Statistical Machine Learning (BE4M33SSU)
Lecture 5: Artificial Neural Networks

Jan Drchal

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Outline

Topics covered in the lecture:

- Neuron types
- Layers
- Loss functions
- Regularization
- Computing loss gradients via backpropagation
- Learning neural networks, Stochastic Gradient Descent

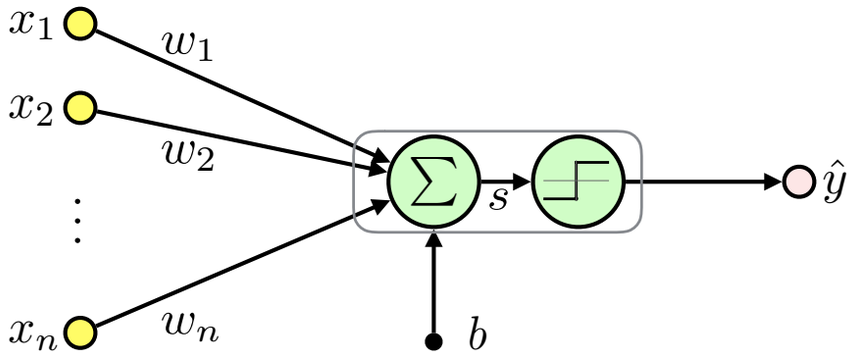
Neural Networks Overview

- Composition of simple linear or non-linear functions (neurons) parametrized by *weights* and *biases*
- Training examples: $\mathcal{T}^m = \{(x_i, y_i) \in (\mathcal{X} \times \mathcal{Y}) \mid i = 1, \ldots, m\}$, where $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^K$
- Here we consider $\mathcal{H}$ a hypothesis space of neural networks having a fixed architecture
- Learning methods are based on Empirical Risk Minimization:

$$R_{\mathcal{T}^m}(h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \ell(y_i, h_{\boldsymbol{\theta}}(x_i)),$$

where $h_{\boldsymbol{\theta}} \in \mathcal{H}$ denotes a neural network parametrized by $\boldsymbol{\theta}$

McCulloch-Pitts Perceptron

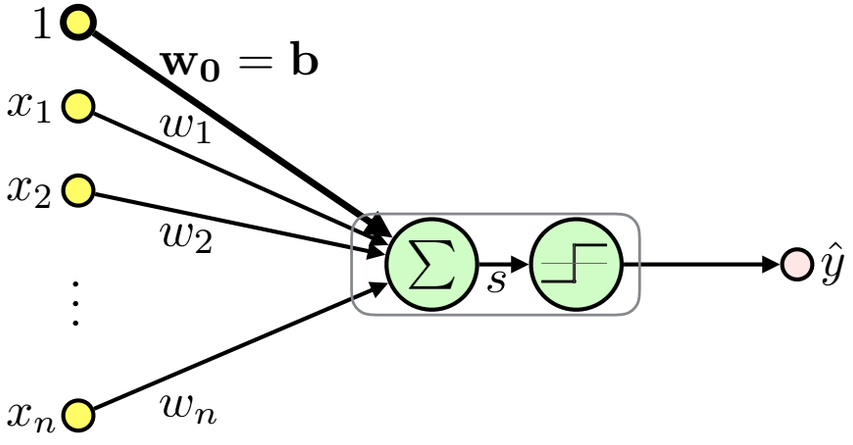


$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$	input (feature vector)
$\boldsymbol{w} = (w_1, w_2, \ldots, w_n)^T \in \mathbb{R}^n$	weights
$b \in \mathbb{R}$	bias (threshold)
$s = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \in \mathbb{R}$	inner potential
$f(s) = \begin{cases} -1 & \text{if } s < 0 \\ 1 & \text{else} \end{cases}$	activation function
$\hat{y} = h_{(\boldsymbol{w},\boldsymbol{b})}(\boldsymbol{x}) \in \{-1, 1\}$	output (activity)

$$\hat{y} = f(s) = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\right)$$

- It is the linear classifier we have already seen.

McCulloch-Pitts Perceptron: Treating Bias

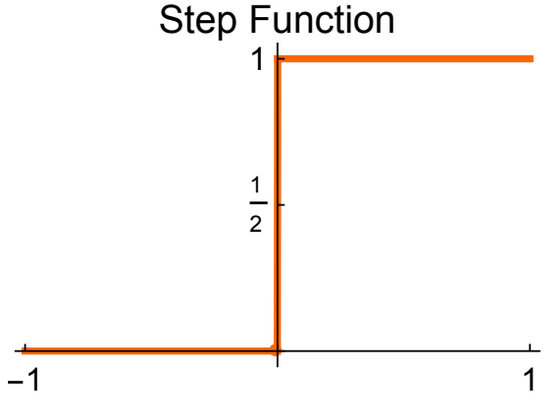


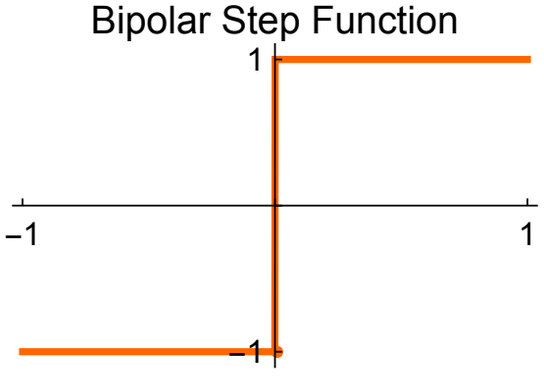
- Treat bias as an extra fixed input $x_0 = 1$ weighted $w_0 = b$:

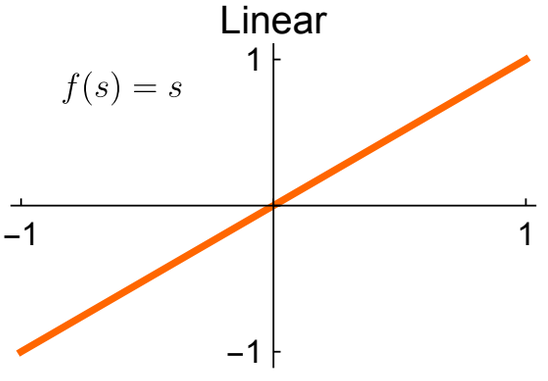
$$\hat{y} = f\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\right) = f\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + w_0 \cdot 1\right) = f\left(\langle \boldsymbol{w'}, \boldsymbol{x'} \rangle\right)$$

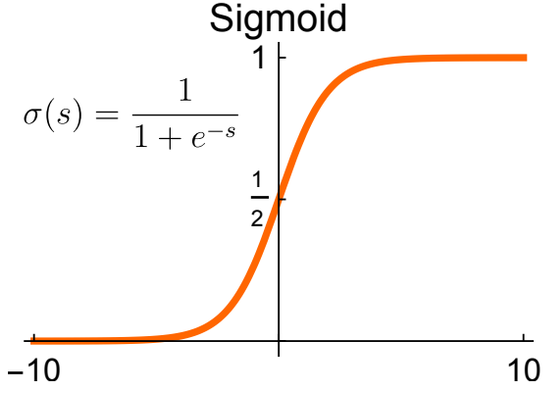
- $\boldsymbol{x'} = \left(1, x_1, \ldots, x_n\right)^T \in \mathbb{R}^{n+1}$
- $\boldsymbol{w'} = \left(w_0, w_1, \ldots, w_n\right)^T \in \mathbb{R}^{n+1}$
- Unless otherwise noted we will use $\boldsymbol{x}$, $\boldsymbol{w}$ instead of $\boldsymbol{x'}$, $\boldsymbol{w'}$

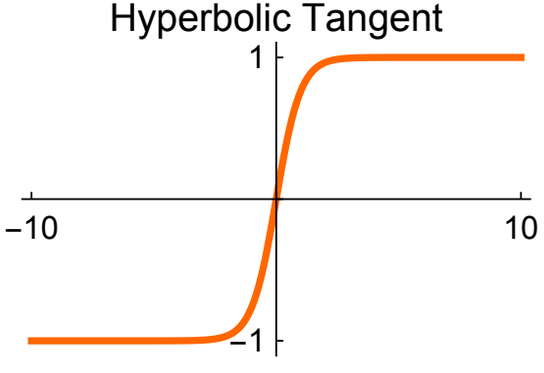
Activation Functions

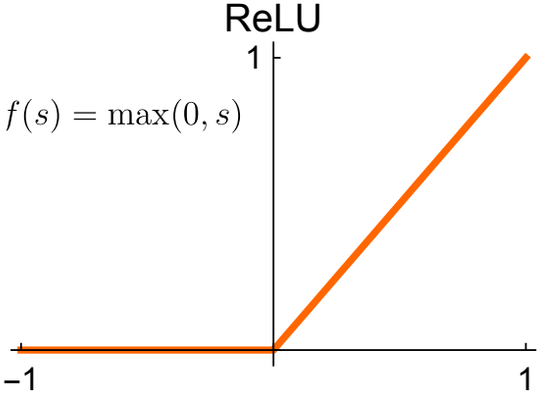












- Logistic sigmoid: $\sigma(s) \triangleq \dfrac{1}{1+e^{-s}} = \dfrac{e^s}{e^s+1}$
- Note: $\tanh(s) = \dfrac{e^s - e^{-s}}{e^s + e^{-s}} = 2\,\sigma(s) - 1$

Linear Neuron

- Training examples: $\mathcal{T}^m = \{(\boldsymbol{x}_i, y_i) \in (\mathbb{R}^{n+1} \times \mathbb{R}) \mid i = 1, \ldots, m\}$
- Single neuron with linear activation function $\equiv$ **linear regression**:

$$\hat{y} = s = \langle \boldsymbol{x}, \boldsymbol{w} \rangle, \quad \hat{y} \in \mathbb{R}$$

- Inputs: $\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1n} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \ldots & x_{mn} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_m^T \end{pmatrix}$
- Targets: $\boldsymbol{y} = \big(y_1, \ldots, y_m\big)^T, \quad y_i \in \mathbb{R}$
- Outputs: $\hat{\boldsymbol{y}} = \big(\hat{y}_1, \ldots, \hat{y}_m\big)^T, \quad \hat{y}_i \in \mathbb{R}$
- For the whole dataset we get:

$$\hat{\boldsymbol{y}} = \mathbf{X}\boldsymbol{w}, \quad \hat{\boldsymbol{y}} \in \mathbb{R}^m$$

Linear Neuron: Maximum Likelihood Estimation

- Assumption: data are Gaussian distributed with mean $\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle$ and variance σ^2:

$$y_i \sim \mathcal{N}\left(\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle, \sigma^2\right) = \langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + \mathcal{N}\left(0, \sigma^2\right)$$

- Likelihood for i.i.d. data:

$$p\left(\boldsymbol{y}|\boldsymbol{w}, \mathbf{X}, \sigma\right) = \prod_{i=1}^{m} p\left(y_i|\boldsymbol{w}, \boldsymbol{x_i}, \sigma\right) = \prod_{i=1}^{m} \left(2\pi\sigma^2\right)^{-\frac{1}{2}} e^{-\frac{1}{2\sigma^2}\left(y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle\right)^2} =$$

$$= \left(2\pi\sigma^2\right)^{-\frac{m}{2}} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{m}\left(y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle\right)^2} =$$

$$= \left(2\pi\sigma^2\right)^{-\frac{m}{2}} e^{-\frac{1}{2\sigma^2}\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)^T\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)}$$

- Negative Log Likelihood (switching to minimization):

$$\mathcal{L}\left(\boldsymbol{w}\right) = \frac{m}{2}\log\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)^T\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)$$

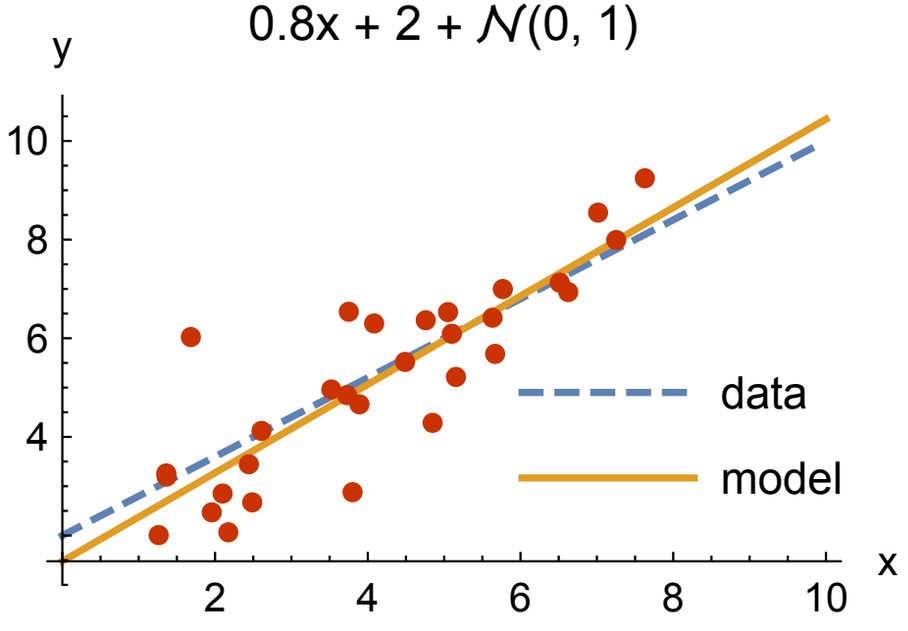
Linear Neuron: Maximum Likelihood Estimation (contd.)

- Note that

$$\sum_{i=1}^{m} \underbrace{\left(y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle\right)^2}_{\ell(y_i, \hat{y}_i)} = \left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)^T \left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)$$

  is the **sum-of-squares** or **squared error** (SE)

- Minimization of $\mathcal{L}(\boldsymbol{w}) \equiv$ least squares estimaton
- Solving $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0$ we get $\boldsymbol{w}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\boldsymbol{y}$ (see seminar)



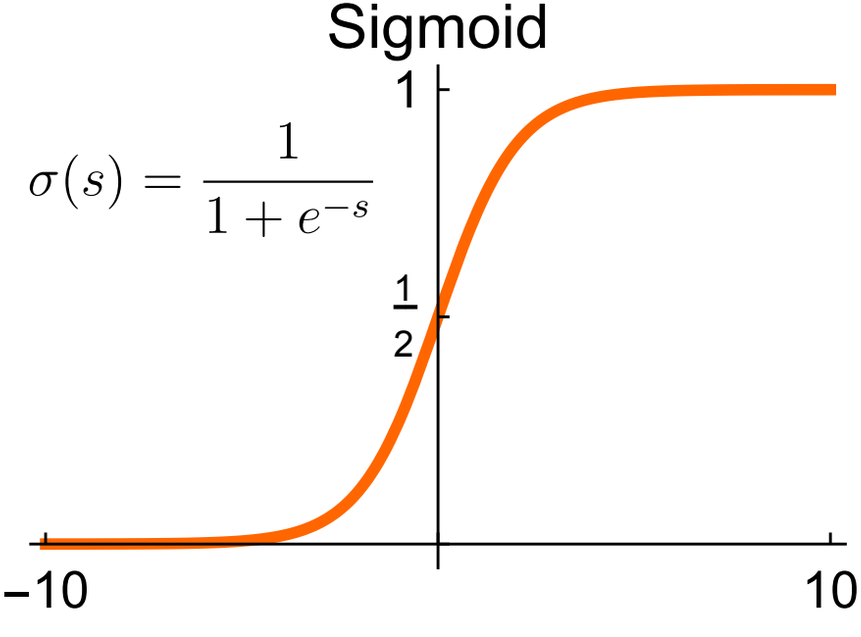
Logistic Sigmoid and Probability

- Denote: $\hat{y} = \sigma(s),\ \hat{y} \in (0, 1)$
- Sigmoid output can represent a parameter of the Bernoulli distribution:

$$p(y|\hat{y}) = \mathrm{Ber}(y|\hat{y}) = \hat{y}^y \left(1 - \hat{y}\right)^{1-y} = \begin{cases} \hat{y} & \text{for } y = 1 \\ 1 - \hat{y} & \text{for } y = 0 \end{cases}$$

- Models confidence of the positive class $y = 1$
- Motivation: log-odds linear model
  $\Rightarrow$ see AE4B33RPZ
- Binary classifier:

$$h(\hat{y}) = \begin{cases} 1 & \text{if } \hat{y} > \frac{1}{2} \\ 0 & \text{else} \end{cases}$$



Logistic Regression

- MCP neuron using sigmoid activation function $\equiv$ **logistic regression**:

$$\hat{y} = \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle),\ \hat{y} \in (0, 1)$$

- Inputs: $\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1n} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \dots & x_{mn} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_m^T \end{pmatrix}$
- Target class: $\boldsymbol{y} = \big(y_1, \dots, y_m\big)^T,\ y_i \in \{0, 1\}$
- Output class: $\hat{\boldsymbol{y}} = \big(\hat{y}_1, \dots, \hat{y}_m\big)^T,\ \hat{y}_i \in (0, 1)$
- Note that the logistic regression solves actually a classification task

Logistic Regression MLE Leads to the Cross-Entropy

- Likelihood, for the logistic regression:

$$p(\boldsymbol{y}|\boldsymbol{w}, \mathbf{X}) = \prod_{i=1}^{m} \mathrm{Ber}(y_i|\hat{y}_i) = \prod_{i=1}^{m} \hat{y}_i^{y_i} \left(1 - \hat{y}_i\right)^{1-y_i}$$

- Negative Log Likelihood:

$$\mathcal{L}(\boldsymbol{w}) = \sum_{i=1}^{m} \underbrace{-\left[y_i \log \hat{y}_i + (1 - y_i) \log \left(1 - \hat{y}_i\right)\right]}_{\ell(y_i, \hat{y}_i)}$$

- This *loss function* is called the **cross-entropy**
- The $\ell(y_i, \hat{y}_i)$ is the negative log probability of the correct answer $y_i \in \{0, 1\}$ given by the model output $\hat{y}_i \in (0, 1)$

Maximum Likelihood Estimation

- Maximum Likelihood Estimation: $w^* = \underset{w}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{w})$
- Derivative of the loss w.r.t. to the sigmoid argument:

$$\frac{\partial \mathcal{L}}{\partial s_i} = \hat{y}_i - y_i \quad \text{(see seminar)}$$

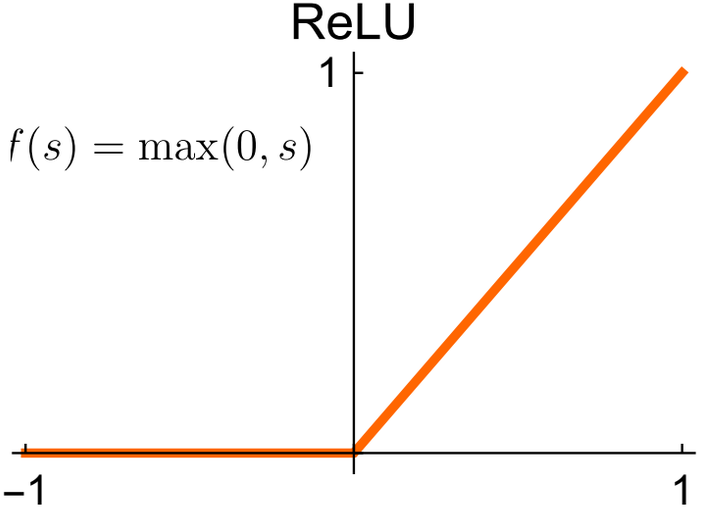
- Gradient w.r.t. logistic regression parameters:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial s_i} \cdot \frac{\partial s_i}{\partial \boldsymbol{w}} = \sum_{i=1}^{m} \boldsymbol{x}_i(\hat{y}_i - y_i) = \mathbf{X}^T(\hat{\boldsymbol{y}} - \boldsymbol{y})$$

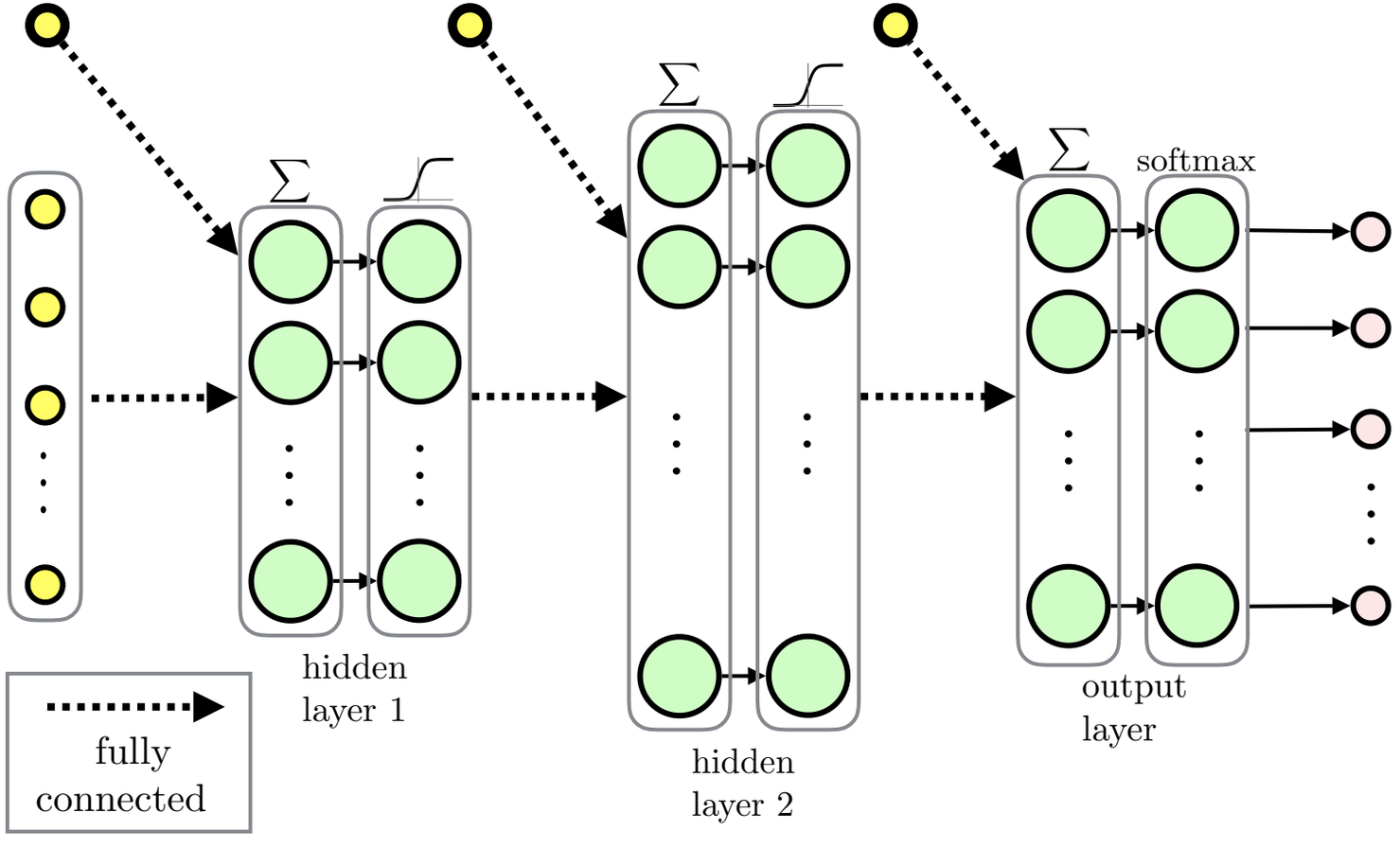
- $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \mathbf{0}$ has no analytical solution $\implies$ use numerical methods

Rectified Linear Unit (ReLU)

- Definition $f(s) = \max(0, s)$
- Fast to compute
- Helps with *vanishing gradients* problem: the gradient is constant for $s > 0$, while for sigmoid-like activations it becomes increasingly small
- Leads to sparse representations: $s < 0$ turns the neuron completely off
- Might block gradient propagation $\rightarrow$ dead units $\rightarrow$ Leaky ReLU
- Unbounded: use regularization to prevent numerical problems

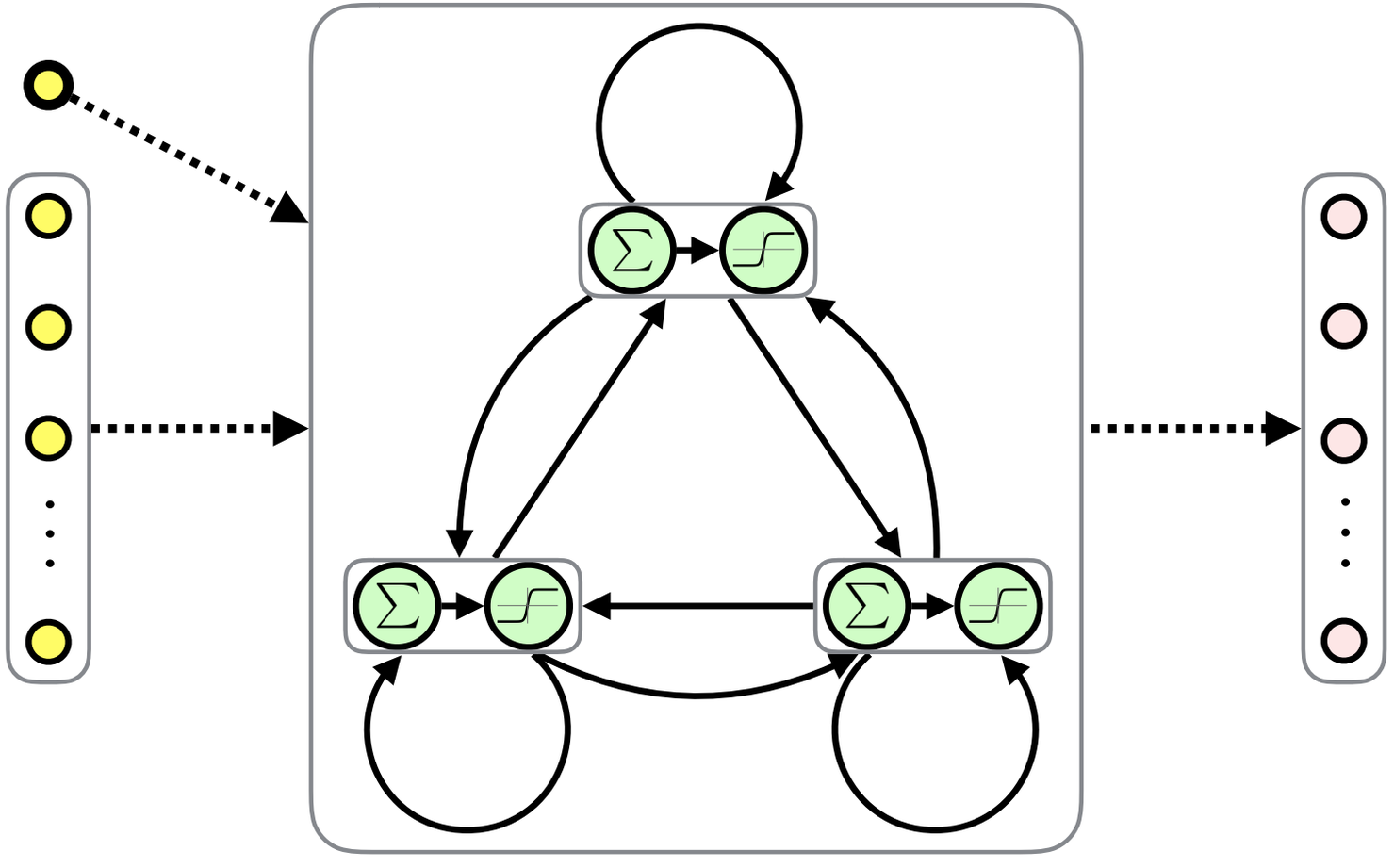


Multilayer Perceptron (MLP)



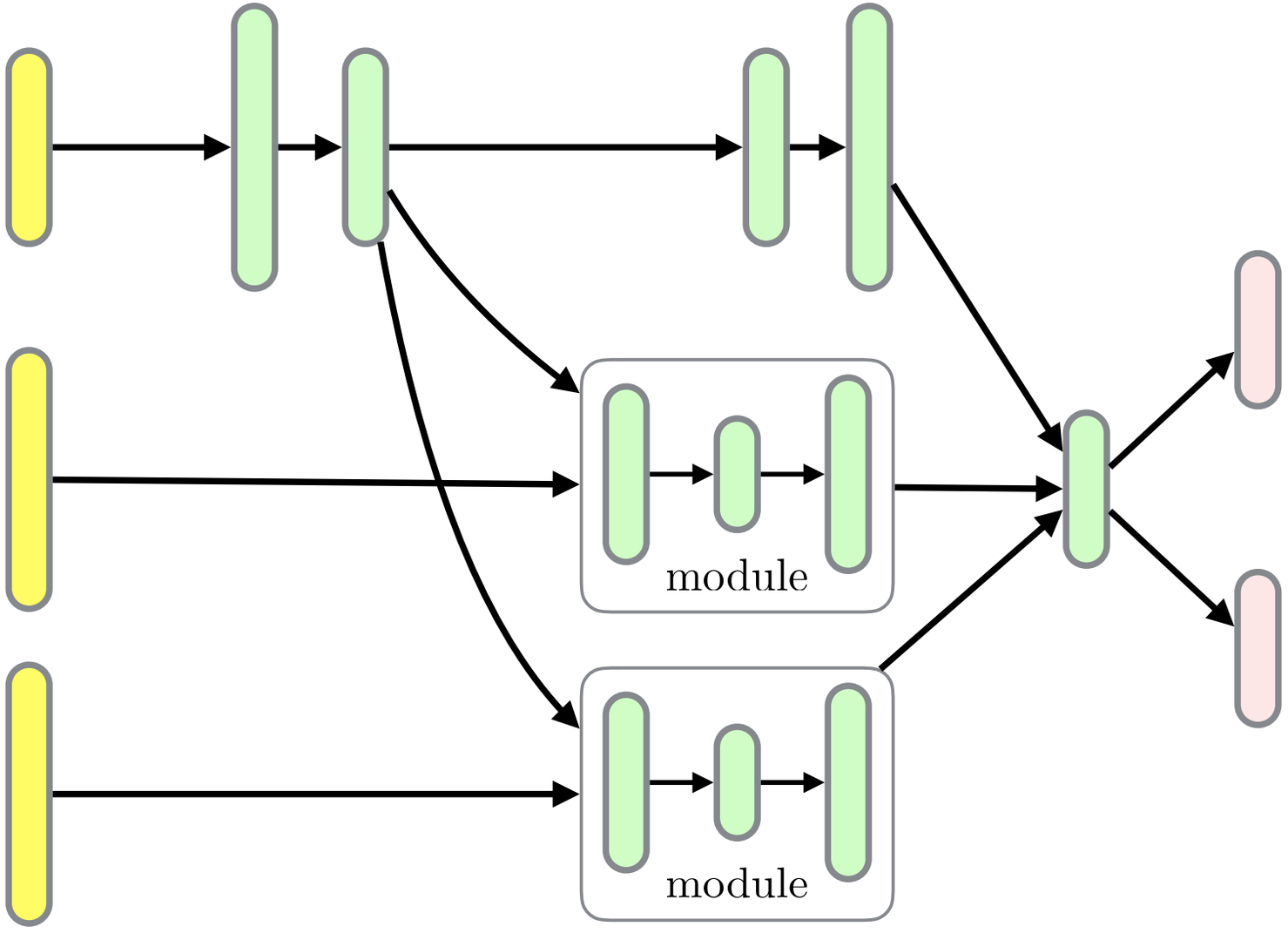
- Feed-forward ANN
- Fully-connected layers
- MLP for regression would typically use linear output layer

Recurrent Neural Network (RNN)



- Fully-Connected Recurrent Neural Network (FRNN)
- Both inputs and outputs are sequences
- Feedback connections $\rightarrow$ memory (similarly to sequential circuitry)

Modular and Hierarchical Architectures

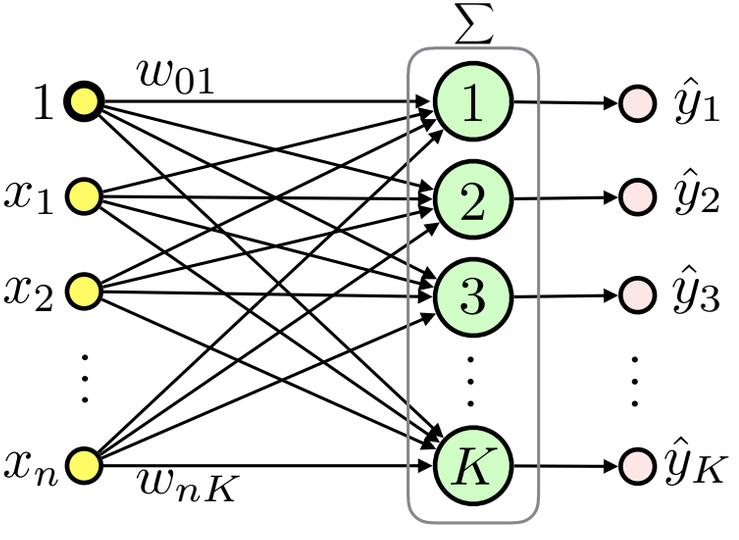


- Layers can be organized in *modules*
- Hierarchies of modules
- Module reuse

Linear (Dense) Layer

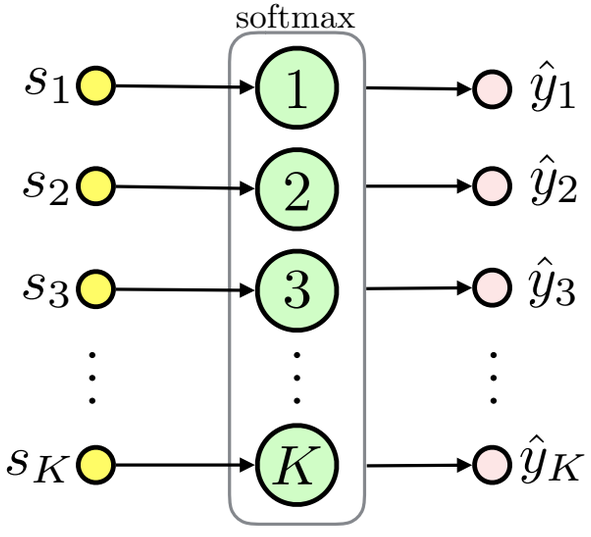
- Output k: $\hat{y}_k = \langle \boldsymbol{x}, \boldsymbol{w}_k \rangle$, $k = 0, 1, \ldots, K$
- All outputs using *weight matrix* $\mathbf{W}$: $\hat{\boldsymbol{y}} = \boldsymbol{x}^T \mathbf{W}$
- Multiple samples: $\hat{\mathbf{Y}} = \mathbf{X}\mathbf{W}$

$$\mathbf{W} = (\boldsymbol{w}_1 \ldots \boldsymbol{w}_K) = \begin{pmatrix} w_{01} & \ldots & w_{0K} \\ \vdots & \ddots & \vdots \\ w_{n1} & \ldots & w_{nK} \end{pmatrix}$$



$$\mathbf{X} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_m^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1n} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \ldots & x_{mn} \end{pmatrix} \qquad \hat{\mathbf{Y}} = \begin{pmatrix} \hat{\boldsymbol{y}}_1^T \\ \vdots \\ \boldsymbol{y}_m^T \end{pmatrix} = \begin{pmatrix} \hat{y}_{11} & \ldots & \hat{y}_{1K} \\ \vdots & \ddots & \vdots \\ \hat{y}_{m1} & \ldots & \hat{y}_{mK} \end{pmatrix}$$

Softmax Layer



- Multinominal classification, K mutually exclusive classes
- Definition: $\sigma_k(\boldsymbol{s}) \triangleq \frac{e^{s_k}}{\sum_{c=1}^{K} e^{s_c}}$, where K is the number of classes
- Softmax represents a categorical probability distribution: $\sigma_k \in (0, 1)$ for $k \in \{1 \dots K\}$ and $\sum_{c=1}^{K} \sigma_c = 1$
- Describes class membership probabilities: $p(y = k|\boldsymbol{s}) = \sigma_k(\boldsymbol{s})$

Softmax Layer MLE

- Target: $\boldsymbol{y} = \begin{pmatrix} y_1 & \dots & y_m \end{pmatrix}^T$, $y_i \in \{1, 2, \dots, K\}$
- One-hot encoding for sample i and class k: let $y_{ik} = [\![y_i = k]\!]$
- Likelihood:

$$p(\boldsymbol{y}|\boldsymbol{w}, \mathbf{X}) = \prod_{i=1}^{m} \prod_{c=1}^{K} \hat{y}_{ic}^{y_{ic}}$$

- Negative Log Likelihood:

$$\mathcal{L}(\boldsymbol{w}) = -\sum_{i=1}^{m} \sum_{c=1}^{K} y_{ic} \log(\hat{y}_{ic})$$

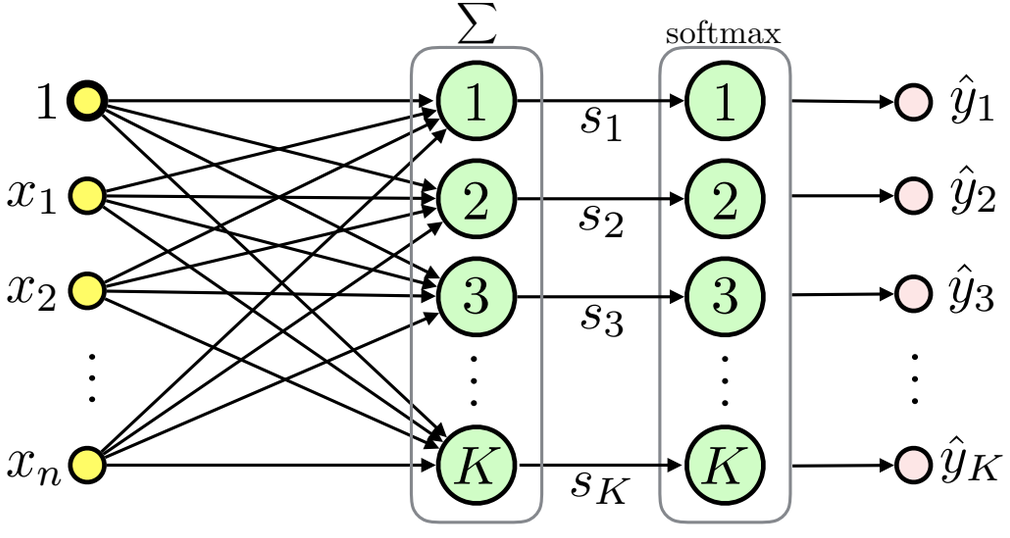
  Again the **cross-entropy**

- See seminar for the gradient

Multinominal Logistic Regression

- linear layer + softmax layer = **multinominal logistic regression**:

$$\hat{y}_k = \sigma_k(\boldsymbol{x}^T \mathbf{W})$$



- Classifier: $h(\boldsymbol{x}, \mathbf{W}) = \underset{k}{\operatorname{argmax}}\, \hat{y}_k$

Loss Functions: Summary

problem	**suggested loss function**
binary classification	cross-entropy $-\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log \hat{y}_i + (1-y_i)\log\left(1-\hat{y}_i\right)\right]$
multinominal classification	multinominal cross-entropy $-\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{K} y_{ic}\log(\hat{y}_{ic})$
regression	mean squared error $\frac{1}{m}\sum_{i=1}^{m}\left(y_i-\hat{y}_i\right)^2$
multi-output regression	mean squared error $\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{K}\left(y_{ic}-\hat{y}_{ic}\right)^2$

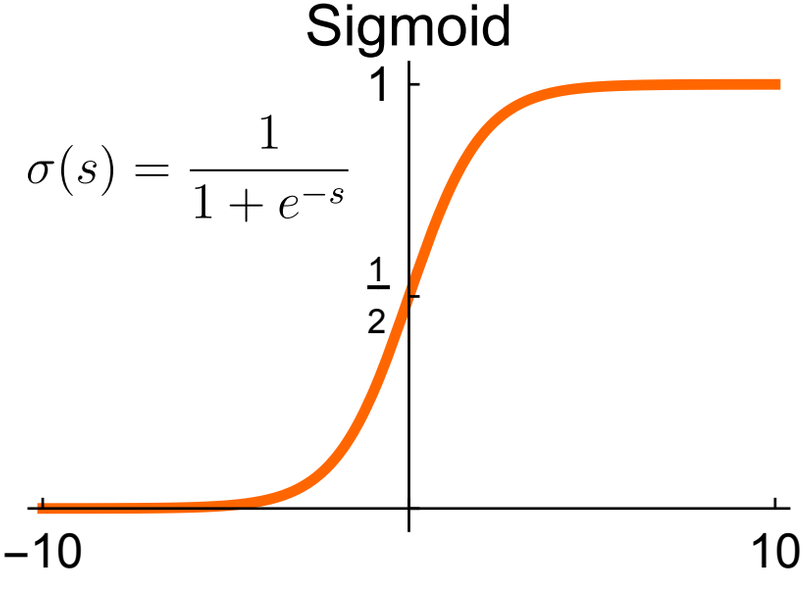
- These exactly correspond to the empirical risk $R_{\mathcal{T}^m}(h)$

Regularization

- How to deal with overfitting?
  - get more data
  - find simpler model, search for optimal architecture, e.g., number, type and size of layers
  - constrain model by *regularization*
- Most types of regularization are based on constraining the parameter space
- Bayesian point of view: introduce prior distribution on model parameters

L2 Regularization (Weight Decay): Motivation

- Limit hypothesis space by limiting the size of the weight vector
- You already know this from SVMs!
- *Intuition:* sigmoid-like neurons kept near zero potential (via small weights) behave similarly to linear neurons
- L2 regularization (weight decay): zero mean Gaussian prior



Example: L2 Regularization for Linear Regression

- Recall the linear regression likelihood:

$$p\left(\boldsymbol{y}|\boldsymbol{w},\mathbf{X}\right)=\left(2\pi\sigma^{2}\right)^{-\frac{m}{2}}e^{-\frac{1}{2\sigma^{2}}(\boldsymbol{y}-\mathbf{X}\boldsymbol{w})^{T}(\boldsymbol{y}-\mathbf{X}\boldsymbol{w})}$$

- Define a Gaussian prior with zero mean and variance σ_0^2 for the parameters:

$$p\left(\boldsymbol{w}\right)=\left(2\pi\sigma_{0}^{2}\right)^{-\frac{1}{2}}e^{-\frac{1}{2\sigma_{0}^{2}}\boldsymbol{w}^{T}\boldsymbol{w}}$$

- Then the posterior is:

$$p(\boldsymbol{w}|\boldsymbol{y},\mathbf{X})=\frac{p(\boldsymbol{y}|\boldsymbol{w},\mathbf{X})\cdot p(\boldsymbol{w})}{p(\boldsymbol{y}|\mathbf{X})}$$

The denominator does not depend on the parameters $\boldsymbol{w}$:

$$p(\boldsymbol{w}|\boldsymbol{y},\mathbf{X})\propto p(\boldsymbol{y}|\boldsymbol{w},\mathbf{X})\cdot p(\boldsymbol{w})$$

MAP Estimate

- Maximizing $p(\boldsymbol{w}|\boldsymbol{y}, \mathbf{X})$ gives us the Maximum a posteriori (MAP) estimate:

$$\boldsymbol{w}_{MAP} = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, p(\boldsymbol{w}|\boldsymbol{y}, \mathbf{X}) = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, (-\log p(\boldsymbol{w}|\boldsymbol{y}, \mathbf{X}))$$

where

$$-\log p(\boldsymbol{w}|\boldsymbol{y}, \mathbf{X}) = \frac{1}{2\sigma^2} (\boldsymbol{y} - \mathbf{X}\boldsymbol{w})^T (\boldsymbol{y} - \mathbf{X}\boldsymbol{w}) + \frac{1}{2\sigma_0^2} \boldsymbol{w}^T \boldsymbol{w} + C$$

- We can omit C, define $\lambda = \dfrac{\sigma^2}{\sigma_0^2}$ and minimize the loss function:

$$\mathcal{L}(\boldsymbol{w}) = (\boldsymbol{y} - \mathbf{X}\boldsymbol{w})^T (\boldsymbol{y} - \mathbf{X}\boldsymbol{w}) + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

- The term $\lambda \boldsymbol{w}^T \boldsymbol{w} = \lambda \|\boldsymbol{w}\|_2^2$ minimizes the size of the weight vector
- Note that we omit bias in $\lambda \boldsymbol{w}^T \boldsymbol{w}$

L2 Regularization Improves Numerical Stability

- Recall the solution for the linear regression $\boldsymbol{w}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\boldsymbol{y}$
- What if $\mathbf{X}^T\mathbf{X}$ has no inverse?
- We can modify the solution by adding a small element to the diagonal:

$$\boldsymbol{w}^* = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\boldsymbol{y}, \quad \lambda > 0$$

- It turns out that the solution is the minimizer of our *regularized* loss function:

$$\mathcal{L}\left(\boldsymbol{w}\right) = \left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)^T\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right) + \lambda\boldsymbol{w}^T\boldsymbol{w},$$

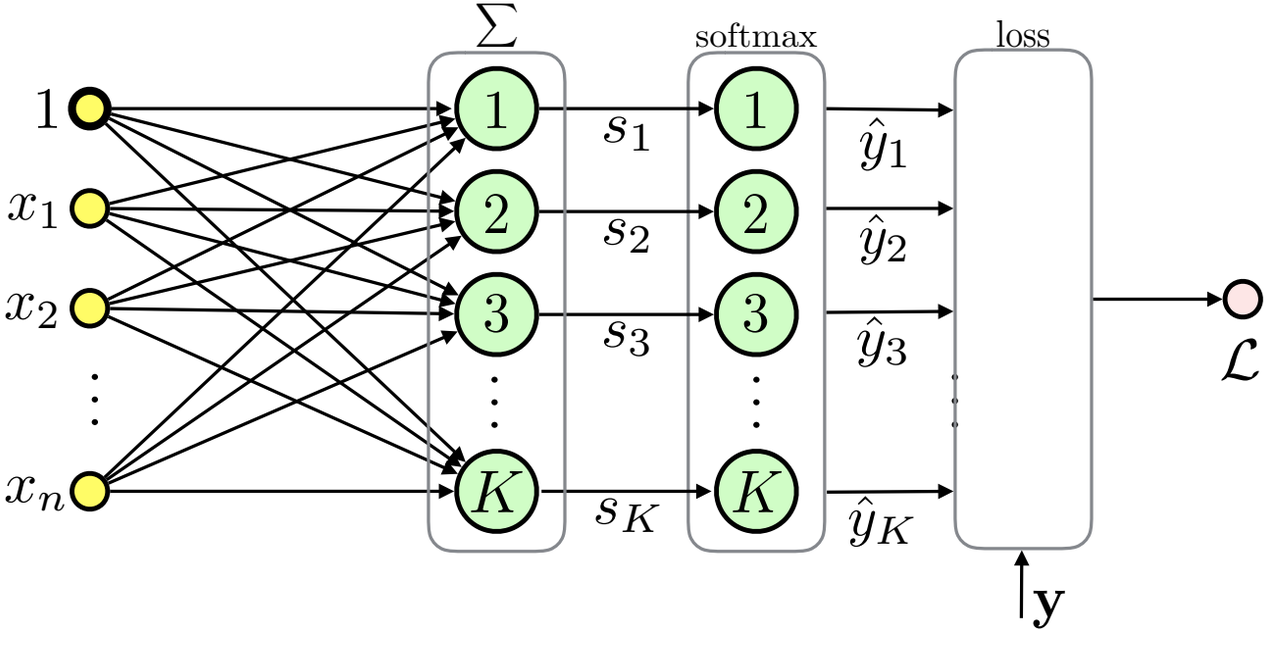
see seminar for the derivation

Other Regularization Related Approaches

- L1 regularization: sum absolute values, i.e., use $\lambda \|\boldsymbol{w}\|_1$
- Randomize inputs: same as the weight decay for linear neurons
- Dataset augmentation
- Early stopping: start with small weights, stop when validation loss starts to grow, often used for limited time-budget (see next lecture on initialization)
- Weight sharing and sparse connectivity: Convolutional Neural Networks (next lecture)
- Model averaging (see lectures on Ensembmling)
- Dropout and DropConnect

Backpropagation Overview

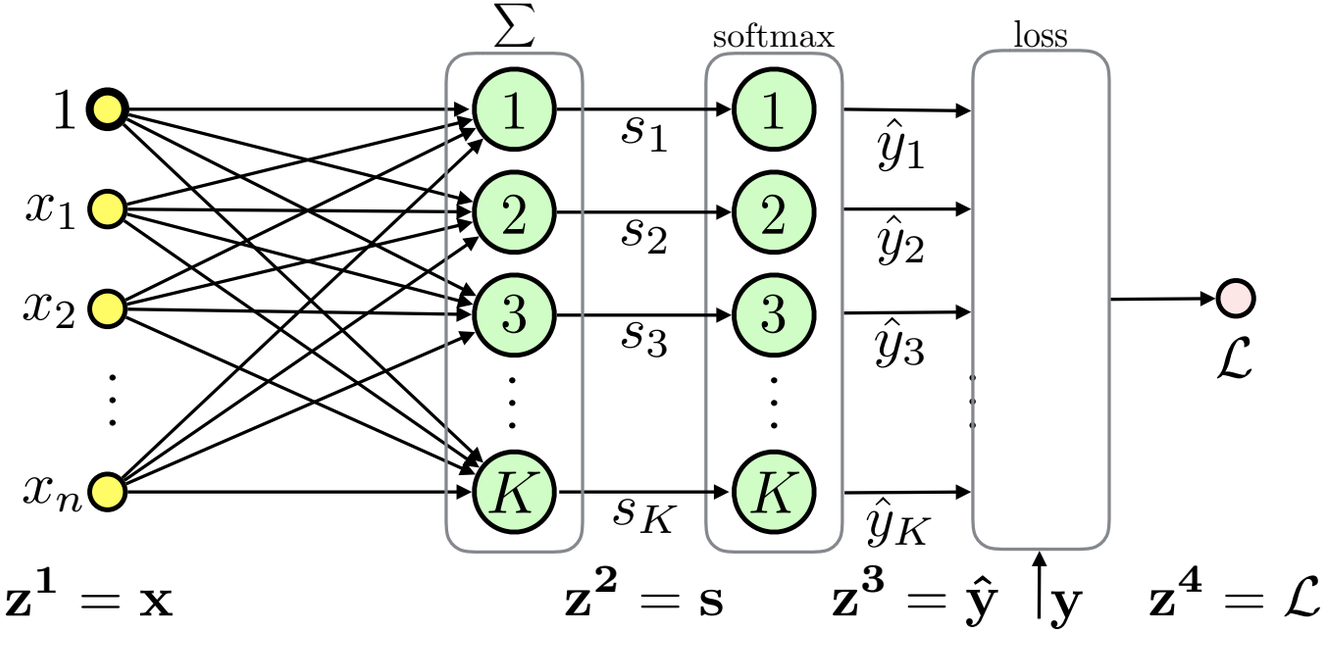
- A method to compute a gradient of the *loss function* with respect to its parameters: $\nabla \mathcal{L}(\boldsymbol{w})$
- $\nabla \mathcal{L}(\boldsymbol{w})$ is in turn used by optimization methods like gradient descent
- Here, we present the "modular" backpropagation (see Nando de Freitas' Machine Learning course: https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/)
- Let us use multinominal logistic regression as an example



Backpropagation: the Loss Function

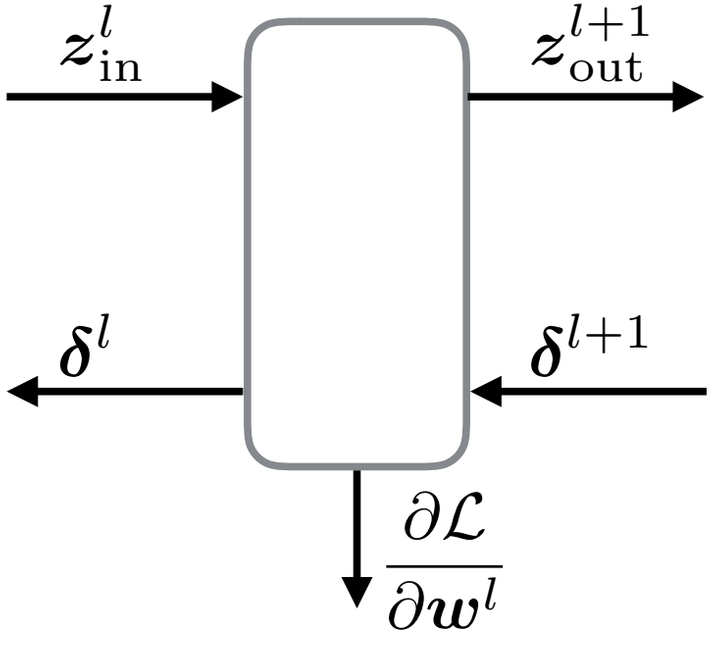
- The loss function is the multinominal cross-entropy in this case:

$$\mathcal{L}(\boldsymbol{w}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{K}[\![y_i = c]\!]\log\left(\frac{\exp\left(\langle \boldsymbol{x}_i, \boldsymbol{w}_c\rangle\right)}{\sum_{k=1}^{K}\exp\left(\langle \boldsymbol{x}_i, \boldsymbol{w}_k\rangle\right)}\right)$$



Backpropagation Based on Modules

- Computation of $\nabla \mathcal{L}(\boldsymbol{w})$ involves repetitive use of the *chain rule*
- We can make things simpler by divide and conquer approach
- Divide to simplest possible modules (these can be later combined into complex networks)
- Represent even the loss function as a module
- Passing messages

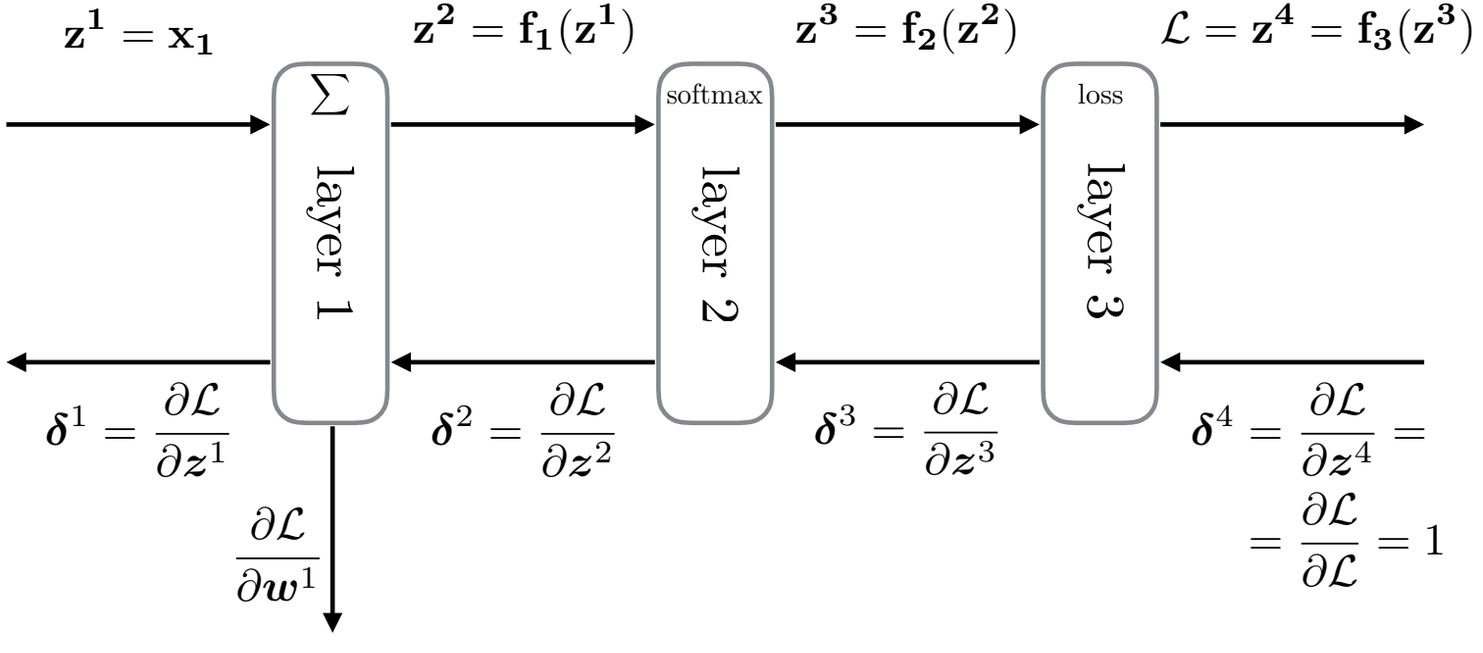


Backpropagation: Backward Pass Message

- Let $\boldsymbol{\delta}^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l}$ be the sensitivity of the loss to the module input for layer l, then:

$$\delta_i^l = \frac{\partial \mathcal{L}}{\partial z_i^l} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j^{l+1}} \cdot \frac{\partial z_j^{l+1}}{\partial z_i^l} = \sum_j \delta_j^{l+1} \frac{\partial z_j^{l+1}}{\partial z_i^l}$$

- We need to know how to compute derivatives of outputs w.r.t. inputs only!

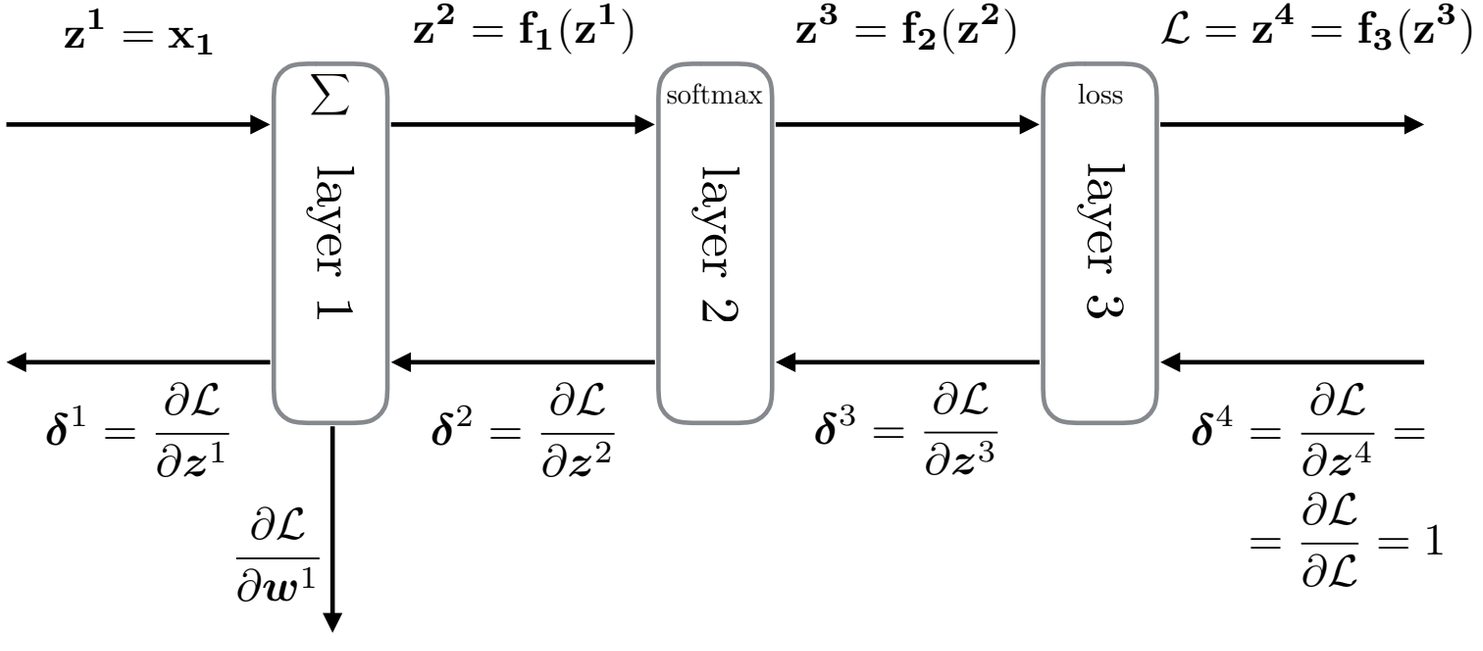


Backpropagation: Parameters

- Similarly if the module has parameters we want to know how the loss changes w.r.t. them:

$$\frac{\partial \mathcal{L}}{\partial w_i^l} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j^{l+1}} \cdot \frac{\partial z_j^{l+1}}{\partial w_i^l} = \sum_j \delta_j^{l+1} \frac{\partial z_j^{l+1}}{\partial w_i^l}$$

- Derivatives of module outputs w.r.t. to the parameters are all we need



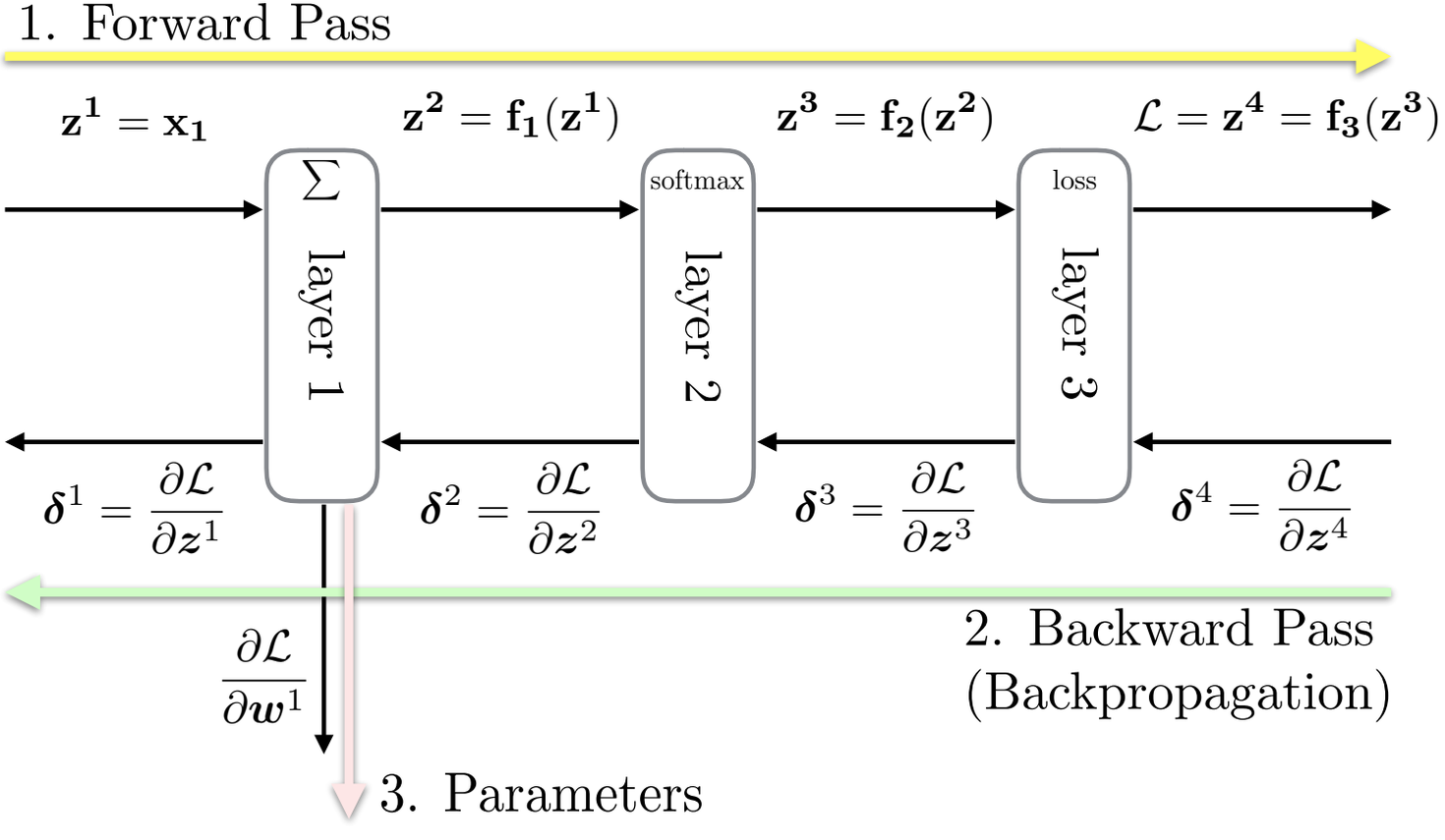
Backpropagation: Steps

- So for each module we need only to specify these three messages:

  forward: $z^{l+1} = f(z^l)$

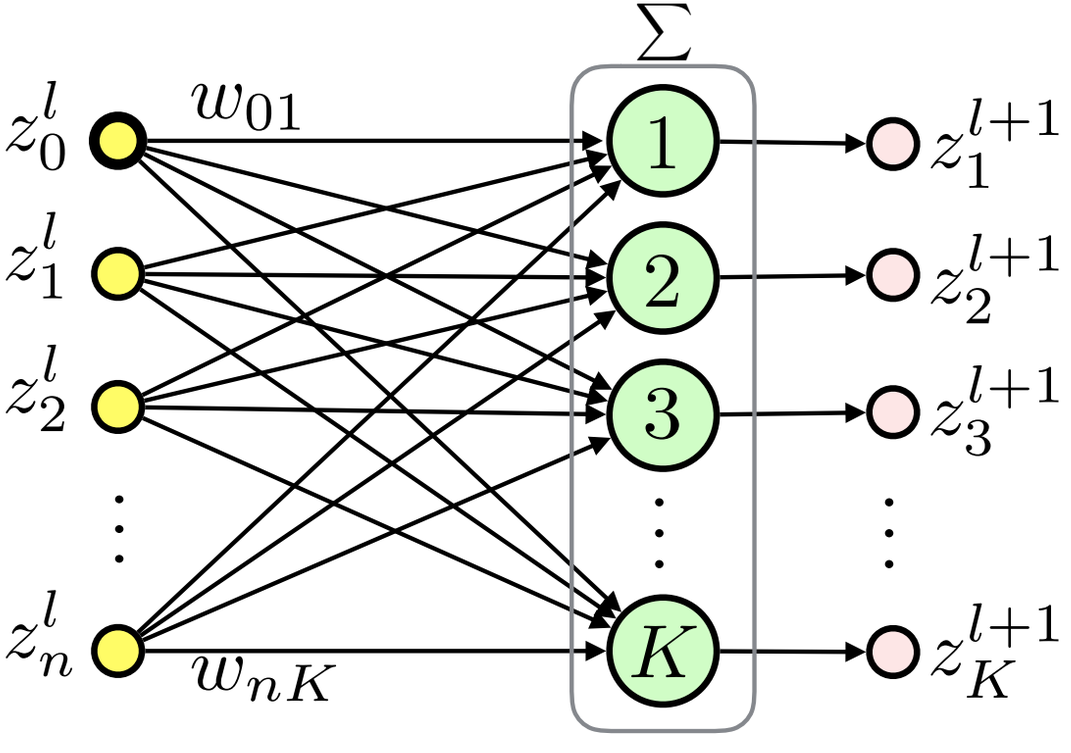
  backward: $\frac{\partial z^{l+1}}{\partial z^l}$

  parameter (optional): $\frac{\partial z^{l+1}}{\partial w^l}$



Example: Linear Layer

- **forward**: $z_j^{l+1} = \sum_{i=0}^{n} w_{ij} z_i^l, \quad j = 1, \ldots, K$
- **backward**: $\frac{\partial z_j^{l+1}}{\partial z_i^l} = w_{ij}, \quad i = 0, \ldots, n, \quad j = 1, \ldots, K$
- **parameter**: $\frac{\partial z_j^{l+1}}{\partial w_{ik}} = [\![j = k]\!] z_i^l$



Example: Mean Squared Error

- **forward**: $z^{l+1} = \frac{1}{m} \sum_{i=1}^{m} \left(y_i - z_i^l\right)^2$
- **backward**: $\frac{\partial z^{l+1}}{\partial z_i^l} = -\frac{2}{m}(y_i - z_i^l), \quad i \in \{1, \ldots, n\}$

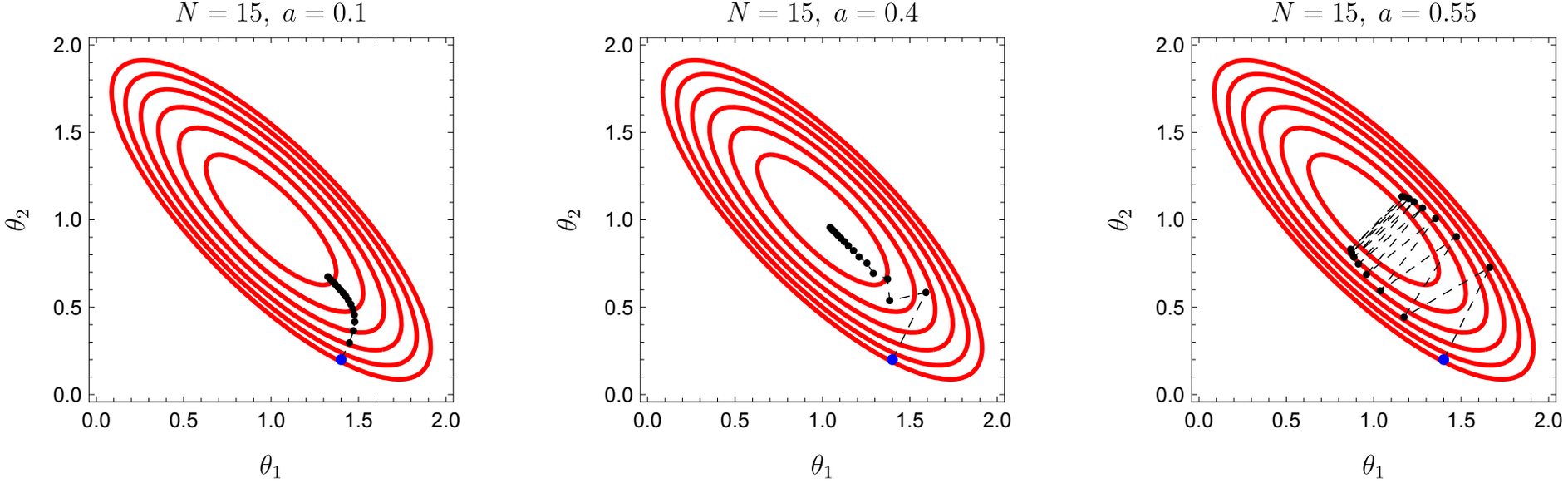
Gradient Descent (GD)

- **Task**: find parameters which minimize loss over the training dataset:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta})$$

where $\boldsymbol{\theta}$ is a set of all parameters defining the ANN, e.g., all weight matrices and biases

- Gradient descent: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla \mathcal{L}(\boldsymbol{\theta}_k)$
where $\alpha_k > 0$ is the **learning rate** or **stepsize** at iteration k



Stochastic Gradient Descent (SGD): Motivation

- The loss has typically this additive structure:

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(y_i, h_{\boldsymbol{\theta}}(\boldsymbol{x}_i))$$

- Evaluation of $\nabla \mathcal{L}(\boldsymbol{\theta})$ takes $\mathcal{O}(m)$ time
- Duplicate samples in $\mathcal{T}_m$?
- Online learning?
- What if we use a single sample or a *mini-batch* instead of the *full-batch* approach? $\Rightarrow$ Stochastic Gradient Descent (SGD)
- The following is based on *Bottou, Curtis and Nocedal: Optimization Methods for Large-Scale Machine Learning, 2018*

Simplifying the Notation

- Let's simplify and generalize the notation before digging deeper into SGD
- The gradient of loss (empirical risk) is

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(y_i, h_{\boldsymbol{\theta}}(\boldsymbol{x}_i))$$

- Let's represent a sample (or a set of samples) by a *seed* s, meaning the realization of s is either an input-output pair (x, y) or a set of pairs $\{(x_i, y_i)\}_{i \in \mathcal{S}}$
- Define f to be a composite of ℓ and prediction h
- As an example, for GD above we can define $s_i \triangleq (x_i, y_i) \in \mathcal{T}^m$ and write

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla f(\theta, s_i)$$

SGD Algorithm

- Stochastic Gradient Descent

  1 Choose an initial iterate $\boldsymbol{\theta}_1$
  2 **for** $k = 1, 2, \ldots$
  3 Generate a realization of the random variable s_k
  4 Compute a stochastic vector $g(\boldsymbol{\theta}_k, s_k)$
  5 Choose a stepsize $\alpha_k > 0$
  6 Set the new iterate as $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha_k \, g(\boldsymbol{\theta}_k, s_k)$

- Possible options of a stochastic vector

$$g(\boldsymbol{\theta}_k, s_k) = \begin{cases} \nabla f(\boldsymbol{\theta}_k, s_k) & \text{single sample, online learning} \\ \frac{1}{m_k} \sum_{i=1}^{m_k} \nabla f(\boldsymbol{\theta}_k, s_{k,i}) & \text{batch/mini-batch} \\ H_k \, \frac{1}{n_k} \sum_{i=1}^{n_k} \nabla f(\boldsymbol{\theta}_k, s_{k,i}) & \text{Newton/quasi-Newton direction} \end{cases}$$

- Holds for picking samples of $\mathcal{T}_m$ *with replacement*, for picking *without replacement* it holds only until dataset gets exhausted in general
- We consider the elements of the random sequence $\{s_k\}$ independent

SGD Convergence Theorem: Overview

- Assumptions and related lemmas
  - Lipschitz continuous gradient $\nabla\mathcal{L}$
  - Bounds on $\mathcal{L}$ and $g(\boldsymbol{\theta}_k, s_k)$
  - Strong convexity of $\mathcal{L}$
- The main theorem shows that the *expected optimality gap*

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] \xrightarrow{k\to\infty} 0$$

where $\mathcal{L}_*$ is the optimal (minimal) loss

Assumption 1: Lipschitz Continuous Gradient

- The loss function is continuously differentiable and the gradient is *Lipschitz continuous* with *Lipschitz constant* $L > 0$:

$$\left\|\nabla\mathcal{L}(\boldsymbol{\theta}) - \nabla\mathcal{L}(\bar{\boldsymbol{\theta}})\right\|_2 \leq L\left\|\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}\right\|_2, \text{ for all } \{\boldsymbol{\theta}, \bar{\boldsymbol{\theta}}\} \in \mathbb{R}^d$$

- *Intuition:* the gradient does not change too quickly w.r.t. $\boldsymbol{\theta}$
- Lemma (see Bottou et al. for the proof):

$$\mathcal{L}(\boldsymbol{\theta}) \leq \mathcal{L}(\bar{\boldsymbol{\theta}}) + \nabla\mathcal{L}(\bar{\boldsymbol{\theta}})^T(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}) + \frac{1}{2}L\left\|\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}\right\|_2^2$$

Lemma 1

- When $\mathcal{L}$ is Lipschitz continuous (Assumption 1) SGD iterates satisfy the following (for all $k \in \mathcal{N}$):

$$\mathbb{E}_{s_k}[\mathcal{L}(\boldsymbol{\theta}_{k+1})] - \mathcal{L}(\boldsymbol{\theta}_k) \leq -\alpha_k \nabla \mathcal{L}(\boldsymbol{\theta}_k)^T \mathbb{E}_{s_k}[g(\boldsymbol{\theta}_k, s_k)] + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{s_k}\left[\|g(\boldsymbol{\theta}_k, s_k)\|_2^2\right] \qquad \text{(L1)}$$

- Proof: use previous lemma and SGD iterate $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha_k \, g(\boldsymbol{\theta}_k, s_k)$

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}_{k+1}) - \mathcal{L}(\boldsymbol{\theta}_k) &\leq \nabla \mathcal{L}(\boldsymbol{\theta}_k)^T (\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k) + \frac{1}{2} L \, \|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\|_2^2 \\ &\leq -\alpha_k \nabla \mathcal{L}(\boldsymbol{\theta}_k)^T g(\boldsymbol{\theta}_k, s_k) + \frac{1}{2}\alpha_k^2 L \, \|g(\boldsymbol{\theta}_k, s_k)\|_2^2\end{aligned}$$

then apply $\mathbb{E}_{s_k}$ and note that only $\boldsymbol{\theta}_{k+1}$ depends on s_k not $\boldsymbol{\theta}_k$

Lemma 1: Discussion

- If $g(\boldsymbol{\theta}_k, s_k)$ is an unbiased estimate of $\nabla \mathcal{L}(\boldsymbol{\theta}_k)$:

$$\mathbb{E}_{s_k}[\mathcal{L}(\boldsymbol{\theta}_{k+1})] - \mathcal{L}(\boldsymbol{\theta}_k) \leq \underbrace{-\alpha_k \left\|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2}_{<0} + \underbrace{\frac{1}{2}\alpha_k^2 L \mathbb{E}_{s_k}\left[\left\|g(\boldsymbol{\theta}_k, s_k)\right\|_2^2\right]}_{\geq 0}$$

- In general, we want the LHS < 0 and bounded above by a deterministic quantity
- The second term harms the convergence
- We will restrict the variance of $g(\boldsymbol{\theta}_k, s_k)$:

$$\mathrm{Var}_{s_k}[g(\boldsymbol{\theta}_k, s_k)] \triangleq \mathbb{E}_{s_k}\left[\left\|g(\boldsymbol{\theta}_k, s_k)\right\|_2^2\right] - \left\|\mathbb{E}_{s_k}[g(\boldsymbol{\theta}_k, s_k)]\right\|_2^2$$

to make the $\mathbb{E}_{s_k}\left[\left\|g(\boldsymbol{\theta}_k, s_k)\right\|_2^2\right]$ small enough

Assumption 2: Bounds

- (B1): $\mathcal{L}$ is bounded below by a scalar $\mathcal{L}_{inf}$
- (B2) make direction and norm of $g(\boldsymbol{\theta}_k, s_k)$ comparable to $\nabla\mathcal{L}(\boldsymbol{\theta}_k)$: there exist $\mu_G \geq \mu > 0$ that for all $k \in \mathbb{N}$
  - (B2a): $\nabla\mathcal{L}(\boldsymbol{\theta}_k)^T \mathbb{E}_{s_k}[g(\boldsymbol{\theta}_k, s_k)] \geq \mu \left\|\nabla\mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2$
  - (B2b): $\left\|\mathbb{E}_{s_k}[g(\boldsymbol{\theta}_k, s_k)]\right\|_2 \leq \mu_G \left\|\nabla\mathcal{L}(\boldsymbol{\theta}_k)\right\|_2$
- (B3): there exist $M \geq 0$ and $M_V \geq 0$ that for all $k \in \mathbb{N}$ we have:

$$\mathrm{Var}_{s_k}[g(\boldsymbol{\theta}_k, s_k)] \leq M + M_V \left\|\nabla\mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2$$

- Corollary (from B3 and B2b):

$$\mathbb{E}_{s_k}\left[\left\|g(\boldsymbol{\theta}_k, s_k)\right\|_2^2\right] \leq M + M_G \left\|\nabla\mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2$$

with $M_G \triangleq M_V + \mu_G^2 \geq \mu^2 > 0$

Lemma 2

- Assuming Lipschitz continuity and the Bounds for all $k \in \mathbb{N}$ we have:

$$\mathbb{E}_{s_k}[\mathcal{L}(\boldsymbol{\theta}_{k+1})] - \mathcal{L}(\boldsymbol{\theta}_k) \leq$$

$$\leq -\mu\alpha_k \left\|\nabla\mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2 + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{s_k}\left[\left\|g(\boldsymbol{\theta}_k, s_k)\right\|_2^2\right] \quad \text{(L2a)}$$

$$\leq -(\mu - \frac{1}{2}\alpha_k L M_G)\alpha_k \left\|\nabla\mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2 + \frac{1}{2}\alpha_k^2 L M \quad \text{(L2b)}$$

- Proof:

$$\mathbb{E}_{s_k}[\mathcal{L}(\boldsymbol{\theta}_{k+1})] - \mathcal{L}(\boldsymbol{\theta}_k) \overset{(L1)}{\leq} -\alpha_k \nabla\mathcal{L}(\boldsymbol{\theta}_k)^T \mathbb{E}_{s_k}[g(\boldsymbol{\theta}_k, s_k)]$$

$$+ \frac{1}{2}\alpha_k^2 L \mathbb{E}_{s_k}[\left\|g(\boldsymbol{\theta}_k, s_k)\right\|_2^2]$$

$$\overset{(B2a)}{\leq} -\mu\alpha_k \left\|\nabla\mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2 + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{s_k}\left[\left\|g(\boldsymbol{\theta}_k, s_k)\right\|_2^2\right]$$

Lemma 2

- Use $\mathbb{E}_{s_k}\left[\|g(\boldsymbol{\theta}_k, s_k)\|_2^2\right] \leq M + M_G \|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\|_2^2$:

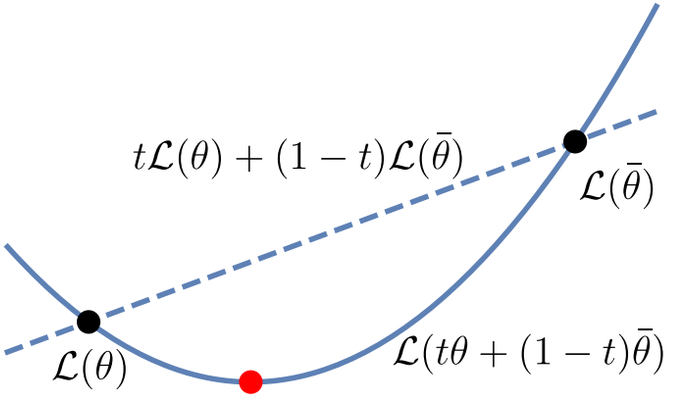
$$\begin{aligned}
\mathbb{E}_{s_k}[\mathcal{L}(\boldsymbol{\theta}_{k+1})] - \mathcal{L}(\boldsymbol{\theta}_k) &\leq -\mu\alpha_k \|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\|_2^2 + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{s_k}\left[\|g(\boldsymbol{\theta}_k, s_k)\|_2^2\right] \\
&\leq -\mu\alpha_k \|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\|_2^2 + \frac{1}{2}\alpha_k^2 L \left(M + M_G \|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\|_2^2\right) \\
&\leq -(\mu - \frac{1}{2}\alpha_k L M_G)\alpha_k \|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\|_2^2 + \frac{1}{2}\alpha_k^2 L M
\end{aligned}$$

Convexity

- Convex function definition:

$$\mathcal{L}(t\boldsymbol{\theta} + (1-t)\bar{\boldsymbol{\theta}}) \leq t\mathcal{L}(\boldsymbol{\theta}) + (1-t)\mathcal{L}(\bar{\boldsymbol{\theta}})$$

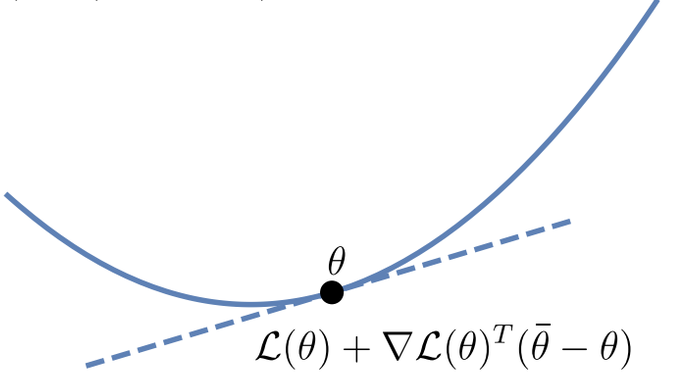
for all $(\boldsymbol{\theta}, \bar{\boldsymbol{\theta}}) \in \mathbb{R}^d \times \mathbb{R}^d$



- Equivalently (first-order condition):

$$\mathcal{L}(\bar{\boldsymbol{\theta}}) \geq \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^T(\bar{\boldsymbol{\theta}} - \boldsymbol{\theta})$$

the function lies above all its tangents



- See A4B33OPT
- But we need a stronger assumption...

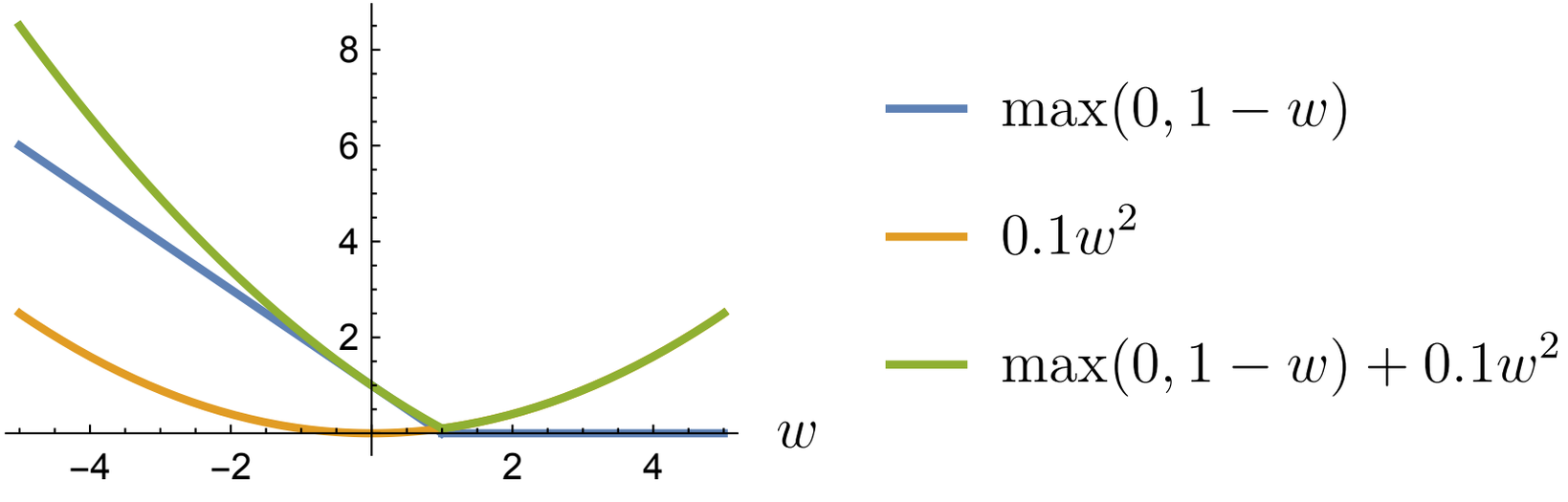
Strong Convexity

- The loss function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ is strongly convex if there exists constant $c > 0$ such that

$$\mathcal{L}(\bar{\boldsymbol{\theta}}) \geq \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^T(\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}) + \frac{1}{2}c\left\|\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}\right\|_2^2 \qquad \text{(C1)}$$

  for all $(\boldsymbol{\theta}_k, \boldsymbol{\theta}) \in \mathbb{R}^d \times \mathbb{R}^d$

- *Intuition*: quadratic lower bound on function growth
- Example (SVM objective): $\max(0, 1 - y(\boldsymbol{w}^T\boldsymbol{x} + b)) + \frac{\lambda}{2}\left\|\boldsymbol{w}\right\|_2^2$



Strong Convexity: Corollaries

- $\mathcal{L}$ has an uniqe minimizer $\boldsymbol{\theta}_* \in \mathbb{R}^d$ with $\mathcal{L}_* \triangleq \mathcal{L}(\boldsymbol{\theta}_*)$
- **Corollary 1** Polyak-Lojasiewicz inequality:

$$2c(\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}_*) \leq \|\nabla \mathcal{L}(\boldsymbol{\theta})\|_2^2 \qquad \text{for all } \boldsymbol{\theta} \in \mathbb{R}^d \qquad \text{(C2)}$$

- **Corollary 2**: $c \leq L$:
  use (C1):

$$\mathcal{L}(\bar{\boldsymbol{\theta}}) \geq \mathcal{L}(\boldsymbol{\theta}) + \nabla \mathcal{L}(\boldsymbol{\theta})^T(\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}) + \frac{1}{2}c\left\|\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}\right\|_2^2$$

  and the Lipschitz continuous gradient:

$$\mathcal{L}(\bar{\boldsymbol{\theta}}) \leq \mathcal{L}(\boldsymbol{\theta}) + \nabla \mathcal{L}(\boldsymbol{\theta})^T(\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}) + \frac{1}{2}L\left\|\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}\right\|_2^2$$

SGD Convergence: Strongly Convex $\mathcal{L}$, Fixed Stepsize

- **Theorem:** assuming Lipschitz continuity, the Bounds and strong convexity of $\mathcal{L}$, the SGD is run with a fixed stepsize $\alpha_k = \alpha$ for all $k \in \mathbb{N}$, where $0 < \alpha \leq \frac{\mu}{LM_G}$. Then the *expected optimality gap* satisfies the following for all k:

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] \leq \frac{\alpha LM}{2c\mu} + (1 - \alpha c\mu)^{k-1} \left(\mathcal{L}(\boldsymbol{\theta}_1) - \mathcal{L}_* - \frac{\alpha LM}{2c\mu} \right)$$
$$\xrightarrow{k \to \infty} \frac{\alpha LM}{2c\mu}$$

- Note: $(1 - \alpha c\mu)^{k-1} \xrightarrow{k \to \infty} 0$ as $0 < \alpha c\mu \leq \frac{c\mu^2}{LM_G} \leq \frac{c\mu^2}{L\mu^2} = \frac{c}{L} \leq 1$
- In general, for the fixed stepsize, the *optimality gap* tends to zero, but converges to $\frac{\alpha LM}{2c\mu} \geq 0$

SGD Convergence: Strongly Convex $\mathcal{L}$, Fixed Stepsize 2

- **Proof:**

$$\begin{aligned}\mathbb{E}_{s_k}[\mathcal{L}(\boldsymbol{\theta}_{k+1})] - \mathcal{L}(\boldsymbol{\theta}_k) &\overset{(L2b)}{\leq} -(\mu - \frac{1}{2}\alpha L M_G)\alpha \left\|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2 + \frac{1}{2}\alpha^2 L M \\ &\leq -\frac{1}{2}\alpha\mu \left\|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2 + \frac{1}{2}\alpha^2 L M \\ &\overset{(C2)}{\leq} -\alpha c \mu(\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*) + \frac{1}{2}\alpha^2 L M\end{aligned}$$

- Subtract $\mathcal{L}_*$ from both sides and rearange:

$$\mathbb{E}_{s_k}[\mathcal{L}(\boldsymbol{\theta}_{k+1})] - \mathcal{L}_* \leq (1 - \alpha c \mu)(\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*) + \frac{1}{2}\alpha^2 L M$$

- Take total expectation, i.e., $\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k)] = \mathbb{E}_{s_1}\mathbb{E}_{s_2}\ldots\mathbb{E}_{s_{k-1}}[\mathcal{L}(\boldsymbol{\theta}_k)]$:

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_{k+1}) - \mathcal{L}_*] \leq (1 - \alpha c \mu)\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] + \frac{1}{2}\alpha^2 L M$$

SGD Convergence: Strongly Convex $\mathcal{L}$, Fixed Stepsize 3

- We have:

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_{k+1}) - \mathcal{L}_*] \leq (1 - \alpha c\mu)\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] + \frac{1}{2}\alpha^2 LM$$

- Subtract $\frac{\alpha LM}{2c\mu}$ from both sides:

$$\begin{aligned}\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_{k+1}) - \mathcal{L}_*] - \frac{\alpha LM}{2c\mu} &\leq (1 - \alpha c\mu)\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] + \frac{\alpha^2 LM}{2} - \frac{\alpha LM}{2c\mu} \\ &= (1 - \alpha c\mu)\left(\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] - \frac{\alpha LM}{2c\mu}\right)\end{aligned}$$

- Now just iterate to get:

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] \leq \frac{\alpha LM}{2c\mu} + (1 - \alpha c\mu)^{k-1}\left(\mathcal{L}(\boldsymbol{\theta}_1) - \mathcal{L}_* - \frac{\alpha LM}{2c\mu}\right)$$

Batch Gradient Descent

- How does the theorem apply to the batch setting (GD)?
- The $g(\boldsymbol{\theta}_k, s_k)$ is an unbiased estimate of $\nabla\mathcal{L}(\boldsymbol{\theta}_k)$
- No noise in gradient estimate $g(\boldsymbol{\theta}_k, s_k)$, i.e., $\mathrm{Var}_{s_k}[g(\boldsymbol{\theta}_k, s_k)] = 0$ implies $M = 0$, as we have $\mathrm{Var}_{s_k}[g(\boldsymbol{\theta}_k, s_k)] \leq M + M_V \left\|\nabla\mathcal{L}(\boldsymbol{\theta}_k)\right\|_2^2$
- The optimality gap simplifies to:

$$\epsilon_k = \mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] \leq (1 - \alpha c \mu)^{k-1} \left(\mathcal{L}(\boldsymbol{\theta}_1) - \mathcal{L}_*\right) \xrightarrow{k\to\infty} 0$$

- Asymptotically we have $\epsilon_k \leq \mathcal{O}(\rho^k)$, $\rho \in [0, 1)$
- For a given gap ϵ, the number of iterations k is proportional to $\log(1/\epsilon)$ in the worst case.

SGD Convergence: Strongly Convex $\mathcal{L}$, Diminishing Stepsize

- **Theorem:** assuming the Lipschitz continuity of $\nabla\mathcal{L}$, the Bounds and the strong convexity of $\mathcal{L}$, the SGD is run with a stepsize such that, for all k

$$\alpha_k = \frac{\beta}{\gamma + k} \text{ for some } \beta > \frac{1}{c\mu} > 0 \text{ and } \gamma > 0 \text{ such that } \alpha_1 \leq \frac{\mu}{LM_G}$$

Then the *expected optimality gap* satisfies the following for all k:

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] \leq \frac{v}{\gamma + k}$$

where

$$v \triangleq \max\left\{\frac{\beta^2 LM}{2(\beta c\mu - 1)}, (\gamma + 1)(\mathcal{L}(\boldsymbol{\theta}_1) - \mathcal{L}_*)\right\}$$

- Now we have $\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}_*] \xrightarrow{k\to\infty} 0$
- Asymptotically the gap is $\epsilon_k \leq \mathcal{O}(1/k)$ which means that the number of iterations k is proportional to $1/\epsilon$ in the worst case

GD vs SGD

	GD	SGD
time per iteration	m	1
iterations for accuracy ϵ	$\log(1/\epsilon)$	$1/\epsilon$
time for accuracy ϵ	$m \log(1/\epsilon)$	$1/\epsilon$

- SGD time does not depend on dataset size (if not exhausted)
- For large-scale problems (large m) SGD is faster
- It is harder to tune stepsize schedule for SGD, but you can experiment on a small representative subset of the dataset
- In practise *mini-batches* are used to leverage optmization/parallelization on CPU/GPU

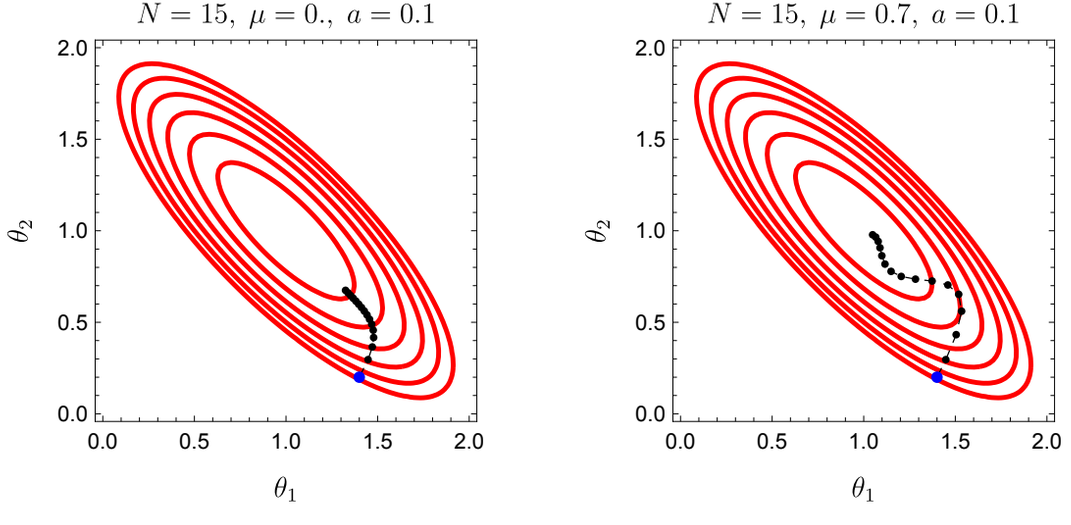
Momentum

- Simulate inertia to overcome plateaus in the error landscape:

$$\boldsymbol{v}_{k+1} \leftarrow \mu \boldsymbol{v}_k - \alpha_k \, g(\boldsymbol{\theta}_k, s_k)$$
$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \boldsymbol{v}_{k+1}$$

  where $\mu \in [0, 1]$ is the *momentum parameter*

- Momentum damps oscillations in directions of high curvature
- It builds velocity in directions with consistent (possibly small) gradient



Adagrad

- Adaptive Gradient method (Duchi, Hazan and Singer, 2011)
- Motivation: a magnitude of gradient differs a lot for different parameters
- Idea: reduce learning rates for parameters having high values of gradient

$$G_{k+1,i} \leftarrow G_{k,i} + [g(\boldsymbol{\theta}_k, s_k)]_i^2$$

$$\theta_{k+1,i} \leftarrow \theta_{k,i} - \frac{\alpha}{\sqrt{G_{k+1,i}} + \epsilon} \cdot [g(\boldsymbol{\theta}_k, s_k)]_i$$

- $G_{k,i}$ accumulates squared partial derivative approximations w.r.t. to the parameter $\theta_{k,i}$
- ϵ is a small positive number to prevent division by zero
- Weakness: ever increasing G_i leads to slow convergence eventually

RMSProp

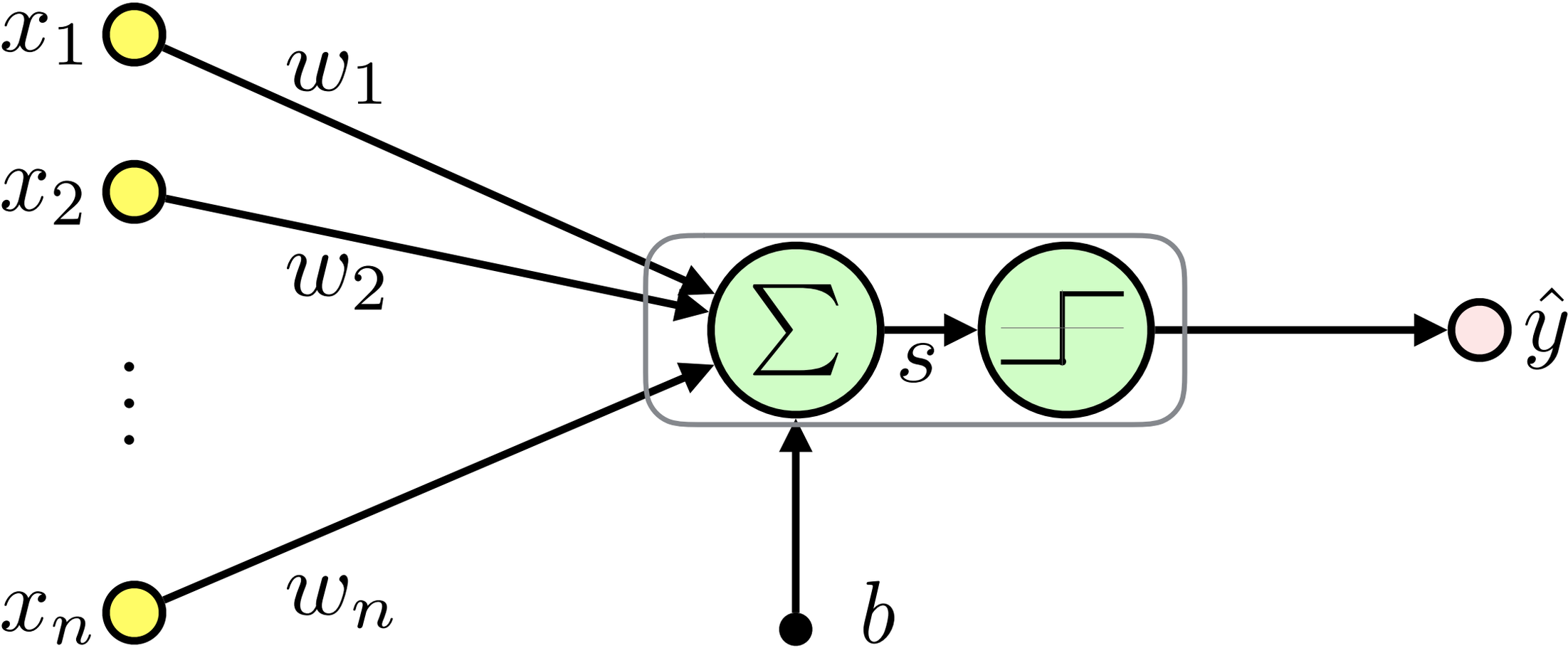
- Similar to Adagrad but employs a moving average:

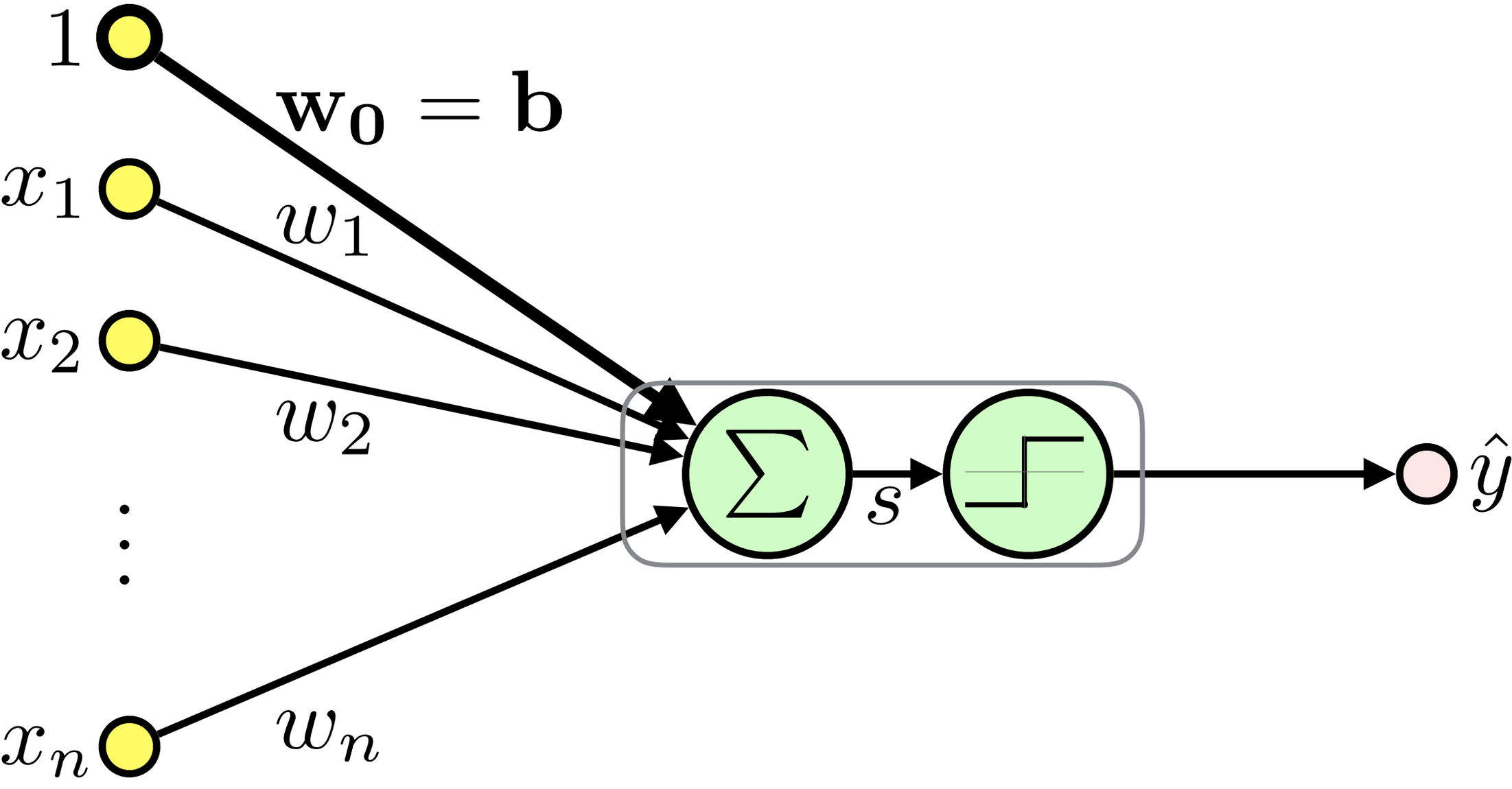
$$G_{k+1,i} = \gamma G_{k,i} + (1-\gamma)\left[g(\boldsymbol{\theta}_k, s_k)\right]_i^2$$

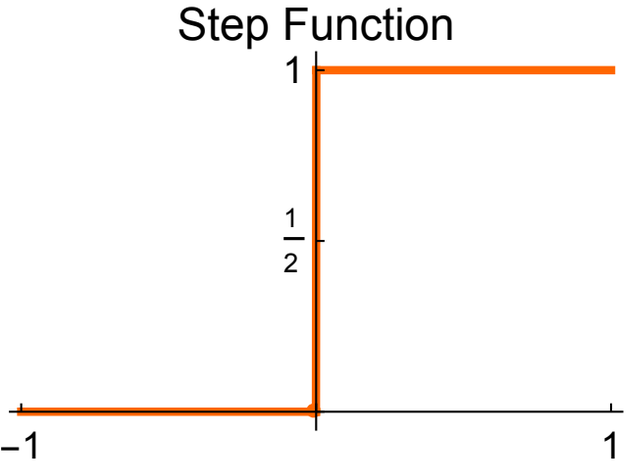
- γ is a *decay* parameter (typical value $\gamma = 0.9$)
- Unlike for Adagrad updates do not get infinitesimally small

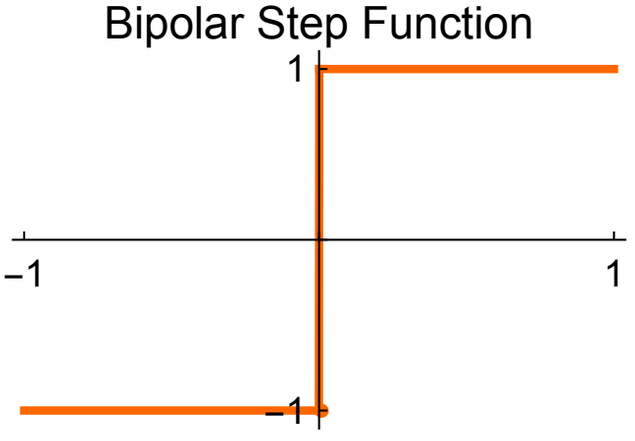
Next Lecture

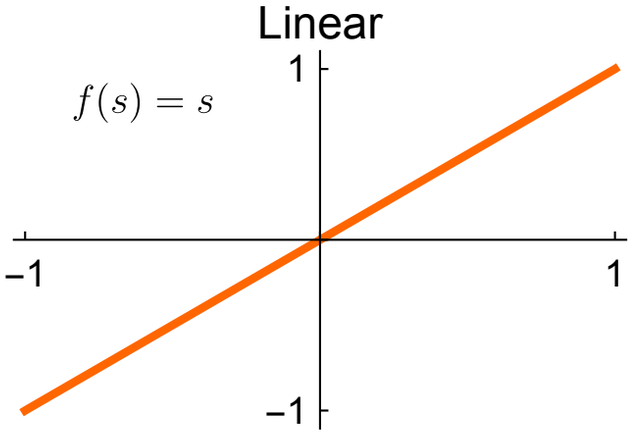
- Deep Neural Networks
- Convolutional Neural Networks
- Transfer learning

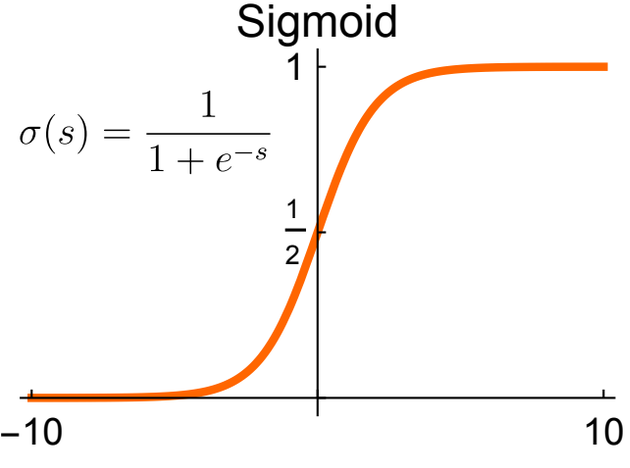

x_1
w_1
x_2
w_2
x_n
w_n
Σ
s
b
$\hat{y}$

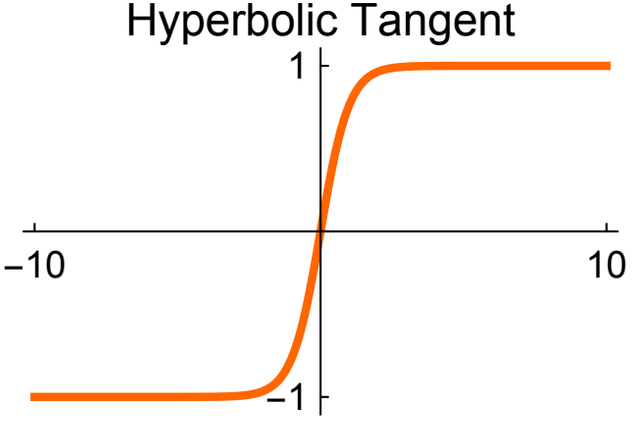

1
$\mathbf{w_0 = b}$
x_1
w_1
x_2
w_2
x_n
w_n
$\sum$
s
$\hat{y}$

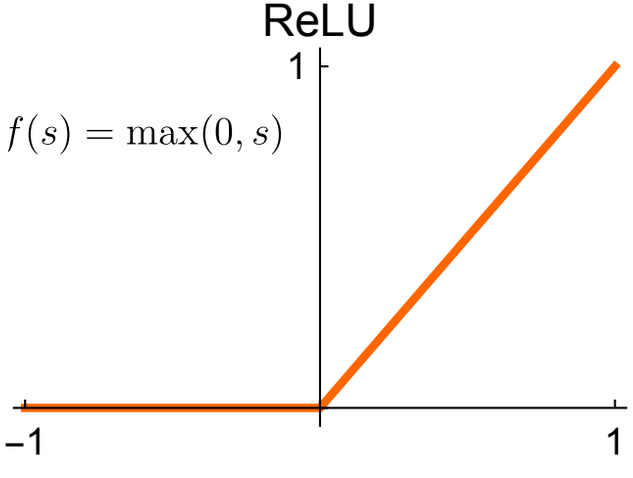

Step Function
1
1/2
−1
1

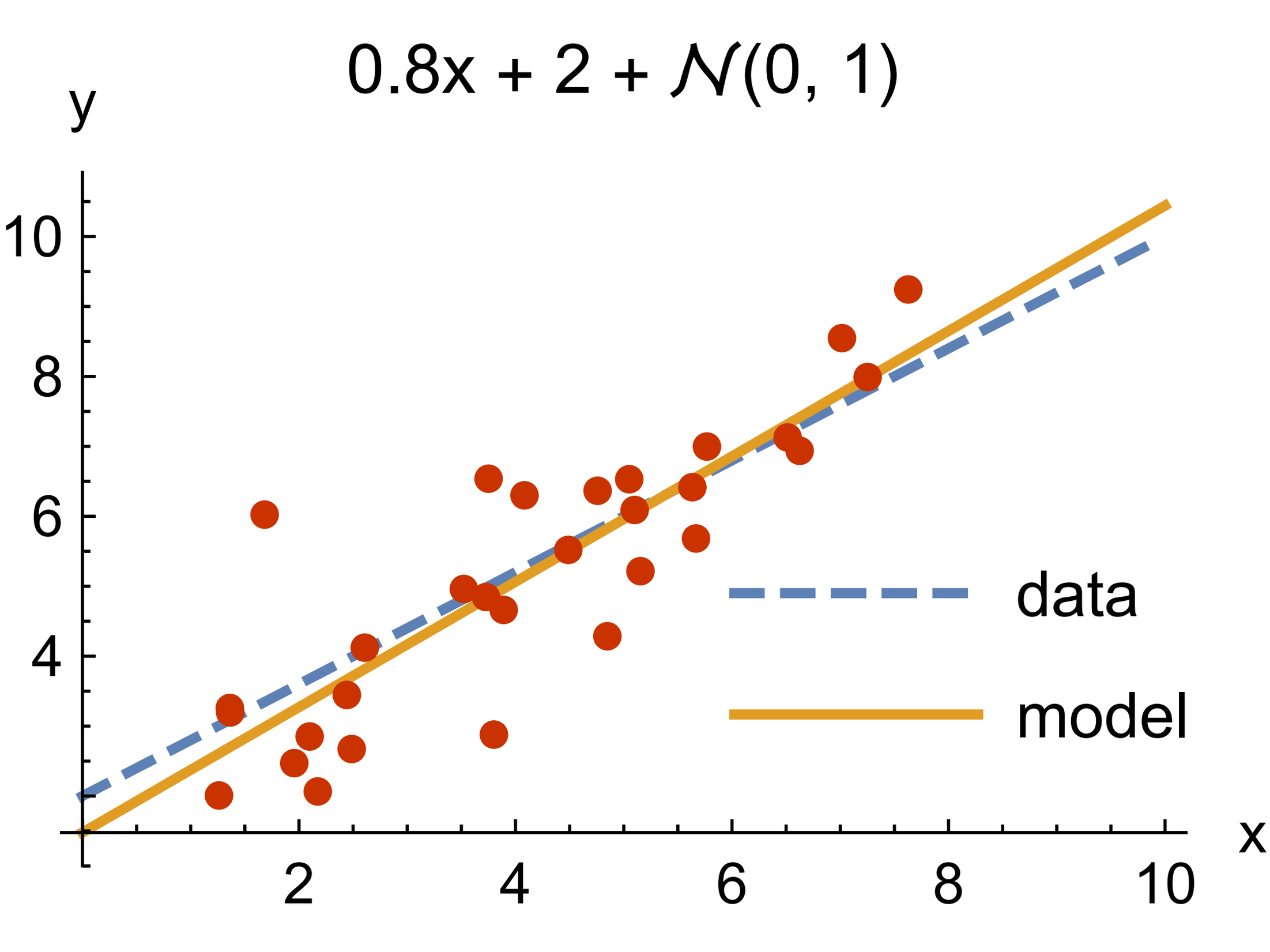

Bipolar Step Function
1
−1
1
−1

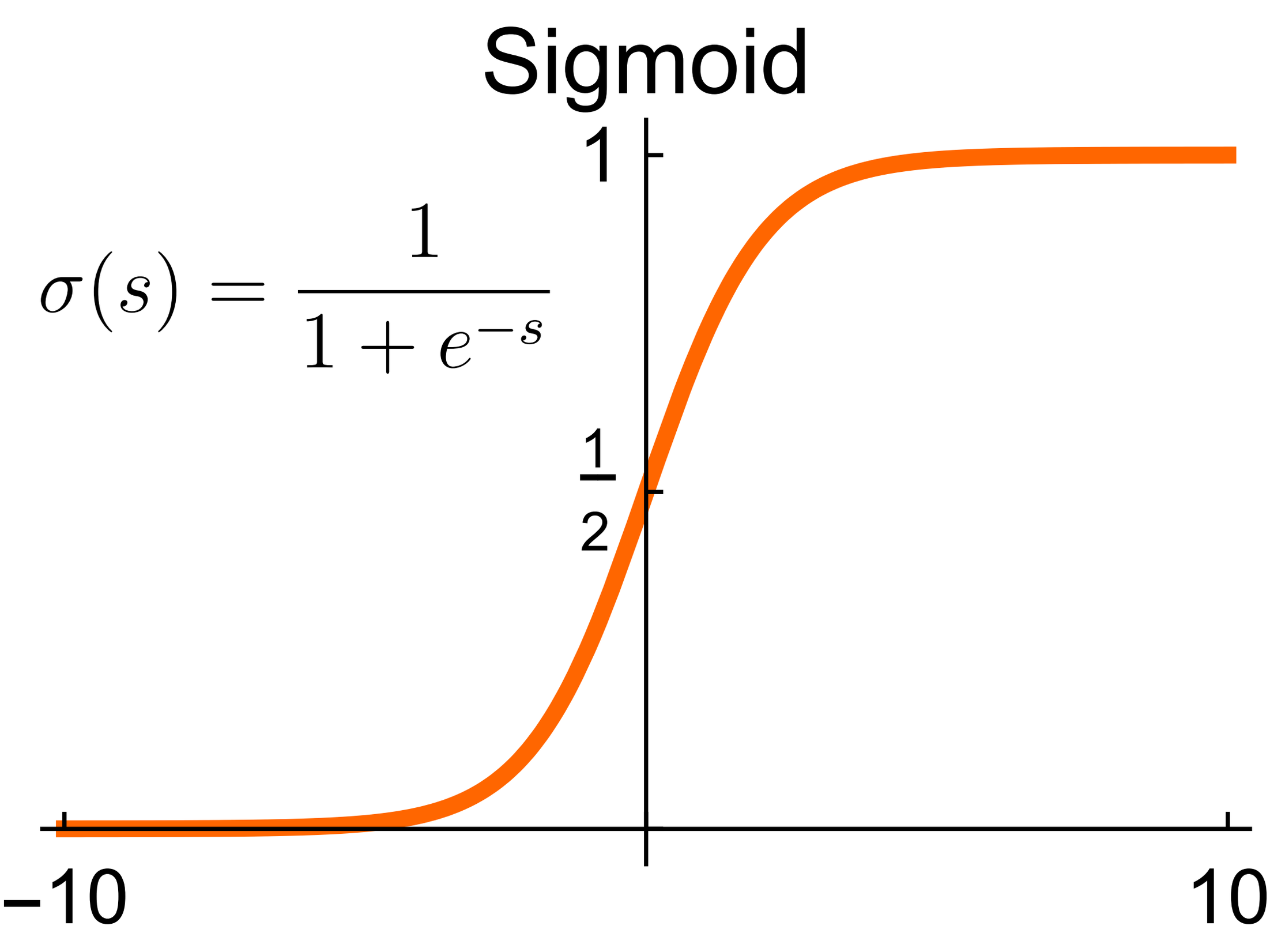

Linear
$f(s) = s$
1
−1
1
−1

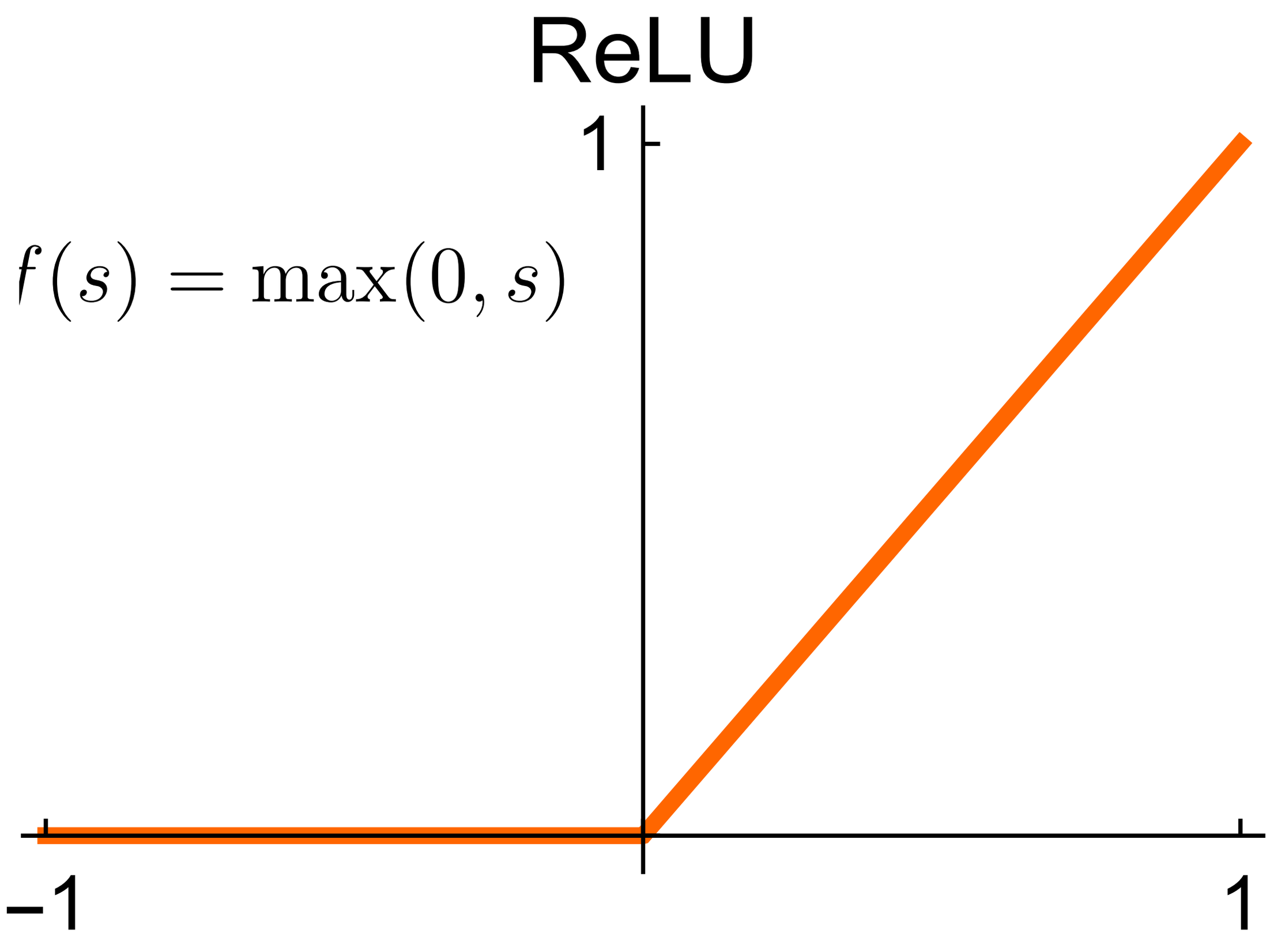

Sigmoid
$\sigma(s) = \frac{1}{1+e^{-s}}$
1
1/2
−10
10

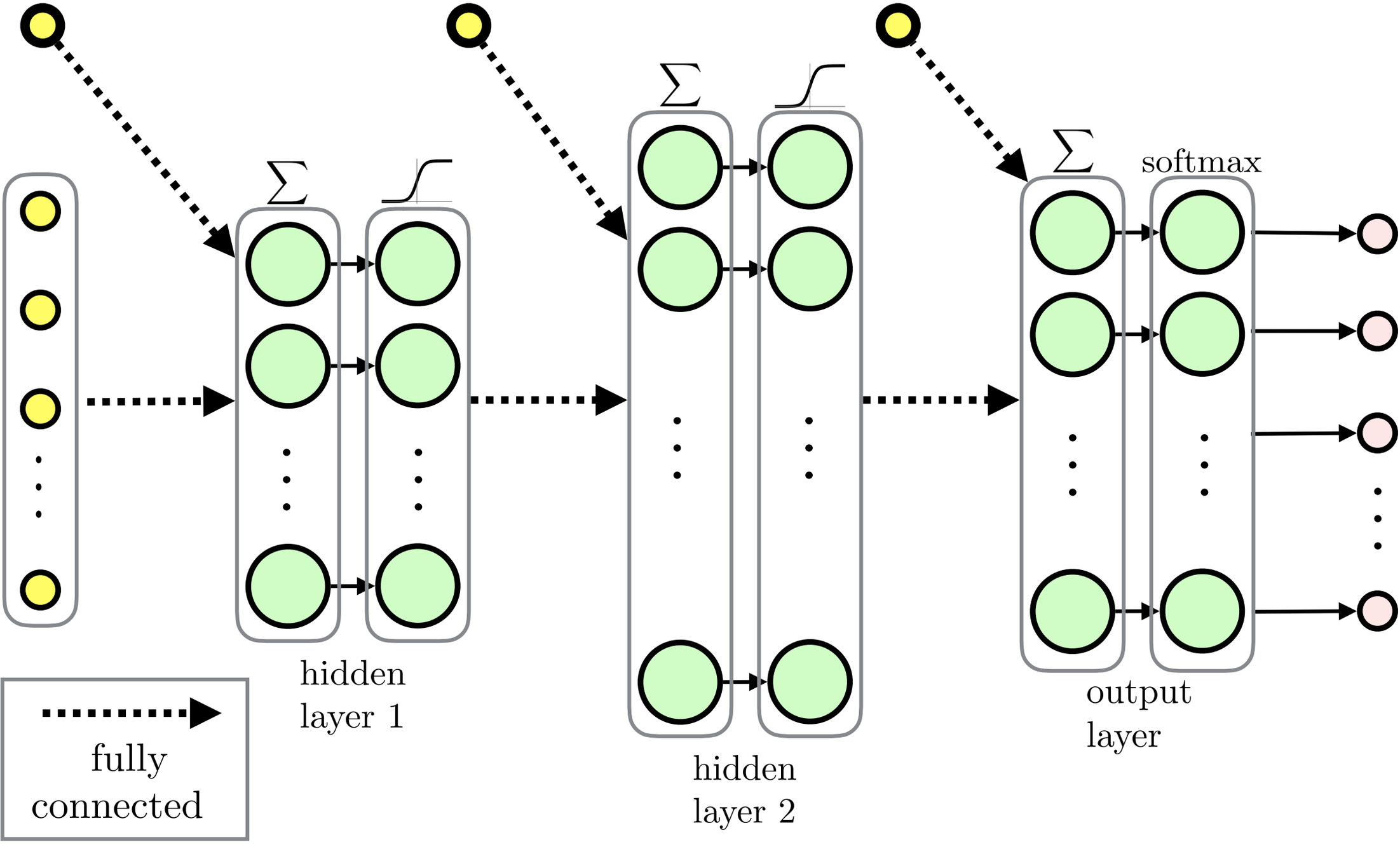

Hyperbolic Tangent
1
−10
10
−1

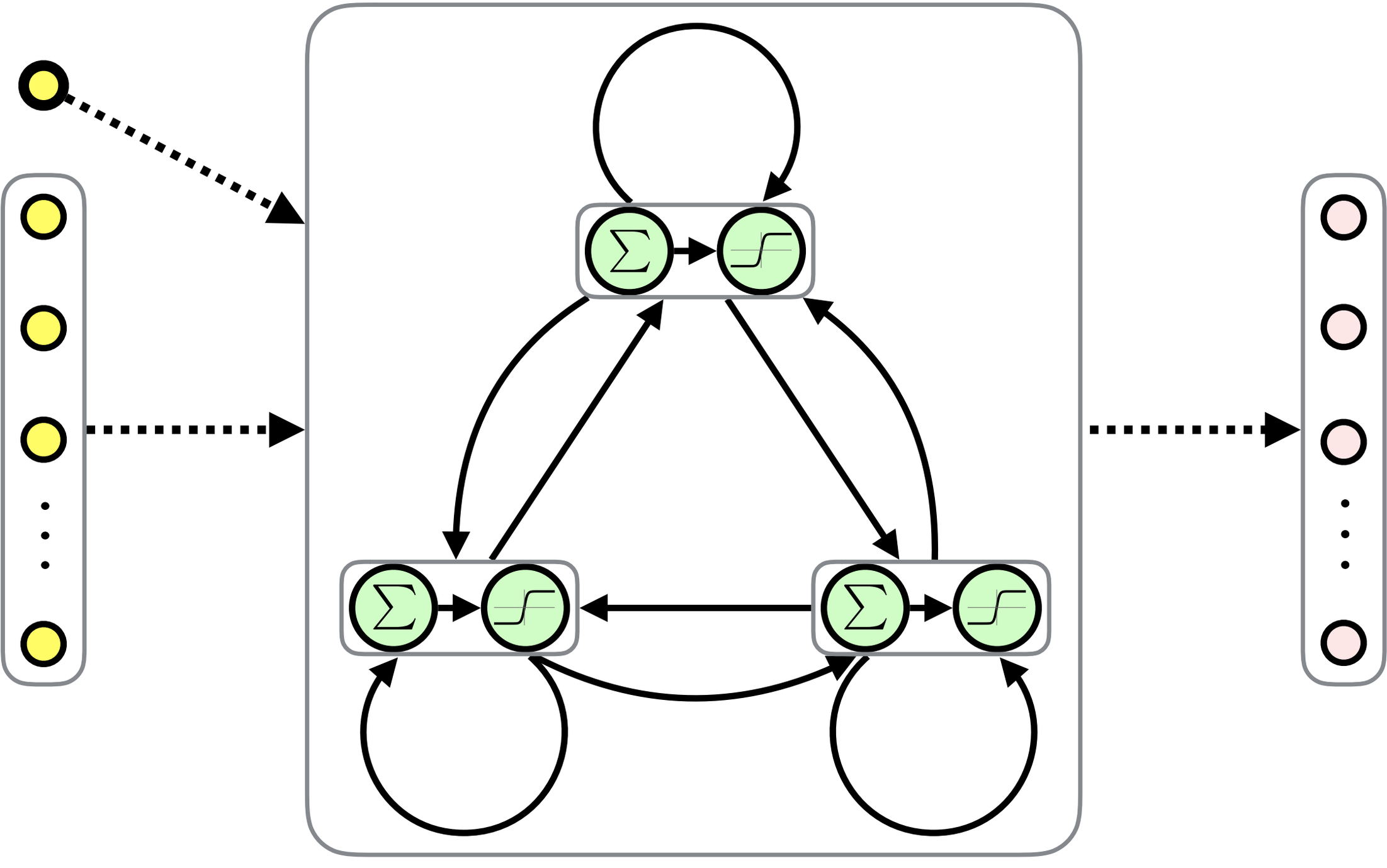

ReLU
$f(s) = \max(0, s)$
1
−1
1

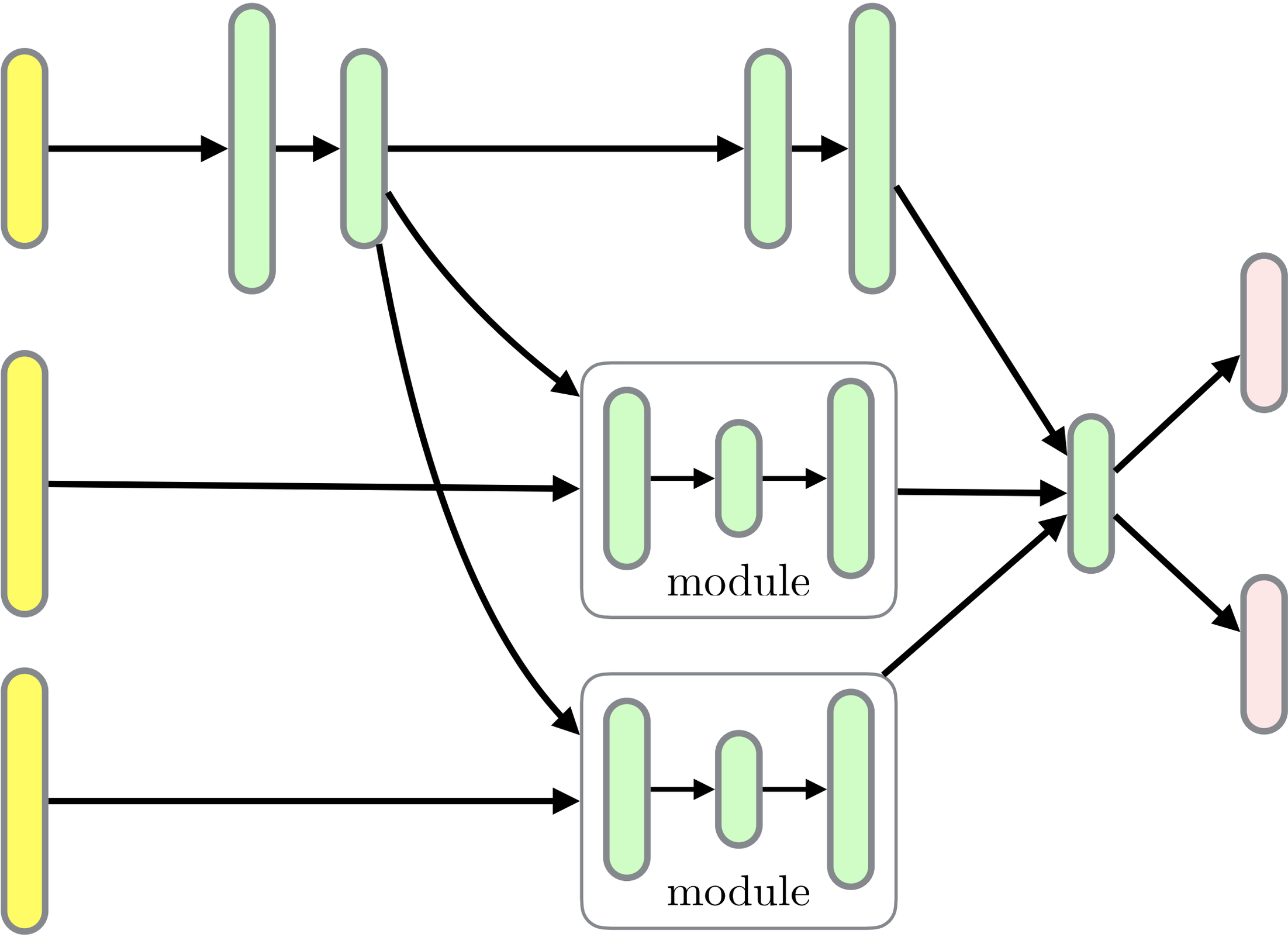

0.8x + 2 + 𝒩(0, 1)
y
10
8
6
4
2
4
6
8
10
x
data
model

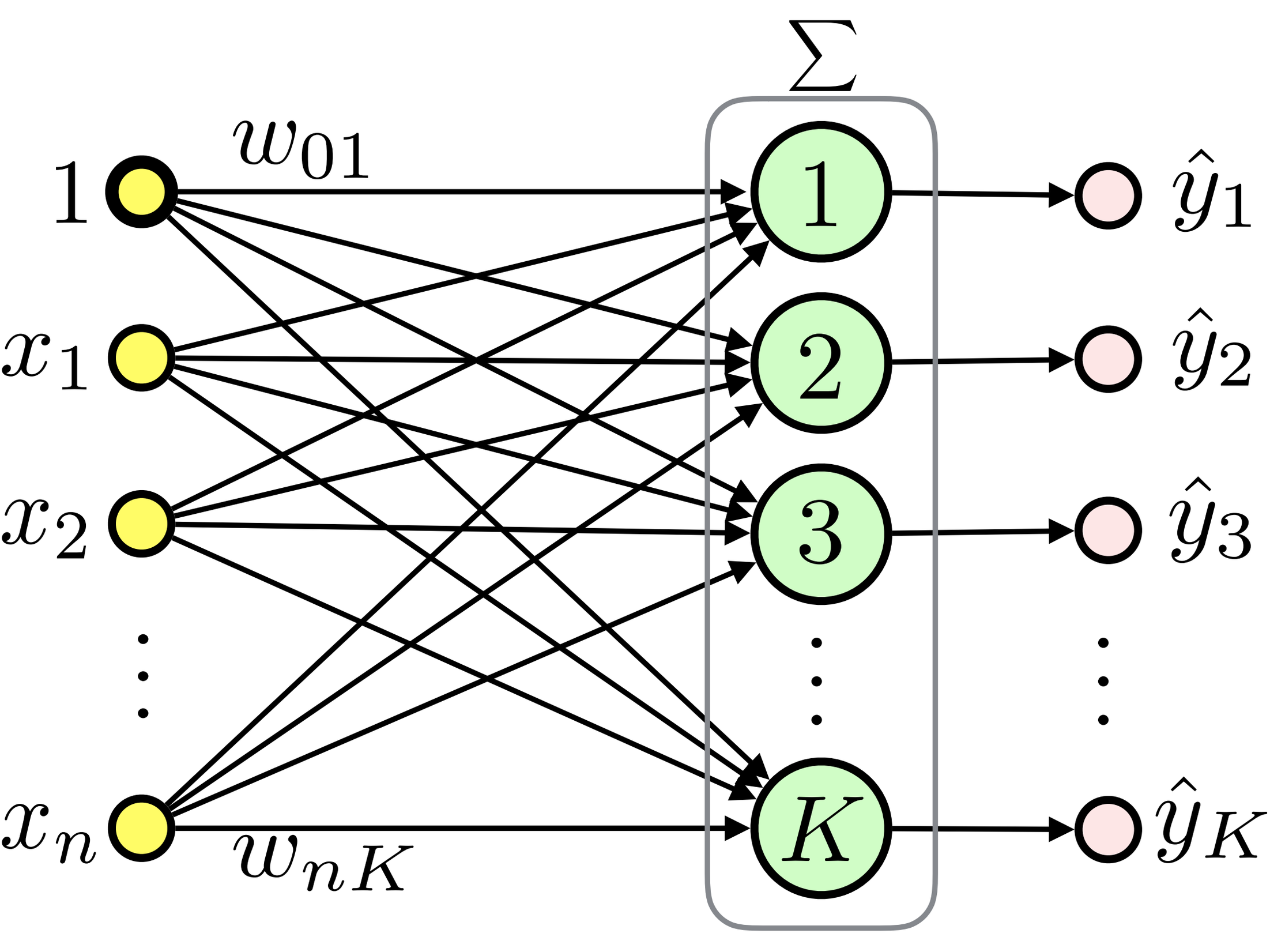

Sigmoid
$\sigma(s) = \frac{1}{1+e^{-s}}$
1
$\frac{1}{2}$
−10
10

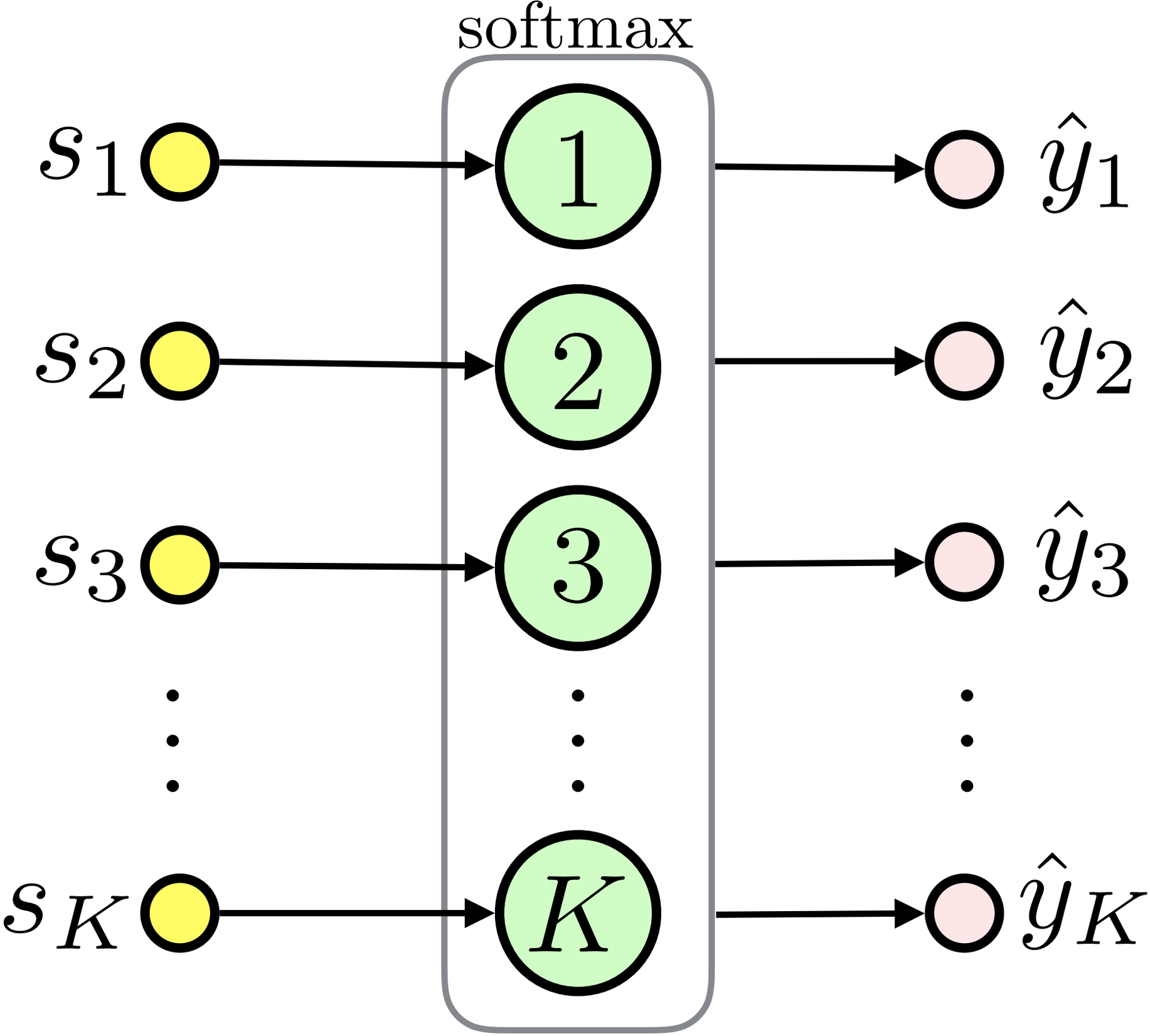

ReLU
$f(s) = \max(0, s)$
1
−1
1

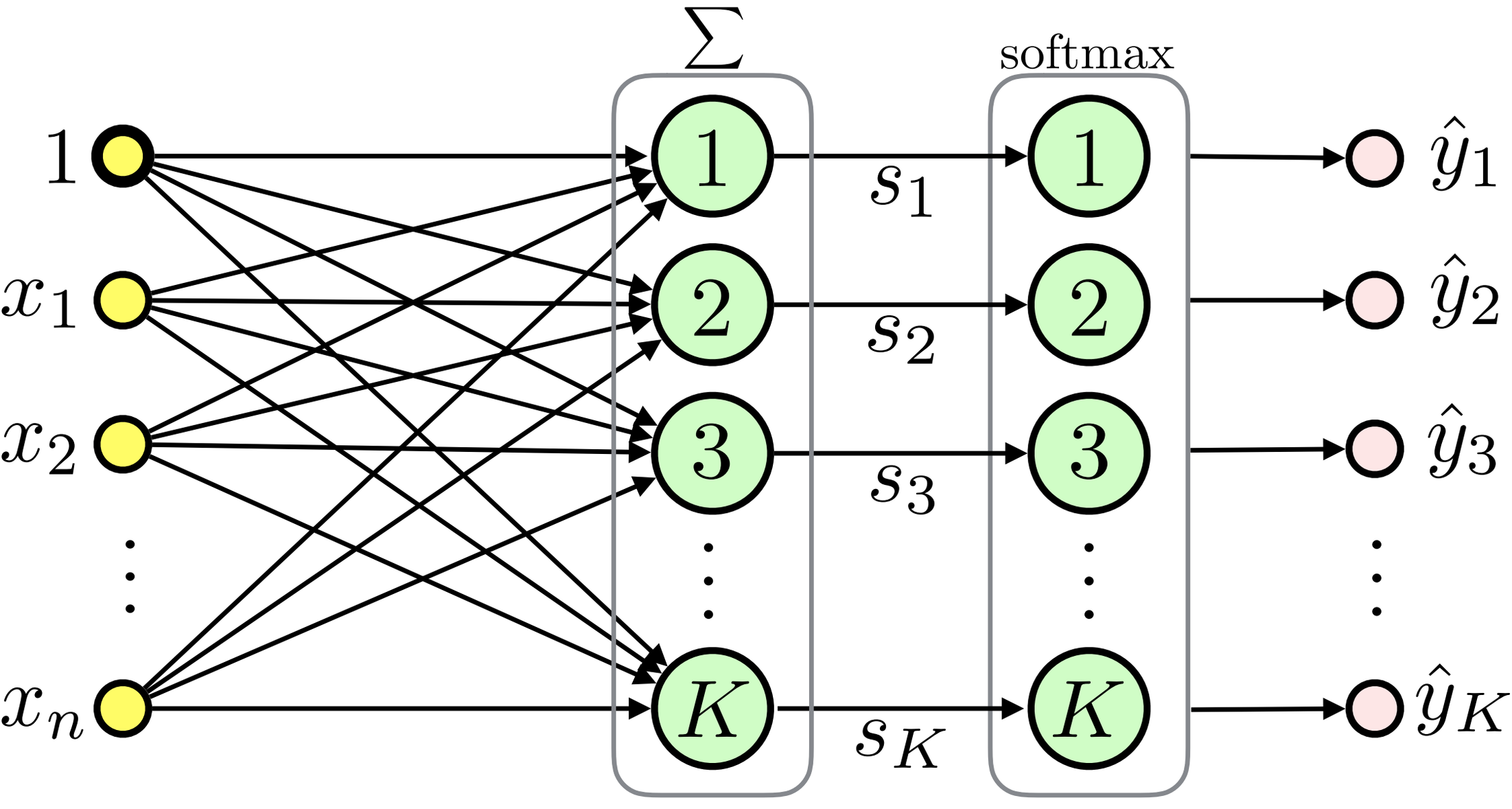

Σ
Σ
Σ
softmax
hidden
layer 1
hidden
layer 2
output
layer
fully
connected

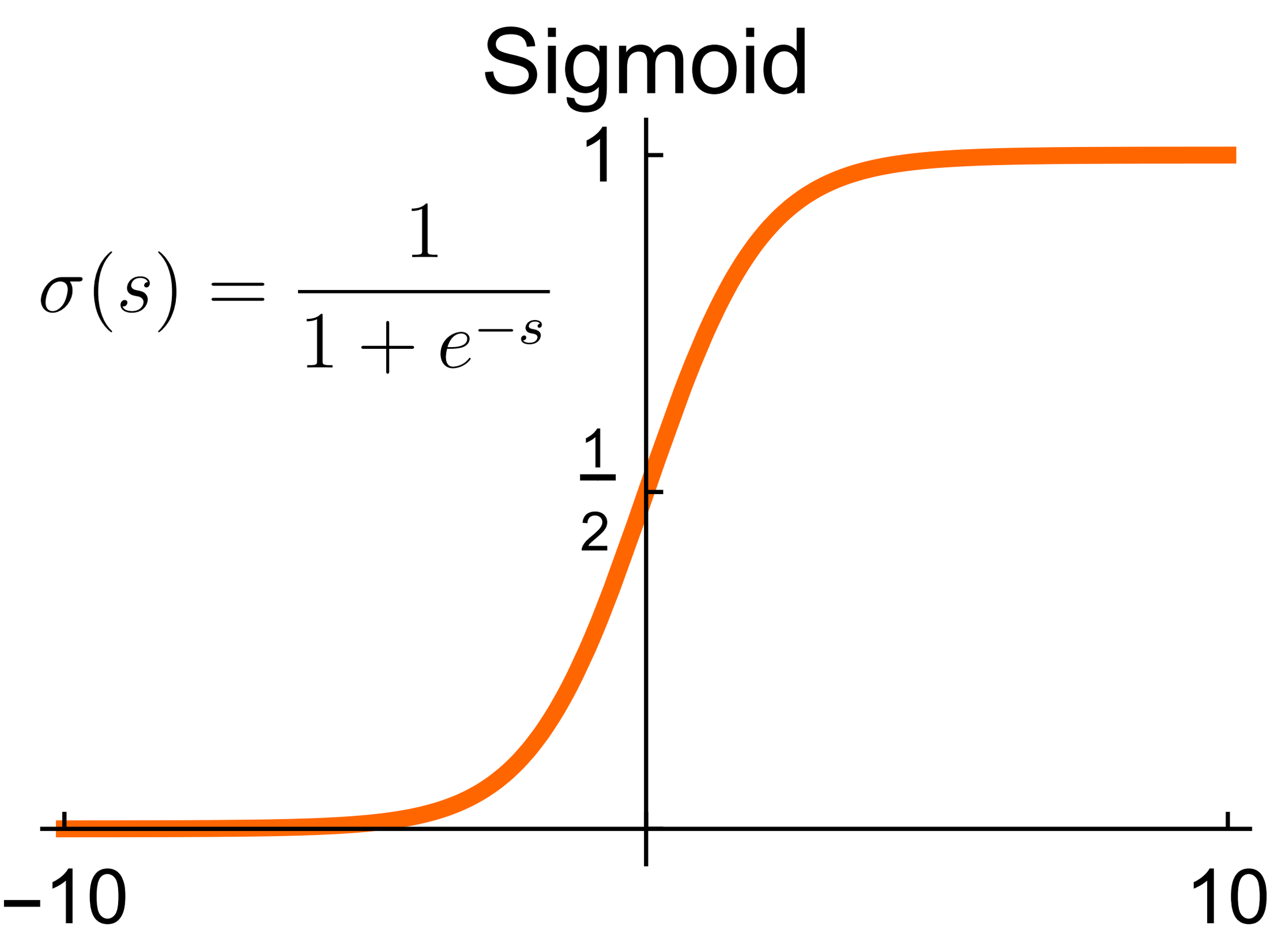


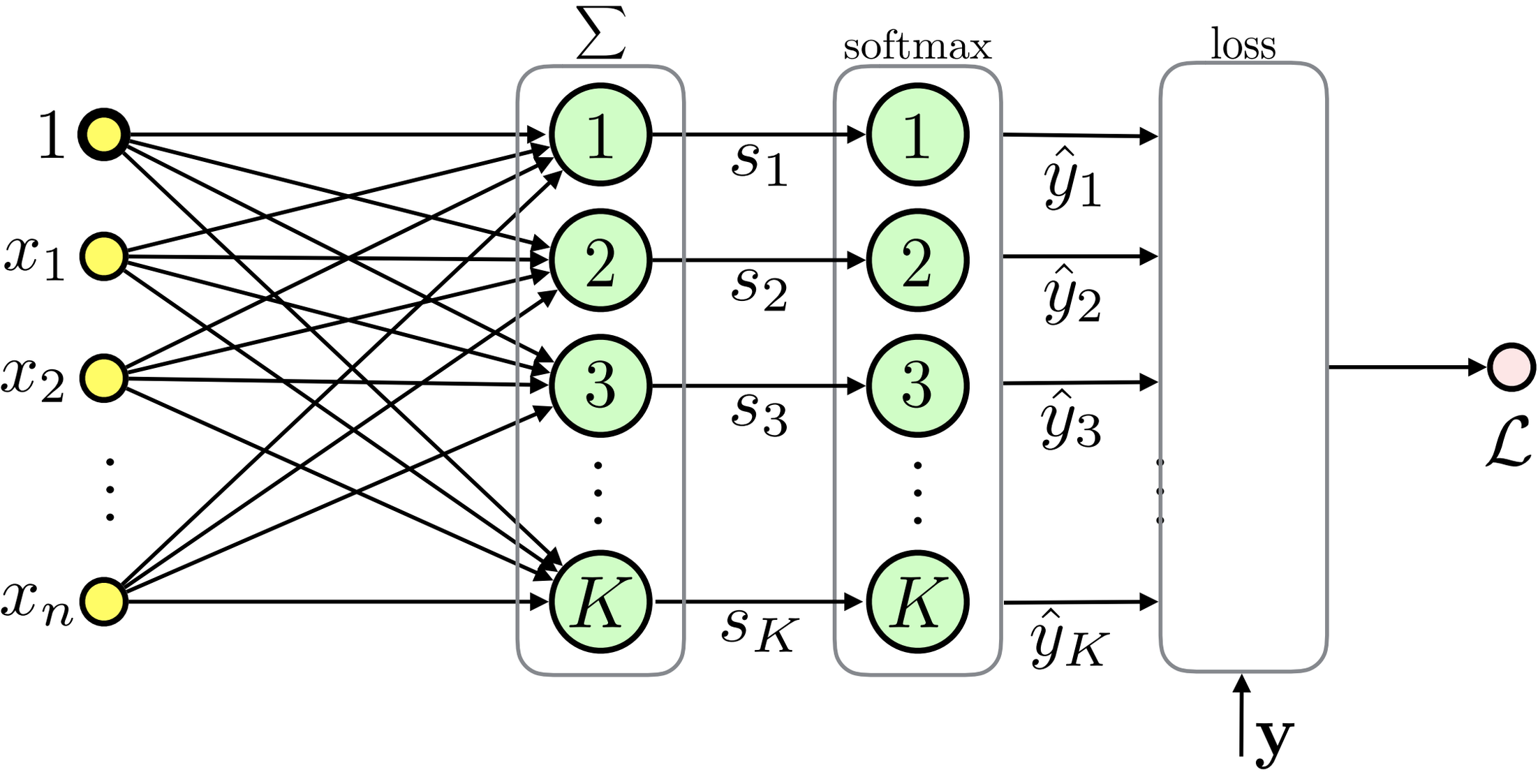

module
module

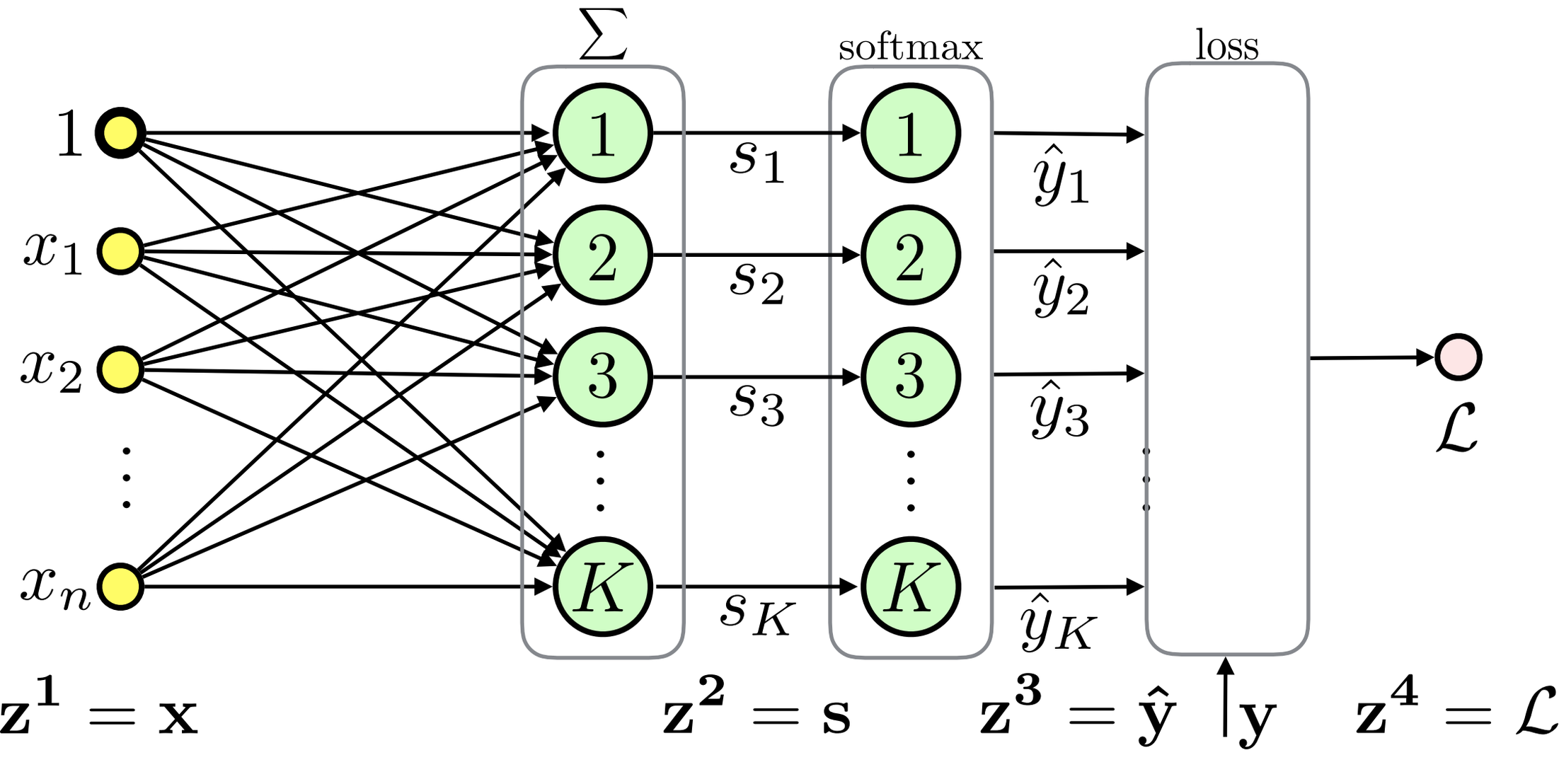

$\sum$
w_{01}
1
x_1
x_2
x_n
w_{nK}
1
2
3
K
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_K$

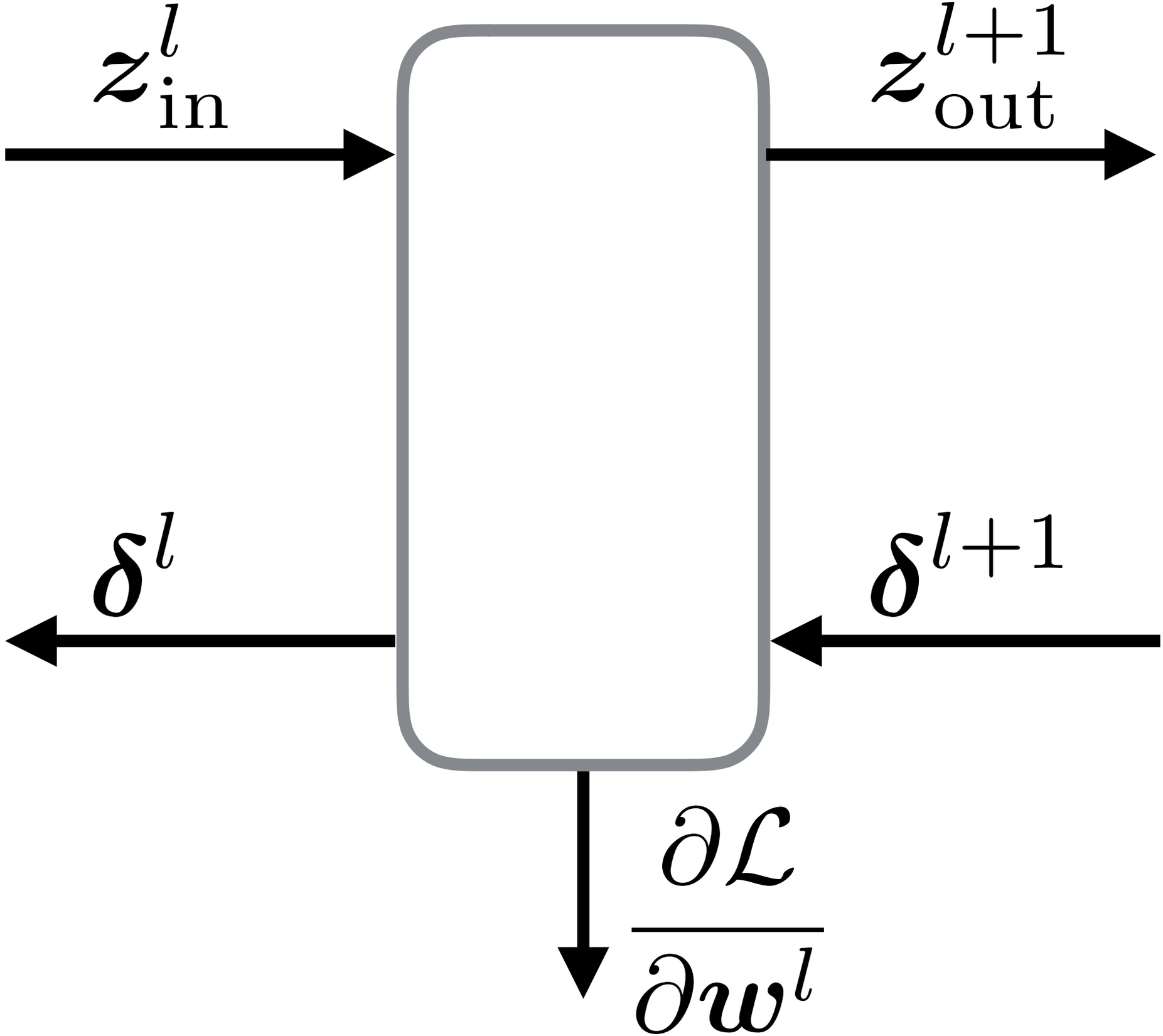

softmax
s_1
s_2
s_3
s_K
1
2
3
K
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_K$

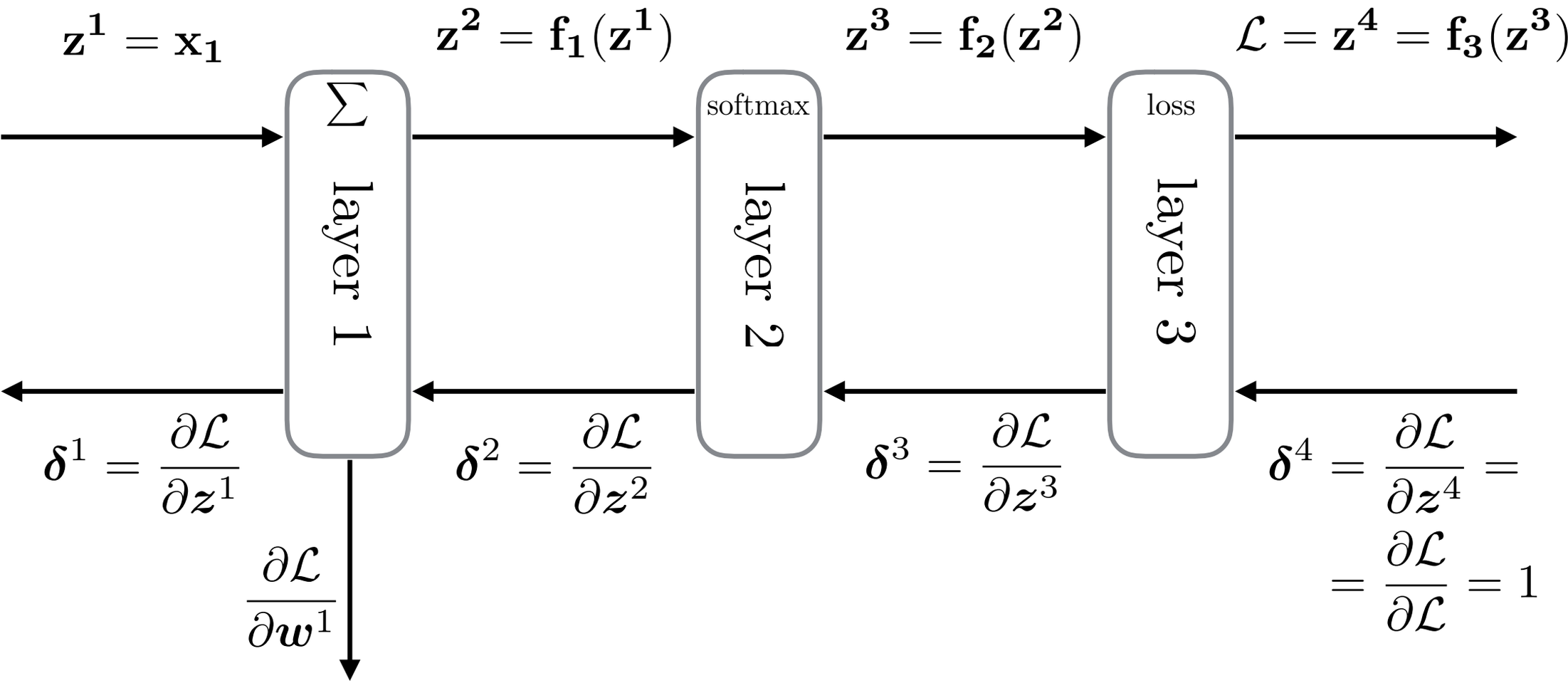

$\sum$
softmax
1
x_1
x_2
x_n
1
2
3
K
s_1
s_2
s_3
s_K
1
2
3
K
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_K$

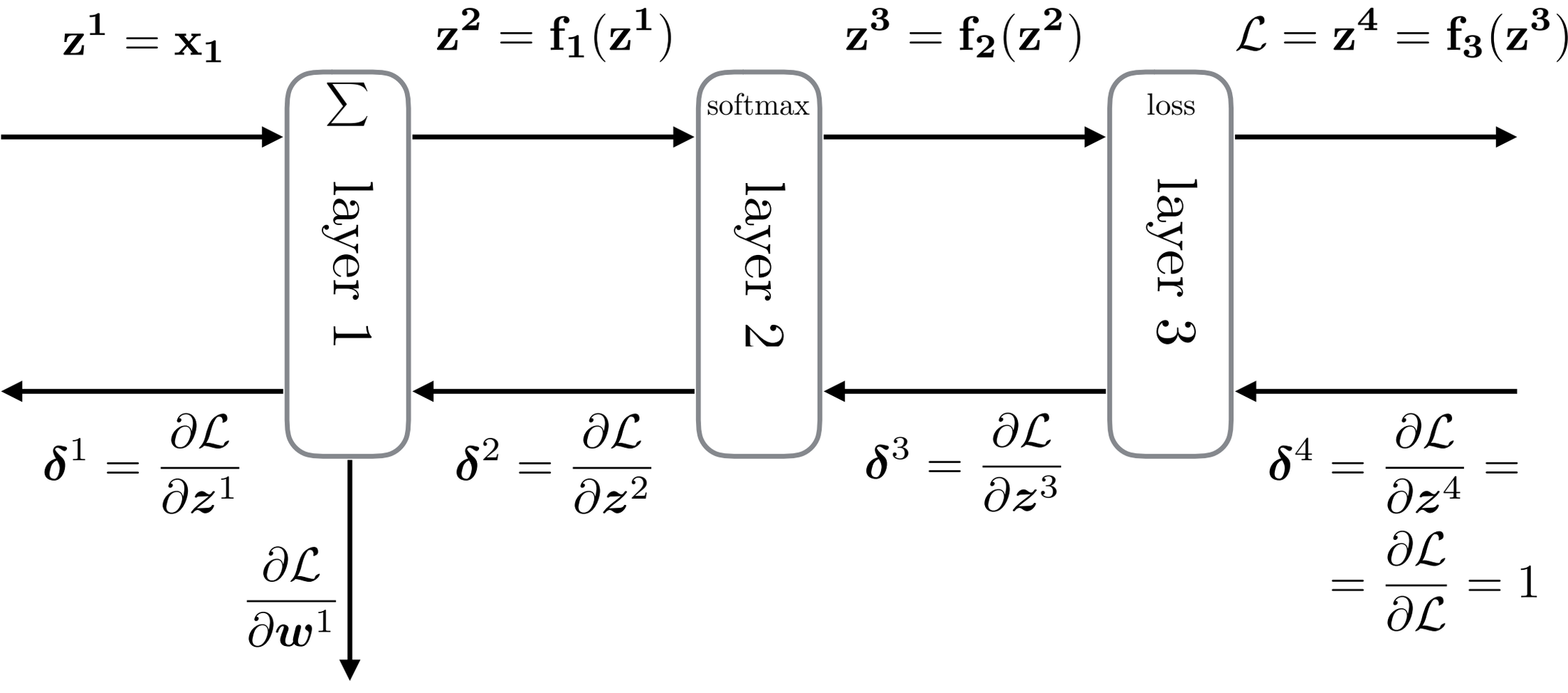

Sigmoid
$\sigma(s) = \frac{1}{1+e^{-s}}$
1
$\frac{1}{2}$
−10
10

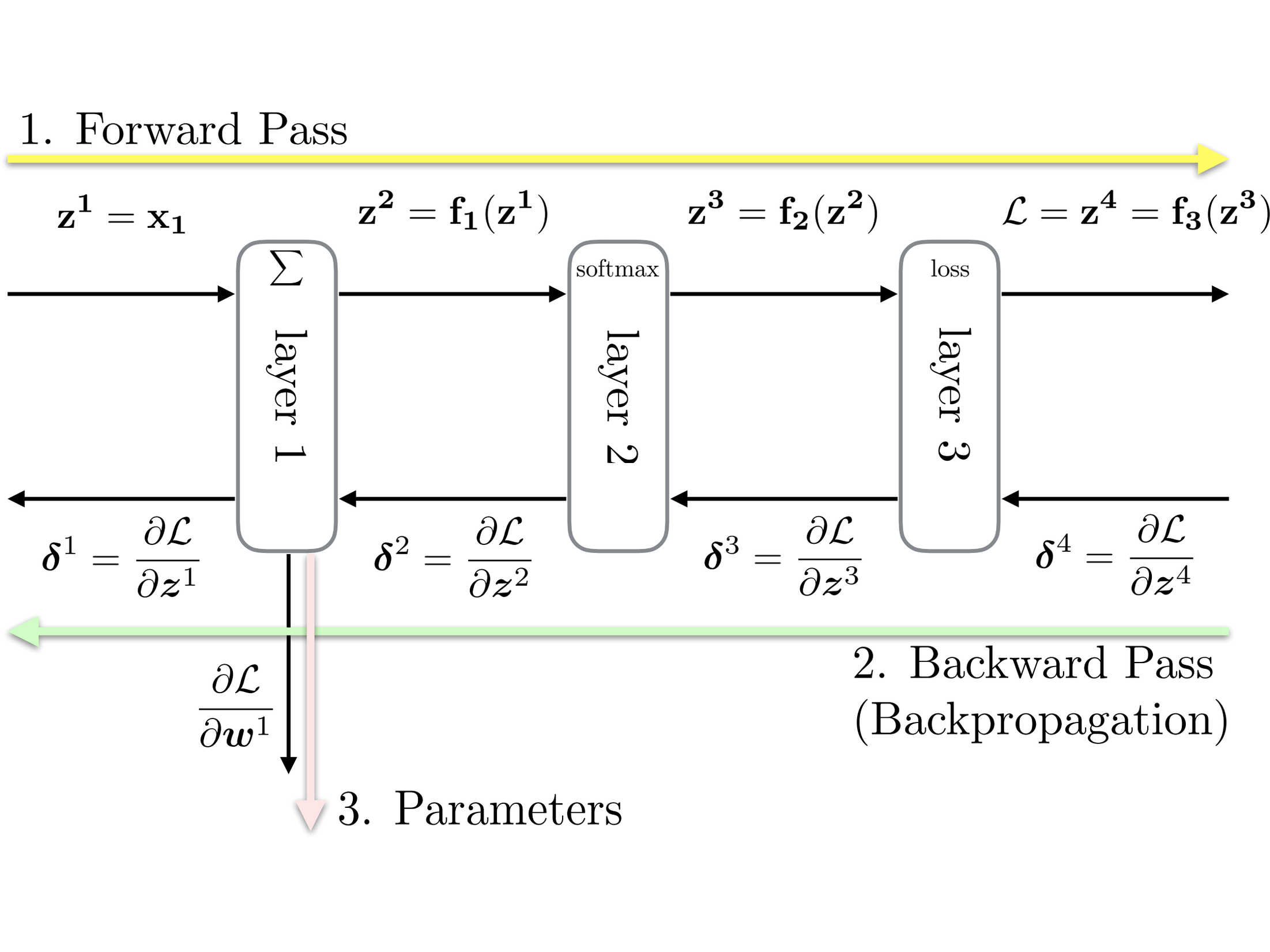

$\sum$
softmax
loss
1
x_1
x_2
x_n
1
2
3
K
s_1
s_2
s_3
s_K
1
2
3
K
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_K$
$\mathcal{L}$
$\mathbf{y}$

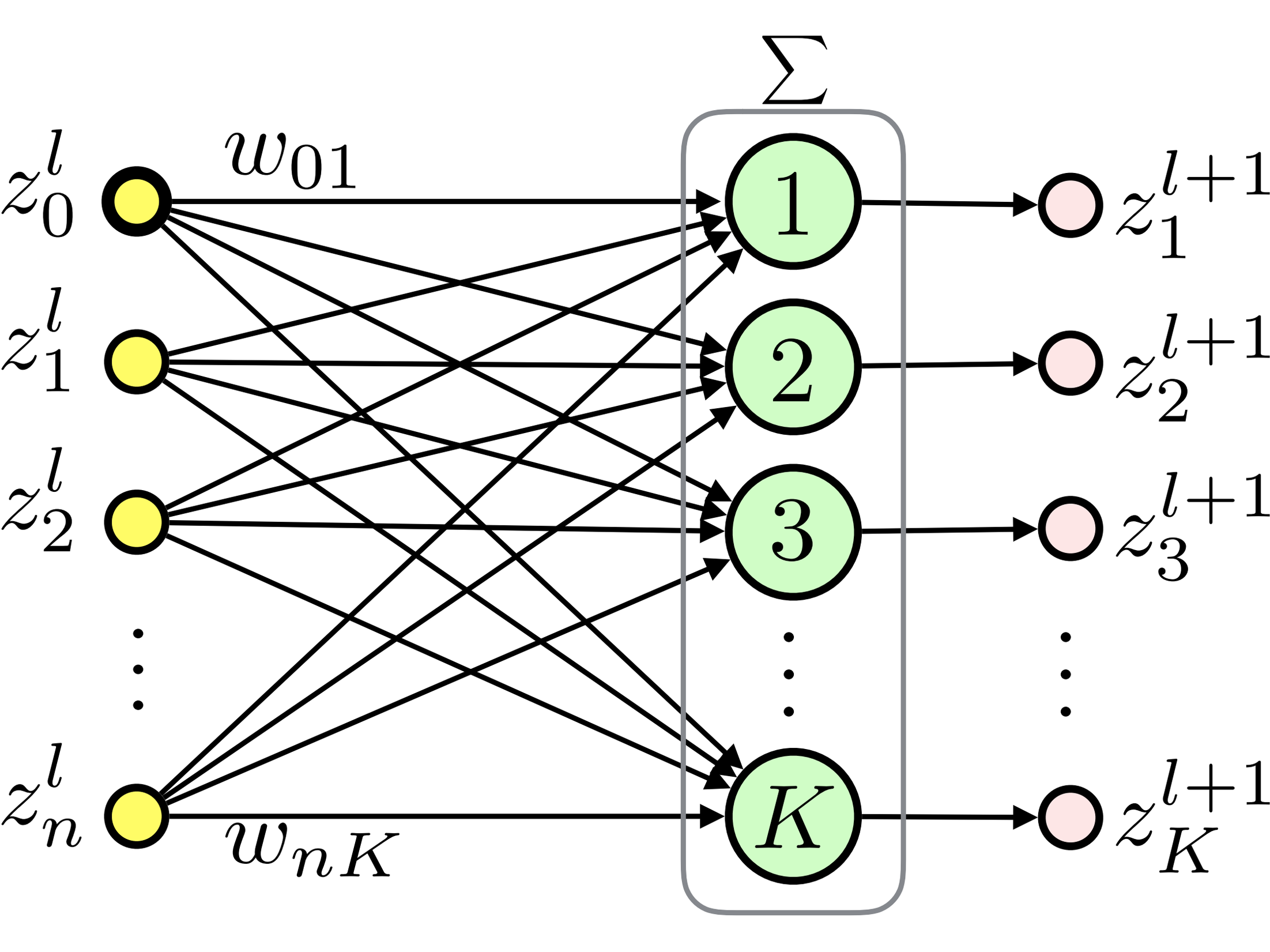

$\sum$
softmax
loss
1
x_1
x_2
x_n
1
2
3
K
s_1
s_2
s_3
s_K
1
2
3
K
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_K$
$\mathcal{L}$
$\mathbf{z^1 = x}$
$\mathbf{z^2 = s}$
$\mathbf{z^3 = \hat{y}}$
$\mathbf{y}$
$\mathbf{z^4 = \mathcal{L}}$

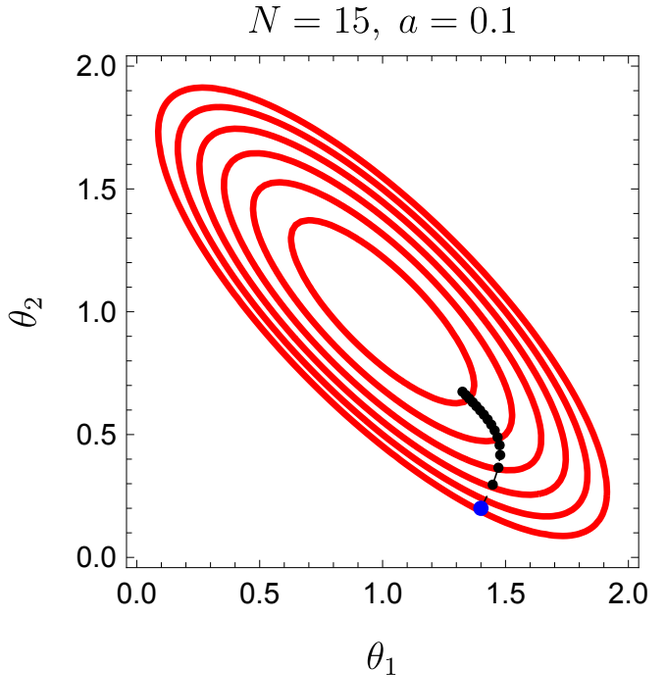

z_{in}^{l}
z_{out}^{l+1}
δ^{l}
δ^{l+1}
$\frac{\partial \mathcal{L}}{\partial w^{l}}$

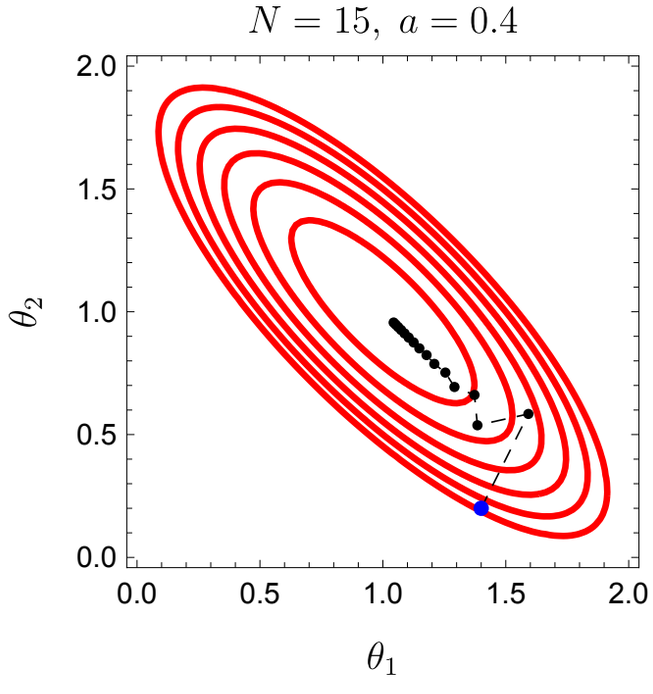

$\mathbf{z^1 = x_1}$
$\mathbf{z^2 = f_1(z^1)}$
$\mathbf{z^3 = f_2(z^2)}$
$\mathcal{L} = \mathbf{z^4 = f_3(z^3)}$
$\sum$ layer 1
softmax
layer 2
loss
layer 3
$\boldsymbol{\delta}^1 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^1}$
$\boldsymbol{\delta}^2 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^2}$
$\boldsymbol{\delta}^3 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^3}$
$\boldsymbol{\delta}^4 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^4} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1$
$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^1}$

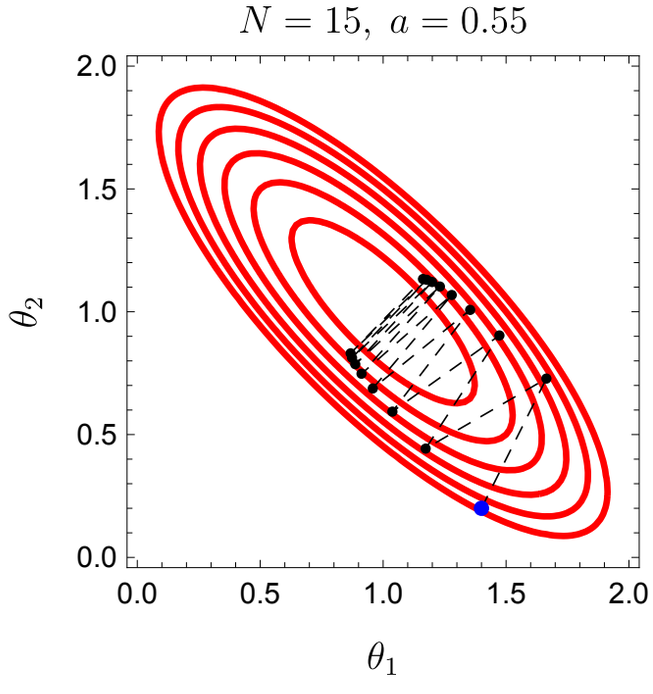

$\mathbf{z^1 = x_1}$
$\mathbf{z^2 = f_1(z^1)}$
$\mathbf{z^3 = f_2(z^2)}$
$\mathcal{L} = \mathbf{z^4 = f_3(z^3)}$
$\sum$ layer 1
softmax
layer 2
loss
layer 3
$\boldsymbol{\delta}^1 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^1}$
$\boldsymbol{\delta}^2 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^2}$
$\boldsymbol{\delta}^3 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^3}$
$\boldsymbol{\delta}^4 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^4} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1$
$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^1}$

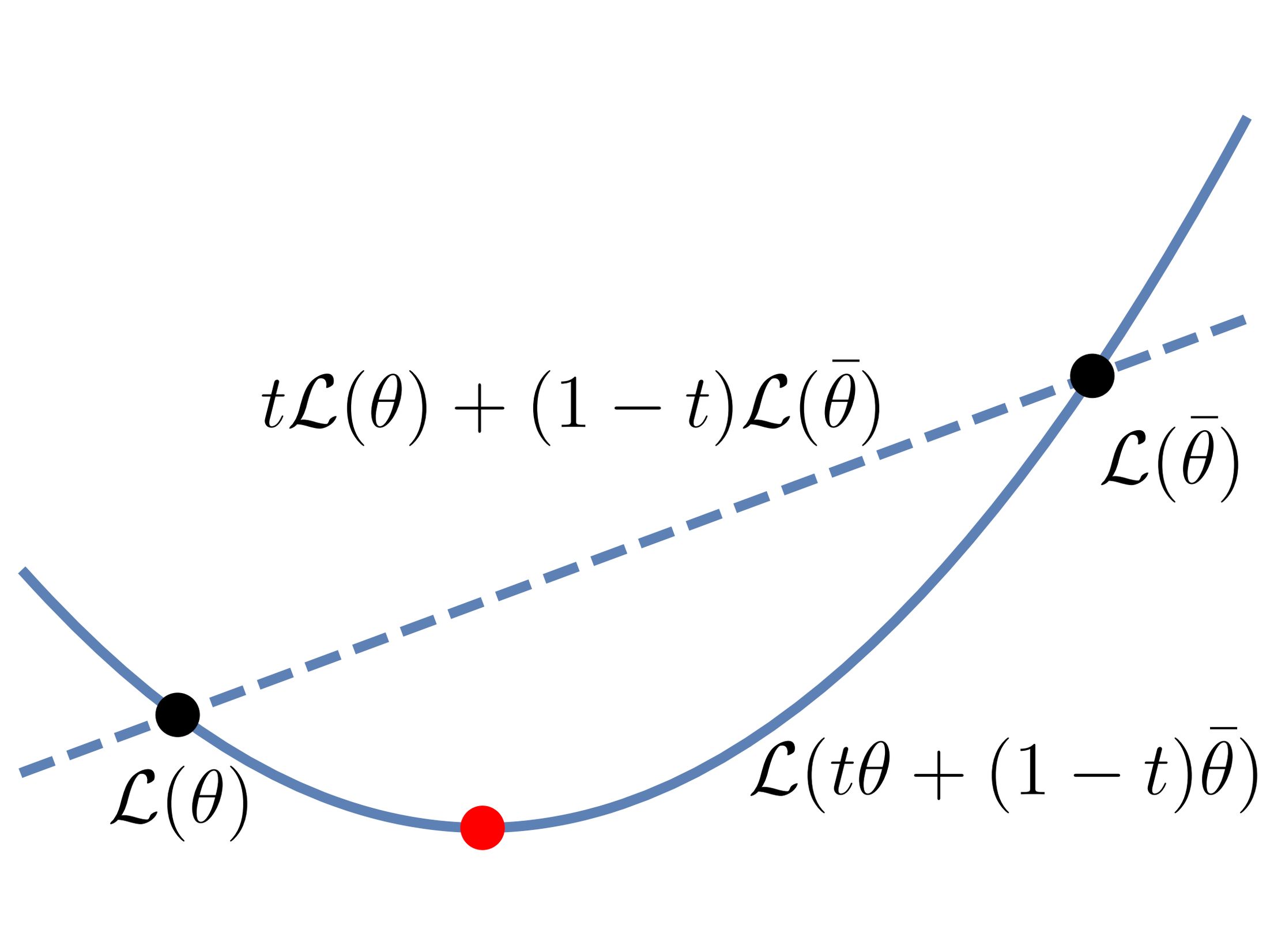

1. Forward Pass
$\mathbf{z^1 = x_1}$
$\mathbf{z^2 = f_1(z^1)}$
$\mathbf{z^3 = f_2(z^2)}$
$\mathcal{L} = \mathbf{z^4 = f_3(z^3)}$
$\sum$ layer 1
softmax
layer 2
loss
layer 3
$\boldsymbol{\delta}^1 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^1}$
$\boldsymbol{\delta}^2 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^2}$
$\boldsymbol{\delta}^3 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^3}$
$\boldsymbol{\delta}^4 = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^4}$
2. Backward Pass
(Backpropagation)
$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^1}$
3. Parameters

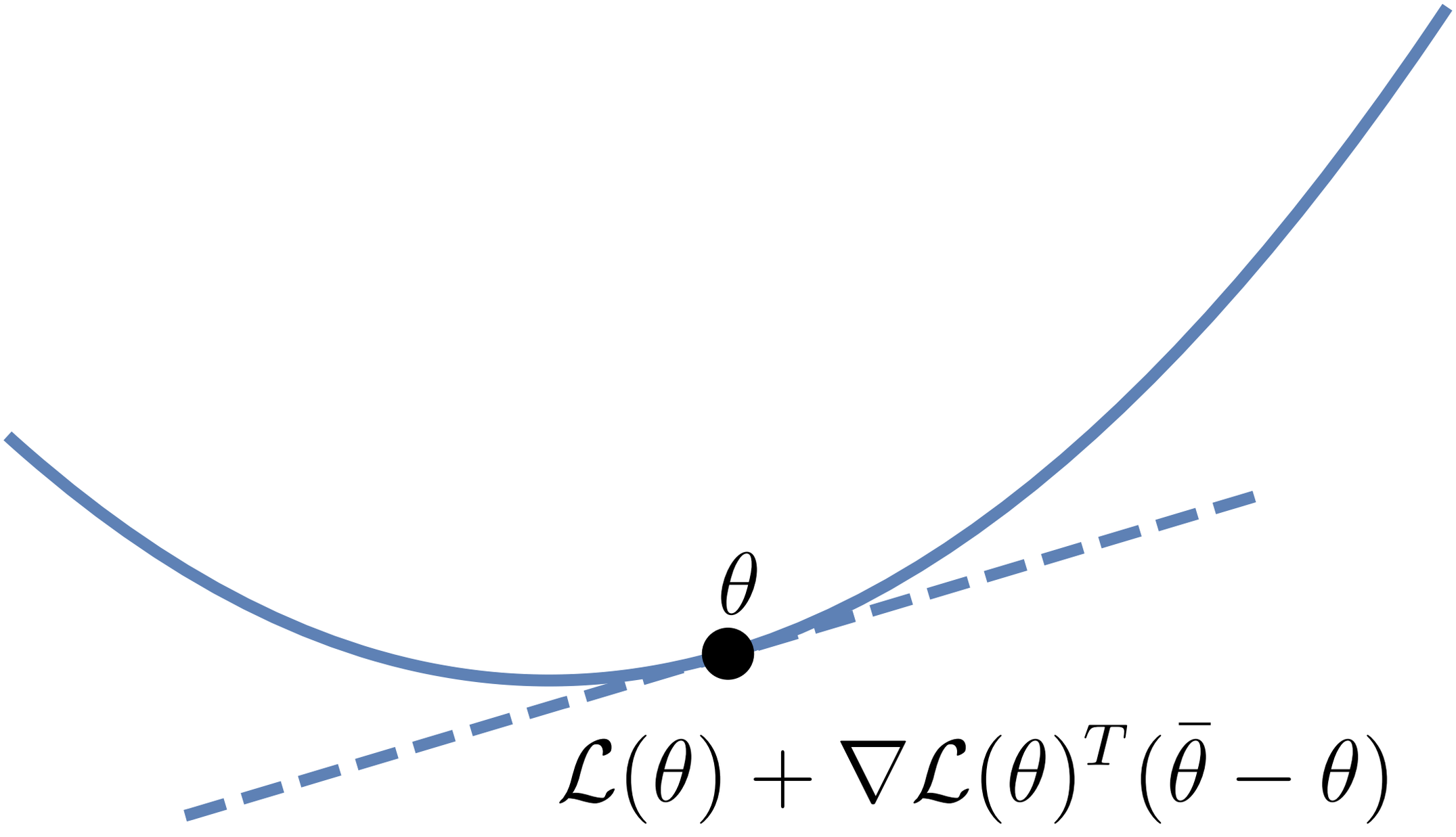

Σ
z_0^l
w_{01}
z_1^l
z_2^l
z_n^l
w_{nK}
1
2
3
K
z_1^{l+1}
z_2^{l+1}
z_3^{l+1}
z_K^{l+1}

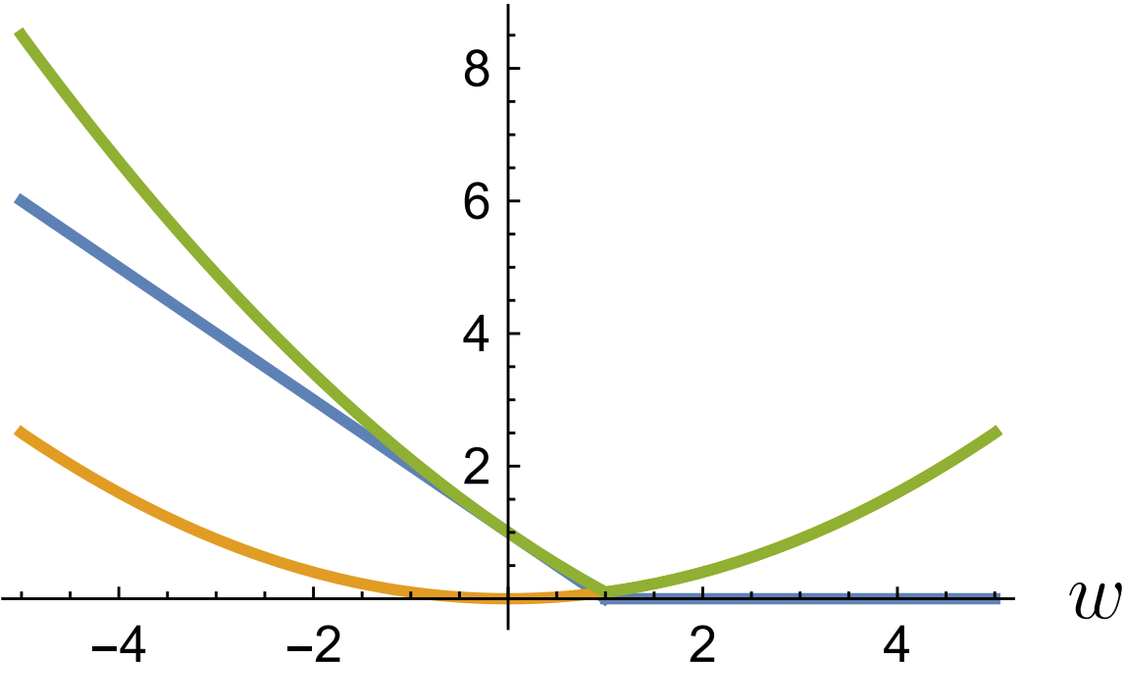

$N = 15,\ a = 0.1$
2.0
1.5
1.0
0.5
0.0
θ_2
0.0
0.5
1.0
1.5
2.0
θ_1


$N = 15,\ a = 0.4$
2.0
1.5
1.0
0.5
0.0
θ_2
0.0
0.5
1.0
1.5
2.0
θ_1


$N = 15,\ a = 0.55$
2.0
1.5
1.0
0.5
0.0
θ_2
0.0
0.5
1.0
1.5
2.0
θ_1


$t\mathcal{L}(\theta) + (1-t)\mathcal{L}(\bar{\theta})$
$\mathcal{L}(\bar{\theta})$
$\mathcal{L}(\theta)$
$\mathcal{L}(t\theta + (1-t)\bar{\theta})$

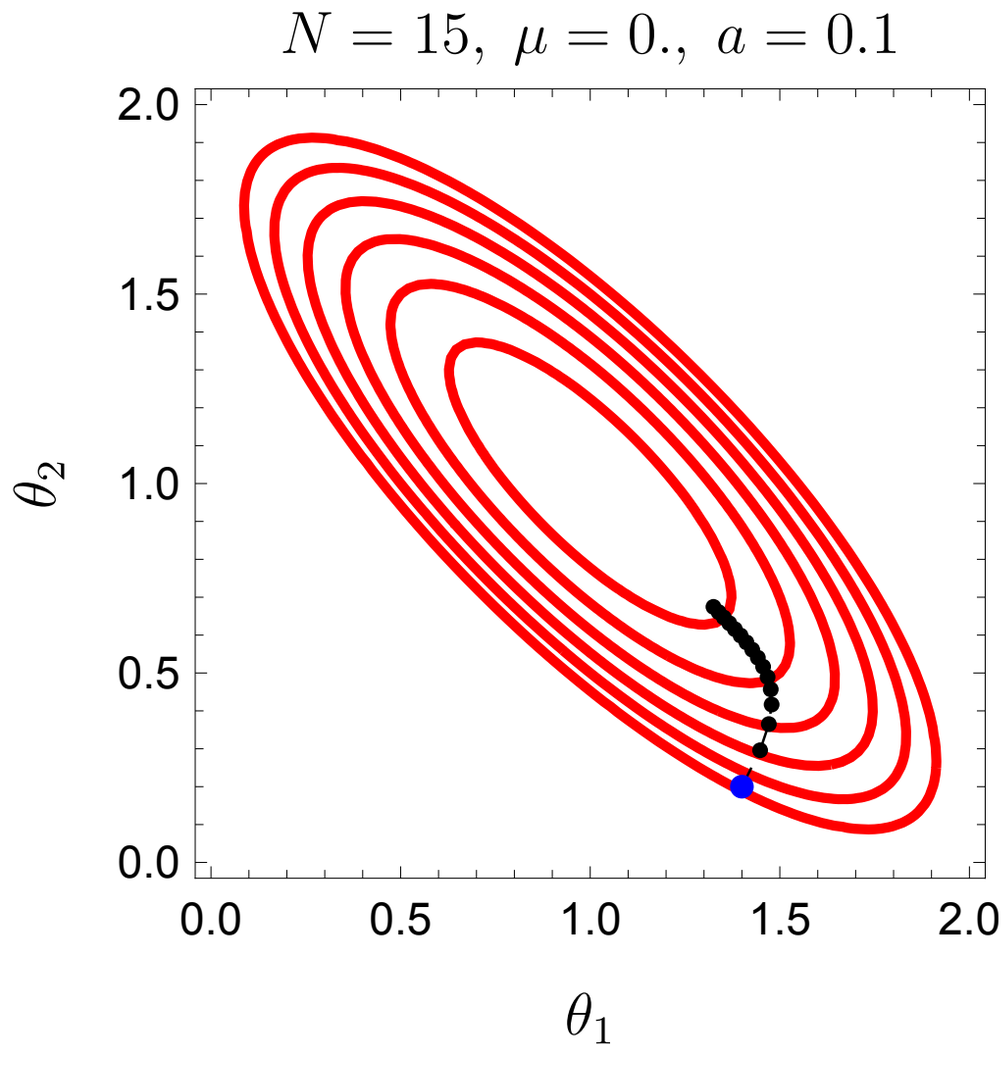

θ
$\mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^T(\bar{\theta} - \theta)$

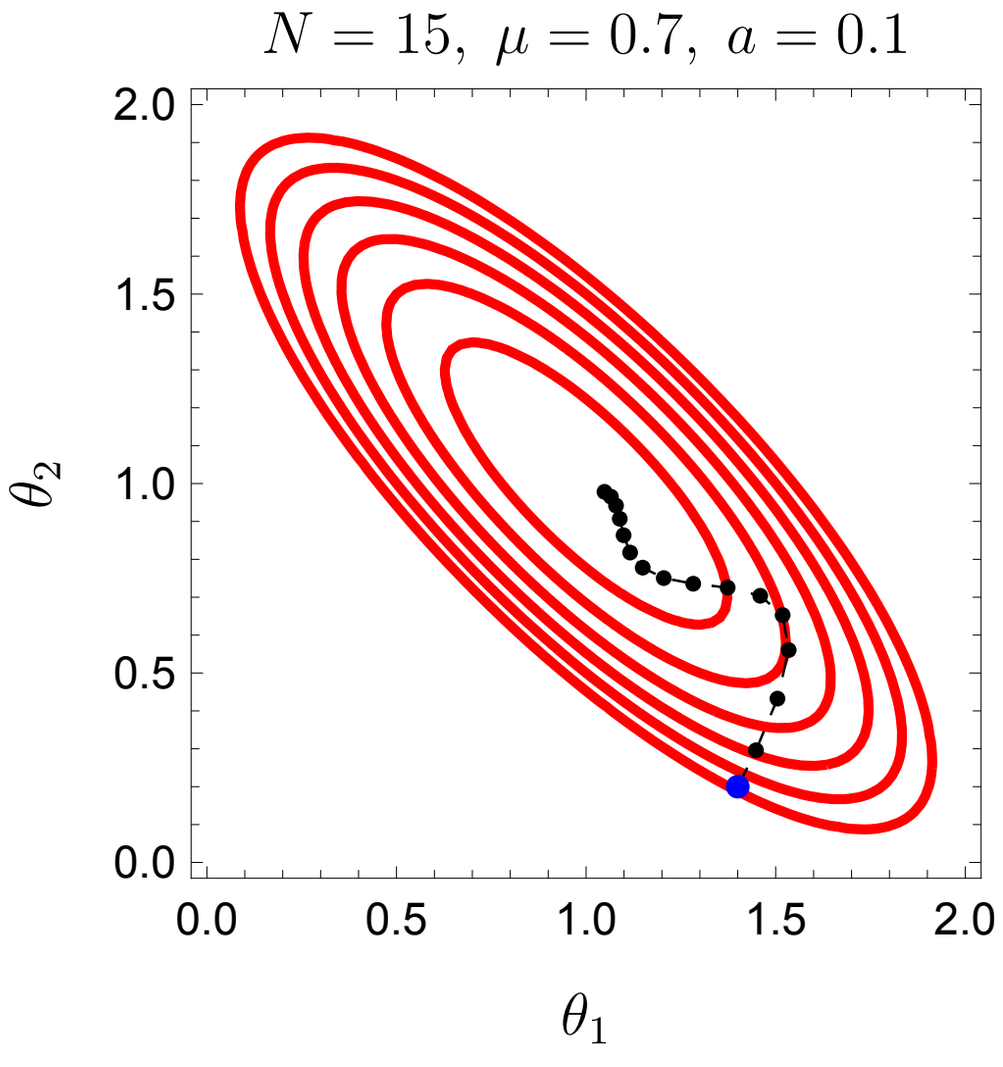


$\max(0, 1 - w)$

$0.1w^2$

$\max(0, 1 - w) + 0.1w^2$


$N = 15,\ \mu = 0.,\ a = 0.1$
2.0
1.5
1.0
0.5
0.0
θ_2
0.0
0.5
1.0
1.5
2.0
θ_1


$N = 15,\ \mu = 0.7,\ a = 0.1$
2.0
1.5
1.0
0.5
0.0
θ_2
0.0
0.5
1.0
1.5
2.0
θ_1