

Statistical Machine Learning Theory

Neural Networks

Hisashi Kashima
kashima@i.Kyoto-u.ac.jp

Neural networks:

Foundation of deep learning

- Linear models to non-linear models
 - “Classic” non-linear models:
 - Decision trees, kernel machines, boosting, ...
- Neural networks
 - Inherently nonlinear models
 - Foundation of the successful “deep learning”
- To see what they are and how to train them, we will start from reviewing the logistic regression model



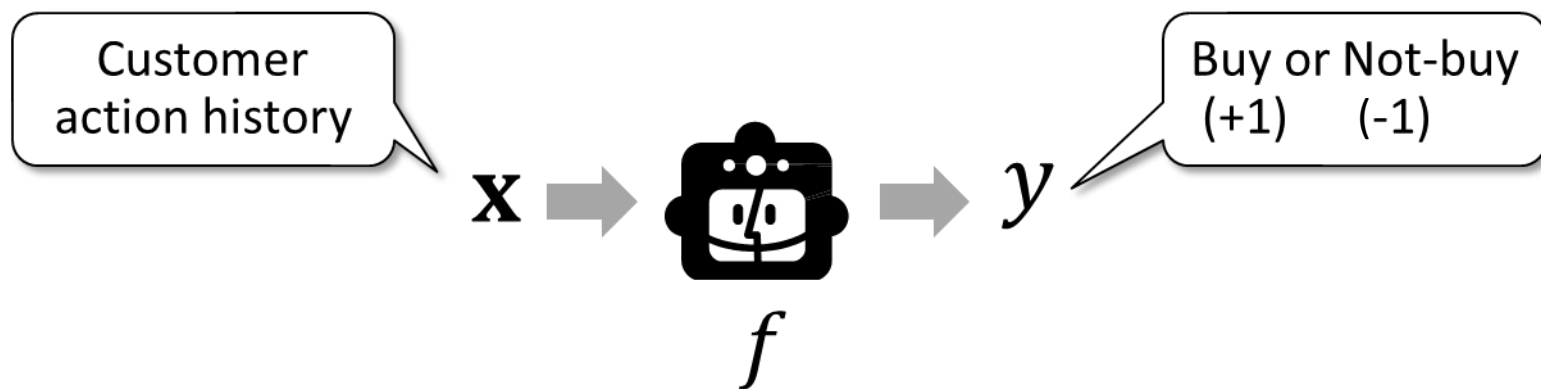
Logistic Regression



Classification:

Supervised learning for predicting discrete variable

- Goal: Obtain a function $f: \mathcal{X} \rightarrow \mathcal{Y}$
 - $\mathcal{X} = \mathbb{R}^D$: Input domain
 - \mathcal{Y} : discrete output domain
 - ◆ We focus on **two-class classification**: $\mathcal{Y} = \{+1, -1\}$
- Training dataset: N pairs of an input and an output $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$





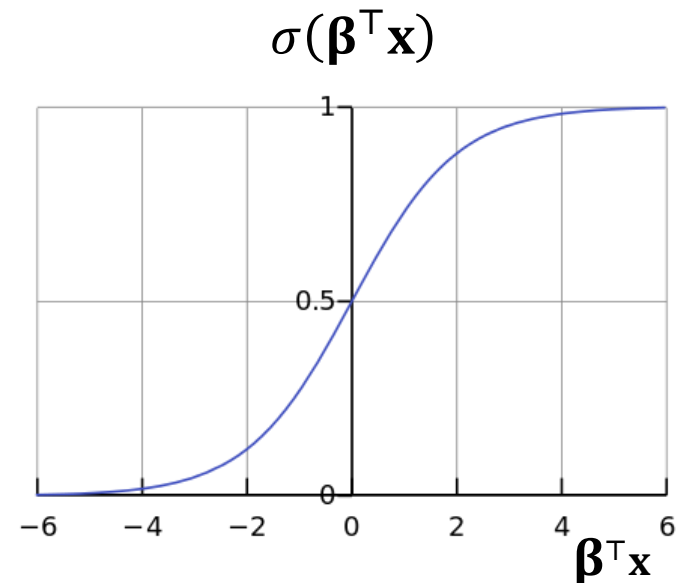
Logistic regression:

A probabilistic model for binary classification

- Logistic regression model gives the conditional probability

$$f_{\mathbf{w}}(y = +1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

- Logistic (sigmoid) function $\sigma: \mathbb{R} \rightarrow (0,1)$
 - $\mathbf{w}^T \mathbf{x}$ has the same form as linear regression and takes a real number
 - σ converts real numbers to “probabilities”



Cross entropy:

Objective function to train a logistic regression model

- Cross entropy (to minimize):

$$\begin{aligned}
 L(\mathbf{w}) &= - \sum_{i=1}^N \delta(y^{(i)} = +1) \log f_{\mathbf{w}}(y^{(i)} = +1 | \mathbf{x}^{(i)}) \\
 &\quad - \sum_{i=1}^N \delta(y^{(i)} = -1) \log(1 - f_{\mathbf{w}}(y^{(i)} = +1 | \mathbf{x}^{(i)})) \\
 &= \sum_{i=1}^N \log(1 + \exp(-y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)}))
 \end{aligned}$$

For positive class data

For negative class data

- Cross entropy is equivalent to
 - ◆ Logistic loss: upper bound of 0-1 loss (#mistakes)
 - ◆ Negative log-likelihood (for maximum likelihood estimation)

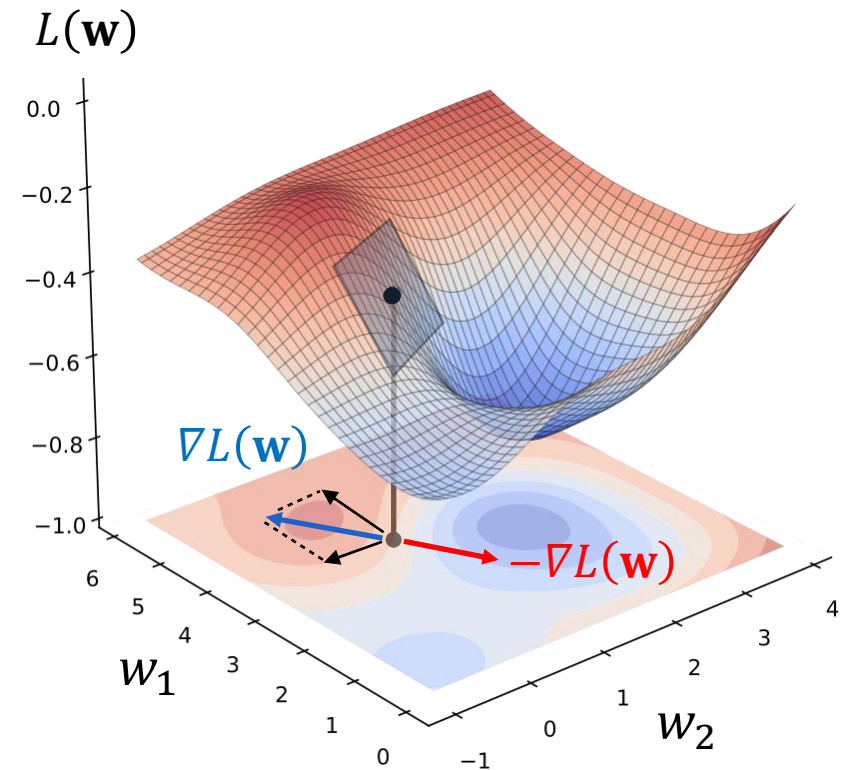
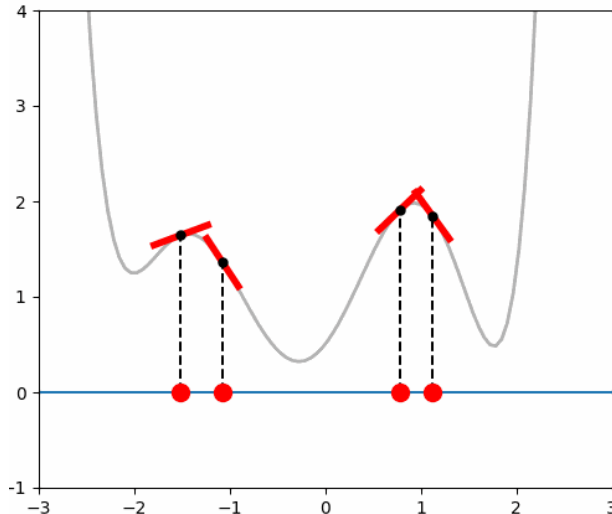
Gradient descent:

A simplest parameter optimization method

- Iteratively refine the current parameter \mathbf{w} to \mathbf{w}^{NEW} :

$$\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$$

- Gradient $\nabla L(\mathbf{w})$ is the steepest direction of the objective function $L(\mathbf{w})$
- “learning rate” $\eta > 0$





Gradient descent for logistic regression:

Gradient of objective function is all you need

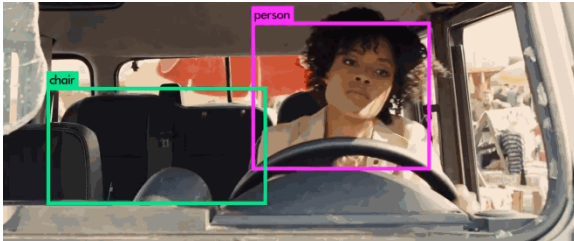
- Once we obtain the gradient, we can apply GD
 - Obj. func.: $L(\mathbf{w}) = \sum_{i=1}^n \ln(1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}))$
 - Gradient: $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{y^{(i)} \mathbf{x}^{(i)}}{1 + \exp(\mathbf{w}^\top \mathbf{x}^{(i)})}$
 - GD update: $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \sum_{i=1}^n \frac{y^{(i)} \mathbf{x}^{(i)}}{1 + \exp(\mathbf{w}^\top \mathbf{x}^{(i)})}$
- Approximation using only one data instance $(\mathbf{x}^{(i)}, y^{(i)})$
 - Stochastic gradient descent (SGD):

$$\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \frac{y^{(i)} \mathbf{x}^{(i)}}{1 + \exp(\mathbf{w}^\top \mathbf{x}^{(i)})}$$

Neural Networks

Success of “deep learning” : Real world applications

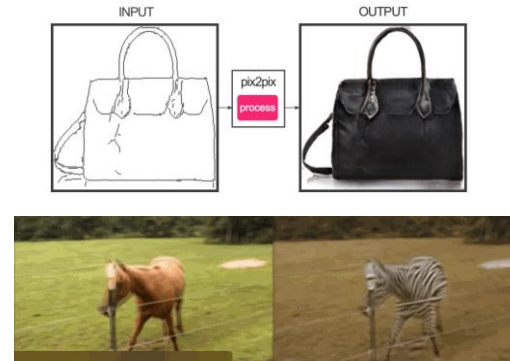
Image recognition



Machine Translation



Image/movie transformation



“Deep Fake”



AlphaGo



AlphaGo

AlphaFold2

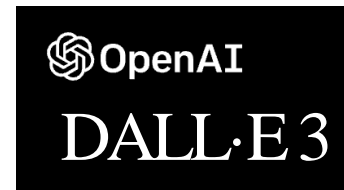
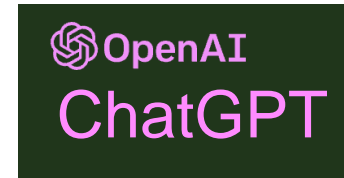


AlphaFold

AlphaTensor



AlphaTensor



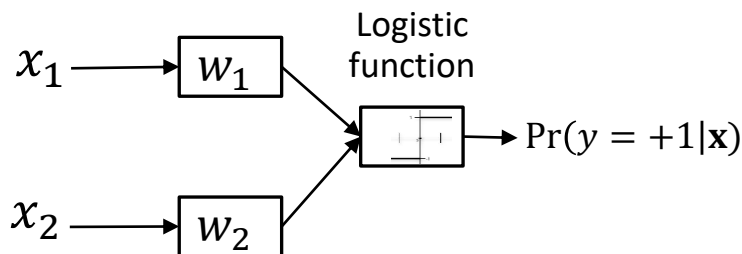
“Generative” AI

Neural networks:

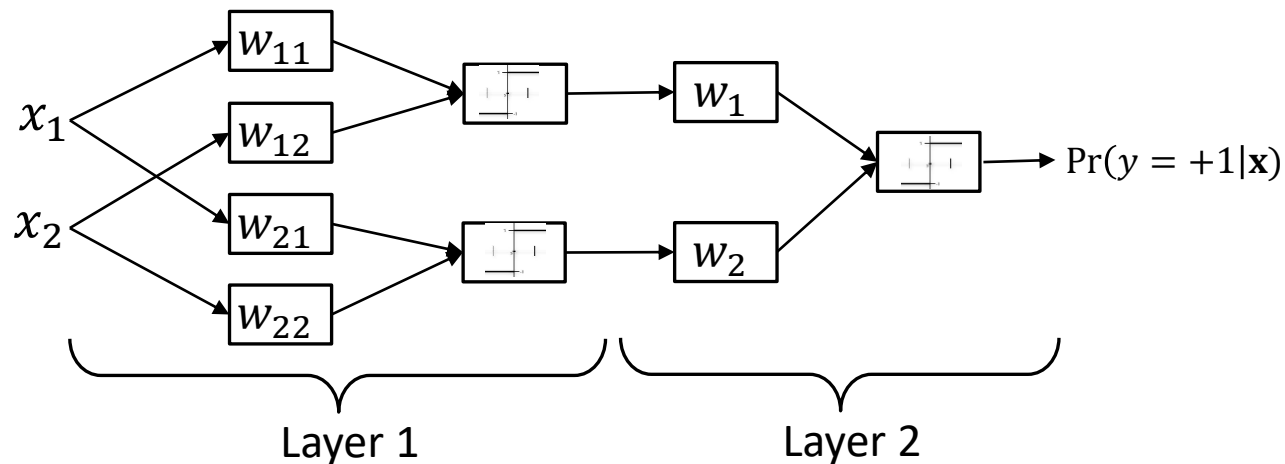
Multi-layered logistic regression models

- (Very roughly speaking,) a neural network is multi-layered logistic regression models
 - Outputs of some logistic regression models are inputs to other logistic regression models
- The form of final output is still $\Pr(y = +1|\mathbf{x})$

Logistic regression



Neural network (2 layers)

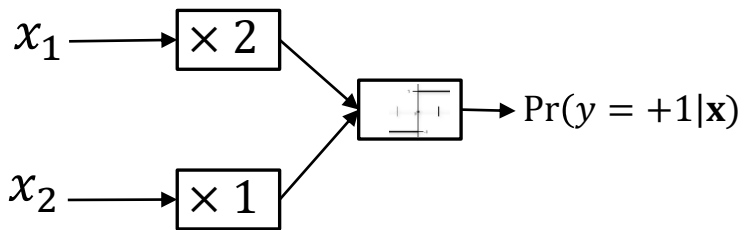


Why do we need to stack layers?

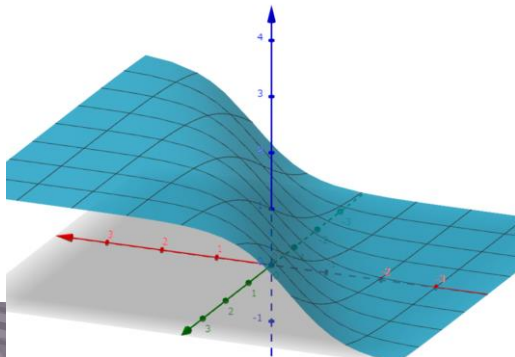
Nonlinear classification

- (1-layer) logistic regression only allow linear classification (AND/OR)
- Gains *non-linear* expressive power by stacking two layers (XOR)
- Universal approximation theorem: neural network can represent arbitrary functions by introducing many intermediate units

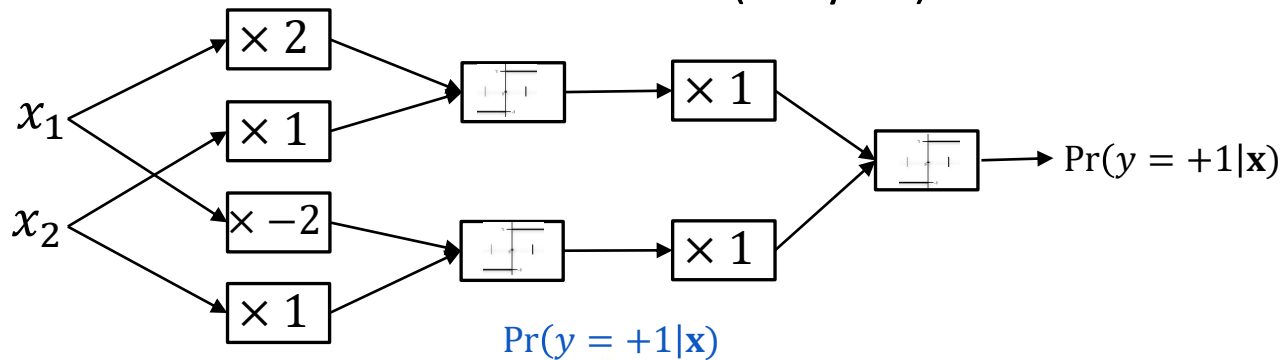
Logistic regression



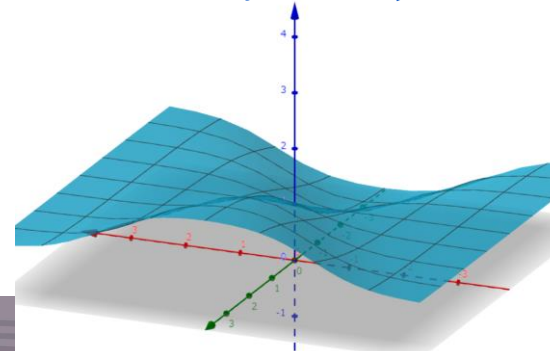
$\Pr(y = +1|\mathbf{x})$



Neural network (2 layers)



$\Pr(y = +1|\mathbf{x})$



Parameter estimation for neural networks:

Gradient is all you need

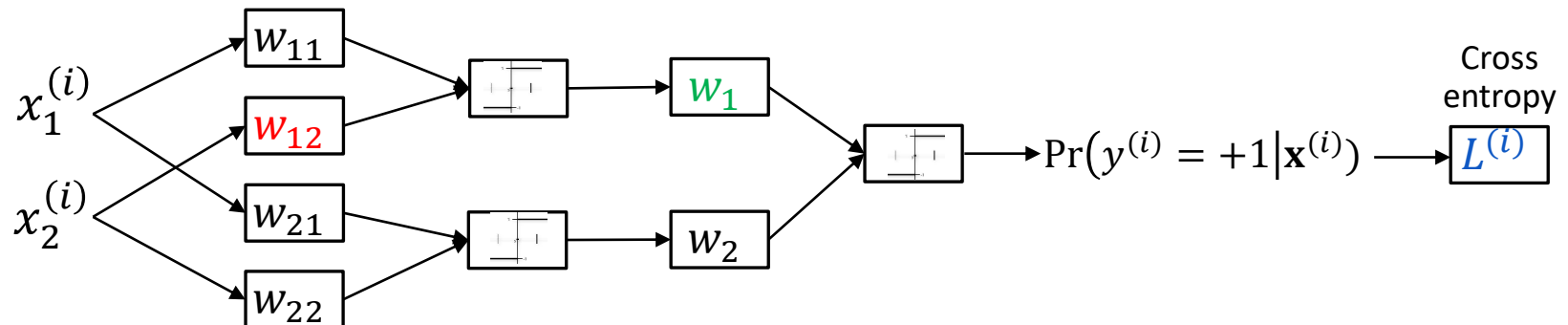
- The objective function for training a neural network is the same as that of logistic regression, i.e., the *cross entropy*

- When SGD is used, the cross entropy (only for the i -th instance) is

$$L^{(i)}(\mathbf{w}) = -\delta(y^{(i)} = 1)\log f_{\mathbf{w}}(y^{(i)} = +1|\mathbf{x}^{(i)}) - \delta(y^{(i)} = -1)\log f_{\mathbf{w}}(y^{(i)} = -1|\mathbf{x}^{(i)})$$

- Gradient descent update: $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \nabla L^{(i)}(\mathbf{w})$
- How can we obtain the gradient $\nabla L^{(i)}(\mathbf{w}) = \partial L^{(i)} / \partial \mathbf{w}$?

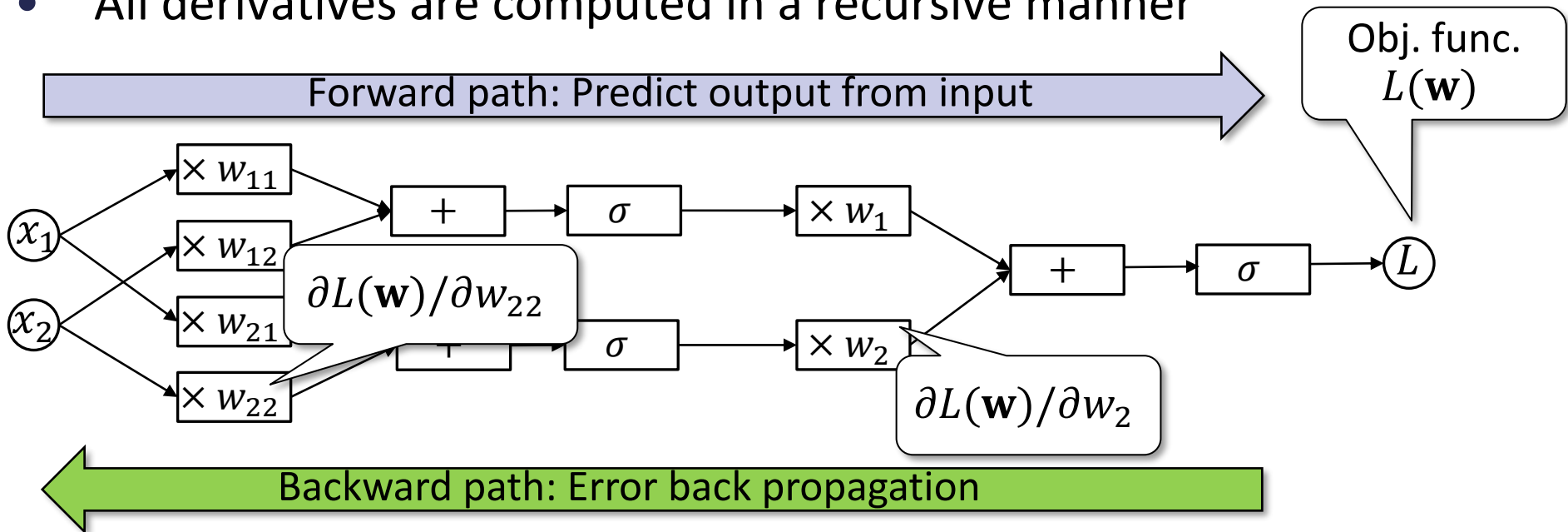
- E.g., how can we obtain $\partial L^{(i)} / \partial w_{12}$? (They are separated via w_1)



Error back propagation:

An efficient strategy to compute the gradient

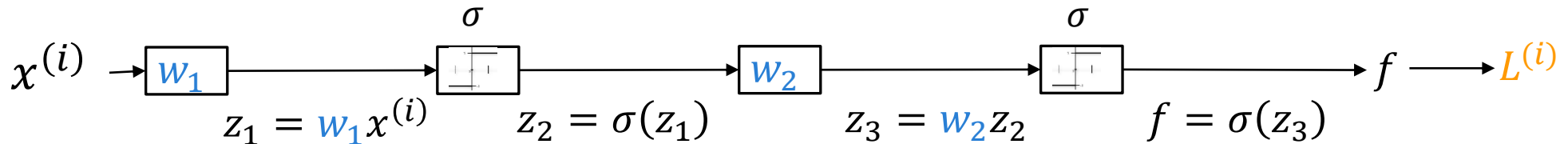
- Once we differentiate the objective function, we can apply SGD
- *Error backpropagation* can compute the gradient
 - Forward path: computes output (objective func.) from the input
 - Backward path: computes derivatives from output to input
- All derivatives are computed in a recursive manner



How error back propagation works:

Efficient computation using the chain rule of derivatives

- 1-dimensional case (of no practical use)



$$L^{(i)}(w_1, w_2) = -\delta(y^{(i)} = +1) \log f(x^{(i)}) - \delta(y^{(i)} = -1) \log (1 - f(x^{(i)}))$$

- Chain rule of derivatives:

$$\begin{aligned} \bullet \quad \frac{\partial L^{(i)}}{\partial w_2} &= \frac{\partial L^{(i)}}{\partial f} \cdot \frac{\partial f}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_2} \\ \bullet \quad \frac{\partial L^{(i)}}{\partial w_1} &= \frac{\partial L^{(i)}}{\partial f} \cdot \frac{\partial f}{\partial z_3} \cdot \frac{\partial z_3}{\partial z_2} \cdot \frac{\partial z_2}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} \end{aligned}$$

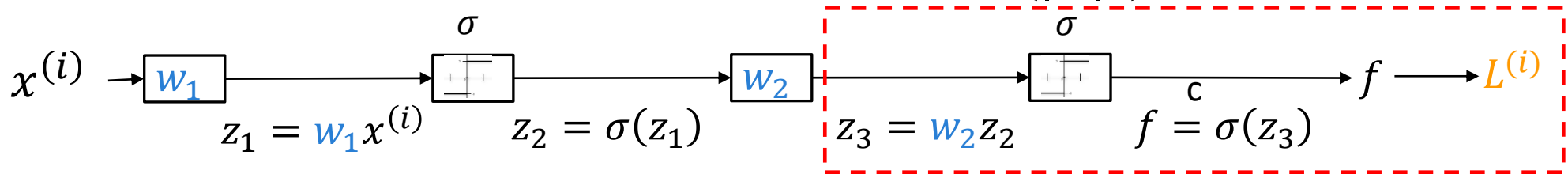
$$\left\{ \begin{aligned} \frac{\partial L^{(i)}}{\partial f} &= \frac{\delta(y^{(i)} = -1)}{1 - f(x^{(i)})} - \frac{\delta(y^{(i)} = +1)}{f(x^{(i)})} \\ \frac{\partial f}{\partial z_3} &= \sigma(z_3)(1 - \sigma(z_3)) \\ \frac{\partial z_3}{\partial w_2} &= z_2 \\ \frac{\partial z_3}{\partial z_2} &= w_2 \\ \frac{\partial z_2}{\partial z_1} &= \sigma(z_1)(1 - \sigma(z_1)) \\ \frac{\partial z_1}{\partial w_1} &= x^{(i)} \end{aligned} \right.$$

Already obtained in forward path

How error back propagation works:

Efficient computation using the chain rule of derivatives

- Naïve application of the chain rules requires $O(|U|^2)$ computation



- Backward computation allows reuse of common parts
 \Rightarrow results in $O(|U|^2)$ computation $|U|$: number of operation units
- Chain rule of derivatives:

$$\begin{aligned} \bullet \quad \frac{\partial L^{(i)}}{\partial w_2} &= \frac{\partial L^{(i)}}{\partial f} \cdot \frac{\partial f}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_2} \\ \bullet \quad \frac{\partial L^{(i)}}{\partial w_1} &= \frac{\partial L^{(i)}}{\partial f} \cdot \frac{\partial f}{\partial z_3} \cdot \frac{\partial z_3}{\partial z_2} \cdot \frac{\partial z_2}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} \end{aligned}$$

Reusable common part in backward path

Computational Graphs and Automatic Differentiation

Computational graph:

General representation of NN structure as a DAG

- A neural net can be constructed by stacking logistic regression models
 - Parameter estimation requires differentiating the output (the objective function) by parameters in the middle of the NN
 - This can be computed by applying the chain rule on this graph

⇒ Let us generalize them!

- Computational graph:

A directed acyclic graph representing computational process from input to output (NN decision process) using simple computational units:

- Addition and multiplication (matrix multiplication)
- Sigmoid transformation (simple nonlinear transformation)

⇒ Can we also use other types of units?

Requirements for computational unit in NN:

Output and its derivatives w.r.t. inputs and parameters

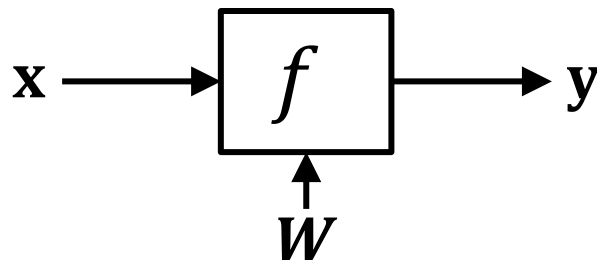
- Q: What computational units are allowed to be used in NN?

A: Required to have its output is differentiable w.r.t its

1. Inputs
2. Parameters

- In other words, we can use arbitrary units that offer

1. Output (for its input) : $\mathbf{y} = f(\mathbf{x}; \mathbf{W})$ } Used in forward path
2. Derivative of the output w.r.t its inputs: $\partial \mathbf{y} / \partial \mathbf{x}$
3. Derivative of the output w.r.t its parameters : $\partial \mathbf{y} / \partial \mathbf{W}$ } Used in backward path

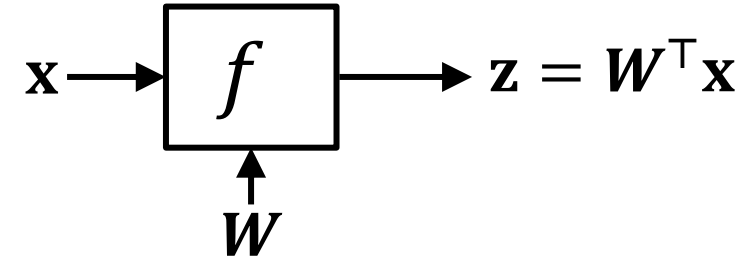


Examples of computational units in NN:

Output and its derivatives w.r.t. inputs and parameters

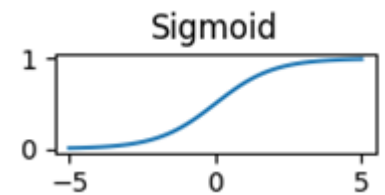
■ Linear unit:

- Output: $\mathbf{z} = \mathbf{W}^\top \mathbf{x}$
- Derivatives: $\partial \mathbf{z} / \partial \mathbf{x} = \mathbf{W}$, $\partial \mathbf{z} / \partial \mathbf{W} = \mathbf{x}$



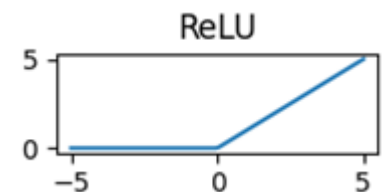
■ Sigmoid unit (Logistic unit):

- Output: $\mathbf{z} = \sigma(\mathbf{x})$ (σ is element-wise application of sigmoid function)
- Derivatives: $\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \sigma(\mathbf{x})(1 - \sigma(\mathbf{x}))$ (No parameter derivative)



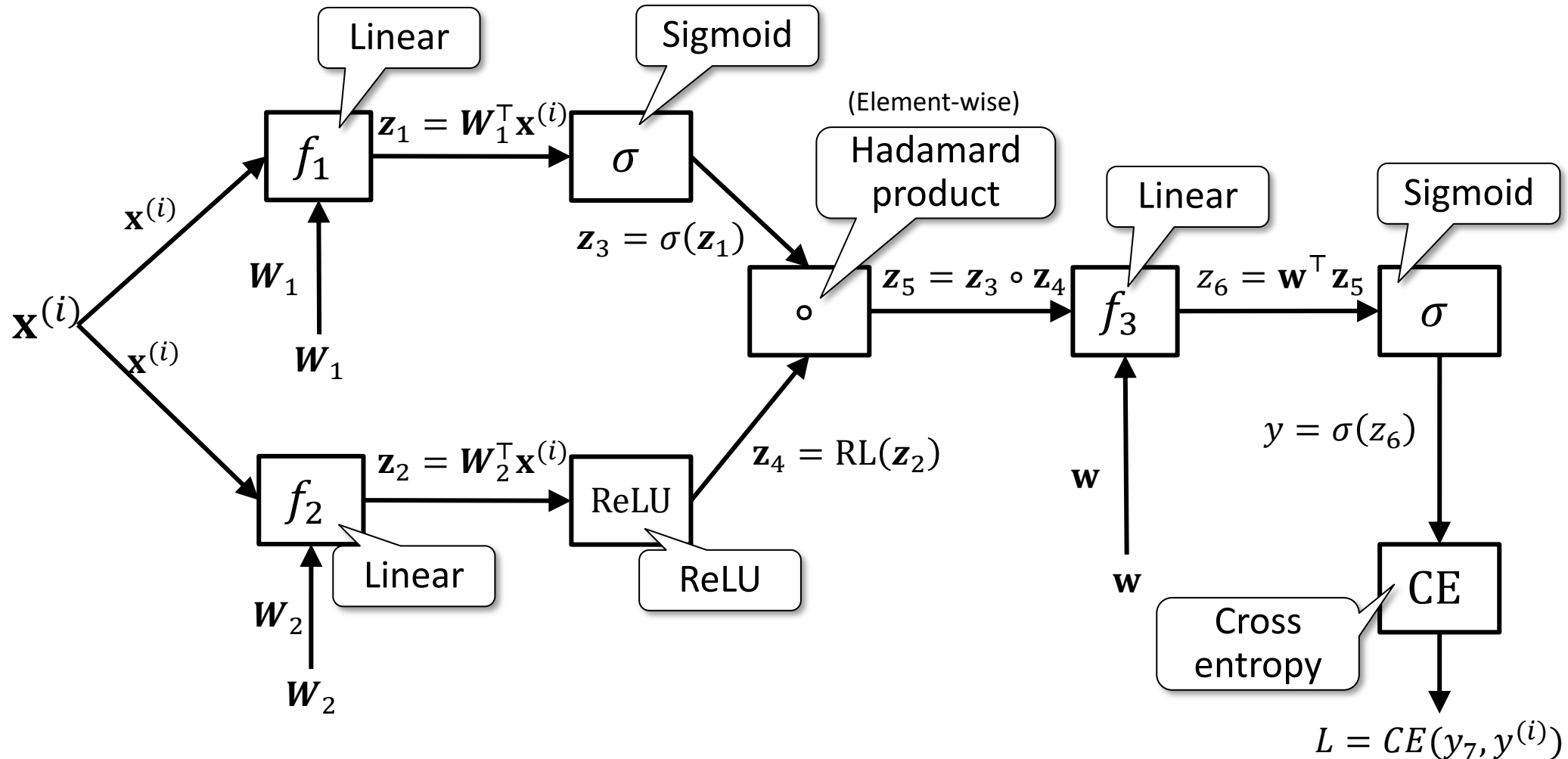
■ ReLU unit (Rectangular Linear Unit):

- Output: $\mathbf{z} = \max\{\mathbf{0}, \mathbf{x}\}$ (element-wise max)
- Derivatives: $\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \min\{\mathbf{0}, \mathbf{1}\}$ (No parameter derivative)



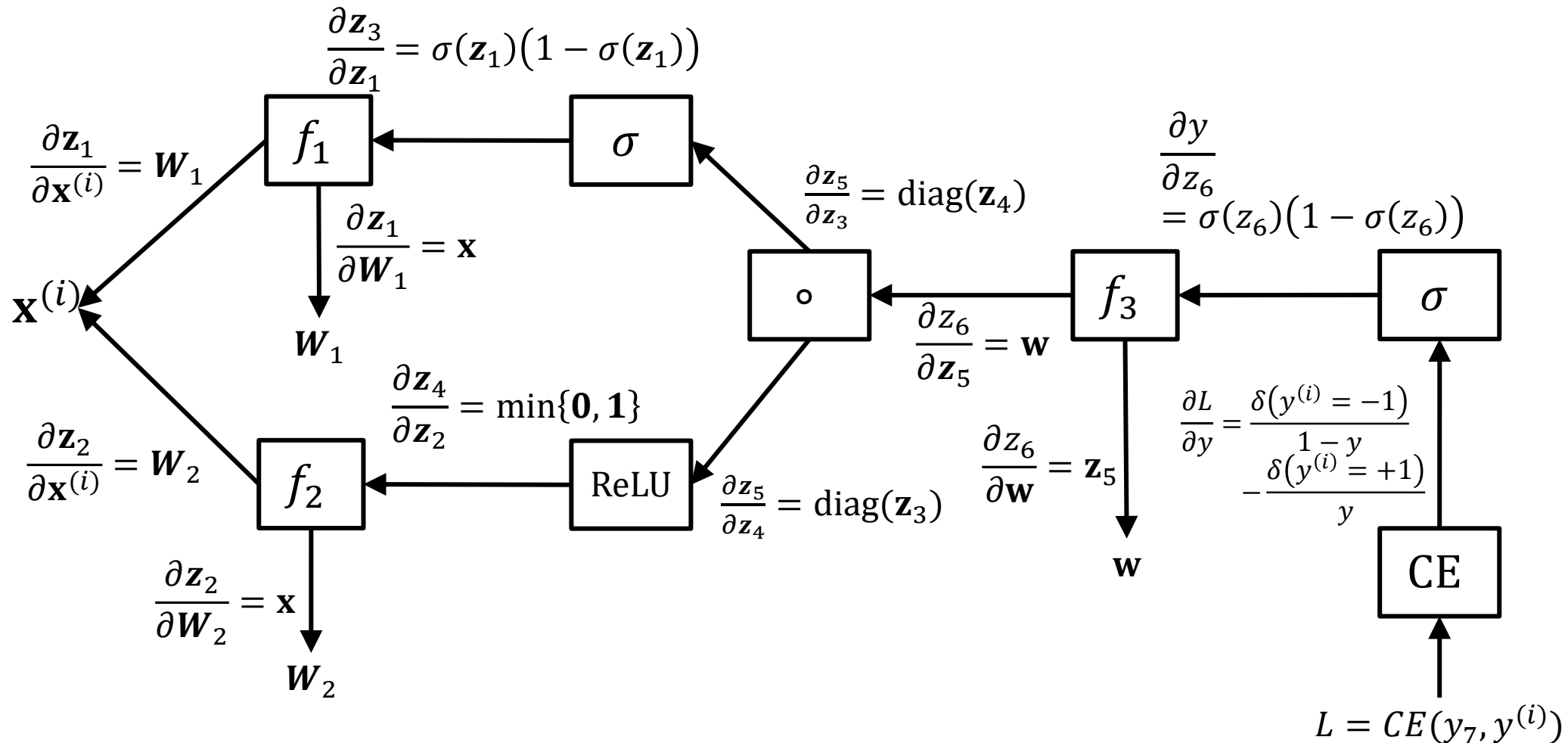
Automatic differentiation over computational graph: Forward path

- All units are differentiable w.r.t. inputs & parameters



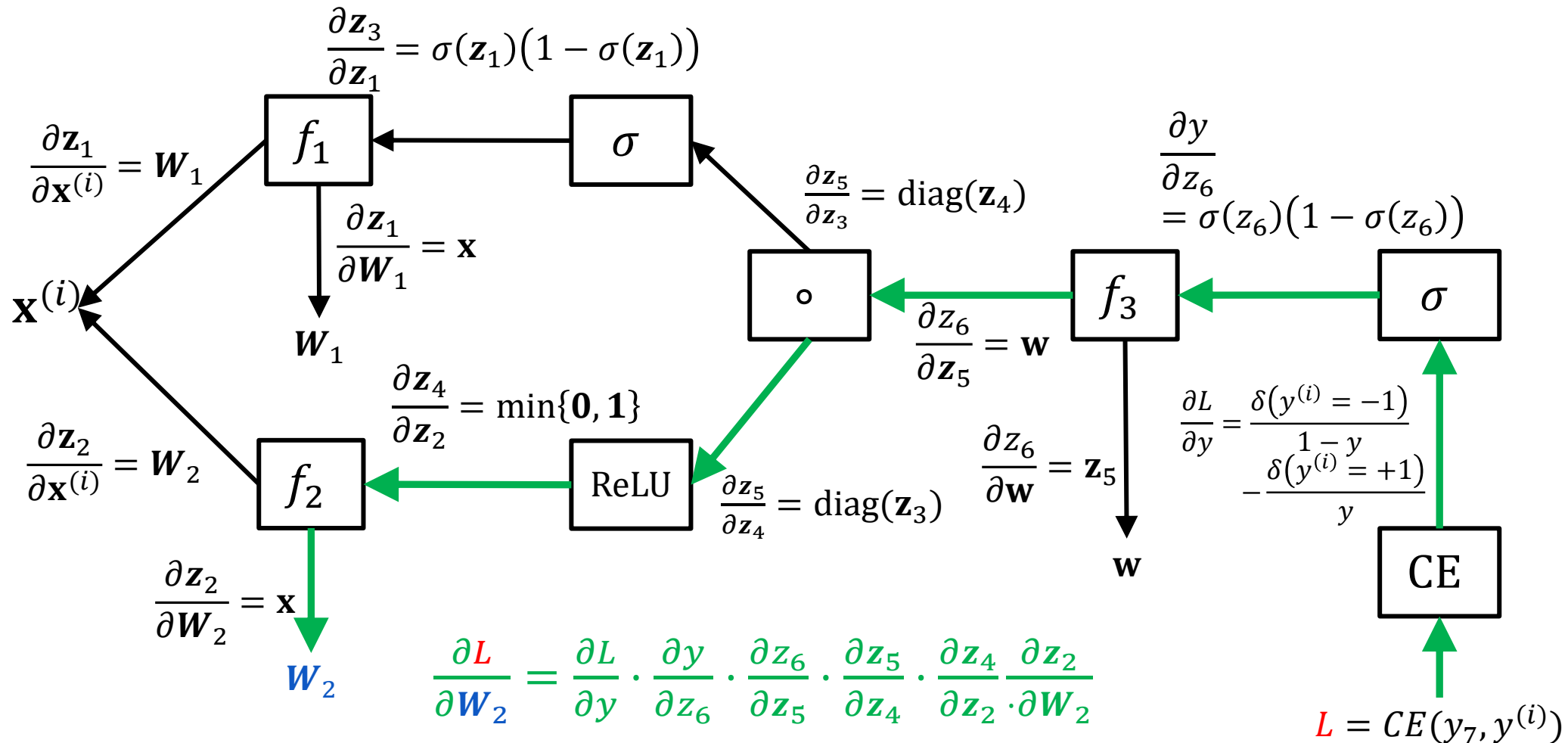
Automatic differentiation over computational graph: Backward path

- Traversals on backward paths give arbitrary derivatives



Automatic differentiation over computational graph: Backward path

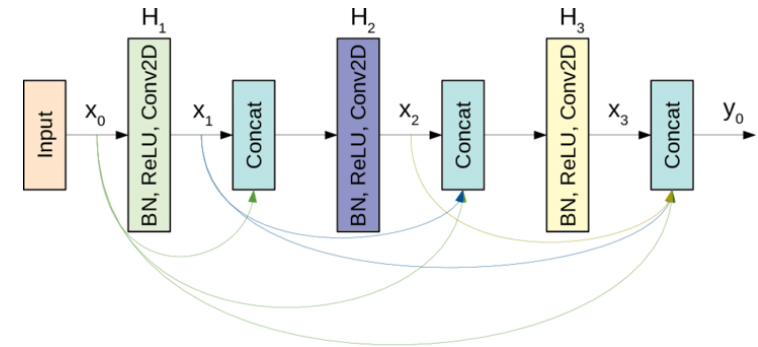
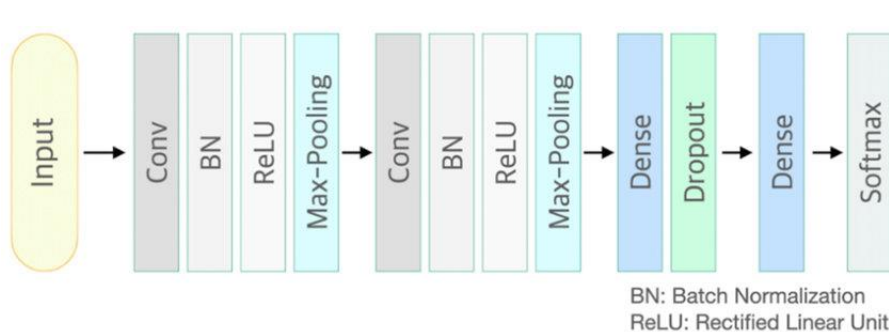
- Traversals on backward paths give arbitrary derivatives



Various deep neural network structure:

Flexible modeling by combining units

- NN allows intuitive and flexible modeling by combining various types of units like “LEGO®” blocks to form complex networks



- Error back propagation can be left to DL frameworks such as PyTorch
- Various task dependent neural networks have been proposed:
 - Images: Convolutional Neural Network (CNN); Vision Transformer (ViT)
 - Languages: Recurrent Neural Network (RNN), Transformer,
 - Graphs: Graph neural networks (GNN)

Training Deep Networks

Gradient vanishing problem:

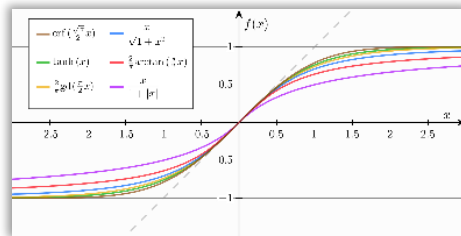
A major obstacle in training deep networks

- You can design and train arbitrary large networks, but in practice, we face the problem that training does not go well..
- Gradient vanishing problem: the derivative becomes weaker and weaker in the process of error back propagation
 - In the previous example, $\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z_6} \cdot \frac{\partial z_6}{\partial z_5} \cdot \frac{\partial z_5}{\partial z_4} \cdot \frac{\partial z_4}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_2}$
 - Repeated multiplication of small derivative values results in progressively smaller overall derivative values
- In deep networks, the gradient tends to be small, and hence gradient descent optimization does not proceed

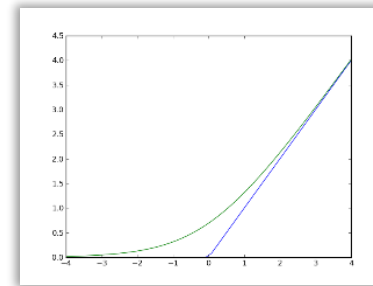
How to deal with gradient vanishing problem:

ReLU and skip connection

- Gradient of logistic function becomes small away from the origin
- Activation functions other than logistic function
 - Rectangular linear unit: $\max(0, x)$
 - Its derivative is 1 (or 0)



Logistic function



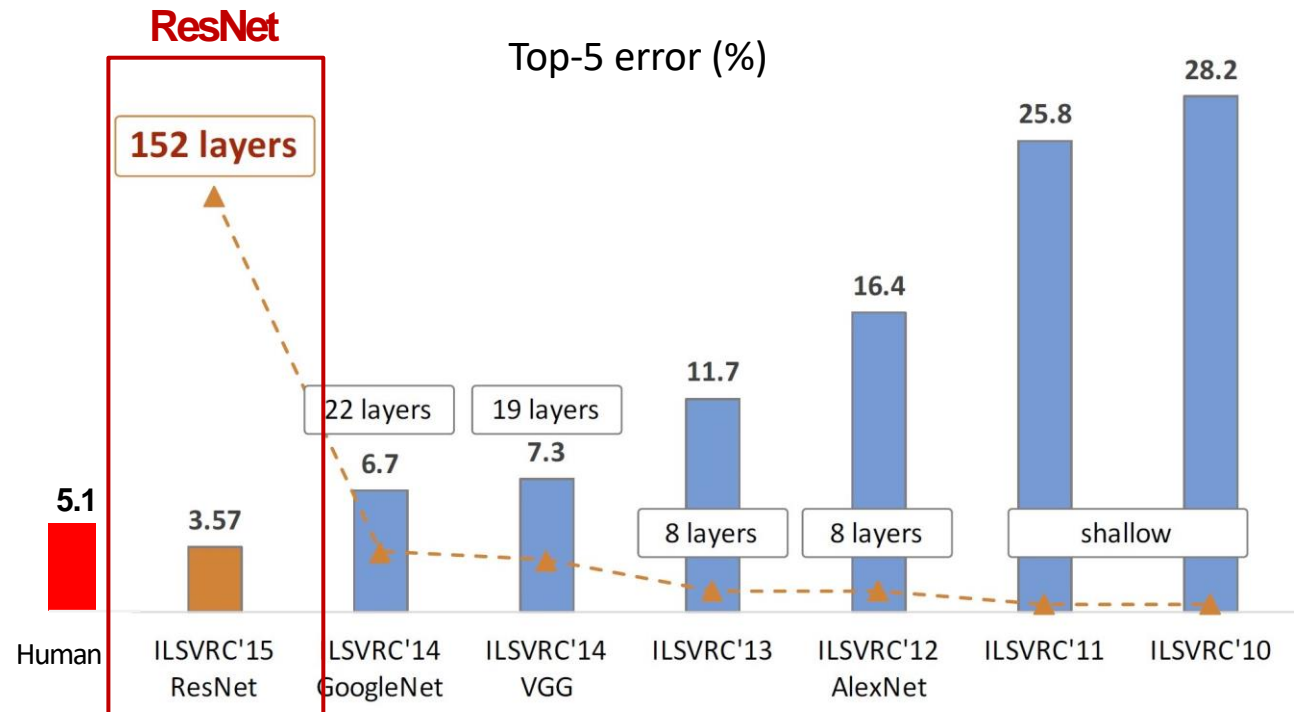
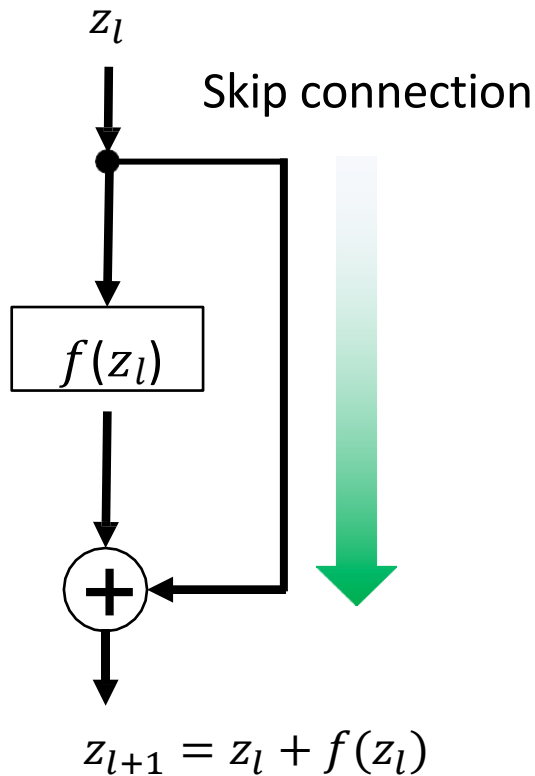
ReLU

- Softplus: $\ln(1 + e^x)$, LeakyReLU, and others

How to deal with gradient vanishing problem:

ReLU and skip connection

- Skip connection: $z_{l+1} = z_l + f(z_l)$
- Gradient is propagated directly without decay



Summary:

Neural networks

1. Neural network: (Very roughly speaking) stacked logistic regression
2. Training neural networks: gradient method (SGD, mini-batch, ...)
3. Computational graphs and automatic differentiation (error back propagation): gradients can be computed efficiently and systematically on a computational graph consisting of simple differentiable units
4. Training deep learning models: Dealing with the Gradient vanishing problem with ReLU, skip connection, ...