UiT The Arctic University of Norway

DTE-2502 Neural Networks: Multi-layer perceptron theory

Part 01: Backpropagation

Kalyan Ram Ayyalasomayajula, PhD

Associate Professor, UiT

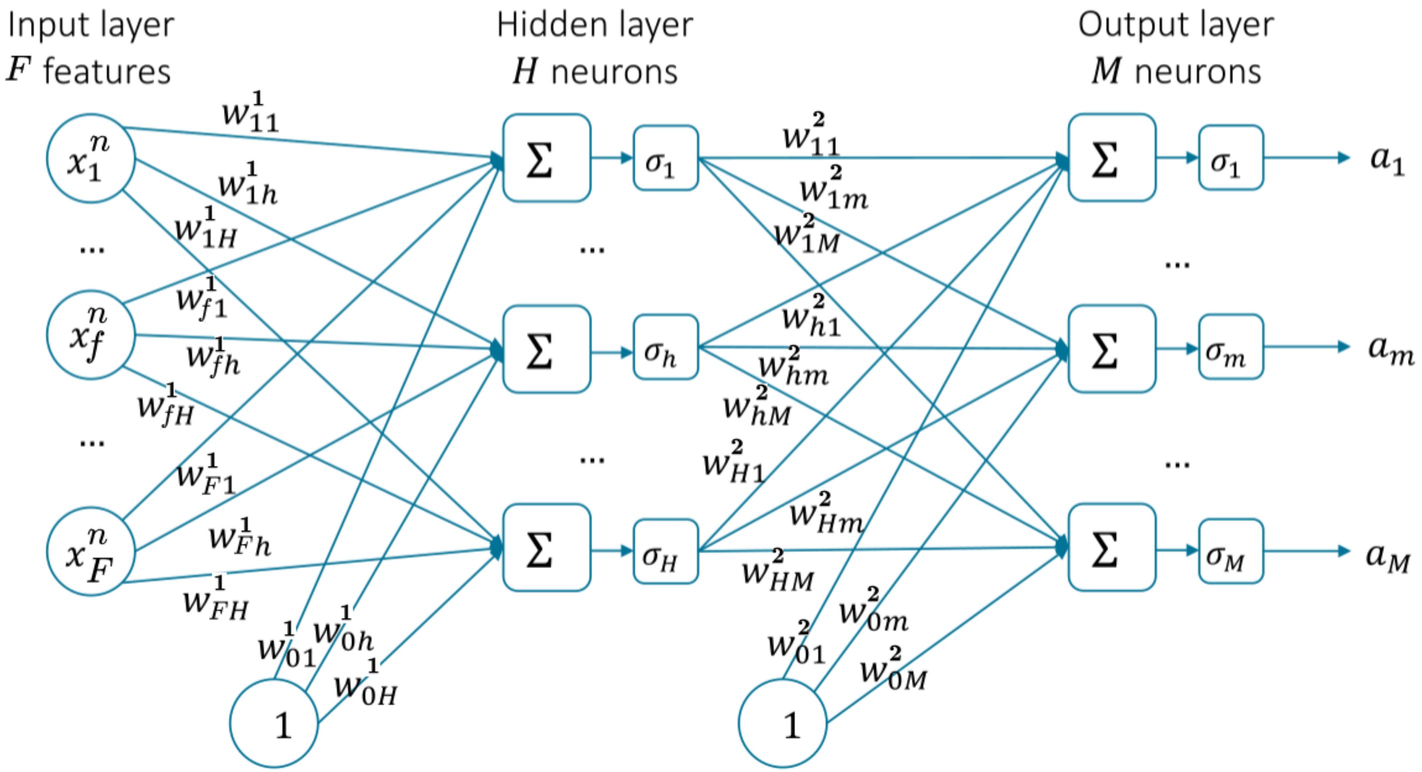
Email: kay001@post.uit.no

Overview

- Part 1: MLP theory
  - Multilayer perceptron architecture
  - Backpropagation
    - Stochastic gradient descent (SGD)
    - Gradient computation: Back propagation algorithm
- Part 2: Heuristics
  - Activation functions and their derivatives
  - Loss functions
  - Training NNs: Covergence and practical issues

Multilayer perceptron

$$\text{Input: } \mathbf{x}_n \in X = \{\mathbf{x}_1, \cdots, \mathbf{x}_n, \cdots, \mathbf{x}_N\},$$
$$\mathbf{x}_n = (x_1^n, \cdots, x_f^n, \cdots, x_F^n)$$
$$\text{Output: } \mathbf{y}_n \in Y = \{\mathbf{y}_1, \cdots, \mathbf{y}_n, \cdots, \mathbf{y}_N\},$$
$$\mathbf{y}_n = (y_1^n, \cdots, y_m^n, \cdots, y_M^n)$$
$$\text{Parameters: } W = \left(\mathbf{w}^1 = \{w_{11}^1, \cdots, w_{fh}^1, \cdots, w_{FH}^1\}, \quad \mathbf{w}^2 = \{w_{11}^2, \cdots, w_{hm}^2, \cdots, w_{HM}^2\}\right)$$



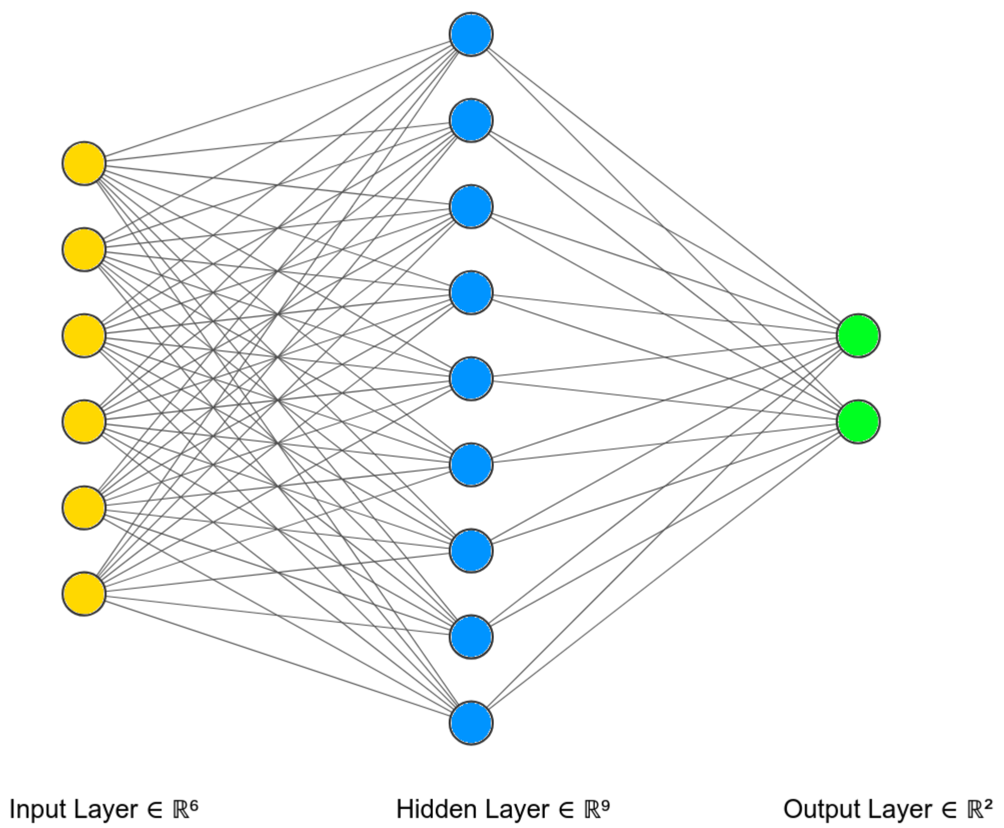
Multilayer perceptron: example

id	height	weight	bone density	femur length	jawline width	jawline length	gender
112	125	56	1.2	32	3	10	F
208	189	90	2.5	48	5	15	M
///	///						
///		///					
///			///				
///				///			
///					///		
///						///	
232	150	70	1.5	37	3.2	10.5	F

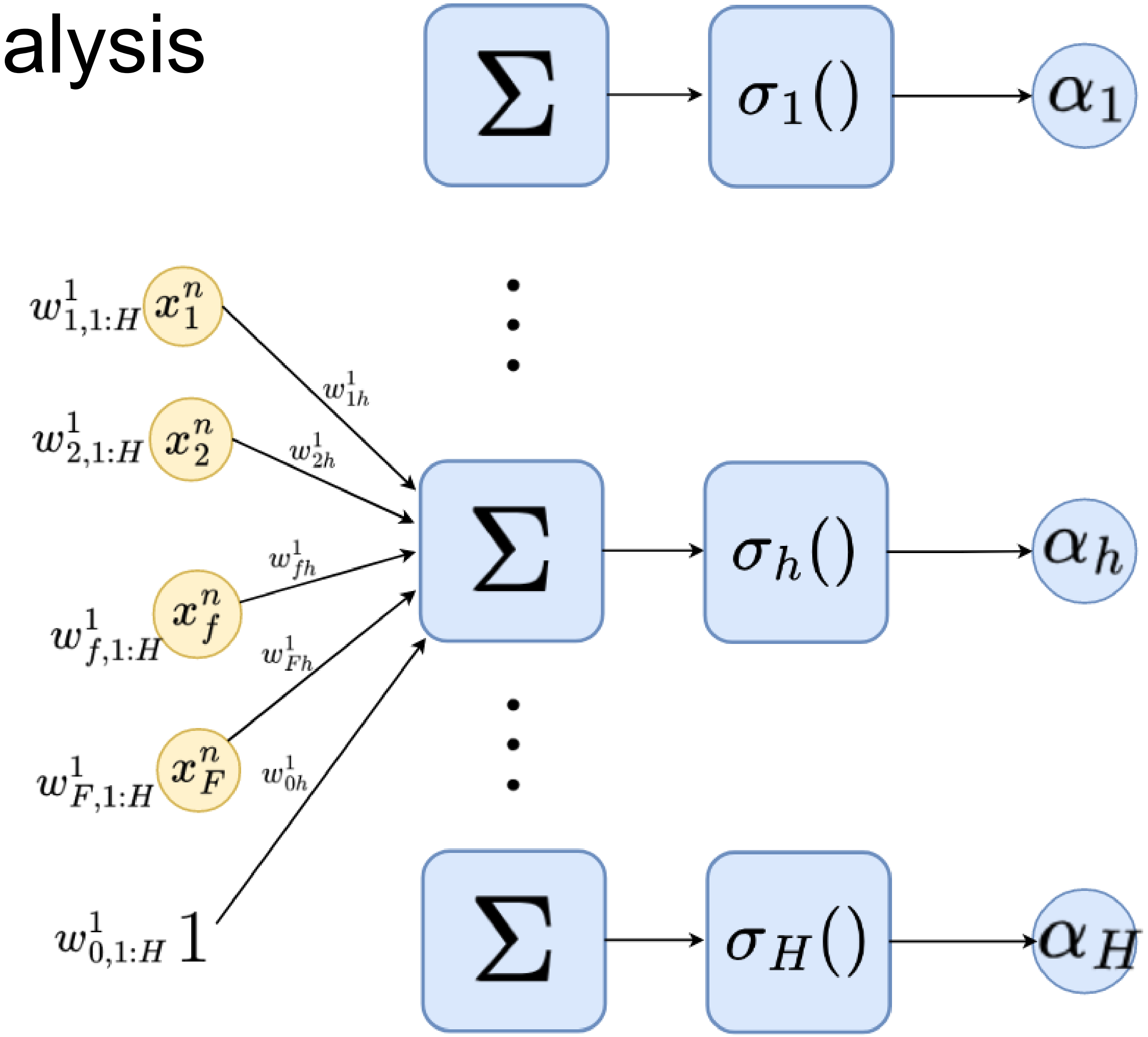
$$\text{Input: } \mathbf{x}_n \in X = \{\mathbf{x}_1, \cdots, \mathbf{x}_n, \cdots, \mathbf{x}_N\},$$
$$\mathbf{x}_n = (x_1^n, \cdots, x_f^n, \cdots, x_F^n)$$

$$\text{Output: } \mathbf{y}_n \in Y = \{\mathbf{y}_1, \cdots, \mathbf{y}_n, \cdots, \mathbf{y}_N\},$$
$$\mathbf{y}_n = (y_1^n, \cdots, y_m^n, \cdots, y_M^n)$$

$$\text{Parameters: } W = \Big(\mathbf{w}^1 = \{w_{11}^1, \cdots, w_{fh}^1, \cdots, w_{FH}^1\},$$
$$\mathbf{w}^2 = \{w_{11}^2, \cdots, w_{hm}^2, \cdots, w_{HM}^2\}\Big)$$



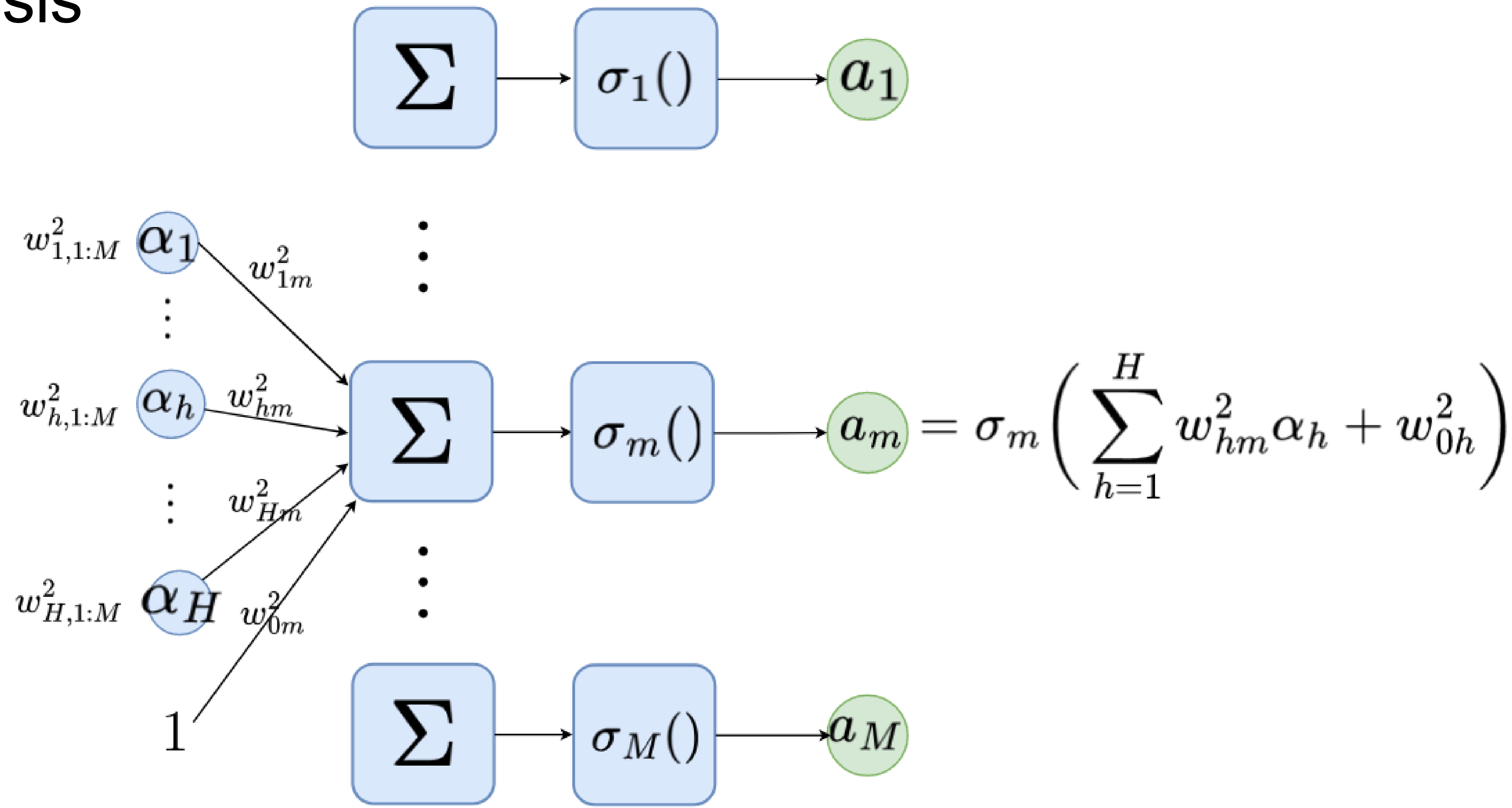
Layer 1 analysis



$$\alpha_h = \sigma_h\left(\sum_{f=1}^{F} w^1_{fh} x^n_f + w^1_{0h}\right)$$

1

Layer 2 analysis



1

Forward phase: sample-wise

Given an input; $\mathbf{x}_n = (x_1^n, \cdots, x_f^n, \cdots, x_F^n)$

and its corresponding output; $\mathbf{y}_n = (y_1^n, \cdots, y_m^n, \cdots, y_M^n)$

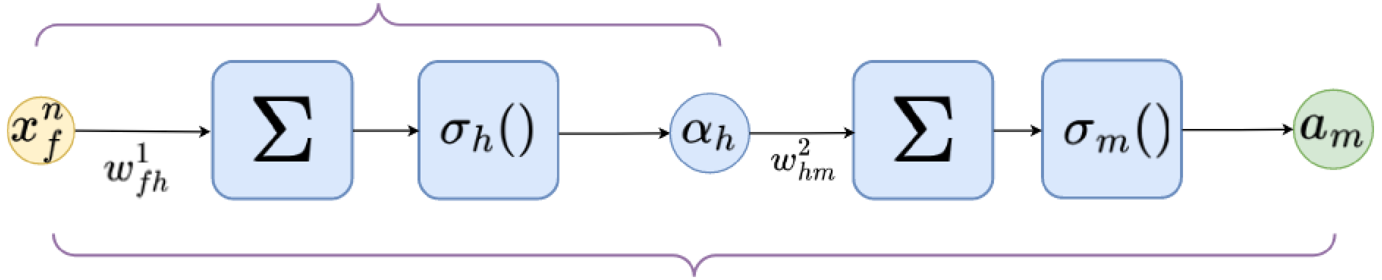
Hidden layer output; $\vec{\alpha} = (\alpha_1, \cdots, \alpha_h, \cdots, \alpha_H)$

and predicted output; $\mathbf{a} = (a_1, \cdots, a_m, \cdots, a_M)$

Consider the loss function:

$$\boxed{\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2) = \frac{1}{2} \sum_{m=1}^{M} \left(a_m(\mathbf{x}_n) - y_m^n \right)^2}$$

$$\alpha_h(\mathbf{x}_n) = \sigma_h\left(\sum_{f=0}^{F} w_{fh}^1 x_f^n \right)$$



$$a_m(\mathbf{x}_n) = \sigma_m\left(\sum_{h=0}^{H} w_{hm}^2 \alpha_h(\mathbf{x}_n) \right)$$

Loss function gradients: sample-wise (1)

$$\boxed{a_m(\mathbf{x}_n) = \sigma_m\left(\sum_{h=0}^{H} w_{hm}^2 \alpha_h\right)}$$

The loss function for a given input $\mathbf{x}_n$ is dependent on the weights $\mathbf{w}^1, \mathbf{w}^2$ as

$$\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2) = \frac{1}{2}\sum_{m=1}^{M}\Big(a_m(\mathbf{x}_n) - y_m^n\Big)^2$$

The weight updates in the output layer w_{hm}^2 $w.r.t$ the loss $\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)$ is:

$$\frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial w_{hm}^2} = \frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial a_m} \cdot \frac{\partial a_m}{\partial w_{hm}^2}$$

$$\frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial a_m} = \varepsilon_m(\mathbf{x_n}) = \Big(a_m(\mathbf{x_n}) - y_m^n\Big);\ \text{prediction error}$$

$$\frac{\partial a_m}{\partial w_{hm}^2} = \sigma'_m() \cdot \alpha_h(\mathbf{x_n})$$

$$\therefore \frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial w_{hm}^2} = \varepsilon_m(\mathbf{x_n}) \cdot \sigma'_m() \cdot \alpha_h(\mathbf{x_n})$$

Loss function gradients: sample-wise (2)

$$\boxed{\begin{aligned} \alpha_h(\mathbf{x}_n) &= \sigma_h\left(\sum_{f=0}^{F} w_{fh}^1 x_f^n\right) \\ a_m(\mathbf{x}_n) &= \sigma_m\left(\sum_{h=0}^{H} w_{hm}^2 \alpha_h\right) \end{aligned}}$$

The weight updates in the hidden layer w_{fh}^1 $w.r.t$ the loss $\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)$ is:

$$\frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial w_{fh}^1} = \frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial w_{fh}^1}$$

$$\frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial \alpha_h} = \sum_{m=1}^{M} \varepsilon_m(\mathbf{x_n}) \cdot \sigma_m'() \cdot w_{hm}^2, \qquad \frac{\partial \alpha_h}{\partial w_{fh}^1} = \sigma_h'() \cdot x_f^n$$

$$\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2) = \frac{1}{2}\left[\left(a_1(\mathbf{x}_n) - y_1^n\right)^2 + \cdots + \left(a_m(\mathbf{x}_n) - y_m^n\right)^2 + \cdots + \left(a_M(\mathbf{x}_n) - y_M^n\right)^2\right]$$

$$= \frac{1}{2}\left[\left(\sigma_1\left(\sum_{h=0}^{H} w_{h1}^2 \alpha_h\right) - y_1^n\right)^2 + \cdots + \left(\sigma_m\left(\sum_{h=0}^{H} w_{hm}^2 \alpha_h\right) - y_m^n\right)^2 + \cdots + \left(\sigma_M\left(\sum_{h=0}^{H} w_{hM}^2 \alpha_h\right) - y_M^n\right)^2\right]$$

Loss function and gradients: whole data

$$\mathcal{L}(\mathbf{w}^1, \mathbf{w}^2) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)$$

$$\alpha_h(\mathbf{x}_n) = \sigma_h\left(\sum_{f=0}^{F} w^1_{fh} x^n_f\right)$$

$$a_m(\mathbf{x}_n) = \sigma_m\left(\sum_{h=0}^{H} w^2_{hm} \alpha_h\right)$$

$$\frac{\partial \mathcal{L}(\mathbf{w}^1, \mathbf{w}^2)}{\partial w^1_{fh}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial w^1_{fh}}$$

$$\frac{\partial \mathcal{L}(\mathbf{w}^1, \mathbf{w}^2)}{\partial w^2_{hm}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial w^2_{hm}}$$

Backward phase. Gradient computation

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{hm}} = \underbrace{\frac{\partial \mathcal{L}_i(w)}{\partial a_m}}_{a_m(x_i) - y_m(x_i)} \underbrace{\frac{\partial a_m}{\partial w_{hm}}}_{\sigma'_m \alpha_h(x_i)}$$

$a_m(x_i) - y_m(x_i)$ $\xrightarrow{}$ output layer error

$$\underbrace{\varepsilon_m(x_i)}$$

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{hm}} = \varepsilon_m(x_i)\ \sigma'_m \alpha_h(x_i),$$

$$m = 1, \ldots, M, h = 0, \ldots, H$$

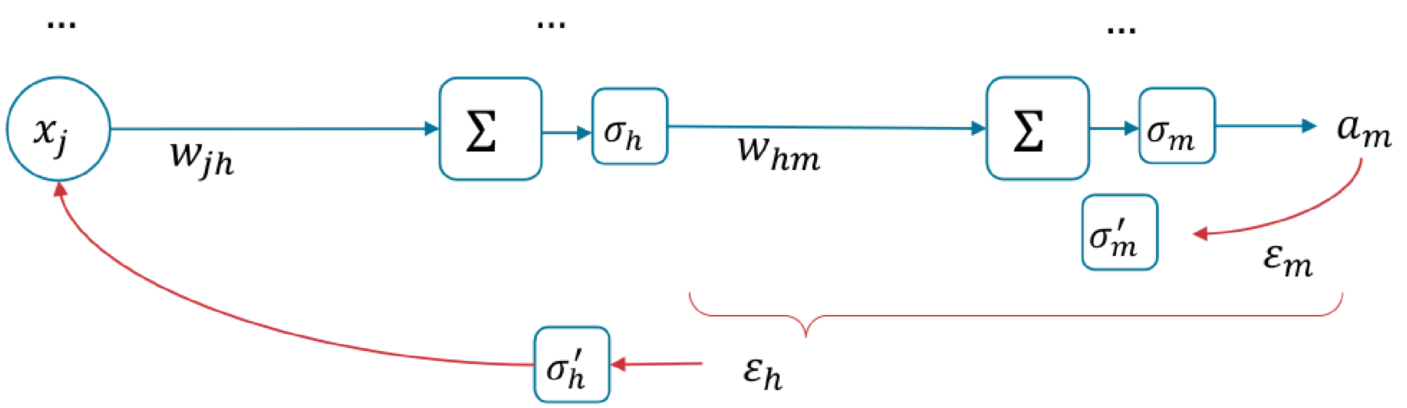
$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{jh}} = \underbrace{\frac{\partial \mathcal{L}_i(w)}{\partial u_h}}_{\sum_{m=1}^{M} \varepsilon_m(x_i) \sigma'_m w_{hm}} \underbrace{\frac{\partial \alpha_h}{\partial w_{jh}}}_{\sigma'_h x_j}$$

$\sum_{m=1}^{M} \varepsilon_m(x_i) \sigma'_m w_{hm}$ $\xrightarrow{}$ hidden layer error

$$\underbrace{\varepsilon_h(x_i)}$$

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{jh}} = \varepsilon_h(x_i)\ \sigma'_h x_j,$$

$$h = 1, \ldots, H, j = 0, \ldots, n$$



Backpropagation algorithm

Input: $X^l = (x_i, y_i)_{i=1}^l \subset \mathbb{R}^n \times \mathbb{R}^M$, parameters H, μ, λ
Output: weights w_{jh}, w_{hm}

Initialize w_{jh}, w_{hm}
do

select x_i from X^l
forward phase:

$$u_h(x_i) = \sigma_h\left(\textstyle\sum_{j=0}^{n} w_{jh} x_j\right), \quad h = 1, \dots, H$$

$$a_m(x_i) = \sigma_m\left(\textstyle\sum_{h=0}^{H} w_{hm} \alpha_h(x_i)\right), \quad m = 1, \dots, M$$

$$\varepsilon_m(x_i) = \frac{\partial \mathcal{L}_i(w)}{\partial a_m} = a_m(x_i) - y_m(x_i), \quad m = 1, \dots, M$$

$Q \coloneqq \lambda \mathcal{L}_i + (1 - \lambda) Q$
backward phase:

$$\varepsilon_h(x_i) = \textstyle\sum_{m=1}^{M} \varepsilon_m(x_i) \sigma'_m w_{hm}, \quad h = 1, \dots, H$$

gradient step:

$$w_{hm} = w_{hm} - \mu \varepsilon_m(x_i) \sigma'_m \alpha_h(x_i), \quad h = 0, \dots, H, \ m = 1, \dots, M$$

$$w_{jh} = w_{jh} - \mu \varepsilon_h(x_i) \sigma'_h x_j, \quad j = 0, \dots, n, \ h = 1, \dots, H$$

until Q converges

UiT The Arctic University of Norway

DTE-2502 Neural Networks: Multi-layer perceptron theory

Part 02: Gradient descent algorithms and Heuristics

Kalyan Ram Ayyalasomayajula, PhD

Associate Professor, UiT

Email: kay001@post.uit.no

III Loss functions

Regression loss functions

- Mean squared error (MSE) loss

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(a_i - y_i)^2$$

- Mean squared logarithmic error (MSLE) loss

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(\log(y_i + 1) - \log(a_i + 1))^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\log\left(\frac{y_i + 1}{a_i + 1}\right)\right)^2$$

- Mean absolute error (MAE) loss

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}|a_i - y_i|$$

Clarification 2. On way to understand the utility of MSLE loss is when we does not penalize as heavily for smaller % errors. As per the given example

$a\|$	$y\|$	$a - y\|$	$\% = \frac{\|a-y\|}{y} \times 100\|$	$MSE\|$	$MSLE$
3	5	−2	40	4	0.22
3000	3200	−200	6.25	40000	0.02

We can see that the % percentage error in first and second cases are 40% and 6.25% respectively. MSLE loss penalizes the second case less severely unlike MSE loss.

Classification loss functions

- Categorical crossentropy loss

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \log a_i$$

- Binary crossentropy loss

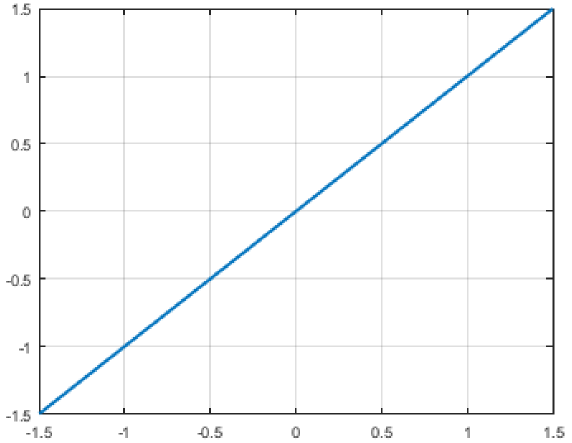
$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log a_i + (1 - y_i)\log(1 - a_i)$$

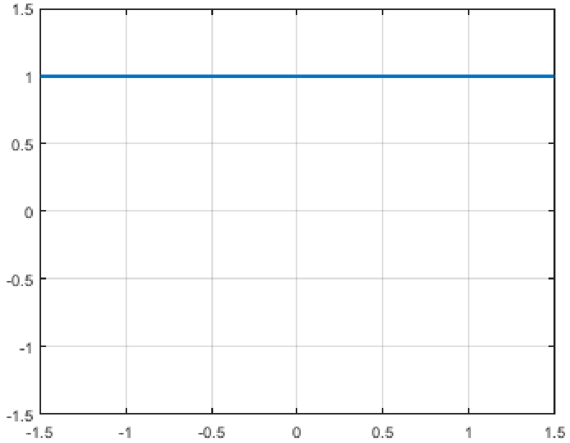
- Squared hinge loss

$$\mathcal{L} = \sum_{i=1}^{N} (\max\{0, 1 - y_i a_i\}^2)$$

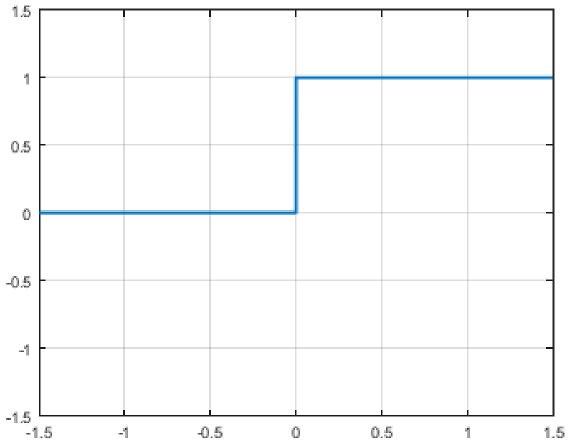
IV Activation functions and their derivatives

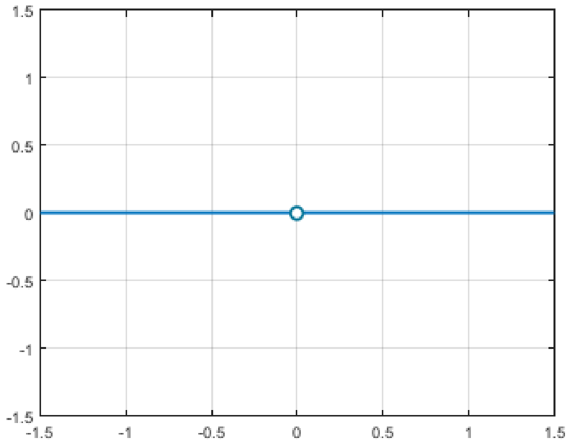
Identity $\sigma(x) = x$ $\sigma'(x) = 1$





Sign $\sigma(x) = \mathrm{sign}(x)$ $\sigma'(x) = 0$ everywhere except $x = 0$

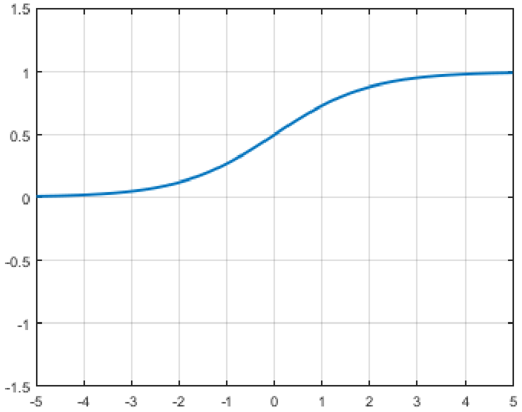


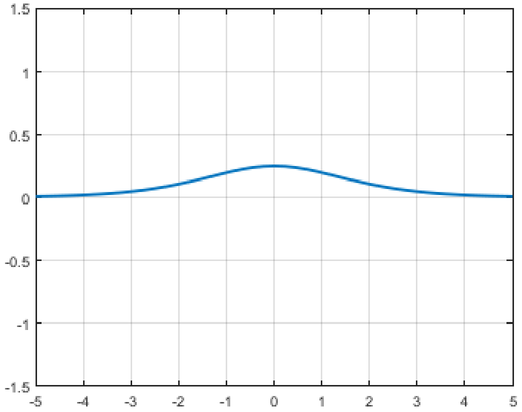


Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(1 - \sigma)$$

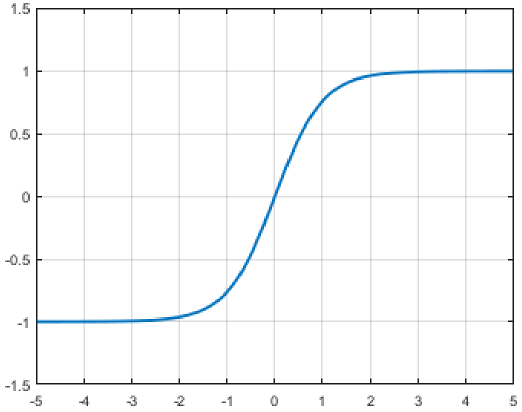


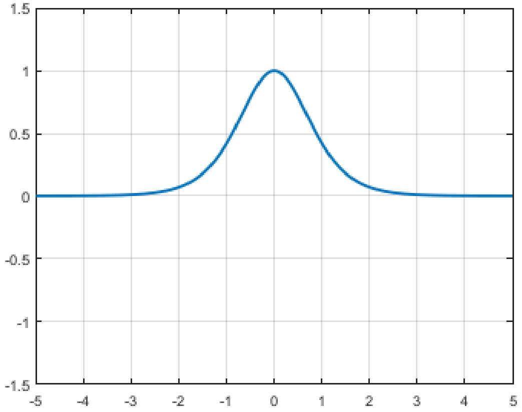


tanh

$$\sigma(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$\sigma'(x) = \frac{4e^{2x}}{(e^{2x} + 1)^2} = 1 - \sigma^2$$



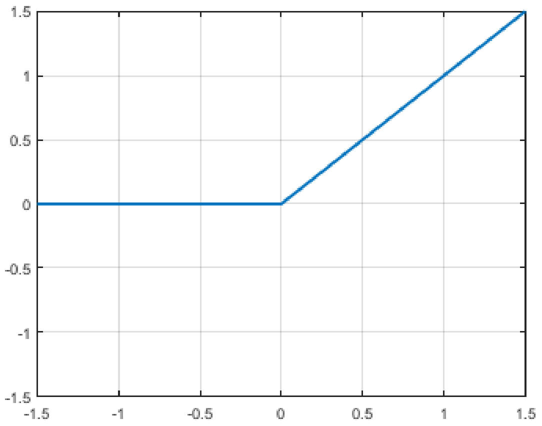


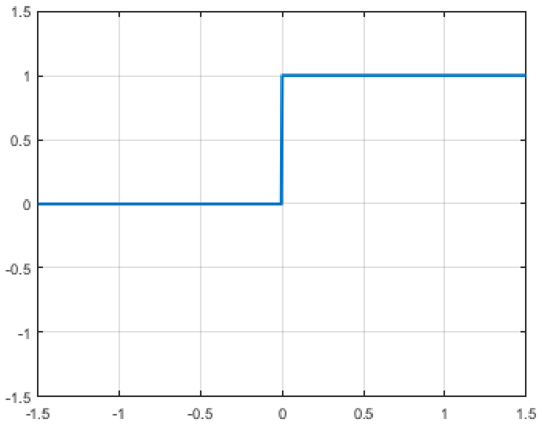
ReLU

Rectified Linear Unit

$$\sigma(x) = \max\{x, 0\}$$

$$\sigma'(x) = \mathrm{sign}(x)$$

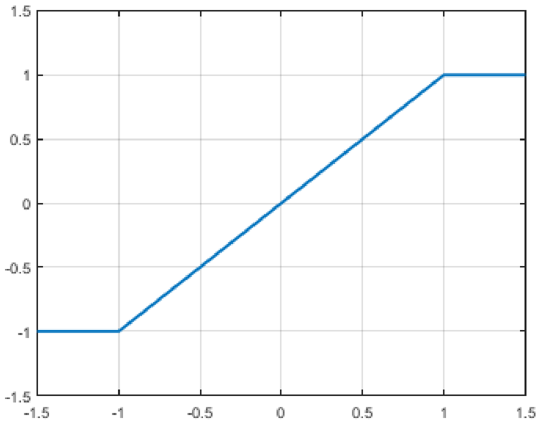


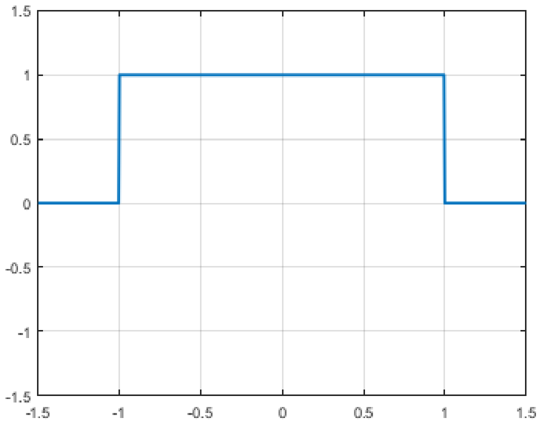


Hard tanh

$$\sigma(x) = \max\{\min\{x, 1\}, -1\}$$

$$\sigma'(x) = \begin{cases} 1, & x \in [-1,1] \\ 0, & \text{otherwise} \end{cases}$$





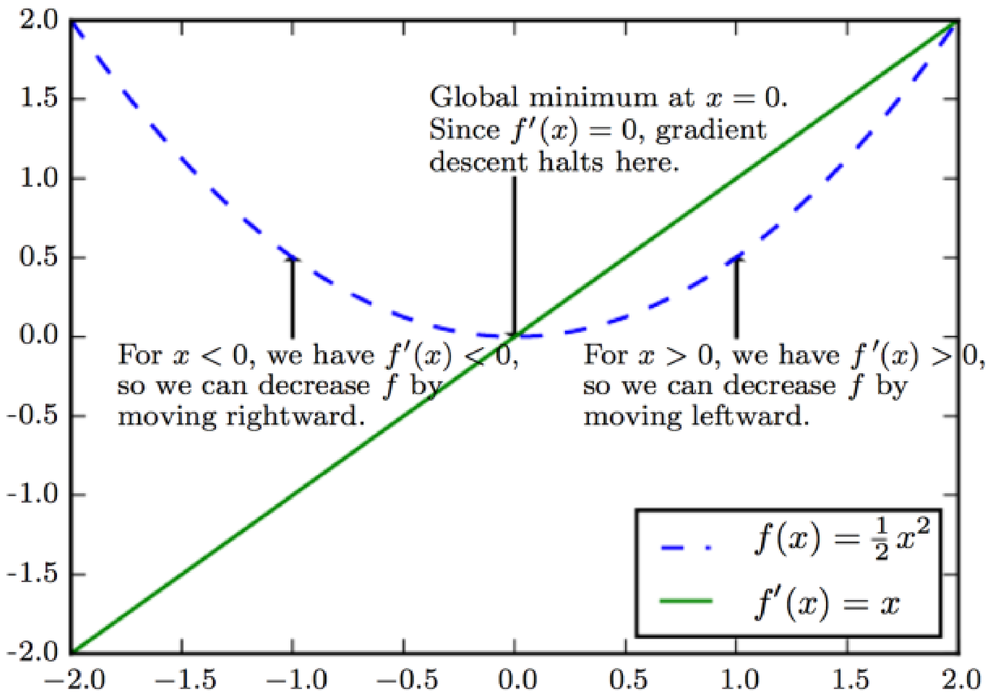
Gradient descent algorithms

Gradient-Based Optimization

- Most ML algorithms involve optimization
- Minimize/maximize a function $f(\boldsymbol{x})$ by altering $\boldsymbol{x}$
  - Usually stated a minimization
  - Maximization accomplished by minimizing $-f(\boldsymbol{x})$
- $f(\boldsymbol{x})$ referred to as objective or criterion
  - In minimization also referred to as loss function cost, or error
  - Example is linear least squares $\boxed{f(x) = \frac{1}{2} \| Ax - b \|^2}$
  - Denote optimum value by $\boldsymbol{x}^* = \operatorname{argmin} f(\boldsymbol{x})$

Gradient Descent Illustrated

- Given function is $f(x)=½\ x^2$ which has a bowl shape with global minimum at $x=0$
  - Since $f'(x)=x$
    - For $x>0$, $f(x)$ increases with x and $f'(x)>0$
    - For $x<0$, $f(x)$ decreases with x and $f'(x)<0$
- Use $f'(x)$ to follow function downhill
  - Reduce $f(x)$ by going in direction opposite sign of derivative $f'(x)$

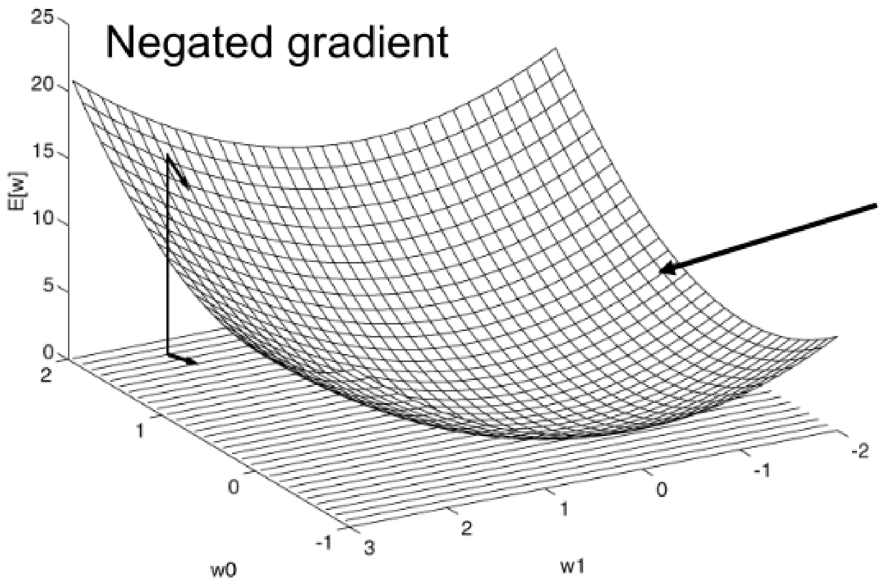


Minimizing with multiple inputs

- We often minimize functions with multiple inputs: $f\colon \mathbb{R}^n \to \mathbb{R}$
- For minimization to make sense there must still be only one (scalar) output

Application in ML: Minimize Error

Direction in w_0-w_1 plane producing steepest descent



Error surface shows desirability of every weight vector, a parabola with a single global minimum

- Gradient descent search determines a weight vector w that minimizes $E(\mathrm{w})$ by
  - Starting with an arbitrary initial weight vector
  - Repeatedly modifying it in small steps
  - At each step, weight vector is modified in the direction that produces the steepest descent along the error surface

Definition of Gradient Vector

- The *Gradient (derivative)* of E with respect to each component of the vector w

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n} \right]$$

- Notice $\nabla E[\vec{w}]$ is a vector of partial derivatives
- Specifies the direction that produces steepest increase in E
- Negative of this vector specifies direction of steepest decrease

Gradient Descent Rule

$$\boxed{\vec{w} \leftarrow \vec{w} + \overrightarrow{\Delta w}}$$

– where

$$\Delta \overrightarrow{w} = -\eta \nabla E[\overrightarrow{w}]$$

– η is a positive constant called the learning rate

  - Determines step size of gradient descent search

- Component Form of Gradient Descent

  – Can also be written as

$$w_i \leftarrow w_i + \Delta w_i$$

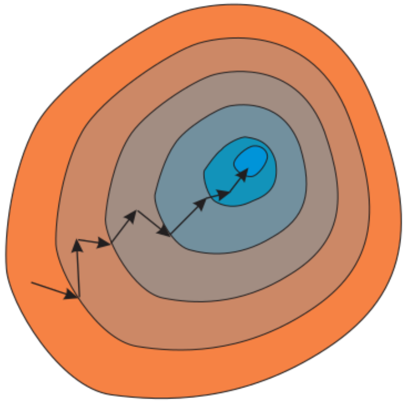
  - where

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradient descent

Minimization of average loss over the training data

$$Q(w) = \frac{1}{l}\sum_{i=1}^{l}\mathcal{L}_i(w) \to \min_w$$



Input: dataset X^l, learning rate μ, parameter λ
Output: weights $w = (w_{jh}, w_{hm})$

Initialization
Set all the weights w to small random numbers
Evaluate the objective function $Q(w)$

do
select x_i from X^l
compute the loss function $\mathcal{L}_i(w)$
gradient step $w \coloneqq w - \mu\mathcal{L}_i'(w)$
update the objective function $Q \coloneqq \lambda\mathcal{L}_i + (1-\lambda)Q$
until Q and/or w converges

$$\boxed{\mathcal{L}(\mathbf{w}^1, \mathbf{w}^2) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}$$

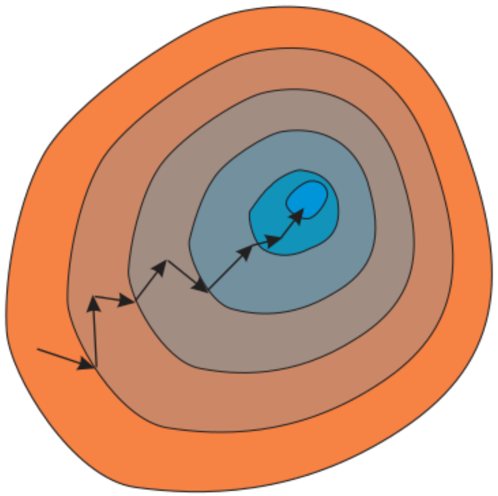
$$\frac{\partial\mathcal{L}(\mathbf{w}^1, \mathbf{w}^2)}{\partial w^1_{fh}} = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial w^1_{fh}}$$

$$\frac{\partial\mathcal{L}(\mathbf{w}^1, \mathbf{w}^2)}{\partial w^2_{hm}} = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial w^2_{hm}}$$

Stochastic gradient descent

Minimization of average loss over the training data

$$Q(w) = \frac{1}{l}\sum_{i=1}^{l} \mathcal{L}_i(w) \to \min_w$$



Input: dataset X^l, learning rate μ, parameter λ
Output: weights $w = \left(w_{jh}, w_{hm}\right)$

Initialization
Set all the weights w to small random numbers
Evaluate the objective function $Q(w)$

do
select x_i from X^l
compute the loss function $\mathcal{L}_i(w)$
gradient step $w \coloneqq w - \mu\mathcal{L}_i'(w)$
update the objective function $Q \coloneqq \lambda\mathcal{L}_i + (1-\lambda)Q$
until Q and/or w converges

$$\mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)$$

$$\frac{\partial \mathcal{L}_n(\mathbf{w}^1, \mathbf{w}^2)}{\partial \alpha_h}$$

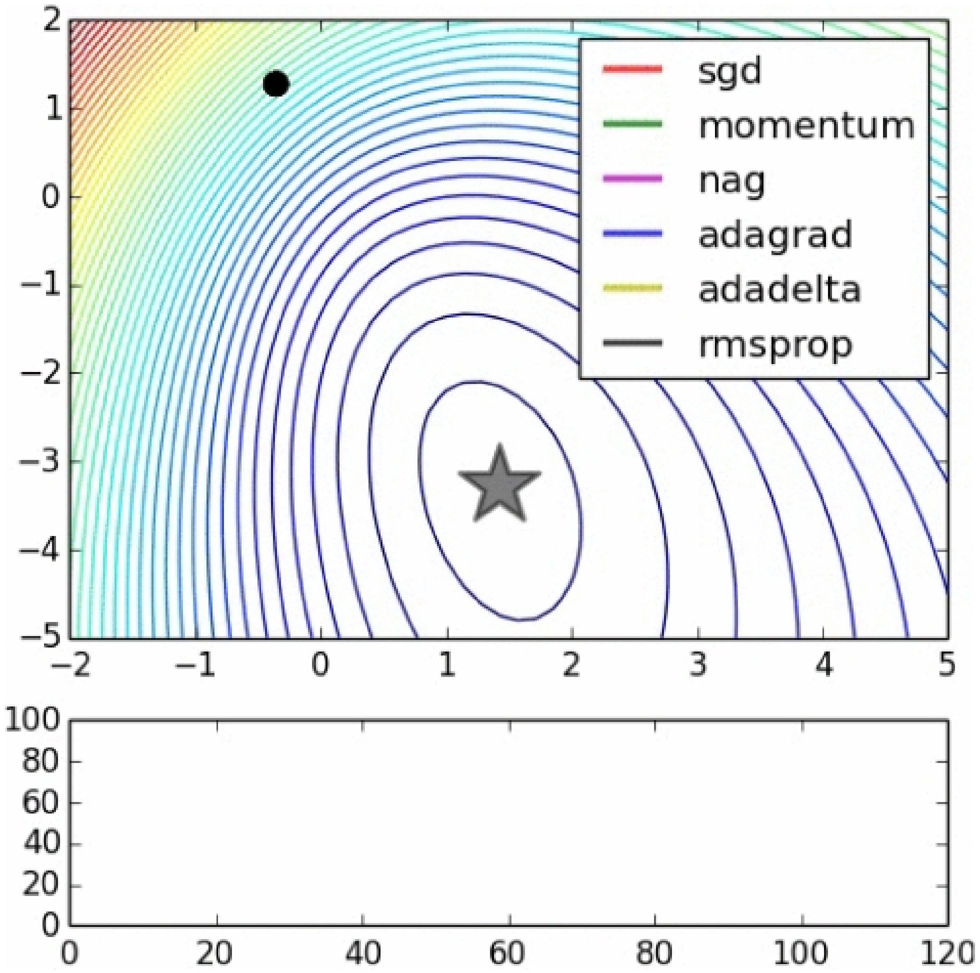
Gradient descent vs SGD

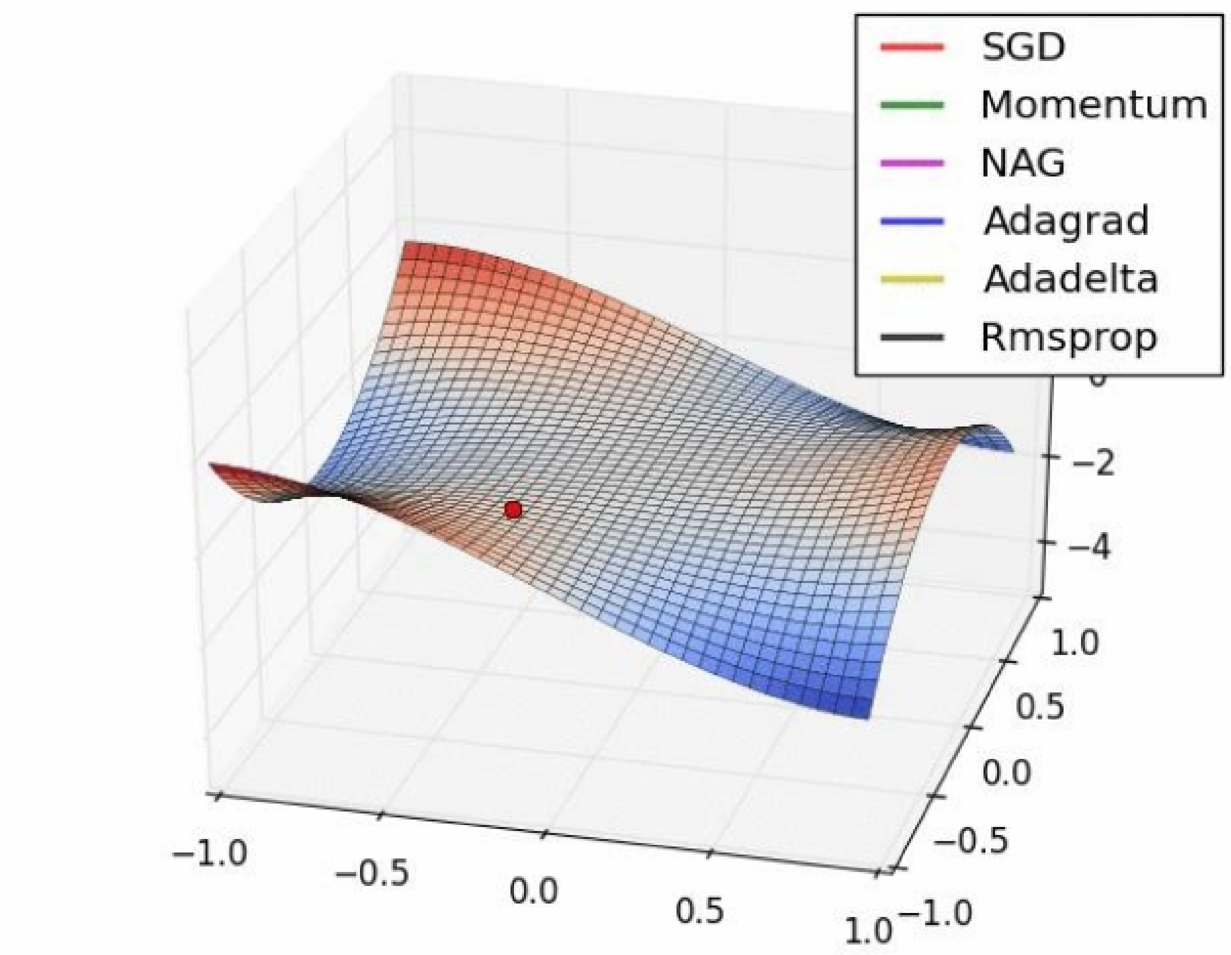
- Gradient descent
  - Too expensive to compute for large datasets
- Stochastic gradient descent
  - Too slow to converge due to noisy gradients
- Mini-batch SDG
  - Divide datasets into manageable batches and average out the stochastic behavior in gradients

Convergence improvement for GD algorithms

- Momentum
  - Updates weights as a linear combination of current gradients and previous update
- NAG (Nesterov's accelerated gradients)
  - Momentum update on parameter space and gradients
- AdaGrad (Adaptive gradients)
  - Increased for sparse parameter updates
  - Decreased for less sparse parameter updates
- RMSProp (running mean square propagation)
  - Learning rate is adjusted to each parameter guarding against rapid decay from preious updates
- Adam (Adaptive moment estimation)
  - Momentum + RMSProp

Comparison of optimization algorithms





http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Momentum vs Nesterov's momentum

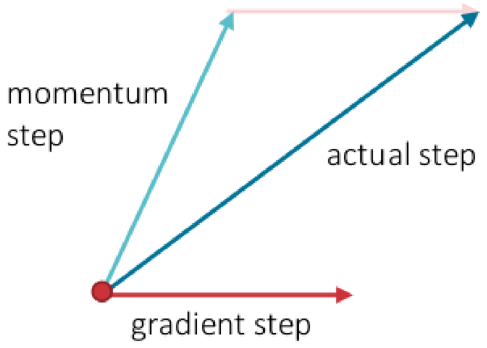
$$\text{Objective: } \min_{\theta} f(\theta)$$

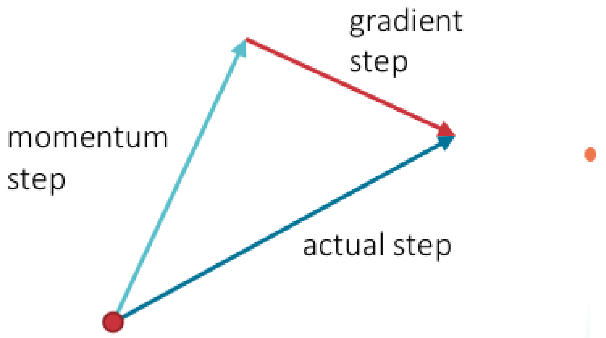
$$\text{Jump in param space: } \theta_{t+1} = \theta_t - \mu v_t$$

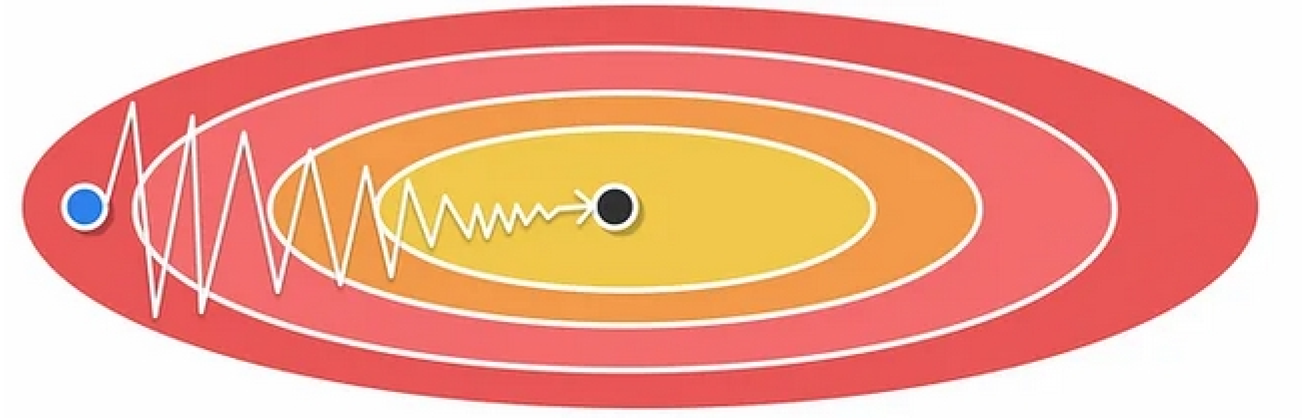
$$\text{Momentum: } v_t = v_{t-1} + \epsilon g$$

$$\text{Classical Momentum: } v_t = v_{t-1} + \epsilon \nabla f\left(\theta_{t-1}\right)$$

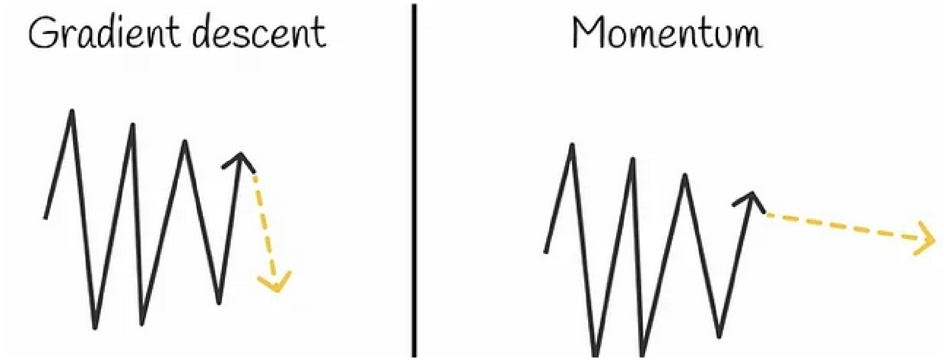
$$\text{Nesterov's Momentum: } v_t = v_{t-1} + \epsilon \nabla f\left(\theta_{t-1} - \mu v_{t-1}\right)$$







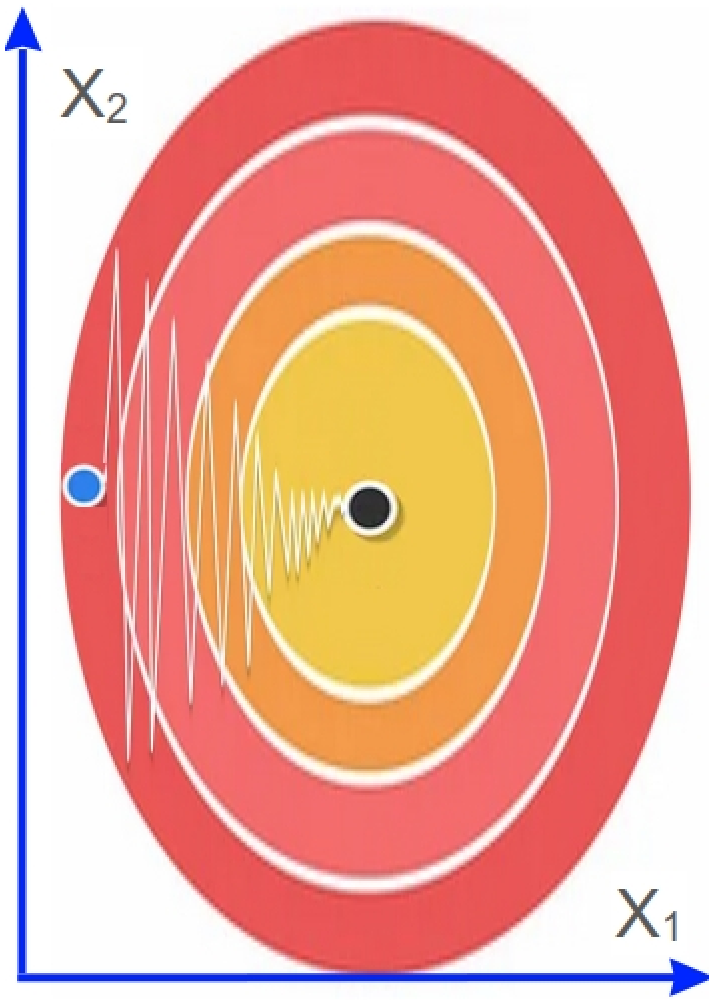
Gradient descent visualization with starting point in blue and local minima shown in black.



Descent prone to oscillations in vertical direction

Momentum capitalizing on trend of horizontal gradients towards minima and canceling out oscillations in vertical direction

Problem with momentum algorithms



- Momentum capitalizes on the trend
  - Large gradients along X_2 and small gradients along X_1 and minima is in the direction on X_1

AdaGrad update

$$v_t = v_{t-1} + g^2$$

$$\theta_{t+1} = \theta_t - \frac{\mu}{\sqrt{v_t} + \varepsilon} g$$

- Accumulates element-wise squares of gradients → g^2 from all previous iterations.

  + Adjusts to sprase updates and tapering out frequent/strong updates.

  - The learning rate constantly decays with the increase of iterations due to positive accumulation of gradients.

RMSProp update

$$v_t = \beta v_{t-1} + (1-\beta) g^2$$

$$\theta_{t+1} = \theta_t - \frac{\mu}{\sqrt{v_t} + \varepsilon} g$$

$$\beta \approx 1$$

$$\theta_{t+1} = \theta_t - \mu \operatorname{sign}(g)$$

- Instead of cumulative sum of squared gradients → g^2, store exponential moving average.

+ Experiments show that RMSProp converges faster than AdaGrad.

- Sensitive to choice of ***μ*** and magnitudes of gradients is lost.

Adam update

$$v_t = \beta_1 v_{t-1} + (1-\beta_1)g; \quad \hat{v}_t = \frac{v_t}{1-\beta_1^t}$$

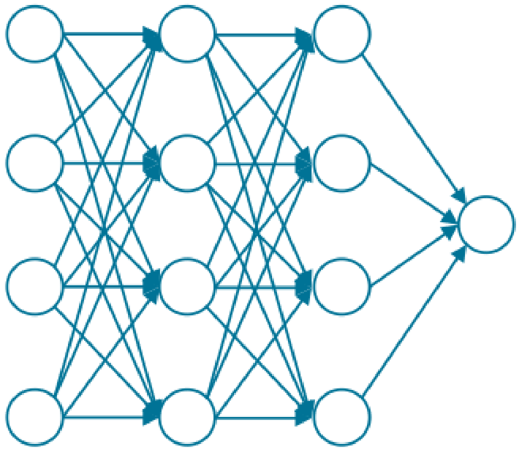
$$s_t = \beta_2 s_{t-1} + (1-\beta_2)g^2; \quad \hat{s}_t = \frac{s_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\mu \hat{v}_t}{\sqrt{\hat{s}_t} + \varepsilon} g$$

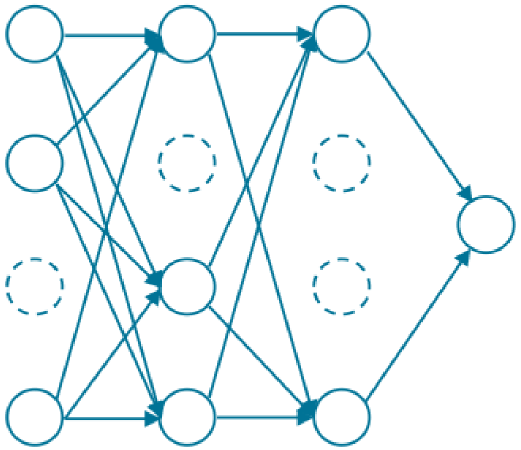
- Combines momentum and RMSProp

VI Practical issues in neural network training

- Overfitting
  - Regularization
  - Neural architecture
  - Dropout
- Vanishing and exploding gradient
  - Adaptive learning rate
  - Conjugate gradient methods
  - Batch normalization
- Local optima
  - Pretraining
- Computational challenges
  - Torch



standard neural network



neural network after dropout