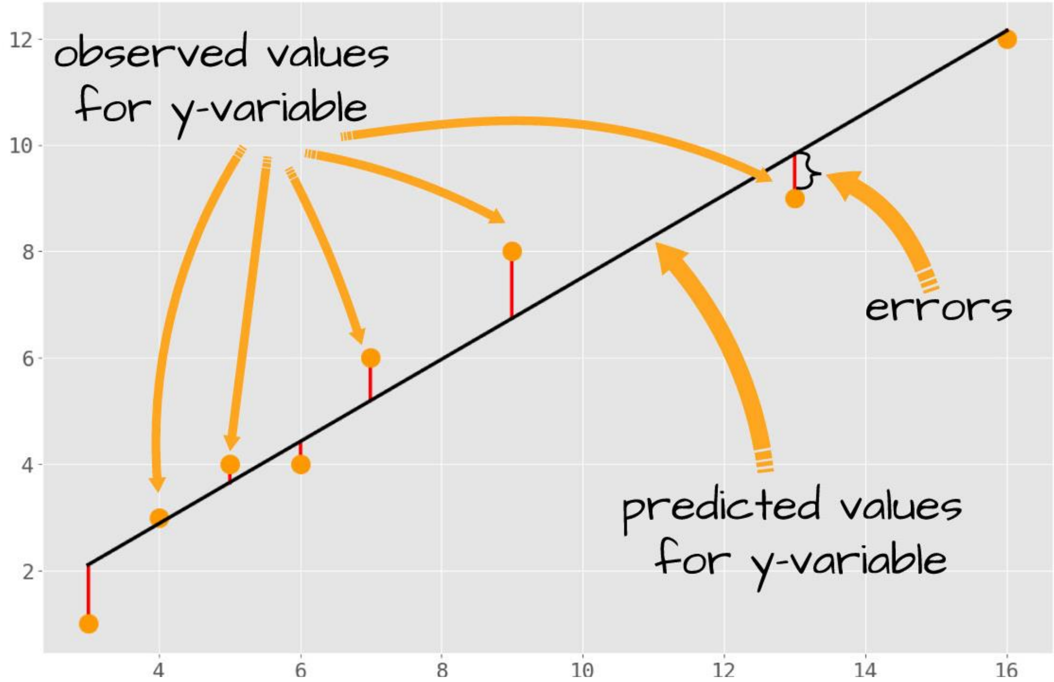
INTRODUCTION TO NEURAL NETWORKS

General questions about Supervised Machine Learning: Loss function



$$y = w^T x + b$$

$$\mathcal{L}(w, b) = \frac{1}{N} \sum_i (y_i - (w^T x_i + b))^2 = \frac{1}{N} \sum_i (y_i - \hat{y}_i)^2$$

Other Loss are possible

- Linear regression

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - (w^T x_i + w_0))^2$$

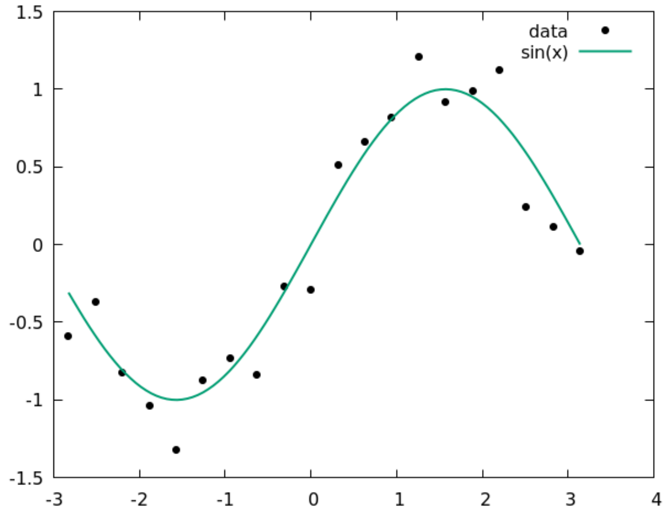
- Logistic regression

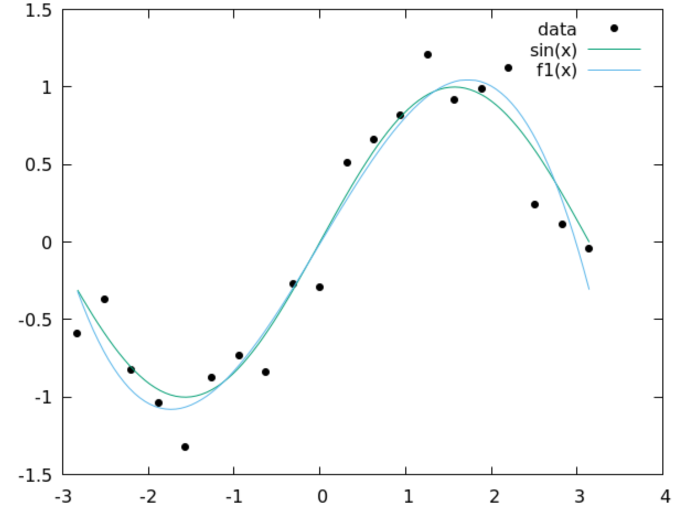
$$\frac{1}{n}\sum_{i=1}^{n}\log(1 + e^{-y_i(w^T G_i + w_0)})$$

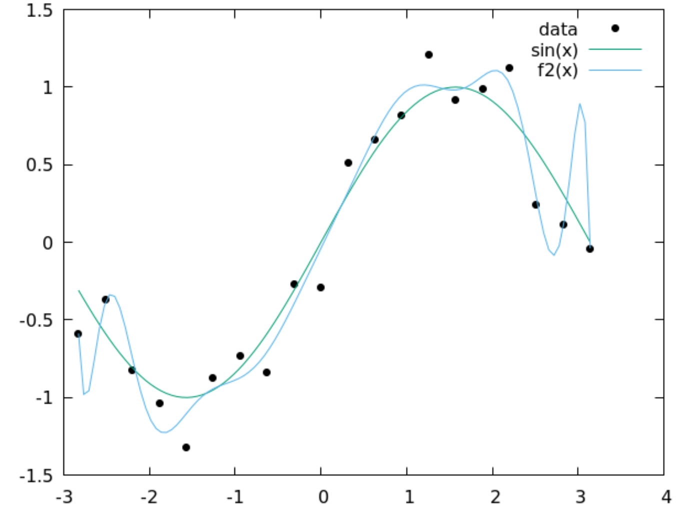
- SVM

$$\frac{1}{n}\sum_{i=1}^{n}\max(0, -y_i(w^T x_i + w_0))$$

General questions about Supervised Machine Learning: Overfit



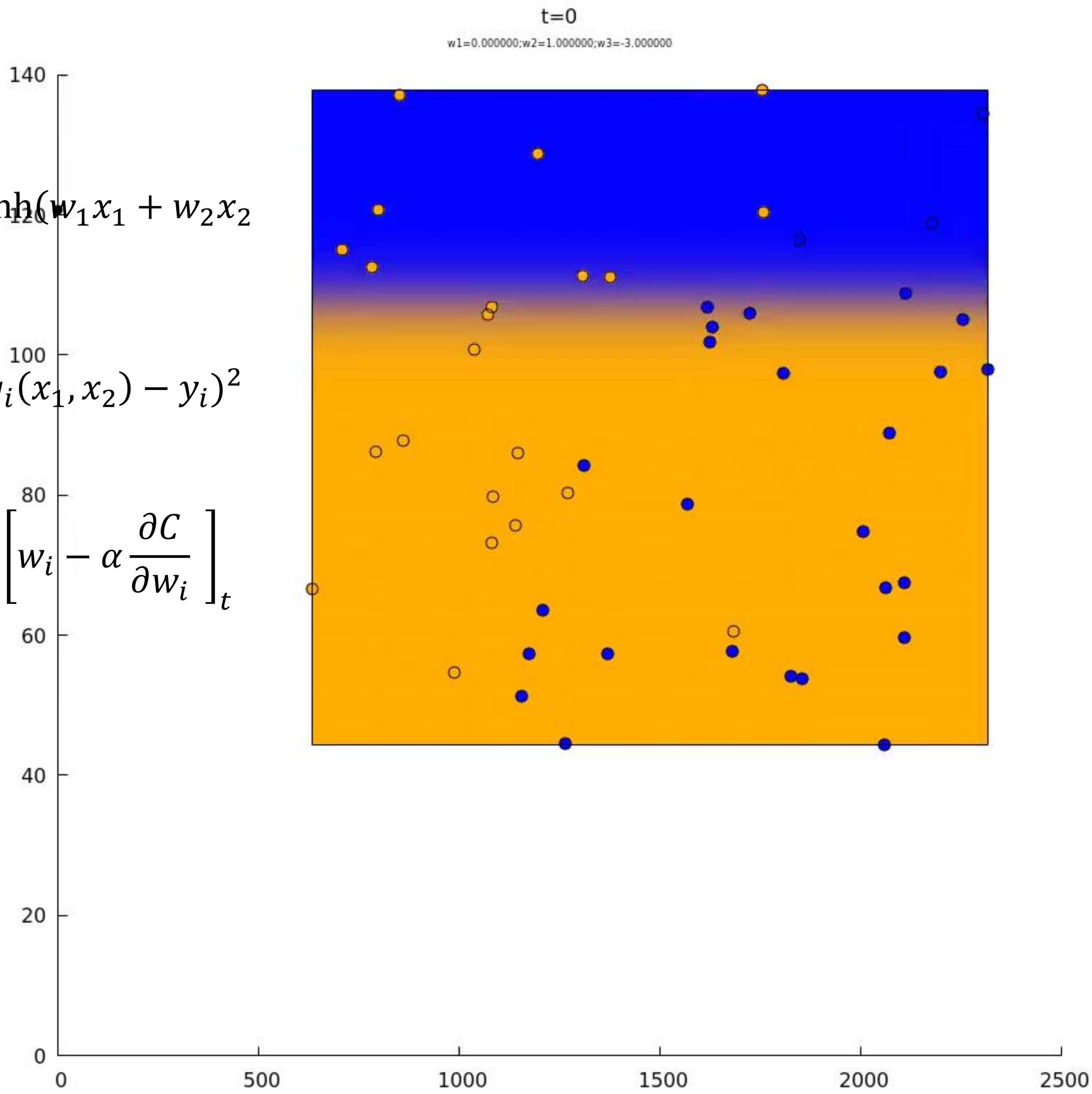




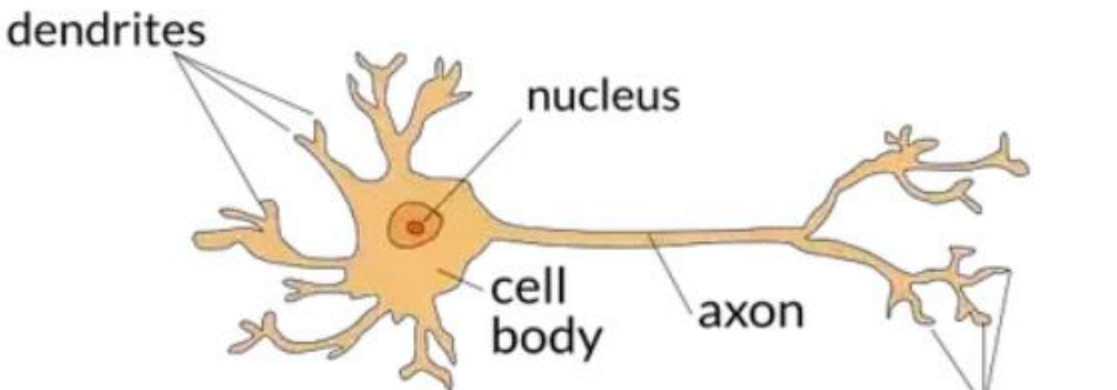
$$g(x_1, x_2) \tanh(w_1 x_1 + w_2 x_2 + w_3)$$

$$C = \sum_i (g_i(x_1, x_2) - y_i)^2$$

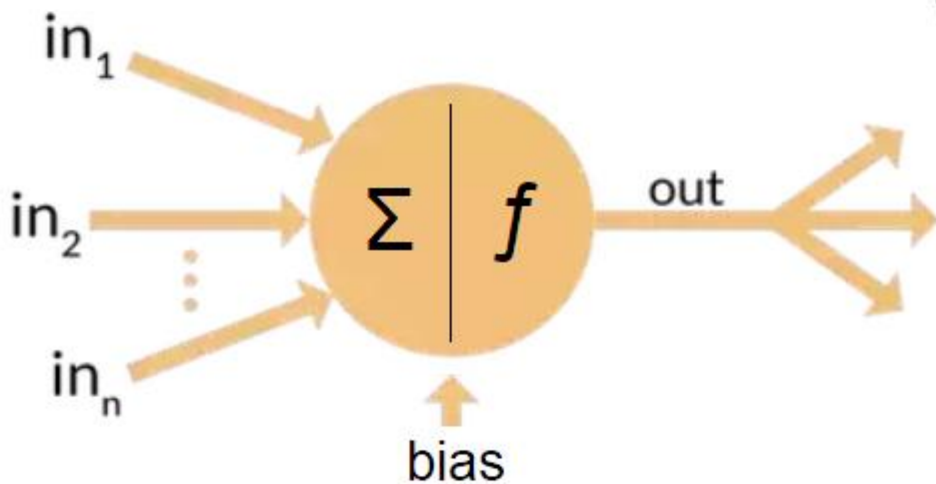
$$[w_i]_{t+1} = \left[w_i - \alpha \frac{\partial C}{\partial w_i} \right]_t$$



Natural vs Artificial Neurons



$$f_i(x_1, x_2) = \tanh(w_1 x_1 + w_2 x_2 + w_3)$$

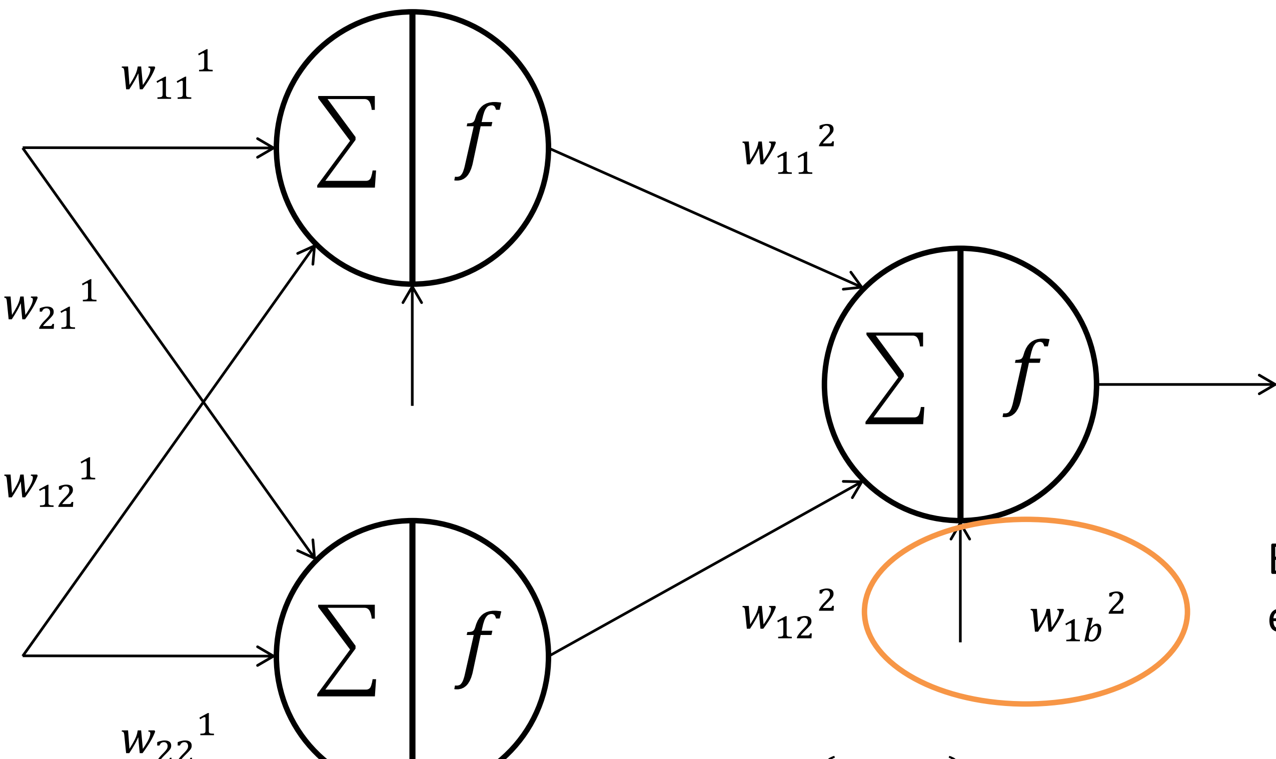


Only hyperplanar separations

Outline

- Perceptron
- Natural vs Artificial Neurons
- **Multilayer perceptron: Fully connected NN**
- Training:
  - Backpropagation
  - Stochastic Gradient Descent
  - Overfitting

The next step: Towards non-linear separations

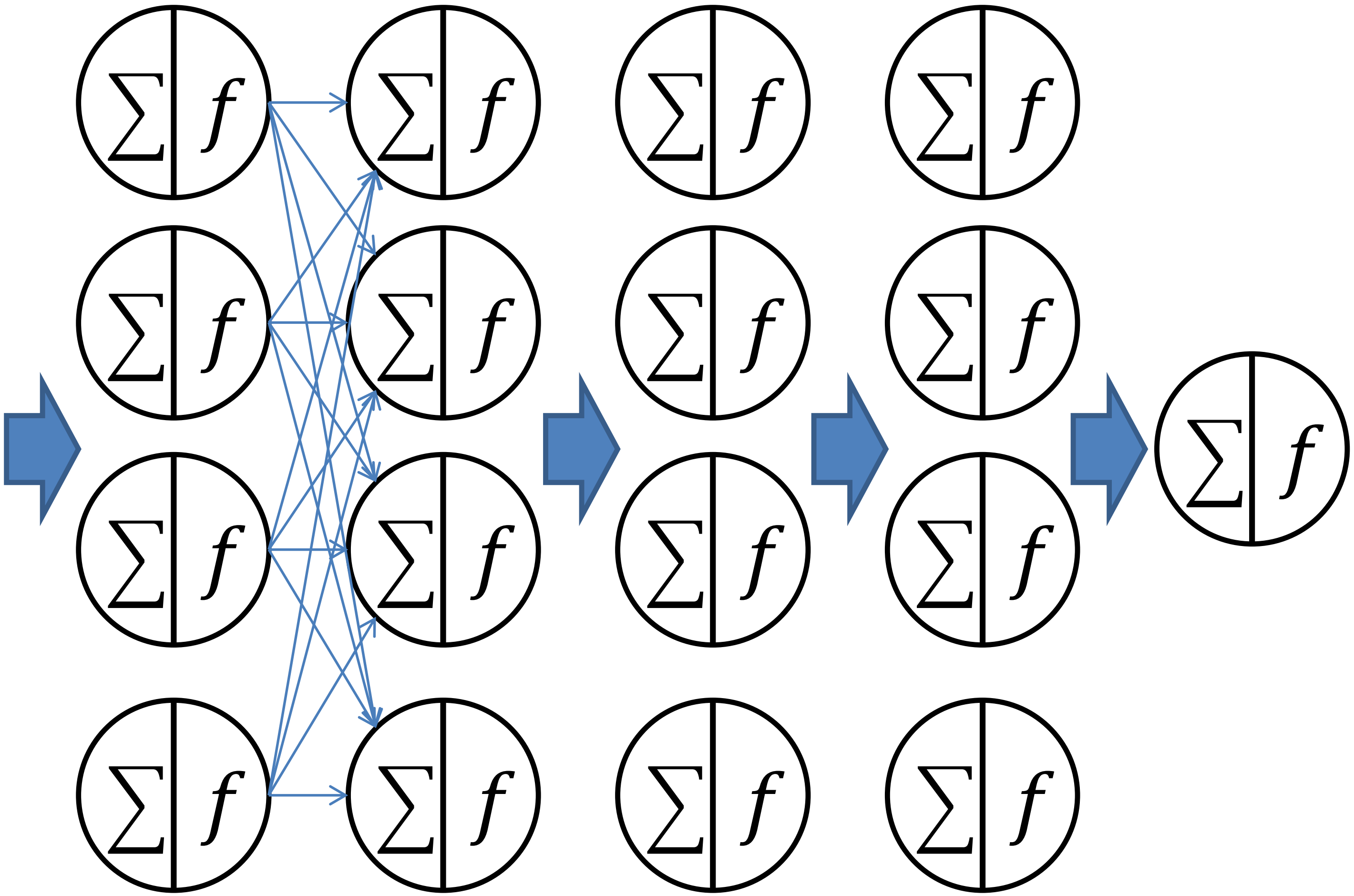


Bias term: Add an extra input =1

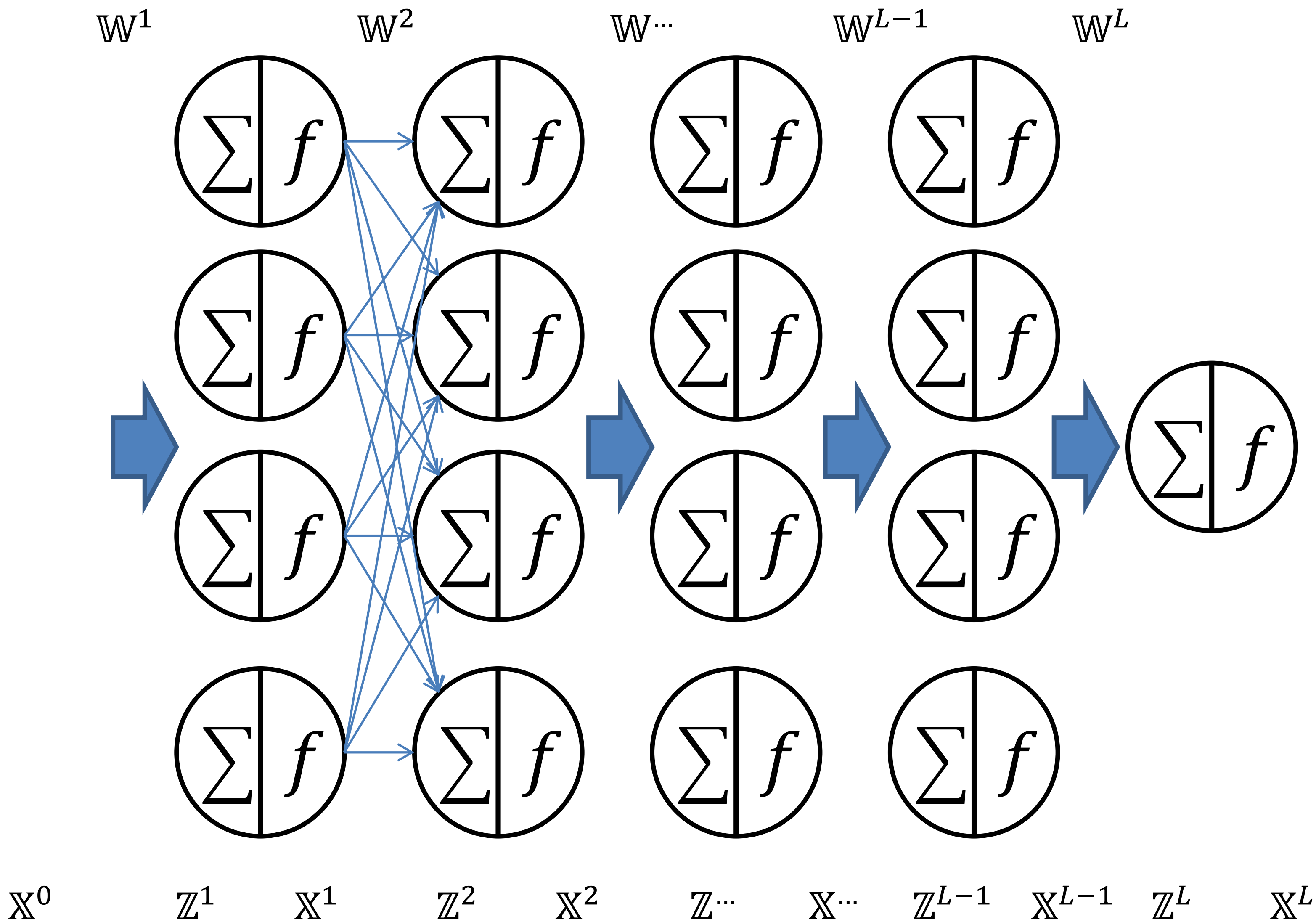
$$g(x_1, x_2) = \tanh(w_1 x_1 + w_2 x_2 + w_3)$$

$$g(x_1, x_2) = \tanh(w_{11}{}^2(\tanh(w_{11}{}^1 x_1 + w_{12}{}^1 x_2 + w_{1b}{}^1)\ ) + w_{12}{}^2(\tanh(w_{21}{}^1 x_1 + w_{22}{}^1 x_2 + w_{2b}{}^1)\ ) + w_{1b}{}^2)$$

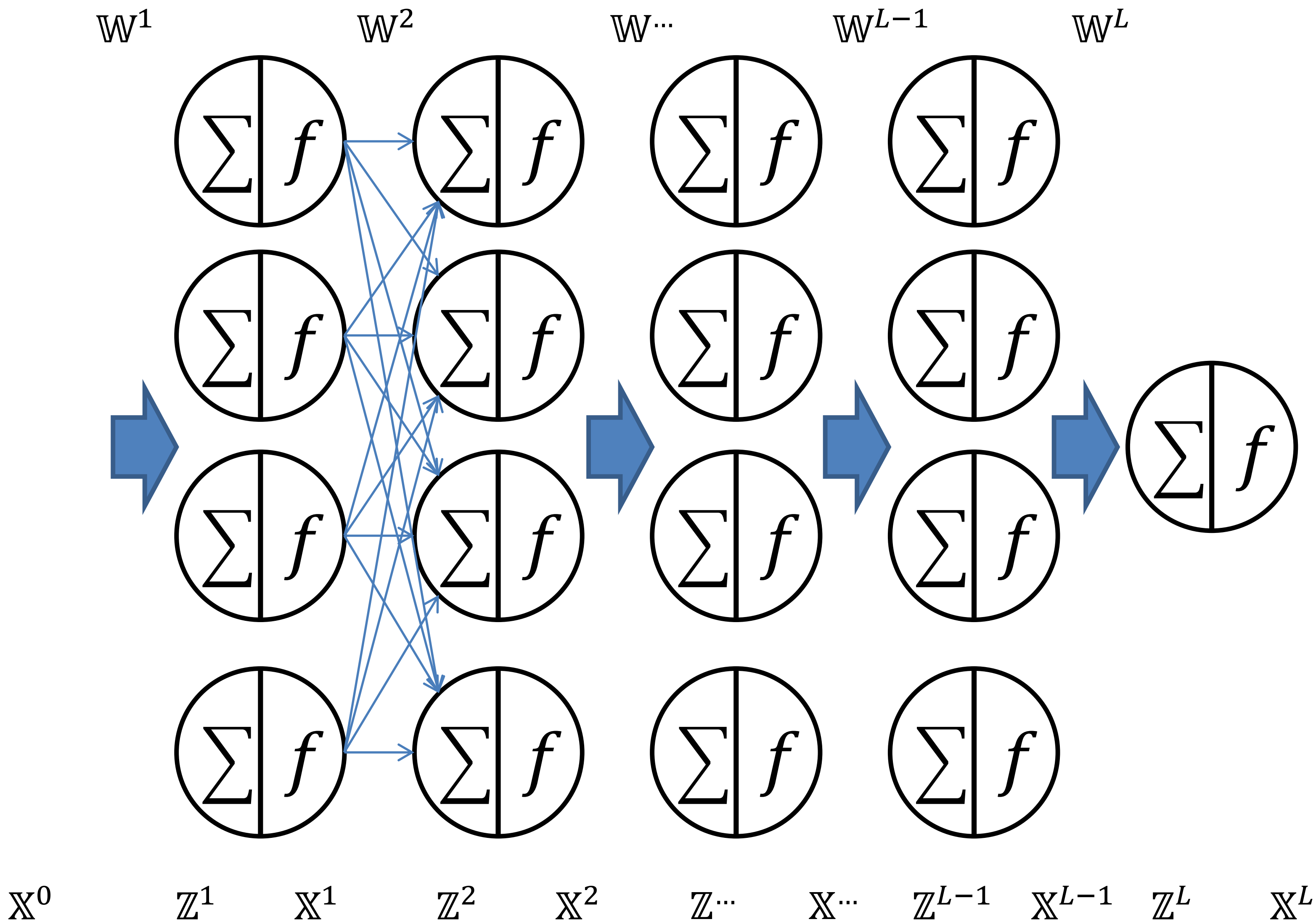
It can go messy really fast…



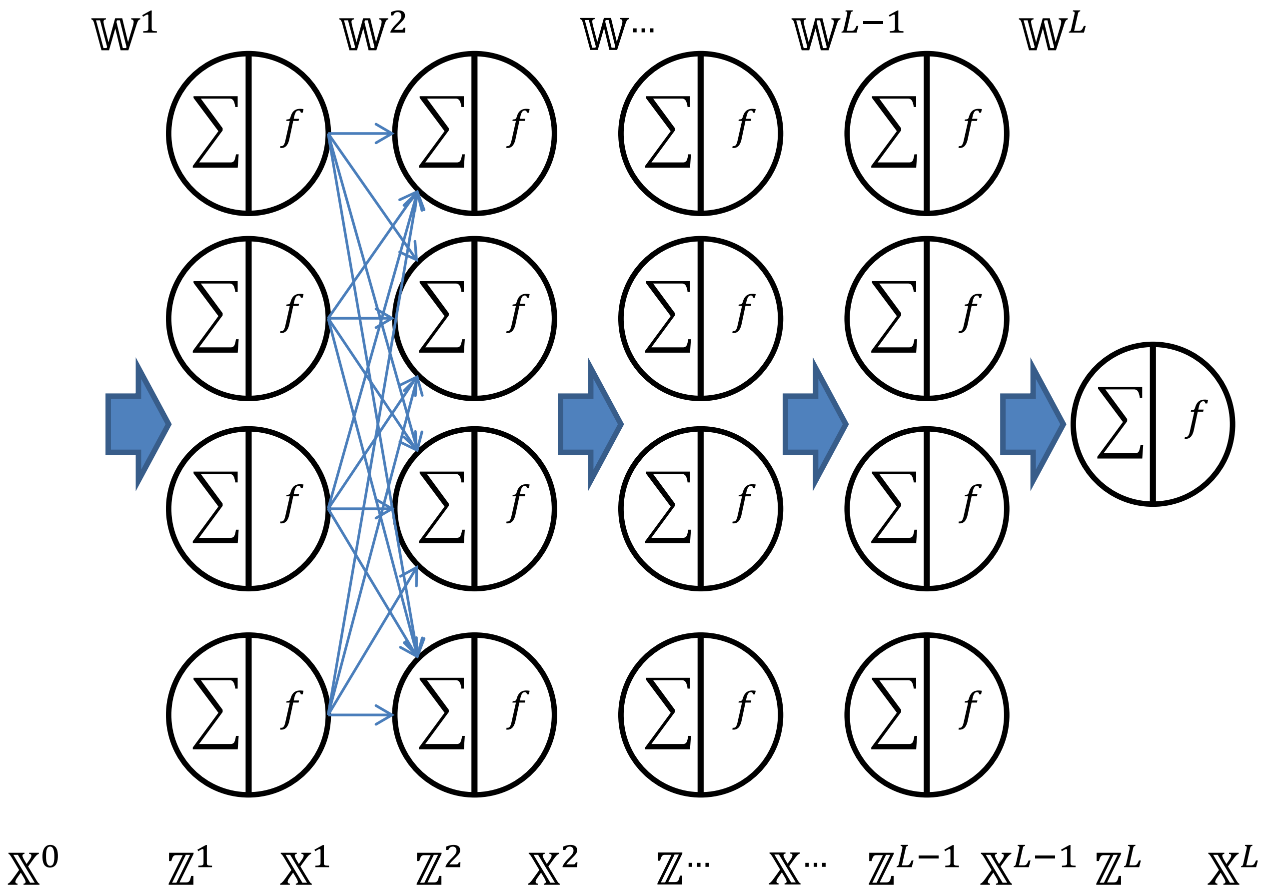
But God gave us MATRICES



But God gave us MATRICES



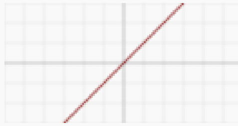
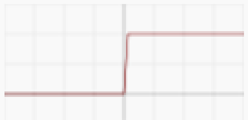
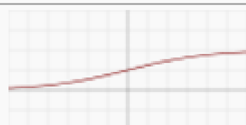
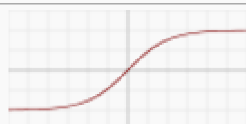
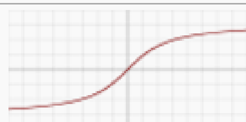
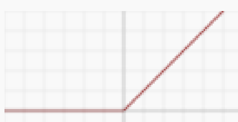
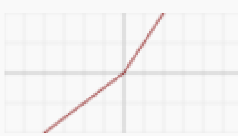
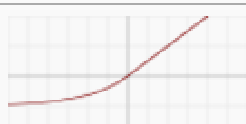
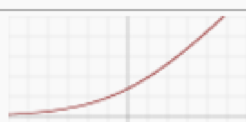
But God gave us MATRICES



$$
\begin{aligned}
g(\mathbb{X}^0) &= \mathbb{X}^L = f(\mathbb{Z}^L) = f(\mathbb{X}^{L-1}\mathbb{W}^L) = f(f(\mathbb{Z}^{L-1})\mathbb{W}^L) \\
&= f(f(\mathbb{X}^{L-2}\mathbb{W}^{L-1})\mathbb{W}^L) \ldots
\end{aligned}
$$

Two important concepts:

- Activation function
- Cost function/Loss function

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Two important concepts:

- Activation function
- Cost function/Loss function

$C = (\mathbb{X}^L - \mathbb{Y})^2 = (g(\mathbb{X}^0) - \mathbb{Y})^2$ Standard Loss for fitting

What happens with many class problems?

Softmax + Cross-Entropy

$$S_i = \frac{e^{x_i}}{\sum e^{x_i}}$$

$$H(p,q) = -\sum p_i(x)\log\big(q_i(x)\big)$$

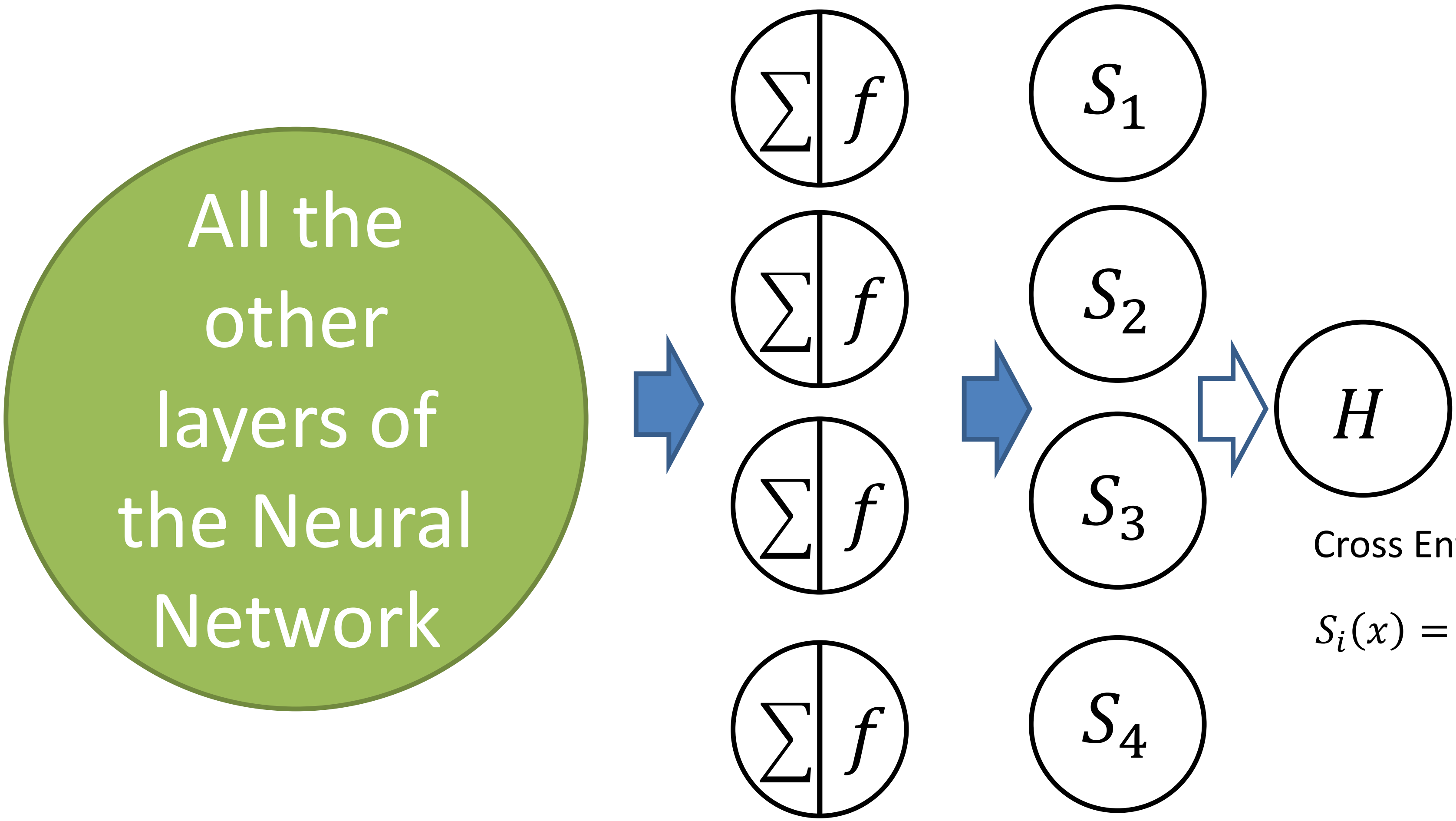
$p(x)$ is the true label in vector form

$$y(x_1) = 3 \Rightarrow p(x_1) = [0,0,1,0]$$

$$y(x_2) = 1 \Rightarrow p(x_2) = [1,0,0,0]$$

$q(x)$ is the estimated label (also in vector form)

4 class problem



Cross Entropy

$S_i(x) = q_i(x)$

Softmax layer $S_i = \frac{e^{x_i}}{\sum e^{x_i}}$

Outline

- Perceptron
- Natural vs Artificial Neurons
- Multilayer perceptron: Fully connected NN
- **Training:**
  - Backpropagation
  - Stochastic Gradient Descent
  - Overfitting

Backpropagation algorithm

$$\left[{w_{ij}}^L\right]_{t+1} = \left[{w_{ij}}^L - \alpha \frac{\partial C}{\partial {w_{ij}}^L}\right]_t$$

Steepest descent

α is known as Learning Rate

How to compute $\frac{\partial C}{\partial {w_{ij}}^L}$?

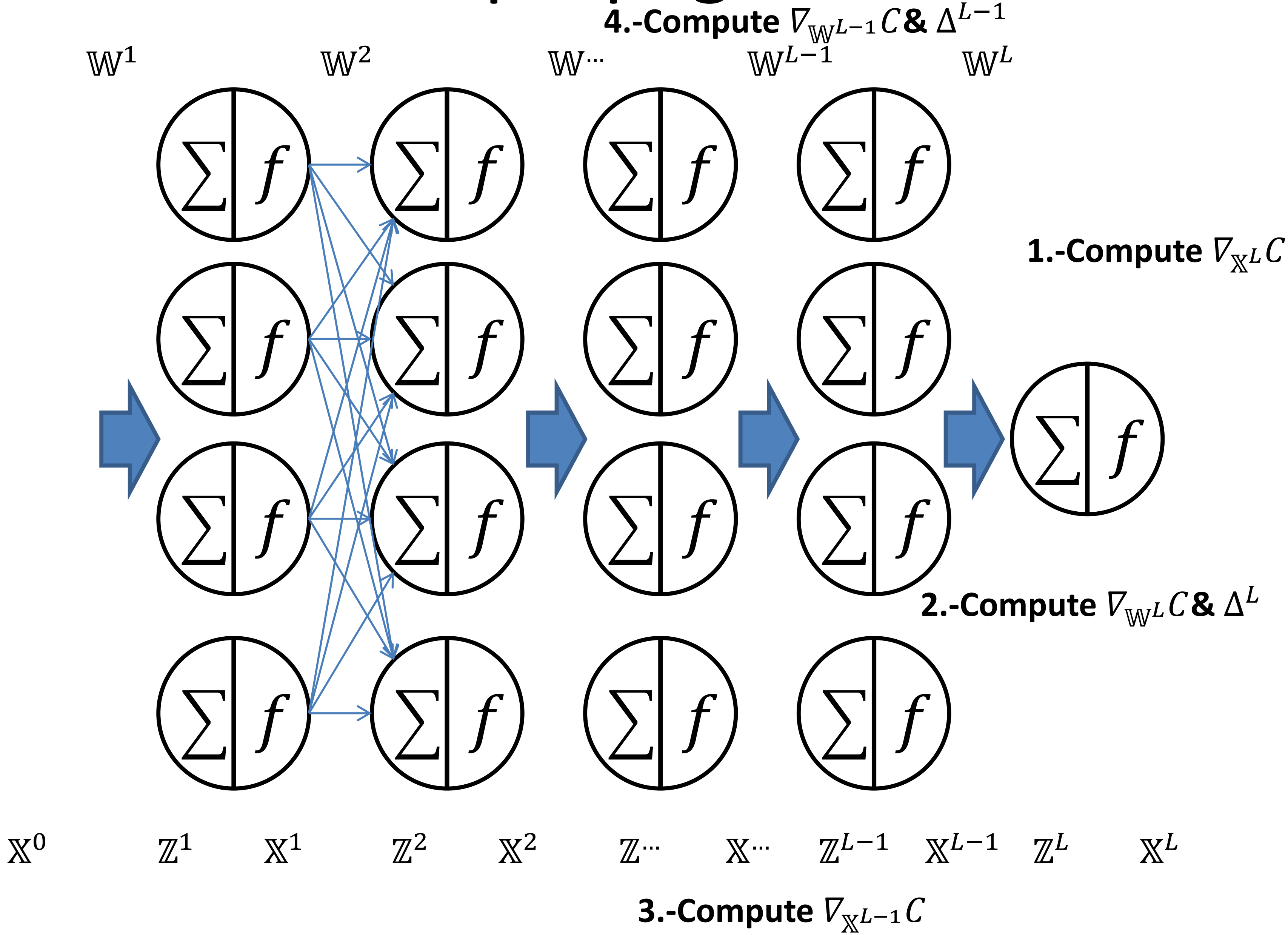
Chain rule

$$\nabla_{\mathbb{W}^L} C = (\mathbb{X}^{L+1})^T \cdot \Delta^{L+1}$$

$$\Delta^{L+1} = \nabla_{\mathbb{X}^{L+1}} C \odot f'^{L+1}(\mathbb{Z}^{L+1})$$

$$\nabla_{\mathbb{X}^L} C = \Delta^{L+1} \cdot (\mathbb{W}^{L+1})^T$$

Backpropagation



Stochastic Gradient descent

Gradient Descent

t=0

While not converged:

    t=t+1 *Epochs*

    $\nabla_{\mathbb{W}^L} C$=0

    for i in N:

        accumulate ∇

    weight update*

Stochastic Gradient descent

t=0

While not converged:

    t=t+1

    shuffle data

    for j in <u>minibatches</u>:

        $\nabla_{\mathbb{W}^L} C$=0

        for i in N/m:

            accumulate ∇

        weight update*

$$*\left[{w_{ij}}^L\right]_{t+1} = \left[{w_{ij}}^L - \alpha \frac{\partial C}{\partial {w_{ij}}^L}\right]_t$$

Overfitting

- SGD helps to avoid overfitting
- Early stopping (dividing our system).
- Data augmentation (add transformed input data points).
- We can use regularization: Add a term to the Loss function that depends on the norm of the weights in order to obtain the minimum possible number of significant parameters
- Dropout (one layer in which randomly some weights are set to 0).