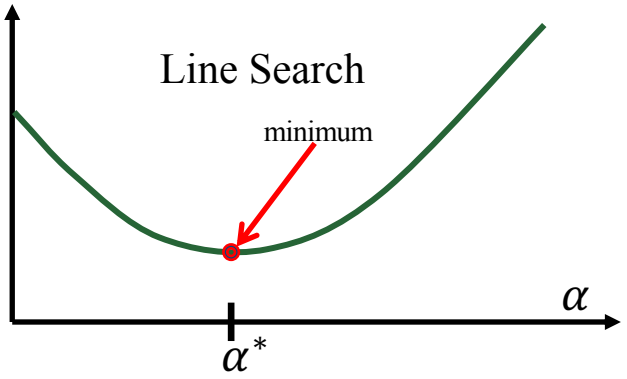
$$d \leftarrow -\nabla L(\theta)$$

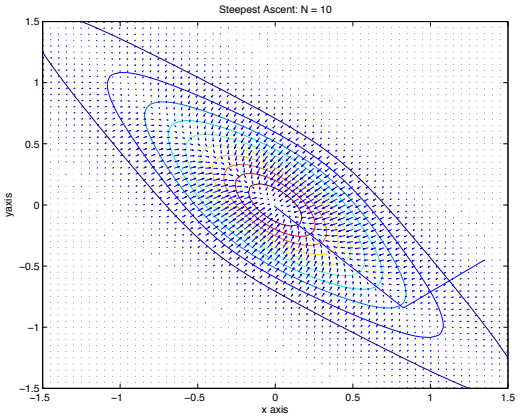
$$\alpha^* \leftarrow \arg\min_{\alpha}\{L(\theta + \alpha d^t)\}$$

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

}

Steepest Descent Algorithm

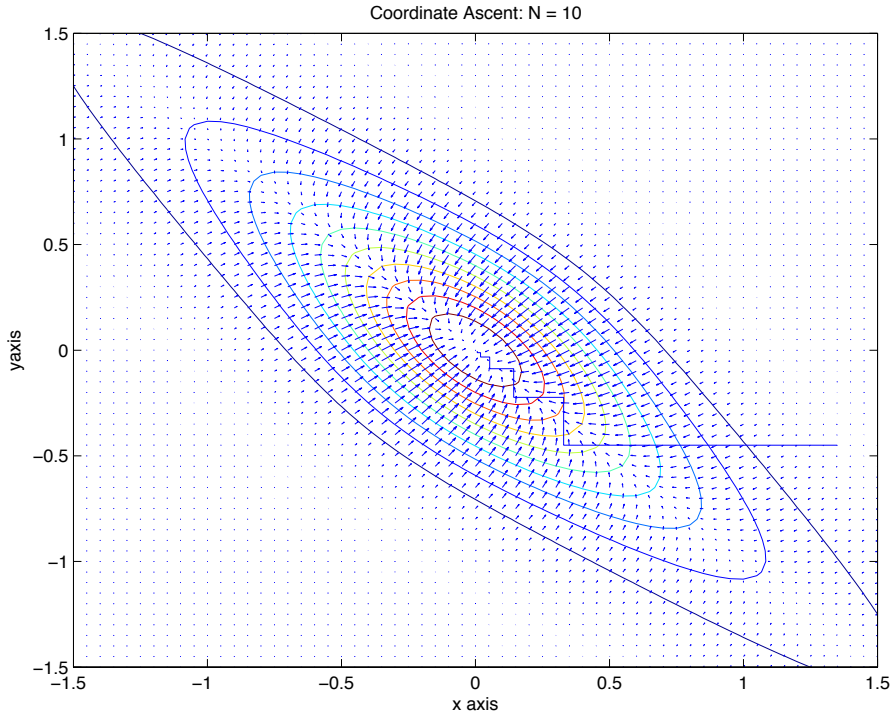




Greedy Steepest Descent ⇒
Too Expensive

Coordinate Descent

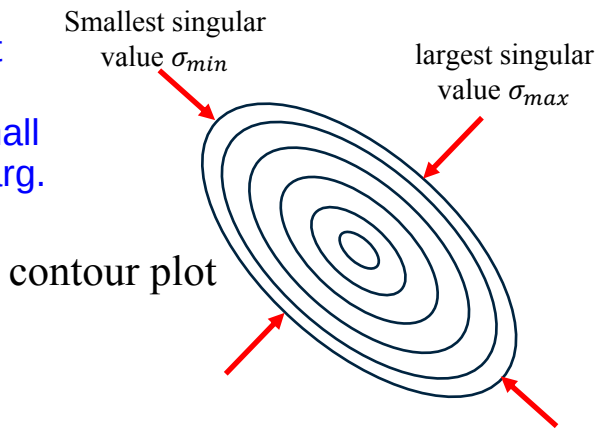
- Update one parameter at a time
  - Removes problem of selecting step size
  - Each update can be very fast, but lots of updates



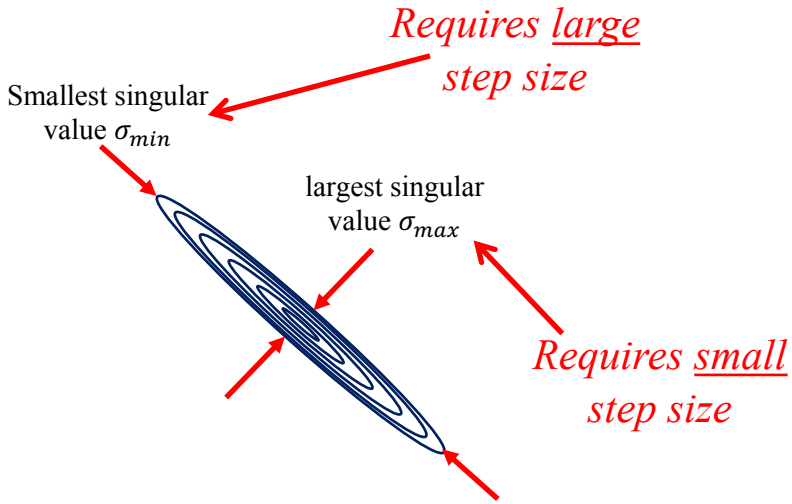
Slow Convergence of Gradient Descent

- 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
  - Preconditioning: Easy, but tends to be ad-hoc, not so robust
  - Momentum: More latter

Condition number:
how much the output
of a function
Can change for a small
change in the input arg.



Condition Number $\frac{\sigma_{max}}{\sigma_{min}} = 2$



Condition Number $\frac{\sigma_{max}}{\sigma_{min}} = 10$

Computing the Loss Gradient

- Use chain rule to compute the loss gradient

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

Der[f1(f2(f3(theta))] = der(f1(theta))*der(f2(theta)) * der(f3(theta))

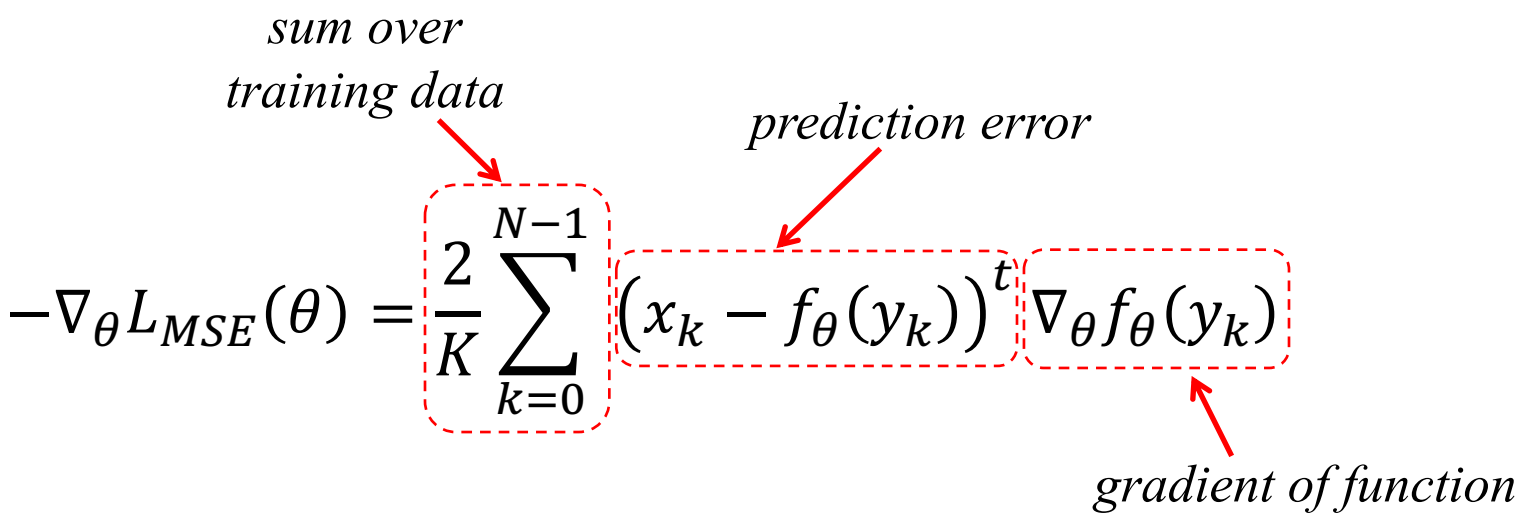
$$= \frac{1}{K} \sum_{k=0}^{K-1} \nabla_\theta \{\|x_k - f_\theta(y_k)\|^2\}$$

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

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

What does this mean?

Interpretation of Loss Gradient



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

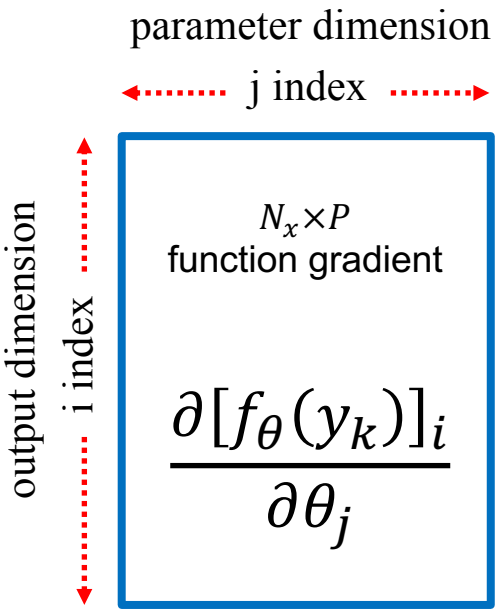
$$\underbrace{\text{error vector}}_{1\times N_x} = \epsilon_k^t = \left(x_k - f_\theta(y_k)\right)^t = \text{error vector}$$

- Then the parameter vector is given by

$$\underbrace{\text{parameter vector}}_{P\times 1} = [-\nabla_\theta L_{MSE}]^t = \text{dimension of parameter vector}$$

Function Gradient

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

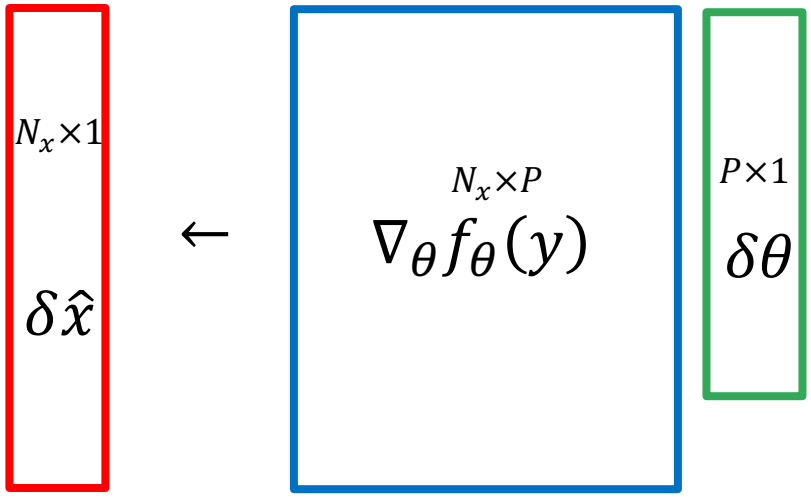


$= \nabla_\theta f_\theta(y_k) =$ gradient of function

Forward vs Backward Propogation

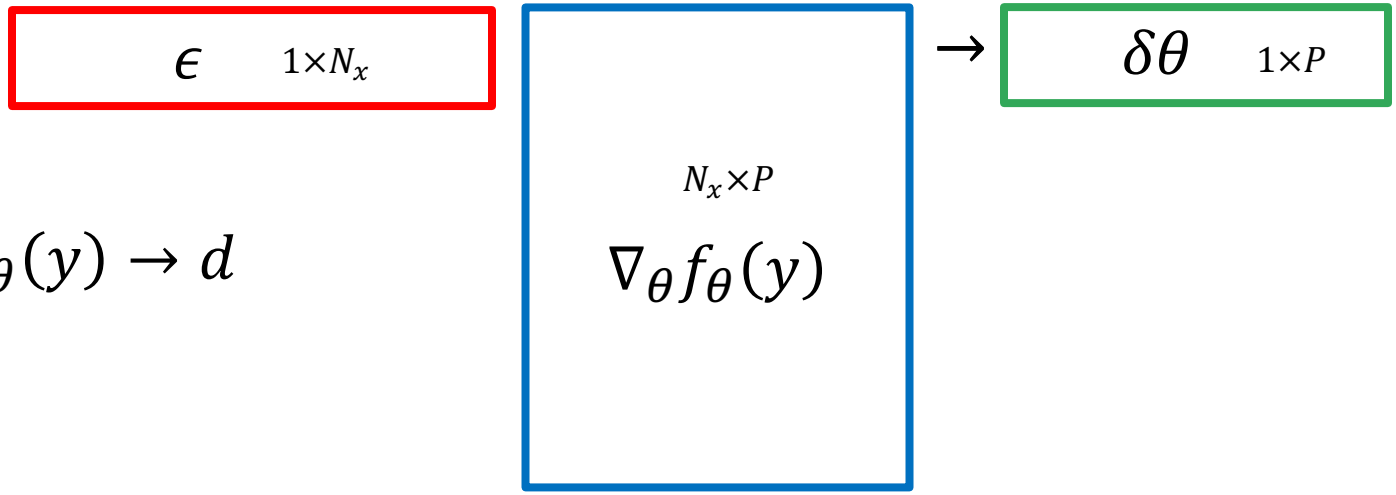
- Forward gradient

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



- Backward (adjoint) gradient

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



Loss Gradient Computation

- Equation is

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

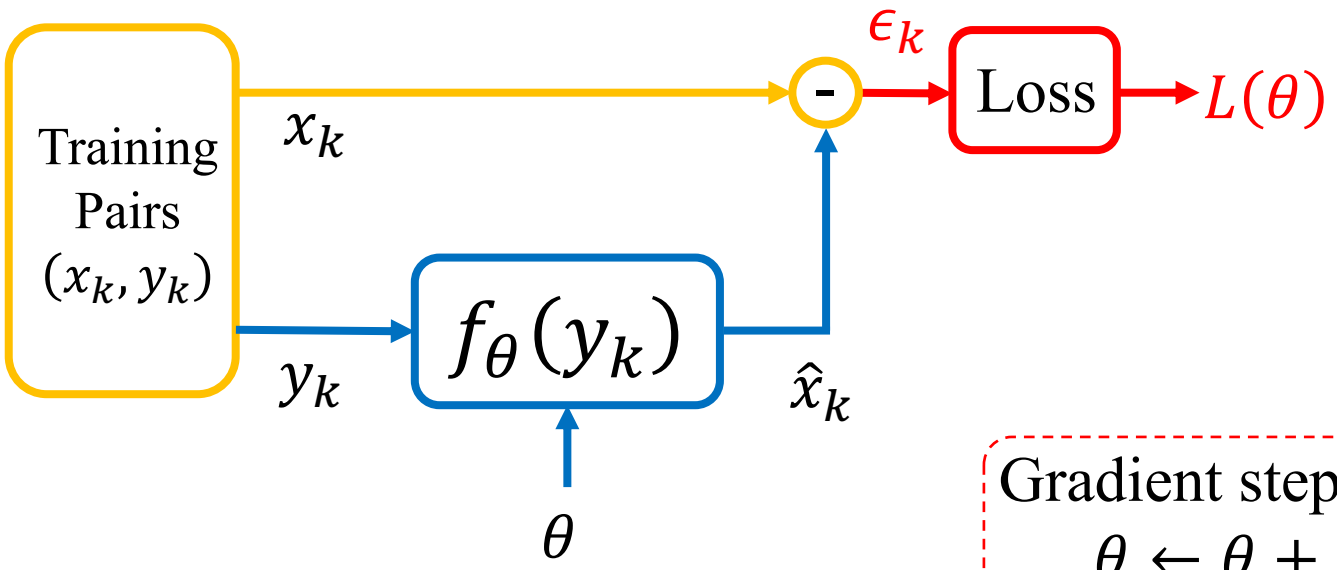
prediction error

gradient of function

- Looks like

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

Update Direction for Supervised Training



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

- d is given by

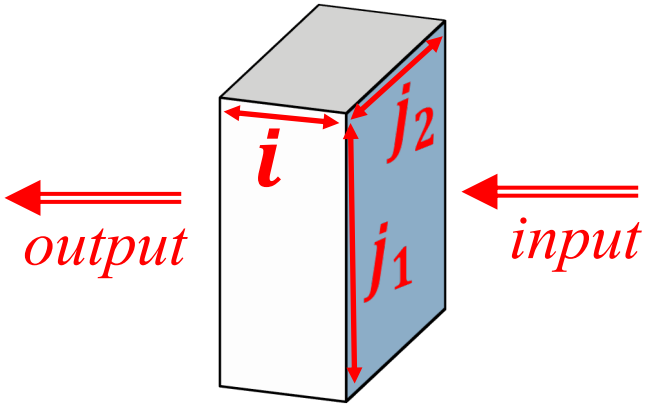
$$\underbrace{d}_{1\times P} = \frac{2}{K}\sum_{k=0}^{K-1} \underbrace{\epsilon_k}_{1\times N_x} \underbrace{\nabla_\theta f_\theta(y_k)}_{N_x \times P}$$

Tensors

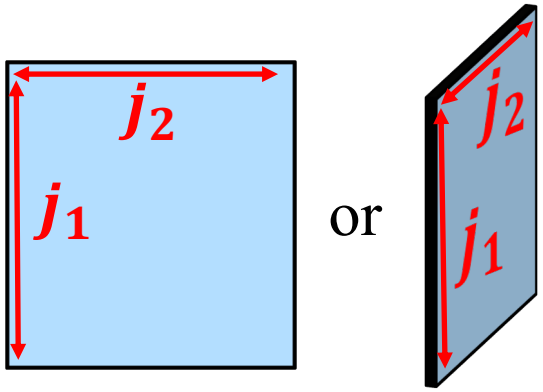
- Contravariant and covariant tensors
- Einstein notation
- Interpretation

What is a Tensor?

- It is the generalization of a:
  - Matrix operator to >2 dimensions



  - Vector data to >1 dimension



- Why do we need it?
  - Came out of differential geometry.
  - Was used by Albert Einstein to formulate the theory of General Relativity.
  - It needed for many problems, including Deep NNs.

Contravariant and Covariant Vectors

$$x = gy$$

- Contravariant vectors:
  - "Column Vectors" that represent data
  - Vectors that describe the position of something
  - y^j for $0 \leq j < N$ and $x \in \Re$
- Covariant vector:
  - "Row vectors" that operate on data
  - Gradient vectors
  - g_j for $0 \leq i < N$
- Einstein notation

$$x = g_j\, y^j = \sum_{j=0}^{N-1} g_j\, y^i$$

  - Leave out the sum to make notation cleaner
  - Always sum over any two indices that appear twice

Vector-Matrix Products as Tensors

$$x = Gy$$

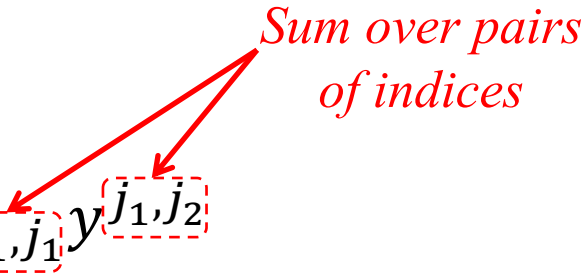
- 1D Contravariant vectors:
  - y^j for $0 \leq j < N_y$
  - x^j for $0 \leq j < N_x$
- 2D tensor (i.e., matrix):
  - $G^i{}_j$ for $0 \leq i < N_x$ and $0 \leq j < N_y$
  - There is a gradient for each component x^i
- Einstein notation

$$x^i = G^i{}_j \, y^j = \sum_{j=0}^{N_y - 1} G^i{}_j \, y^j$$

  - Leave out the sum to make notation cleaner
  - Always sum over any two indices that appear twice

Tensor Products

- Einstein notation



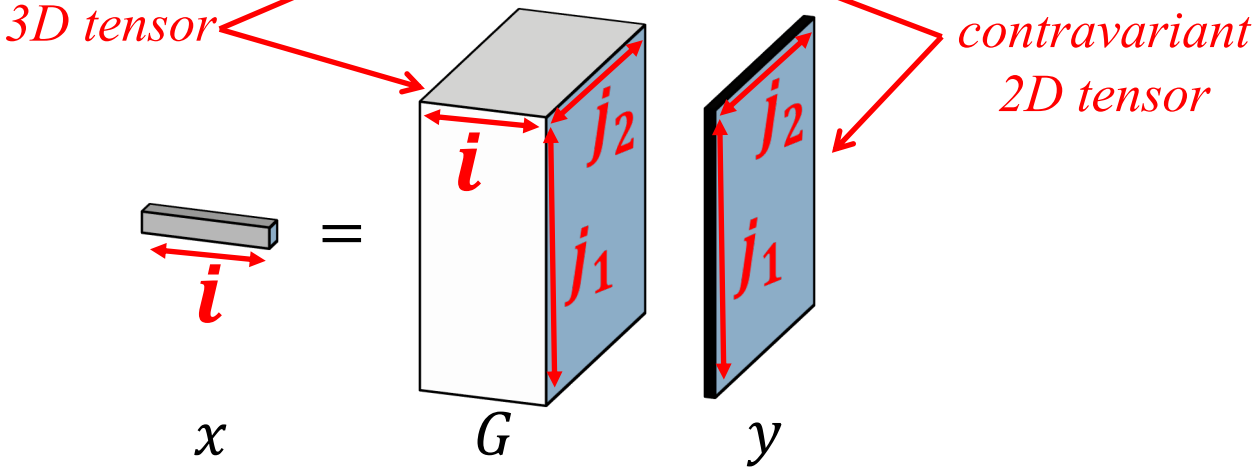
$$x^{i_1,i_2} = G^{i_1,i_2}{}_{j_1,j_1} y^{j_1,j_2}$$

- 2-D Contravariant vectors:
  - y^{j_1,j_2} for $0 \leq j_1, j_2 < N_y$
  - x^{i_1,i_2} for $0 \leq i_1, i_2 < N_x$
- 4-D Tensor:
  - $G^{i_1,i_2}{}_{j_1,j_1}$ for $0 \leq i_1, i_2 < N_x$ and $0 \leq j_1, j_2 < N_y$
  - G is known as a tensor
    - 2D covariant input
    - 2D contravariant output

Example of a Tensor Product

- For example, if we have

$$x^i = G^i{}_{j_1,j_2}\, y^{j_1,j_2}$$



  - Tensor 3-D tensor G has

    Input: 2D image indexed by (j_1, j_2)

    Output: 1D vector indexed by i

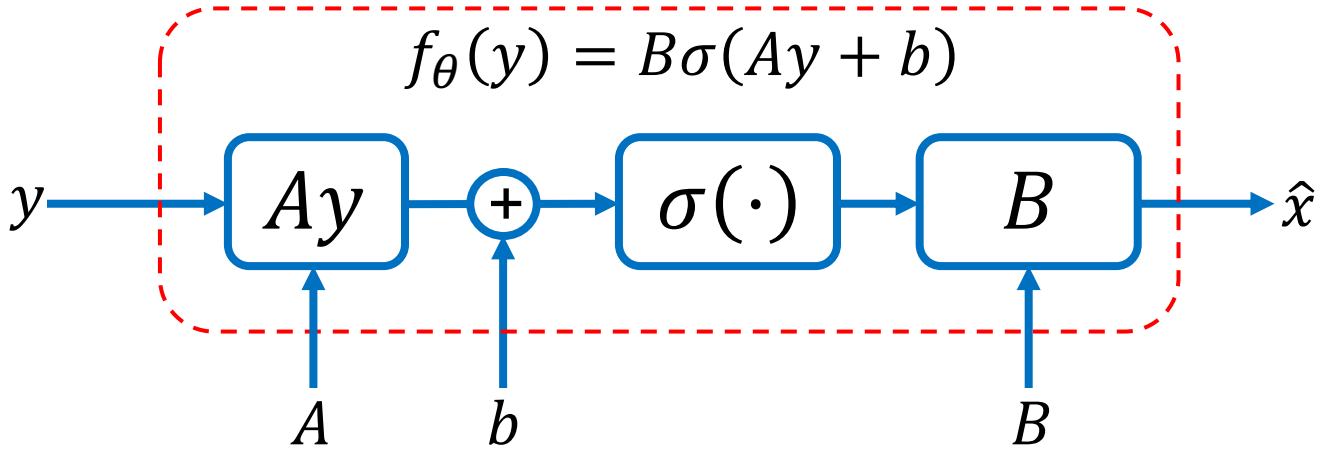
  - General idea: $G \in \Re^{N_o \times N_i}$

GD for Single Layer NN

- Structure of the Gradient
- Gradients for NN parameters
- Updates for NN parameters

Gradient Direction for Single Layer NN

- Single layer NN:



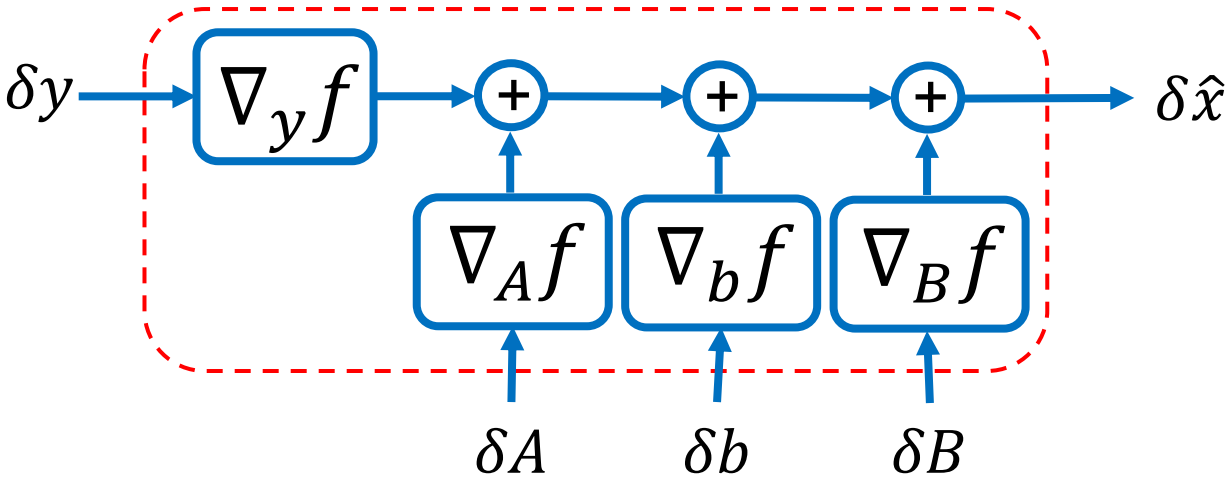
  - We will need gradient w.r.t the parameters $\theta = (A, B, b)$:

$$\nabla_\theta f_\theta(y) = \left[\nabla_A f_{(A,b,B)}(y), \nabla_b f_{(A,b,B)}(y), \nabla_B f_{(A,b,B)}(y)\right]$$

  - Later, we will also need:

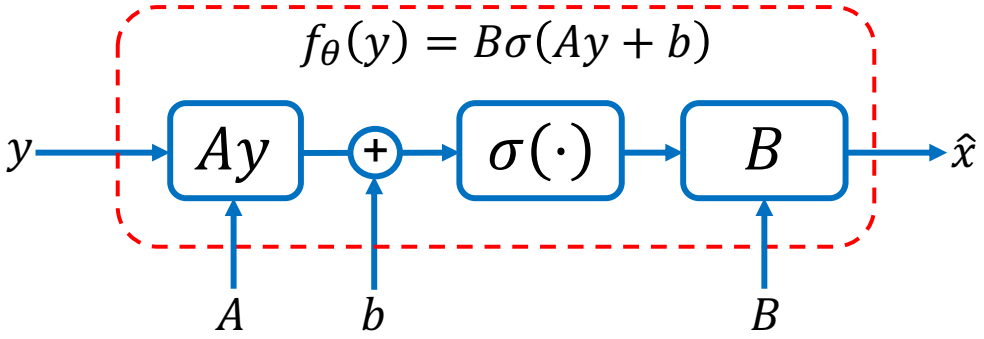
$$\nabla_y f_\theta(y)$$

Gradient Structure for Single Layer NN



- Single layer NN:
  - Parameters are $\nabla_A f_{(A,B,b)}$
  - We will need the parameter gradients:
    $$\nabla_\theta f_\theta(y) = \left[\nabla_A f_{(A,b,B)}(y), \nabla_b f_{(A,b,B)}(y), \nabla_B f_{(A,b,B)}(y)\right]$$
  - And the input gradient:
    $$\nabla_y f_\theta(y)$$

Gradient w.r.t. y



- For this case,

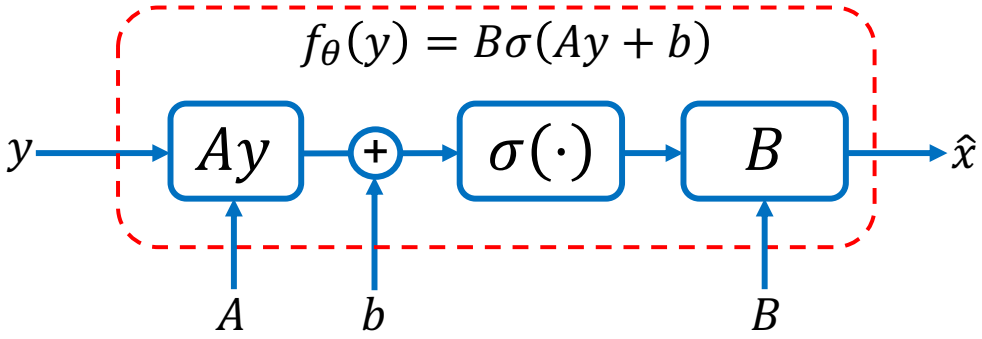
$$\nabla_y f = \nabla_y f_\theta(y) = B[\nabla\sigma(Ay + b)]A$$

  - Using Einstein notation

$$\left[\nabla_y f\right]^i{}_j = B^i{}_{i_1}[\nabla\sigma]^{i_1}{}_{i_2}A^{i_2}{}_j$$

This is a matrix or 2-D tensor

Gradient w.r.t. A



- For this case,

$$\nabla_A f = B\,[\nabla\sigma(Ay + b)]\,\nabla_A(Ay)$$

This is a tensor!

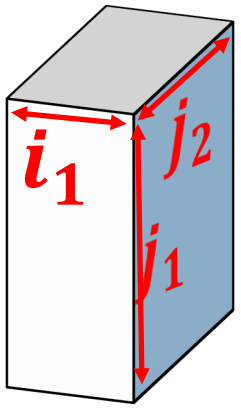
  - Using Einstein notation, since

$$[\nabla_A(Ay)]^{i}{}_{j_1,j_2} = \delta^{i}{}_{j_1} y^{j_2}$$

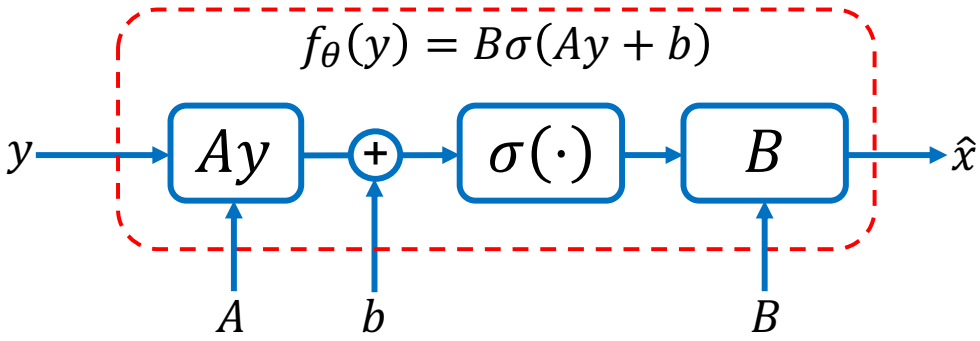
  - Then

$$\begin{aligned}[\nabla_A f]^{i_1}{}_{j_1,j_2} &= B^{i_1}{}_{i_2}[\nabla\sigma]^{i_2}{}_{i_3}[\nabla_A(Ay)]^{i_3}{}_{j_1,j_2} \\ &= B^{i_1}{}_{i_2}[\nabla\sigma]^{i_2}{}_{i_3}\delta^{i_3}{}_{j_1} y^{j_2}\end{aligned}$$

$$[\nabla_A f]^{i_1}{}_{j_1,j_2} = B^{i_1}{}_{i_2}[\nabla\sigma]^{i_2}{}_{j_1}\, y^{j_2}$$



Gradient w.r.t. b



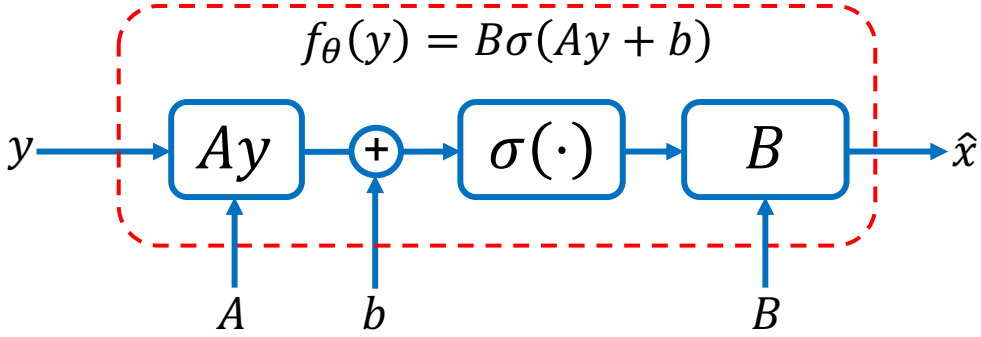
- For this case,

$$\nabla_b f = B\,\nabla\sigma$$

  - Using Einstein notation,

$$[\nabla_b f]^{i_1}{}_{j_1} = B^{i_1}{}_{i_2}[\nabla\sigma]^{i_2}{}_{j_1}$$

Gradient w.r.t. B



- For this case,

$$\nabla_B f = \nabla_B (B\sigma)$$

This is a tensor!

  - Using Einstein notation, since

$$[\nabla_B (B\sigma)]^i{}_{j_1,j_2} = \delta^i{}_{j_1}\sigma^{j_2}$$

  - We have that

$$[\nabla_b f]^{i_1}{}_{j_1,j_2} = \delta^{i_1}{}_{j_1}\sigma^{j_2}$$

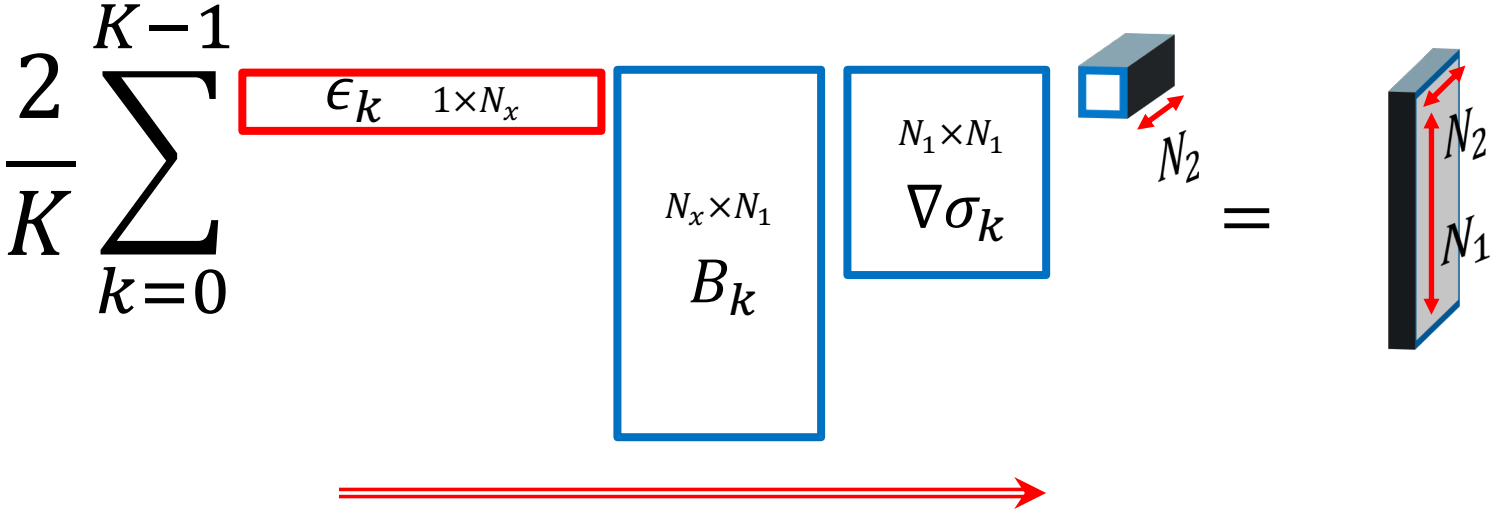
Update Direction for A

Gradient step:

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

- d is given by

$$d_{j_1,j_2} = \frac{2}{K} \sum_{k=0}^{K-1} \epsilon_{k,i_1} \, {B_k}^{i_1}{}_{i_2} [\nabla \sigma_k]^{i_2}{}_{j_1} \, {y_k}^{j_2}$$



For efficiency, computation goes this way!

Update Direction for b

Gradient step:

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

- d is given by

$$d_{j_1} = \frac{2}{K} \sum_{k=0}^{K-1} \epsilon_{k,i_1} \, B_k{}^{i_1}{}_{i_2} [\nabla \sigma_k]^{i_2}{}_{j_1}$$

$$\frac{2}{K} \sum_{k=0}^{K-1} \underbrace{\epsilon_k}_{1 \times N_x} \underbrace{B_k}_{N_x \times N_1} = \underbrace{}_{N_1 \times 1}$$

For efficiency, computation goes this way!

Update Direction for B

Gradient step:

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

- d is given by

$$d_{j_1,j_2} = \frac{2}{K} \sum_{k=0}^{K-1} \epsilon_{k,j_1} \sigma_k{}^{j_2}$$

$$\frac{2}{K} \sum_{k=0}^{K-1} \epsilon_k \quad 1 \times N_x \qquad \sigma_k \quad N_2 \qquad = \qquad N_x \quad N_2$$

For efficiency, computation goes this way!