Neural Network

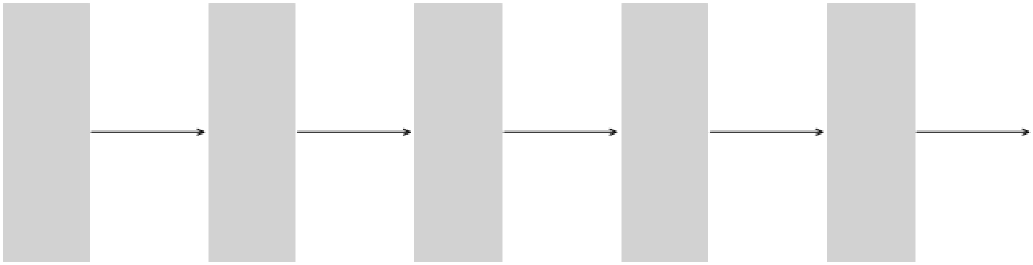
A Neural Network (Sequential architecture) is composed of

- sequence of Layers
    - layer $\ll$ transforms its input $\backslash\mathbf{y}_{(\ll-1)}$ to output $\backslash\mathbf{y}_{\backslash\mathrm{llp}}$
        - through a transformation: operation parameterized by weights $\backslash\mathbf{W}_{\backslash\mathrm{llp}}$

In [6]:

```
fig_tf_seq
```

Out[6]:



- initial layer 0 is Input layer:
    - outputs the network's inputs $\backslash\mathbf{x}$

$$\backslash\mathbf{y}_{(0)} = \backslash\mathbf{x}$$

- final layer L transforms its input $\backslash\mathbf{y}_{(L-1)}$ to prediction $\backslash\hat{\mathbf{y}}$

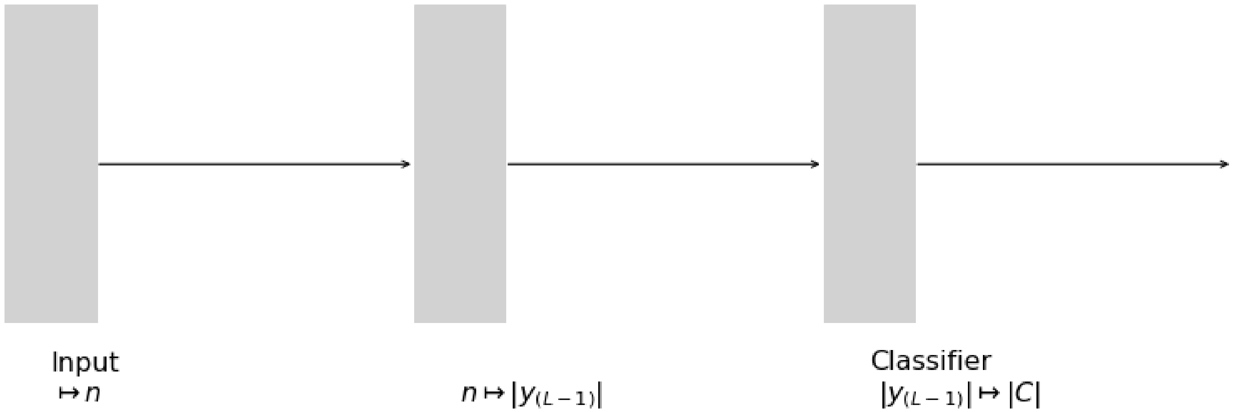
$$\backslash\hat{\mathbf{y}} = \backslash\mathbf{y}_{\backslash\mathrm{llp}}$$

-
    - transformation of layer L usually: Regression or Classification

In [7]:

```
fig_tf_test
```

Out[7]:



In the above diagram

- the central box represents a sequence of 1 or more layers
- a sub-network
- just for brevity

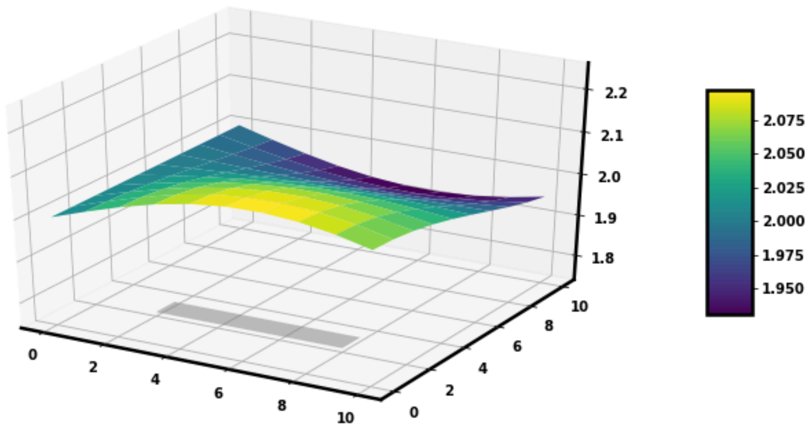
The Neural network thus computes a *function* from $\backslash\mathbf{x}$ to $\hat{\backslash\mathbf{y}}$.

The function *mimics* the training data

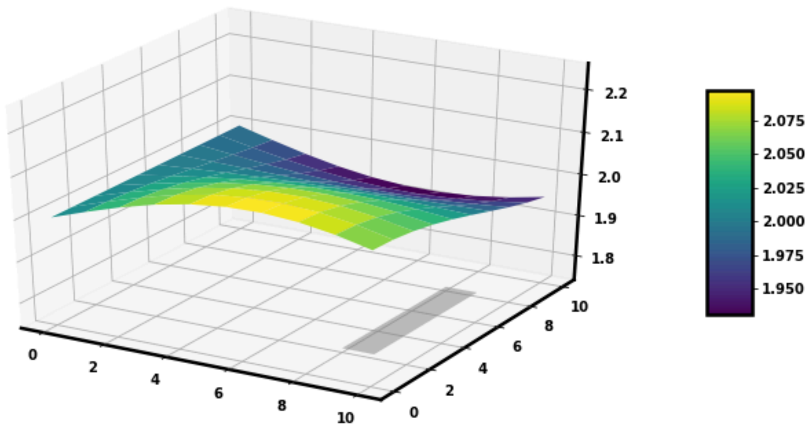
$$\langle \backslash\mathbf{X}, \backslash\mathbf{y} \rangle = [\backslash\mathbf{x}^{\backslash\mathrm{ip}}, \backslash\mathbf{y}^{\backslash\mathrm{ip}} | 1 \leq i \leq m]$$

where each *example* $\langle \backslash\mathbf{x}^{\backslash\mathrm{ip}}, \backslash\mathbf{y}^{\backslash\mathrm{ip}} \rangle$

- describes the mapping of the function on input $\backslash\mathbf{x}^{\backslash\mathrm{ip}}$ to output $\backslash\mathbf{y}^{\backslash\mathrm{ip}}$
- e.g., from input features $\backslash\mathbf{x}^{\backslash\mathrm{ip}}$ to
    - continuous value $\backslash\mathbf{y}^{\backslash\mathrm{ip}}$
    - discrete class $\backslash\mathbf{y}^{\backslash\mathrm{ip}}$
        - really: output is a probability vector over finite set C of discrete classes



2.2
2.1
2.0
1.9
1.8
0
2
4
6
8
10
0
2
4
6
8
10
2.075
2.050
2.025
2.000
1.975
1.950



2.2
2.1
2.0
1.9
1.8
0
2
4
6
8
10
0
2
4
6
8
10
2.075
2.050
2.025
2.000
1.975
1.950

The Neural Network is trained ("learns") to mimic the training data

- by solving for the weights $\backslash\mathbf{W}_{\backslash\mathbf{llp}}$ of each layer $1 \leq \ll \leq L$
- that minimize a loss function

$$\backslash\mathbf{loss} = \sum_{i-1}^{m} \backslash\mathbf{loss}^{\backslash\mathbf{ip}}$$

- where $\backslash\mathbf{loss}^{\backslash\mathbf{ip}}$ if a function of
  - how much prediction $\hat{\backslash\mathbf{y}}^{\backslash\mathbf{ip}}$ deviates from true target/label $\backslash\mathbf{y}^{\backslash\mathbf{ip}}$

The minimization procedure is usually a variant of *Gradient Descent*

The challenge is that the operation of each layer $1 \leq \ll < L$

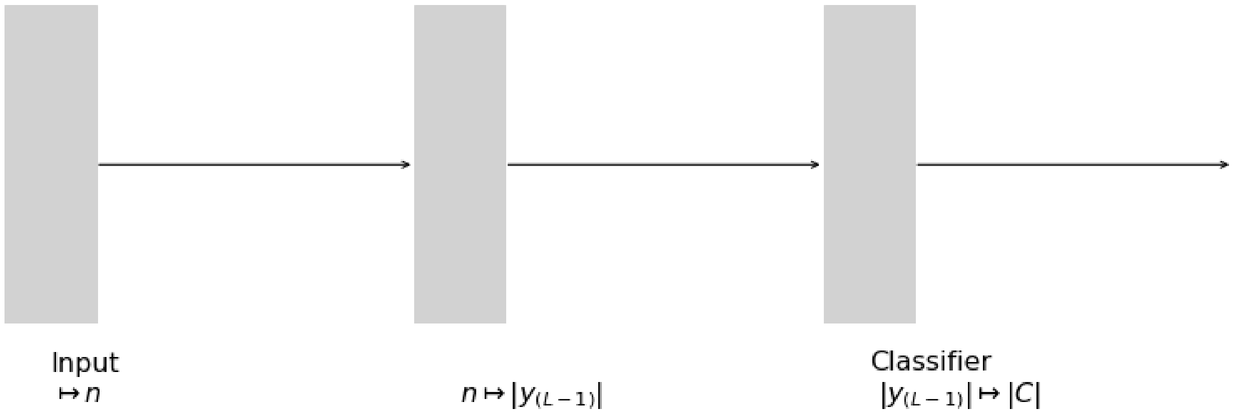
- is usually not *interpretable*
- we can describe *how* it transforms $\backslash\mathbf{y}_{(\ll-1)}$ to $\backslash\mathbf{y}_{\backslash\mathrm{llp}}$
- but not *why* it is performing the transformation
    - objective (describe) rather than subjection (why)

So the sub-network in the diagram

In [8]:

```
fig_tf_test
```

Out[8]:



computes an *unknown* function

- from input of length n (the raw input)
- to a vector of length $|\backslash\mathbf{y}_{(L-1)}|$

whose purpose is

- to make the final layer L (e.g., the Classifier)
- create predictions (outputs)
- that have a very-low loss $\backslash\mathrm{loss}$
    - a good "approximation" of the function described by the training data

In [9]:

```
print("Done")
```

```
Done
```