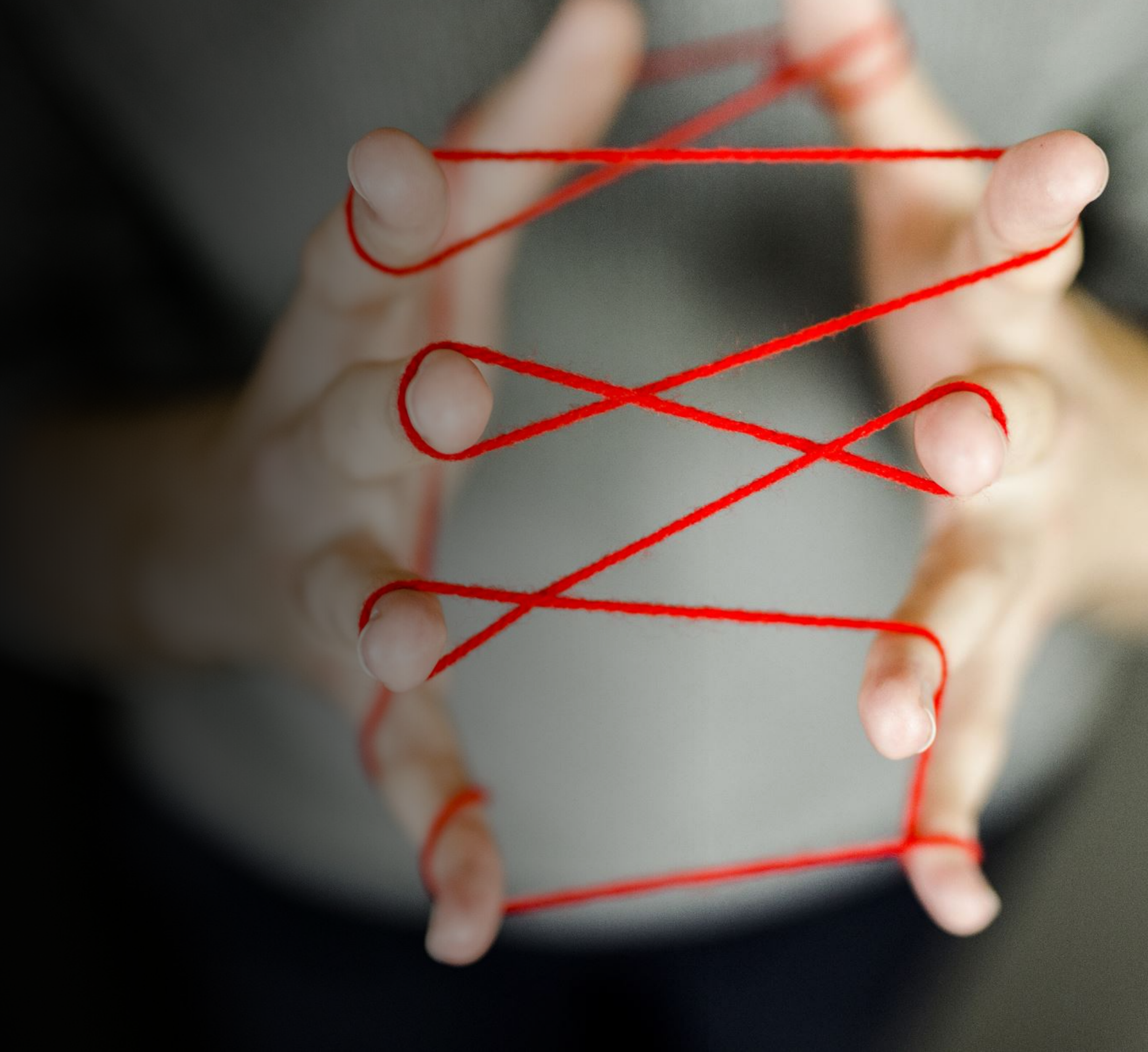
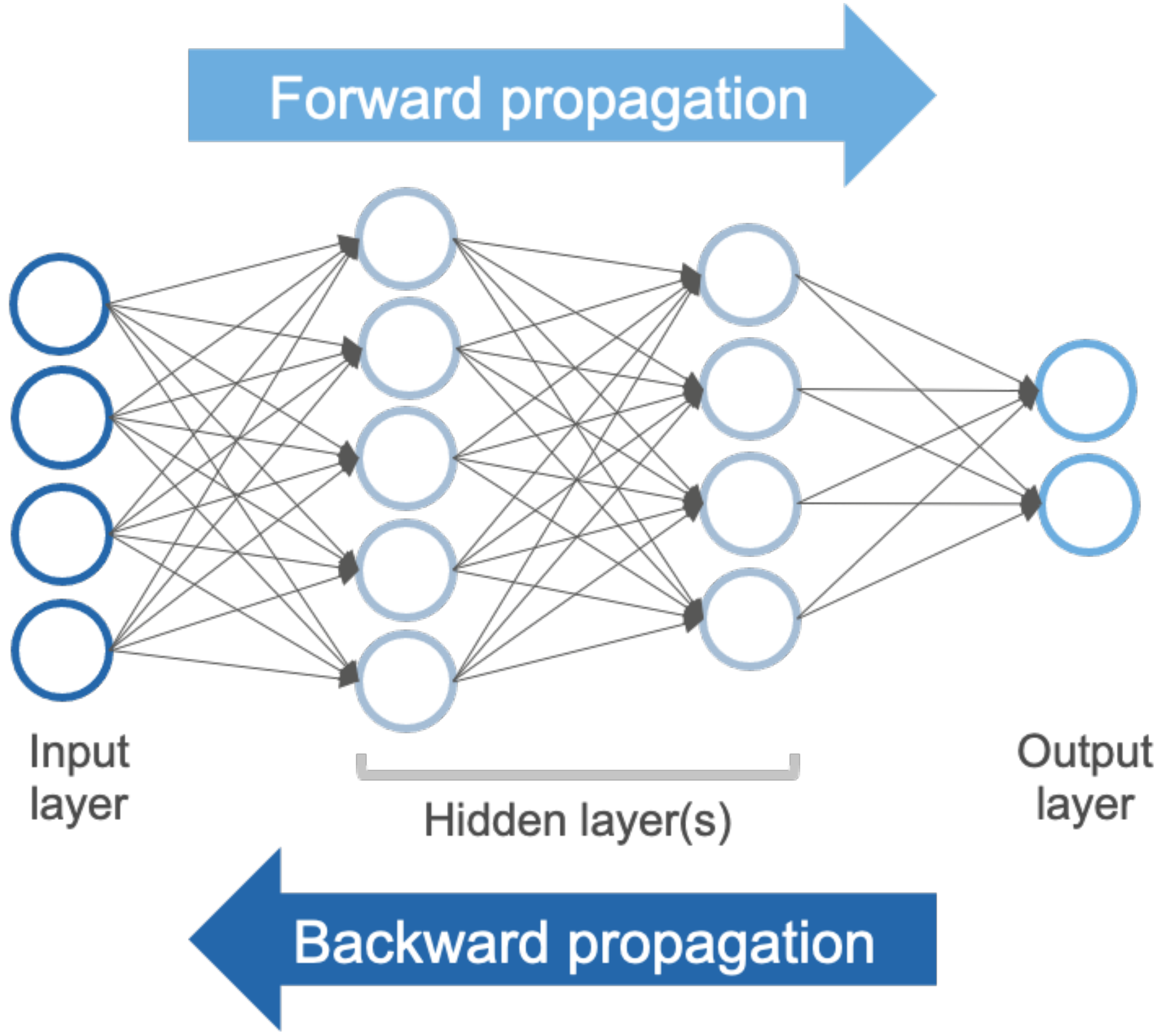
Introduction to Neural Networks Operations and Distributed Training



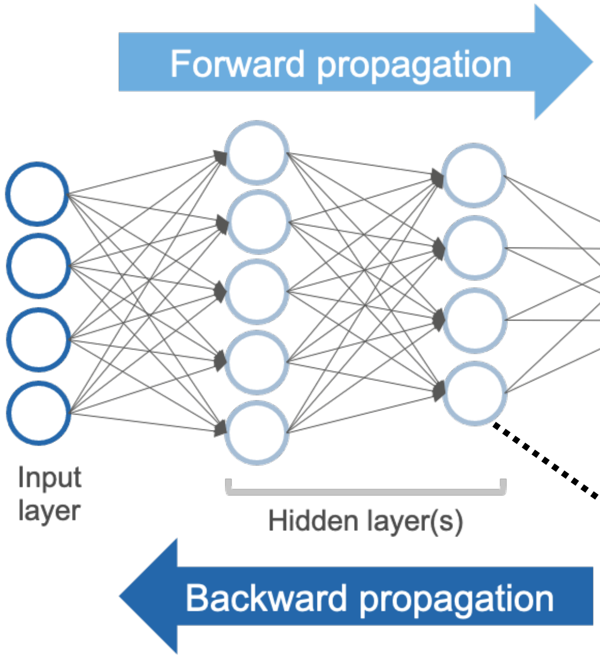
PD. Dr. Juan J. Durillo

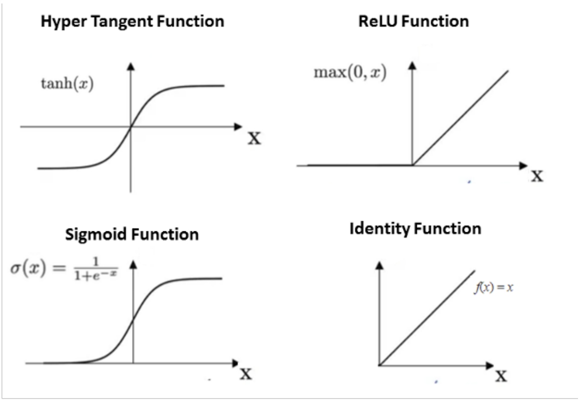
Deep Learning and GPU Programming Workshop @ CSC 10.5-13.5.2022

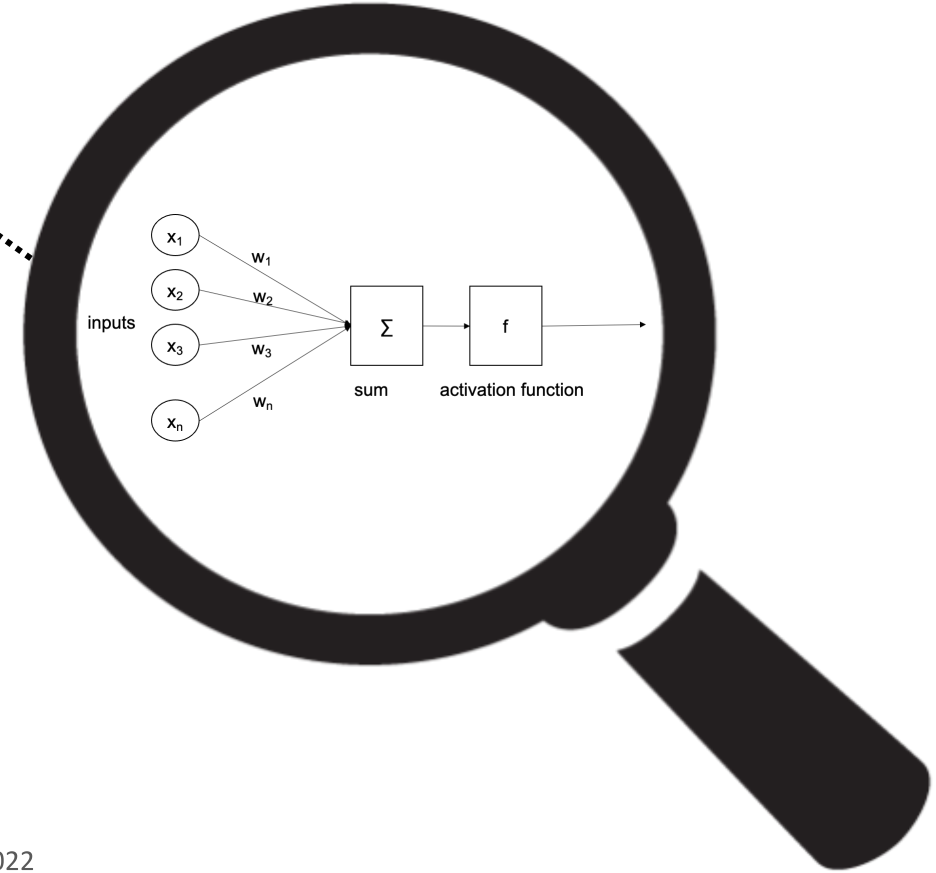
Machine Learning with (deep) Neural Networks



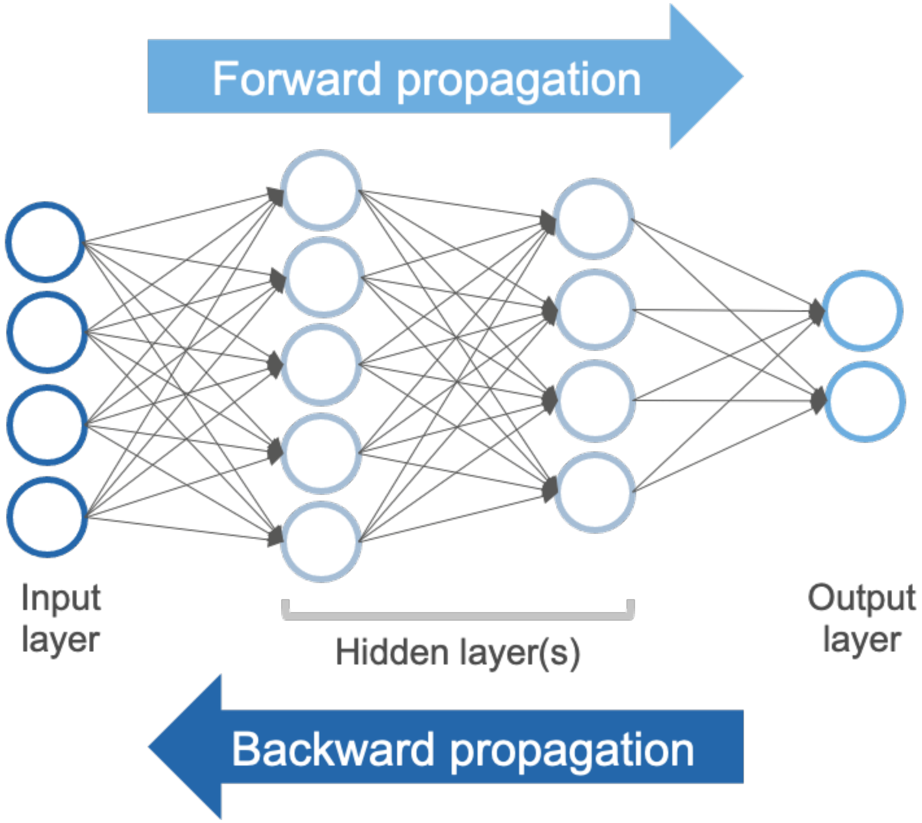
Machine Learning with (deep) Neural Networks







Machine Learning with (deep) Neural Networks



Loss function

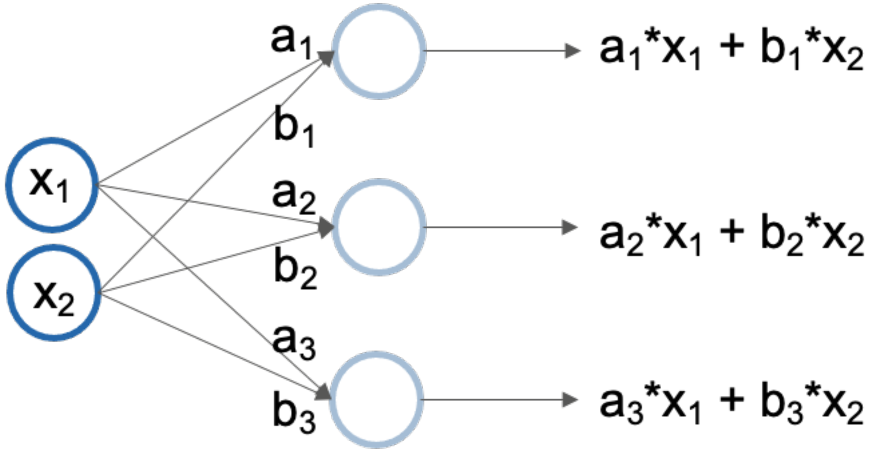
$$l = MSE = \frac{\sum_{i=1}^{n}(y_i - y_i^{p})^2}{n}$$

Optimizer SGD

$$\theta_t \leftarrow \theta_{t-1} - n_t * g(\theta_t;\ B_t)$$

$$g(\theta_t;\ B_t) = \frac{1}{|B_t|}\sum_{z \in B_t} \nabla l(\theta_t; z)$$

Forward Operations



Matrix Multiplication Operation

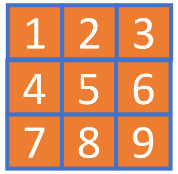
$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a_1{*}x_1 + b_1{*}x_2 \\ a_2{*}x_1 + b_2{*}x_2 \\ a_3{*}x_1 + b_3{*}x_2 \end{pmatrix}$$

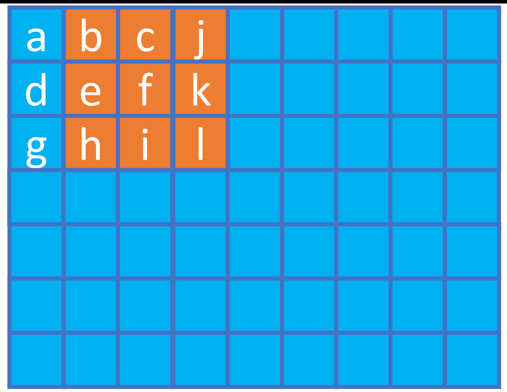
$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{pmatrix} \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} = \begin{pmatrix} a_1{*}x_1 + b_1{*}x_2 & a_1{*}y_1 + b_1{*}y_2 \\ a_2{*}x_1 + b_2{*}x_2 & a_2{*}y_1 + b_2{*}y_2 \\ a_3{*}x_1 + b_3{*}x_2 & a_3{*}y_1 + b_3{*}y_2 \end{pmatrix}$$

batch of two inputs

Forward Operations

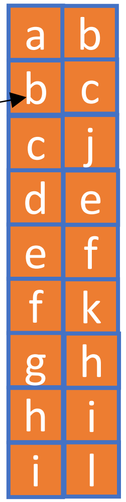
Convolutions as Matrix Multiplication



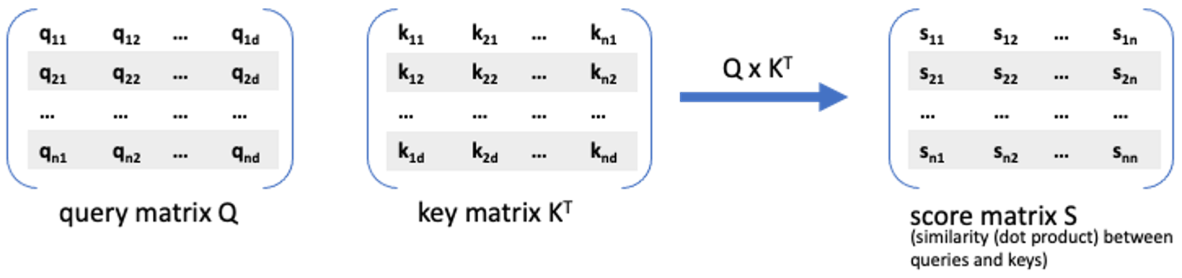


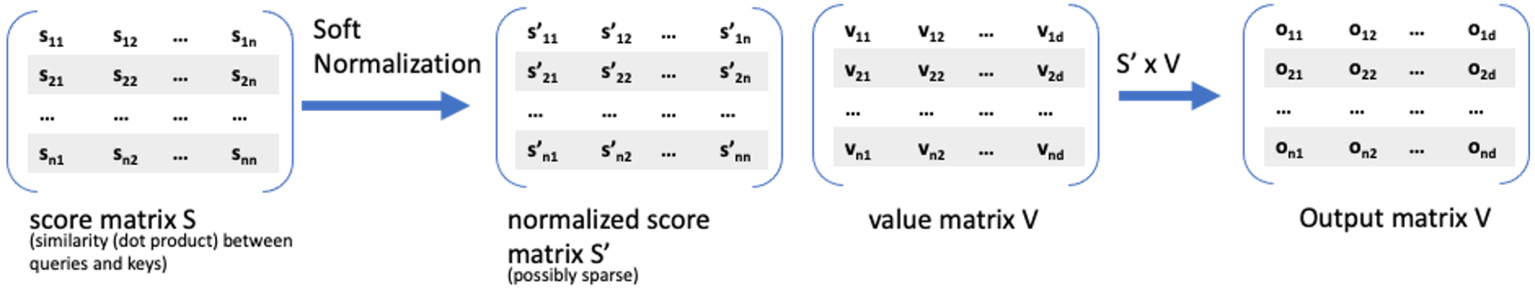
Duplication of elements

Built of special matrices causes overhead



Attention Layers are also Matrix Multiplications

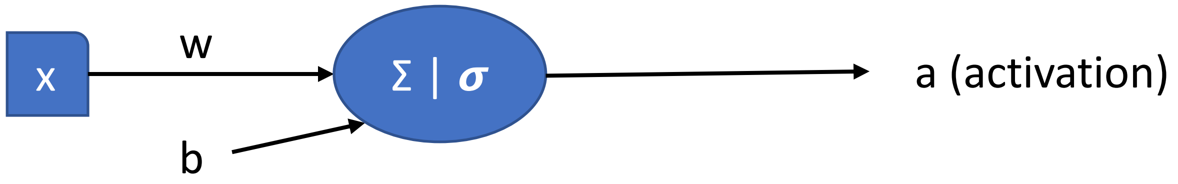




Back Propagation and Gradient Descent

- Linear regression y = w * x + b (I.e., a NN of a single neuron, and identity, f(x) = x, as activation function)

$\boldsymbol{\sigma}$ (w * x + b) = $\boldsymbol{\sigma}$ (z) if we define z as w * x + b

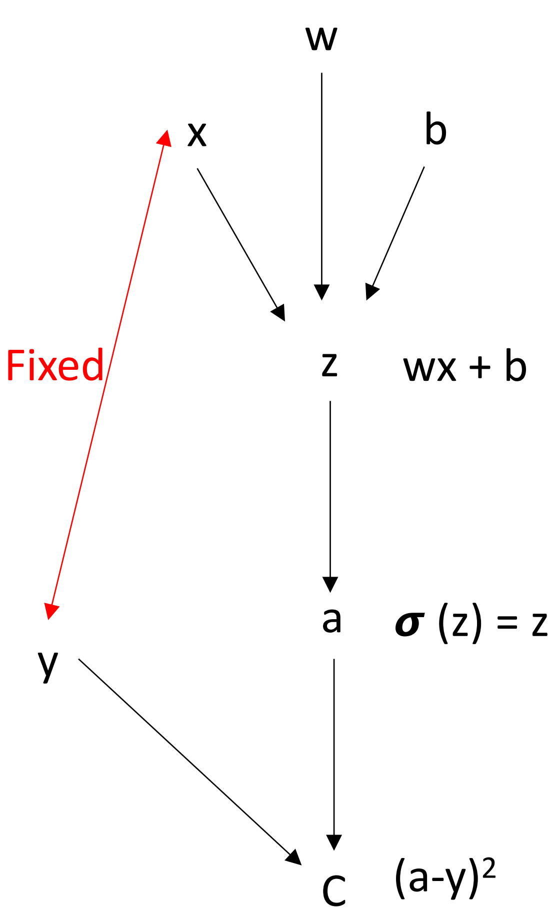


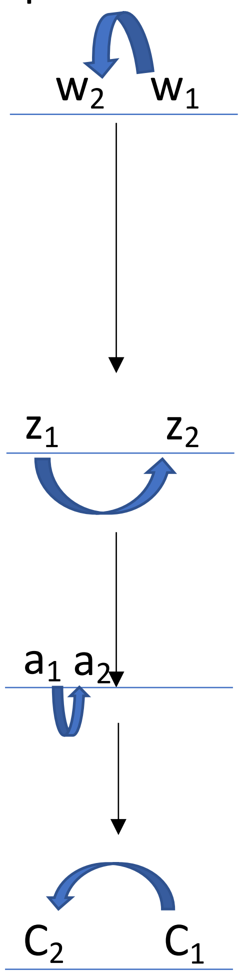
- Loss function defined as C = (a – y)2

- How does C change with w and b variations?
  - compute the ratio at with C changes with changes in w and b
  - use this ratio to modify then w and b in order to move C towards a minimum

Computing the Gradient

Gradient with a single input, that generates prediction a





$$\frac{\partial C}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial a}{\partial z}\frac{\partial C}{\partial a} = 2x(a-y)$$

$$\frac{\partial z}{\partial w} = x$$

$$\frac{\partial a}{\partial z} = 1$$

$$\frac{\partial C}{\partial a} = 2(a-y)$$

$$\frac{\partial C}{\partial b} = \frac{\partial z}{\partial b}\frac{\partial a}{\partial z}\frac{\partial C}{\partial a} = 2(a-y)$$

$$\frac{\partial z}{\partial b} = 1$$

$$\frac{\partial a}{\partial z} = 1$$

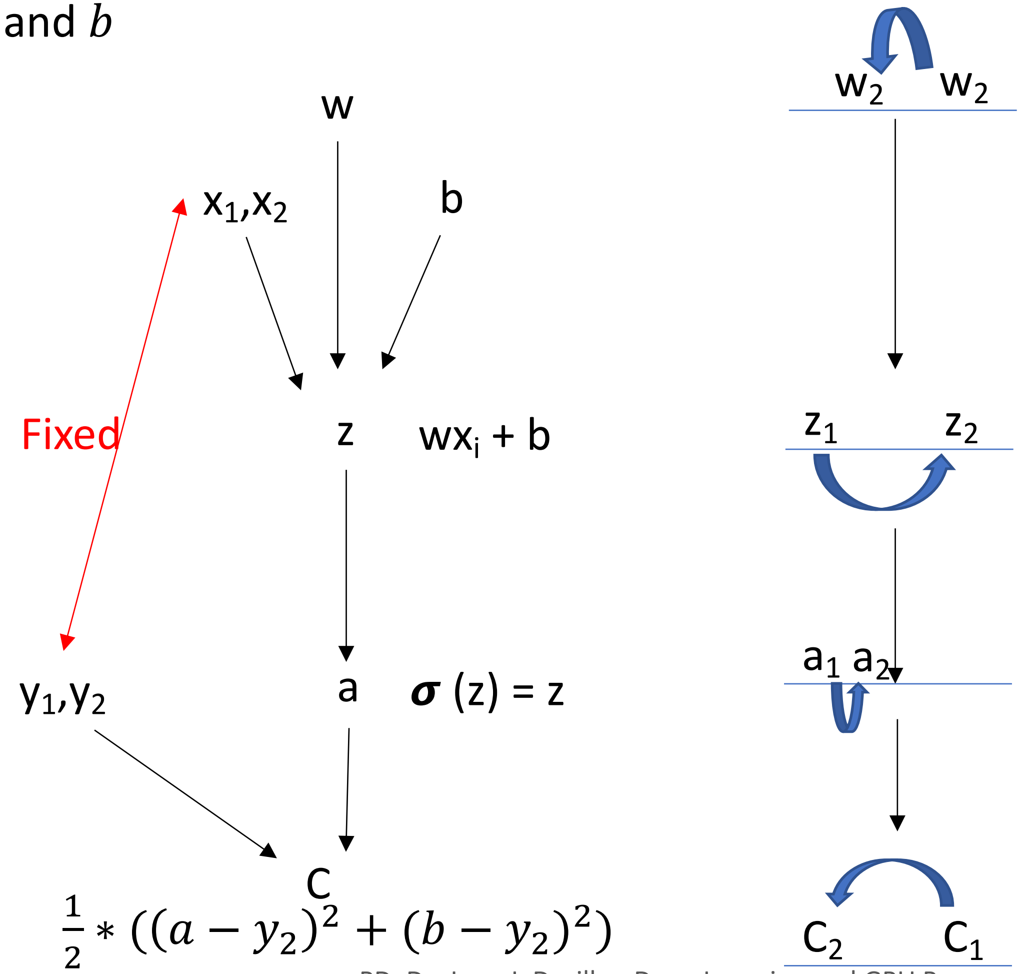
$$\frac{\partial C}{\partial a} = 2(a-y)$$

Gradient Vector

$$\begin{pmatrix} \frac{\partial C}{\partial w} \\ \frac{\partial C}{\partial b} \end{pmatrix} = \begin{pmatrix} 2x(a-y) \\ 2(a-y) \end{pmatrix}$$

Computing the Gradient

Gradient with two inputs that generates predictions: a and b



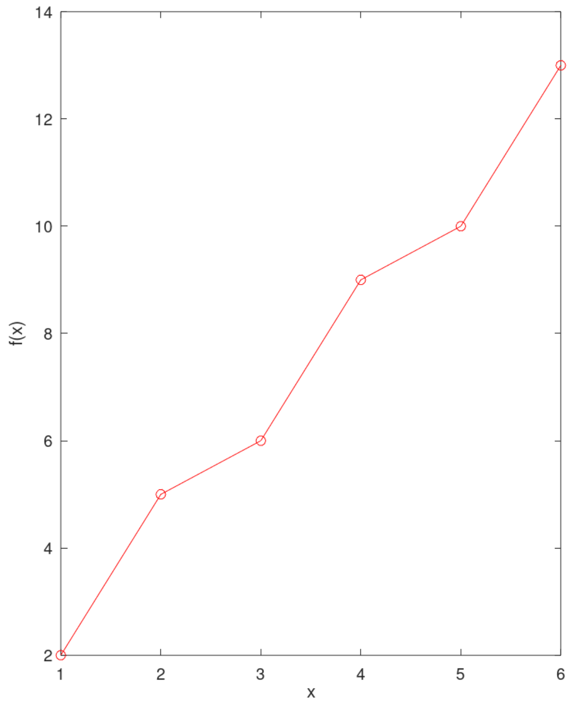
$$\frac{\partial C}{\partial w} = \frac{1}{2} * (2x(a - y_1) + 2x(b - y_2))$$

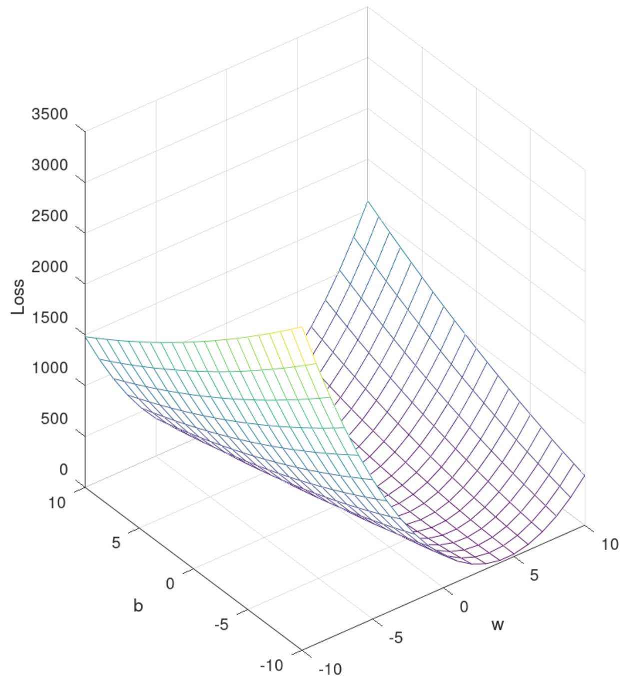
$$\frac{\partial C}{\partial b} = \frac{1}{2} * (2(a - y_1) + 2(b - y_2))$$

Gradient Vector

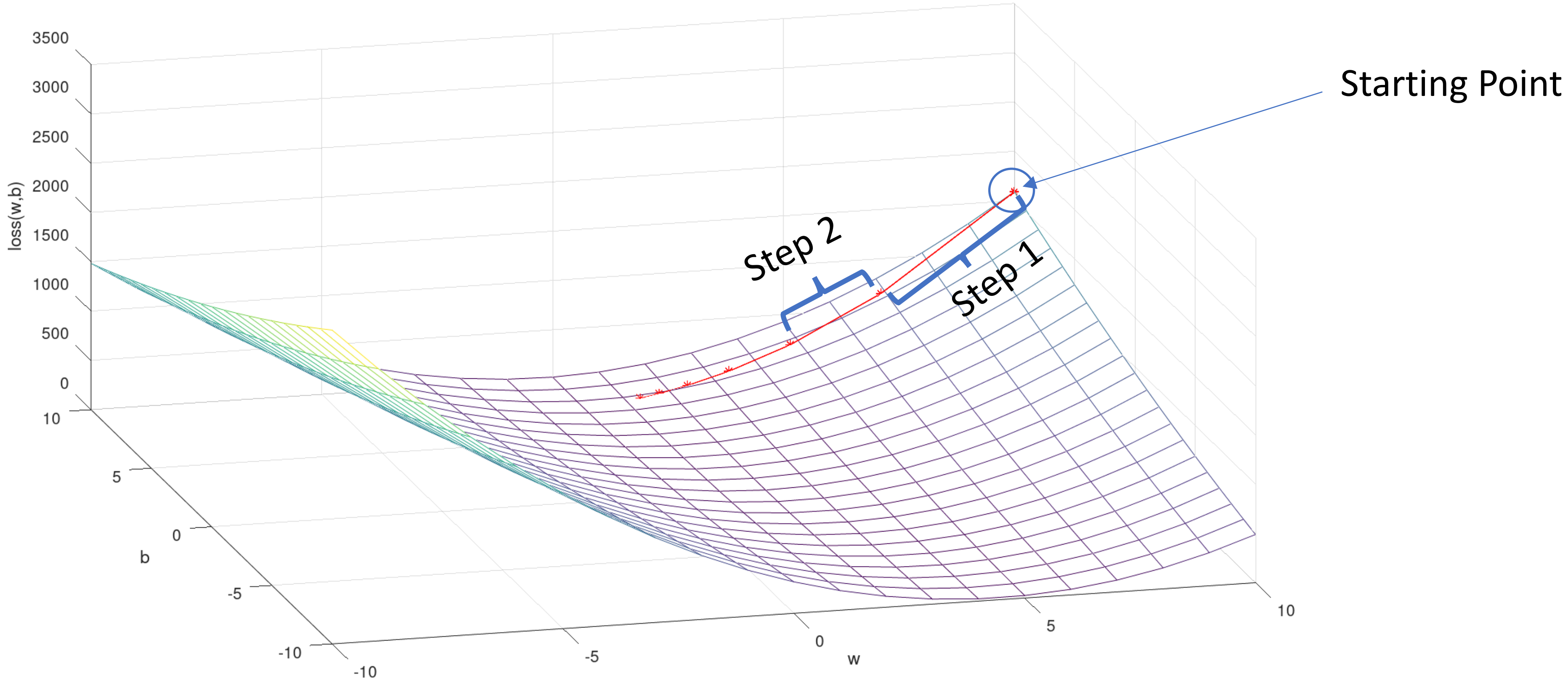
$$\begin{pmatrix} \frac{\partial C}{\partial w} \\ \frac{\partial C}{\partial b} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} * (2x(a - y_1) + 2x(b - y_2)) \\ \frac{1}{2} * (2(a - y_1) + 2(b - y_2)) \end{pmatrix}$$

Back Propagation and Gradient Descent



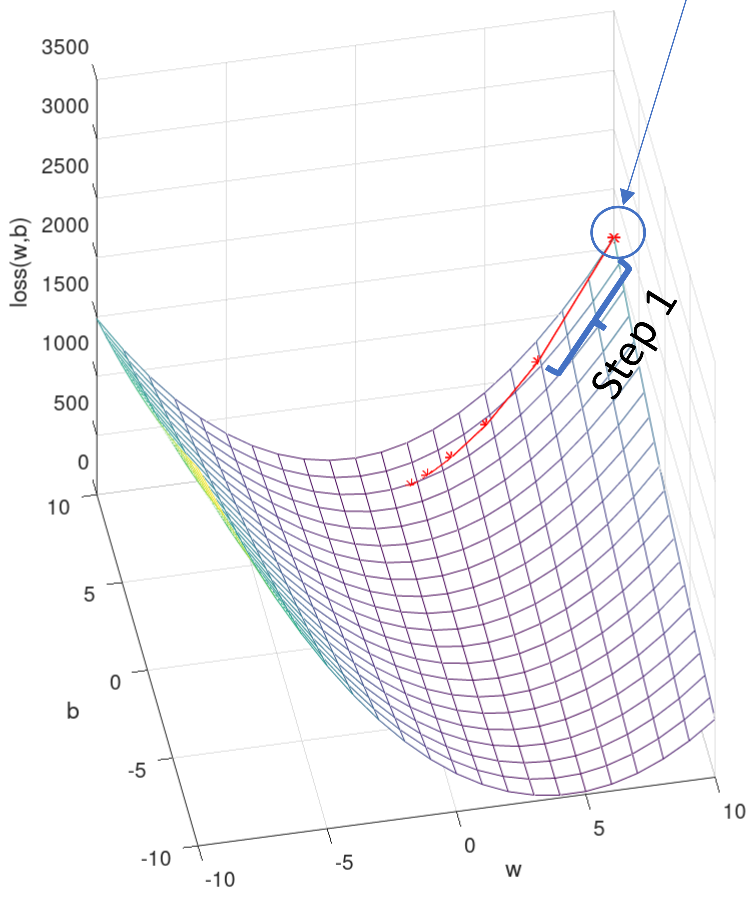


Back Propagation and Gradient Descent

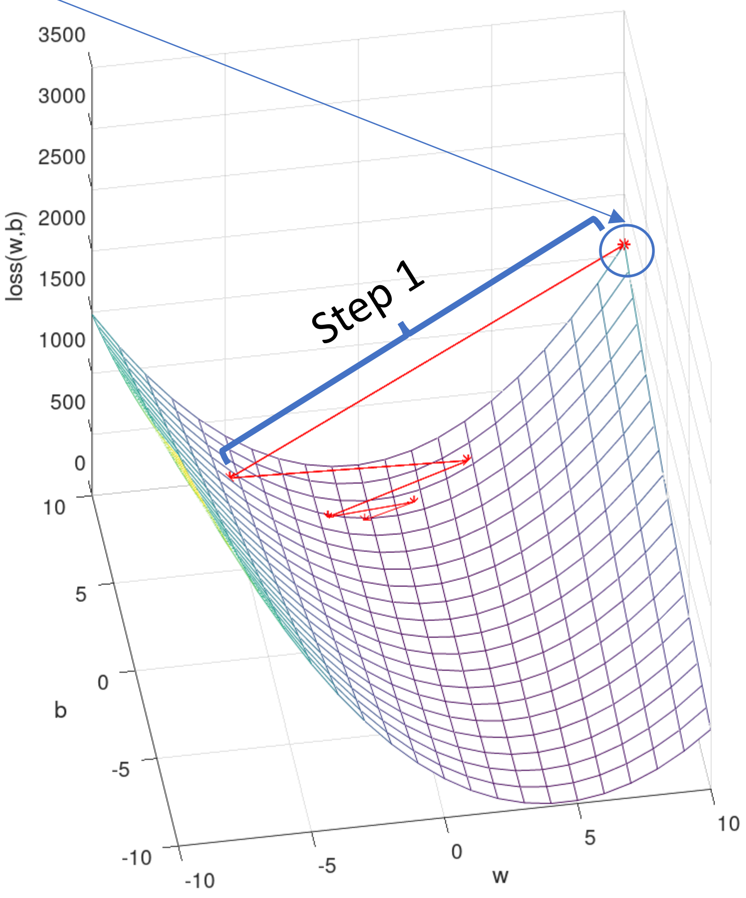


Back Propagation and Gradient Descent

Starting Point

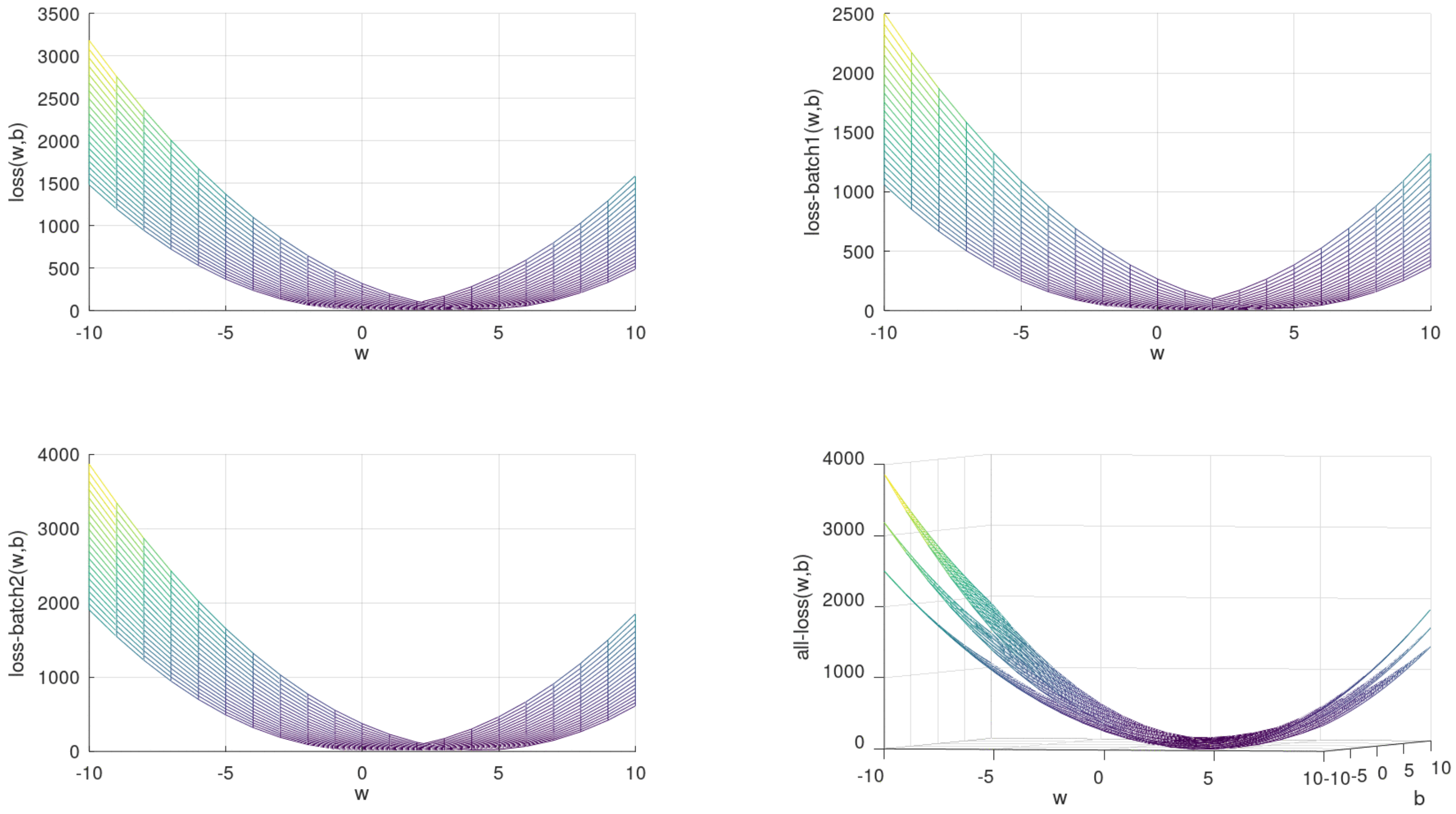


smaller learning rate

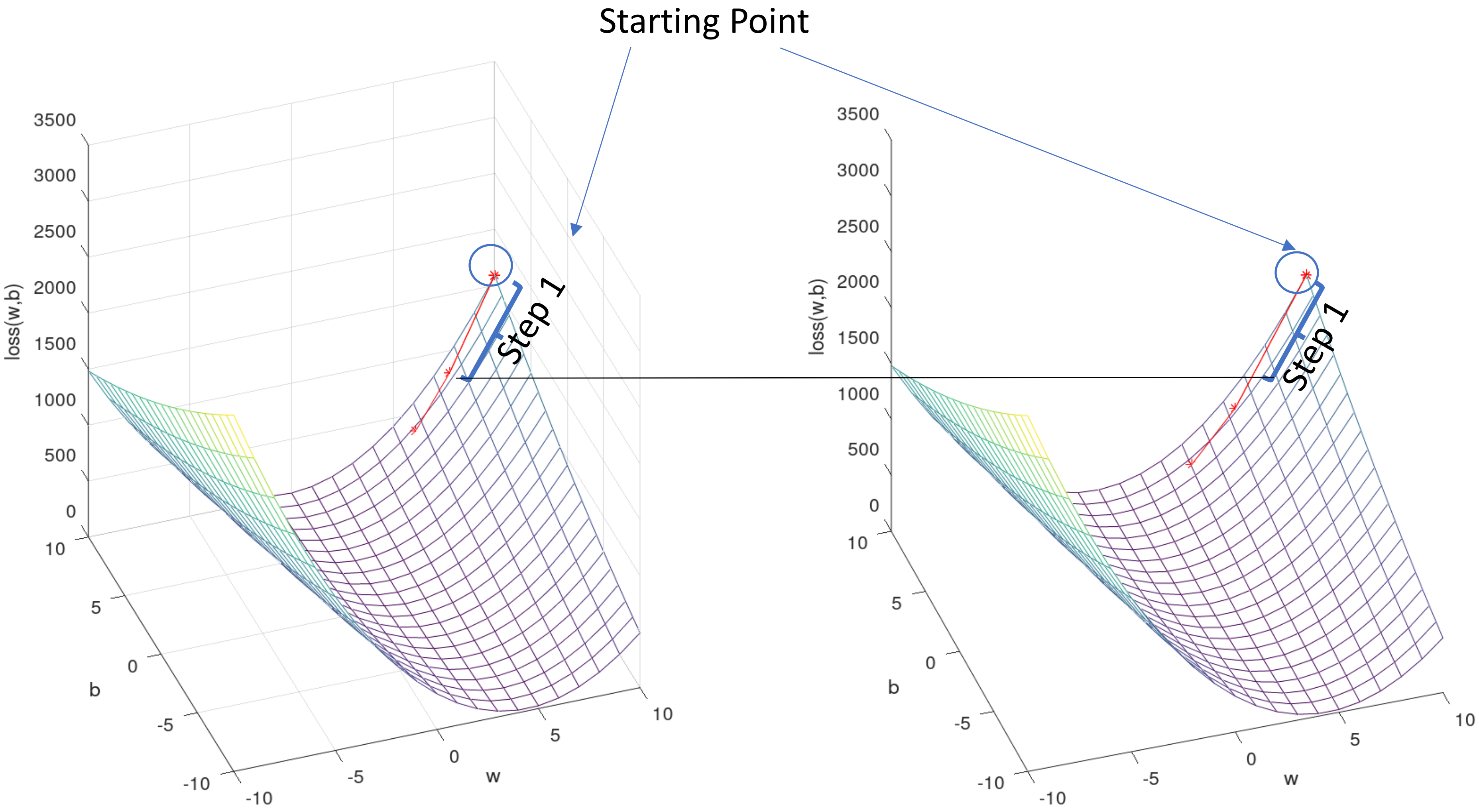


larger learning rate

Back Propagation and Gradient Descent

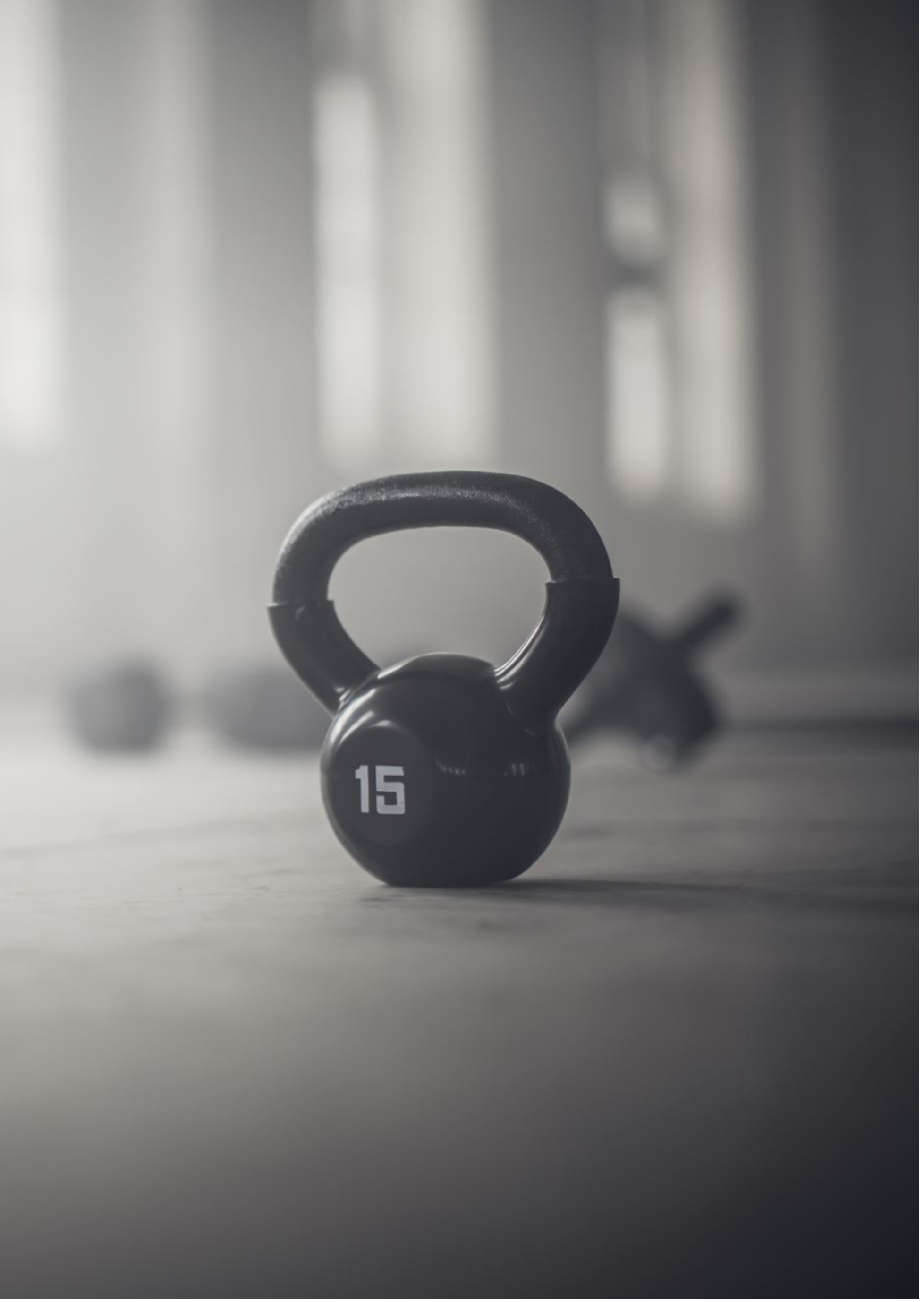


Back Propagation and Gradient Descent



gradient with batch 1

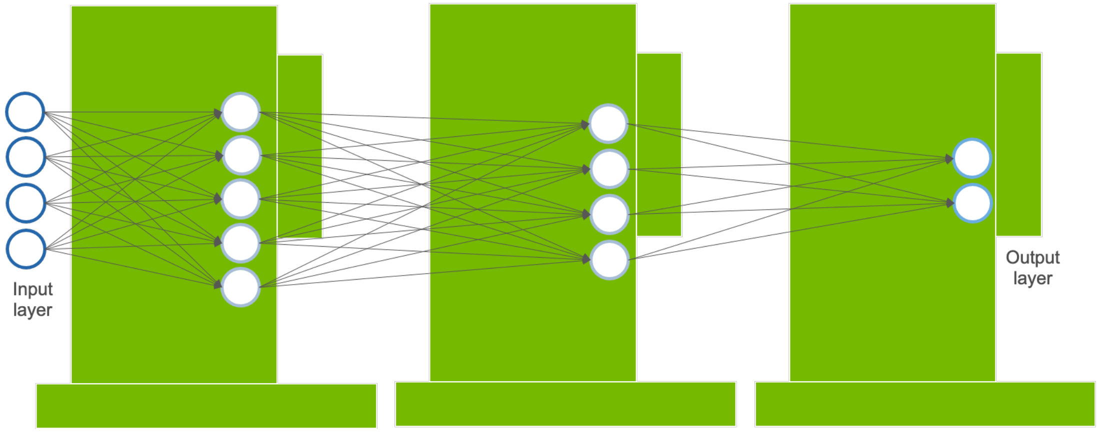
gradient with batch 2



Back Propagation and Gradient Descent

- Batch size implications
  - Smaller batches imply more steps per epoch:
    - More updates to weights --> More updates to the net
  - Smaller batches do not imply larger/smaller gradients

Parallel/Distributed ML Training



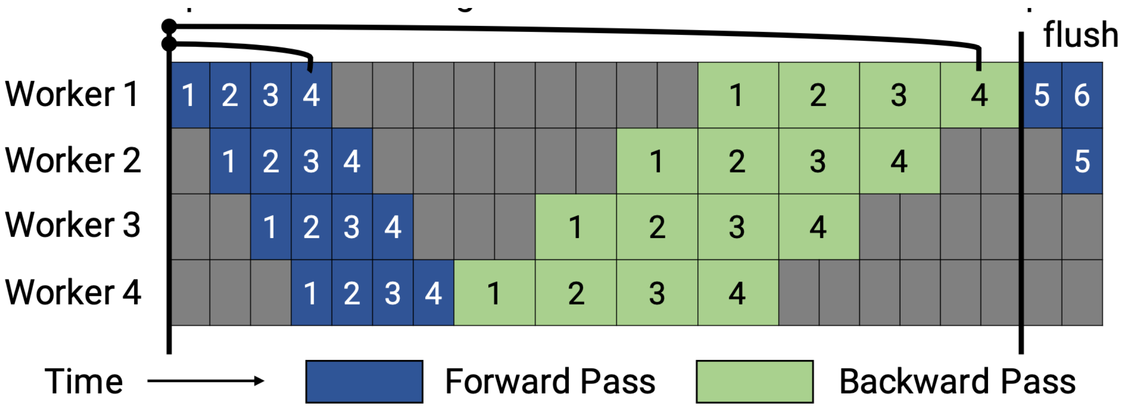
1. Model Parallelism: Memory usage and computation of a model distributed across devices

Two main variants:

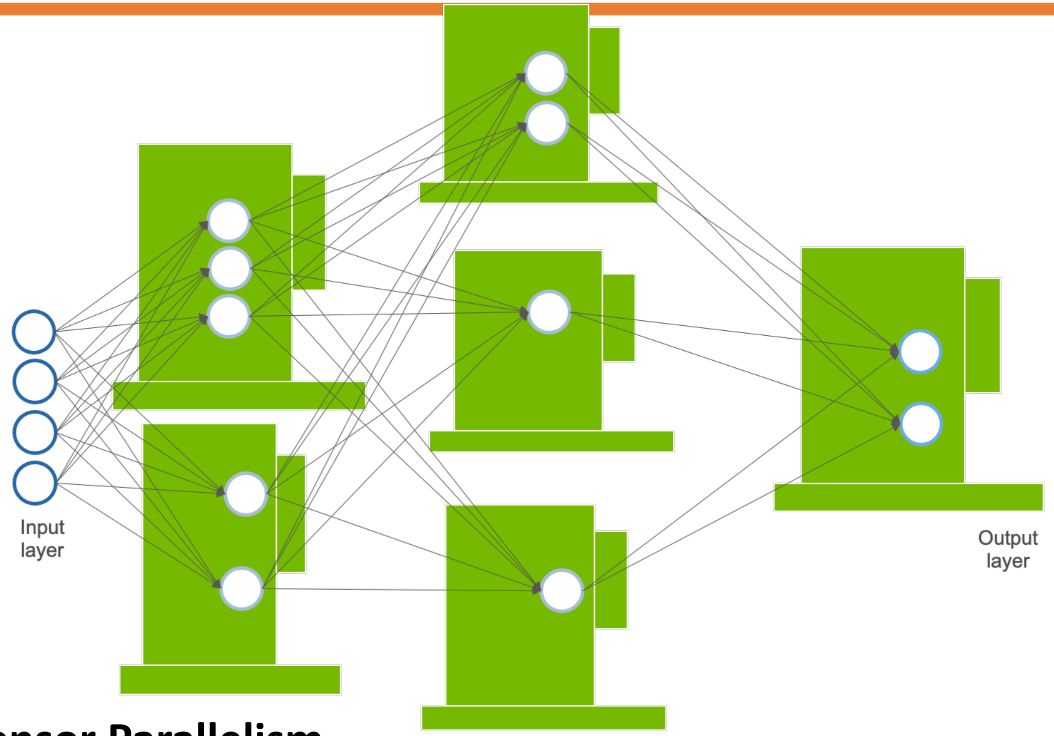
a) Pipeline parallelism
b) Tensor parallelism

Pipeline Model

- Complete layer per device
  - Weights stay within device
- Activations are communicated between GPUs
- Non efficient implementations may lead to inefficient usage of resources
  - Research area



Parallel/Distributed ML Training



1. Model Parallelism: Memory usage and computation of a model distributed across devices

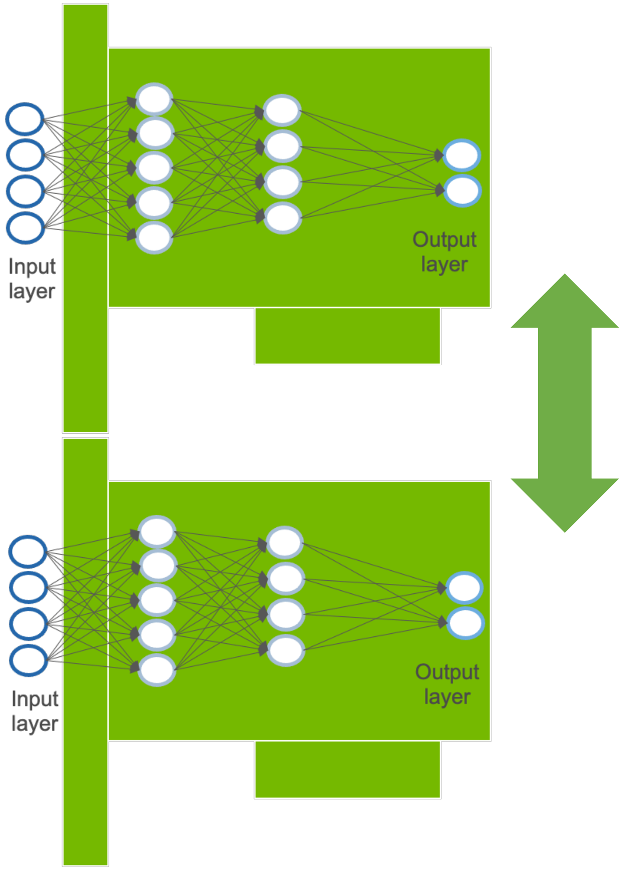
Two main variants:

a) Pipeline parallelism
b) Tensor parallelism

Tensor Parallelism

- Tensor operations (e.g., computing a layer output) distributed across device
  - Allows larger, more computationally expensive models
- Activations are communicated between GPUs
- Further points for inefficiencies
  - A device might depend on the activations computed by more than one device

Parallel/Distributed ML Training



2. Data Parallelism: Training mini-batch is split across devices

- Model must fit into the memory of a single device
- Weights are the same in each device
    - Gradients are communicated across all devices (all-to-all)