

Deep Learning

Bojan Božić

TU Dublin

Summer 2021

- 1 Big Idea
- 2 Fundamentals
- 3 Standard Approach: Backpropagation and Gradient Descent
- 4 Extensions and Variations
- 5 Summary
- 6 Further Reading

Big Idea

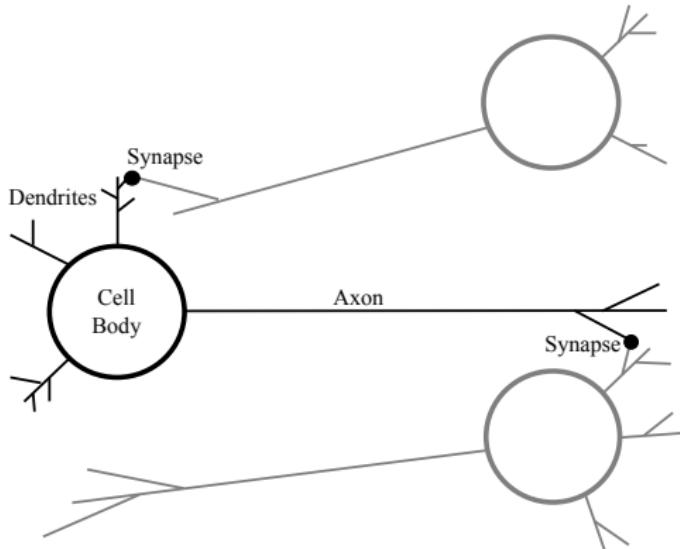


Figure 1: A high-level schematic of the structure of a neuron. This figure illustrates three interconnected neurons; the middle neuron is highlighted in black, and the major structural components of this neuron are labeled cell body, dendrites, and axon. Also marked are the synapses connecting the axon of one neuron and the dendrite of another, which allow signals to pass between the neurons.



Fundamentals

Artificial Neurons

$$z = \underbrace{\mathbf{w}[0] \times \mathbf{d}[0] + \mathbf{w}[1] \times \mathbf{d}[1] + \cdots + \mathbf{w}[m] \times \mathbf{d}[m]}_{\text{weighted sum}} \quad (1)$$

$$= \sum_{j=0}^m \mathbf{w}[j] \times \mathbf{d}[j]$$

$$= \underbrace{\mathbf{w} \cdot \mathbf{d}}_{\text{dot product}} = \underbrace{\mathbf{w}^T \mathbf{d}}_{\text{matrix product}} = [w_0, w_1, \dots, w_m] \begin{bmatrix} d_0 \\ d_2 \\ \vdots \\ d_m \end{bmatrix} \quad (2)$$



Artificial Neurons

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

$$\text{rectifier}(z) = \max(0, z) \quad (4)$$

Artificial Neurons

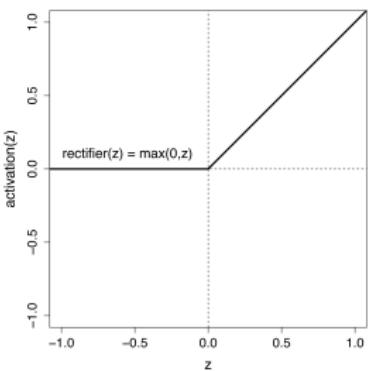
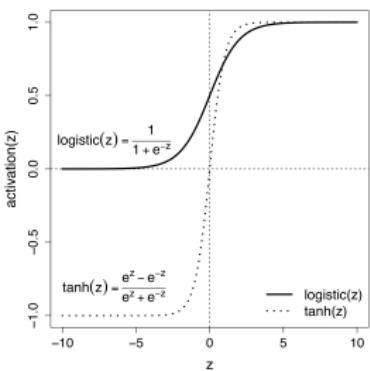
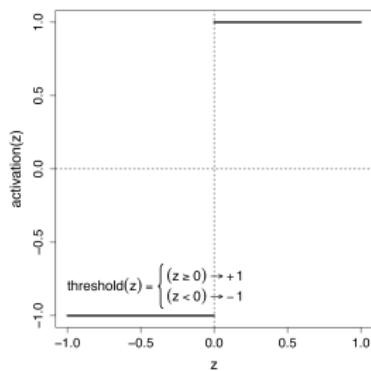


Figure 2: Plots for activation functions that have been popular in the history of neural networks.

$$\begin{aligned}
 M_{\mathbf{w}}(\mathbf{d}) &= \varphi(\mathbf{w}[0] \times \mathbf{d}[0] + \mathbf{w}[1] \times \mathbf{d}[1] + \cdots + \mathbf{w}[m] \times \mathbf{d}[m]) \\
 &= \varphi\left(\sum_{i=0}^m w_i \times d_i\right) = \varphi\left(\underbrace{\mathbf{w} \cdot \mathbf{d}}_{dot\ product}\right) \\
 &= \varphi\left(\underbrace{\mathbf{w}^T \mathbf{d}}_{matrix\ product}\right) = \varphi\left([w_0, w_1, \dots, w_m] \begin{bmatrix} d_0 \\ d_2 \\ \vdots \\ d_m \end{bmatrix}\right)
 \end{aligned} \tag{5}$$

Artificial Neurons

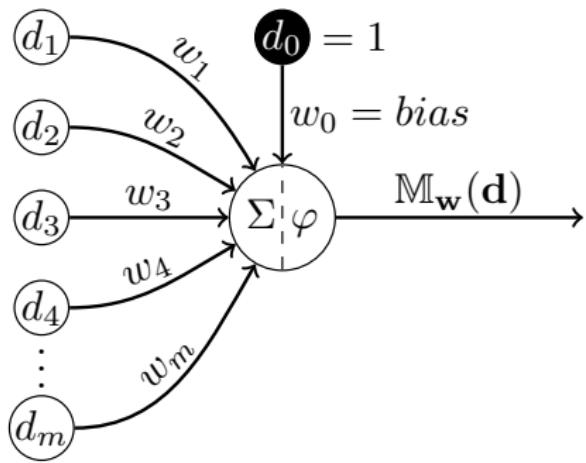


Figure 3: A schematic of an artificial neuron.

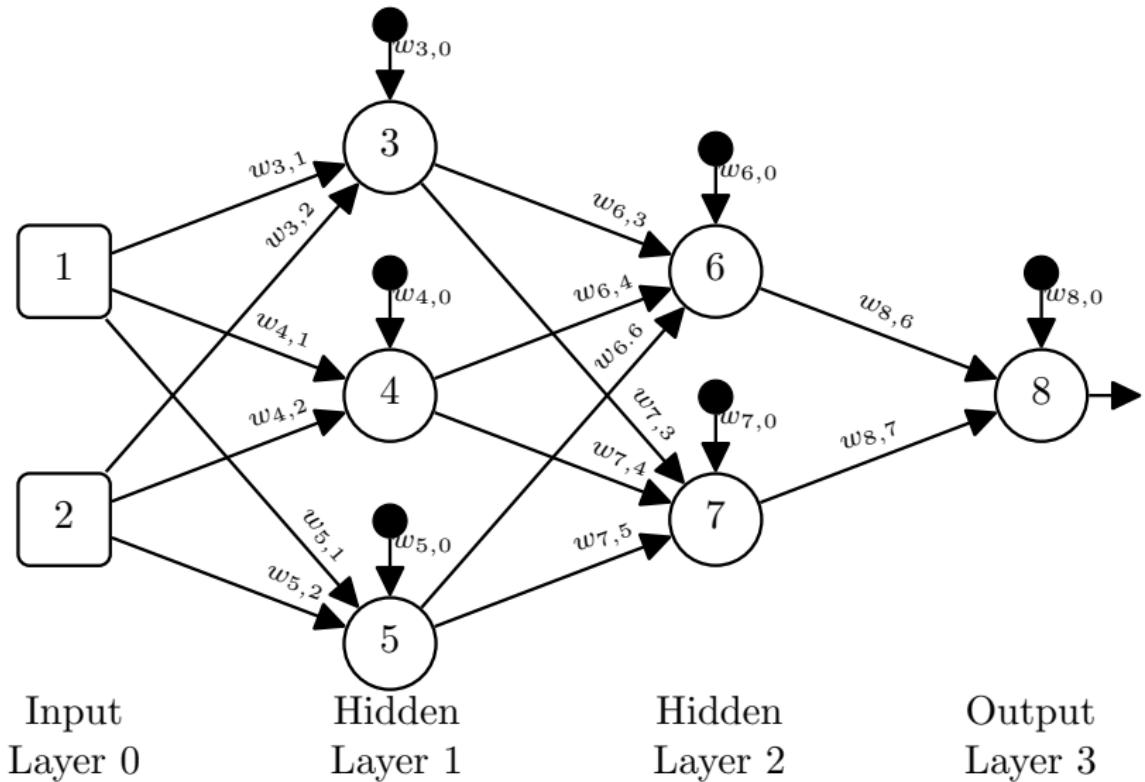


Figure 4: A schematic of a feedforward artificial neural network.

Neural Networks as Matrix Operations

$$\mathbf{z}^{(2)} = \mathbf{W}^{(2)} \mathbf{a}^{(1)} \quad (6)$$

Neural Networks as Matrix Operations

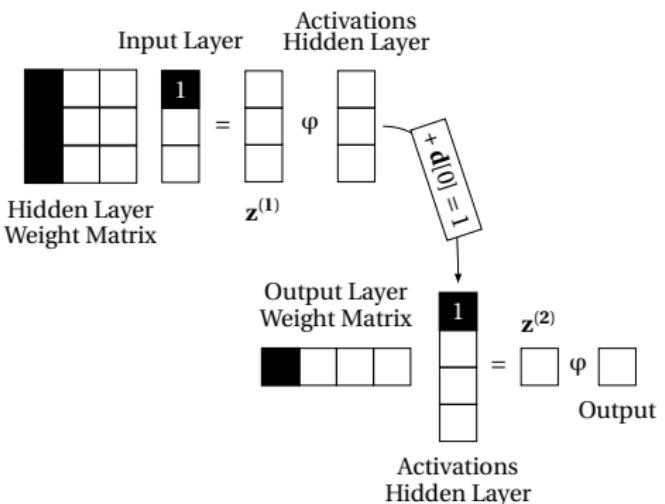
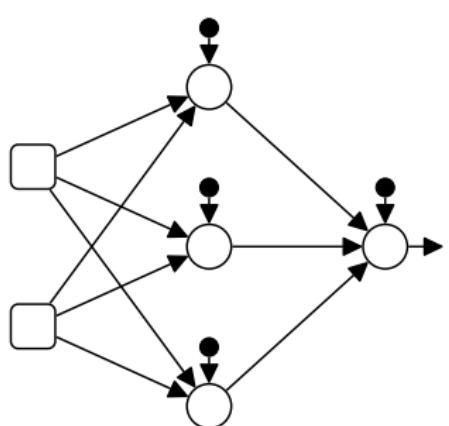


Figure 5: An illustration of the correspondence between graphical and matrix representations of a neural network. This figure is inspired by Figure 3.9 of (Kelleher, 2019).

Neural Networks as Matrix Operations

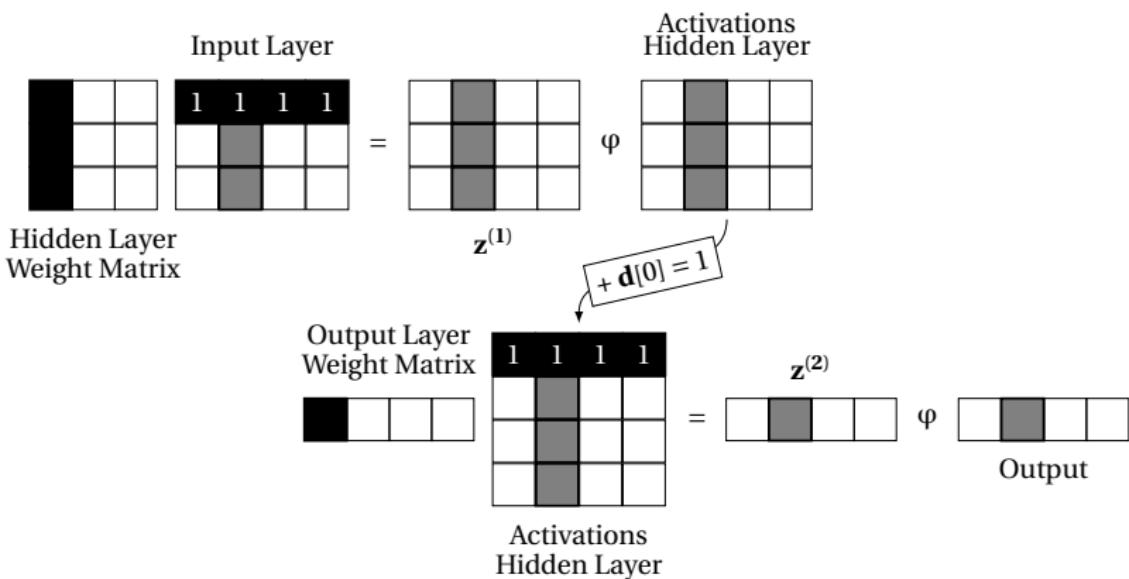


Figure 6: An illustration of how a batch of examples can be processed in parallel using matrix operations.

Why Are Non-Linear Activation Functions Necessary?

$$\mathbf{A}^{(1)} = \mathbf{W}^{(1)} \mathbf{A}^{(0)} \quad (7)$$

$$\mathbf{A}^{(2)} = \mathbf{W}^{(2)} \mathbf{A}^{(1)} \quad (8)$$

$$\mathbf{A}^{(2)} = \mathbf{W}^{(2)} \left(\mathbf{W}^{(1)} \mathbf{A}^{(0)} \right) \quad (9)$$

$$\mathbf{A}^{(2)} = \left(\mathbf{W}^{(2)} \mathbf{W}^{(1)} \right) \mathbf{A}^{(0)} \quad (10)$$

$$\mathbf{A}^{(2)} = \mathbf{W}' \mathbf{A}^{(0)} \quad (11)$$

Why Is Network Depth Important?

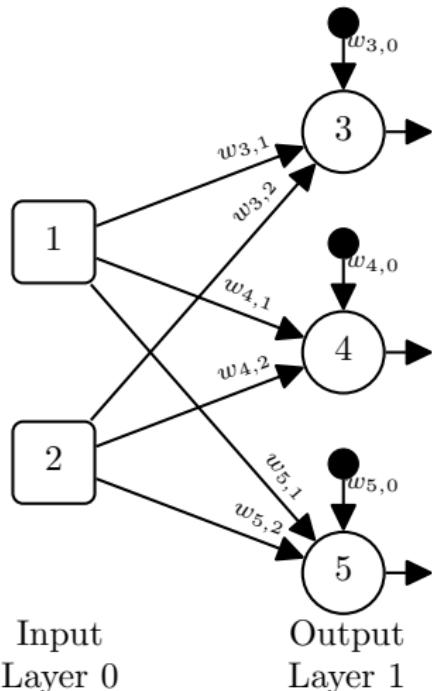


Figure 7: A single-layer network.

Why Is Network Depth Important?

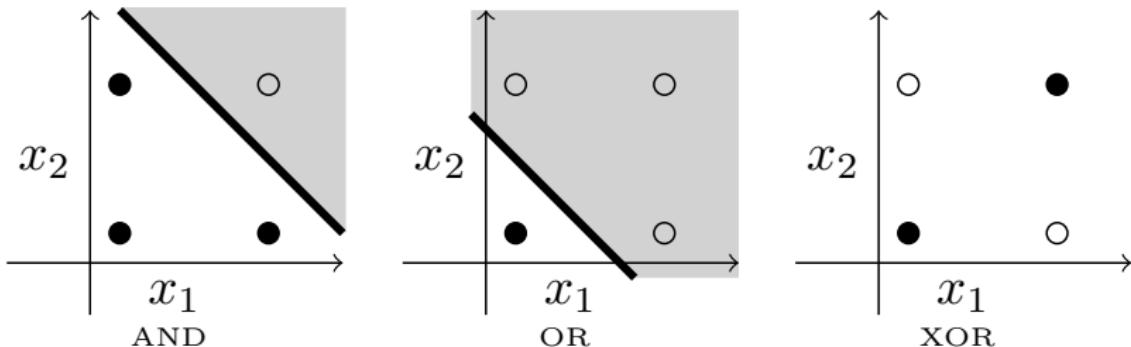


Figure 8: The logical AND and OR functions are linearly separable, but the XOR is not. This figure is Figure 4.2 of (Kelleher, 2019) and is used here with permission.



Why Is Network Depth Important?

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } z \geq 1 \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

Why Is Network Depth Important?

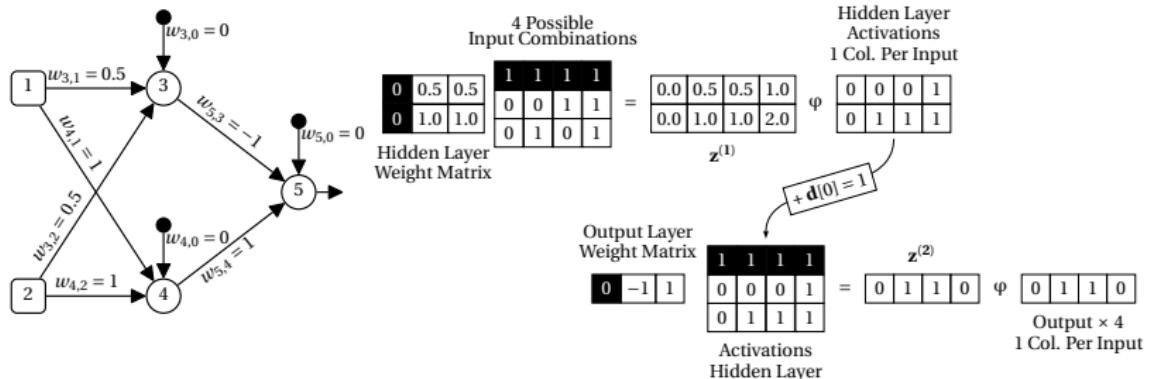


Figure 9: (left) The XOR function implemented as a two-layer neural network. (right) The network processing the four possible input combinations, one combination plus bias input per column: [bias, *FALSE*, *FALSE*] \rightarrow [1, 0, 0]; [bias, *FALSE*, *TRUE*] \rightarrow [1, 0, 1]; [bias, *TRUE*, *FALSE*] \rightarrow [1, 1, 0]; [bias, *TRUE*, *TRUE*] \rightarrow [1, 1, 1].

Why Is Network Depth Important?

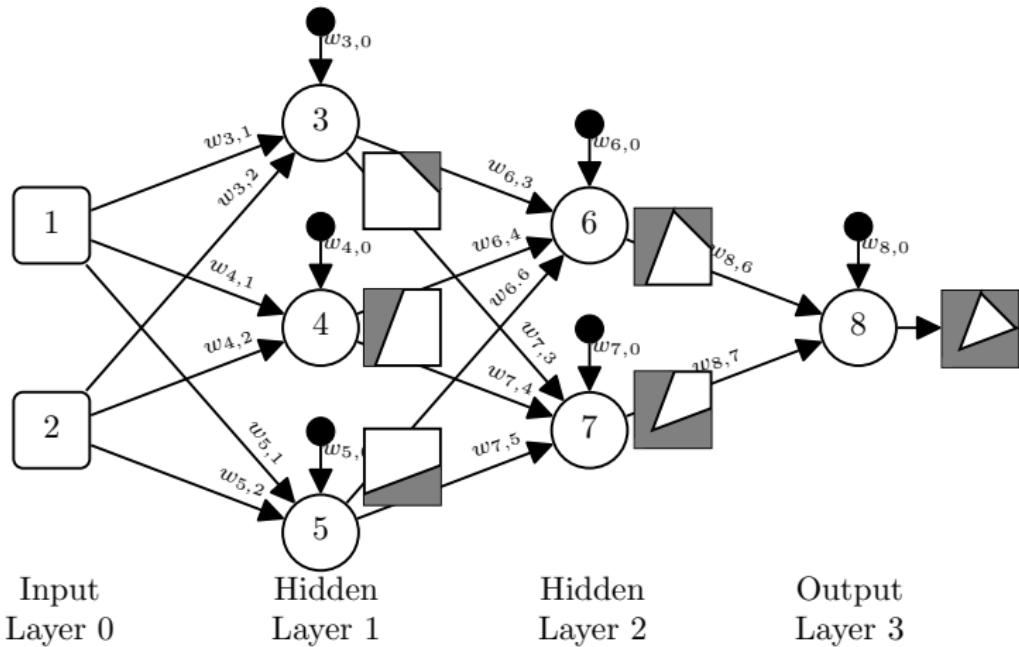
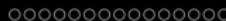


Figure 10: An illustration of how the representational capacity of a network increases as more layers are added to the network. This figure was inspired by Figure 4.2 in (Reed and Marks, 1999) and Figure 3.10 in (Marsland, 2011).



Standard Approach: Backpropagation and Gradient Descent

Backpropagation: The General Structure of the Algorithm

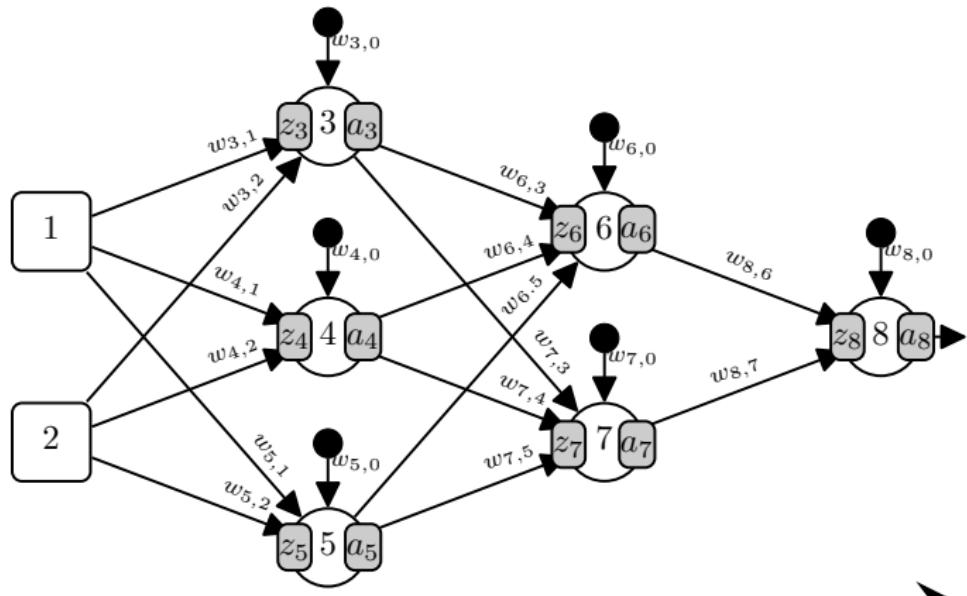


Figure 11: The calculation of the z values and activations of each neuron during the forward pass of the backpropagation algorithm. This figure is based on Figure 6.5 of (Kelleher, 2019).

Backpropagation: The General Structure of the Algorithm

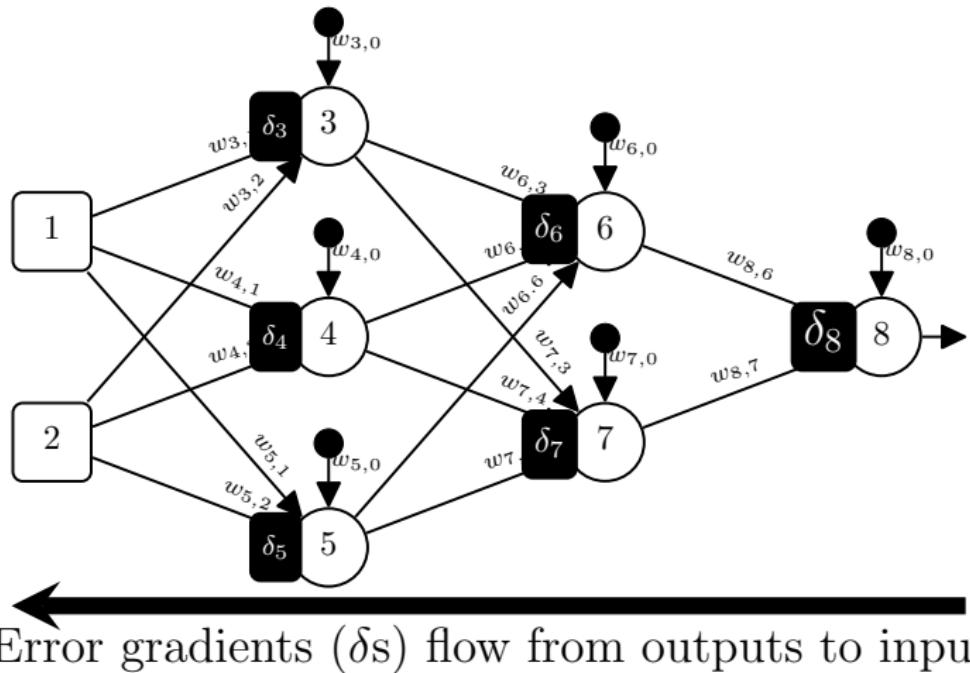


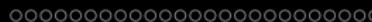
Figure 12: The backpropagation of the δ values during the backward pass of the backpropagation algorithm. This figure is based on Figure 6.6 of (Kelleher, 2019).



Backpropagation: Backpropagating the Error Gradients

$$\delta_k = \frac{\partial \mathcal{E}}{\partial z_k} \quad (13)$$

$$\delta_k = \frac{\partial a_k}{\partial z_k} \times \frac{\partial \mathcal{E}}{\partial a_k} \quad (14)$$



Backpropagation: Backpropagating the Error Gradients

$$\frac{d}{dz} \text{logistic}(z) = \text{logistic}(z) \times (1 - \text{logistic}(z)) \quad (15)$$

$$\begin{aligned} \frac{d}{dz} \text{logistic}(z = 0) &= \text{logistic}(0) \times (1 - \text{logistic}(0)) \\ &= 0.5 \times (1 - 0.5) \\ &= 0.25 \end{aligned} \quad (16)$$

Backpropagation: Backpropagating the Error Gradients

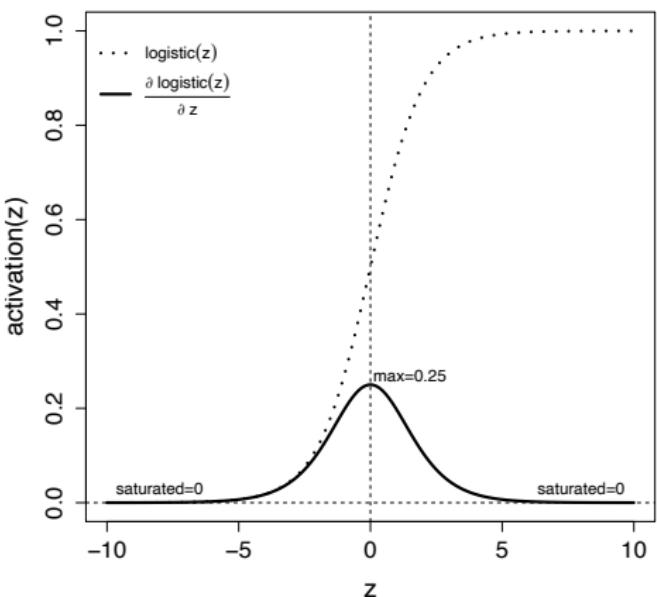


Figure 13: Plots of the logistic function and its derivative. This figure is Figure 4.6 of (Kelleher, 2019) and is used here with permission.

Backpropagation: Backpropagating the Error Gradients

$$L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{2} \sum_{i=1}^n (t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i))^2 \quad (17)$$

$$\frac{\partial}{\partial \mathbf{w}[j]} L_2(\mathbb{M}_{\mathbf{w}}, \mathbf{d}) = (t - \mathbb{M}_{\mathbf{w}}(\mathbf{d})) \times -\mathbf{d}[j] \quad (18)$$

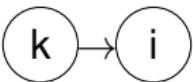
$$\frac{\partial \mathcal{E}}{\partial a_k} = \frac{\partial L_2(\mathbb{M}_{\mathbf{w}}, \mathbf{d})}{\partial \mathbb{M}_{\mathbf{w}}(\mathbf{d})} = t - \mathbb{M}_{\mathbf{w}}(\mathbf{d}) = t_k - a_k \quad (19)$$

$$\frac{\partial \mathcal{E}}{\partial a_k} = -(t_k - a_k) \quad (20)$$

Backpropagation: Backpropagating the Error Gradients

$$\begin{aligned}\delta_k &= \frac{\partial a_k}{\partial z_k} \times \frac{\partial \mathcal{E}}{\partial a_k} \\ &= \frac{\partial a_k}{\partial z_k} \times -(t_k - a_k) \\ &= \underbrace{\frac{d}{dz} \text{logistic}(z)}_{\text{Assuming a logistic activation function}} \times -(t_k - a_k) \\ &= \underbrace{(\text{logistic}(z) \times (1 - \text{logistic}(z)))}_{\text{Assuming a logistic activation function}} \times -(t_k - a_k) \quad (21)\end{aligned}$$

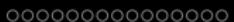
Backpropagation: Backpropagating the Error Gradients



$$\frac{\partial \mathcal{E}}{\partial a_k} = \sum_{i=1}^n w_{i,k} \times \delta_i \quad (22)$$

$$\begin{aligned}
 \delta_k &= \frac{\partial a_k}{\partial z_k} \times \frac{\partial \mathcal{E}}{\partial a_k} \\
 &= \frac{\partial a_k}{\partial z_k} \times \left(\sum_{i=1}^n w_{i,k} \times \delta_i \right) \\
 &= \underbrace{\frac{d}{dz} \text{logistic}(z)}_{\text{Assuming a logistic activation function}} \times \left(\sum_{i=1}^n w_{i,k} \times \delta_i \right)
 \end{aligned}$$

$$\underbrace{(\text{logistic}(z) \times (1 - \text{logistic}(z)))}_{\text{Assuming a logistic activation function}} \times \left(\sum_{i=1}^n w_{i,k} \times \delta_i \right) \quad (23)$$



Backpropagation: Updating the Weights in a Network

$$\frac{\partial \mathcal{E}}{\partial w_{i,k}} = \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i} \times \frac{\partial z_i}{\partial w_{i,k}} \quad (24)$$

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial w_{i,k}} &= \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i} \times \frac{\partial z_i}{\partial w_{i,k}} \\ &= \boldsymbol{\delta}_i \times \frac{\partial z_i}{\partial w_{i,k}} \end{aligned} \quad (25)$$

$$\frac{\partial z_i}{\partial w_{i,k}} = a_k \quad (26)$$



Backpropagation: Updating the Weights in a Network

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial w_{i,k}} &= \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i} \times \frac{\partial z_i}{\partial w_{i,k}} \\ &= \boldsymbol{\delta}_i \times \frac{\partial z_i}{\partial w_{i,k}} \\ &= \boldsymbol{\delta}_i \times a_k\end{aligned}\tag{27}$$



Backpropagation: Updating the Weights in a Network

$$w_{i,k} \leftarrow w_{i,k} - \alpha \times \delta_i \times a_k \quad (28)$$



Backpropagation: Updating the Weights in a Network

$$\Delta w_{i,k} = \sum_{j=1}^m \delta_{i,j} \times a_{k,j} \quad (29)$$

$$w_{i,k} \leftarrow w_{i,k} - \alpha \times \Delta w_{i,k} \quad (30)$$



Backpropagation: The Algorithm

Require: set of training instances \mathcal{D}

Require: a learning rate α that controls how quickly the algorithm converges

Require: a batch size B specifying the number of examples in each batch

Require: a convergence criterion

```

1: Shuffle  $\mathcal{D}$  and create the mini-batches:  $[(\mathbf{X}^{(1)}, \mathbf{Y}^{(1)}), \dots, (\mathbf{X}^k, \mathbf{Y}^k)]$ 
2: Initialize the weight matrices for each layer:  $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}$ 
3: repeat ▷ Each repeat loop is one epoch
4:   for  $t=1$  to number of mini-batches do ▷ Each for loop is one iteration
5:      $\mathbf{A}^{(0)} \leftarrow \mathbf{X}^{(t)}$ 
6:     for  $l=1$  to  $L$  do
7:        $\mathbf{v} \leftarrow [1_0, \dots 1_m]$  ▷ Create  $\mathbf{v}$  the vector of bias terms
8:        $\mathbf{A}^{(l-1)} \leftarrow [\mathbf{v}; \mathbf{A}^{(l-1)}]$  ▷ Insert  $\mathbf{v}$  into the activation matrix
9:        $\mathbf{Z}^{(l)} \leftarrow \mathbf{W}^l \mathbf{A}^{(l-1)}$ 
10:       $\mathbf{A}^{(l)} \leftarrow \varphi(\mathbf{Z}^{(l)})$  ▷ Elementwise application of  $\varphi$  to  $\mathbf{Z}^{(l)}$ 
11:    end for
12:    for each weight  $w_{i,k}$  in the network do
13:       $\Delta w_{i,k} = 0$ 
14:    end for
15:    for each example in the mini-batch do ▷ Backpropagate the  $\delta$ s
16:      for each neuron  $i$  in the output layer do
17:         $\delta_i = \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i}$  ▷ See Equation (21)[28]
18:      end for
19:      for  $l = L-1$  to  $1$  do
20:        for each neuron  $i$  in the layer  $l$  do
21:           $\delta_i = \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i}$  ▷ See Equation (23)[29]
22:        end for
23:      end for
24:      for each weight  $w_{i,k}$  in the network do
25:         $\Delta w_{i,k} = \Delta w_{i,k} + (\delta_i \times a_k)$  ▷ Equation (29)[33]
26:      end for
27:    end for
28:    for each weight  $w_{i,k}$  in the network do
29:       $w_{i,k} \leftarrow w_{i,k} - \alpha \times \Delta w_{i,k}$  ▷ Equation (30)[33]
30:    end for
31:  end for
32:  shuffle([( $\mathbf{X}^{(1)}, \mathbf{Y}^{(1)}$ ),  $\dots$ , ( $\mathbf{X}^k, \mathbf{Y}^k$ )])
33: until convergence occurs

```

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

Table 1: Hourly samples of ambient factors and full load electrical power output of a combined cycle power plant.

ID	AMBIENT TEMPERATURE °C	RELATIVE HUMIDITY %	ELECTRICAL OUTPUT MW
1	03.21	86.34	491.35
2	31.41	68.50	430.37
3	19.31	30.59	463.00
4	20.64	99.97	447.14



A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

Table 2: The minimum and maximum values for the AMBIENT TEMPERATURE, RELATIVE HUMIDITY, and ELECTRICAL OUTPUT features in the power plant dataset.

	AMBIENT TEMPERATURE	RELATIVE HUMIDITY	ELECTRICAL OUTPUT
Min	1.81°C	25.56%	420.26MW
Max	37.11°C	100.16%	495.76MW

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

Table 3: The *range-normalized* hourly samples of ambient factors and full load electrical power output of a combined cycle power plant, rounded to two decimal places.

ID	AMBIENT TEMPERATURE °C	RELATIVE HUMIDITY %	ELECTRICAL OUTPUT MW
1	0.04	0.81	0.94
2	0.84	0.58	0.13
3	0.50	0.07	0.57
4	0.53	1.00	0.36

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

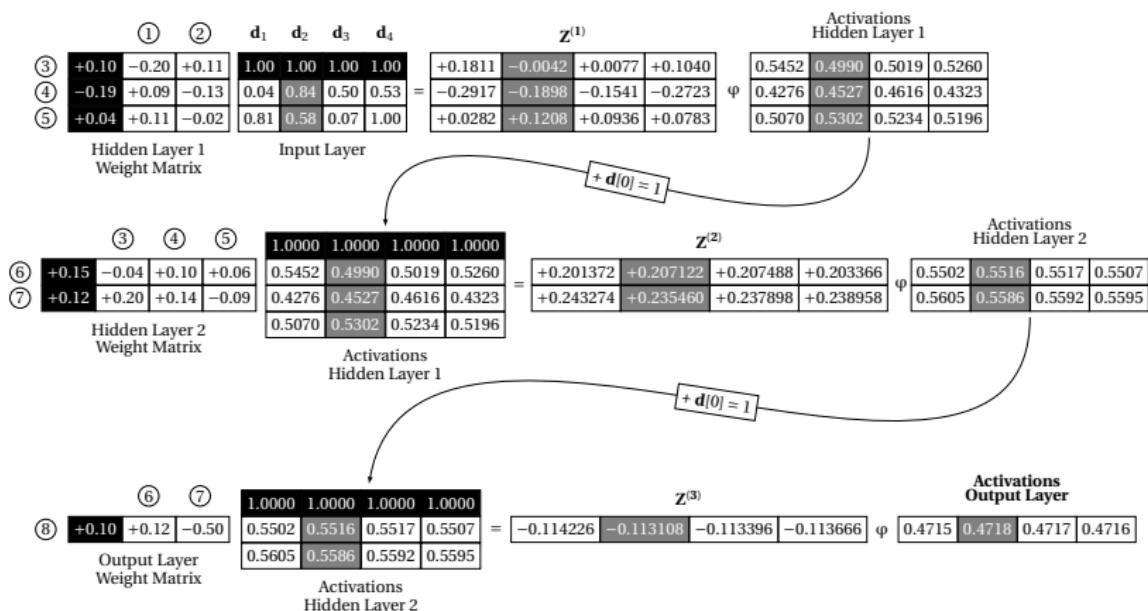


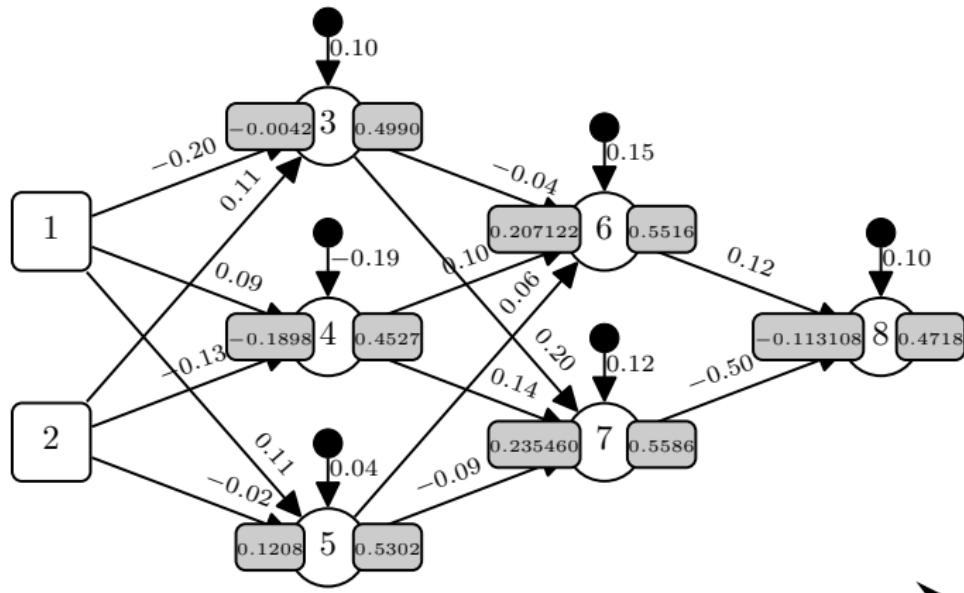
Figure 14: The forward pass of the examples listed in Table 3^[38] through the network in Figure 4^[11].

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

Table 4: The per example error after the forward pass illustrated in Figure 14^[39], the per example $\partial E / \partial a_8$, and the **sum of squared errors** for the model over the dataset of four examples.

	d ₁	d ₂	d ₃	d ₄
Target	0.9400	0.1300	0.5700	0.3600
Prediction	0.4715	0.4718	0.4717	0.4716
Error	0.4685	-0.3418	0.0983	-0.1116
$\partial E / \partial a_8$: Error $\times -1$	-0.4685	0.3418	-0.0983	0.1116
Error ²	0.21949225	0.11682724	0.00966289	0.01245456
SSE:				0.17921847

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task



Activations flow from inputs to outputs

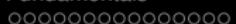
Figure 15: An illustration of the forward propagation of d_2 through the network showing the weights on each connection, and the weighted sum z and activation a value for each neuron in the network.



A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

$$\frac{\partial \mathcal{E}}{\partial a} = 0.3418 \quad (31)$$

$$\begin{aligned}\delta_8 &= \frac{\partial \mathcal{E}}{\partial a_8} \times \frac{\partial a_8}{\partial z_8} \\ &= 0.3418 \times 0.2492 \\ &= 0.0852\end{aligned} \quad (32)$$



A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

Table 5: The $\partial a / \partial z$ for each neuron for Example 2 rounded to four decimal places.

NEURON	z	$\partial a / \partial z$
3	-0.004200	0.2500
4	-0.189800	0.2478
5	0.120800	0.2491
6	0.207122	0.2473
7	0.235460	0.2466
8	-0.113108	0.2492

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

$$\begin{aligned}\delta_6 &= \frac{\partial \mathcal{E}}{\partial a_6} \times \frac{\partial a_6}{\partial z_6} \\ &= \left(\sum \delta_i \times w_{i,6} \right) \times \frac{\partial a_6}{\partial z_6} \\ &= (\delta_8 \times w_{8,6}) \times \frac{\partial a_6}{\partial z_6} \\ &= (0.0852 \times 0.12) \times 0.2473 \\ &= 0.0025\end{aligned}\tag{33}$$

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

$$\begin{aligned}\delta_7 &= \frac{\partial \mathcal{E}}{\partial a_7} \times \frac{\partial a_7}{\partial z_7} \\&= \left(\sum \delta_i \times w_{i,7} \right) \times \frac{\partial a_7}{\partial z_7} \\&= (\delta_8 \times w_{8,7}) \times \frac{\partial a_6}{\partial z_6} \\&= (0.0852 \times -0.50) \times 0.2466 \\&= -0.0105\end{aligned}\tag{34}$$



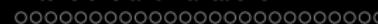
A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

$$\begin{aligned}\delta_3 &= \frac{\partial \mathcal{E}}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \\&= \left(\sum \delta_i \times w_{i,3} \right) \times \frac{\partial a_3}{\partial z_3} \\&= ((\delta_6 \times w_{6,3}) + (\delta_7 \times w_{7,3})) \times \frac{\partial a_3}{\partial z_3} \\&= ((0.0025 \times -0.04) + (-0.0105 \times 0.20)) \times 0.2500 \\&= -0.0006 \quad (35)\end{aligned}$$



A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

$$\begin{aligned}\delta_4 &= \frac{\partial \mathcal{E}}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \\&= \left(\sum \delta_i \times w_{i,4} \right) \times \frac{\partial a_4}{\partial z_4} \\&= ((\delta_6 \times w_{6,4}) + (\delta_7 \times w_{7,4})) \times \frac{\partial a_4}{\partial z_4} \\&= ((0.0025 \times 0.10) + (-0.0105 \times 0.14)) \times 0.2478 \\&= -0.0003 \quad (36)\end{aligned}$$



A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

$$\begin{aligned}\delta_5 &= \frac{\partial \mathcal{E}}{\partial a_5} \times \frac{\partial a_5}{\partial z_5} \\&= \left(\sum \delta_i \times w_{i,5} \right) \times \frac{\partial a_5}{\partial z_5} \\&= ((\delta_6 \times w_{6,5}) + (\delta_7 \times w_{7,5})) \times \frac{\partial a_5}{\partial z_5} \\&= ((0.0025 \times 0.06) + (-0.0105 \times -0.09)) \times 0.2491 \\&= 0.0003 \quad (37)\end{aligned}$$

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

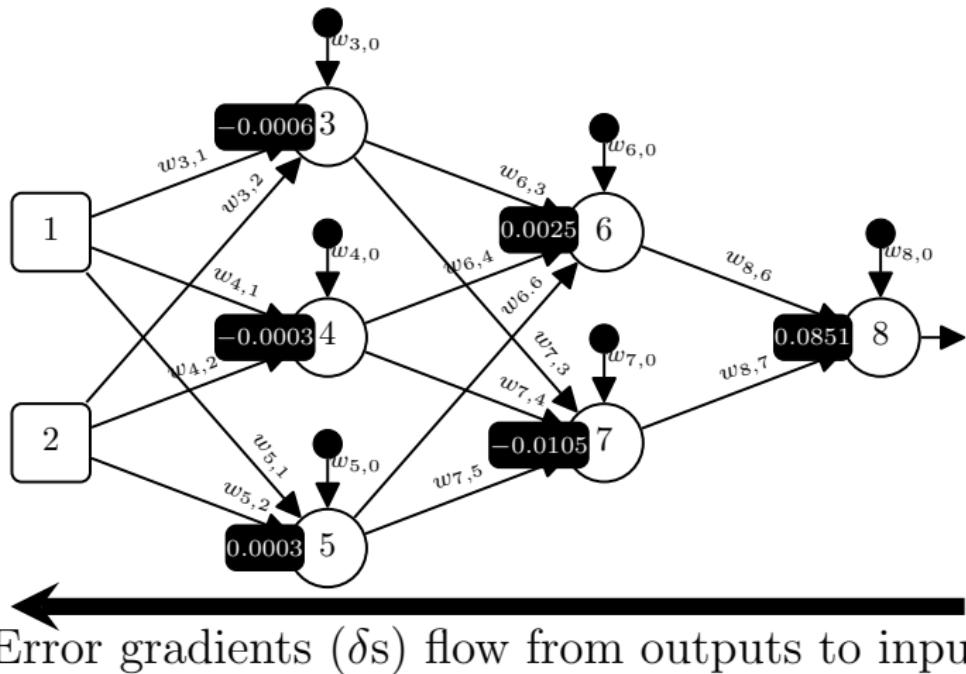


Figure 16: The δ_s for each of the neurons in the network for Ex. 2.

Table 6: The $\partial\mathcal{E}/\partial w_{i,k}$ calculations for \mathbf{d}_2 for every weight in the network. The neuron index 0 denotes the bias input for each neuron.

NEURON _i	NEURON _k	$w_{i,k}$	δ_i	a_k	$\partial\mathcal{E}/\partial w_{i,k}$
8	0	$w_{8,0}$	0.0852	1	$0.0852 \times 1 = 0.0852$
8	6	$w_{8,6}$	0.0852	0.5516	$0.0852 \times 0.5516 = 0.04699632$
8	7	$w_{8,7}$	0.0852	0.5586	$0.0852 \times 0.5586 = 0.04759272$
7	0	$w_{7,0}$	-0.0105	1	$-0.0105 \times 1 = -0.0105$
7	3	$w_{7,3}$	-0.0105	0.4990	$-0.0105 \times 0.4527 = -0.0052395$
7	4	$w_{7,4}$	-0.0105	0.4527	$-0.0105 \times 0.4527 = -0.00475335$
7	5	$w_{7,5}$	-0.0105	0.5302	$-0.0105 \times 0.5302 = -0.0055671$
6	0	$w_{6,0}$	0.0025	1	$0.0025 \times 1 = 0.0025$
6	3	$w_{6,3}$	0.0025	0.4990	$0.0025 \times 0.4527 = 0.0012475$
6	4	$w_{6,4}$	0.0025	0.4527	$0.0025 \times 0.4527 = 0.00113175$
6	5	$w_{6,5}$	0.0025	0.5302	$0.0025 \times 0.5302 = 0.0013255$
5	0	$w_{5,0}$	0.0003	1	$0.0003 \times 1 = 0.0003$
5	1	$w_{5,1}$	0.0003	0.84	$0.0003 \times 0.84 = 0.000252$
5	2	$w_{5,2}$	0.0003	0.58	$0.0003 \times 0.58 = 0.000174$
4	0	$w_{4,0}$	-0.0003	1	$-0.0003 \times 1 = -0.0003$
4	1	$w_{4,1}$	-0.0003	0.84	$-0.0003 \times 0.84 = -0.000252$
4	2	$w_{4,2}$	-0.0003	0.58	$-0.0003 \times 0.58 = -0.000174$
3	0	$w_{3,0}$	-0.0006	1	$-0.0006 \times 1 = -0.0006$
3	1	$w_{3,1}$	-0.0006	0.84	$-0.0006 \times 0.84 = -0.000504$
3	2	$w_{3,2}$	-0.0006	0.58	$-0.0006 \times 0.58 = -0.000348$

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

$$\begin{aligned} w_{7,5} &= w_{7,5} - \alpha \times \delta_7 \times a_5 \\ &= w_{7,5} - \alpha \times \frac{\partial \mathcal{E}}{\partial w_{i,k}} \\ &= -0.09 - 0.2 \times -0.0055671 \\ &= -0.08888658 \end{aligned} \tag{38}$$

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

Table 7: The calculation of $\Delta w_{7,5}$ across our four examples.

MINI-BATCH EXAMPLE	$\frac{\partial \mathcal{E}}{\partial w_{7,5}}$
\mathbf{d}_1	0.00730080
\mathbf{d}_2	-0.00556710
\mathbf{d}_3	0.00157020
\mathbf{d}_4	-0.00176664
$\Delta w_{7,5} =$	0.00153726



A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

$$\begin{aligned} w_{7,5} &= w_{7,5} - \alpha \times \Delta w_{i,k} \\ &= -0.09 - 0.2 \times 0.00153726 \\ &= -0.0903074520 \end{aligned} \tag{39}$$

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

Table 8: The per example error after each weight has been updated once, the per example $\partial E / \partial a_8$, and the **sum of squared errors** for the model.

	d ₁	d ₂	d ₃	d ₄
Target	0.9400	0.1300	0.5700	0.3600
Prediction	0.4738	0.4741	0.4740	0.4739
Error	0.4662	-0.3441	0.0960	-0.1139
$\partial E / \partial a_8$: Error $\times -1$	-0.4662	0.3441	-0.0960	0.1139
Error ²	0.21734244	0.11840481	0.009216	0.01297321
SSE:				0.17896823

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

Table 9: The per example prediction, error, and the sum of squared errors after training has converged to an $SSE < 0.0001$.

	d_1	d_2	d_2	d_2
Target	0.9400	0.1300	0.5700	0.3600
Prediction	0.9266	0.1342	0.5700	0.3608
Error	0.0134	-0.0042	0.0000	-0.0008
Error ²	0.00017956	0.00001764	0.00000000	0.00000064
SSE:				0.00009892

A Worked Example: Using Backpropagation to Train a Feedforward Network for a Regression Task

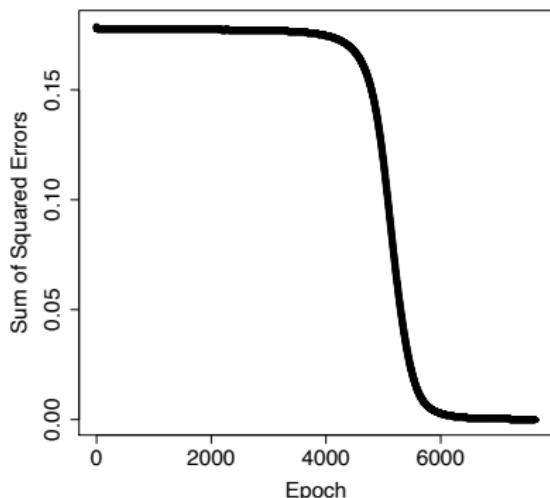


Figure 17: A plot showing how the sum of squared errors of the network changed during training.

Big Idea

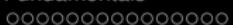
Fundamentals

Standard Approach: Backpropagation and Gradient Descent

Extensions and Variations



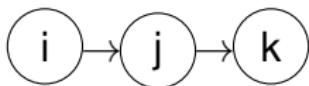
Extensions and Variations



Vanishing Gradients and ReLUs

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial w_{i,k}} &= \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i} \times \frac{\partial z_i}{\partial w_{i,k}} \\ &= \boldsymbol{\delta}_i \times \frac{\partial z_i}{\partial w_{i,k}}\end{aligned}\tag{40}$$

Vanishing Gradients and ReLUs



$$\begin{aligned}
 \delta_i &= \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i} \\
 &= \overbrace{w_{j,i} \times \delta_j}^{\delta_j} \times \frac{\partial a_i}{\partial z_i} \\
 &= w_{j,i} \times w_{k,j} \times \overbrace{\delta_k \times \frac{\partial a_j}{\partial z_j} \times \frac{\partial a_i}{\partial z_i}}^{\delta_k} \\
 &= w_{j,i} \times w_{k,j} \times \overbrace{\frac{\partial \mathcal{E}}{\partial a_k} \times \frac{\partial a_k}{\partial z_k}}^{\frac{\partial \mathcal{E}}{\partial a_k}} \times \frac{\partial a_j}{\partial z_j} \times \frac{\partial a_i}{\partial z_i}
 \end{aligned} \tag{41}$$

Vanishing Gradients and ReLUs

$$\text{rectifier}(z) = \max(0, z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \quad (42)$$

$$\frac{d}{dz} \text{rectifier}(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \quad (43)$$

Vanishing Gradients and ReLUs

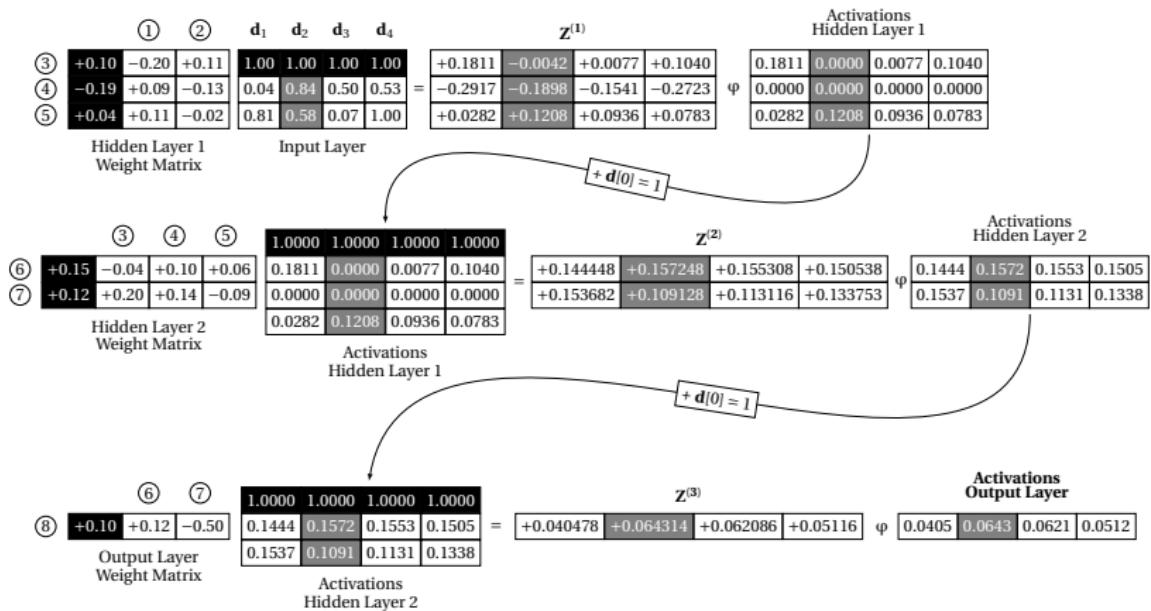
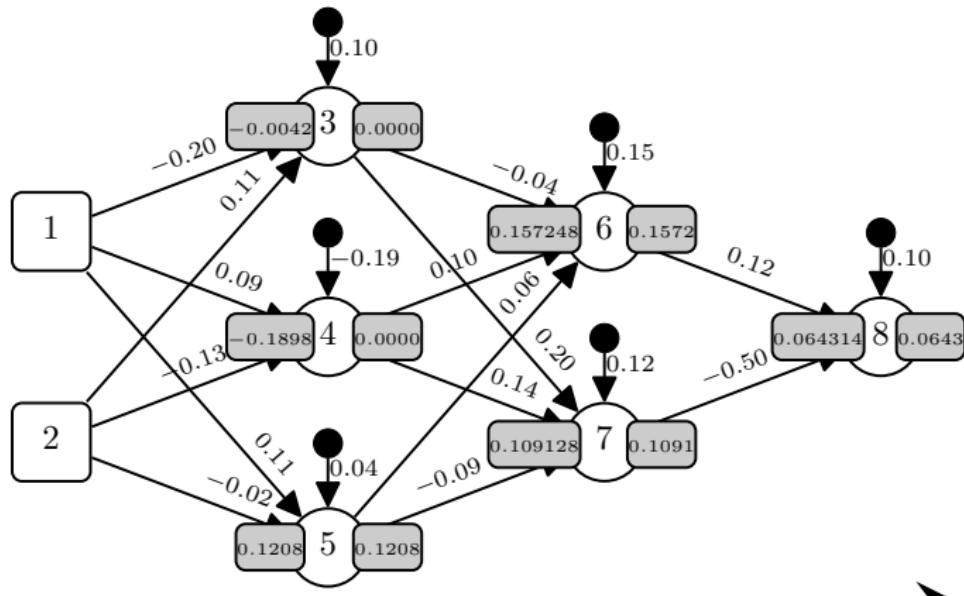


Figure 18: The forward pass of the examples listed in Table 3^[38] through the network in Figure 4^[11] when all the neurons are ReLUs.

Vanishing Gradients and ReLUs



Activations flow from inputs to outputs

Figure 19: An illustration of the forward propagation of d_2 through the ReLU network showing the weights on each connection, and the weighted sum z and activation a value for each neuron in the network.

Table 10: The per example error of the ReLU network after the forward pass illustrated in Figure 18^[61], the per example $\partial\mathcal{E}/\partial a_8$, and the **sum of squared errors** for the ReLU model.

	d ₁	d ₂	d ₃	d ₄
Target	0.9400	0.1300	0.5700	0.3600
Prediction	0.0405	0.0643	0.0621	0.0512
Error	0.8995	0.0657	0.5079	0.3088
$\partial\mathcal{E}/\partial a_8$: Error $\times -1$	-0.8995	-0.0657	-0.5079	-0.3088
Error ²	0.80910025	0.00431649	0.25796241	0.09535744
SSE:				0.58336829

Vanishing Gradients and ReLUs

Table 11: The $\partial a / \partial z$ for each neuron for d_2 rounded to four decimal places.

NEURON	z	$\partial a / \partial z$
3	-0.004200	0
4	-0.189800	0
5	0.120800	1
6	0.157248	1
7	0.109128	1
8	0.064314	1

$$\begin{aligned}
\delta_k &= \frac{\partial \mathcal{E}}{\partial a_k} \times \frac{\partial a_k}{\partial z_i} \\
\delta_8 &= -0.0657 \times 1.0 \\
&= -0.0657 \\
\delta_7 &= (\delta_8 \times w_{8,7}) \times \frac{\partial a_7}{\partial z_7} \\
&= (-0.0657 \times -0.50) \times 1 \\
&= 0.0329 \\
\delta_6 &= (\delta_8 \times w_{8,6}) \times \frac{\partial a_6}{\partial z_6} \\
&= (-0.0657 \times 0.12) \times 1 \\
&= -0.0079 \\
\delta_5 &= ((\delta_6 \times w_{6,5}) + (\delta_7 \times w_{7,5})) \times \frac{\partial a_5}{\partial z_5} \\
&= ((-0.0079 \times 0.06) + (0.0329 \times -0.09)) \times 1 \\
&= -0.0034 \\
\delta_4 &= ((\delta_6 \times w_{6,4}) + (\delta_7 \times w_{7,4})) \times \frac{\partial a_4}{\partial z_4} \\
&= ((-0.0079 \times 0.10) + (0.0329 \times 0.14)) \times 0 \\
&= 0 \\
\delta_3 &= ((\delta_6 \times w_{6,3}) + (\delta_7 \times w_{7,3})) \times \frac{\partial a_3}{\partial z_3} \\
&= ((-0.0079 \times -0.04) + (0.0329 \times 0.20)) \times 0 \\
&= 0
\end{aligned}$$

(44)

Vanishing Gradients and ReLUs

Table 12: The ReLU network's per example prediction, error, and the sum of squared errors after training has converged to an $SSE < 0.0001$.

	d_1	d_2	d_3	d_4
Target	0.9400	0.1300	0.5700	0.3600
Prediction	0.9487	0.1328	0.5772	0.3679
Error	-0.0087	-0.0028	-0.0072	-0.0079
Error ²	0.00007569	0.00000784	0.00005184	0.00006241
SSE:				0.00009889

Vanishing Gradients and ReLUs

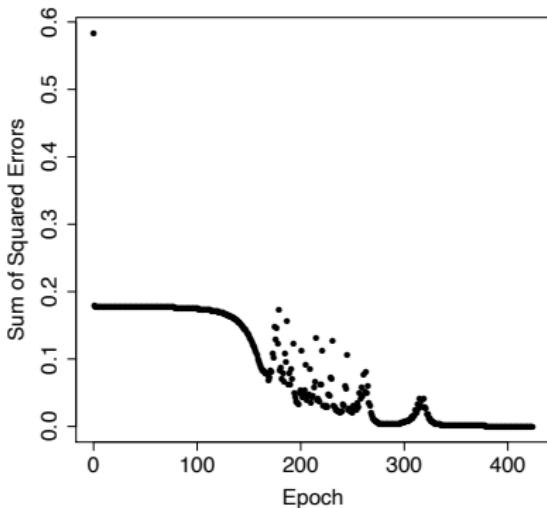


Figure 20: A plot showing how the sum of squared errors of the ReLU network changed during training when $\alpha = 0.2$.

Vanishing Gradients and ReLUs

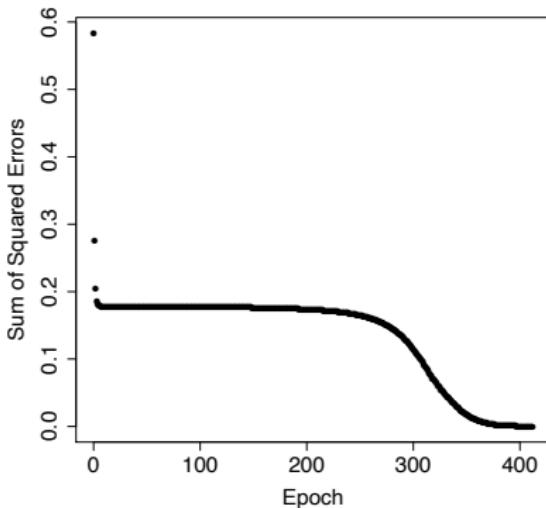


Figure 21: A plot showing how the sum of squared errors of the ReLU network changed during training when $\alpha = 0.1$.

Vanishing Gradients and ReLUs

$$\text{rectifierleaky}(z) = \begin{cases} z & \text{if } z > 0 \\ 0.01 \times z & \text{otherwise} \end{cases} \quad (45)$$

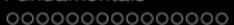
$$\frac{d}{dz} \text{rectifierleaky}(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0.01 & \text{otherwise} \end{cases} \quad (46)$$



Vanishing Gradients and ReLUs

$$\text{rectifier}_{\text{parametric}}(z_i) = \begin{cases} z_i & \text{if } z_i > 0 \\ \lambda_i \times z_i & \text{otherwise} \end{cases} \quad (47)$$

$$\frac{d}{dz} \text{rectifier}_{\text{parametric}}(z_i) = \begin{cases} 1 & \text{if } z_i > 0 \\ \lambda_i & \text{otherwise} \end{cases} \quad (48)$$



Vanishing Gradients and ReLUs

$$\frac{\partial \mathcal{E}}{\partial \lambda_i} = \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial \lambda_i} \quad (49)$$

$$\frac{\partial a_i}{\partial \lambda_i} = \begin{cases} 0 & \text{if } z_i > 0 \\ z_i & \text{otherwise} \end{cases} \quad (50)$$

$$\lambda_i \leftarrow \lambda_i - \alpha \times \frac{\partial \mathcal{E}}{\partial \lambda_i} \quad (51)$$

Weight Initialization and Unstable Gradients

$$\delta_i = \underbrace{w_{j,i} \times w_{k,j} \times}_{\substack{\text{extreme weights} \\ \rightarrow \text{unstable gradients}}} \underbrace{\frac{\partial \mathcal{E}}{\partial a_k} \times \frac{\partial a_k}{\partial z_k} \times \frac{\partial a_j}{\partial z_j} \times \frac{\partial a_i}{\partial z_i}}_{\substack{\text{extreme weights} \\ \rightarrow \text{saturated activations} \\ \rightarrow \text{vanishing gradients}}} \quad (52)$$

Weight Initialization and Unstable Gradients

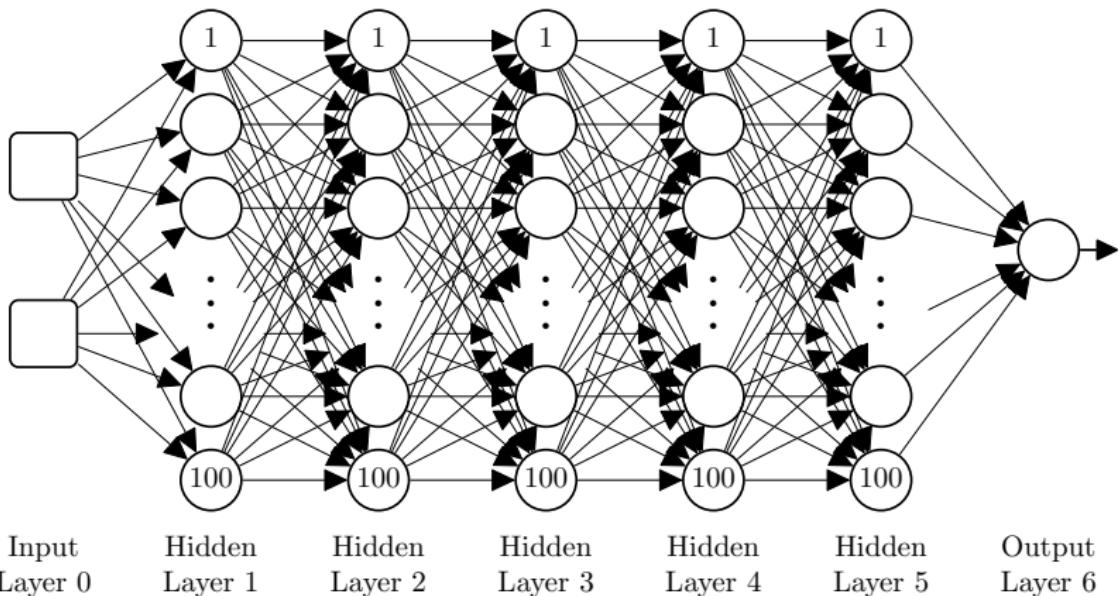
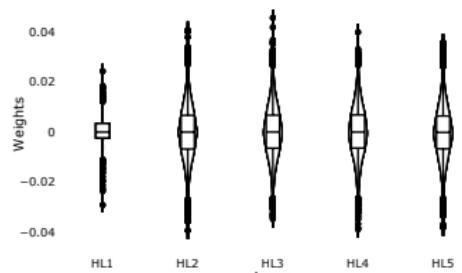
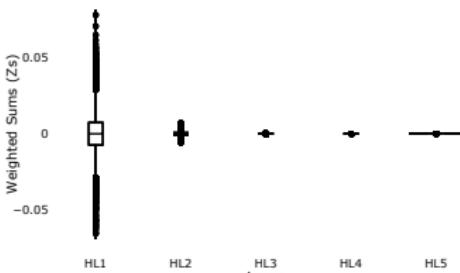
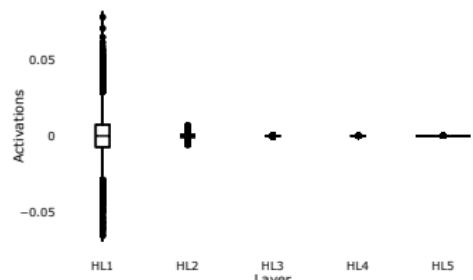


Figure 22: The architecture of the neural network used in the weight initialization experiments. Note that the neurons in this network use a linear activation function: $a_i = z_i$.

Weight Initialization and Unstable Gradients



(a) Weights by Layer

(b) Weighted Sum (z) by Layer

(c) Activations by Layer

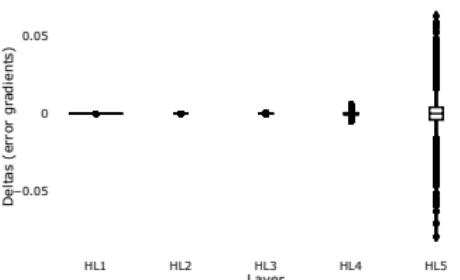
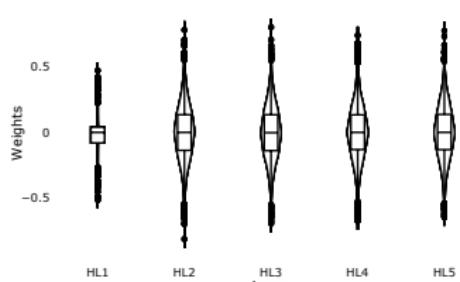
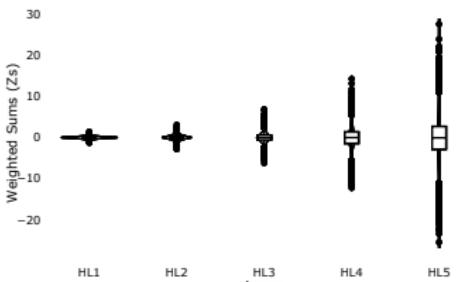
(d) δ_s by Layer

Figure 23: The internal dynamics of the network in Figure 22^[73] during the first training iteration when the weights were initialized using a normal distribution with $\mu=0.0$, $\sigma=0.01$.

Weight Initialization and Unstable Gradients



(a) Weights by Layer

(b) Weighted Sum (z) by Layer

(c) Activations by Layer

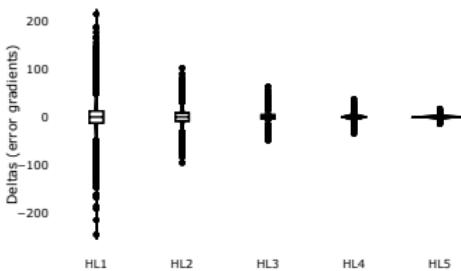
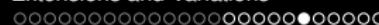
(d) δ s by Layer

Figure 24: The internal dynamics of the network in Figure 22^[73] during the first training iteration when the weights were initialized using a normal distribution with $\mu = 0.0$ and $\sigma = 0.2$.



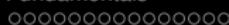
Weight Initialization and Unstable Gradients

$$z = (w_1 \times d_1) + (w_2 \times d_2) + \cdots + (w_{n_{in}} \times d_{n_{in}}) \quad (53)$$



Weight Initialization and Unstable Gradients

$$\text{var} \left(\sum_{i=1}^n X_i \right) = \sum_{i=1}^n \text{var} (X_i) \quad (54)$$



Weight Initialization and Unstable Gradients

$$\begin{aligned} \text{var}(z) &= \text{var}((w_1 \times d_1) + (w_2 \times d_2) + \dots (w_{n_{in}} \times d_{n_{in}})) \\ &= \sum_{i=1}^{n_{in}} \text{var}(w_i \times d_i) \end{aligned} \tag{55}$$

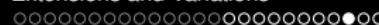
$$\text{var}(w \times d) = [E(\mathbf{W})]^2 \text{var}(\mathbf{d}) + [E(\mathbf{d})]^2 \text{var}(\mathbf{W}) + \text{var}(\mathbf{W}) \text{var}(\mathbf{d}) \tag{56}$$

$$\text{var}(w \times d) = \text{var}(\mathbf{W}) \text{var}(\mathbf{d}) \tag{57}$$



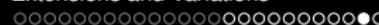
Weight Initialization and Unstable Gradients

$$\text{var}(z) = \sum_{i=1}^{n_{in}} \text{var}(w_i \times d_i) = n_{in} \text{var}(\mathbf{W}) \text{var}(\mathbf{d}) \quad (58)$$



Weight Initialization and Unstable Gradients

$$\begin{aligned} var(Z^{(HL1)}) &= n_{in}^{(HL1)} \times var(\mathbf{W}^{(HL1)}) \times var(\mathbf{d}^{(HL1)}) \quad (59) \\ &= 2 \times 0.0001 \times 1 \\ &= 0.0002 \end{aligned}$$



Weight Initialization and Unstable Gradients

$$\begin{aligned} \text{var}(Z^{(HL2)}) &= n_{in}^{(HL2)} \times \text{var}(\mathbf{W}^{(HL2)}) \times \text{var}(\mathbf{d}^{(HL2)}) \quad (60) \\ &= 100 \times 0.0001 \times 0.0002 \\ &= 0.000002 \end{aligned}$$

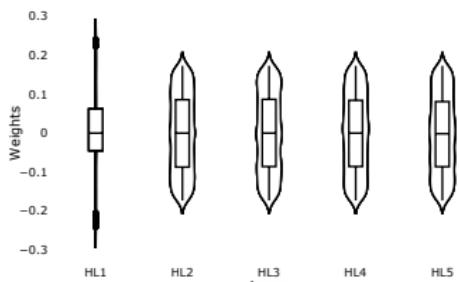


Weight Initialization and Unstable Gradients

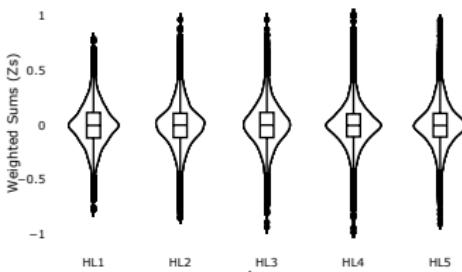
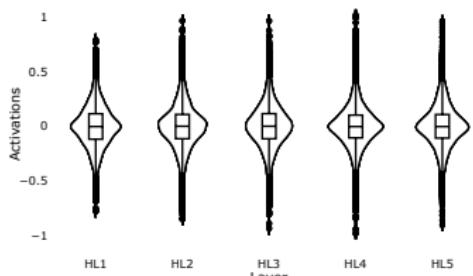
$$\text{var}(\mathbf{W}^{(k)}) = \frac{2}{n_{in}^{(k)} + n_{out}^{(k)}} \quad (61)$$

$$\text{var}(\mathbf{W}^{(k)}) = \frac{1}{n_{in}^{(k)}} \quad (62)$$

Weight Initialization and Unstable Gradients



(a) Weights by Layer

(b) Weighted Sum (z) by Layer

(c) Activations by Layer

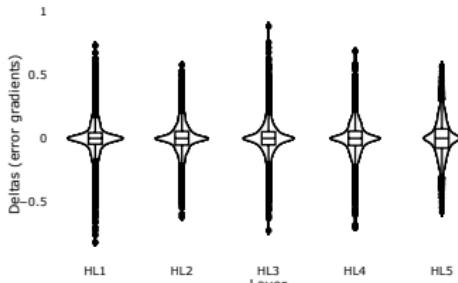
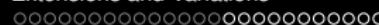
(d) δ s by Layer

Figure 25: The internal dynamics of the network in Figure 22^[73] during the first training iteration when the weights were initialized using Xavier initialization.



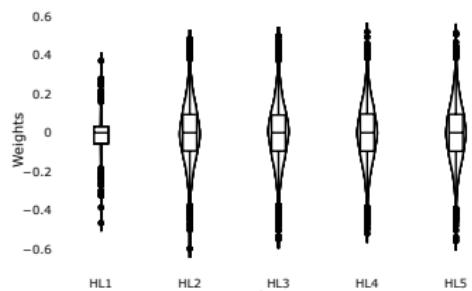
Weight Initialization and Unstable Gradients

$$var(\mathbf{W}^{(k)}) = \frac{2}{n_{in}^{(k)}} \quad (63)$$

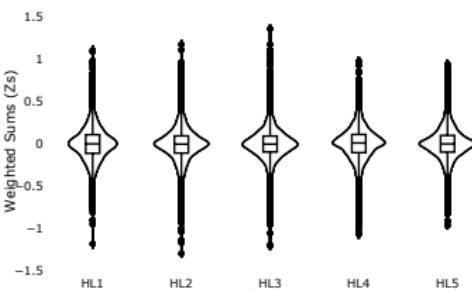
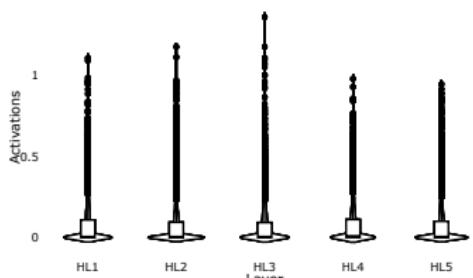
Weight Initialization and Unstable Gradients

$$\begin{aligned} W^{(1)} &\sim \mathcal{N}\left(0, \sqrt{\frac{1}{100}}\right) \\ W^{(2)} &\sim \mathcal{N}\left(0, \sqrt{\frac{2}{80}}\right) \\ W^{(3)} &\sim \mathcal{N}\left(0, \sqrt{\frac{2}{50}}\right) \end{aligned} \tag{64}$$

Weight Initialization and Unstable Gradients



(a) Weights by Layer

(b) Weighted Sum (z) by Layer

(c) Activations by Layer

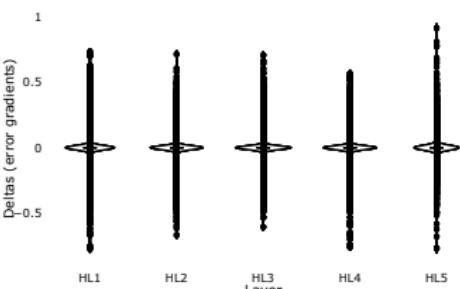
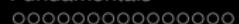
(d) δ s by Layer

Figure 26: The internal dynamics of the network in Figure 22^[73], using ReLUs, during the first training iteration when the weights were initialized using He initialization.



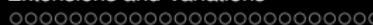
Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

- ➊ represent the target feature using **one-hot encoding**;
- ➋ change the output layer of the network to be a **softmax layer**; and
- ➌ change the error (or loss) function we use for training to be the **cross-entropy** function.

Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

Table 13: The *range-normalized* hourly samples of ambient factors and full load electrical power output of a combined cycle power plant, rounded to two decimal places, and with the (binned) target feature represented using one-hot encoding.

ID	AMBIENT TEMPERATURE °C	RELATIVE HUMIDITY %	Electrical Output		
			low	medium	high
1	0.04	0.81	0	0	1
2	0.84	0.58	1	0	0
3	0.50	0.07	0	1	0
4	0.53	1.00	0	1	0



Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$\varphi_{sm}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^m e^{z_m}} \quad (65)$$

Table 14: The calculation of the softmax activation function φ_{sm} over a vector of three logits \mathbf{l} .

	\mathbf{l}_0	\mathbf{l}_1	\mathbf{l}_2
\mathbf{l}	1.5	-0.9	0.6
$e^{\mathbf{l}_i}$	4.48168907	0.40656966	1.8221188
$\sum_i e^{\mathbf{l}_i}$			6.71037753
$\varphi_{sm}(\mathbf{l}_i)$	0.667874356	0.060588195	0.27153745

Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

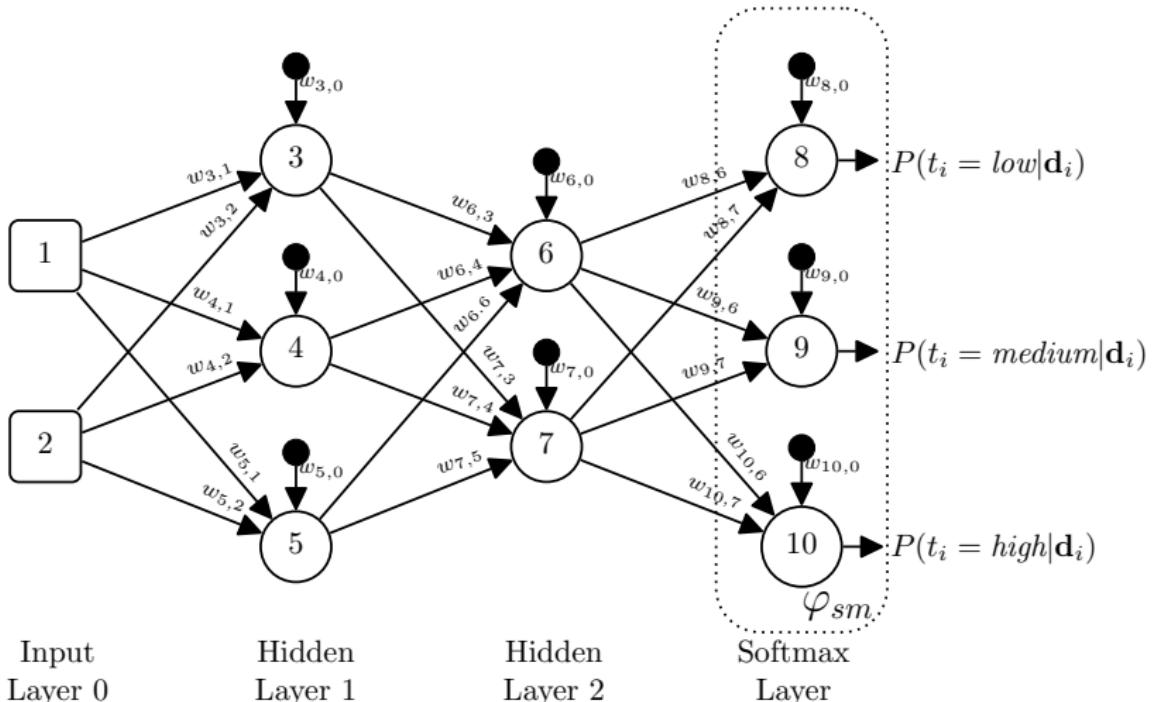


Figure 27: A schematic of a feedforward artificial neural network with a three-neuron softmax output layer.

Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$L_{CE}(\mathbf{t}, \hat{\mathbf{P}}) = - \sum_j \mathbf{t}_j \ln (\hat{\mathbf{P}}_j) \quad (66)$$

$$L_{CE}(\mathbf{t}, \hat{\mathbf{P}}) = - \ln (\hat{\mathbf{P}}_\star) \quad (67)$$



Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$\begin{aligned} L_{CE}(\mathbf{t}, \hat{\mathbf{P}}) &= - \sum_j \mathbf{t}_j \ln(\hat{\mathbf{P}}_j) \\ &= - \left((\mathbf{t}_0 \ln(\hat{\mathbf{P}}_0)) + (\mathbf{t}_1 \ln(\hat{\mathbf{P}}_1)) + (\mathbf{t}_2 \ln(\hat{\mathbf{P}}_2)) \right) \\ &= - \left((0 \ln(\hat{\mathbf{P}}_0)) + (1 \ln(\hat{\mathbf{P}}_1)) + (0 \ln(\hat{\mathbf{P}}_2)) \right) \\ &= -1 \ln(\hat{\mathbf{P}}_1) \end{aligned} \tag{68}$$



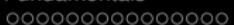
Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$\delta_k = \frac{\partial \mathcal{E}}{\partial z_k} \quad (69)$$

$$= \frac{\partial L_{CE} (\mathbf{t}, \hat{\mathbf{P}})}{\partial \mathbf{l}_k} \quad (70)$$

$$= \frac{\partial -\ln(\hat{\mathbf{P}}_*)}{\partial \mathbf{l}_k} \quad (71)$$

$$= \frac{\partial -\ln(\hat{\mathbf{P}}_*)}{\partial (\hat{\mathbf{P}}_*)} \times \frac{\partial (\hat{\mathbf{P}}_*)}{\partial \mathbf{l}_k} \quad (72)$$



Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$\frac{d \ln x}{dx} = \frac{1}{x} \quad (73)$$

$$\frac{\partial -\ln(\hat{\mathbf{P}}_\star)}{\partial(\hat{\mathbf{P}}_\star)} = -\frac{1}{\hat{\mathbf{P}}_\star} \quad (74)$$

$$\frac{\partial(\hat{\mathbf{P}}_\star)}{\partial l_k} = \begin{cases} \hat{\mathbf{P}}_\star(1 - \hat{\mathbf{P}}_k) & \text{if } k = \star \\ -\hat{\mathbf{P}}_\star\hat{\mathbf{P}}_k & \text{otherwise} \end{cases} \quad (75)$$

Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$\delta_k = \frac{\partial -\ln(\hat{P}_*)}{\partial(\hat{P}_*)} \times \frac{\partial(\hat{P}_*)}{\partial l_k} \quad (76)$$

$$= -\frac{1}{\hat{P}_*} \times \frac{\partial(\hat{P}_*)}{\partial l_k} \quad (77)$$

$$= -\frac{1}{\hat{P}_*} \times \begin{cases} \hat{P}_*(1 - \hat{P}_k) & \text{if } k = * \\ -\hat{P}_*\hat{P}_k & \text{otherwise} \end{cases} \quad (78)$$

$$= \begin{cases} -(1 - \hat{P}_k) & \text{if } k = * \\ \hat{P}_k & \text{otherwise} \end{cases} \quad (79)$$

Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$\delta_{k=\star} = - \left(1 - \hat{\mathbf{P}}_k \right) \quad (80)$$

$$\delta_{k \neq \star} = \hat{\mathbf{P}}_k \quad (81)$$

Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

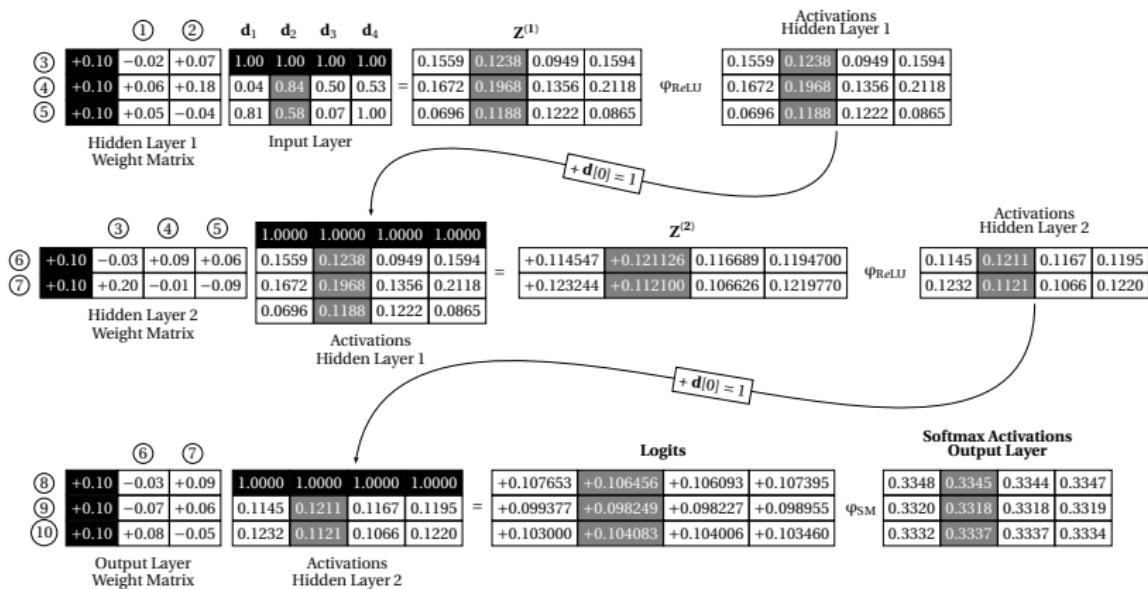
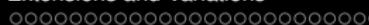
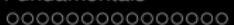


Figure 28: The forward pass of the mini-batch of examples listed in Table 13^[88] through the network in Figure 27^[91].

Table 15: The calculation of the softmax activations for each of the neurons in the output layer for each example in the mini-batch, and the calculation of the δ for each neuron in the output layer for each example in the mini-batch.

	d_1	d_2	d_3	d_4
Per Neuron Per Example logits				
Neuron 8	0.107653	0.106456	0.106093	0.107395
Neuron 9	0.099377	0.098249	0.098227	0.098955
Neuron 10	0.103000	0.104083	0.1040060	0.103460
Per Neuron Per Example e^{l_i}				
Neuron 8	1.113661238	1.112328983	1.111925281	1.11337395
Neuron 9	1.104482611	1.103237457	1.103213186	1.104016618
Neuron 10	1.108491409	1.109692556	1.109607113	1.109001432
$\sum_i e^{l_i}$	3.326635258	3.325258996	3.324745579	3.326392
Per Neuron Per Example Softmax Activations				
Neuron 8	0.3348	0.3345	0.3344	0.3347
Neuron 9	0.3320	0.3318	0.3318	0.3319
Neuron 10	0.3332	0.3337	0.3337	0.3334
Per Neuron Target One-Hot Encodings				
Neuron 8	0	1	0	0
Neuron 9	0	0	1	1
Neuron 10	1	0	0	0
Per Neuron Per Example δ s				
Neuron 8	0.3348	-0.6655	0.3344	0.3347
Neuron 9	0.3320	0.3318	-0.6682	-0.6681
Neuron 10	-0.6668	0.3337	0.3337	0.3334



Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$\begin{aligned}\Delta w_{9,6} &= \sum_{j=1}^4 \delta_{9,j} \times a_{6,j} \\&= (0.3320 \times 0.1145) + (0.3318 \times 0.1211) \\&\quad + (-0.6682 \times 0.1167) + (-0.6681 \times 0.1195) \\&= 0.038014 + 0.04018098 + -0.07797894 + -0.07983795 \\&= -0.07962191\end{aligned}\tag{82}$$

Handling Categorical Target Features: Softmax Output Layers and Cross-Entropy Loss Functions

$$\begin{aligned} w_{9,6} &= w_{9,6} - \alpha \times \Delta w_{9,6} \\ &= -0.07 - 0.01 \times -0.07962191 \\ &= -0.07 - (-0.000796219) \\ &= -0.069203781 \end{aligned} \tag{83}$$

Algorithm 1 The early stopping algorithm

Require: p the patience parameter

Require: \mathcal{D}_v a validation set

```
1: bestValidationError =  $\infty$ 
2: tmpValidationError = 0
3:  $\theta$  = initial model parameters
4:  $\theta^{best}$  = 0
5: patienceCount = 0
6: while patienceCount <  $p$  do
7:    $\theta$  = new model parameters after most recent weight update
8:   tmpValidationError = calculateValidationError( $\theta$ ,  $\mathcal{D}_v$ )
9:   if bestValidationError  $\geq$  tmpValidationError then
10:    bestValidationError = tmpValidationError
11:     $\theta^{best}$  =  $\theta$ 
12:    patienceCount = 0
13:   else
14:     patienceCount = patienceCount + 1
15:   end if
16: end while
17: return Best Model Parameters  $\theta^{best}$ 
```

Early Stopping and Dropout: Preventing Overfitting

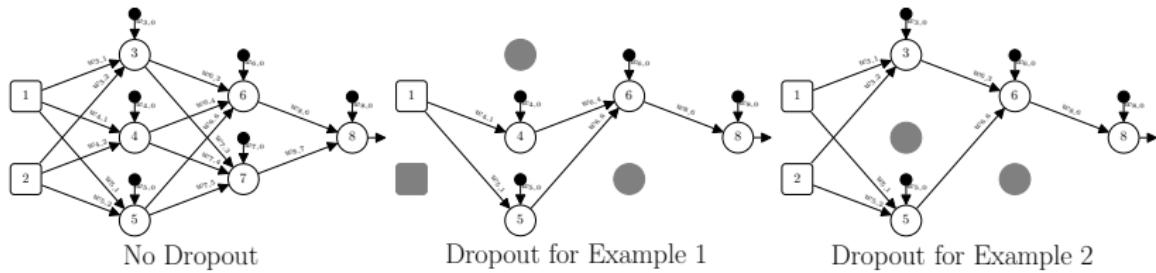


Figure 29: An illustration of how different small networks are generated for different training examples by applying dropout to the original large network. The gray nodes mark the neurons that have been dropped from the network for the training example.

Algorithm 2 Extensions to Backpropagation to Use Inverted Dropout

Require: ρ probability that a neuron in a layer will not be dropped

- 1: **for** each input or hidden layer l **do** ▷ Forward Pass
 - 2: $\text{DropMask}^{(l)} = (m_1, \dots, m_{\text{size}(l)}) \sim \text{Bernoulli}(\rho)$
 - 3: $\mathbf{a}^{(l)'} = \mathbf{a}^{(l)} \odot \text{DropMax}^{(l)}$
 - 4: $\mathbf{a}^{(l)''} = \frac{1}{\rho} \mathbf{a}^{(l)'}$
 - 5: **end for**
 - 6: **for** each layer l in backward pass **do** ▷ Backward Pass
 - 7: $\delta^{(l)} = \delta^{(l)} \odot \text{DropMax}^{(l)}$
 - 8: **end for**
-

Convolutional Neural Networks



Figure 30: Samples of the handwritten digit images from the MNIST dataset. Image attribution: Josef Steppan, used here under the Creative Commons Attribution-Share Alike 4.0 International license (<https://creativecommons.org/licenses/by-sa/4.0>) and was sourced via Wikimedia Commons

Convolutional Neural Networks

$$\begin{bmatrix} 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 \\ 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 \\ 000 & 000 & 000 & \mathbf{255} & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 \end{bmatrix} \quad (84)$$

Convolutional Neural Networks

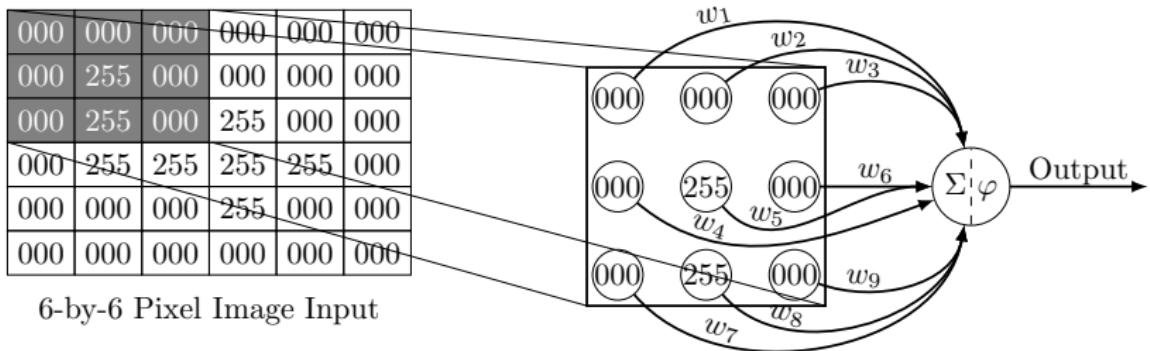


Figure 31: A 6-by-6 matrix representation of a grayscale image of a 4, and a neuron with a receptive field that covers the top-left corner of the image. This figure was inspired by Figure 2 of (Kelleher and Dobnik, 2017).

Convolutional Neural Networks

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (85)$$

$$\begin{aligned} a_i &= \text{rectifier}((w_1 \times 000) + (w_2 \times 000) + (w_3 \times 000) \\ &\quad + (w_4 \times 000) + (w_5 \times 255) + (w_6 \times 000) \\ &\quad + (w_7 \times 000) + (w_8 \times 255) + (w_9 \times 000)) \\ &= \text{rectifier}((0 \times 000) + (0 \times 000) + (0 \times 000) \\ &\quad + (1 \times 000) + (1 \times 255) + (1 \times 000) \\ &\quad + (0 \times 000) + (0 \times 255) + (0 \times 000)) \\ &= 255 \end{aligned} \quad (86)$$

Convolutional Neural Networks

000	000	000	000	000	000
000	255	000	000	000	000
000	255	000	255	000	000
000	255	255	255	255	000
000	000	000	255	000	000
000	000	000	000	000	000

6-by-6 Pixel Image Input

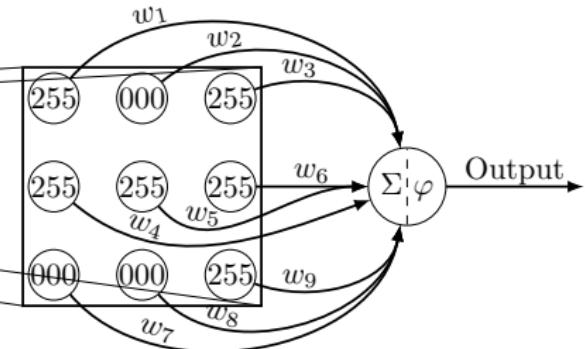
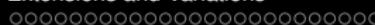
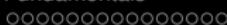


Figure 32: A 6-by-6 matrix representation of a grayscale image of a 4, and a neuron with a different receptive field from the neuron in Figure 31^[107]. This figure was inspired by Figure 2 of (Kelleher and Dobnik, 2017).

Convolutional Neural Networks

$$\begin{aligned} a_i &= \text{rectifier}((w_1 \times 255) + (w_2 \times 000) + (w_3 \times 255) \\ &\quad + (w_4 \times 255) + (w_5 \times 255) + (w_6 \times 255) \\ &\quad + (w_7 \times 000) + (w_8 \times 000) + (w_9 \times 255)) \\ &= \text{rectifier}((0 \times 255) + (0 \times 000) + (0 \times 255) \\ &\quad + (1 \times 255) + (1 \times 255) + (1 \times 255) \\ &\quad + (0 \times 000) + (0 \times 000) + (0 \times 255)) \\ &= 765 \end{aligned} \tag{87}$$



Convolutional Neural Networks

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & +1 & -1 \\ -1 & +1 & -1 \\ -1 & +1 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ +1 & +1 & +1 \\ -1 & -1 & -1 \end{bmatrix} \quad (88)$$



Convolutional Neural Networks

$$\Delta w_{i,*} = \sum_{i=1}^m \boldsymbol{\delta}_i \times a_*$$
$$w_{i,*} \leftarrow w_{i,*} - \alpha \times \Delta w_{i,*} \quad (89)$$

Convolutional Neural Networks

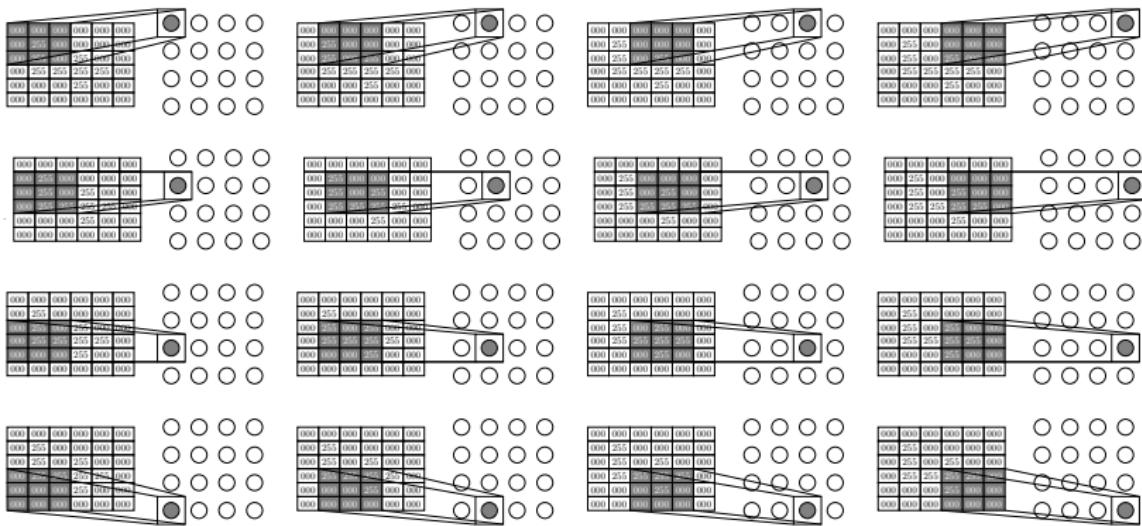


Figure 33: Illustration of the organization of a set of neurons that share weights (use the same filter) and their local receptive fields such that together the receptive fields cover the entirety of the input image.

Convolutional Neural Networks

$$\begin{bmatrix} 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 \\ 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 \\ 000 & 000 & 000 & \mathbf{255} & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 \end{bmatrix} \xrightarrow{\text{Input Image}} \begin{bmatrix} -1 & +1 & -1 \\ -1 & +1 & -1 \\ -1 & +1 & -1 \end{bmatrix} \xrightarrow{\text{Convolved Filter}} \begin{bmatrix} 510 & 0 & 255 & 0 \\ 510 & 0 & 0 & 0 \\ 255 & 0 & 255 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{Feature Map}}$$

(90)

Convolutional Neural Networks

$$\begin{bmatrix} 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 \\ 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 \\ 000 & 000 & 000 & \mathbf{255} & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 \end{bmatrix} \xrightarrow{\text{Convolved Filter}} \begin{bmatrix} -1 & -1 & -1 \\ +1 & +1 & +1 \\ -1 & -1 & -1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 255 & 0 & 255 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Input Image Convolved Filter Feature Map

(91)

Convolutional Neural Networks

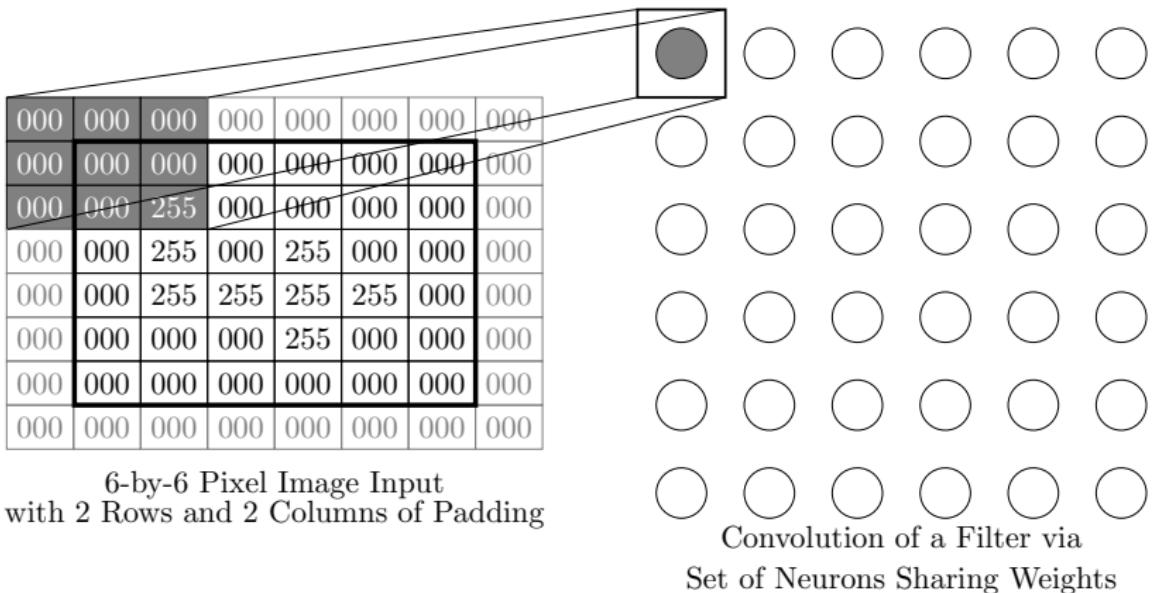


Figure 34: A grayscale image of a 4 after padding has been applied to the original 6-by-6 matrix representation, and the local receptive field of a neuron that includes both valid and padded pixels.

Convolutional Neural Networks

$$\begin{array}{cccccccccc} \left[\begin{matrix} 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \end{matrix} \right] & \xrightarrow{\quad} & \underbrace{\left[\begin{matrix} -1 & +1 & -1 \\ -1 & +1 & -1 \\ -1 & +1 & -1 \end{matrix} \right]}_{\text{Convolved Filter}} & \xrightarrow{\quad} & \underbrace{\left[\begin{matrix} 0 & 255 & 0 & 0 & 0 & 0 \\ 0 & 510 & 0 & 255 & 0 & 0 \\ 0 & 510 & 0 & 0 & 0 & 0 \\ 0 & 255 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 0 & 0 \end{matrix} \right]}_{\text{Feature Map}} \end{array}$$

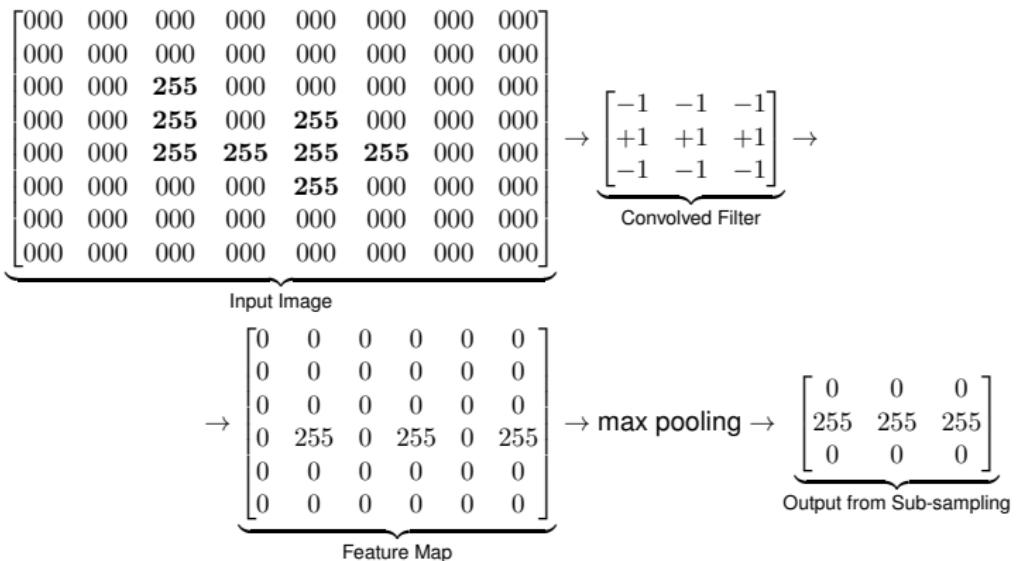
Input Image (92)

Convolutional Neural Networks

$$\begin{array}{ccccccccc} \left[\begin{matrix} 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \end{matrix} \right] & \xrightarrow{\quad} & \underbrace{\left[\begin{matrix} -1 & -1 & -1 \\ +1 & +1 & +1 \\ -1 & -1 & -1 \end{matrix} \right]}_{\text{Convolved Filter}} & \xrightarrow{\quad} & \left[\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 0 & 255 & 0 & 255 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \right] & \xrightarrow{\quad} & \left[\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \right] \\ \text{Input Image} & & & & & & \text{Feature Map} \end{array}$$

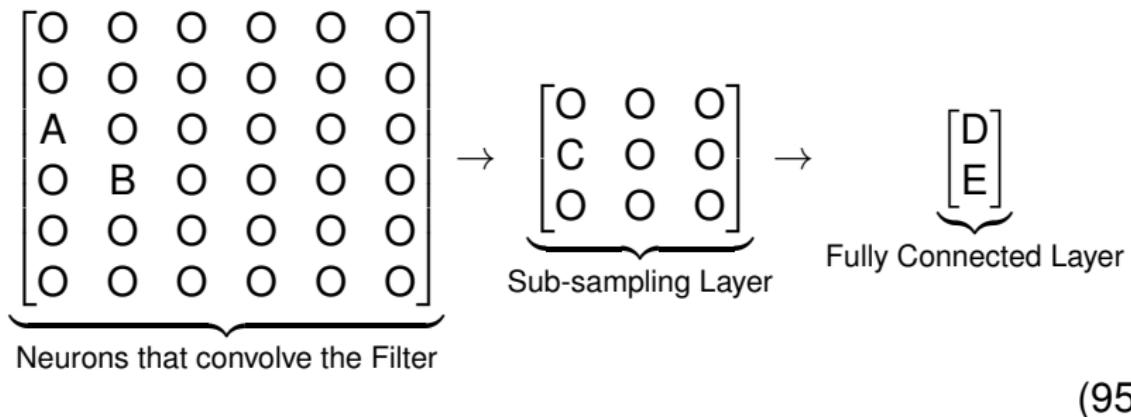
(93)

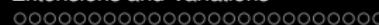
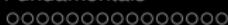
Convolutional Neural Networks



(94)

Convolutional Neural Networks



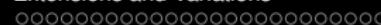


Convolutional Neural Networks

$$\begin{aligned}\delta_C &= \frac{\partial \mathcal{E}}{\partial a_C} \times \frac{\partial a_C}{\partial z_C} \\ &= ((\delta_D \times w_{D,C}) + (\delta_E \times w_{E,C})) \times 1\end{aligned}\tag{96}$$

Convolutional Neural Networks

$$\begin{aligned}\delta_B &= \frac{\partial \mathcal{E}}{\partial a_B} \times \frac{\partial a_B}{\partial z_B} \\ &= (\delta_D \times w_{C,B}) \times \frac{\partial a_B}{\partial z_B} \\ &= (\delta_D \times 1) \quad \times \quad 1\end{aligned}\tag{97}$$



Convolutional Neural Networks

$$\underbrace{w_0}_{\text{bias}} \left[\begin{array}{cc} w_1 & w_2 \\ w_3 & w_4 \end{array} \right] \left[\begin{array}{cc} w_5 & w_6 \\ w_7 & w_8 \end{array} \right] \left[\begin{array}{cc} w_9 & w_{10} \\ w_{11} & w_{12} \end{array} \right] \quad (98)$$

Convolutional Neural Networks

$$\left[\underbrace{w_0 = 0.5}_{\text{bias}} \underbrace{\begin{bmatrix} w_1 = 1 & w_2 = 1 \\ w_3 = 0 & w_4 = 0 \end{bmatrix}}_{\text{Red Channel}} \underbrace{\begin{bmatrix} w_5 = 0 & w_6 = 1 \\ w_7 = 0 & w_8 = 1 \end{bmatrix}}_{\text{Green Channel}} \underbrace{\begin{bmatrix} w_9 = 1 & w_{10} = 0 \\ w_{11} = 0 & w_{12} = 1 \end{bmatrix}}_{\text{Blue Channel}} \right] \quad (99)$$

Convolutional Neural Networks

$$\left[\begin{array}{ccc} \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{Red Channel}} & \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 2 \\ 0 & 0 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix} \\ \end{array} \right] \quad (100)$$

Convolutional Neural Networks

$$\left[\begin{array}{c} \underbrace{\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}}_{\text{Red Channel}} \quad \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}_{\text{Green Channel}} \quad \underbrace{\begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}}_{\text{Blue Channel}} \end{array} \right] \quad (101)$$



Convolutional Neural Networks

$$\begin{aligned} z &= ((w_0 \times 1) \\ &\quad + (w_1 \times 1) + (w_2 \times 1) + (w_3 \times 0) + (w_4 \times 0) \\ &\quad + (w_5 \times 0) + (w_6 \times 0) + (w_7 \times 0) + (w_8 \times 0) \\ &\quad + (w_9 \times 3) + (w_{10} \times 0) + (w_{11} \times 0) + (w_{12} \times 3)) \\ &= 0.5 + 1 + 1 + 0 + 0 + 0 + 0 + 0 + 3 + 0 + 0 + 3 \\ &= 8.5 \end{aligned}$$

$$\begin{aligned} a &= \text{rectifier}(z) \\ &= \text{rectifier}(8.5) \\ &= 8.5 \end{aligned} \tag{102}$$



Convolutional Neural Networks

$$\begin{bmatrix} 8.5 & 6.5 \\ 0.5 & 10.5 \end{bmatrix} \quad (103)$$

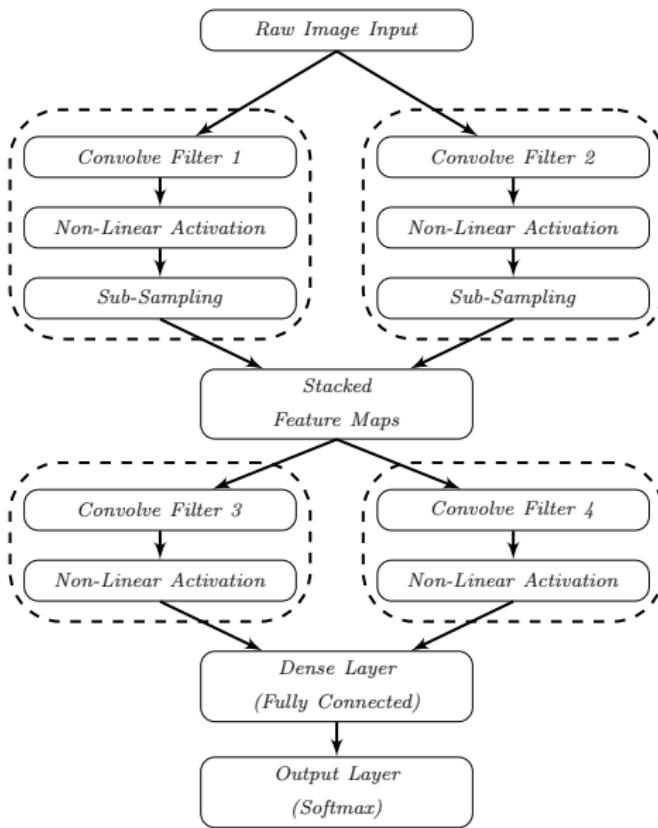


Figure 35: Schematic of the typical sequences of layers found in a convolutional neural network.

Convolutional Neural Networks

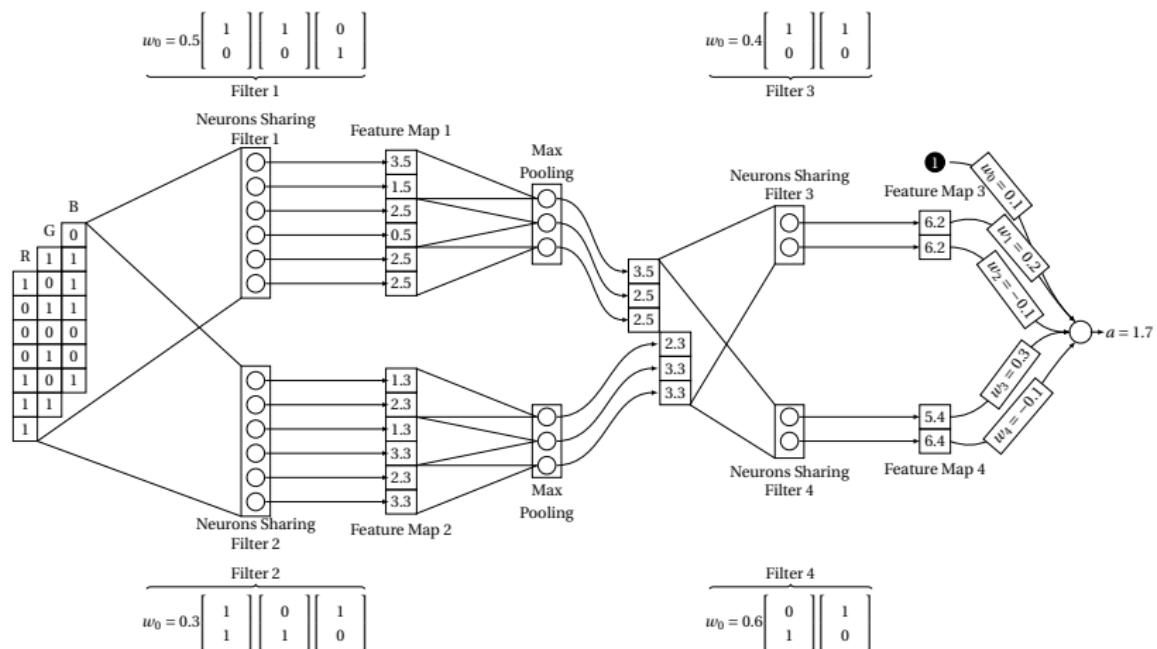


Figure 36: Worked example illustrating the dataflow through a multilayer, multifilter CNN.

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

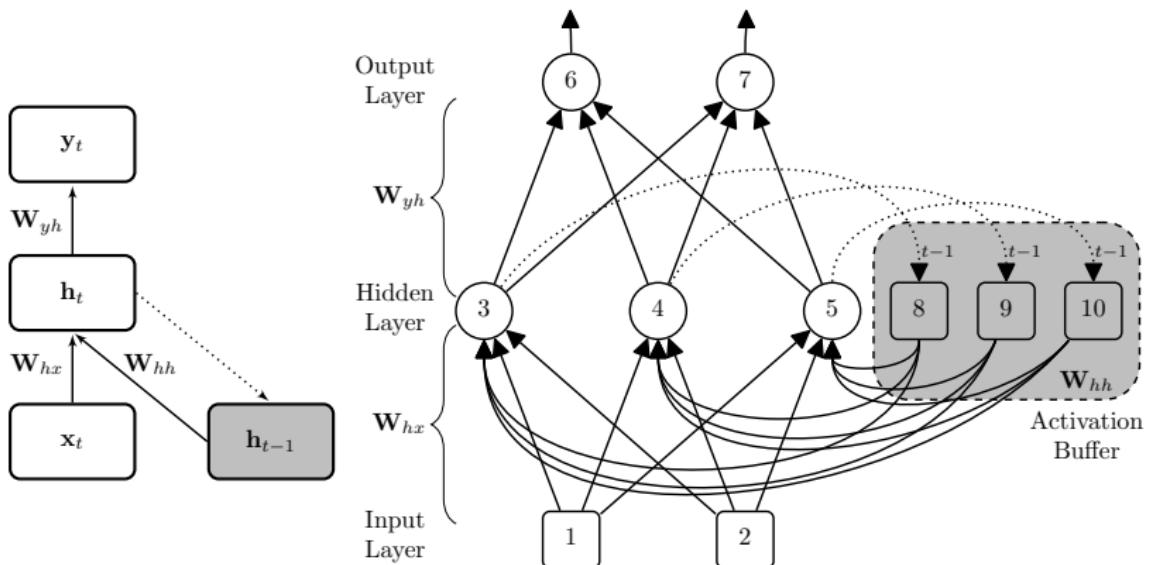


Figure 37: Schematic of the simple recurrent neural architecture.



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\mathbf{h}_t = \varphi((\mathbf{W}_{hh} \cdot \mathbf{h}_{t-1}) + (\mathbf{W}_{hx} \cdot \mathbf{x}_t) + \mathbf{w}_0) \quad (104)$$

$$\mathbf{y}_t = \varphi(\mathbf{W}_{yh} \cdot \mathbf{h}_t) \quad (105)$$

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

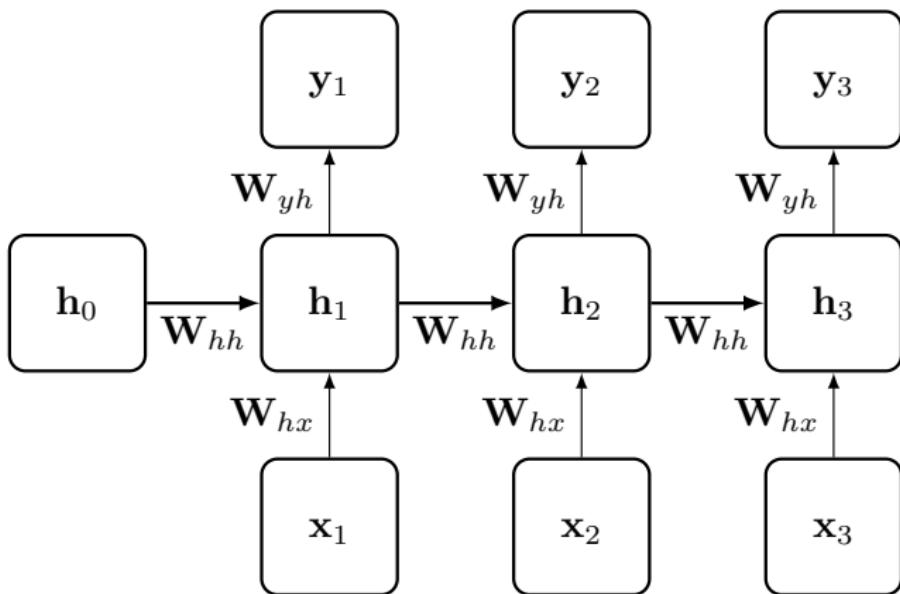


Figure 38: A simple RNN model unrolled through time (in this instance, three time-steps).

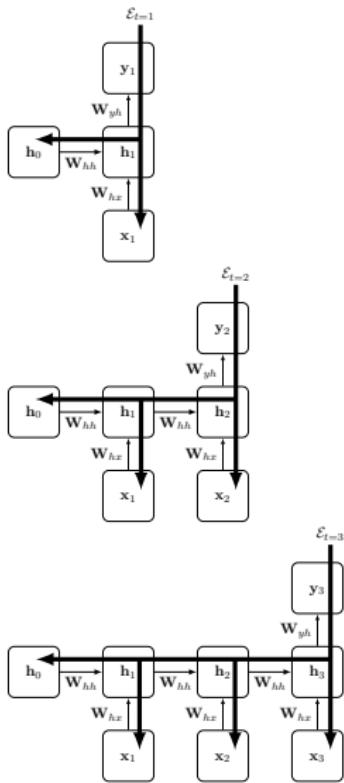


Figure 39: An illustration of the different iterations of backpropagation during backpropagation through time.

Algorithm 3 The Backpropagation Through Time Algorithm

Require: h_0 initialized hidden state

Require: x a sequence of inputs

Require: y a sequence of target outputs

Require: n length of the input sequence

Require: Initialized weight matrices (with associated biases)

Require: Δw a data structure to accumulate the summed weight updates for each weight across time-steps

```
1: for  $t = 1$  to  $n$  do
2:    $Inputs = [x_0, \dots, x_t]$ 
3:    $h_{tmp} = h_0$ 
4:   for  $i = 0$  to  $t$  do                                 $\triangleright$  Unroll the network through  $t$  steps
5:      $h_{tmp} = ForwardPropagate(Inputs[i], h_{tmp})$ 
6:   end for
7:    $\hat{y}_t = OutputLayer(h_{tmp})$                  $\triangleright$  Generate the output for time-step  $t$ 
8:    $\mathcal{E}_t = y[t] - \hat{y}_t$                        $\triangleright$  Calculate the error at time-step  $t$ 
9:    $Backpropagate(\mathcal{E}_t)$                        $\triangleright$  Backpropagate  $\mathcal{E}_t$  through  $t$  steps
10:  For each weight, sum the weight updates across the unrolled network and update  $\Delta w$ 
11: end for
12: Update the network weights using  $\Delta w$ 
```

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

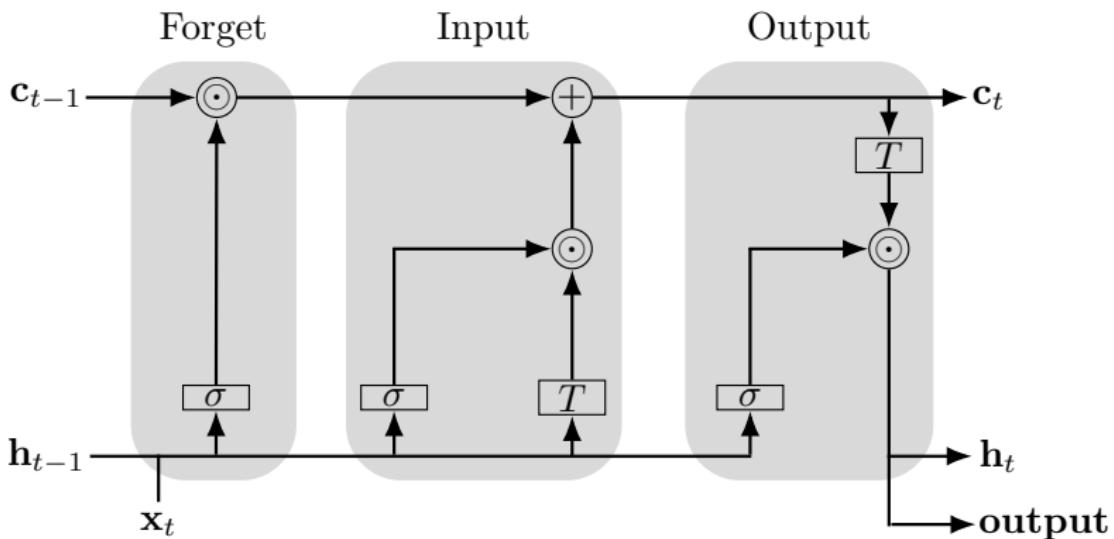


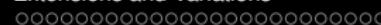
Figure 40: A schematic of the internal structure of a long short-term memory unit. This figure is based on Figure 5.4 of (Kelleher, 2019), which in turn was inspired by an image by Christopher Olah (available at: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>).



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\mathbf{f}_t = \varphi_{sigmoid}(\mathbf{W}^{(f)} \cdot \mathbf{h} \mathbf{x}_t) \quad (106)$$

$$\mathbf{c}_t^+ = \mathbf{c}_{t-1} \odot \mathbf{f}_t \quad (107)$$



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\mathbf{c}_{t-1} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{h}_{t-1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \mathbf{x}_t = [4]$$
$$\mathbf{W}^{(f)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(108)

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\underbrace{\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{W}^{(f)}} \times \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 4 \end{bmatrix}}_{\mathbf{hx}_t} = \underbrace{\begin{bmatrix} 6 \\ -6 \\ 0 \end{bmatrix}}_{\mathbf{z}_t^{(f)}} \rightarrow \varphi_{sigmoid} \rightarrow \underbrace{\begin{bmatrix} 0.997527377 \\ 0.002472623 \\ 0.500000000 \end{bmatrix}}_{\mathbf{f}_t}$$

$$\underbrace{\begin{bmatrix} 0.997527377 \\ 0.002472623 \\ 0.500000000 \end{bmatrix}}_{\mathbf{f}_t} \odot \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\mathbf{c}_{t-1}} = \underbrace{\begin{bmatrix} 0.997527377 \\ 0.002472623 \\ 0.500000000 \end{bmatrix}}_{\mathbf{c}_t^{\dagger}}$$

(109)

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\mathbf{i}_{\dagger t} = \varphi_{sigmoid}(\mathbf{W}^{(i\dagger)} \cdot \mathbf{h}\mathbf{x}_t) \quad (110)$$

$$\mathbf{i}_{\ddagger t} = \varphi_{tanh}(\mathbf{W}^{(i\dagger)} \cdot \mathbf{h}\mathbf{x}_t) \quad (111)$$

$$\mathbf{i}_t = \mathbf{i}_{\dagger t} \odot \mathbf{i}_{\ddagger t} \quad (112)$$

$$\mathbf{c}_t = \mathbf{c}_{\dagger t} + \mathbf{i}_t \quad (113)$$

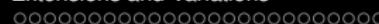
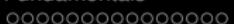
Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\mathbf{o}_t^\dagger = \varphi_{sigmoid}(\mathbf{W}^{(o\dagger)} \cdot \mathbf{h} \mathbf{x}_t) \quad (114)$$

$$\mathbf{o}_t^\ddagger = \varphi_{tanh}(\mathbf{W}^{(o\dagger)} \cdot \mathbf{c}_t) \quad (115)$$

$$\mathbf{o}_t = \mathbf{o}_t^\dagger \odot \mathbf{o}_t^\ddagger \quad (116)$$

$$\mathbf{h}_{t+1} = \mathbf{o}_t \quad (117)$$



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\mathbf{f}_t = \varphi_{sigmoid}(\mathbf{W}^{(f)} \cdot \mathbf{h}\mathbf{x}_t)$$

$$\mathbf{c}_t^\ddagger = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$

$$\mathbf{i}_t^\dagger = \varphi_{sigmoid}(\mathbf{W}^{(i\dagger)} \cdot \mathbf{h}\mathbf{x}_t)$$

$$\mathbf{i}_t^\ddagger = \varphi_{tanh}(\mathbf{W}^{(i\ddagger)} \cdot \mathbf{h}\mathbf{x}_t)$$

$$\mathbf{i}_t = \mathbf{i}_t^\dagger \odot \mathbf{i}_t^\ddagger$$

$$\mathbf{c}_t = \mathbf{c}_t^\ddagger + \mathbf{i}_t$$

$$\mathbf{o}_t^\dagger = \varphi_{sigmoid}(\mathbf{W}^{(o\dagger)} \cdot \mathbf{h}\mathbf{x}_t)$$

$$\mathbf{o}_t^\ddagger = \varphi_{tanh}(\mathbf{W}^{(o\ddagger)} \cdot \mathbf{c}_t)$$

$$\mathbf{o}_t = \mathbf{o}_t^\dagger \odot \mathbf{o}_t^\ddagger$$

$$\mathbf{h}_{t+1} = \mathbf{o}_t$$

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

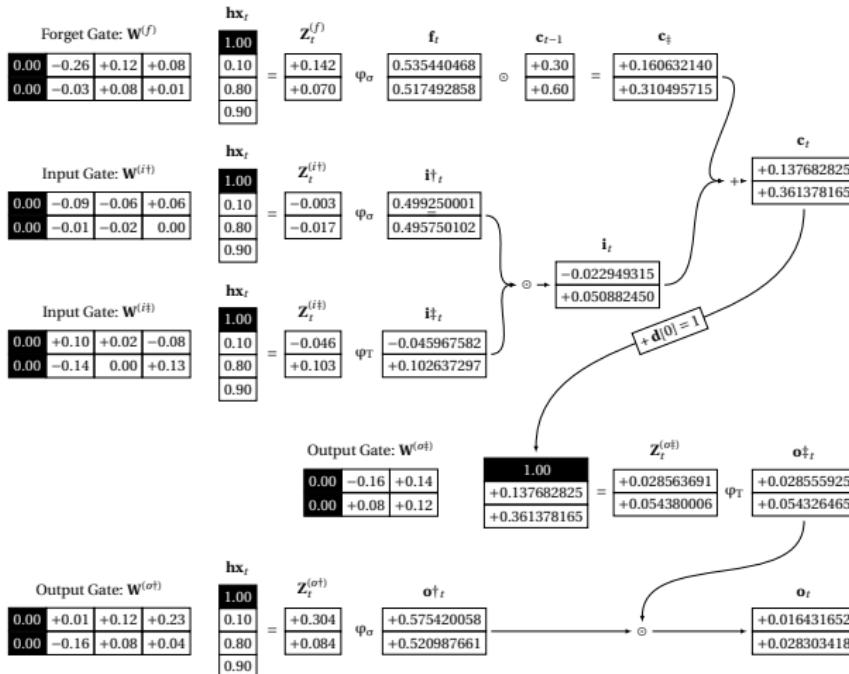


Figure 41: The flow of activations through a long short-term memory unit during forward propagation when $\mathbf{c}_{t-1} = [0.3, 0.6]$, $\mathbf{h}_t = [0.1, 0.8]$, and $\mathbf{x}_t = [0.9]$.

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

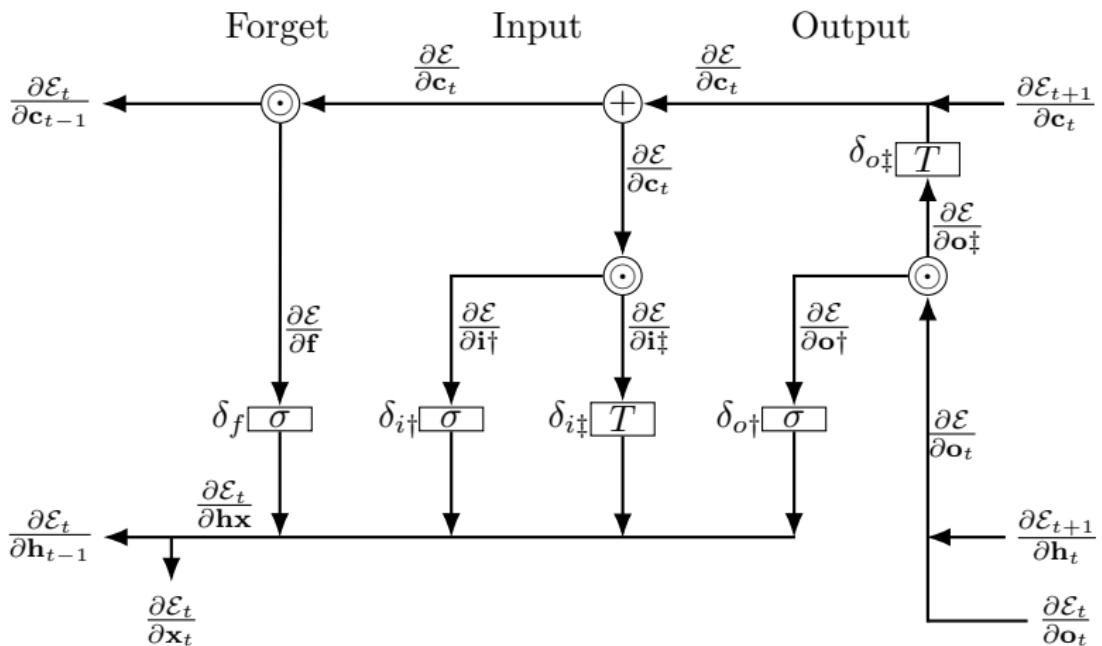


Figure 42: The flow of error gradients through a long short-term memory unit during backpropagation.



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\frac{\partial \mathcal{E}}{\partial \mathbf{o}_t} = \frac{\partial \mathcal{E}_t}{\partial \mathbf{o}_t} + \frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{h}_t} \quad (118)$$



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

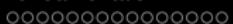
$$\frac{\partial \mathcal{E}}{\partial \mathbf{o}_t^\ddagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}_t} \odot \mathbf{o}_t^\dagger \quad (119)$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{o}_t^\dagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}_t} \odot \mathbf{o}_t^\ddagger \quad (120)$$

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\delta_{o_t^{\dagger}} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}_t^{\dagger}} \odot \frac{\partial \mathbf{o}_{t+1}^{\dagger}}{\partial \mathbf{c}_{t+1}^{\dagger}} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}_t^{\dagger}} \odot \underbrace{(1 - \tanh^2(\mathbf{c}_{t+1}^{\dagger}))}_{\text{Derivate of tanh, i.e.: } \frac{\partial a}{\partial z}} \quad (121)$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} = \delta_{o_t^{\dagger}} + \frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{c}_t} \quad (122)$$



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\frac{\partial \mathcal{E}}{\partial \mathbf{i}_t^\ddagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} \odot \mathbf{i}_t^\dagger \quad (123)$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{i}_t^\dagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} \odot \mathbf{i}_t^\ddagger \quad (124)$$



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\frac{\partial \mathcal{E}}{\partial \mathbf{c}_{t-1}} = \frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} \odot \mathbf{f}_t \quad (125)$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{f}} = \frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} \odot \mathbf{c}_{t-1} \quad (126)$$

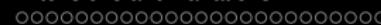
Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\delta_f = \frac{\partial \mathcal{E}}{\partial \mathbf{f}} \odot \frac{\partial \mathbf{f}_t}{\partial \mathbf{z}_f} = \frac{\partial \mathcal{E}}{\partial \mathbf{f}} \odot (\mathbf{f}_t \odot (\mathbf{1} - \mathbf{f}_t)) \quad (127)$$

$$\delta_{i\dagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{i}\dagger} \odot \frac{\partial \mathbf{i}\dagger_t}{\partial \mathbf{z}_{i\dagger}} = \frac{\partial \mathcal{E}}{\partial \mathbf{i}\dagger} \odot (\mathbf{i}\dagger_t \odot (\mathbf{1} - \mathbf{i}\dagger_t)) \quad (128)$$

$$\delta_{i\dagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{i}\dagger} \odot \frac{\partial \mathbf{i}\dagger_t}{\partial \mathbf{z}_{i\dagger}} = \frac{\partial \mathcal{E}}{\partial \mathbf{i}\dagger} \odot (\mathbf{1} - \tanh^2(\mathbf{i}\dagger_t)) \quad (129)$$

$$\delta_{o\dagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}\dagger} \odot \frac{\partial \mathbf{o}\dagger_t}{\partial \mathbf{z}_{o\dagger}} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}\dagger} \odot (\mathbf{o}\dagger_t \odot (\mathbf{1} - \mathbf{o}\dagger_t)) \quad (130)$$



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\Delta \mathbf{W}^{(f)} = \delta_f \cdot \mathbf{h} \mathbf{x}^\top \quad (131)$$



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{c}_t} = \begin{bmatrix} 0.35 \\ 0.50 \end{bmatrix} \quad \frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{h}_t} = \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix} \quad \frac{\partial \mathcal{E}_t}{\partial \mathbf{o}_t} = \begin{bmatrix} 0.15 \\ 0.60 \end{bmatrix}$$

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\frac{\partial \mathcal{E}}{\partial \mathbf{o}_t} = \underbrace{\begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix}}_{\partial \mathcal{E}_{t+1}/\partial \mathbf{h}_t} + \underbrace{\begin{bmatrix} 0.15 \\ 0.60 \end{bmatrix}}_{\partial \mathcal{E}_t/\partial \mathbf{o}_t} = \begin{bmatrix} 0.9 \\ 0.85 \end{bmatrix} \quad (132)$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{o}^\ddagger} = \underbrace{\begin{bmatrix} 0.9 \\ 0.85 \end{bmatrix}}_{\partial \mathcal{E}/\partial \mathbf{o}_t} \odot \underbrace{\begin{bmatrix} 0.575420058 \\ 0.52098661 \end{bmatrix}}_{\mathbf{o}^\dagger} = \begin{bmatrix} 0.517878052 \\ 0.442839512 \end{bmatrix} \quad (133)$$

$$\delta_{o^\ddagger} = \underbrace{\begin{bmatrix} 0.517878052 \\ 0.442839512 \end{bmatrix}}_{\partial \mathcal{E}/\partial \mathbf{o}^\ddagger} \odot \underbrace{\begin{bmatrix} 0.999184559 \\ 0.997048635 \end{bmatrix}}_{1 - \tanh(\mathbf{c}_t)} = \begin{bmatrix} 0.517455753 \\ 0.441532531 \end{bmatrix} \quad (134)$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} = \underbrace{\begin{bmatrix} 0.517455753 \\ 0.441532531 \end{bmatrix}}_{\delta_{o^\ddagger}} + \underbrace{\begin{bmatrix} 0.35 \\ 0.50 \end{bmatrix}}_{\partial \mathcal{E}_{t+1}/\partial \mathbf{c}_t} = \begin{bmatrix} 0.867455753 \\ 0.941532531 \end{bmatrix} \quad (135)$$

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\frac{\partial \mathcal{E}}{\partial \mathbf{f}} = \underbrace{\begin{bmatrix} 0.867455753 \\ 0.941532531 \end{bmatrix}}_{\partial \mathcal{E} / \partial \mathbf{c}_t} \odot \underbrace{\begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}}_{\mathbf{c}_{t-1}} = \begin{bmatrix} 0.260236726 \\ 0.564919518 \end{bmatrix} \quad (136)$$

$$\delta_f = \underbrace{\begin{bmatrix} 0.260236726 \\ 0.564919518 \end{bmatrix}}_{\partial \mathcal{E} / \partial \mathbf{f}} \odot \underbrace{\begin{bmatrix} 0.248743973 \\ 0.249694 \end{bmatrix}}_{\mathbf{f}_t \odot (1 - \mathbf{f}_t)} = \begin{bmatrix} 0.064732317 \\ 0.141057014 \end{bmatrix} \quad (137)$$

$$\begin{aligned} \Delta \mathbf{W}^{(f)} &= \underbrace{\begin{bmatrix} 0.064732317 \\ 0.141057014 \end{bmatrix}}_{\delta_f} \cdot \underbrace{\begin{bmatrix} 1.00 & 0.10 & 0.80 & 0.90 \end{bmatrix}}_{\mathbf{h} \mathbf{x}^\top} \\ &= \begin{bmatrix} 0.064732317 & 0.006473232 & 0.051785854 & 0.058259085 \\ 0.141057014 & 0.014105701 & 0.112845611 & 0.126951313 \end{bmatrix} \quad (138) \end{aligned}$$



Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \mathbf{h}x} = & \left(\mathbf{W}^{(f)\top} \cdot \delta_f \right) + \left(\mathbf{W}^{(i\dagger)\top} \cdot \delta_{i\dagger} \right) \\ & + \left(\mathbf{W}^{(i\dagger\ddagger)\top} \cdot \delta_{i\dagger\ddagger} \right) + \left(\mathbf{W}^{(o\dagger)\top} \cdot \delta_{o\dagger} \right)\end{aligned}\quad (139)$$

Sequential Models: Recurrent Neural Networks and Long Short-Term Memory Networks

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \mathbf{c}_{t-1}} &= \mathbf{f}_t \odot \frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} \\&= \mathbf{f}_t \odot \left(\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{c}_t} + \delta_{o^\ddagger_t} \right) \\&= \mathbf{f}_t \odot \left(\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{c}_t} + \left((\mathbf{1} - \tanh^2(\mathbf{c}_{\dagger t}^\ddagger)) \odot \frac{\partial \mathcal{E}}{\partial \mathbf{o}^\ddagger_t} \right) \right) \\&= \mathbf{f}_t \odot \left(\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{c}_t} + \left((\mathbf{1} - \tanh^2(\mathbf{c}_{\dagger t}^\ddagger)) \odot \left(\mathbf{o}^\ddagger \odot \frac{\partial \mathcal{E}}{\partial \mathbf{o}_t} \right) \right) \right) \\&= \mathbf{f}_t \odot \left(\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{c}_t} + \left((\mathbf{1} - \tanh^2(\mathbf{c}_{\dagger t}^\ddagger)) \odot \left(\mathbf{o}^\ddagger \odot \left(\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{E}_t}{\partial \mathbf{o}_t} \right) \right) \right) \right)\end{aligned}\tag{140}$$

Big Idea

Fundamentals

Standard Approach: Backpropagation and Gradient Descent

Extensions and Variations



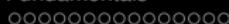
Summary



- Deep neural networks to learn and represent complex mappings from inputs to outputs
- The standard algorithm for training a deep neural network combines the backpropagation algorithm
- Unstable gradients (either vanishing or exploding gradients) can make training a deep network with backpropagation and gradient descent difficult
 - Activation Functions (ReLUs)
 - Weight Initialization
- Dropout is a very simple and effective method that helps to stop overfitting.
- We can tailor the structure of a network toward the characteristics of the data
 - Convolutional Neural Networks
 - Recurrent Neural Networks



Further Reading



- Other texts on neural networks and deep learning:
(Bishop, 1996; Reed and Marks, 1999; Kelleher, 2019;
Goodfellow et al., 2016)
- Programming focused introductions to deep learning:
(Charniak, 2019; Trask, 2019)
- Introduction to neural networks for natural language
processing: (Goldberg, 2017)
- Computer architecture perspective on deep learning:
(Reagen et al., 2017)
- Recent developments in the field: **batch normalization**
(Ioffe and Szegedy, 2015), adaptively learning algorithms
such as **Adam** (Kingma and Ba, 2014), **Generative
Adversarial Networks** (Goodfellow et al., 2014), and
attention-based architectures such as the **Transformer**
(Vaswani et al., 2017).

- 1 Big Idea
- 2 Fundamentals
- 3 Standard Approach: Backpropagation and Gradient Descent
- 4 Extensions and Variations
- 5 Summary
- 6 Further Reading

- Bishop, Christopher M. 1996. *Neural networks for pattern recognition*. Oxford University Press.
- Charniak, Eugene. 2019. *Introduction to deep learning*. MIT Press.
- Goldberg, Yoav. 2017. *Neural network methods for natural language processing. Synthesis lectures on human language technology*. Morgan and Claypool.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT Press.
- Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680.
- Ioffe, Sergey, and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by

reducing internal covariate shift. In *International conference on machine learning*, 448–456.

Kelleher, John D. 2019. *Deep learning. Essential knowledge series*. MIT Press.

Kelleher, John D, and Simon Dobnik. 2017. What is not where: the challenge of integrating spatial representations into deep learning architectures. *CLASP Papers in Computational Linguistics*.

Kingma, Diederik P, and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Marsland, Stephen. 2011. *Machine learning: an algorithmic perspective*. CRC Press.

Reagen, Brandon, Robert Adolf, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017. *Deep learning for computer architects*. Morgan and Claypool.

Reed, Russell D., and Robert J. Marks. 1999. *Neural smithing: supervised learning in feedforward artificial networks*. MIT Press.

Trask, Andrew. 2019. *Grokking deep learning*. Manning.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.