ECE 1508: Applied Deep Learning

Chapter 2: Feedforward Neural Networks

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2025

Various Architectures for NNs

Let's abbreviate the term *Neural Network* from now on with *NN*

Now that we know *what they are* and *how to train them*, we go through

1. *Feedforward NNs* abbreviated as FNNs
   - ↳ Some people call them also *Multi-Layer Perceptrons (MLPs)*: *you may say that this is a misnomer and you are right! Check the wikipedia page*
2. *Convolutional NNs* abbreviated as CNNs
3. *Recurrent NNs* abbreviated as RNNs

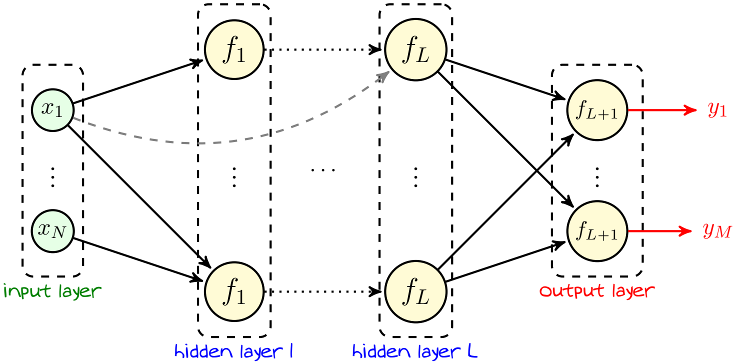
In this chapter, we start with FNNs which are known to be

vanilla NNs,

i.e., the most basic architecture we could think for a NN

FNNs: *Architecture*

In FNNs, the *inputs flow in one direction:* each layer's output is connected to the next layers, and thus *we do not have any feedback*



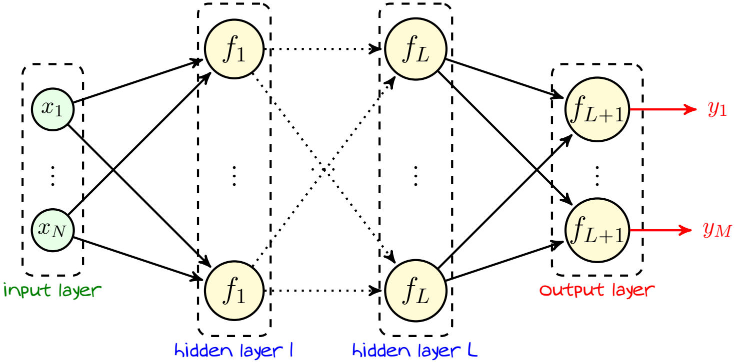
Though it is not a must, we usually use same activation for all neurons in a layer

Fully-Connected FNNs

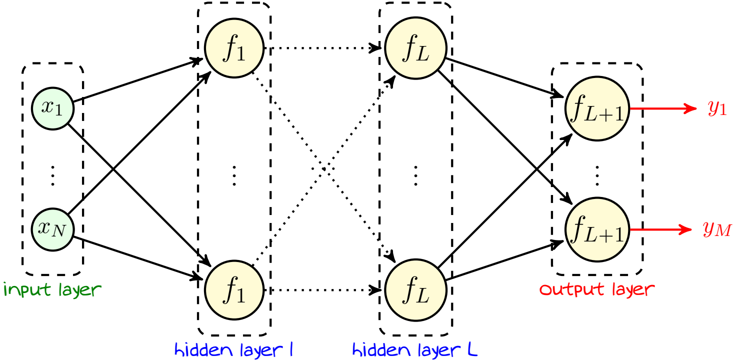
We start with the most straightforward FNNs: *fully-connected FNNs*

Fully-Connected FNNs

In a fully-connected FNN, each node is connected to all nodes in the next layer



Fully-Connected FNNs: *Few Definitions*



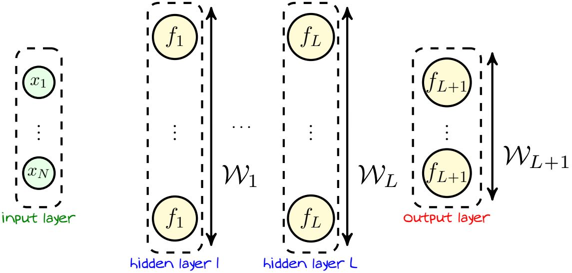
In this FNN, we have L hidden layers; thus, its depth is $L + 1$

Recall: *this network is Deep if $L > 1$*

Fully-Connected FNNs: *Few Definitions*

Width of a Layer

The width of layer ℓ is the number of neurons in layer ℓ



Some people call the largest width, the width of the network, i.e.,

$$\mathcal{W} = \max_{\ell \in \{1,\ldots,L+1\}} \mathcal{W}_\ell$$

Fully-Connected FNNs: *Looking as a Model*

+ *We should look at a fully-connected FNN as a model. Then, what are the hyperparameters and learnable parameters?*
– I am glad that you ask! Let's take a look

Assume that someone tells us that we should *use a fully-connected FNN* with only *ReLU activation*. Now, we could say

- To write down the model, we need to know *the number of hidden layers* L *and width of each layer* $\mathcal{W}_\ell$: these are the *hyperparameters*
- If we set the L and $\mathcal{W}_\ell$, we can specify the *learnable parameters*
  - *in hidden layer* 1, *we have* $\mathcal{W}_1$ *neurons each having* N *weights and a bias*
  - *in hidden layer* 2, *we have* $\mathcal{W}_2$ *neurons each having* $\mathcal{W}_1$ *weights and a bias*

    $\vdots$
  - *in output layer, we have* $\mathcal{W}_{L+1}$ *neurons each having* $\mathcal{W}_L$ *weights and a bias*

$$\text{\# model parameters} = (N+1)\,\mathcal{W}_1 + \sum_{\ell=1}^{L} (\mathcal{W}_\ell + 1)\,\mathcal{W}_{\ell+1}$$

Fully-Connected FNNs: *Forward Pass*

Let us first see *how a given data-point propagates through the FNN*: we want to write the outputs $y_1, \ldots, y_M$ when inputs $x_1, \ldots, x_N$ are given

this is called *forward propagation* through the network

or simply

the *forward pass*

which tracks values *passed* through the NN from the *input* to output layer

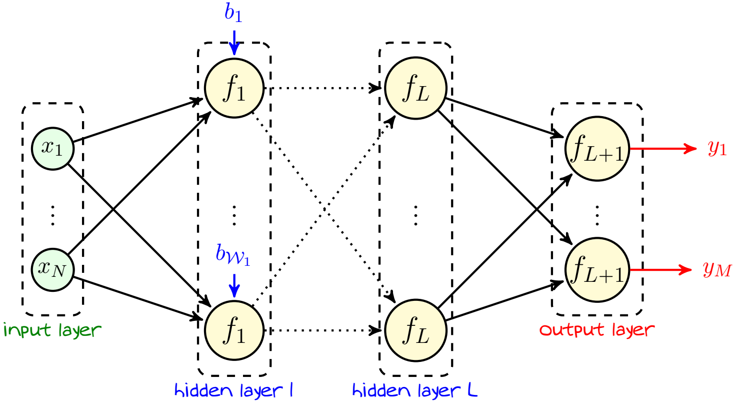
---

To present forward pass compactly, we need to *define some notations* and apply *some modifications* in the network

Fully-Connected FNNs: *Few Definitions*

We can get rid of biases by defining a new constant node in each layer

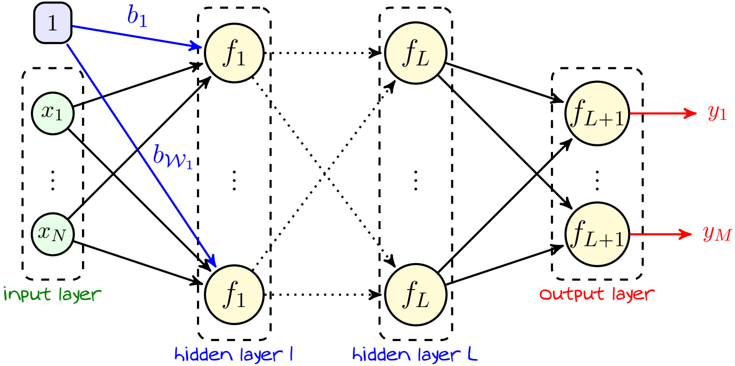
Let's look at the first layer: *we have $\mathcal{W}_1$ neurons and each has a bias*



Fully-Connected FNNs: *Few Definitions*

We can get rid of biases by defining a new constant node in each layer

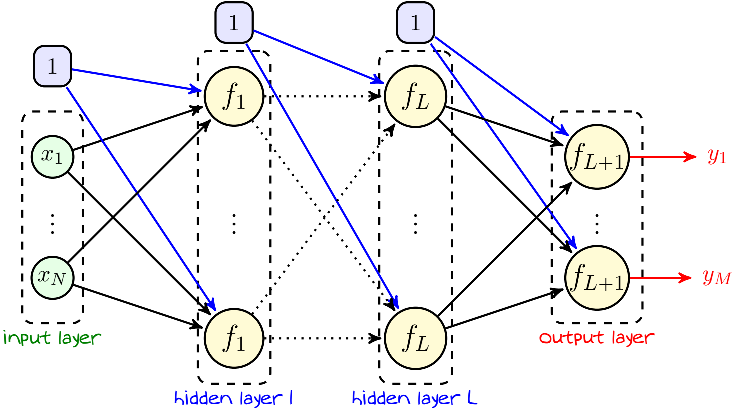
We introduce a constant input and let these biases being the weights of its links



Fully-Connected FNNs: *Few Definitions*

We can get rid of biases by defining a new constant node in each layer

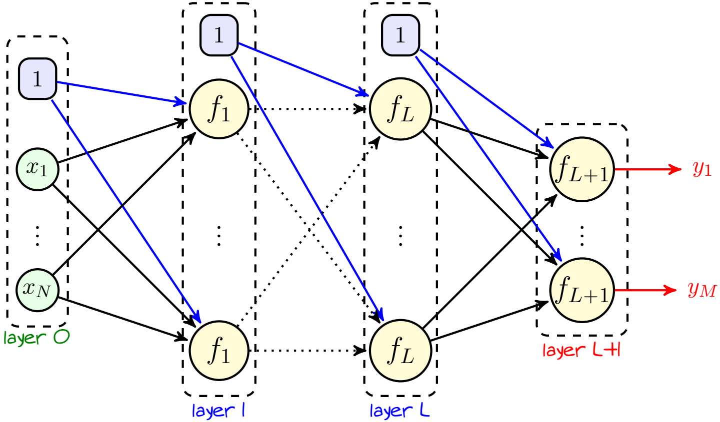
We do the same in all layers: now neurons have no biases



Fully-Connected FNNs: *Few Definitions*

We next give an index to each layer each layer

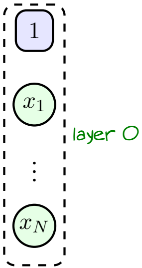
- Input layer is *layer 0*
- Hidden layer ℓ is *layer* ℓ
- Output layer is *layer* $L+1$

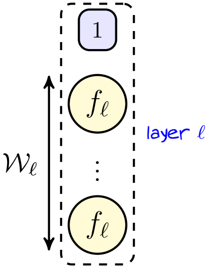


Fully-Connected FNNs: *Few Definitions*

So, our layers are indexed by $\ell \in \{0, \dots, L+1\}$

- *We denote the width of layer ℓ with $\mathcal{W}_\ell$*
  - ↳ *For $\ell \geqslant 1$ this is exactly the layer width $\equiv$ # of neurons in the layer*
  - ↳ *For $\ell = 0$ this is the number of inputs, i.e., $\mathcal{W}_0 = N$*
- *In layer ℓ, we have $\mathcal{W}_\ell + 1$ nodes*
  - ↳ *$\mathcal{W}_\ell$ neurons and one constant node that always returns 1*





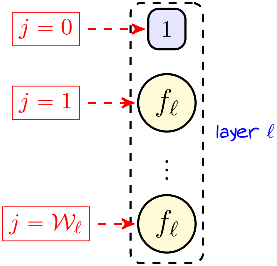
Fully-Connected FNNs: *Few Definitions*

We next index the nodes in each layer

In layer ℓ: *we have* $\mathcal{W}_\ell + 1$ *nodes*

↳ *One constant node* $\equiv$ *node* $j = 0$

↳ $\mathcal{W}_\ell$ *neurons/inputs* $\equiv$ *node* $j = 1, \dots, \mathcal{W}_\ell$



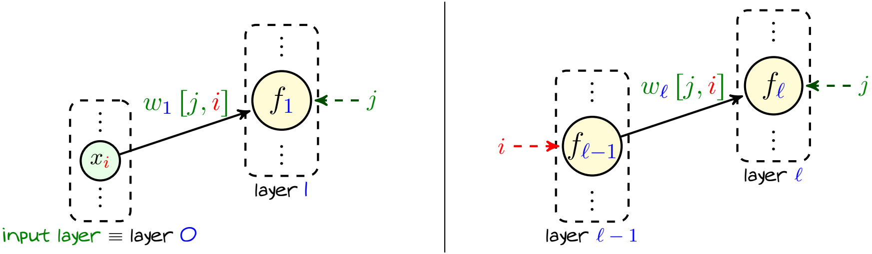
Fully-Connected FNNs: *Forward Pass*

We next give weights to the links

Weight of the link connecting

node i in layer $\ell - 1$ $\rightarrow$ *node j* in *layer ℓ*

is denoted by $w_\ell [j, i]$



Fully-Connected FNNs: *Forward Pass*

We next give weights to the links

Weight of the link connecting

node i in layer $\ell - 1$ $\to$ *node j* in *layer ℓ*

is denoted by $w_\ell[j, i]$

↳ The weights coming out of $i = 0$ are *biases*

$\boxed{w_\ell[j, 0] \text{ is the bias of neuron } j \text{ in layer } \ell}$

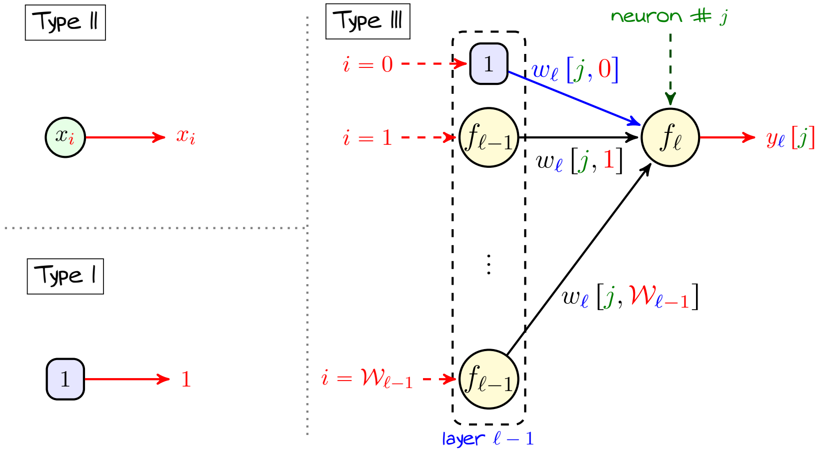
↳ There is no link from a node to a *constant node*: $w_\ell[j, i]$ *exists for*

- $i = 0, \ldots, \mathcal{W}_{\ell-1}$
- $j = 1, \ldots, \mathcal{W}_\ell$

This means that *there exists no such a weight* $w_\ell[0, i]$

Fully-Connected FNNs: *Forward Pass*

We finally specify the output of each node



Fully-Connected FNNs: *Forward Pass*

We finally specify the output of each node

We represent the output of node j in layer ℓ with $y_\ell[j]$

↳ *Since $j = 0$ is the constant node: $y_\ell[0] = 1$ for $\ell = 0, \ldots, L+1$*

↳ *Since $\ell = 0$ is the input layer: $y_0[j] = x_j$ for $j = 1, \ldots, N$*

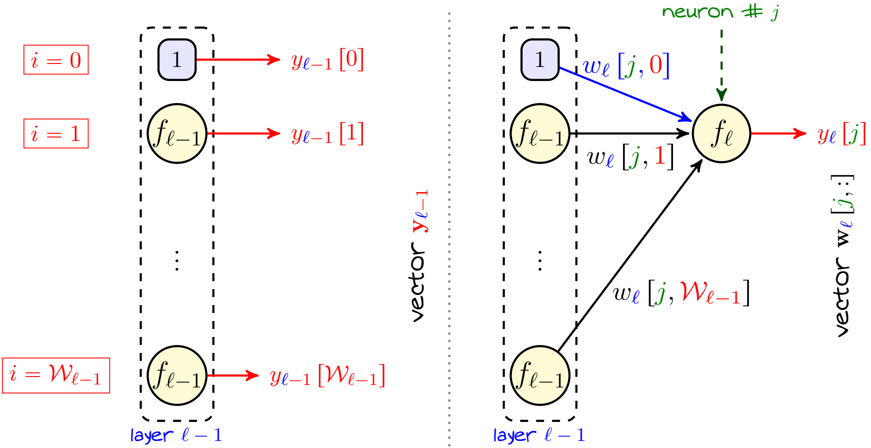
↳ *For neuron j in layer ℓ we can write*

$$y_\ell[j] = f_\ell(z_\ell[j])$$

where $z_\ell[j]$ is the output of the affine function in neuron j

$$\begin{aligned} z_\ell[j] &= w_\ell[j,0] 1 y_{\ell-1}[0] + \sum_{i=1}^{\mathcal{W}_{\ell-1}} w_\ell[j,i]\, y_{\ell-1}[i] \\ &= \sum_{i=0}^{\mathcal{W}_{\ell-1}} w_\ell[j,i]\, y_{\ell-1}[i] \end{aligned}$$

Fully-Connected FNNs: *Forward Pass*



Fully-Connected FNNs: *Forward Pass*

We can represent $z_j[\ell]$ *more compactly via vectorized notation*

$$
\begin{aligned}
z_\ell[j] &= \sum_{i=0}^{\mathcal{W}_{\ell-1}} w_\ell[j,i]\, y_{\ell-1}[i] \\
&= \underbrace{\begin{bmatrix} w_\ell[j,0] & w_\ell[j,1] & \dots & w_\ell[j,\mathcal{W}_{\ell-1}] \end{bmatrix}}_{\mathbf{w}_\ell[j,:]^\mathsf{T}} \underbrace{\begin{bmatrix} y_{\ell-1}[0] \\ y_{\ell-1}[1] \\ \vdots \\ y_{\ell-1}[\mathcal{W}_{\ell-1}] \end{bmatrix}}_{\mathbf{y}_{\ell-1}} \\
&= \mathbf{w}_\ell[j,:]^\mathsf{T}\, \mathbf{y}_{\ell-1}
\end{aligned}
$$

Fully-Connected FNNs: *Forward Pass*

We can further extend vectorized notation by defining

$$\mathbf{z}_\ell = \begin{bmatrix} z_\ell[1] \\ \vdots \\ z_\ell[\mathcal{W}_\ell] \end{bmatrix}$$

and thus writing $\mathbf{y}_\ell$ as

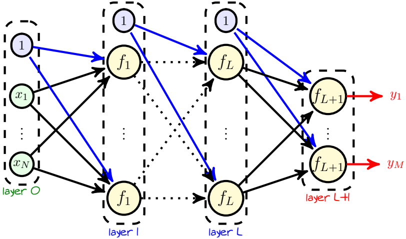
$$\mathbf{y}_\ell = f_\ell(\mathbf{z}_\ell)$$

where $f_\ell(\cdot)$ *is applied entry-wise*

and don't forget to add the dummy input 1, *i.e.,*

$$\mathbf{y}_\ell \leftarrow \begin{bmatrix} 1 \\ \mathbf{y}_\ell \end{bmatrix}$$

Fully-Connected FNNs: *Forward Pass*



Initiate the output of the first layer as $\mathbf{y}_0 = \boldsymbol{x}$
for $\ell = 0, \dots, L$ **do**
    for $j = 1, \dots, \mathcal{W}_\ell$ **do**
        Add $y_\ell[0] = 1$ and set $z_{\ell+1}[j] = \mathbf{w}_{\ell+1}[j,:]^\mathsf{T} \mathbf{y}_\ell$ `# affine function`
    end for
    Compute $\mathbf{y}_{\ell+1} = f_{\ell+1}(\mathbf{z}_{\ell+1})$ `# activation`
end for
for $\ell = 1, \dots, L+1$ **do**
    Return $\mathbf{y}_\ell$ and $\mathbf{z}_\ell$
end for

Fully-Connected FNNs: *Forward Pass*

We can present everything even more compactly: *let's write down* $\mathbf{z}_\ell$

$$\mathbf{z}_\ell = \begin{bmatrix} z_\ell[1] \\ \vdots \\ z_\ell[\mathcal{W}_\ell] \end{bmatrix} = \begin{bmatrix} \mathbf{w}_\ell[1,:]^\mathsf{T} \mathbf{y}_{\ell-1} \\ \vdots \\ \mathbf{w}_\ell[\mathcal{W}_\ell,:]^\mathsf{T} \mathbf{y}_{\ell-1} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_\ell[1,:]^\mathsf{T} \\ \vdots \\ \mathbf{w}_\ell[\mathcal{W}_\ell,:]^\mathsf{T} \end{bmatrix} \mathbf{y}_{\ell-1}$$

Now, we can define the matrix $\mathbf{W}_\ell$ *as*

$$\mathbf{W}_\ell = \begin{bmatrix} \mathbf{w}_\ell[1,:]^\mathsf{T} \\ \vdots \\ \mathbf{w}_\ell[\mathcal{W}_\ell,:]^\mathsf{T} \end{bmatrix} = \begin{bmatrix} w_\ell[1,0] & \dots & w_\ell[1,\mathcal{W}_{\ell-1}] \\ \vdots & & \vdots \\ w_\ell[\mathcal{W}_\ell,0] & \dots & w_\ell[\mathcal{W}_\ell,\mathcal{W}_{\ell-1}] \end{bmatrix}$$

This matrix collects all learning parameters of layer ℓ

$$\boxed{\textit{Note that } \mathbf{W}_\ell \textit{ has } \mathcal{W}_\ell \textit{ rows and } \mathcal{W}_{\ell-1}+1 \textit{ columns}}$$

Forward Propagation: *Pseudo Code*

So, we can compactly present the forward propagation algorithm as follow

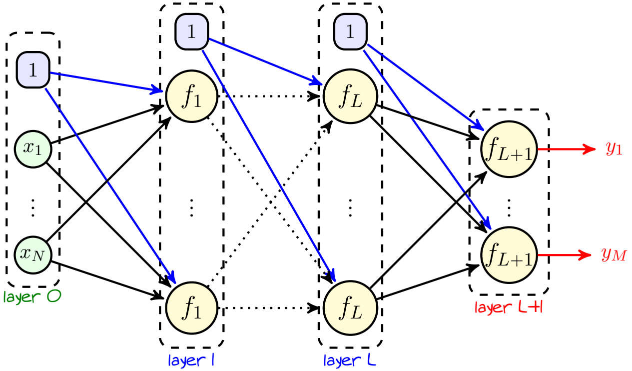
```
ForwardProp():
```

1: Initiate with $\mathbf{y}_0 = \boldsymbol{x}$
2: **for** $\ell = 0, \dots, L$ **do**
3: Add $y_\ell[0] = 1$ and determine $\mathbf{z}_{\ell+1} = \mathbf{W}_{\ell+1}\mathbf{y}_\ell$ `# forward affine`
4: Determine $\mathbf{y}_{\ell+1} = f_{\ell+1}(\mathbf{z}_{\ell+1})$ `# forward activation`
5: **end for**
6: **for** $\ell = 1, \dots, L+1$ **do**
7: Return $\mathbf{y}_\ell$ and $\mathbf{z}_\ell$
8: **end for**

After getting data-point x, we pass it through a linear layer whose weights are learnable and a nonlinear transform that is specified by activation. The output of this layer passes forward to the next layer till we get to the output.

Forward Propagation: *Compact Diagram*

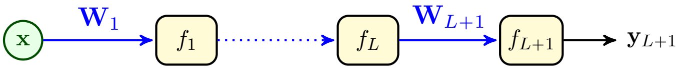
Inspired by forward propagation, we can represent the FNN



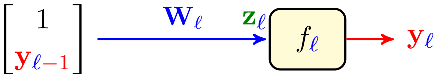
by the following compact diagram

$$\mathbf{x} \xrightarrow{\mathbf{W}_1} \boxed{f_1} \cdots\cdots\rightarrow \boxed{f_L} \xrightarrow{\mathbf{W}_{L+1}} \boxed{f_{L+1}} \longrightarrow \mathbf{y}_{L+1}$$

Forward Propagation: *Compact Diagram*



Here, we compactly represent layer ℓ as



- *The link $\mathbf{W}_\ell$ represents the affine function of layer ℓ*
- *The block f_ℓ represents the activation of layer ℓ*
  - ↳ *The input of this block can be considered $\mathbf{z}_\ell$*
  - ↳ *The output of this block can be considered $\mathbf{y}_\ell$*
- *We always add $y_\ell[0] = 1$ after computing $\mathbf{y}_\ell$*

This compact diagram will come in handy when we derive backpropagation!