Introduction to Deep Learning

Jiong Zhang

Apr 2020

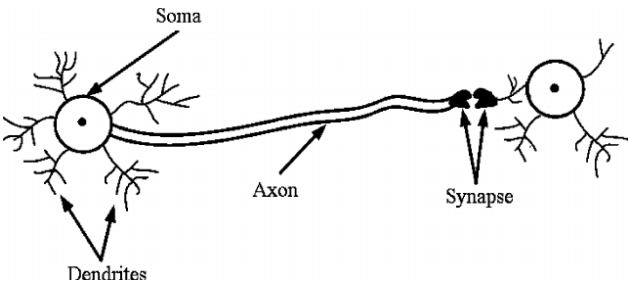
1 Overview

Example 1. Computer Vision (CV)

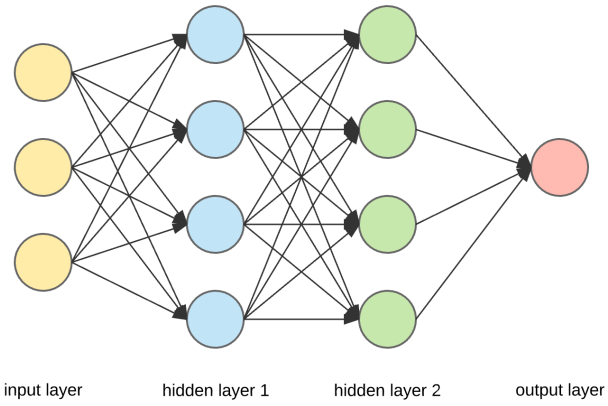
- object detection
- face recognition
- self-driving cars

Example 2. Natural Language Processing (NLP)

- machine translation
- article summarization
- search suggestion



Biological Neural Networks



Artificial Neural Networks

Model	Neurons	Synapses
Human	8.6e10	1.5e14
Cat	7.6e8	1.0e13
Bee	1.0e6	1.0e9
ResNet101	$\sim$ 1.0e5	4.4e7
BERT(large)[1]	$\sim$ 1.0e6	1.1e8

Table 1: Size comparison of Biological Neural Networks and Artificial Neural Networks

2 Feed-forward Neural Networks

2.1 Matrix Differentiation

Preliminaries for FNN back propagation derivation. Two major layouts for matrix differentiation:

- numerator layout:

$$\frac{\partial y}{\partial \overrightarrow{\boldsymbol{x}}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix} \qquad \frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \overrightarrow{\boldsymbol{x}}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \in \mathcal{R}^{m \times n} \tag{1}$$

- denominator layout

$$\frac{\partial y}{\partial \overrightarrow{\boldsymbol{x}}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \cdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix} \qquad \frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \overrightarrow{\boldsymbol{x}}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \in \mathcal{R}^{n \times m} \tag{2}$$

To simplify the notation, we adopt the numerator layout.

2.1.1 Basic Definitions

- scalar/vector: $y \in \mathcal{R}$, $\boldsymbol{x} \in \mathcal{R}^n$

$$\frac{\partial y}{\partial \overrightarrow{\boldsymbol{x}}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}; \left[\frac{\partial y}{\partial \overrightarrow{\boldsymbol{x}}}\right]^\top \in \mathcal{R}^n \tag{3}$$

- vector/scalar: $\boldsymbol{y} \in \mathcal{R}^m$, $x \in \mathcal{R}$

$$\frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \cdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix} \in \mathcal{R}^m \tag{4}$$

- vector/vector: $y \in \mathcal{R}^m$, $\boldsymbol{x} \in \mathcal{R}^n$

$$\frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \overrightarrow{\boldsymbol{x}}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \in \mathcal{R}^{m \times n} \tag{5}$$

- scalar/matrix: $y \in \mathcal{R}$, $\boldsymbol{X} \in \mathcal{R}^{m \times n}$

$$\frac{\partial y}{\partial \boldsymbol{X}} = \begin{bmatrix} \frac{\partial y}{\partial X_{1,1}} & \cdots & \frac{\partial y}{\partial X_{m,1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial X_{1,n}} & \cdots & \frac{\partial y}{\partial X_{m,n}} \end{bmatrix} \in \mathcal{R}^{n \times m} \tag{6}$$

- a order tensor/b order tensor:$\boldsymbol{Y} \in \mathcal{R}^{m_1 \times m_2 \times \cdots \times m_a}$, $\boldsymbol{X} \in \mathcal{R}^{n_1 \times n_2 \times \cdots \times n_b}$

$$\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{X}} \in \mathcal{R}^{m_1 \times \cdots \times m_a \times n_b \times \cdots \times n_1} \tag{7}$$

2.1.2 Derivative Formulas

Case 1. $\overrightarrow{\boldsymbol{y}} = \boldsymbol{A}\overrightarrow{\boldsymbol{x}}$, $\overrightarrow{\boldsymbol{x}} \in \mathcal{R}^n$, $\boldsymbol{A} \in \mathcal{R}^{m\times n}$

$$\left[\frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \overrightarrow{\boldsymbol{x}}}\right]_{i,j} = \frac{\partial y_i}{\partial x_j} = \frac{\partial \sum_{t=1}^{n} A_{i,t}x_t}{\partial x_j} = A_{i,j} \tag{8}$$

Therefore

$$\frac{\partial \boldsymbol{A}\overrightarrow{\boldsymbol{x}}}{\partial \overrightarrow{\boldsymbol{x}}} = \boldsymbol{A} \tag{9}$$

Case 2. $\overrightarrow{\boldsymbol{y}} = \boldsymbol{X}\overrightarrow{\boldsymbol{a}}$, $\boldsymbol{X} \in \mathcal{R}^{m\times n}$, $\overrightarrow{\boldsymbol{a}} \in \mathcal{R}^n$

$$\left[\frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \boldsymbol{X}}\right]_{i,j,k} = \frac{\partial y_i}{\partial X_{k,j}} = \frac{\partial \sum_{t=1}^{n} X_{i,t}a_t}{\partial X_{k,j}} = a_j \mathbb{I}(i=k) \tag{10}$$

Therefore

$$\frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \boldsymbol{X}} = \rho(\overrightarrow{\boldsymbol{a}}, m) \in \mathcal{R}^{m\times n\times m} \tag{11}$$

$$\rho(\overrightarrow{\boldsymbol{a}}, m)_{i,j,k} \equiv a_j \mathbb{I}(i=k) \tag{12}$$

Case 3. Chain Rule: $\overrightarrow{\boldsymbol{y}} \in \mathcal{R}^m$ is a function of $\overrightarrow{\boldsymbol{z}} \in \mathcal{R}^r$ and $\overrightarrow{\boldsymbol{z}}$ is a function of $\overrightarrow{\boldsymbol{x}} \in \mathcal{R}^n$.

$$\left[\frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \overrightarrow{\boldsymbol{x}}}\right]_{i,j} = \frac{\partial y_i}{\partial x_j} \tag{13}$$

By chain rule

$$\frac{\partial y_i}{\partial x_j} = \sum_{t=1}^{r} \frac{\partial y_i}{\partial z_t}\frac{\partial z_t}{\partial x_j} = \frac{\partial y_i}{\partial \overrightarrow{\boldsymbol{z}}}\frac{\partial \overrightarrow{\boldsymbol{z}}}{\partial x_j} = \left[\frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \overrightarrow{\boldsymbol{z}}}\right]_{i,:}\left[\frac{\partial \overrightarrow{\boldsymbol{z}}}{\partial \overrightarrow{\boldsymbol{x}}}\right]_{:,j} \tag{14}$$

Therefore

$$\frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \overrightarrow{\boldsymbol{x}}} = \frac{\partial \overrightarrow{\boldsymbol{y}}}{\partial \overrightarrow{\boldsymbol{z}}}\frac{\partial \overrightarrow{\boldsymbol{z}}}{\partial \overrightarrow{\boldsymbol{x}}} \tag{15}$$

Chain rule of matrix differentiation (numerator layout) "toward right". For denominator layout it is "toward left".

2.2 Multi-layer Perceptrons: neural networks

Previously we learned about perceptrons, however, in most real applications, a linear combination of input features is not enough to make accurate predictions. We need to capture interactions among input features (feature extraction).

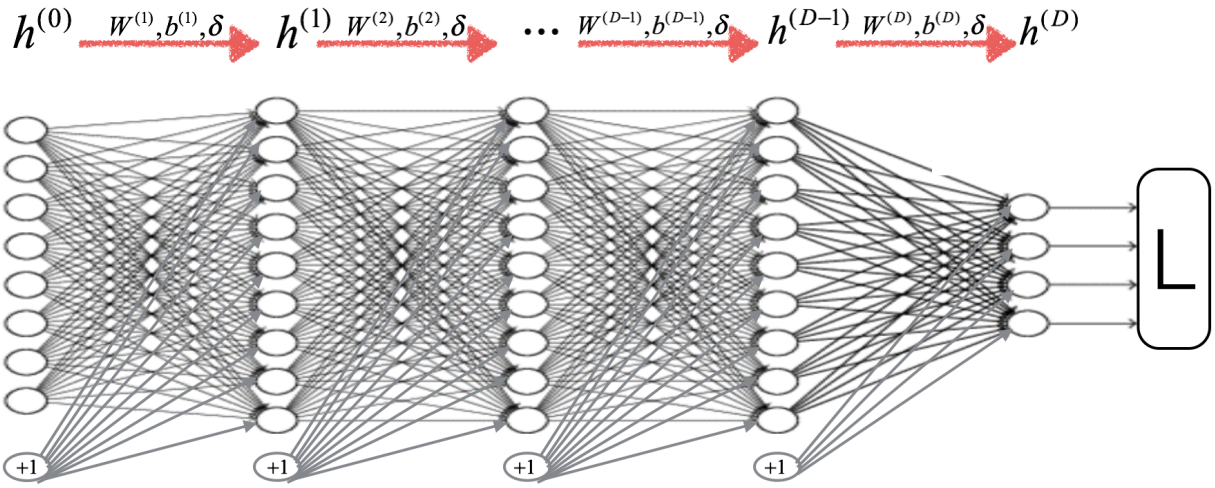


Figure 1: A Multi-layer Perceptron with depth D

Consider regression problem with input data $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^M$ and $\boldsymbol{Y} = \{\boldsymbol{y}_i\}_{i=1}^M$, where $\boldsymbol{x}_i \in \mathcal{R}^{n_0}$ and $\boldsymbol{y}_i \in \mathcal{R}^{n_D}$. A Multi-layer Perceptron (MLP) with depth D is defined as:

$$\boldsymbol{h}^{(d)} = \delta(\boldsymbol{W}^{(d)}\boldsymbol{h}^{(d-1)} + \boldsymbol{b}^{(d)}), \text{ for } d = 1, 2, \cdots, D \tag{16}$$

$$\boldsymbol{h}^{(0)} = \boldsymbol{x} \tag{17}$$

where $\boldsymbol{h}^{(d)} \in \mathcal{R}^{n_d}$ are the hidden vectors, $\boldsymbol{W}^{(d)} \in \mathcal{R}^{n_d \times n_{d-1}}$ and $\boldsymbol{b}^{(d)} \in \mathcal{R}^{n_d}$ are trainable parameters. The loss function is defined on the final outputs $\boldsymbol{h}^{(D)}$:

$$L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{M}\sum_{i=1}^{M} \widehat{l}(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = \frac{1}{M}\sum_{i=1}^{M} l(\boldsymbol{h}^{(D)}(\boldsymbol{\theta}, \boldsymbol{x}_i), \boldsymbol{y}_i) \tag{18}$$

where $\boldsymbol{\theta} = [\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \cdots, \boldsymbol{W}^{(D)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \cdots, \boldsymbol{b}^{(D)}]$.

Model Training:SGD

- Initialize $\boldsymbol{W}_0^{(d)}, \boldsymbol{b}_0^{(d)}$ for $d = 1, 2, \cdots, D$, learning rate $\eta \in \mathcal{R}$
- For $t = 1, 2, \cdots, T$:
  - Sample $i \sim Uniform(1, 2, \cdots, M)$
  - For $d = D, D-1, \cdots, 1$:
    - $\boldsymbol{W}_{t+1}^{(d)} \leftarrow \boldsymbol{W}_t^{(d)} - \eta\left[\frac{\partial \widehat{l}}{\partial \boldsymbol{W}^{(d)}}(\boldsymbol{\theta}_t; \boldsymbol{x}_i, \boldsymbol{y}_i)\right]^\top$
    - $\boldsymbol{b}_{t+1}^{(d)} \leftarrow \boldsymbol{b}_t^{(d)} - \eta\left[\frac{\partial \widehat{l}}{\partial \boldsymbol{b}^{(d)}}(\boldsymbol{\theta}_t; \boldsymbol{x}_i, \boldsymbol{y}_i)\right]^\top$

Forward Propagation. Note that each hidden vector $\boldsymbol{h}^{(d)}$ is dependent on its previous layer $\boldsymbol{h}^{(d-1)}$. Therefore, to compute the model output $\boldsymbol{h}^{(D)}$, all the hidden vectors needs to be evaluated in a forwarding sequence.

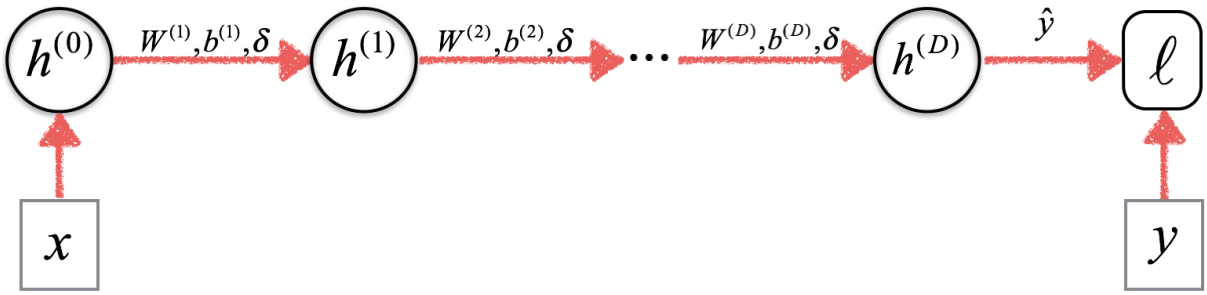


Figure 2: Forward propagation

2.3 Back Propagation.

To train parameters $\boldsymbol{W}^{(d)}$, we need to evaluate $\frac{\partial l}{\partial \boldsymbol{w}^{(d)}}(\boldsymbol{\theta}_t; \boldsymbol{x}_i, \boldsymbol{y}_i)$. The explicit formulation for the MLP can be written as:

$$l_i = \widehat{l}(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = l(\delta(\boldsymbol{W}^{(D)}\delta(\boldsymbol{W}^{(D-1)} \cdots \delta(\boldsymbol{W}^{(1)}\boldsymbol{x}_i + \boldsymbol{b}^{(1)}) + \cdots + \boldsymbol{b}^{(D)}), \boldsymbol{y}_i). \tag{19}$$

We suppress subscript i for simplicity. By definition:

$$\frac{\partial l}{\partial \boldsymbol{h}^{(D)}} = \begin{bmatrix} \frac{\partial l}{\partial h_1^{(D)}} & \frac{\partial l}{\partial h_2^{(D)}} & \cdots & \frac{\partial l}{\partial h_{n_D}^{(D)}} \end{bmatrix} \tag{20}$$

Denoting $\boldsymbol{z}^{(d)} = \boldsymbol{W}^{(d)}\boldsymbol{h}^{(d-1)} + \boldsymbol{b}^{(d)}$ The derivatives between adjacent layers can be written as

$$\frac{\partial \boldsymbol{h}^{(d)}}{\partial \boldsymbol{h}^{(d-1)}} = \frac{\partial \boldsymbol{h}^{(d)}}{\partial \boldsymbol{z}^{(d)}} \frac{\partial \boldsymbol{z}^{(d)}}{\partial \boldsymbol{h}^{(d-1)}} \tag{21}$$

where

$$\frac{\partial \boldsymbol{h}^{(d)}}{\partial \boldsymbol{z}^{(d)}} \equiv diag(\dot{\delta}^{(d)}) = \begin{bmatrix} \delta'(\boldsymbol{z}_1^{(d)}) & & \\ & \ddots & \\ & & \delta'(\boldsymbol{z}_{n_d}^{(d)}) \end{bmatrix} \tag{22}$$

$$= \begin{bmatrix} \delta'(\boldsymbol{W}_{:,1}^{(d)} \cdot \boldsymbol{h}^{(d-1)} + b_1^{(d)}) & & \\ & \ddots & \\ & & \delta'(\boldsymbol{W}_{:,n_d}^{(d)} \cdot \boldsymbol{h}^{(d-1)} + b_{n_d}^{(d)}) \end{bmatrix} \tag{23}$$

And

$$\frac{\partial \boldsymbol{z}^{(d)}}{\partial \boldsymbol{h}^{(d-1)}} = \boldsymbol{W}^{(d)} \in \mathcal{R}^{n_d \times n_{d-1}} \tag{24}$$

Therefore, by chain rule:

$$\frac{\partial l}{\partial \boldsymbol{W}^{(d)}} = \frac{\partial l}{\partial \boldsymbol{h}^{(d)}} \frac{\partial \boldsymbol{h}^{(d)}}{\partial \boldsymbol{W}^{(d)}} \tag{25}$$

$$= \frac{\partial l}{\partial \boldsymbol{h}^{(D)}} \frac{\partial \boldsymbol{h}^{(D)}}{\partial \boldsymbol{h}^{(D-1)}} \cdots \frac{\partial \boldsymbol{h}^{(d+1)}}{\partial \boldsymbol{h}^{(d)}} \frac{\partial \boldsymbol{h}^{(d)}}{\partial \boldsymbol{W}^{(d)}} \tag{26}$$

Note that $\frac{\partial \boldsymbol{h}^{(d)}}{\partial \boldsymbol{W}^{(d)}} = diag(\dot{\delta}^{(d)})\rho(\boldsymbol{h}^{(d-1)}, n_d) \in \mathcal{R}^{n_d \times\ n_{d-1} \times n_d}$ is a third order tensor. The gradient w.r.t. $\boldsymbol{W}^{(d)}$ can be written as:

$$\frac{\partial l}{\partial \boldsymbol{W}^{(d)}} = \frac{\partial l}{\partial \boldsymbol{h}^{(D)}} diag(\dot{\delta}^{(D)})\boldsymbol{W}^{(D)} \cdots diag(\dot{\delta}^{(d+1)})\boldsymbol{W}^{(d+1)} diag(\dot{\delta}^{(d)})\rho(\boldsymbol{h}^{(d-1)}, n_d) \tag{27}$$

Therefore, the gradients can be evaluated as:

$$\frac{\partial l}{\partial \boldsymbol{W}^{(d)}} = \frac{\partial l}{\partial \boldsymbol{h}^{(D)}} \left[\prod_{i=D}^{d+1} diag(\dot{\delta}^{(i)}) \boldsymbol{W}^{(i)} \right] diag(\dot{\delta}^{(d)}) \rho(\boldsymbol{h}^{(d-1)}, n_d) \qquad \in \mathcal{R}^{n_{d-1} \times n_d} \tag{28}$$

$$\frac{\partial l}{\partial \boldsymbol{b}^{(d)}} = \frac{\partial l}{\partial \boldsymbol{h}^{(D)}} \left[\prod_{i=D}^{d+1} diag(\dot{\delta}^{(i)}) \boldsymbol{W}^{(i)} \right] diag(\dot{\delta}^{(d)}) \qquad \in \mathcal{R}^{1 \times n_d} \tag{29}$$

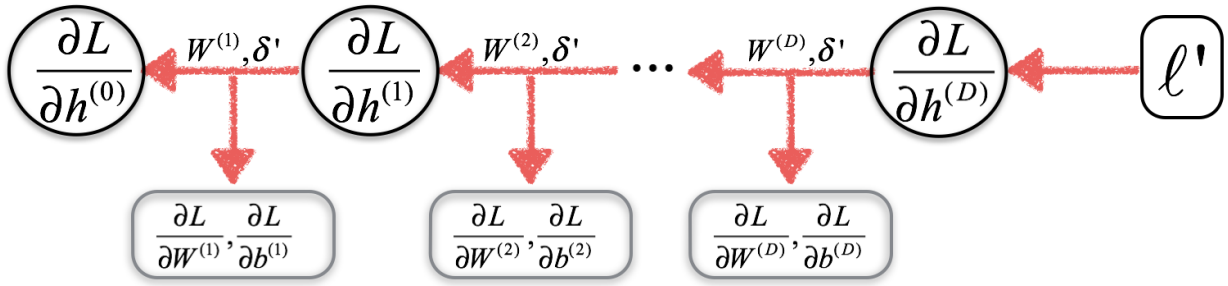


Figure 3: Back propagation

2.4 Activation Functions

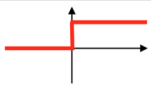
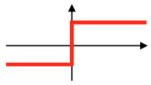
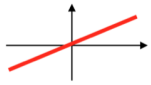
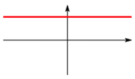
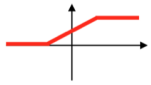
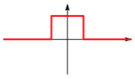
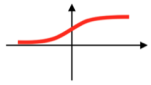
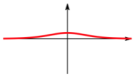
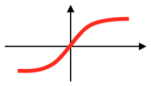
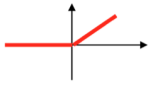
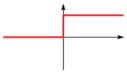
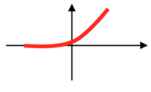
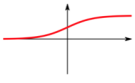
Activation function	Equation	Example	1D Graph	Derivative
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant		
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant		
Linear	$\phi(z) = z$	Adaline, linear regression		
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine		
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN		
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks		
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks		
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks		

Figure 4: Common activation functions and their derivatives.

- Absolute value of input is large for activations $\Rightarrow$ Saturated activation functions ($\dot{\delta}_i \ll 1$) $\Rightarrow$ vanishing gradients
- Solutions: constrain size of each h, use $ReLU$ or $softplus$

3 Recurrent Neural Networks

Question: What if input data are sequences of different lengths? Eg. translation, voice recognition, stock value prediction.

- MLPs can deal with input data with fixed dimensions.
- For sequence data, recurrent models are usually used.

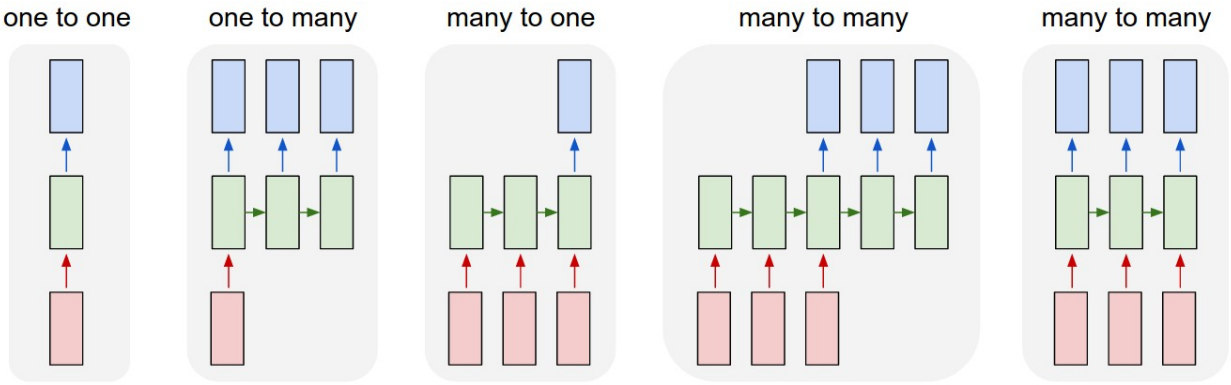


Figure 5: Different use cases of RNNs

Examples:

- Neural Machine Translation, chat robot: many to many (case 4)
- Language modelling, time series prediction: many to many (case 5)
- Voice recognition: many to one (case 3)

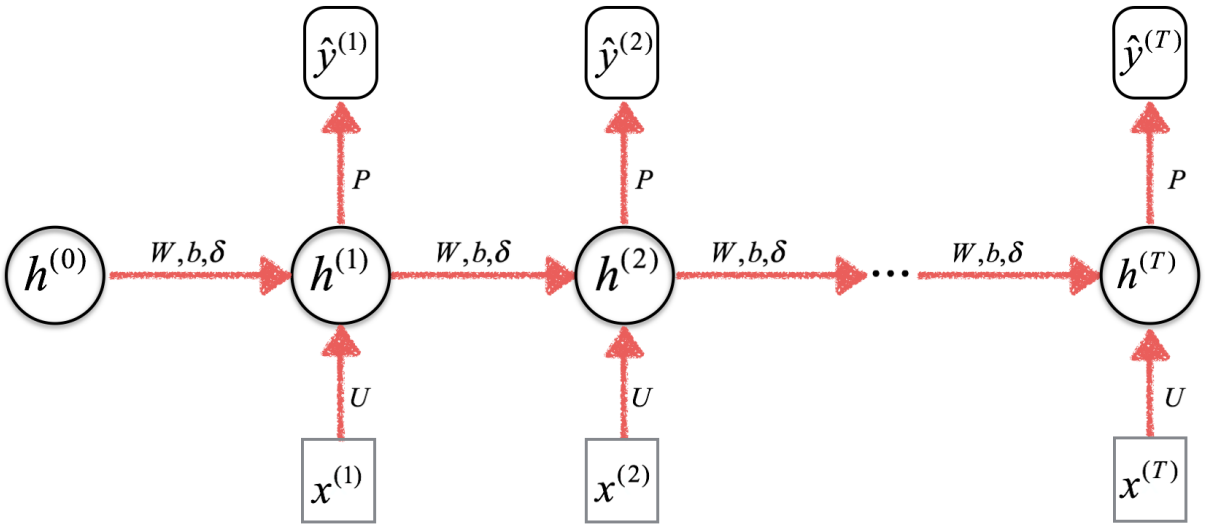


Figure 6: Recurrent Neural Networks: forward propagation

Consider use case 5 with input data $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^M$ and $\boldsymbol{Y} = \{\boldsymbol{y}_i\}_{i=1}^M$, where

$$\boldsymbol{x}_i = \left[\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(2)}, \cdots, \boldsymbol{x}_i^{(T_i)}\right], \boldsymbol{x}_i^{(t)} \in \mathcal{R}^{n_i}$$
$$\boldsymbol{y}_i = \left[\boldsymbol{y}_i^{(1)}, \boldsymbol{y}_i^{(2)}, \cdots, \boldsymbol{y}_i^{(T_i)}\right], \boldsymbol{y}_i^{(t)} \in \mathcal{R}^{n_o}$$

- RNN with activation function δ:

$$\boldsymbol{h}^{(t)} = \delta(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t-1)} + \boldsymbol{b}) \tag{30}$$
$$\widehat{\boldsymbol{y}}^{(t)} = \boldsymbol{P}\boldsymbol{h}^{(t)} \tag{31}$$

  where $\boldsymbol{h}^{(t)} \in \mathcal{R}^{n_h}$ are hidden vectors to encode input information

- Key differences from MLP:
  - Parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}, \boldsymbol{P}\}$ are shared for each cell/time-step
  - Input $\boldsymbol{x}_i = \left[\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(2)}, \cdots, \boldsymbol{x}_i^{(T_i)}\right]$ is fed sequentially
  - Output $\widehat{\boldsymbol{y}} = \{\widehat{\boldsymbol{y}}^{(0)}, \widehat{\boldsymbol{y}}^{(1)}, \ldots \widehat{\boldsymbol{y}}^{(T_i)}\}$ is evaluated at each cell/time-step

- Loss is measured as:

$$L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{M}\sum_{i=1}^{M} \widehat{l}(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i) \tag{32}$$
$$= \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{T_i} l(\widehat{\boldsymbol{y}}_i^{(j)}, \boldsymbol{y}_i^{(j)}) \tag{33}$$

3.1 Back propagation

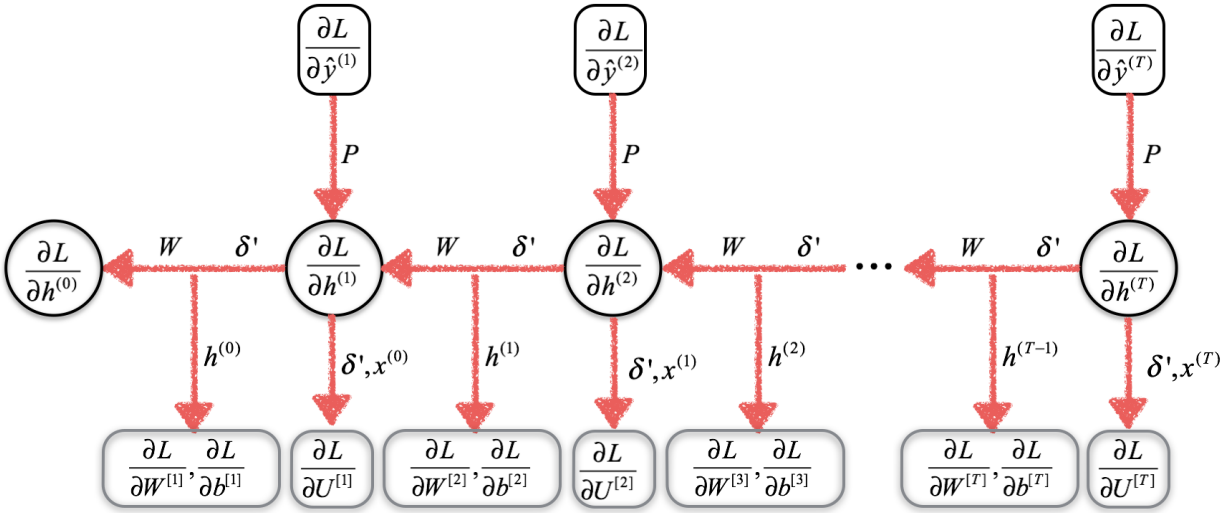


Figure 7: Recurrent Neural Networks: back propagation

Suppressing i, let $l_j \equiv l(\widehat{\boldsymbol{y}}^{(j)}, \boldsymbol{y}^{(j)})$, we have:

$$l_T = l(\boldsymbol{P}\delta(\boldsymbol{W}^{(T)}\delta(\boldsymbol{W}^{(T-1)}\cdots\delta(\boldsymbol{W}^{(0)}\boldsymbol{h}^{(0)} + \boldsymbol{U}^{(0)}\boldsymbol{x}^{(0)} + \boldsymbol{b}^{(1)}) + \cdots + \boldsymbol{U}^{(T)}\boldsymbol{x}^{(T)} + \boldsymbol{b}^{(T)}), \boldsymbol{y}^{(T)}). \tag{34}$$

Since in RNNs the weight matrices are shared for all time steps:

$$\begin{aligned}\boldsymbol{W}^{(0)} &= \boldsymbol{W}^{(1)} = \cdots = \boldsymbol{W}^{(T)}\\ \boldsymbol{U}^{(0)} &= \boldsymbol{U}^{(1)} = \cdots = \boldsymbol{U}^{(T)}\\ \boldsymbol{b}^{(0)} &= \boldsymbol{b}^{(1)} = \cdots = \boldsymbol{b}^{(T)}\end{aligned}$$

the back propagation needs to be done backward through all time steps (Back Propagation Through Time-BPTT):

$$\frac{\partial l_T}{\partial \boldsymbol{W}} = \sum_{t=T}^{0} \frac{\partial l_T}{\partial \boldsymbol{W}^{(t)}}. \tag{35}$$

Each term can be calculated in the same way as MLP:

$$\frac{\partial l_T}{\partial \boldsymbol{W}^{(t)}} = \frac{\partial l_T}{\partial \boldsymbol{h}^{(T)}}\left[\prod_{\tau=T}^{t+1} diag(\dot{\delta_\tau})\boldsymbol{W}\right] diag(\dot{\delta}_t)\rho(\boldsymbol{h}^{(t)}, n_h) \tag{36}$$

3.2 Gradient Vanishing/Exploding

Note that in Eq 36, the same $\boldsymbol{W}$ appears $T-t$ times in the back propagation. This makes the gradient unstable for RNNs. Consider the linear case, $\frac{\partial l_T}{\partial \boldsymbol{W}^{(t)}}$ reduces to:

$$\frac{\partial l_T}{\partial \boldsymbol{x}^{(t)}} = \frac{\partial l_T}{\partial \boldsymbol{h}^{(T)}}[\prod_{\tau=T}^{t} \boldsymbol{W}]\rho(\boldsymbol{h}^{(t)}, n_h) \tag{37}$$

$$= \frac{\partial l_T}{\partial \boldsymbol{h}^{(T)}}\boldsymbol{W}^{T-t}\rho(\boldsymbol{h}^{(t)}, n_h) \tag{38}$$

- If $\|\boldsymbol{W}\| < 1$, $\frac{\partial l_T}{\partial \boldsymbol{W}^{(t)}}$ vanishes for $t << T$
- If $\|\boldsymbol{W}\| > 1$, $\frac{\partial l_T}{\partial \boldsymbol{W}^{(t)}}$ explodes for $t << T$

Only short term dependencies can be captured!

Solutions

- Gated architectures: LSTM, GRU
- Constrain $\|\boldsymbol{W}\| \approx 1$
- Gradient clipping and etc.

4 Convolutional Neural Networks

Question: How to deal with image data?
MLP's Problems:

- High dimensional input: RGB image 256x256, input dimension 196608
- Spatial correlations: input features are only locally dependent, no need to capture global correlations.

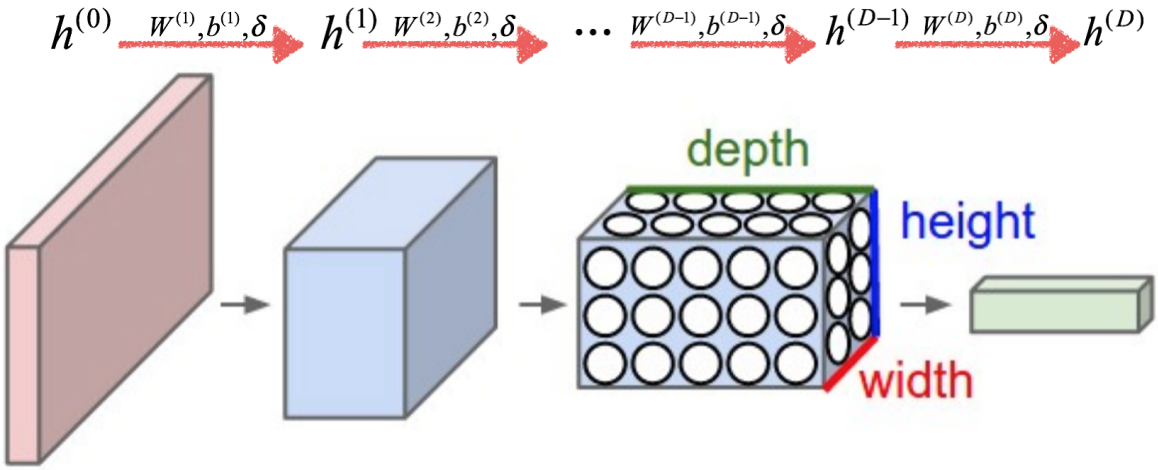


Figure 8: Typical CNN

The most popular model for image data is Convolutional Neural Networks (CNNs).

- Input data and hidden variables are 3rd-order tensors $shape(x) = (Height, Width, Depth)$.
- CONV/POOL are two typical kinds of layers

4.1 CONV layer

The Conv-layer feature map is defined as:

$$\boldsymbol{h}_{i,j,k}^{(t)} = \delta\left(\sum_l \left(\boldsymbol{W}_{::,l,k}^{(t)} * \boldsymbol{h}_{::,l}^{(t-1)}\right)_{i,j} + \boldsymbol{b}_k^{(t)}\right)$$

where $*$ is matrix-matrix conv-operator. For $\boldsymbol{P} \in \mathcal{R}^{n_p \times m_p}$ and $\boldsymbol{Q} \in \mathcal{R}^{n_q \times m_q}$, the conv-operator with stride s is defined as:

$$(\boldsymbol{P} * \boldsymbol{Q})_{ij}^{stride=s} = \sum_{a=0}^{n_p} \sum_{b=0}^{m_p} \boldsymbol{P}_{a,b} \boldsymbol{Q}_{si-a,sj-b}$$

Note in above equation, $si - a$ and $sj - b$ could go out of boundary of $n_q \times m_q$. In practice, people usually use three boundary definitions:

- **Hard boundary**: constrain $0 \leq si - a \leq n_q$ and $0 \leq si - b \leq m_q$.
- **Zero boundary**: filling 0 for out of boundary entrees of $\boldsymbol{Q}$
- **Periodic boundary**: $\boldsymbol{Q}_{i,j} = \boldsymbol{Q}_{i\%n_q, j\%m_q}$

Variables of layer t:

$$\begin{aligned}
\boldsymbol{h}^{(t)} &\in \mathcal{R}^{I^{(t)} \times J^{(t)} \times K^{(t)}} \\
\boldsymbol{h}^{(t-1)} &\in \mathcal{R}^{I^{(t-1)} \times J^{(t-1)} \times K^{(t-1)}} \\
\boldsymbol{W}^{(t)} &\in \mathcal{R}^{\hat{I}^{(t)} \times \hat{J}^{(t)} \times K^{(t-1)} \times K^{(t)}} \\
\boldsymbol{b}^{(t)} &\in \mathcal{R}^{K^{(t)}}
\end{aligned}$$

where $I^{(t-1)}, J^{(t-1)}, K^{(t-1)}$ are the height, width and depth of layer $t-1$ and the convolutional kernel $\boldsymbol{W}^{(t)}$ is used to map

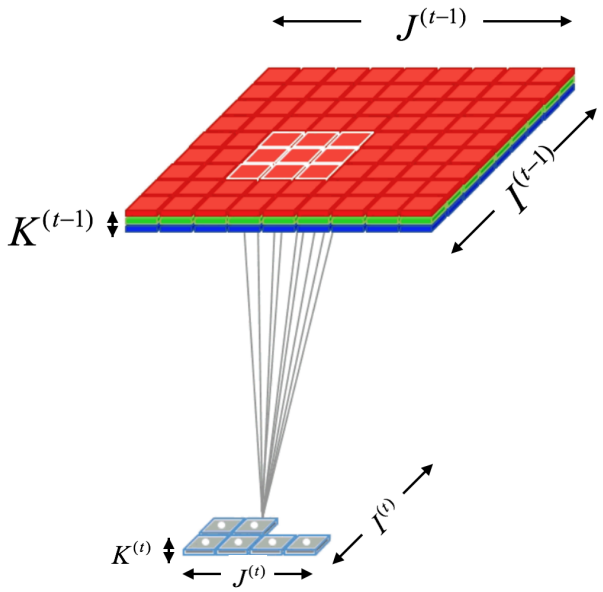


Figure 9: CNN: Conv layer with stride 2. $\boldsymbol{h}^{(t-1)} \in \mathcal{R}^{9 \times 9 \times 3}$, $\boldsymbol{h}^{(t)} \in \mathcal{R}^{4 \times 4 \times 1}$ and $\boldsymbol{K} \in \mathcal{R}^{3 \times 3 \times 3 \times 1}$.

4.2 POOL layer

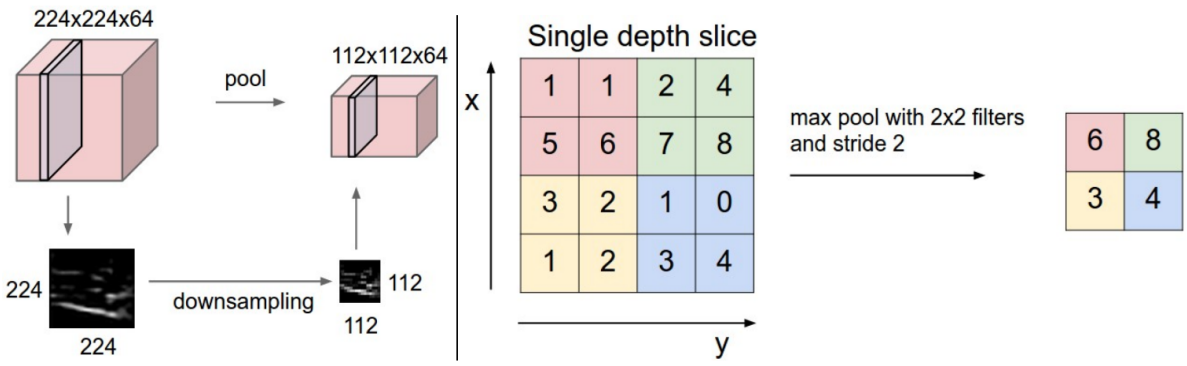


Figure 10: CNN: MAX POOL layer

- To reduce # of parameters, pooling layer is used
- POOL downsamples each depth slice of input feature tensor
- Example: max pooling with stride s:

$$\boldsymbol{h}_{i,j,k}^{(t)} = \max(\boldsymbol{h}_{si:(s+1)i,sj:(s+1)j,k}^{(t-1)})$$

- Max pooling layers have no parameters to learn

4.3 Forward Propagation

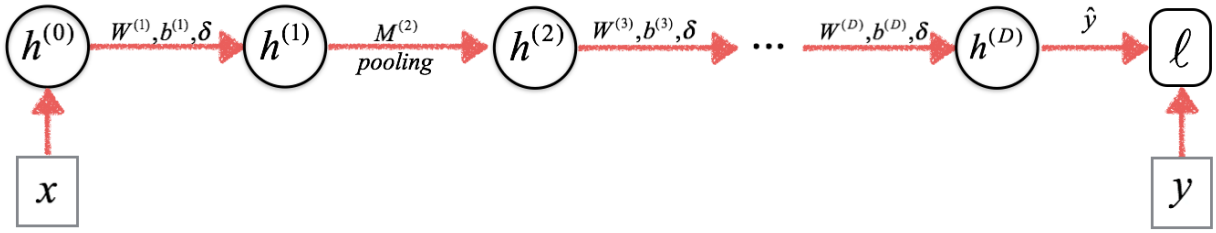


Figure 11: CNN forward propagation

- Evaluate convolutions for each CONV layer

$$\boldsymbol{h}_{i,j,k}^{(t)} = \delta(\sum_{l}(\boldsymbol{W}_{::,l,k}^{(t)} * \boldsymbol{h}_{::,l}^{(t-1)})_{i,j} + \boldsymbol{b}_{k}^{(t)})$$

- Evaluate pooling for each POOL layer

$$\boldsymbol{h}_{i,j,k}^{(t)} = \max(\boldsymbol{h}_{si:(s+1)i-1,sj:(s+1)j-1,k}^{(t-1)}))$$

4.4 Back Propagation

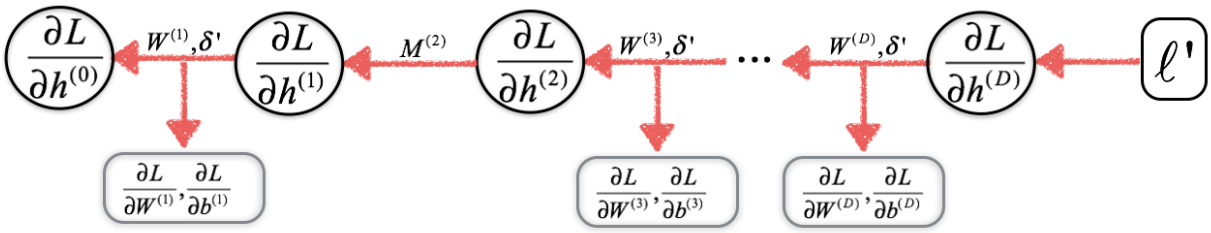


Figure 12: CNN back propagation

- Denoting $\boldsymbol{C}_{i,j,k} = \delta' \frac{dl}{d\boldsymbol{h}^{(t)}_{i,j,k}}$, we have:

$$\frac{dl}{d\boldsymbol{h}^{(t-1)}_{a,b,l}} = \sum_{i,j,k} \boldsymbol{C}_{i,j,k} \boldsymbol{W}^{(t)}_{i-a,j-b,l,k} = \sum_{k} \boldsymbol{C}_{::,k} * \boldsymbol{W}^{(t)}_{::,l,k}$$

$$\frac{dl}{d\boldsymbol{W}^{(t)}_{a,b,l,k}} = \sum_{i,j,k} \boldsymbol{C}_{i,j,k} \boldsymbol{h}^{(t-1)}_{i-a,j-b,l} = \sum_{l} \boldsymbol{C}_{::,k} * \boldsymbol{h}^{(t-1)}_{::,l}$$

- For POOL layer, the derivative only routing the gradient to the input that had the highest value in the forward pass.