Deep Learning

John D. Kelleher and Brian Mac Namee and Aoife D'Arcy

Big Idea

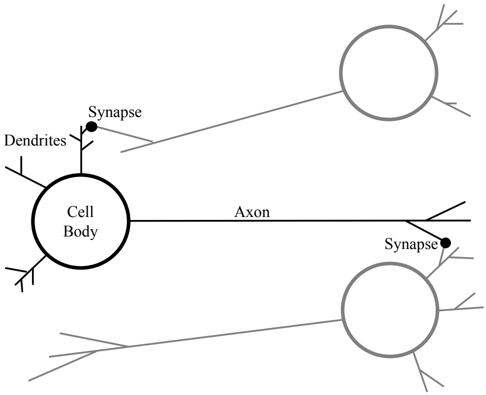


Figure 1: A high-level schematic of the structure of a neuron. This figure illustrates three interconnected neurons; the middle neuron is highlighted in black, and the major structural components of this neuron are labeled cell body, dendrites, and axon. Also marked are the synapses connecting the axon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

Fundamentals

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

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

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

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

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

Artificial Neurons

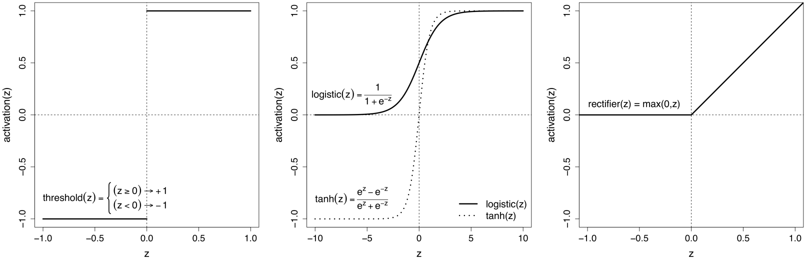


Figure 2: Plots for activation functions that have been popular in the history of neural networks.

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

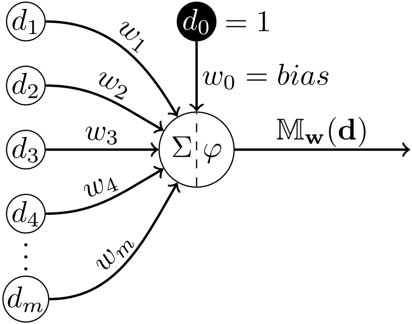


Figure 3: A schematic of an artificial neuron.

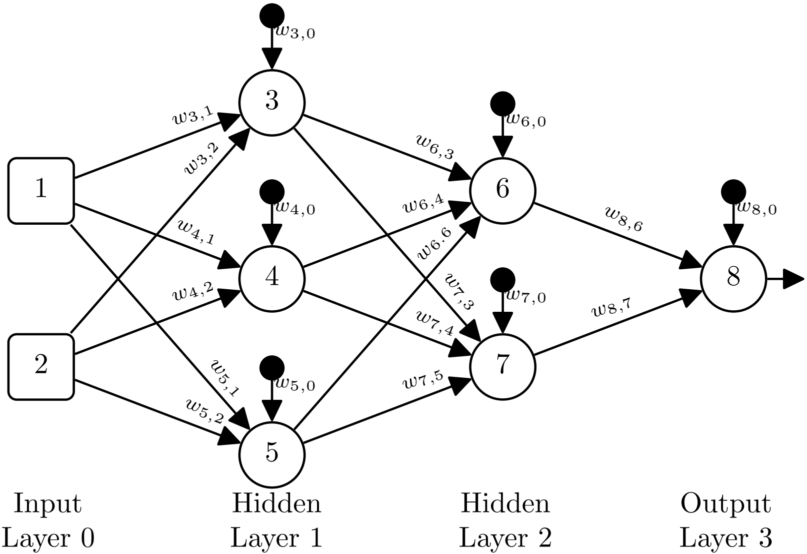


Figure 4: A schematic of a feedforward artificial neural network.

Neural Networks as Matrix Operations

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

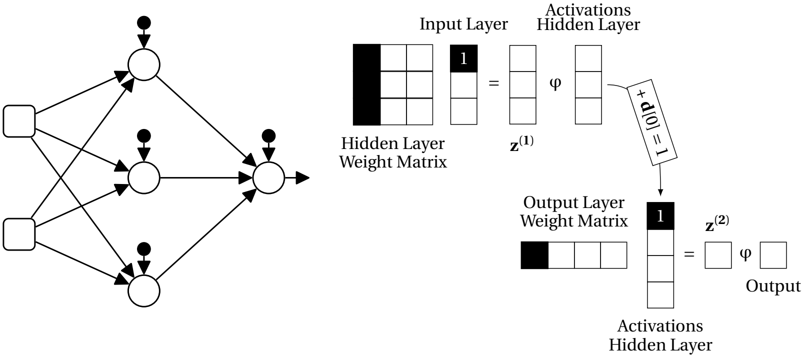


Figure 5: An illustration of the correspondence between graphical and matrix representations of a neural network. This figure is inspired by Figure 3.9 of (Kelleher, 2019).

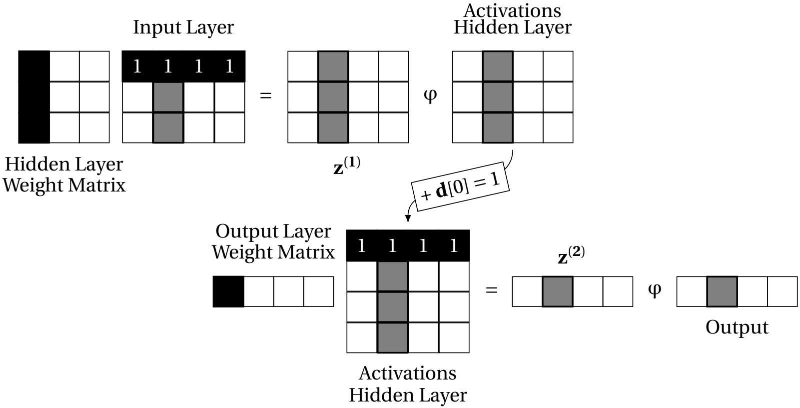


Figure 6: An illustration of how a batch of examples can be processed in parallel using matrix operations.

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

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

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

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

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

Why Is Network Depth Important?

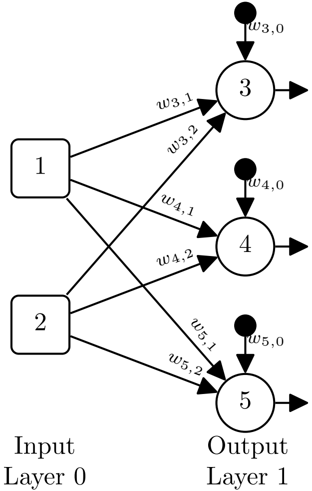


Figure 7: A single-layer network.

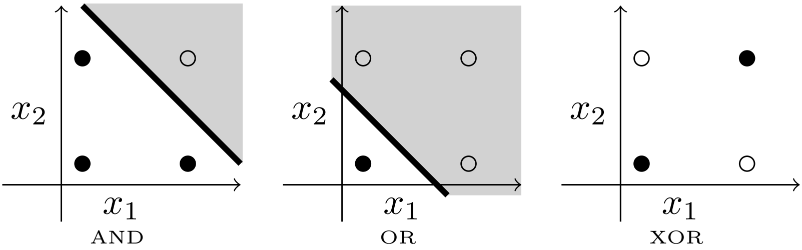


Figure 8: The logical AND and OR functions are linearly separable, but the XOR is not. This figure is Figure 4.2 of (Kelleher, 2019) and is used here with permission.

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

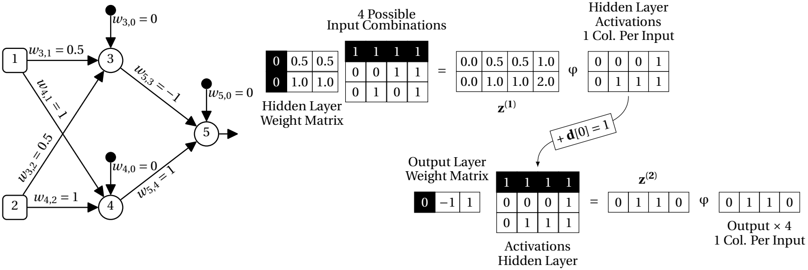


Figure 9: (left) The XOR function implemented as a two-layer neural network. (right) The network processing the four possible input combinations, one combination plus bias input per column: [*bias*, *FALSE*,*FALSE*] $\rightarrow$ [1,0,0]; [*bias*, *FALSE*,*TRUE*] $\rightarrow$ [1,0,1]; [*bias*, *TRUE*,*FALSE*]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

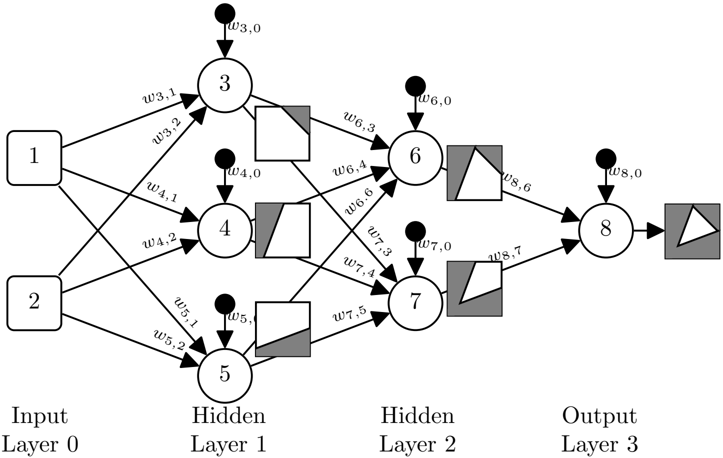


Figure 10: An illustration of how the representational capacity of a network increases as more layers are added to the network. This figure was inspired by Figure 4.2 in (Reed and Marks, 1999) and Figure 3.10 in (Marsland, 2011).

Standard Approach: Backpropagation and Gradient Descent

Backpropagation: The General Structure of the Algorithm

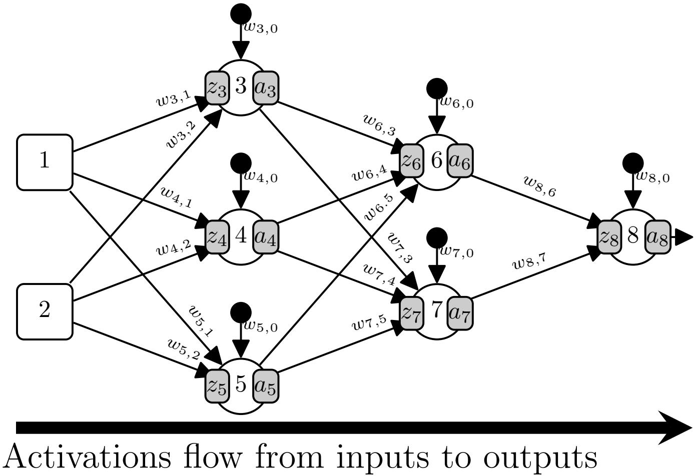


Figure 11: The calculation of the z values and activations of each neuron during the forward pass of the backpropagation algorithm. This figure is based on Figure 6.5 of (Kelleher, 2019).

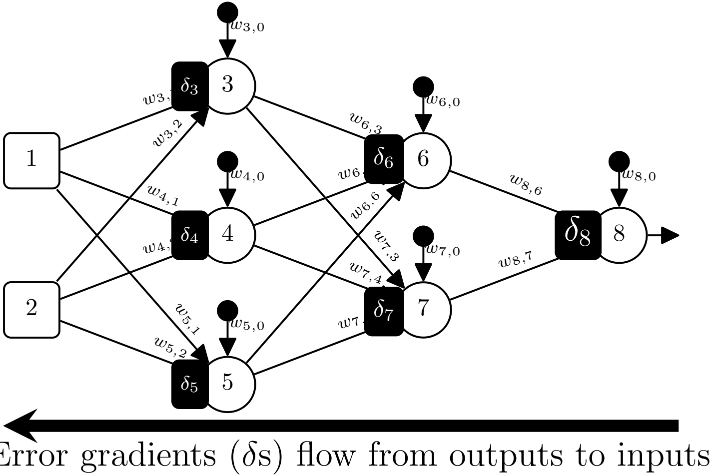


Figure 12: The backpropagation of the δ values during the backward pass of the backpropagation algorithm. This figure is based on Figure 6.6 of (Kelleher, 2019).

$$\boldsymbol{\delta}_k = \frac{\partial \mathcal{E}}{\partial z_k} \tag{13}$$

$$\boldsymbol{\delta}_k = \frac{\partial a_k}{\partial z_k} \times \frac{\partial \mathcal{E}}{\partial a_k} \tag{14}$$

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

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

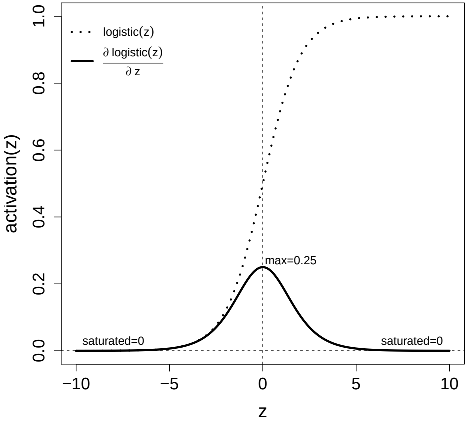


Figure 13: Plots of the logistic function and its derivative. This figure is Figure 4.6 of (Kelleher, 2019) and is used here with permission.

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

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

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

$$\frac{\partial \mathcal{E}}{\partial a_k} = -\left(t_k - a_k\right) \tag{20}$$

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

Backpropagation: Backpropagating the Error Gradients

$$k \rightarrow i$$

$$\frac{\partial \mathcal{E}}{\partial a_k} = \sum_{i=1}^{n} w_{i,k} \times \boldsymbol{\delta}_i \tag{22}$$

$$\begin{aligned}
\boldsymbol{\delta}_k &= \frac{\partial a_k}{\partial z_k} \times \frac{\partial \mathcal{E}}{\partial a_k} \\
&= \frac{\partial a_k}{\partial z_k} \times \left(\sum_{i=1}^{n} w_{i,k} \times \boldsymbol{\delta}_i \right) \\
&= \underbrace{\frac{d}{dz} logistic(z)}_{\text{Assuming a logistic activation function}} \times \left(\sum_{i=1}^{n} w_{i,k} \times \boldsymbol{\delta}_i \right) \\
&= \underbrace{(logistic(z) \times (1 - logistic(z)))}_{\text{Assuming a logistic activation function}} \times \left(\sum_{i=1}^{n} w_{i,k} \times \boldsymbol{\delta}_i \right)
\end{aligned} \tag{23}$$

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

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

$$\frac{\partial z_i}{\partial w_{i,k}} = a_k \tag{26}$$

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

Backpropagation: Updating the Weights in a Network

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

$$\Delta w_{i,k} = \sum_{j=1}^{m} \boldsymbol{\delta}_{i,j} \times a_{k,j} \tag{29}$$

$$w_{i,k} \leftarrow w_{i,k} - \alpha \times \Delta w_{i,k} \tag{30}$$

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

Shuffle $\mathcal{D}$ and create the mini-batches: $[(\mathbf{X}^{(1)}, \mathbf{Y}^{(1)}), \ldots, (\mathbf{X}^{k}, \mathbf{Y}^{k})]$

Initialize the weight matrices for each layer: $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}$

repeat ▷ Each repeat loop is one epoch

  for t=1 to number of mini-batches **do** ▷ Each for loop is one iteration

    $\mathbf{A}^{(0)} \leftarrow \mathbf{X}^{(t)}$

    for l=1 to L **do**

      $\mathbf{v} \leftarrow [1_0, \ldots 1_m]$ ▷ Create $\mathbf{v}$ the vector of bias terms

      $\mathbf{A}^{(l-1)} \leftarrow [\mathbf{v}; \mathbf{A}^{(l-1)}]$ ▷ Insert $\mathbf{v}$ into the activation matrix

      $\mathbf{Z}^{(l)} \leftarrow \mathbf{W}^{l} \mathbf{A}^{(l-1)}$

      $\mathbf{A}^{(l)} \leftarrow \varphi(\mathbf{Z}^{(l)})$ ▷ Elementwise application of φ to $\mathbf{Z}^{(l)}$

    end for

    for each weight $w_{i,k}$ in the network **do**

      $\Delta w_{i,k} = 0$

    end for

    for each example in the mini-batch **do** ▷ Backpropagate the δs

      for each neuron i in the output layer **do**

        $\delta_i = \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i}$ ▷ See Equation (21)[28]

      end for

      for l = L-1 to 1 **do**

        for each neuron i in the layer l **do**

          $\delta_i = \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial z_i}$ ▷ See Equation (23)[29]

        end for

      end for

      for each weight $w_{i,k}$ in the network **do**

        $\Delta w_{i,k} = \Delta w_{i,k} + (\delta_i \times a_k)$ ▷ Equation (29)[33]

      end for

    end for

    for each weight $w_{i,k}$ in the network **do**

      $w_{i,k} \leftarrow w_{i,k} - \alpha \times \Delta w_{i,k}$ ▷ Equation (30)[33]

    end for

  end for

  shuffle($[(\mathbf{X}^{(1)}, \mathbf{Y}^{(1)}), \ldots, (\mathbf{X}^{k}, \mathbf{Y}^{k})]$)

until convergence occurs

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

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

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

Hidden Layer 1 Weight Matrix

		1	2
3	+0.10	−0.20	+0.11
4	−0.19	+0.09	−0.13
5	+0.04	+0.11	−0.02

Input Layer

$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$
1.00	1.00	1.00	1.00
0.04	0.84	0.50	0.53
0.81	0.58	0.07	1.00

=

$\mathbf{Z}^{(1)}$

+0.1811	−0.0042	+0.0077	+0.1040
−0.2917	−0.1898	−0.1541	−0.2723
+0.0282	+0.1208	+0.0936	+0.0783

φ

Activations Hidden Layer 1

0.5452	0.4990	0.5019	0.5260
0.4276	0.4527	0.4616	0.4323
0.5070	0.5302	0.5234	0.5196

+ **d**[0] = 1

Hidden Layer 2 Weight Matrix

		3	4	5
6	+0.15	−0.04	+0.10	+0.06
7	+0.12	+0.20	+0.14	−0.09

Activations Hidden Layer 1

1.0000	1.0000	1.0000	1.0000
0.5452	0.4990	0.5019	0.5260
0.4276	0.4527	0.4616	0.4323
0.5070	0.5302	0.5234	0.5196

=

$\mathbf{Z}^{(2)}$

+0.201372	+0.207122	+0.207488	+0.203366
+0.243274	+0.235460	+0.237898	+0.238958

φ

Activations Hidden Layer 2

0.5502	0.5516	0.5517	0.5507
0.5605	0.5586	0.5592	0.5595

+ **d**[0] = 1

Output Layer Weight Matrix

		6	7
8	+0.10	+0.12	−0.50

Activations Hidden Layer 2

1.0000	1.0000	1.0000	1.0000
0.5502	0.5516	0.5517	0.5507
0.5605	0.5586	0.5592	0.5595

=

$\mathbf{Z}^{(3)}$

−0.114226	−0.113108	−0.113396	−0.113666

φ

Activations Output Layer

0.4715	0.4718	0.4717	0.4716

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

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

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$
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
$\partial\mathcal{E}/\partial a_8$: 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

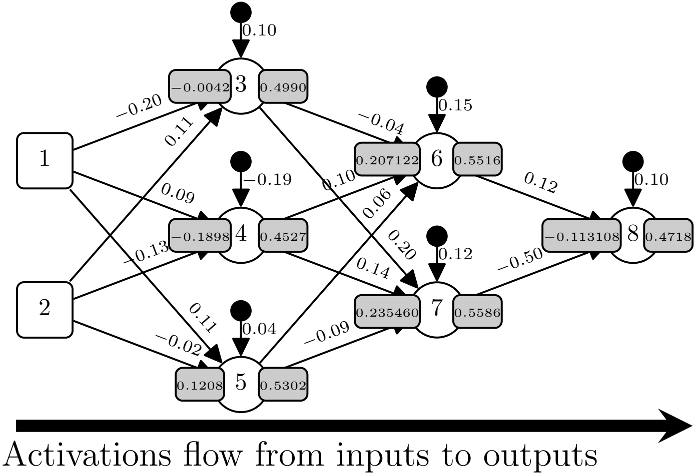


Figure 15: An illustration of the forward propagation of $\mathbf{d}_2$ through the network showing the weights on each connection, and the weighted sum z and activation a value for each neuron in the network.

$$\frac{\partial \mathcal{E}}{\partial a} = 0.3418 \tag{31}$$

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

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

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

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

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

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

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

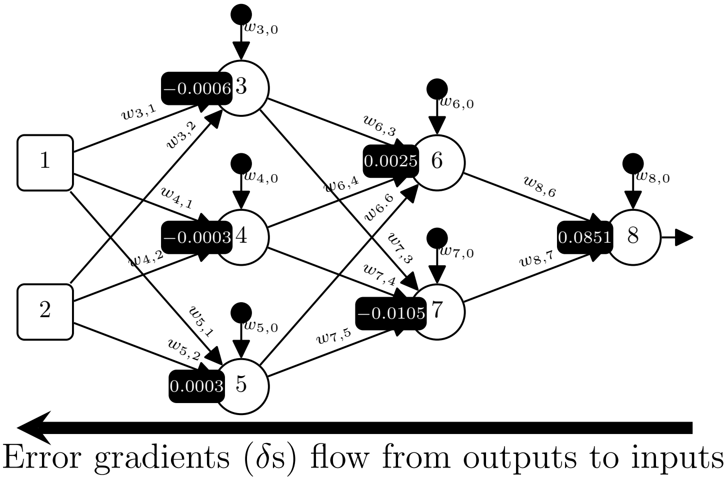


Figure 16: The δs for each of the neurons in the network for Ex. 2.

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	$-0.0006 \times 0.58 = -0.000348$

$$\begin{aligned} w_{7,5} &= w_{7,5} - \alpha \times \boldsymbol{\delta}_7 \times a_5 \\ &= w_{7,5} - \alpha \times \frac{\partial \mathcal{E}}{\partial w_{i,k}} \\ &= -0.09 - 0.2 \times -0.0055671 \\ &= -0.08888658 \end{aligned} \tag{38}$$

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

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

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

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$
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
$\partial\mathcal{E}/\partial a_8$: Error $\times -1$	-0.4662	0.3441	-0.0960	0.1139
Error^2	0.21734244	0.11840481	0.009216	0.01297321
SSE:				0.17896823

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

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_2$	$\mathbf{d}_2$
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

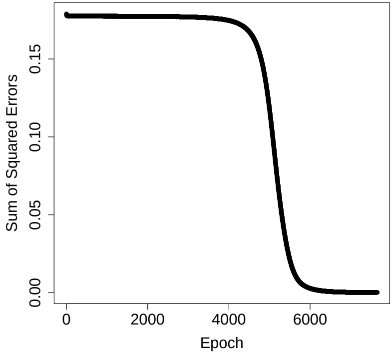


Figure 17: A plot showing how the sum of squared errors of the network changed during training.

Extensions and Variations

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

$$i \rightarrow j \rightarrow k$$

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

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

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

Vanishing Gradients and ReLUs

Hidden Layer 1 Weight Matrix

		①	②
③	+0.10	−0.20	+0.11
④	−0.19	+0.09	−0.13
⑤	+0.04	+0.11	−0.02

Input Layer

$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$
1.00	1.00	1.00	1.00
0.04	0.84	0.50	0.53
0.81	0.58	0.07	1.00

= $\mathbf{Z}^{(1)}$

+0.1811	−0.0042	+0.0077	+0.1040
−0.2917	−0.1898	−0.1541	−0.2723
+0.0282	+0.1208	+0.0936	+0.0783

φ Activations Hidden Layer 1

0.1811	0.0000	0.0077	0.1040
0.0000	0.0000	0.0000	0.0000
0.0282	0.1208	0.0936	0.0783

+ $\mathbf{d}[0] = 1$

Hidden Layer 2 Weight Matrix

		③	④	⑤
⑥	+0.15	−0.04	+0.10	+0.06
⑦	+0.12	+0.20	+0.14	−0.09

Activations Hidden Layer 1

1.0000	1.0000	1.0000	1.0000
0.1811	0.0000	0.0077	0.1040
0.0000	0.0000	0.0000	0.0000
0.0282	0.1208	0.0936	0.0783

= $\mathbf{Z}^{(2)}$

+0.144448	+0.157248	+0.155308	+0.150538
+0.153682	+0.109128	+0.113116	+0.133753

φ Activations Hidden Layer 2

0.1444	0.1572	0.1553	0.1505
0.1537	0.1091	0.1131	0.1338

+ $\mathbf{d}[0] = 1$

Output Layer Weight Matrix

		⑥	⑦
⑧	+0.10	+0.12	−0.50

Activations Hidden Layer 2

1.0000	1.0000	1.0000	1.0000
0.1444	0.1572	0.1553	0.1505
0.1537	0.1091	0.1131	0.1338

= $\mathbf{Z}^{(3)}$

+0.040478	+0.064314	+0.062086	+0.05116

φ **Activations Output Layer**

0.0405	0.0643	0.0621	0.0512

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.

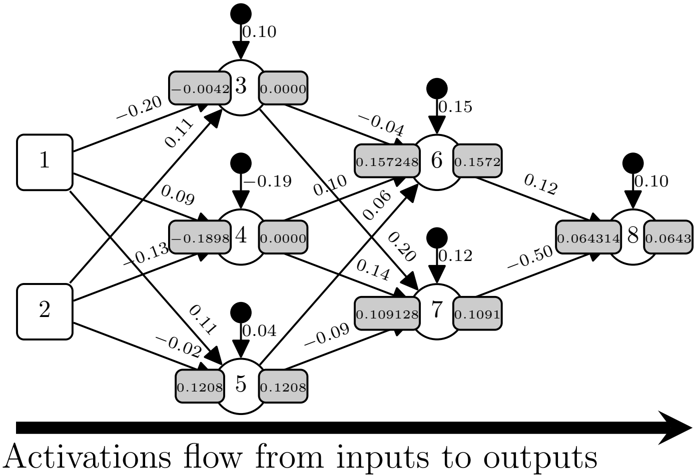


Figure 19: An illustration of the forward propagation of $\mathbf{d}_2$ through the ReLU network showing the weights on each connection, and the weighted sum z and activation a value for each neuron in the network.

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.

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$
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

Table 11: The $\partial a / \partial z$ for each neuron for $\mathbf{d}_2$ 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

$$
\begin{aligned}
\delta_k =& \frac{\partial \mathcal{E}}{\partial a_k} \times \frac{\partial a_k}{\partial z_i} \\
\delta_8 =& -0.0657 \times 1.0 \\
=& -0.0657 \\
\delta_7 =& \left(\delta_8 \times w_{8,7}\right) \times \frac{\partial a_7}{\partial z_7} \\
=& (-0.0657 \times -0.50) \times 1 \\
=& 0.0329 \\
\delta_6 =& \left(\delta_8 \times w_{8,6}\right) \times \frac{\partial a_6}{\partial z_6} \\
=& (-0.0657 \times 0.12) \times 1 \\
=& -0.0079 \\
\delta_5 =& \left(\left(\delta_6 \times w_{6,5}\right) + \left(\delta_7 \times w_{7,5}\right)\right) \times \frac{\partial a_5}{\partial z_5} \\
=& ((-0.0079 \times 0.06) + (0.0329 \times -0.09)) \times 1 \\
=& -0.0034 \\
\delta_4 =& \left(\left(\delta_6 \times w_{6,4}\right) + \left(\delta_7 \times w_{7,4}\right)\right) \times \frac{\partial a_4}{\partial z_4} \\
=& ((-0.0079 \times 0.10) + (0.0329 \times 0.14)) \times 0 \\
=& 0 \\
\delta_3 =& \left(\left(\delta_6 \times w_{6,3}\right) + \left(\delta_7 \times w_{7,3}\right)\right) \times \frac{\partial a_3}{\partial z_3} \\
=& ((-0.0079 \times -0.04) + (0.0329 \times 0.20)) \times 0 \\
=& 0
\end{aligned}
\tag{44}
$$

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

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$
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

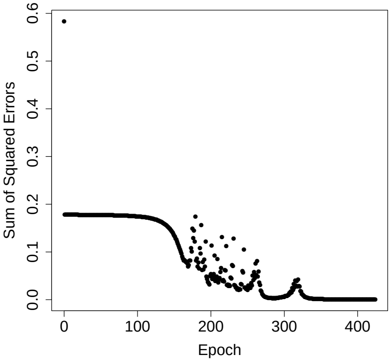


Figure 20: A plot showing how the sum of squared errors of the ReLU network changed during training when $\alpha = 0.2$.

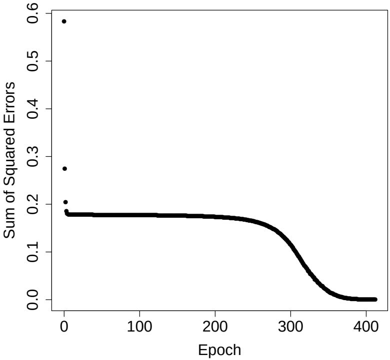


Figure 21: A plot showing how the sum of squared errors of the ReLU network changed during training when $\alpha = 0.1$.

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

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

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

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

$$\frac{\partial \mathcal{E}}{\partial \lambda_i} = \frac{\partial \mathcal{E}}{\partial a_i} \times \frac{\partial a_i}{\partial \lambda_i} \tag{49}$$

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

$$\lambda_i \leftarrow \lambda_i - \alpha \times \frac{\partial \mathcal{E}}{\partial \lambda_i} \tag{51}$$

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

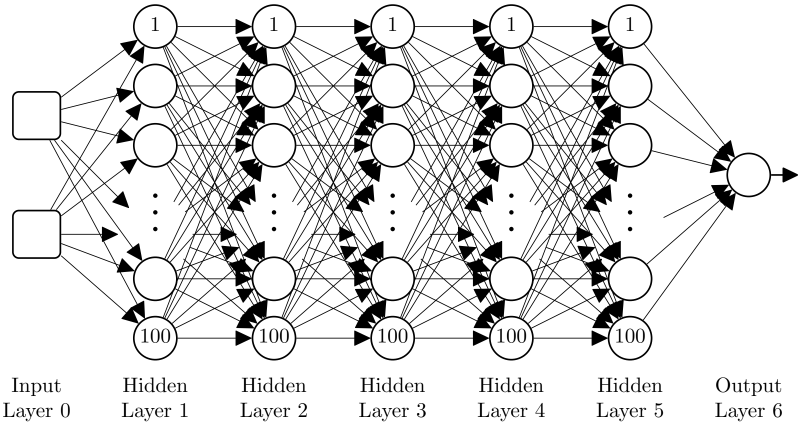
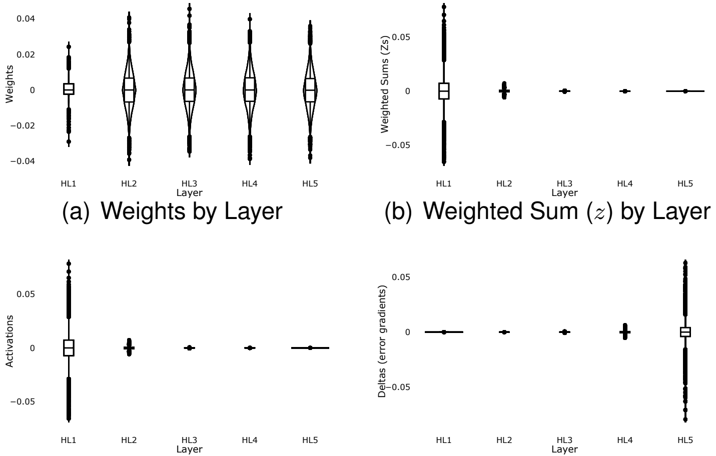
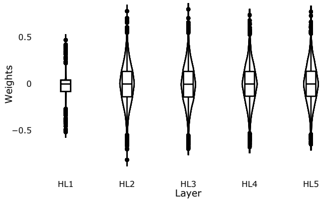


Figure 22: The architecture of the neural network used in the weight initialization experiments. Note that the neurons in this network use a linear activation function: $a_i = z_i$.

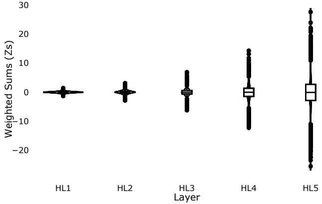


(a) Weights by Layer
(b) Weighted Sum (z) by Layer
(c) Activations by Layer
(d) δs by Layer

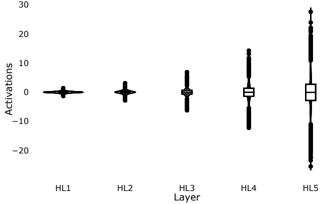
Figure 23: The internal dynamics of the network in Figure 22[73] during the first training iteration when the weights were initialized using a normal distribution with $\mu{=}0.0, \sigma{=}0.01$.



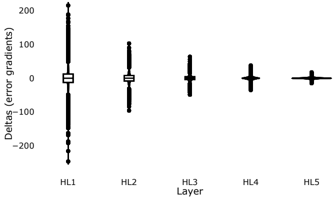
(a) Weights by Layer



(b) Weighted Sum (z) by Layer



(c) Activations by Layer



(d) δs 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 $\sigma = 0.2$.

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

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

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

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

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

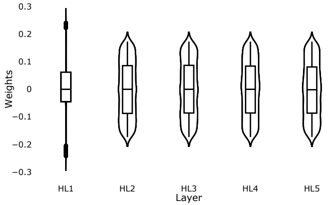
$$var(z) = \sum_{i=1}^{n_{in}} var(w_i \times d_i) = n_{in}\ var(\mathbf{W})\ var(\mathbf{d}) \tag{58}$$

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

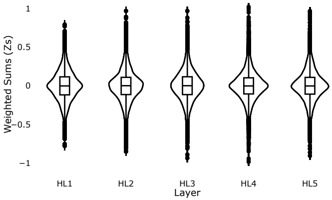
$$
\begin{aligned}
var(Z^{(HL2)}) &= n_{in}^{(HL2)} \times var(\mathbf{W}^{(HL2)}) \times var(\mathbf{d}^{(HL2)}) \\
&= 100 \times 0.0001 \times 0.0002 \\
&= 0.000002
\end{aligned}
\tag{60}
$$

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

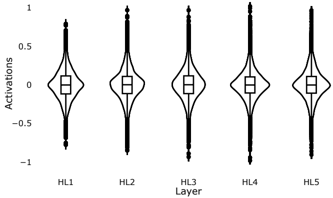
$$var(\mathbf{W}^{(k)}) = \frac{1}{n_{in}^{(k)}} \tag{62}$$



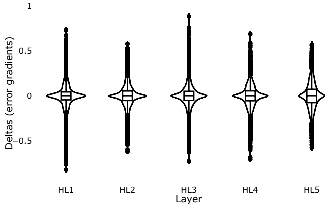
(a) Weights by Layer



(b) Weighted Sum (z) by Layer



(c) Activations by Layer

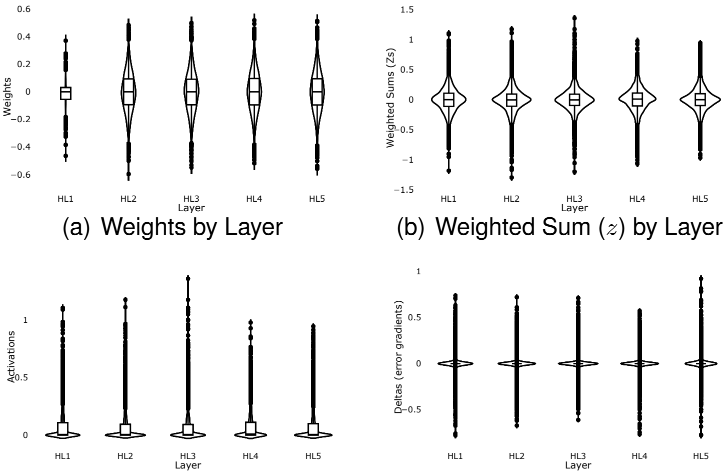


(d) δs by Layer

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.

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

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



(a) Weights by Layer

(b) Weighted Sum (z) by Layer

(c) Activations by Layer

(d) δs 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.

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

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	RELATIVE HUMIDITY	Electrical Output		
	°C	%	*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

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

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

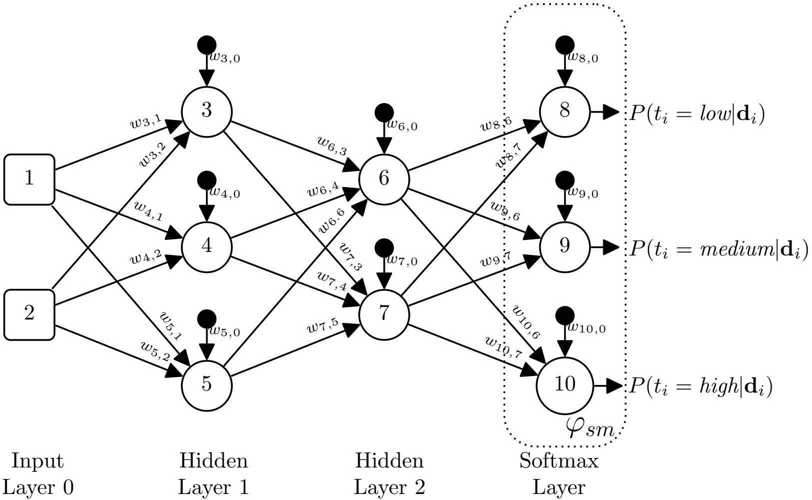


Figure 27: A schematic of a feedforward artificial neural network with a three-neuron softmax output layer.

$$L_{CE}\left(\mathbf{t},\hat{\mathbf{P}}\right) = -\sum_{j} \mathbf{t}_j \ln\left(\hat{\mathbf{P}}_j\right) \tag{66}$$

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

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

$$\boldsymbol{\delta}_k = \frac{\partial \mathcal{E}}{\partial z_k} \tag{69}$$

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

$$= \frac{\partial - \ln\left(\hat{\mathbf{P}}_\star\right)}{\partial \mathbf{l}_k} \tag{71}$$

$$= \frac{\partial - \ln\left(\hat{\mathbf{P}}_\star\right)}{\partial\left(\hat{\mathbf{P}}_\star\right)} \times \frac{\partial\left(\hat{\mathbf{P}}_\star\right)}{\partial \mathbf{l}_k} \tag{72}$$

$$\frac{d \ln x}{dx} = \frac{1}{x} \tag{73}$$

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

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

$$\boldsymbol{\delta}_k = \frac{\partial - \ln\left(\hat{\mathbf{P}}_\star\right)}{\partial\left(\hat{\mathbf{P}}_\star\right)} \times \frac{\partial\left(\hat{\mathbf{P}}_\star\right)}{\partial \mathbf{l}_k} \tag{76}$$

$$= -\frac{1}{\hat{\mathbf{P}}_\star} \times \frac{\partial\left(\hat{\mathbf{P}}_\star\right)}{\partial \mathbf{l}_k} \tag{77}$$

$$= -\frac{1}{\hat{\mathbf{P}}_\star} \times \begin{cases} \hat{\mathbf{P}}_\star\left(1 - \hat{\mathbf{P}}_k\right) & \text{if } k = \star \\ -\hat{\mathbf{P}}_\star \hat{\mathbf{P}}_k & otherwise \end{cases} \tag{78}$$

$$= \begin{cases} -\left(1 - \hat{\mathbf{P}}_k\right) & \text{if } k = \star \\ \hat{\mathbf{P}}_k & otherwise \end{cases} \tag{79}$$

$$\boldsymbol{\delta}_{k=\star} = -\left(1 - \hat{\mathbf{P}}_k\right) \tag{80}$$

$$\boldsymbol{\delta}_{k\neq\star} = \hat{\mathbf{P}}_k \tag{81}$$

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

$$\underbrace{\begin{array}{c} \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{ccc} & 1 & 2 \\ \hline +0.10 & -0.02 & +0.07 \\ +0.10 & +0.06 & +0.18 \\ +0.10 & +0.05 & -0.04 \end{array}}_{\text{Hidden Layer 1 Weight Matrix}} \underbrace{\begin{array}{cccc} \mathbf{d}_1 & \mathbf{d}_2 & \mathbf{d}_3 & \mathbf{d}_4 \\ \hline 1.00 & 1.00 & 1.00 & 1.00 \\ 0.04 & 0.84 & 0.50 & 0.53 \\ 0.81 & 0.58 & 0.07 & 1.00 \end{array}}_{\text{Input Layer}} = \overset{\mathbf{Z}^{(1)}}{\begin{array}{|c|c|c|c|} \hline 0.1559 & 0.1238 & 0.0949 & 0.1594 \\ 0.1672 & 0.1968 & 0.1356 & 0.2118 \\ 0.0696 & 0.1188 & 0.1222 & 0.0865 \\ \hline \end{array}} \;\varphi_{ReLU}\; \overset{\text{Activations Hidden Layer 1}}{\begin{array}{|c|c|c|c|} \hline 0.1559 & 0.1238 & 0.0949 & 0.1594 \\ 0.1672 & 0.1968 & 0.1356 & 0.2118 \\ 0.0696 & 0.1188 & 0.1222 & 0.0865 \\ \hline \end{array}}$$

$+ \mathbf{d}[0] = 1$

$$\underbrace{\begin{array}{c} \\ 6 \\ 7 \end{array}\begin{array}{cccc} & 3 & 4 & 5 \\ \hline +0.10 & -0.03 & +0.09 & +0.06 \\ +0.10 & +0.20 & -0.01 & -0.09 \end{array}}_{\text{Hidden Layer 2 Weight Matrix}} \underbrace{\begin{array}{|c|c|c|c|} \hline 1.0000 & 1.0000 & 1.0000 & 1.0000 \\ 0.1559 & 0.1238 & 0.0949 & 0.1594 \\ 0.1672 & 0.1968 & 0.1356 & 0.2118 \\ 0.0696 & 0.1188 & 0.1222 & 0.0865 \\ \hline \end{array}}_{\text{Activations Hidden Layer 1}} = \overset{\mathbf{Z}^{(2)}}{\begin{array}{|c|c|c|c|} \hline +0.114547 & +0.121126 & 0.116689 & 0.1194700 \\ +0.123244 & +0.112100 & 0.106626 & 0.1219770 \\ \hline \end{array}} \;\varphi_{ReLU}\; \overset{\text{Activations Hidden Layer 2}}{\begin{array}{|c|c|c|c|} \hline 0.1145 & 0.1211 & 0.1167 & 0.1195 \\ 0.1232 & 0.1121 & 0.1066 & 0.1220 \\ \hline \end{array}}$$

$+ \mathbf{d}[0] = 1$

$$\underbrace{\begin{array}{c} \\ 8 \\ 9 \\ 10 \end{array}\begin{array}{ccc} & 6 & 7 \\ \hline +0.10 & -0.03 & +0.09 \\ +0.10 & -0.07 & +0.06 \\ +0.10 & +0.08 & -0.05 \end{array}}_{\text{Output Layer Weight Matrix}} \underbrace{\begin{array}{|c|c|c|c|} \hline 1.0000 & 1.0000 & 1.0000 & 1.0000 \\ 0.1145 & 0.1211 & 0.1167 & 0.1195 \\ 0.1232 & 0.1121 & 0.1066 & 0.1220 \\ \hline \end{array}}_{\text{Activations Hidden Layer 2}} = \overset{\textbf{Logits}}{\begin{array}{|c|c|c|c|} \hline +0.107653 & +0.106456 & +0.106093 & +0.107395 \\ +0.099377 & +0.098249 & +0.098227 & +0.098955 \\ +0.103000 & +0.104083 & +0.104006 & +0.103460 \\ \hline \end{array}} \;\varphi_{SM}\; \overset{\textbf{Softmax Activations Output Layer}}{\begin{array}{|c|c|c|c|} \hline 0.3348 & 0.3345 & 0.3344 & 0.3347 \\ 0.3320 & 0.3318 & 0.3318 & 0.3319 \\ 0.3332 & 0.3337 & 0.3337 & 0.3334 \\ \hline \end{array}}$$

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.

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$
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 $e^{\mathbf{l}_i}$				
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
$\sum_i e^{\mathbf{l}_i}$	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 δs				
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

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

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

Algorithm 1 The early stopping algorithm

Require: p the patience parameter
Require: $\mathcal{D}_\nu$ a validation set

bestValidationError $= \infty$
tmpValidationError $= 0$
$\theta =$ initial model parameters
$\theta^{best} = 0$
patienceCount = 0
while patienceCount $< p$ **do**
  $\theta =$ new model parameters after most recent weight update
  tmpValidationError = calculateValidationError(θ, $\mathcal{D}_\nu$)
  if bestValidationError $\geq$ tmpValidationError **then**
    bestValidationError = tmpValidationError
    $\theta^{best} = \theta$
    patienceCount = 0
  else
    patienceCount = patienceCount + 1
  end if
end while
return Best Model Parameters θ^{best}

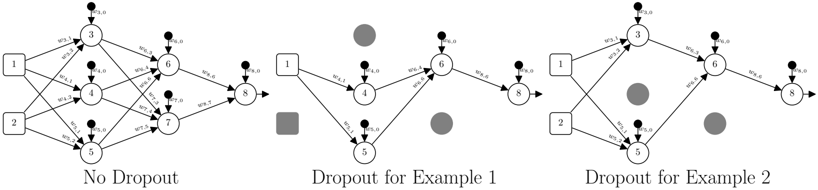


Figure 29: An illustration of how different small networks are generated for different training examples by applying dropout to the original large network. The gray nodes mark the neurons that have been dropped from the network for the training example.

Algorithm 2 Extensions to Backpropagation to Use Inverted Dropout

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

▷ Forward Pass

for each input or hidden layer l **do**

$\quad \mathbf{DropMask}^{(l)} = (m_1, \ldots, m_{size(l)}) \sim Bernoulli(\rho)$

$\quad \mathbf{a}^{(l)'} = \mathbf{a}^{(l)} \odot \mathbf{DropMax}^{(l)}$

$\quad \mathbf{a}^{(l)''} = \frac{1}{\rho}\mathbf{a}^{(l)'}$

end for

▷ Backward Pass

for each layer l in backward pass **do**

$\quad \delta^{(l)} = \delta^{(l)} \odot \mathbf{DropMax}^{(l)}$

end for

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

$$
\begin{bmatrix}
000 & 000 & 000 & 000 & 000 & 000 \\
000 & \mathbf{255} & 000 & 000 & 000 & 000 \\
000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 \\
000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 \\
000 & 000 & 000 & \mathbf{255} & 000 & 000 \\
000 & 000 & 000 & 000 & 000 & 000
\end{bmatrix} \tag{84}
$$

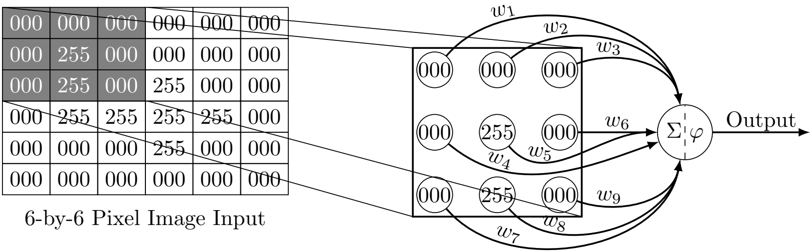


Figure 31: A 6-by-6 matrix representation of a grayscale image of a 4, and a neuron with a receptive field that covers the top-left corner of the image. This figure was inspired by Figure 2 of (Kelleher and Dobnik, 2017).

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{85}$$

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

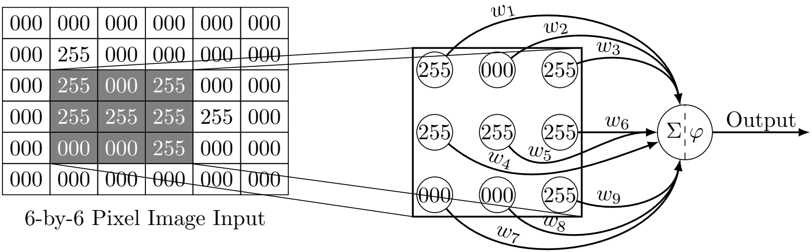


Figure 32: A 6-by-6 matrix representation of a grayscale image of a 4, and a neuron with a different receptive field from the neuron in Figure 31[107]. This figure was inspired by Figure 2 of (Kelleher and Dobnik, 2017).

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

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

$$
\begin{aligned}
\Delta w_{i,*} &= \sum_{i=1}^{m} \boldsymbol{\delta}_i \times a_* \\
w_{i,*} &\leftarrow w_{i,*} - \alpha \times \Delta w_{i,*}
\end{aligned}
\tag{89}
$$

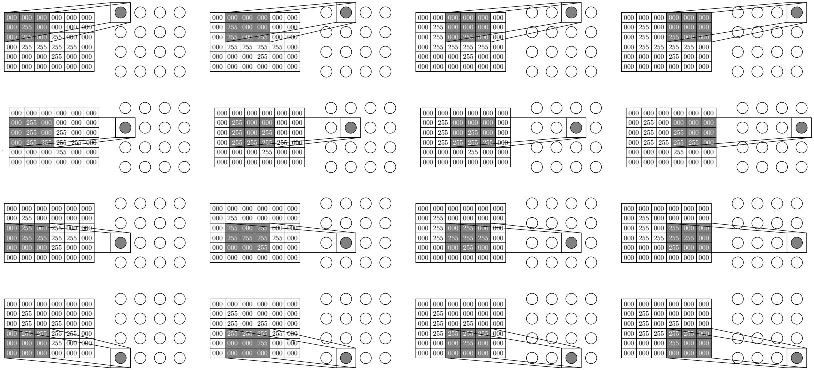


Figure 33: Illustration of the organization of a set of neurons that share weights (use the same filter) and their local receptive fields such that together the receptive fields cover the entirety of the input image.

$$\underbrace{\begin{bmatrix} 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 \\ 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 \\ 000 & 000 & 000 & \mathbf{255} & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 \end{bmatrix}}_{\text{Input Image}} \rightarrow \underbrace{\begin{bmatrix} -1 & +1 & -1 \\ -1 & +1 & -1 \\ -1 & +1 & -1 \end{bmatrix}}_{\text{Convolved Filter}} \rightarrow \underbrace{\begin{bmatrix} 510 & 0 & 255 & 0 \\ 510 & 0 & 0 & 0 \\ 255 & 0 & 255 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{Feature Map}} \tag{90}$$

$$\underbrace{\begin{bmatrix} 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & 000 & 000 & 000 \\ 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 \\ 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 \\ 000 & 000 & 000 & \mathbf{255} & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 \end{bmatrix}}_{\text{Input Image}} \rightarrow \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ +1 & +1 & +1 \\ -1 & -1 & -1 \end{bmatrix}}_{\text{Convolved Filter}} \rightarrow \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 255 & 0 & 255 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{Feature Map}} \tag{91}$$

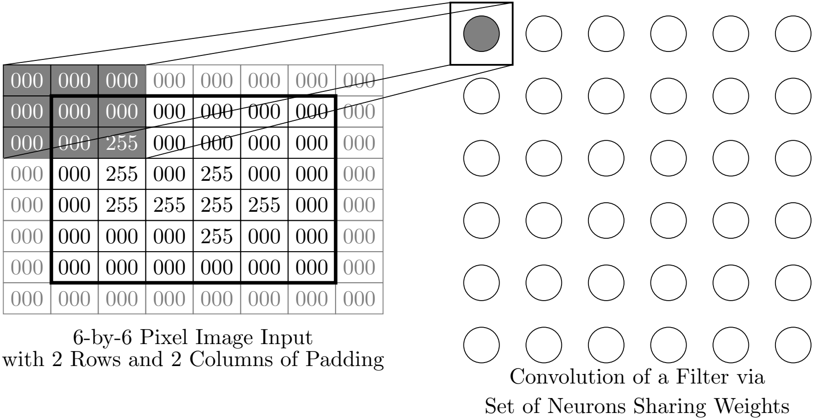


Figure 34: A grayscale image of a 4 after padding has been applied to the original 6-by-6 matrix representation, and the local receptive field of a neuron that includes both valid and padded pixels.

$$\underbrace{\begin{bmatrix} 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 & 000 \\ 000 & 000 & 000 & 000 & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \end{bmatrix}}_{\text{Input Image}} \rightarrow \underbrace{\begin{bmatrix} -1 & +1 & -1 \\ -1 & +1 & -1 \\ -1 & +1 & -1 \end{bmatrix}}_{\text{Convolved Filter}} \rightarrow \underbrace{\begin{bmatrix} 0 & 255 & 0 & 0 & 0 & 0 \\ 0 & 510 & 0 & 255 & 0 & 0 \\ 0 & 510 & 0 & 0 & 0 & 0 \\ 0 & 255 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 0 & 0 \end{bmatrix}}_{\text{Feature Map}} \tag{92}$$

$$\underbrace{\begin{bmatrix} 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 & 000 \\ 000 & 000 & 000 & 000 & \mathbf{255} & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 000 \end{bmatrix}}_{\text{Input Image}} \rightarrow \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ +1 & +1 & +1 \\ -1 & -1 & -1 \end{bmatrix}}_{\text{Convolved Filter}} \rightarrow \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 0 & 255 & 0 & 255 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{Feature Map}} \tag{93}$$

Convolutional Neural Networks

$$
\underbrace{\begin{bmatrix}
000 & 000 & 000 & 000 & 000 & 000 & 000 & 000\\
000 & 000 & 000 & 000 & 000 & 000 & 000 & 000\\
000 & 000 & \mathbf{255} & 000 & 000 & 000 & 000 & 000\\
000 & 000 & \mathbf{255} & 000 & \mathbf{255} & 000 & 000 & 000\\
000 & 000 & \mathbf{255} & \mathbf{255} & \mathbf{255} & \mathbf{255} & 000 & 000\\
000 & 000 & 000 & 000 & \mathbf{255} & 000 & 000 & 000\\
000 & 000 & 000 & 000 & 000 & 000 & 000 & 000\\
000 & 000 & 000 & 000 & 000 & 000 & 000 & 000
\end{bmatrix}}_{\text{Input Image}}
\rightarrow
\underbrace{\begin{bmatrix}
-1 & -1 & -1\\
+1 & +1 & +1\\
-1 & -1 & -1
\end{bmatrix}}_{\text{Convolved Filter}}
\rightarrow
$$

$$
\rightarrow
\underbrace{\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0\\
0 & 255 & 0 & 255 & 0 & 255\\
0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}}_{\text{Feature Map}}
\rightarrow \text{max pooling} \rightarrow
\underbrace{\begin{bmatrix}
0 & 0 & 0\\
255 & 255 & 255\\
0 & 0 & 0
\end{bmatrix}}_{\text{Output from Sub-sampling}}
\tag{94}
$$

$$\underbrace{\begin{bmatrix} O & O & O & O & O & O \\ O & O & O & O & O & O \\ A & O & O & O & O & O \\ O & B & O & O & O & O \\ O & O & O & O & O & O \\ O & O & O & O & O & O \end{bmatrix}}_{\text{Neurons that convolve the Filter}} \rightarrow \underbrace{\begin{bmatrix} O & O & O \\ C & O & O \\ O & O & O \end{bmatrix}}_{\text{Sub-sampling Layer}} \rightarrow \underbrace{\begin{bmatrix} D \\ E \end{bmatrix}}_{\text{Fully Connected Layer}} \tag{95}$$

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

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

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

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

$$\left[\underbrace{\begin{bmatrix}1 & 1 & 1\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}}_{\text{Red Channel}}\underbrace{\begin{bmatrix}0 & 0 & 2\\0 & 0 & 2\\0 & 0 & 2\end{bmatrix}}_{\text{Green Channel}}\underbrace{\begin{bmatrix}3 & 0 & 0\\0 & 3 & 0\\0 & 0 & 3\end{bmatrix}}_{\text{Blue Channel}}\right] \tag{100}$$

$$\begin{bmatrix} \underbrace{\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}}_{\text{Red Channel}} & \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}_{\text{Green Channel}} & \underbrace{\begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}}_{\text{Blue Channel}} \end{bmatrix} \tag{101}$$

$$
\begin{aligned}
z &= ((w_0 \times 1) \\
&\quad + (w_1 \times 1) + (w_2 \times 1) + (w_3 \times 0) + (w_4 \times 0) \\
&\quad + (w_5 \times 0) + (w_6 \times 0) + (w_7 \times 0) + (w_8 \times 0) \\
&\quad + (w_9 \times 3) + (w_{10} \times 0) + (w_{11} \times 0) + (w_{12} \times 3)) \\
&= 0.5 + 1 + 1 + 0 + 0 + 0 + 0 + 0 + 0 + 3 + 0 + 0 + 3 \\
&= 8.5 \\
a &= rectifier(z) \\
&= rectifier(8.5) \\
&= 8.5
\end{aligned} \tag{102}
$$

$$\begin{bmatrix} 8.5 & 6.5 \\ 0.5 & 10.5 \end{bmatrix} \tag{103}$$

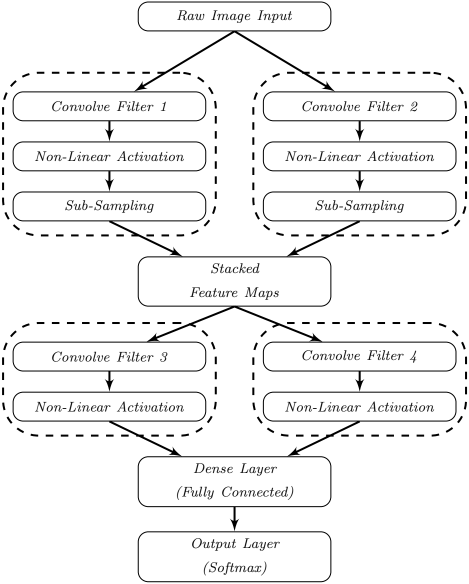


Figure 35: Schematic of the typical sequences of layers found in a convolutional neural network.

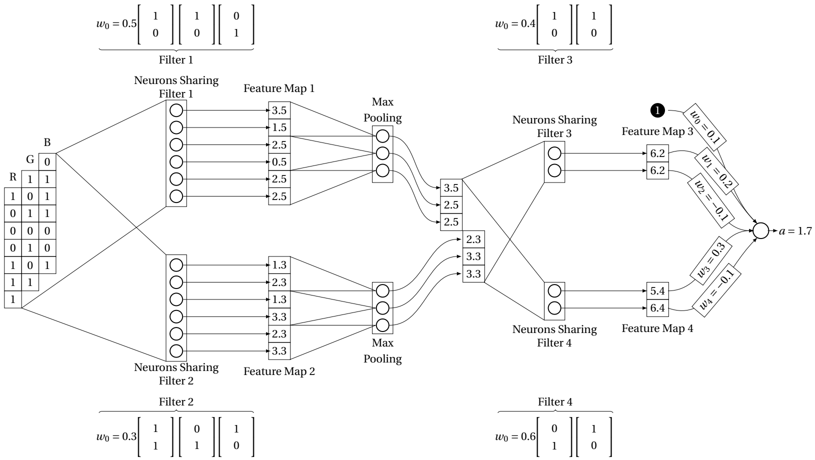


Figure 36: Worked example illustrating the dataflow through a multilayer, multifilter CNN.

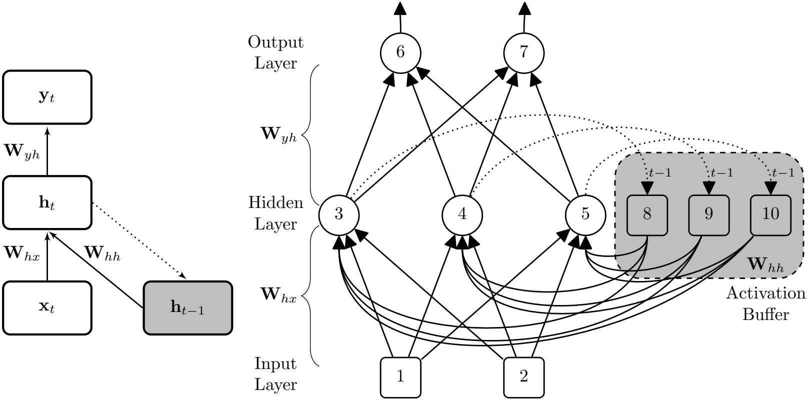


Figure 37: Schematic of the simple recurrent neural architecture.

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

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

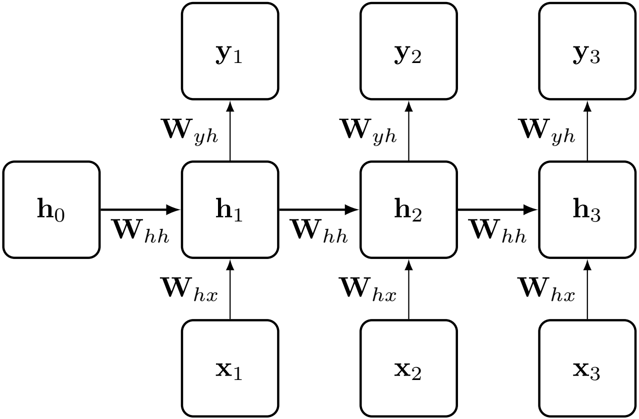


Figure 38: A simple RNN model unrolled through time (in this instance, three time-steps).

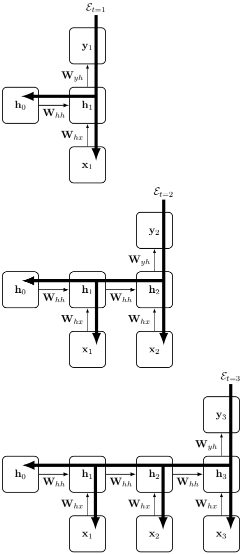


Figure 39: An illustration of the different iterations of backpropagation during backpropagation through time.

Algorithm 3 The Backpropagation Through Time Algorithm

Require: h_0 initialized hidden state
Require: $\mathbf{x}$ a sequence of inputs
Require: $\mathbf{y}$ a sequence of target outputs
Require: n length of the input sequence
Require: Initialized weight matrices (with associated biases)
Require: $\Delta\mathbf{w}$ a data structure to accumulate the summed weight updates for each weight across time-steps

for $t = 1$ to n **do**
    $Inputs = [x_0, \ldots, x_t]$
    $h_{tmp} = h_0$
    for $i = 0$ to t **do** ▷ Unroll the network through t steps
        $h_{tmp} = ForwardPropagate(Inputs[i], h_{tmp})$
    end for
    $\hat{y}_t = OutputLayer(h_{tmp})$ ▷ Generate the output for time-step t
    $\mathcal{E}_t = \mathbf{y}[t] - \hat{y}_t$ ▷ Calculate the error at time-step t
    $Backpropagate(\mathcal{E}_t)$ ▷ Backpropagate $\mathcal{E}_t$ through t steps
    For each weight, sum the weight updates across the unrolled network and update $\Delta\mathbf{w}$
end for
Update the network weights using $\Delta\mathbf{w}$

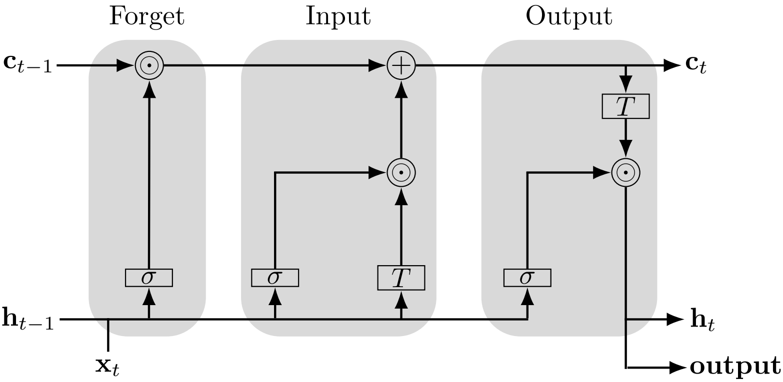


Figure 40: A schematic of the internal structure of a long short-term memory unit. This figure is based on Figure 5.4 of (Kelleher, 2019), which in turn was inspired by an image by Christopher Olah (available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/).

$$\mathbf{f}_t = \varphi_{sigmoid}(\mathbf{W}^{(f)} \cdot \mathbf{hx}_t) \tag{106}$$

$$\mathbf{c}\ddagger = \mathbf{c}_{t-1} \odot \mathbf{f}_t \tag{107}$$

$$\mathbf{c}_{t-1} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \mathbf{h}_{t-1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \mathbf{x}_t = \begin{bmatrix} 4 \end{bmatrix}$$
$$\mathbf{W}^{(f)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{108}$$

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

$$\underbrace{\begin{bmatrix} 0.997527377 \\ 0.002472623 \\ 0.500000000 \end{bmatrix}}_{\mathbf{f}_t} \odot \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\mathbf{c}_{t-1}} = \underbrace{\begin{bmatrix} 0.997527377 \\ 0.002472623 \\ 0.500000000 \end{bmatrix}}_{\mathbf{c}\ddagger} \tag{109}$$

$$\mathbf{i}\dagger_t = \varphi_{sigmoid}(\mathbf{W}^{(i\dagger)} \cdot \mathbf{hx}_t) \quad (110)$$

$$\mathbf{i}\ddagger_t = \varphi_{tanh}(\mathbf{W}^{(i\ddagger)} \cdot \mathbf{hx}_t) \quad (111)$$

$$\mathbf{i}_t = \mathbf{i}\dagger_t \odot \mathbf{i}\ddagger_t \quad (112)$$

$$\mathbf{c}_t = \mathbf{c}\ddagger + \mathbf{i}_t \quad (113)$$

$$\mathbf{o}\dagger_t = \varphi_{sigmoid}(\mathbf{W}^{(o\dagger)} \cdot \mathbf{hx}_t) \tag{114}$$

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

$$\mathbf{o}_t = \mathbf{o}\dagger_t \odot \mathbf{o}\ddagger_t \tag{116}$$

$$\mathbf{h}_{t+1} = \mathbf{o}_t \tag{117}$$

$$\begin{aligned}
\mathbf{f}_t &= \varphi_{sigmoid}(\mathbf{W}^{(f)} \cdot \mathbf{hx}_t) \\
\mathbf{c}\ddagger &= \mathbf{c}_{t-1} \odot \mathbf{f}_t \\
\mathbf{i}\dagger_t &= \varphi_{sigmoid}(\mathbf{W}^{(i\dagger)} \cdot \mathbf{hx}_t) \\
\mathbf{i}\ddagger_t &= \varphi_{tanh}(\mathbf{W}^{(i\ddagger)} \cdot \mathbf{hx}_t) \\
\mathbf{i}_t &= \mathbf{i}\dagger_t \odot \mathbf{i}\ddagger_t \\
\mathbf{c}_t &= \mathbf{c}\ddagger + \mathbf{i}_t \\
\mathbf{o}\dagger_t &= \varphi_{sigmoid}(\mathbf{W}^{(o\dagger)} \cdot \mathbf{hx}_t) \\
\mathbf{o}\ddagger_t &= \varphi_{tanh}(\mathbf{W}^{(o\ddagger)} \cdot \mathbf{c}_t) \\
\mathbf{o}_t &= \mathbf{o}\dagger_t \odot \mathbf{o}\ddagger_t \\
\mathbf{h}_{t+1} &= \mathbf{o}_t
\end{aligned}$$

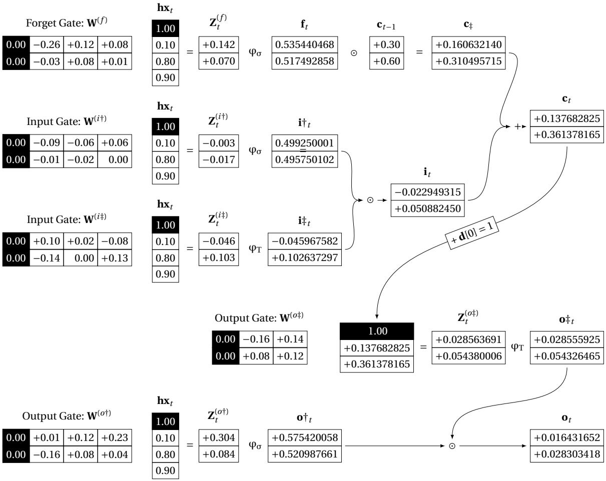


Figure 41: The flow of activations through a long short-term memory unit during forward propagation when $\mathbf{c}_{t-1} = [0.3, 0.6]$, $\mathbf{h}_t = [0.1, 0.8]$, and $\mathbf{x}_t = [0.9]$.

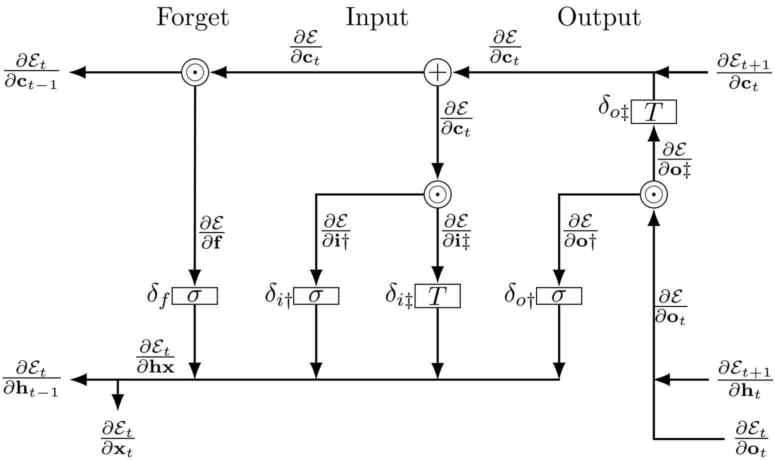


Figure 42: The flow of error gradients through a long short-term memory unit during backpropagation.

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

$$\frac{\partial \mathcal{E}}{\partial \mathbf{o}\ddagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}_t} \odot \mathbf{o}\dagger \tag{119}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{o}\dagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}_t} \odot \mathbf{o}\ddagger \tag{120}$$

$$\delta_{o\ddagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}\ddagger} \odot \frac{\partial \mathbf{o}\ddagger_t}{\partial \mathbf{c}_{t\ddagger}} = \frac{\partial \mathcal{E}}{\partial \mathbf{o}\ddagger} \odot \underbrace{\left(\mathbf{1} - tanh^2(\mathbf{c}\ddagger_t)\right)}_{\text{Derivate of tanh, i.e.:}\frac{\partial a}{\partial z}} \tag{121}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} = \delta_{o\ddagger} + \frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{c}_t} \tag{122}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{i}\ddagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} \odot \mathbf{i}\dagger \tag{123}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{i}\dagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{c}_t} \odot \mathbf{i}\ddagger \tag{124}$$

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

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

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

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

$$\delta_{i\ddagger} = \frac{\partial \mathcal{E}}{\partial \mathbf{i}\ddagger} \odot \frac{\partial \mathbf{i}\ddagger_t}{\partial \mathbf{z}_{i\ddagger}} = \frac{\partial \mathcal{E}}{\partial \mathbf{i}\ddagger} \odot \left(\mathbf{1} - tanh^2(\mathbf{i}\ddagger_t)\right) \tag{129}$$

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

$$\Delta \mathbf{W}^{(f)} = \delta_f \cdot \mathbf{hx}^\intercal \quad (131)$$

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

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

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

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

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

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

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

$$\Delta \mathbf{W}^{(f)} = \underbrace{\begin{bmatrix} 0.064732317 \\ 0.141057014 \end{bmatrix}}_{\delta_f} \cdot \underbrace{\begin{bmatrix} 1.00 & 0.10 & 0.80 & 0.90 \end{bmatrix}}_{\mathbf{hx}^\intercal}$$

$$= \begin{bmatrix} 0.064732317 & 0.006473232 & 0.051785854 & 0.058259085 \\ 0.141057014 & 0.014105701 & 0.112845611 & 0.126951313 \end{bmatrix} \tag{138}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{hx}} = \left(\mathbf{W}^{(f)\intercal} \cdot \delta_f\right) + \left(\mathbf{W}^{(i\dagger)\intercal} \cdot \delta_{i\dagger}\right) + \left(\mathbf{W}^{(i\ddagger)\intercal} \cdot \delta_{i\ddagger}\right) + \left(\mathbf{W}^{(o\dagger)\intercal} \cdot \delta_{o\dagger}\right) \tag{139}$$

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

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).

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.