Neural Networks & Pruning

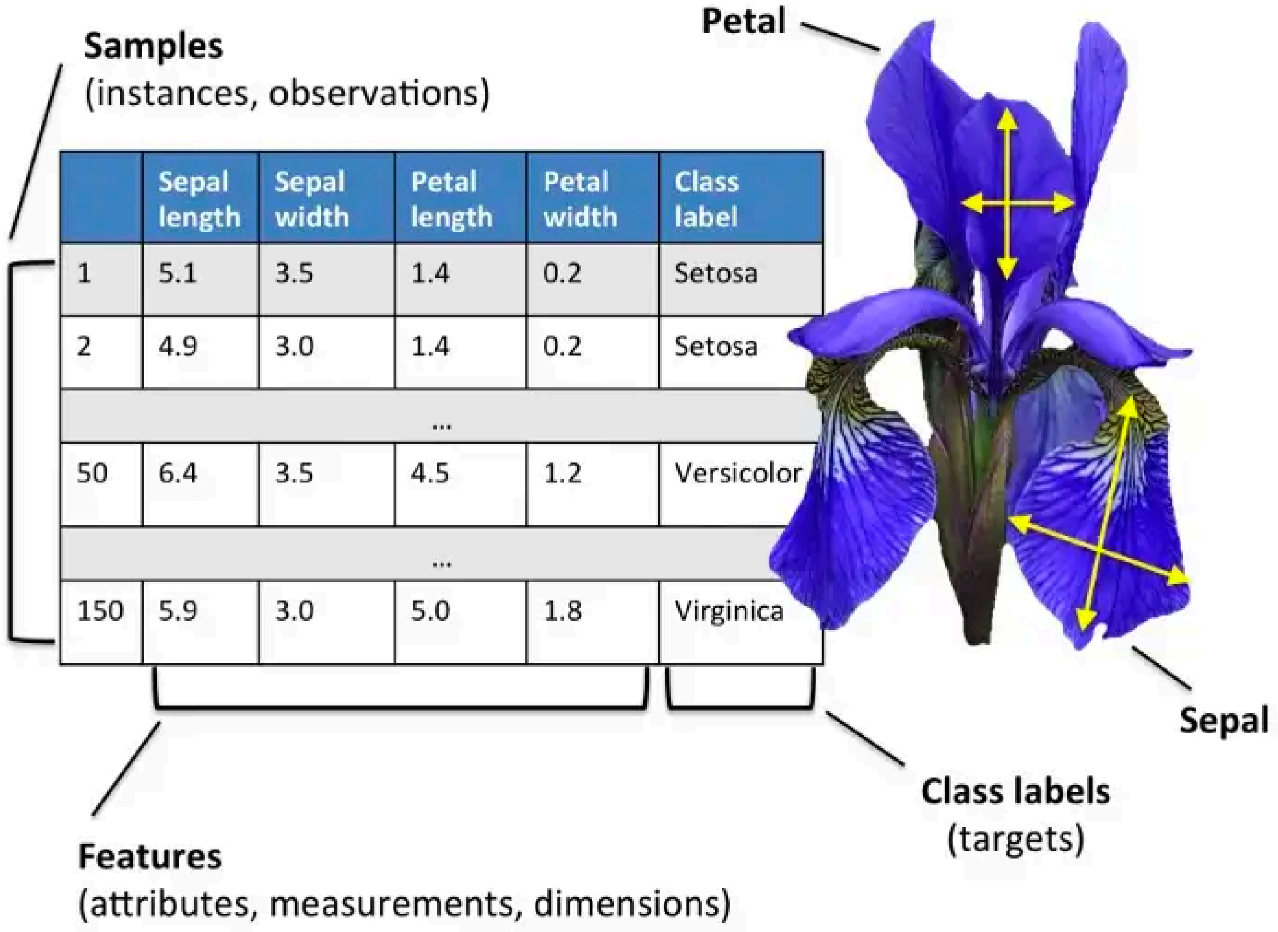
An Implementation from First Principles

Foundations of Machine Learning - Project

Federico Segala

ID: 906213

IRIS Dataset



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

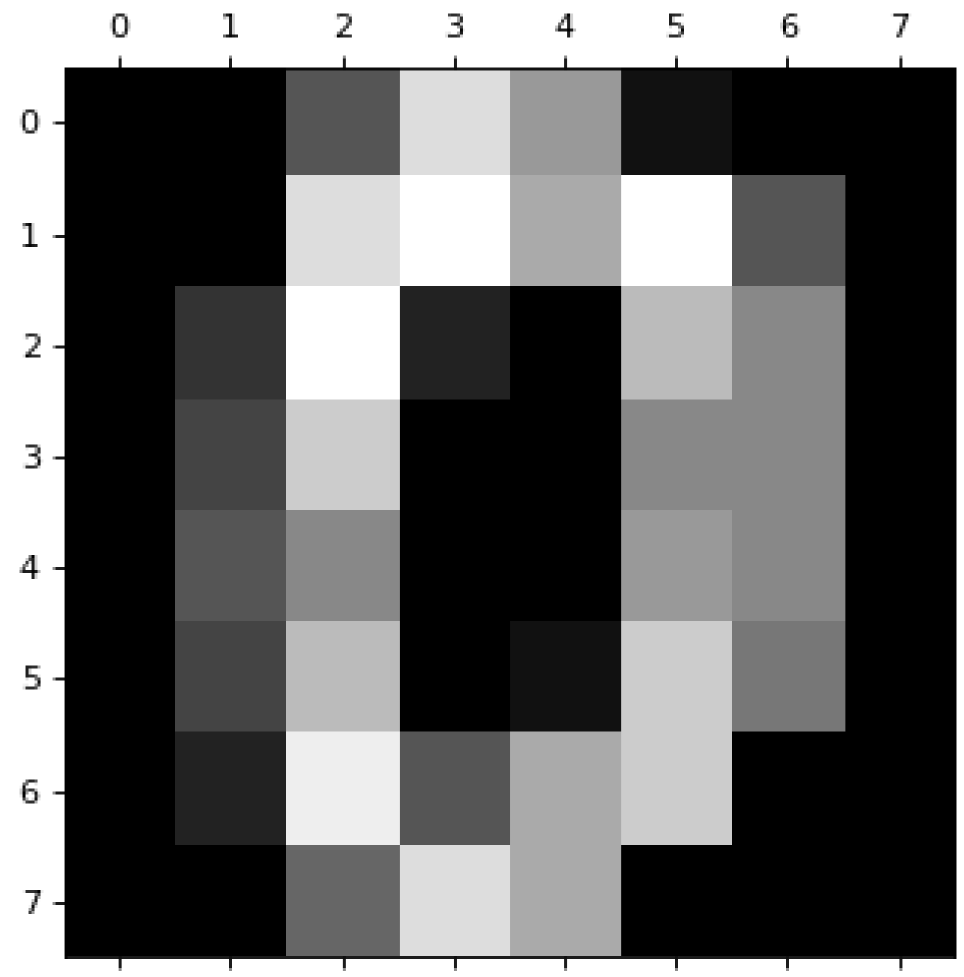
Number of Samples: 150

Number of Classes: 3

Number of Features: 4

Train - Test Split: 90% - 10%

Digits Dataset Presentation



8 x 8 pixels grayscale images

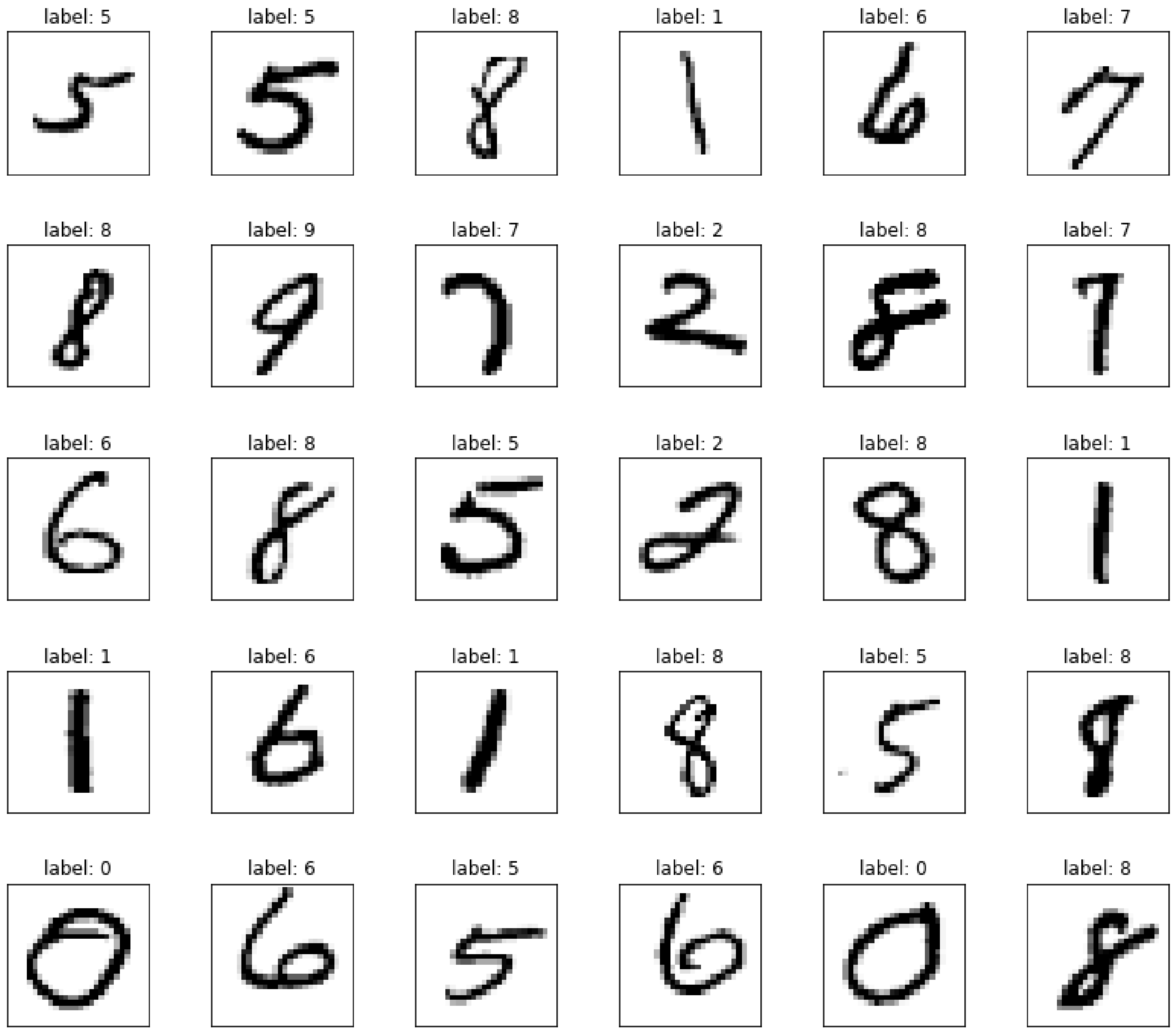
Number of features 64

Number of Samples: 1797

Number of Classes: 10

Train - Test Split: 90% - 10%

MNIST Dataset Presentation



28 x 28 pixels grayscale images

→ Number of features 784

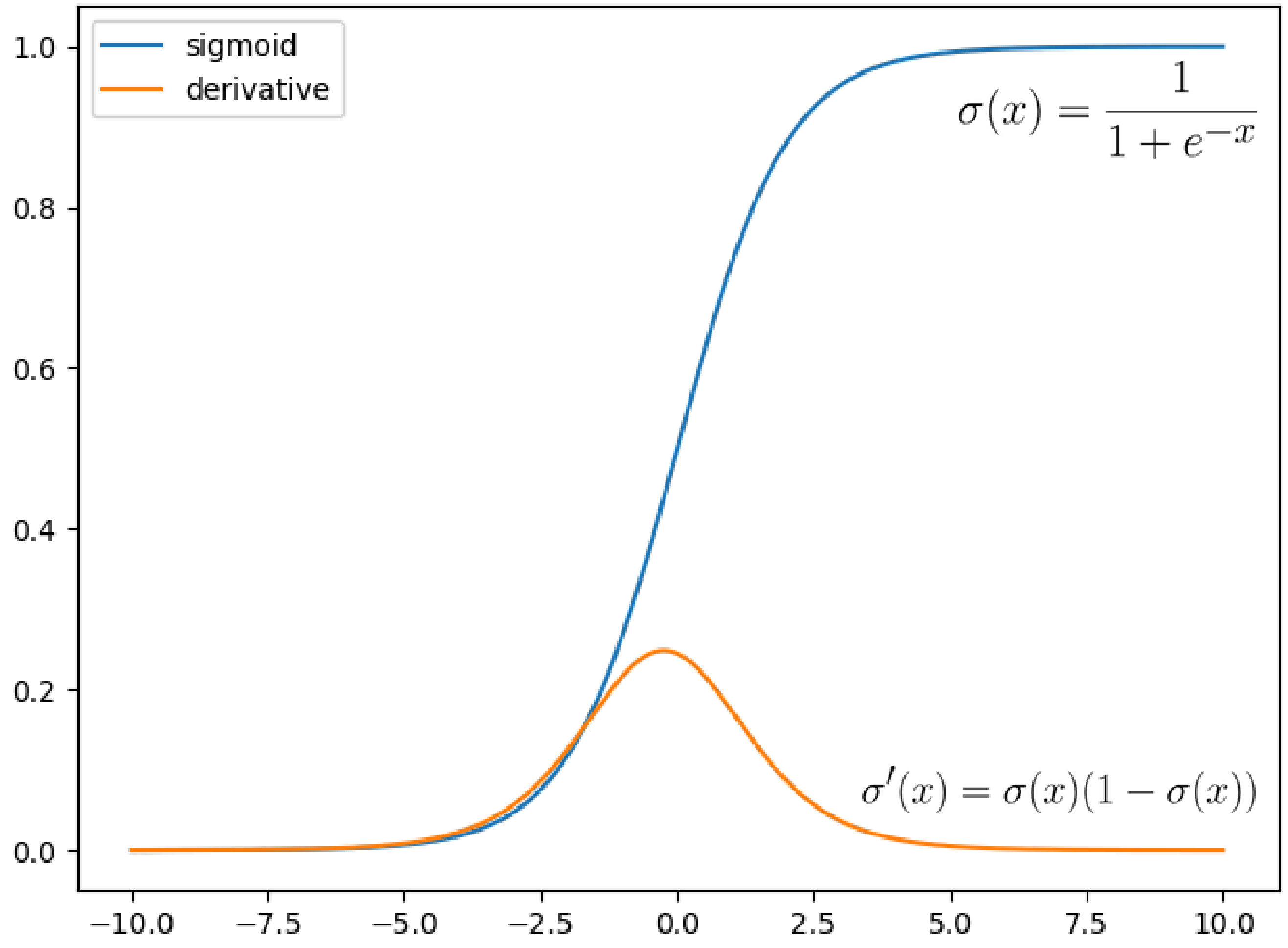
Number of Samples: 60 000

Number of Classes: 10

Train - Test Split: 30% - 70%

Implementation Choices

Activation Function



Online Weight Updates:

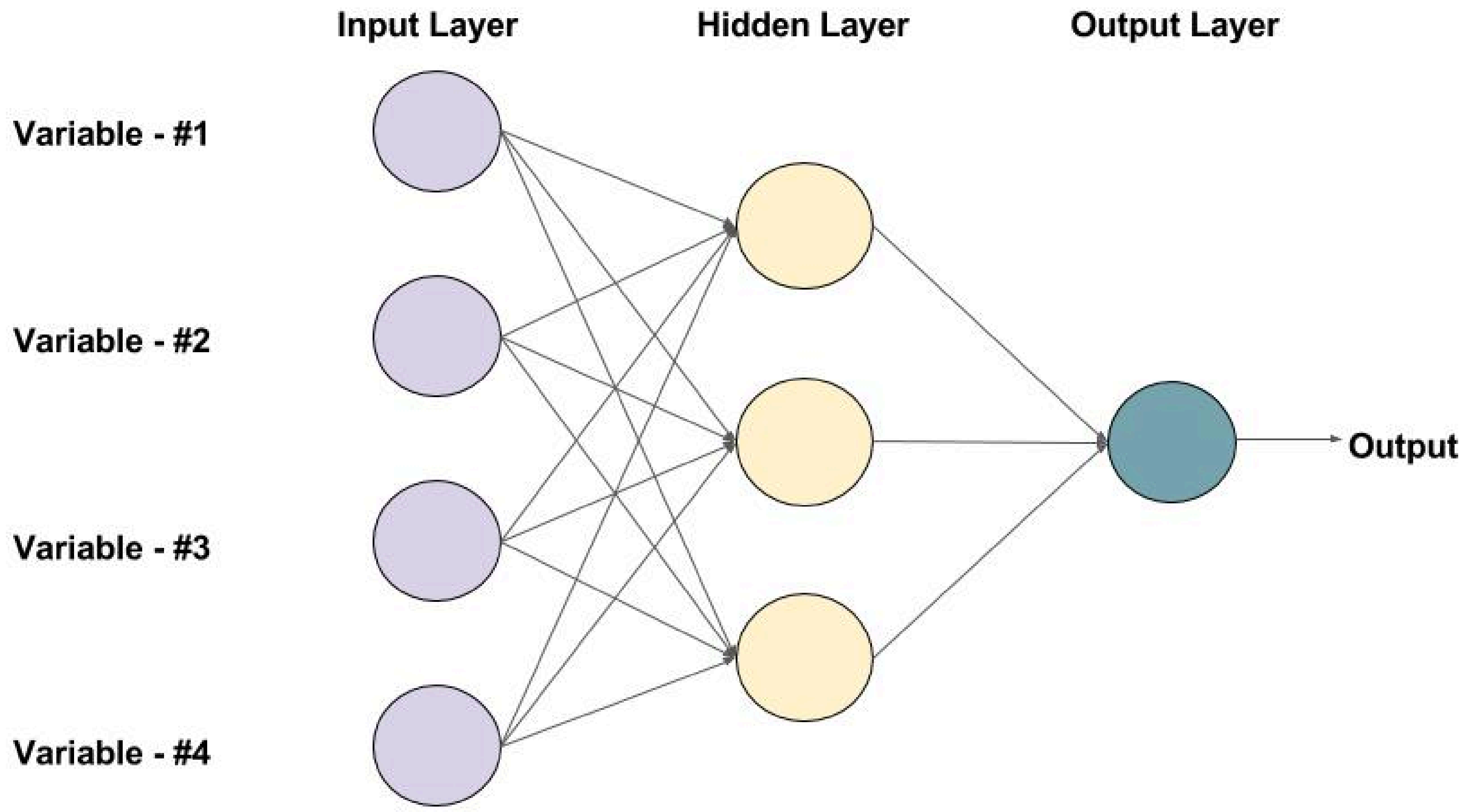
$$\Delta^{\mu}_{w_{pq}} = \eta \delta^{\mu}_{p} V^{\mu}_{q}$$

Loss Function

$$MSE = \frac{1}{2} \sum_{\mu} \sum_{i} (y^{\mu}_{i} - O^{\mu}_{i}(w))^2$$

Training Algorithm

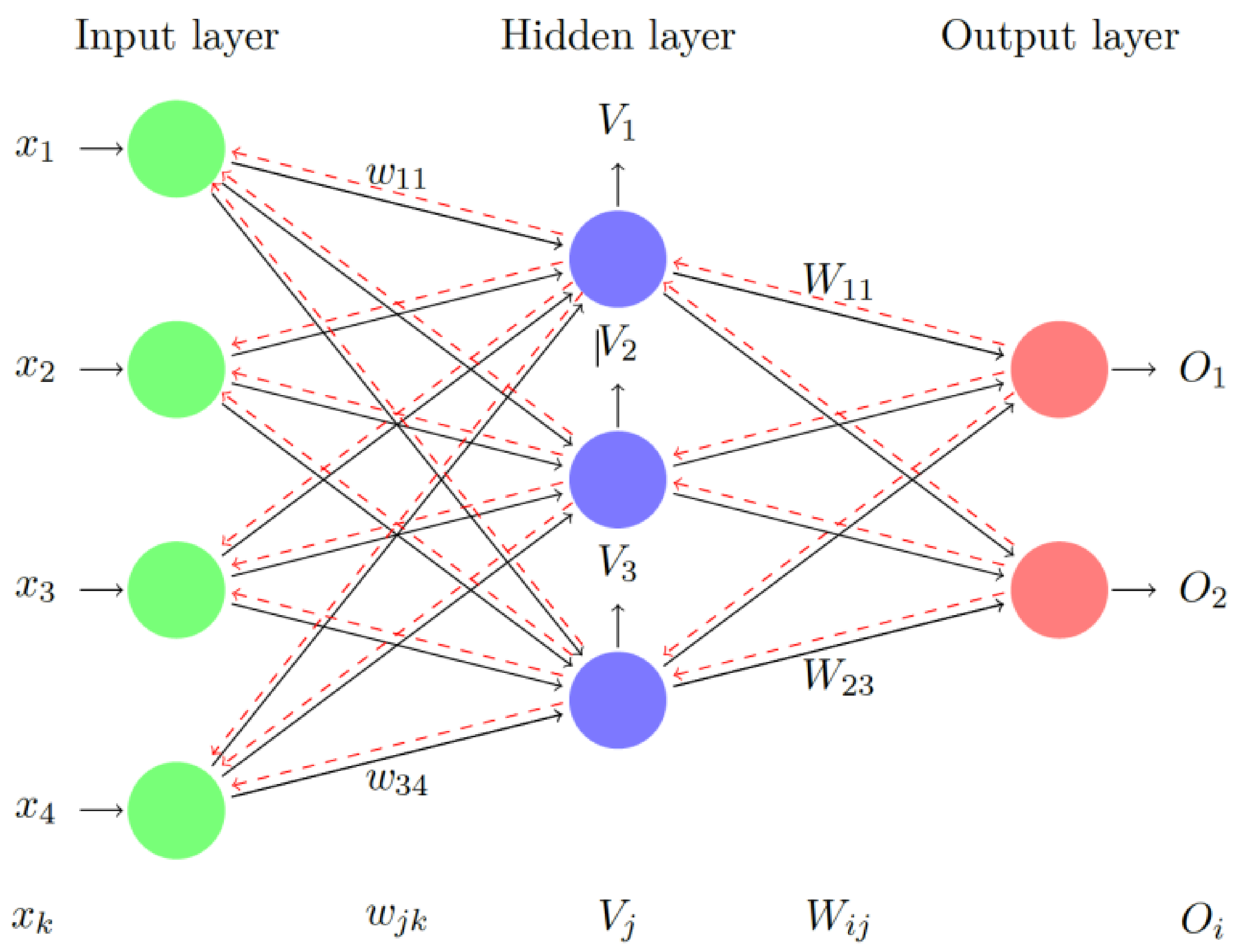
```
def train_procedure(model, epochs, training_set):
  for epoch in range(epochs):
    for x, y in range(training_set):
      forward(model, x)
      model.compute_deltas(model, y)
      model.update_weights(model)
```



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

Forward Pass

```
def forward(model, x):
  for layer in model_layer:
    extend_x_with_bias(x, -1)

    # compute non activated input
    h = dot_prod(x, layer)
    # applicate non linear activation
    V = layer.activate(h)

    model.h.append(h)
    model.V.append(V)

   return model.V.last_value()
```



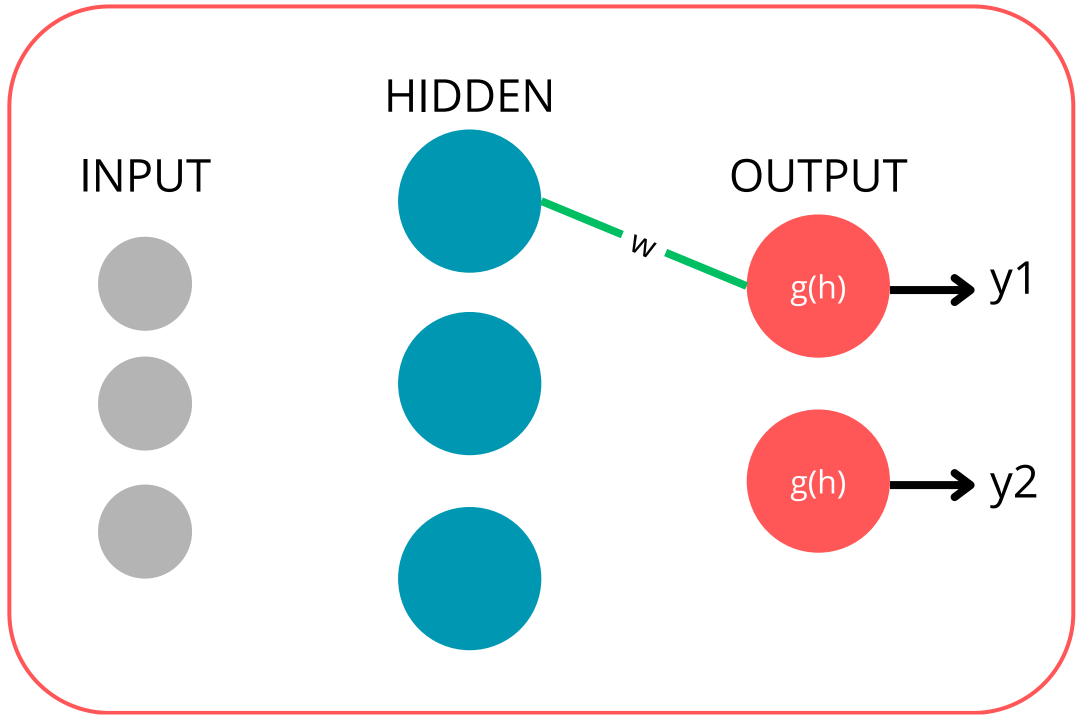
Weights Update

```
def update_weights(model):
  for i, layer in model.layers:
    if i == 0:  # retrieve the preceding activated output
      V = model.input   # in the first layer we use input
    else:
      V = model.V[i-1]

    layer += model.deltas[i] * V # update
```

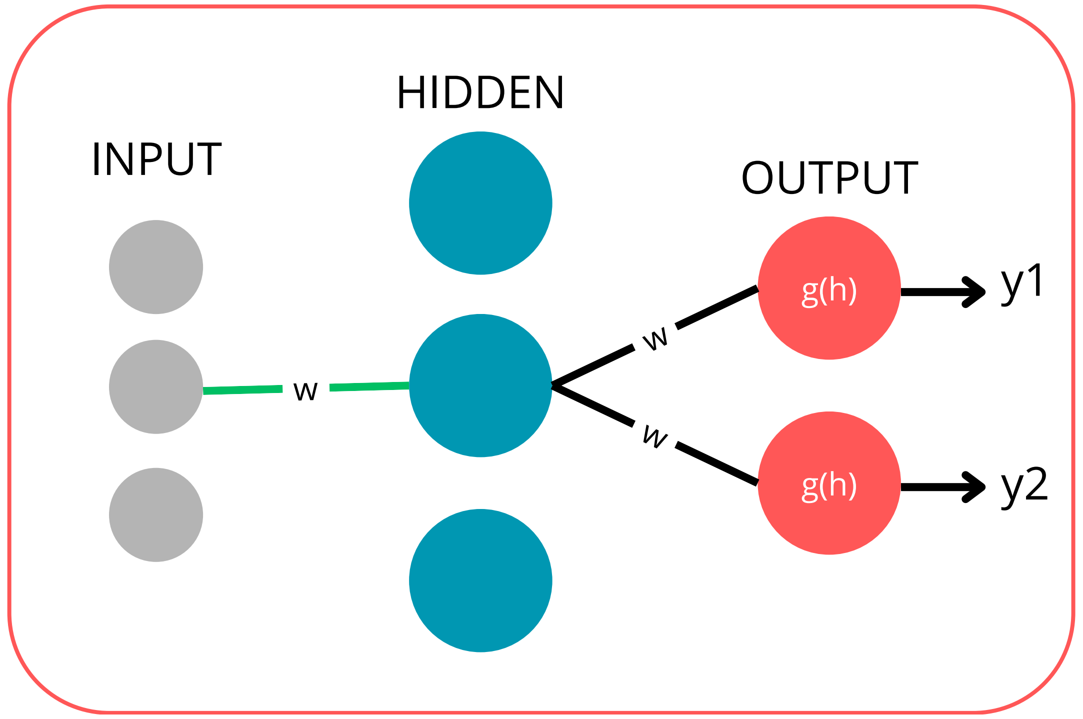
Delta Computation

```
def compute_deltas(model, y):
  # traverse the network in reverse
  for i, layer in reversed(model.layers):
    if hidden_to_output(layer):
      d_ij = layer.activate_derivative(model.h[i])
      model.delta.append((y-model.V[i])) * d_ij
    else:
      d_jk = layer.activate_derivative(model.h[i])

      previous_w = model.layers[i+1]
      previous_delta = model.delta[i+1]

      previous_weighted_error = dot_product(previous_w, previous_delta)

      d_jk *= previous_weighted_error
      model.delta.append(d_jk)
```

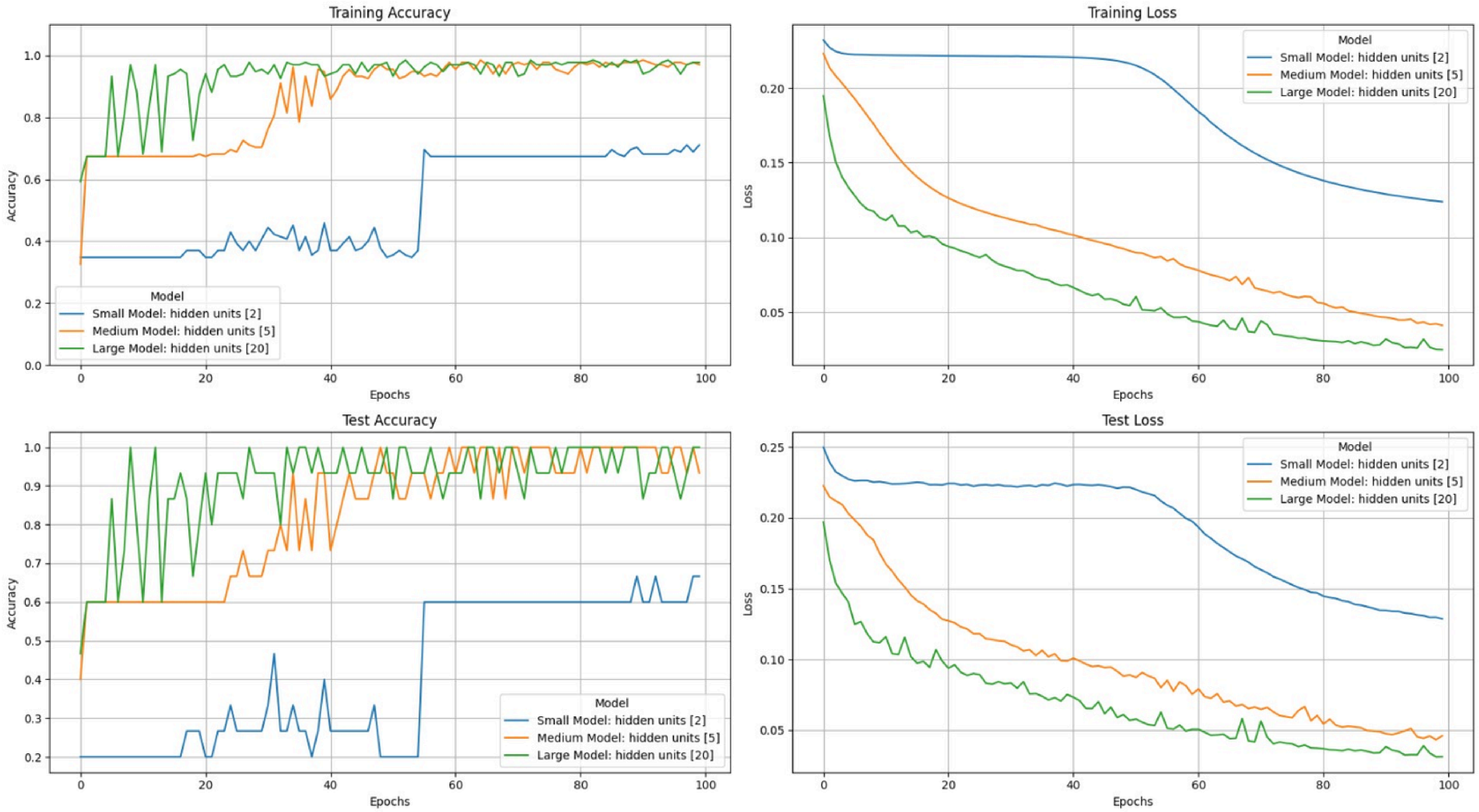
$$\delta_i^\mu = (\mathrm{target}_i^\mu - y_i^\mu) g'(h_i^\mu)$$



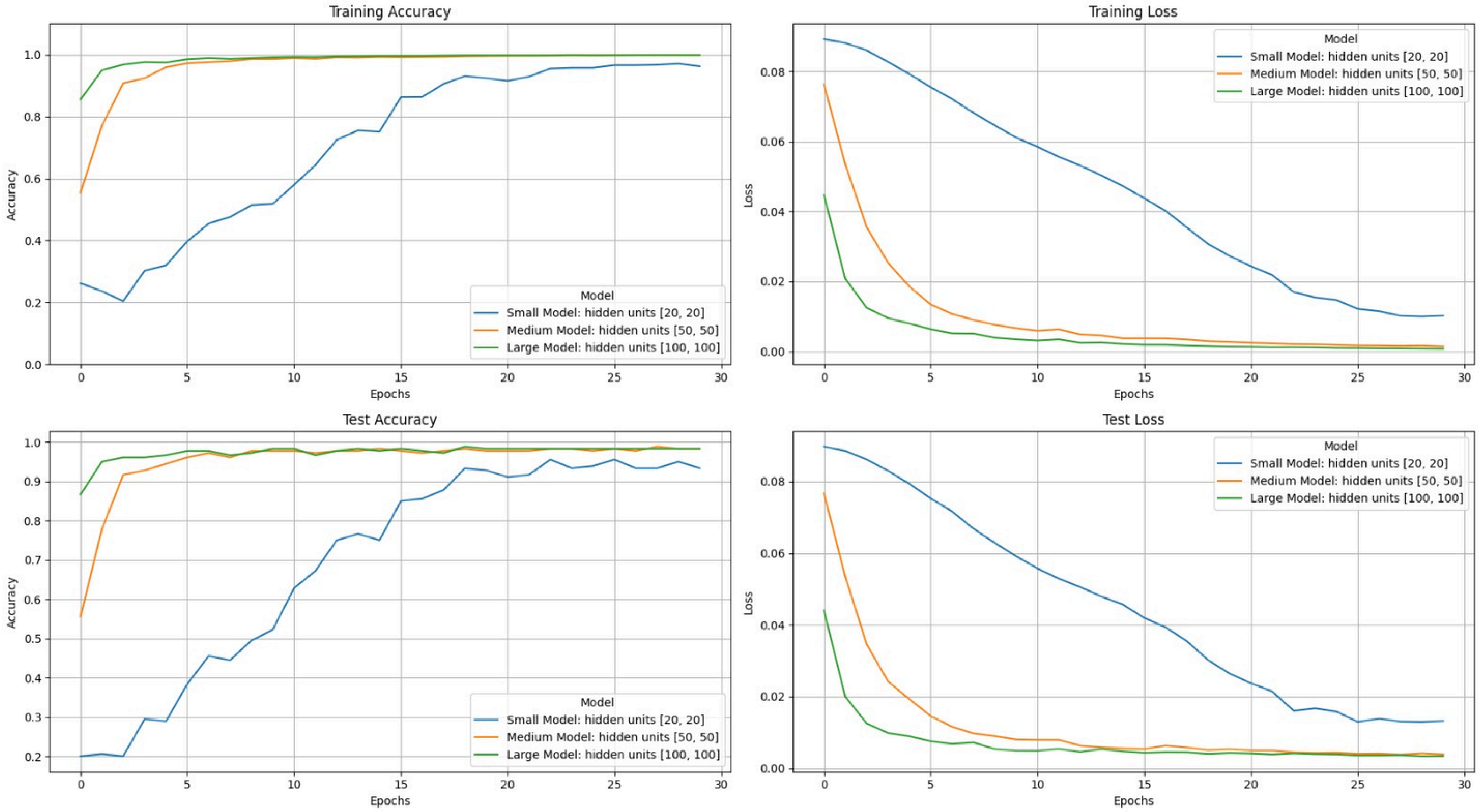
$$\hat{\delta}_j^\mu = g'(h_j^\mu) \sum_i \delta_i^\mu W_i j$$



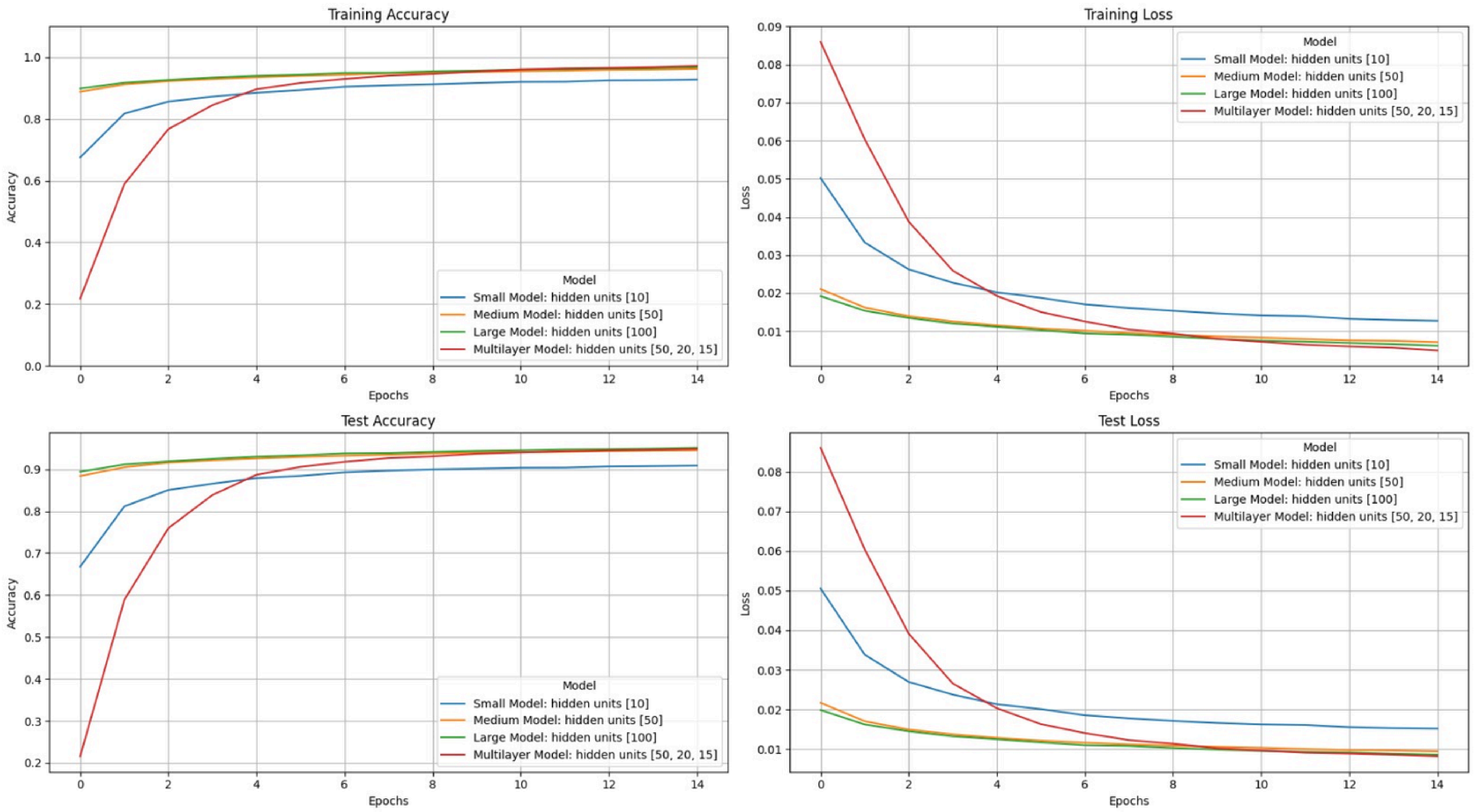
Performance on IRIS Dataset



Performance on DIGITS Dataset



Performance on MNIST Dataset



Iterative Pruning Algorithm

Hidden unit removal selection

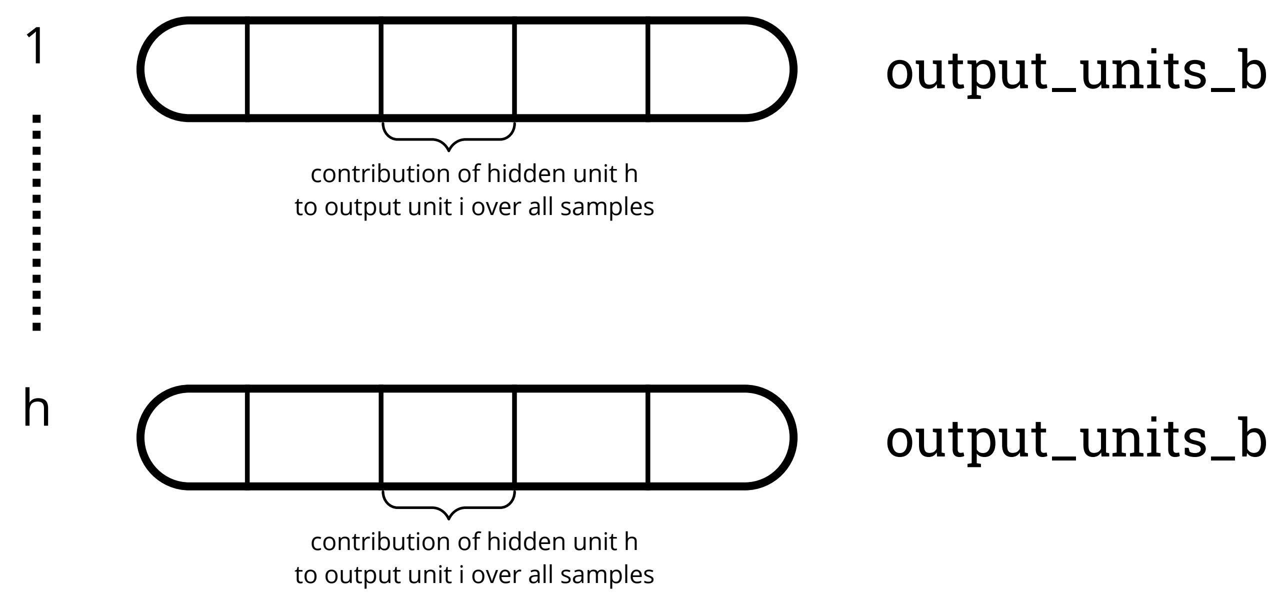
Update of the remaining hidden unit connections to to output layer

Pruning Algorithm - Removal Selection

```
def choose_unit_to_prune(model):
  # shape: (n_train_samples * n_hidden_units)
  hidden_layer_output = model.batch_hidden_layer_output()
  hidden_to_output_weights = model.weights[-1]

  norms = [0 for _ len(model.n_hidden_units)]
  for h in range(model.n_hidden_units):
    # final length = n_output_units * n_input_samples
    output_units_b = []

    for output_idx in range(model.n_outputs):
      # one b_i for each train sample
      b_i = hidden_to_output_weights[h,i] * hidden_layer_out[:,h]
      output_units_b.append(b_i)

    norms.append(compute_norm(output_units_b))

  return argmin(norms)
```

We compute the **contribution** of each **hidden unit** to each **output unit** for every input sample

The contribution is given by the connection weight and the h-th hidden unit's output



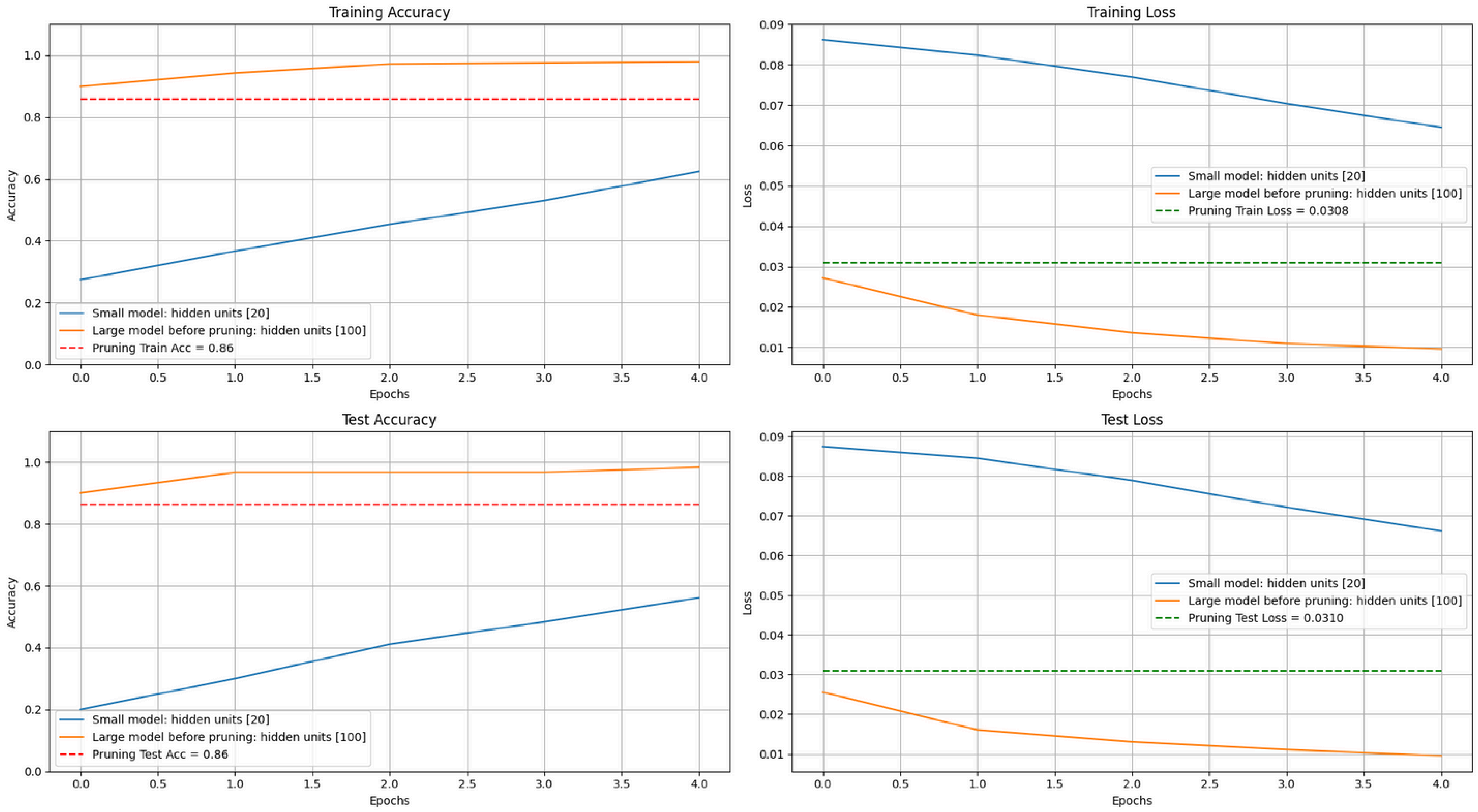
Pruning Algorithm - Update Existing Connections

```
def update_connections(model, h_star):
  hidden_output_without_removed = model.hidden_output_removal(h_star)
  deltas = zeros_with_shape(model.n_output_units, model.n_hidden_units-1)

  for i in range(model.output_unit_count):
    # contribution of the removed hidden to the output unit i
    b_i = model.w[-1][i, h_star] * model.hidden_output[:, h_star]

    delta_i = solve_with_least_squares(hidden_output_without_removed, b_i)

    deltas[i] = delta_i


  model.update_hidden_to_output_weights(deltas)
  model.remove_hidden_layer_input(h_star)
```

We solve an **over-determined** system of equations for each output unit touched by the weight update

$$\underbrace{\begin{bmatrix} y_{11} & y_{12} & \cdots & a_{1,n_{hidden}} \\ y_{21} & y_{22} & \cdots & y_{2,n_{hidden}} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n_{samples},1} & a_{n_{samples},2} & \cdots & a_{n_{samples},n_{hidden}} \end{bmatrix}}_{Y \in \mathbb{R}^{n_{samples} \times n_{hidden}}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_{hidden}} \end{bmatrix}}_{x \in \mathbb{R}^{n_{hidden}}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n_{samples}} \end{bmatrix}}_{b \in \mathbb{R}^{n_{samples}}}$$

Performance on DIGITS Dataset



Neural Networks & Pruning

An Implementation from First Principles

Foundations of Machine Learning - Project

Federico Segala

ID: 906213