

The background is a light blue gradient with an abstract network of nodes and connections. Nodes are represented by circles in various shades of blue, teal, and light green. They are connected by thin, light blue lines, creating a web-like structure that suggests neural network connectivity. The nodes vary in size, with some being significantly larger than others.

# Deep Learning Explained

Module 3: Multi-layer Perceptron

Sayan D. Pathak, Ph.D., Principal ML Scientist, Microsoft

Roland Fernandez, Senior Researcher, Microsoft

# Module outline

Application:

OCR using MNIST data

Model:

Recap Logistic Regression

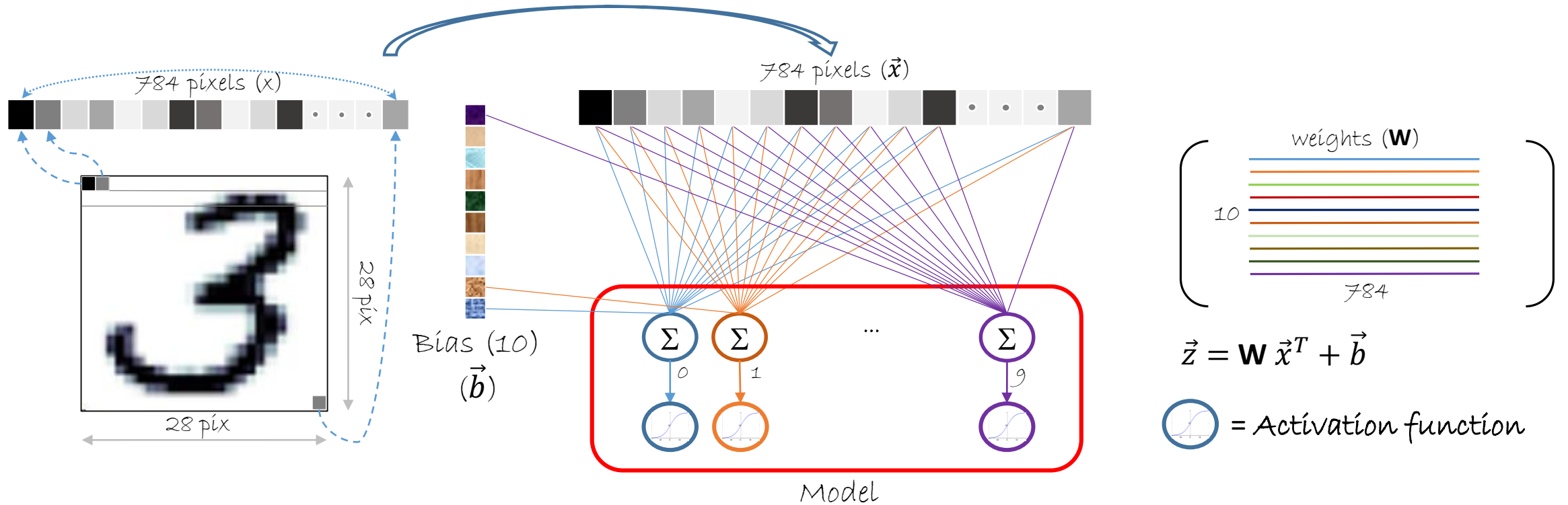
Multi-Layer Perceptron

Concepts:

Activation functions

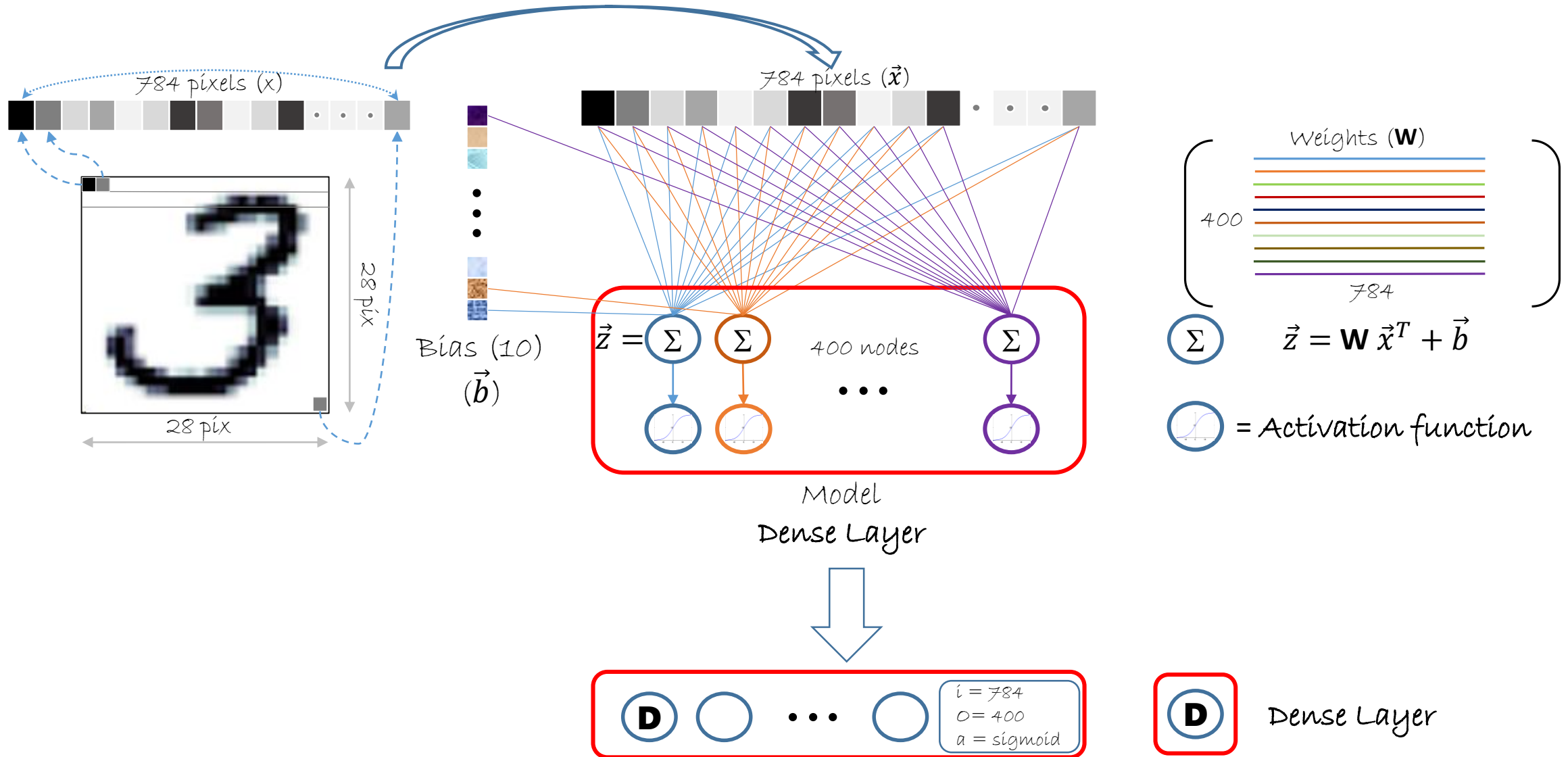
Train-Test-Predict Workflow

# Logistic regression



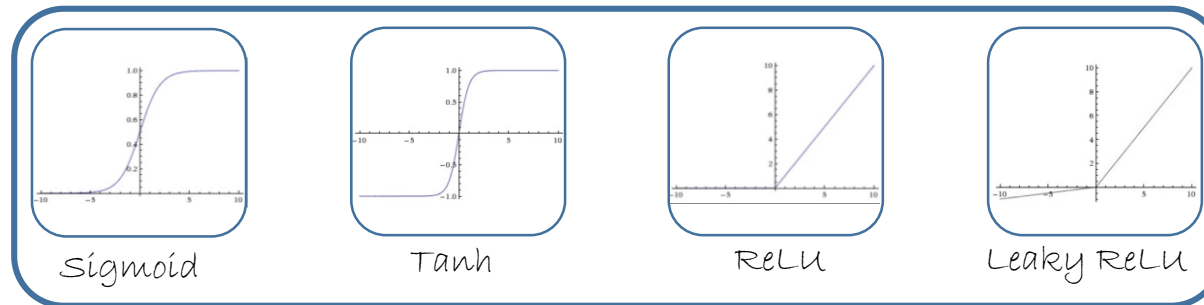
Error rate in detection of MNIST digits with Logistic Regression = 7-8%

# Towards deep networks

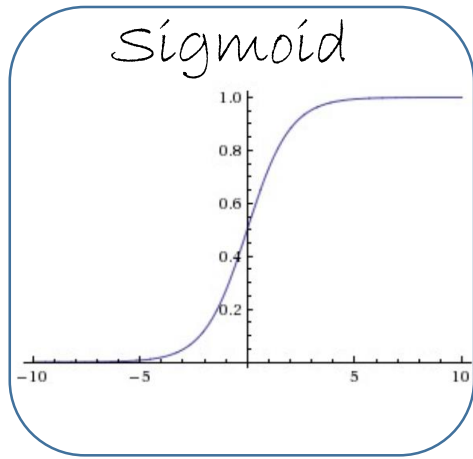


# Activation functions

- Activation functions (a.k.a. non-linearity):
  - ✓ Take a *single number* and maps it to a *different numerical value*
  - ✓ E.g. *Sigmoid* maps values any numerical value to a  $(0,1)$  range
- Popular functions:
  - *Sigmoid*
  - *Tanh*
  - *ReLU (Rectified Linear Unit)*
  - *Leaky / Parametric Relu*

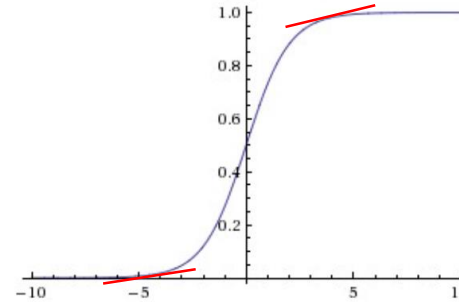


# Activation functions



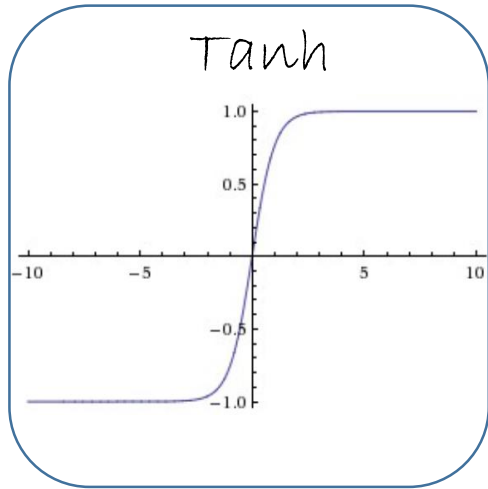
$$\sigma(x) = 1/(1 + e^{-x})$$

- Maps real value number into range 0 – 1
- Historically popular, bears semblance to neurons firing pattern
- Recently its popularity is dipping
  - ✓ Saturation / Vanishing Gradient
    - At either end of the tails, gradients go to zero
    - Caution needed when initializing weights to prevent saturation



- ✓ Non-zero centered output
  - During optimization, causes zig-zagging dynamics
  - More of inconvenience, less severe than saturation

# Activation functions

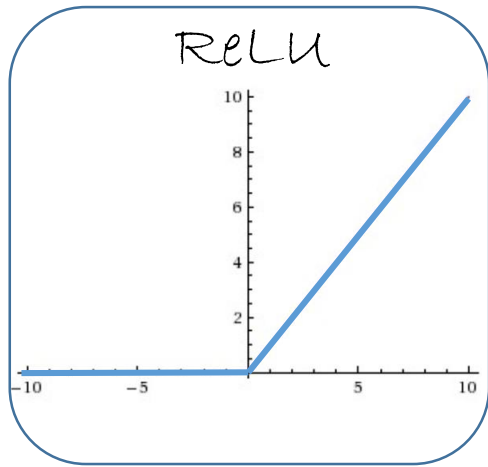


$$\tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})$$

- Maps real value number into range -1 to 1
- Like *Sigmoid*, its activation also saturates but output is *zero* centered
- *Tanh* is usually preferred over *Sigmoid*
- *Tanh* is a scaled *Sigmoid* function

$$\tanh(x) = 2\sigma(2x) - 1$$

# Activation functions

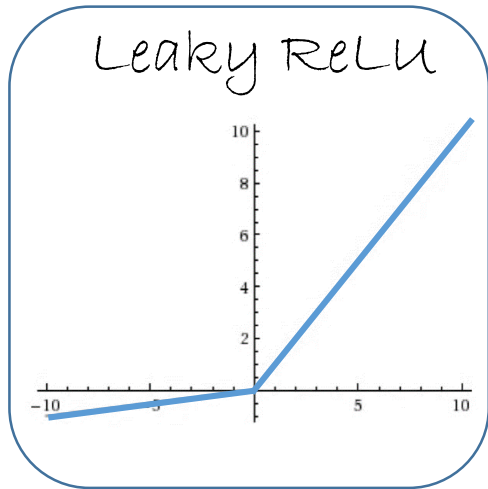


$$f(x) = \max(0, x)$$

- ReLU = Rectified Linear Unit
- Very popular and simply thresholds values below 0
- Pros:
  - ✓ Fast convergence
    - SGD converges much faster compared to sigmoid / tanh
    - Arguably due to its linear (non saturating form)
  - ✓ Simple implementation
    - Involves thresholding of activation matrix at zero
- Cons:
  - ✓ Fragile
    - Irreversibly die when large gradient flows
    - As much as 40% of network can die if learning rate set is too high



# Activation functions (Advanced)



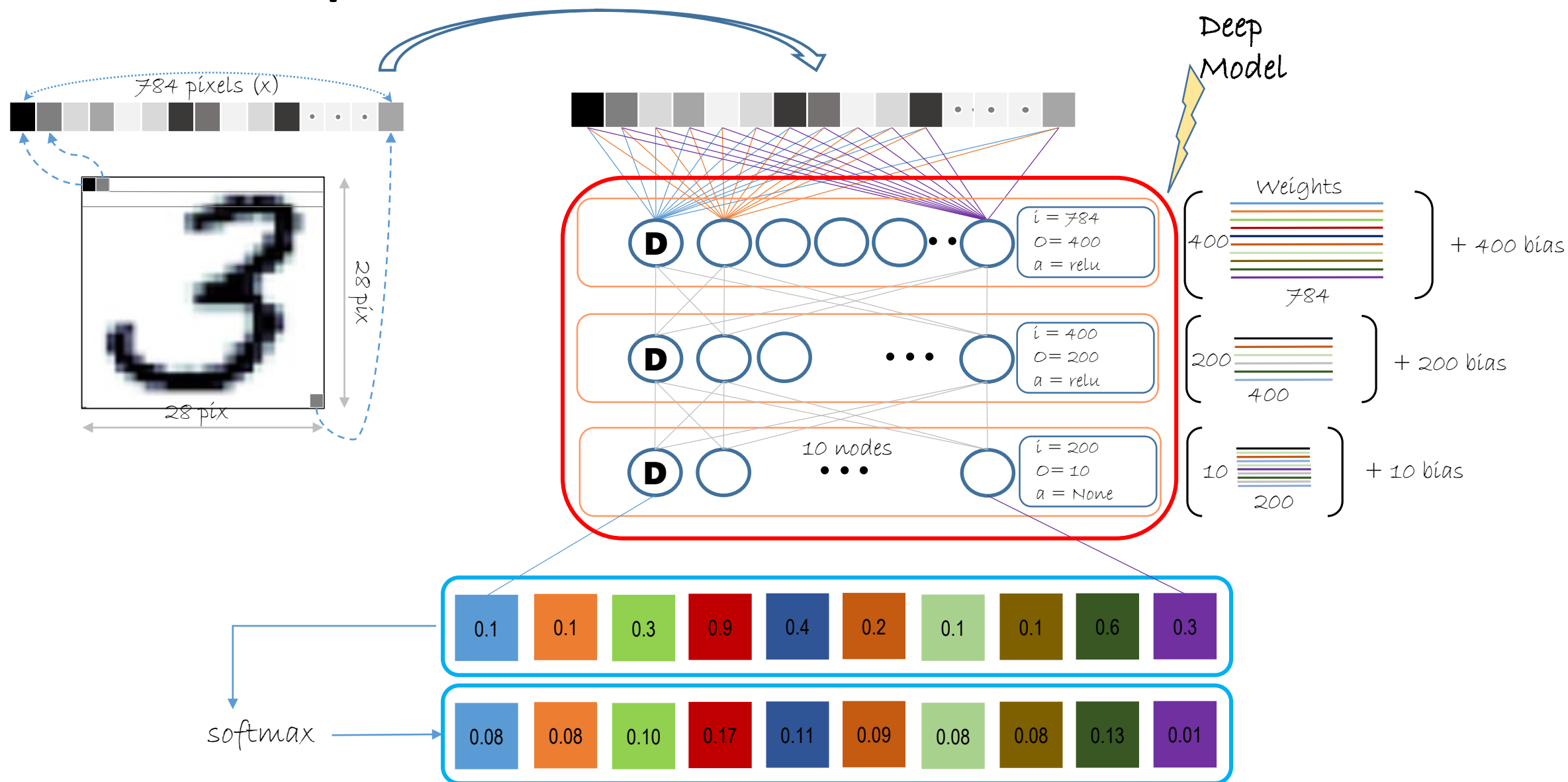
$$f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$$

- Aimed to fix “dying ReLU” problem
- For a negative input, instead of zero have a small negative slope ( $\alpha$ )
  - ✓ Leaky ReLU has a small fixed slope ( $\alpha = 0.1$ )
  - ✓ Parametric ReLU (Param ReLU or PReLU)
    - The slope is parameterized and can be learnt
  - ✓ MaxOut
    - Is applied on the dot product between the weights and the data.

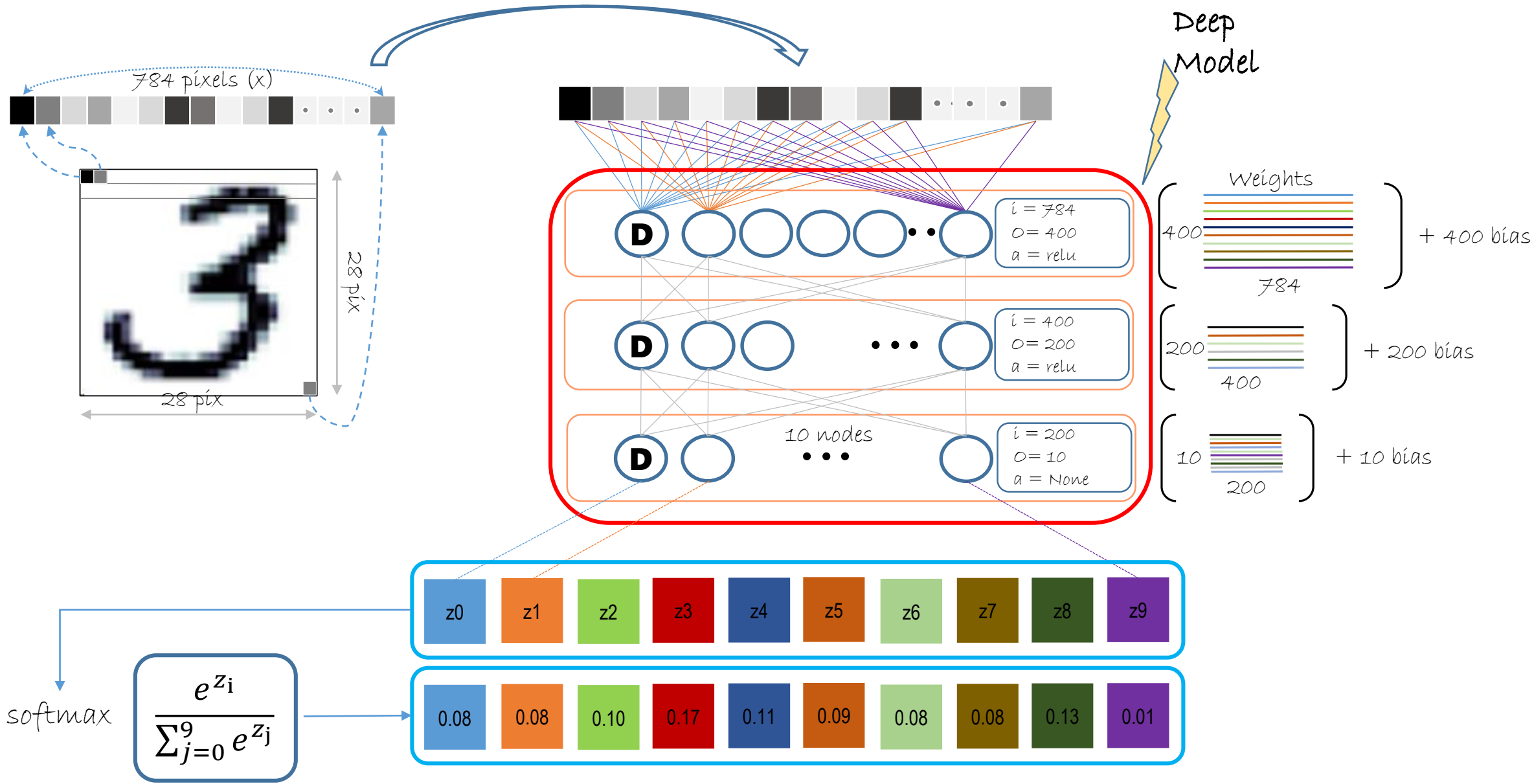
$$f(x) = \max(a_1^T x + c_1, a_2^T x + c_2)$$

- ReLU, LeakyReLU and Param ReLU are special case of Maxout
- Enjoys benefit of all, except number of parameters double

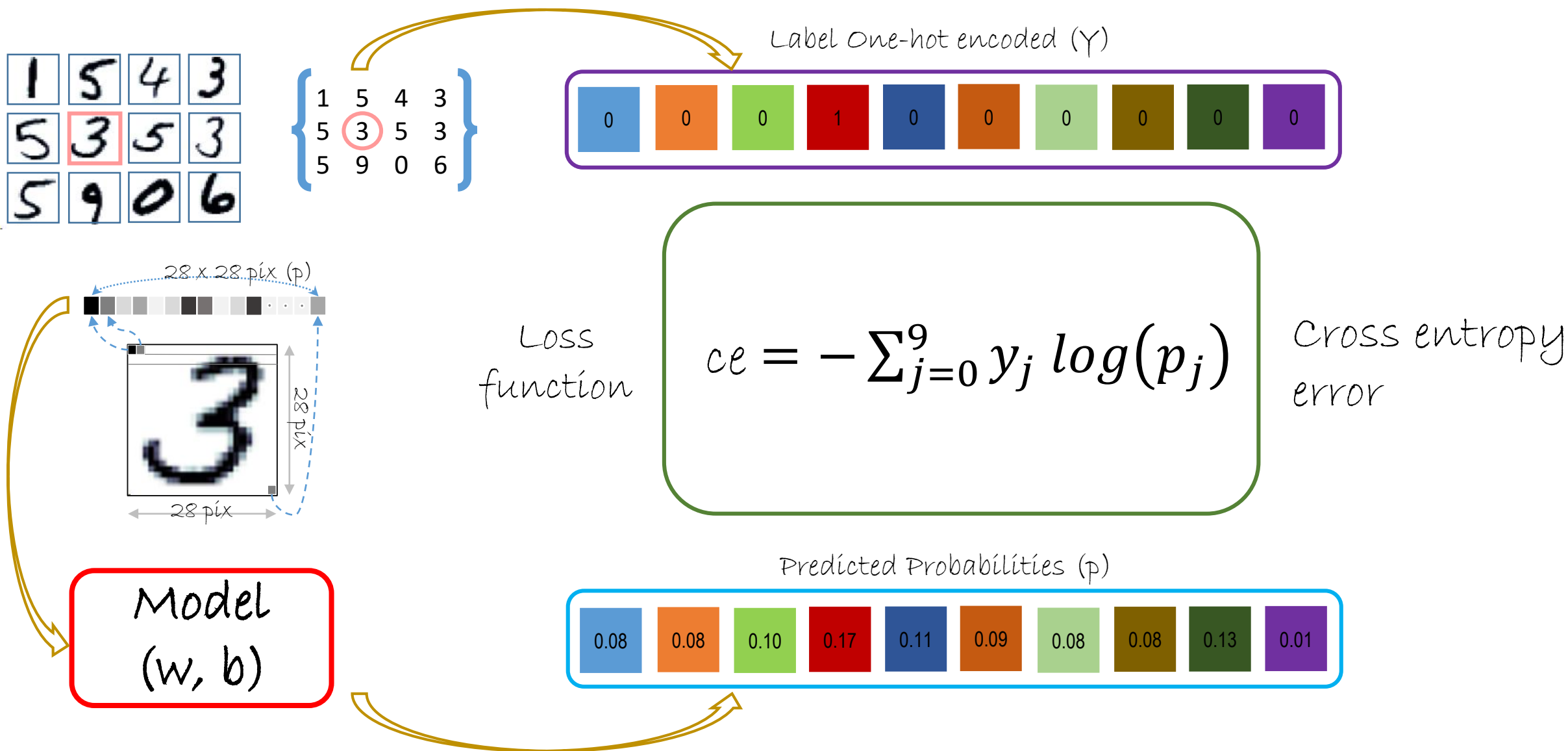
# Our first deep network



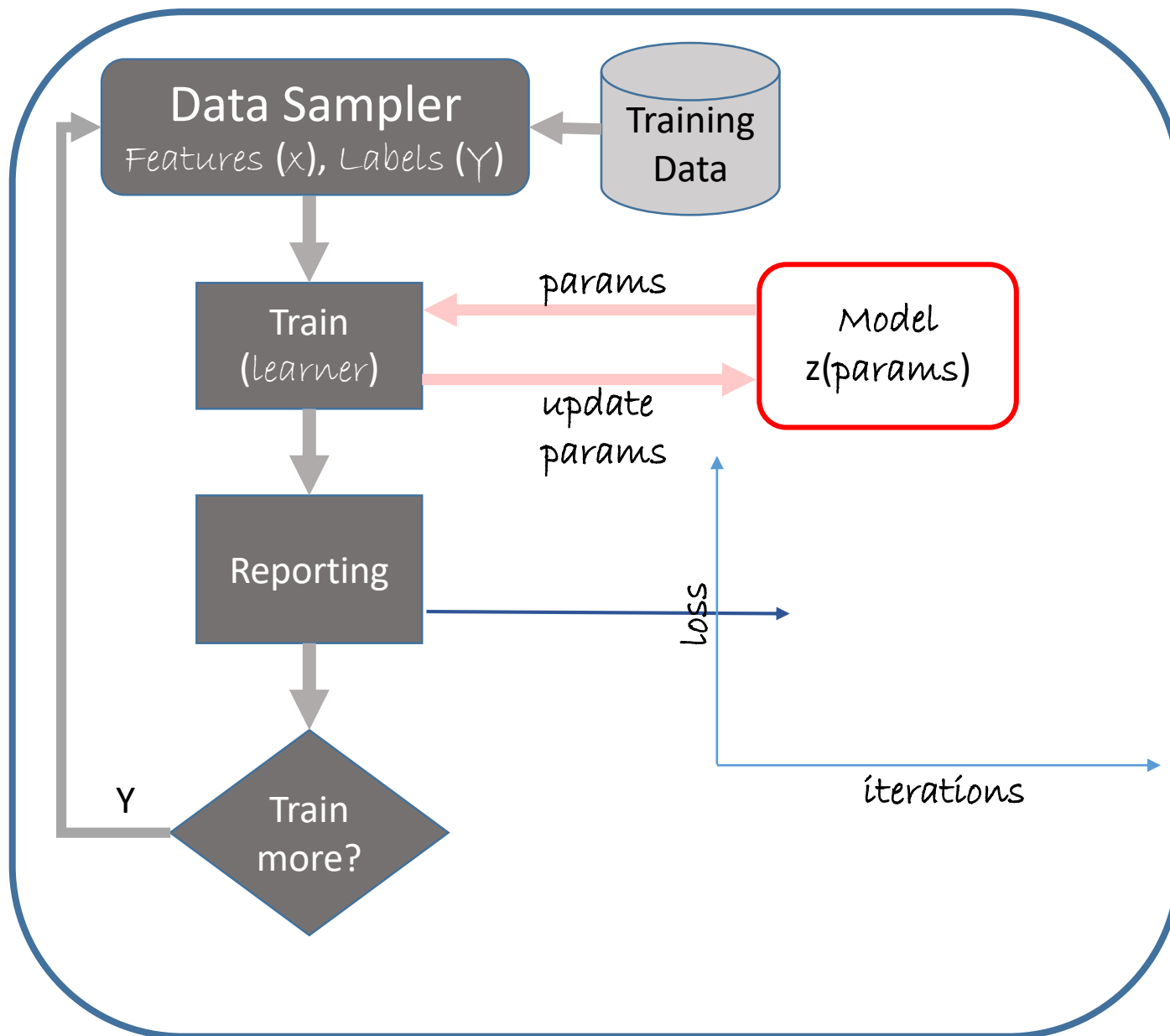
# Multi-layer perceptron



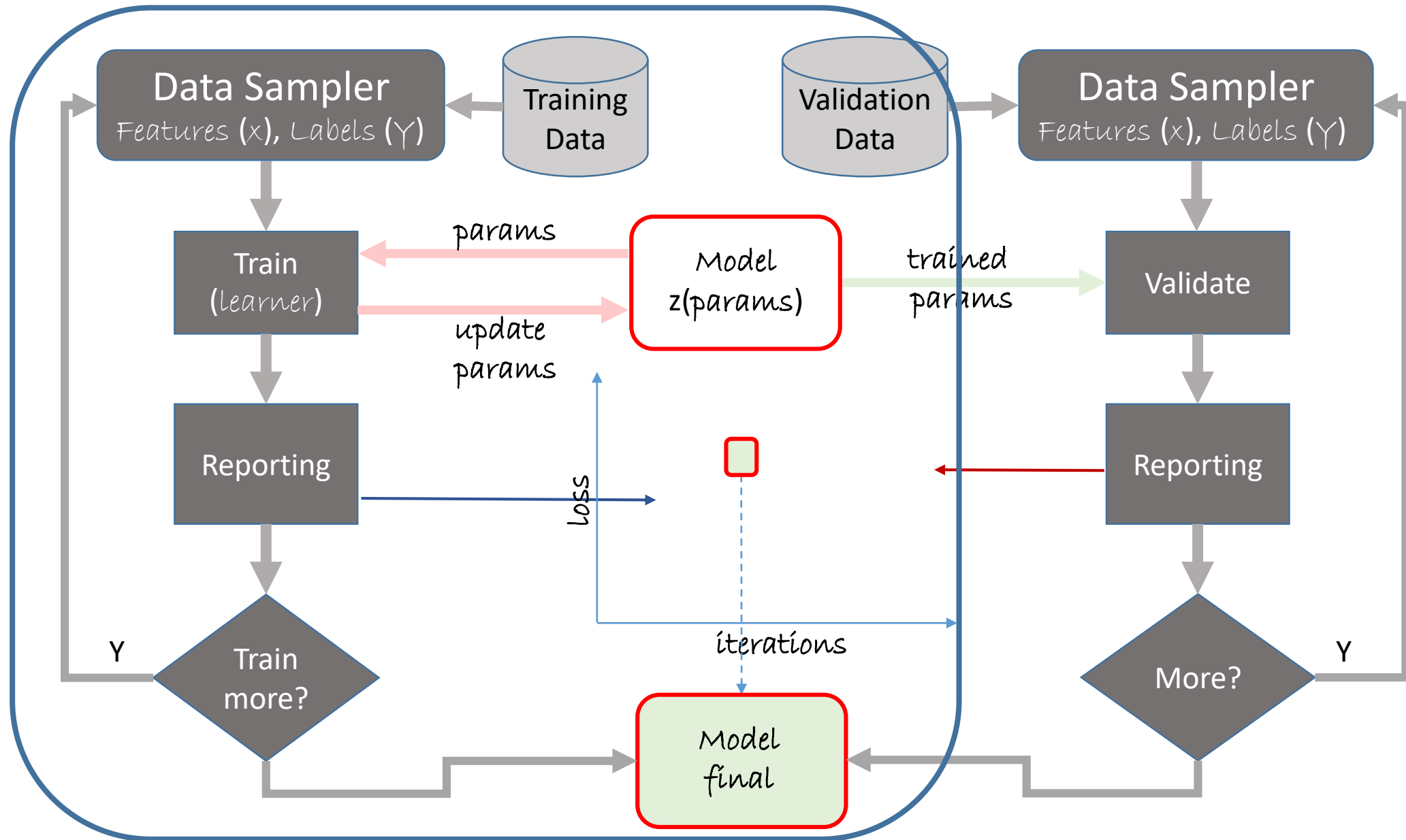
# Loss function



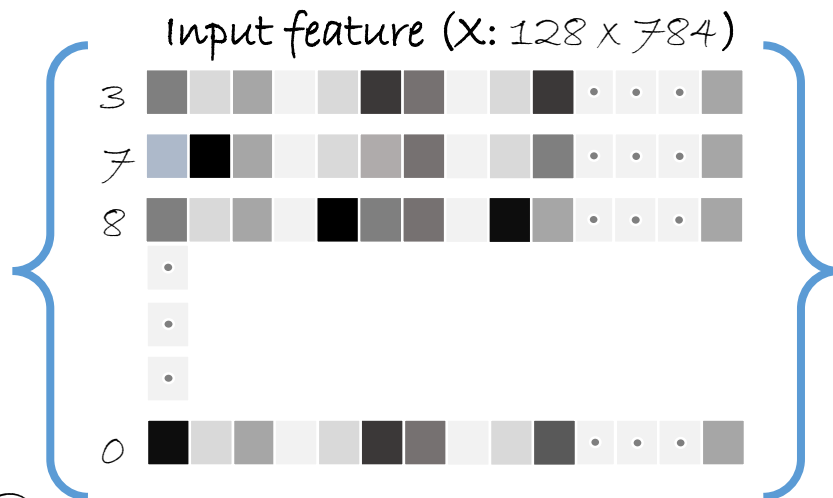
# Train workflow



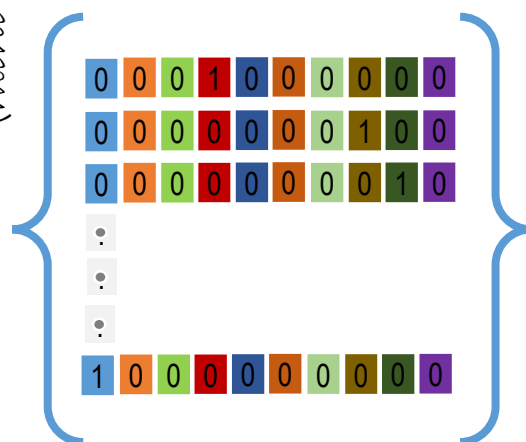
# Validation workflow



# Train workflow

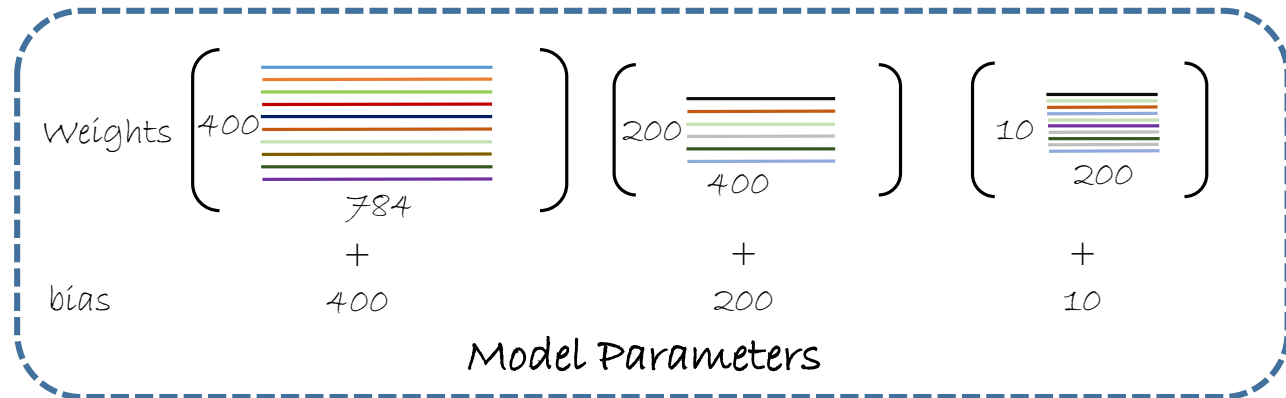


128 samples  
(mini-batch)



One-hot  
encoded  
Label  
(Y: 128 x 10)

Model



```
z = model(X):
    h1 = Dense(400, act = relu)(X)
    h2 = Dense(200, act = relu)(h1)
    r = Dense(10, act = None)(h2)
    return r
```

Loss

```
cross_entropy_with_softmax(z, Y)
```

Error  
(optional)

```
classification_error(z, Y)
```

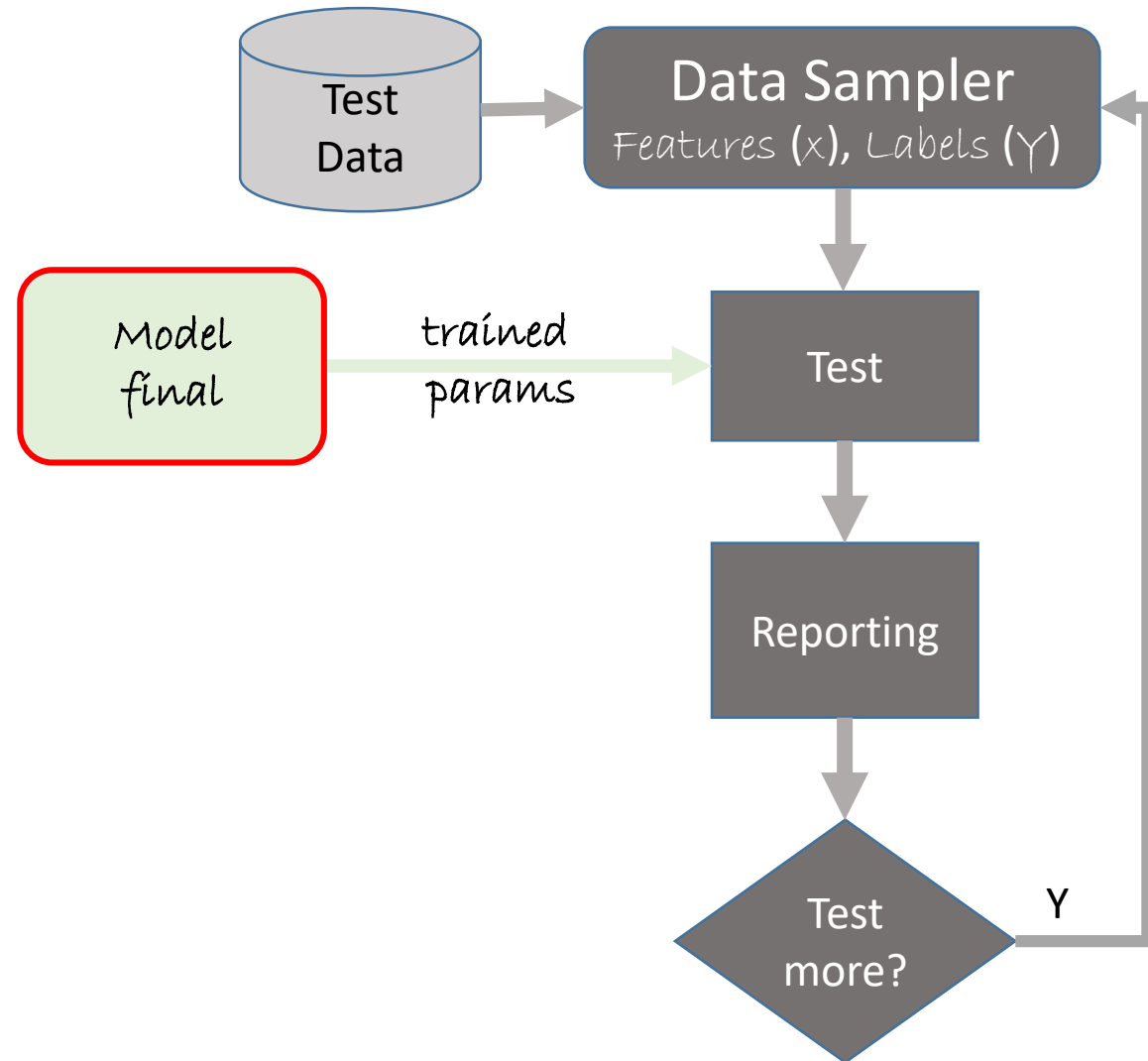
```
Trainer(model, (loss, error), learner)
```

```
Trainer.train_minibatch({X, Y})
```

Learner

sgd, adagrad etc, are solvers to estimate -  $w \& b$

# Test workflow

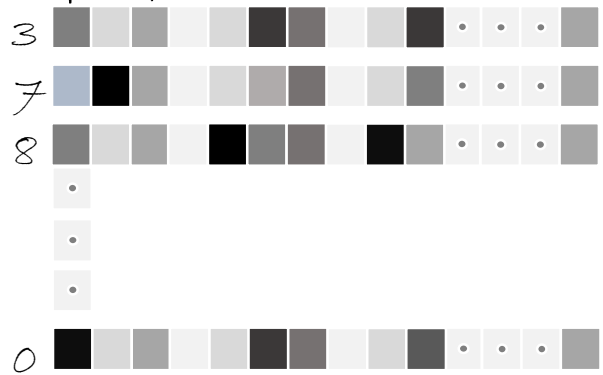




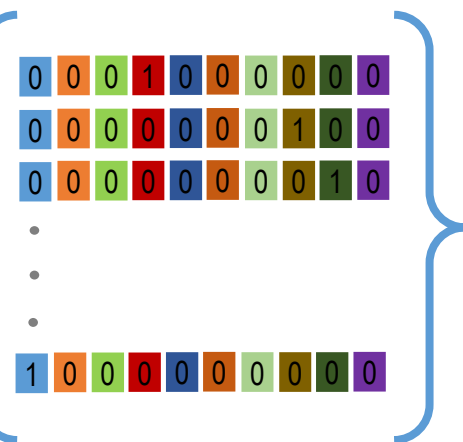
# Test workflow



Input feature ( $X^*$ :  $32 \times 784$ )

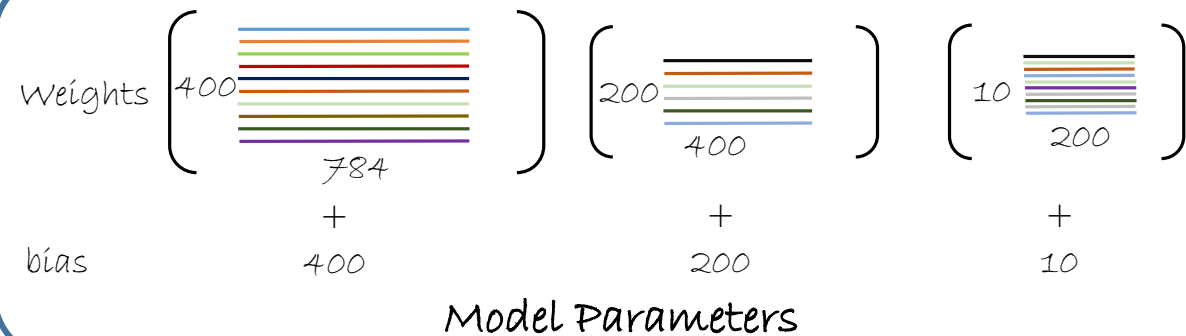


32 samples  
(mini-batch)



One-hot  
encoded  
Label  
( $Y^*$ :  $32 \times 10$ )

Model

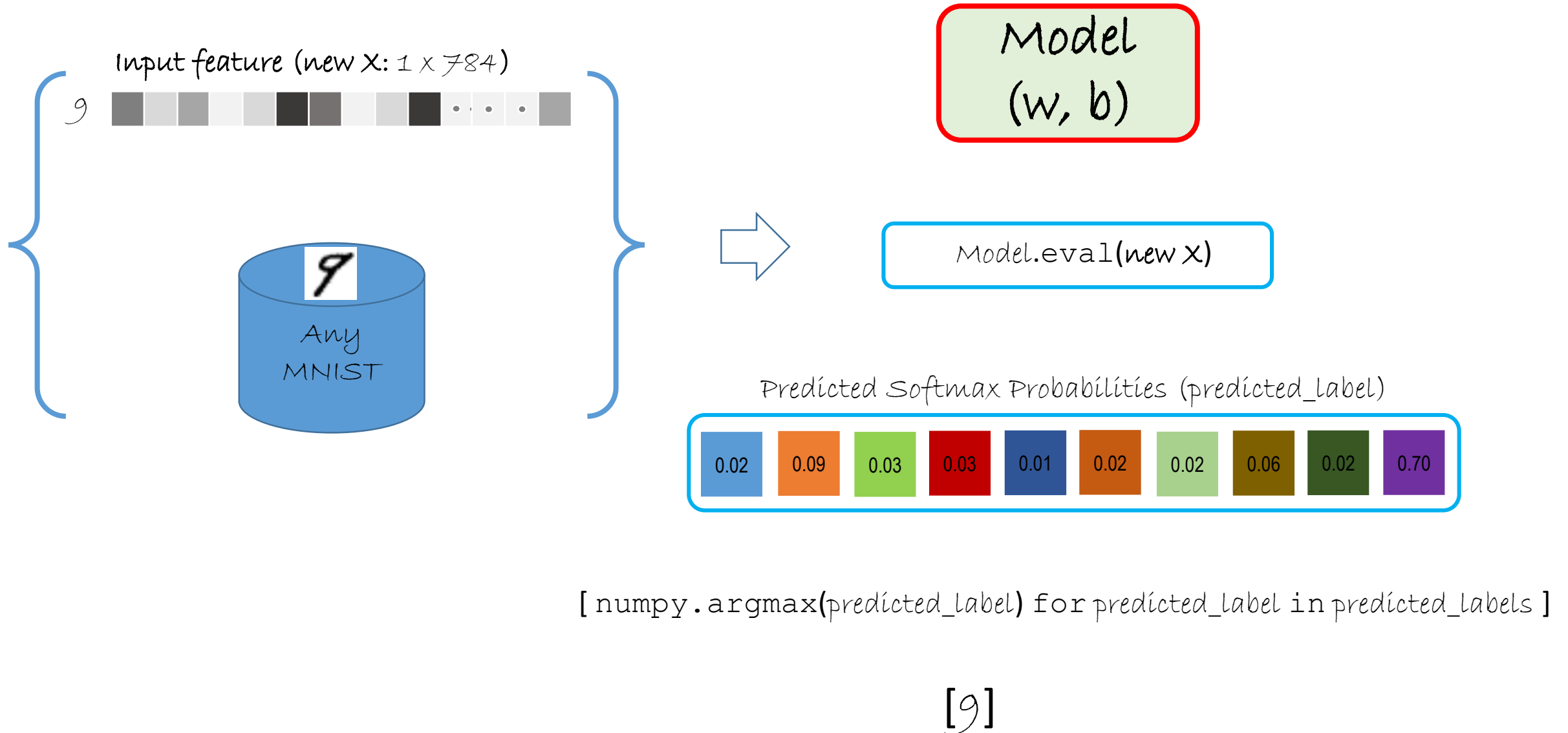


```
z = model(X):
    h1 = Dense(400, act = relu)(X)
    h2 = Dense(200, act = relu)(h1)
    r = Dense(10, act = None)(h2)
    return r
```

`Trainer.test_minibatch({X, Y})`



# Prediction workflow



# Prediction workflow

