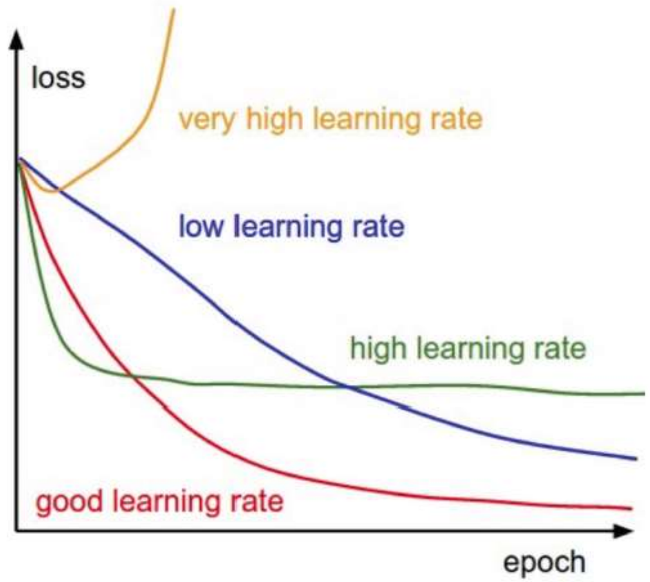
Neural Networks

Dr. Jameel Malik

muhammad.jameel@seecs.edu.pk

Learning Rate

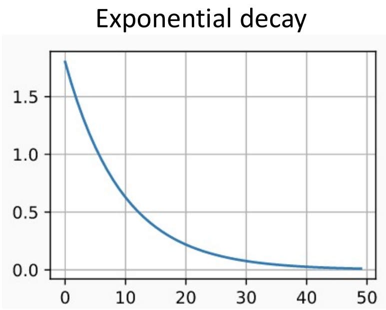
- Training loss for different learning rates
  - High learning rate: the loss increases or plateaus too quickly
  - Low learning rate: the loss decreases too slowly (takes many epochs to reach a solution)

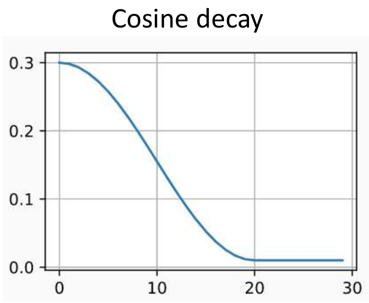


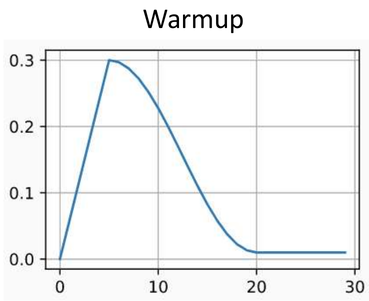
Picture from: https://cs231n.github.io/neural-networks-3/

Learning Rate Scheduling

- *Learning rate scheduling* is applied to change the values of the learning rate during the training
  - *Annealing* is reducing the learning rate over time (a.k.a. learning rate decay)
    - Approach 1: reduce the learning rate by some factor every few epochs
      - Typical values: reduce the learning rate by a half every 5 epochs, or divide by 10 every 20 epochs
    - Approach 2: exponential or cosine decay gradually reduce the learning rate over time
    - Approach 3: reduce the learning rate by a constant (e.g., by half) whenever the validation loss stops improving
  - *Warmup* is gradually increasing the learning rate initially, and afterward let it cool down until the end of the training

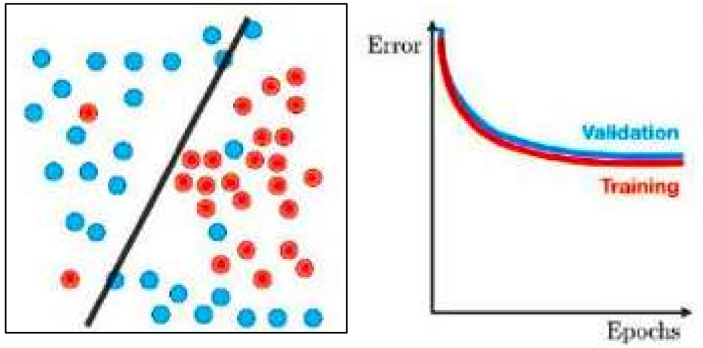




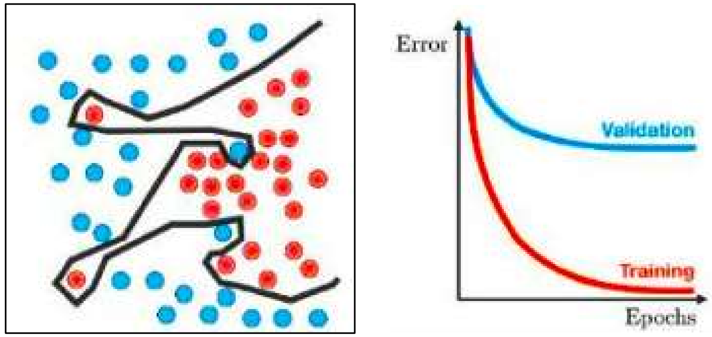


Generalization

- *Underfitting*
  - The model is too "simple" to represent all the relevant class characteristics
  - E.g., model with too few parameters
  - Produces high error on the training set and high error on the validation set

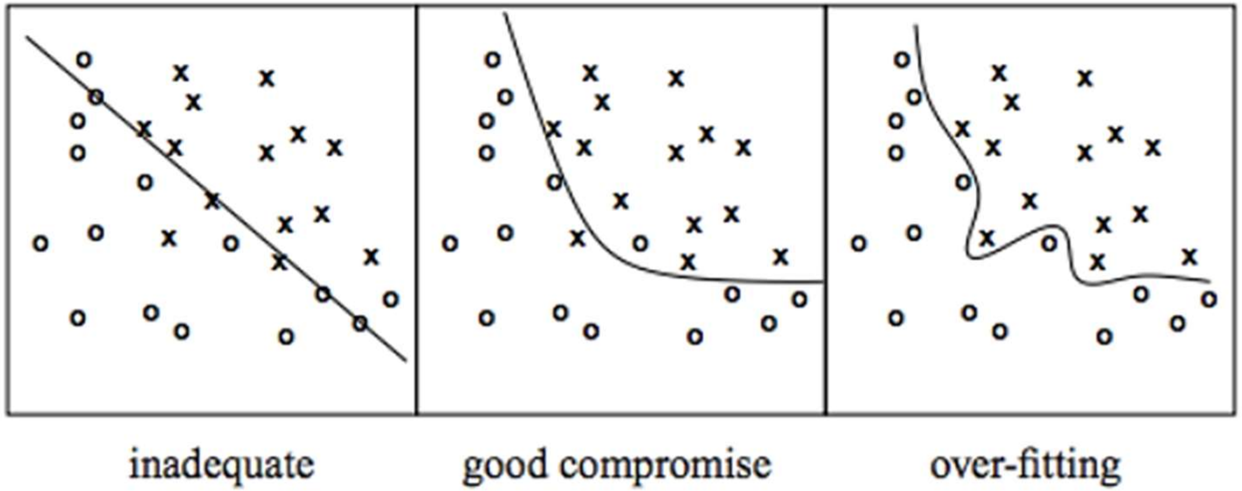


- *Overfitting*
  - The model is too "complex" and fits irrelevant characteristics (noise) in the data
  - E.g., model with too many parameters
  - Produces low error on the training error and high error on the validation set



Model Overfitting

- **Overfitting** – a model with high capacity fits the noise in the data instead of the underlying relationship



Picture from: http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

Regularization: Weight Decay

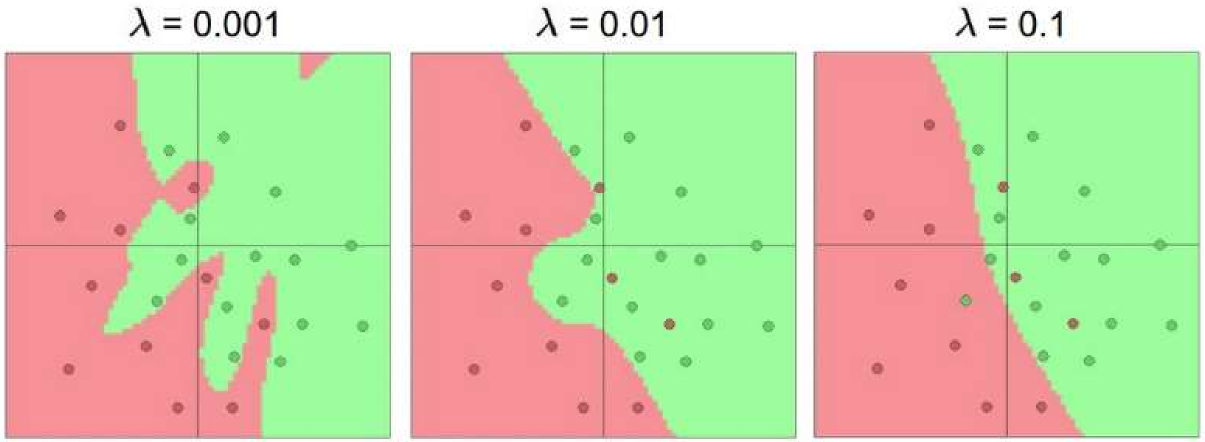
- *ℓ_2 weight decay*
  - A regularization term that penalizes large weights is added to the loss function

$$\mathcal{L}_{reg}(\theta) = \underbrace{\mathcal{L}(\theta)}_{\text{Data loss}} + \underbrace{\lambda \sum_k \theta_k^2}_{\text{Regularization loss}}$$

  - For every weight in the network, we add the regularization term to the loss value
  - The weight decay coefficient λ determines how dominant the regularization is during the gradient computation

Regularization: Weight Decay

- Effect of the decay coefficient λ
  - Large weight decay coefficient → penalty for weights with large values



Regularization: Weight Decay

- ***ℓ_1 weight decay***
  - The regularization term is based on the ℓ_1 norm of the weights

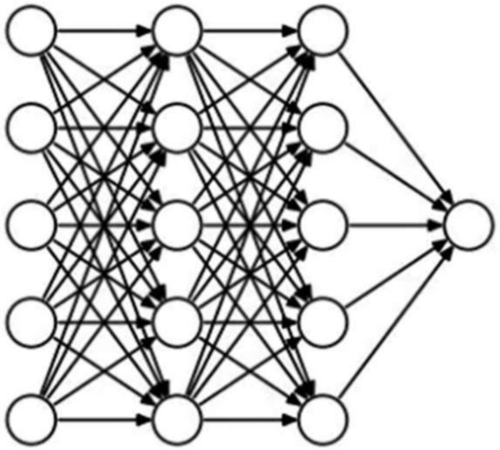
$$\mathcal{L}_{reg}(\theta) = \mathcal{L}(\theta) + \lambda \sum_k |\theta_k|$$

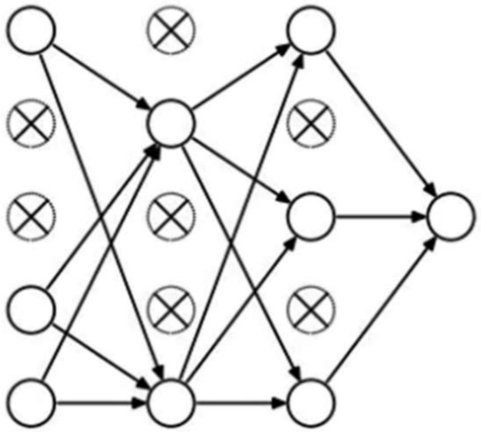
  - ℓ_1 weight decay is less common with NN
    - Often performs worse than ℓ_2 weight decay
  - It is also possible to combine ℓ_1 and ℓ_2 regularization
    - Called elastic net regularization

$$\mathcal{L}_{reg}(\theta) = \mathcal{L}(\theta) + \lambda_1 \sum_k |\theta_k| + \lambda_2 \sum_k \theta_k^2$$

Regularization: Dropout

- *Dropout*
  - Randomly drop units (along with their connections) during training
  - Each unit is retained with a fixed dropout rate p, independent of other units
  - The hyper-parameter p needs to be chosen (tuned)
    - Often, between 20% and 50% of the units are dropped

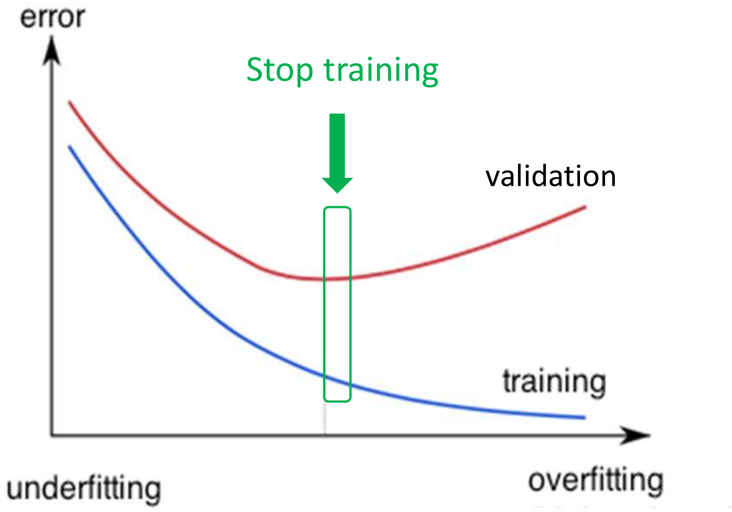




Slide credit: Hung-yi Lee – Deep Learning Tutorial

Regularization: Early Stopping

- *Early-stopping*
  - During model training, use a validation set
    - E.g., validation/train ratio of about 25% to 75%
  - Stop when the validation accuracy (or loss) has not improved after n epochs
    - The parameter n is called patience

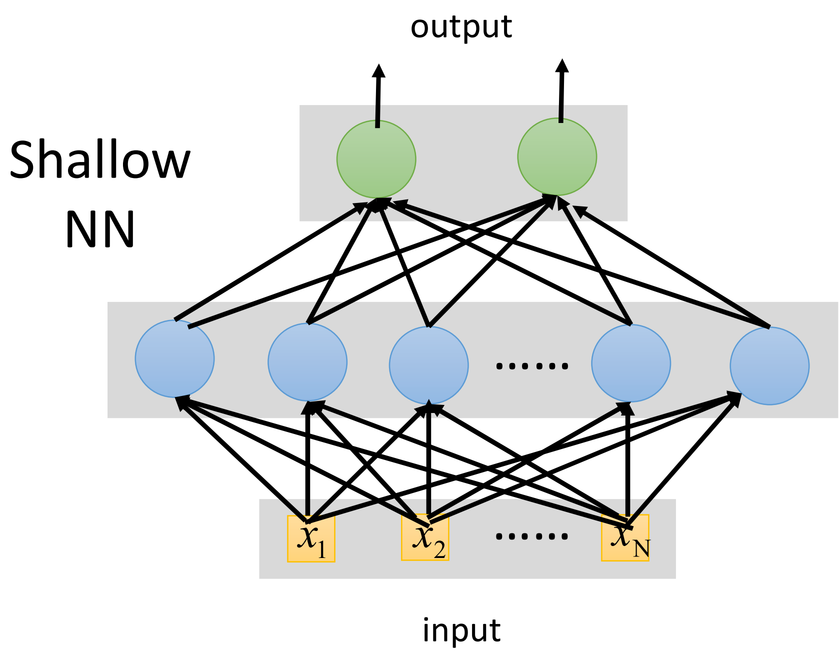


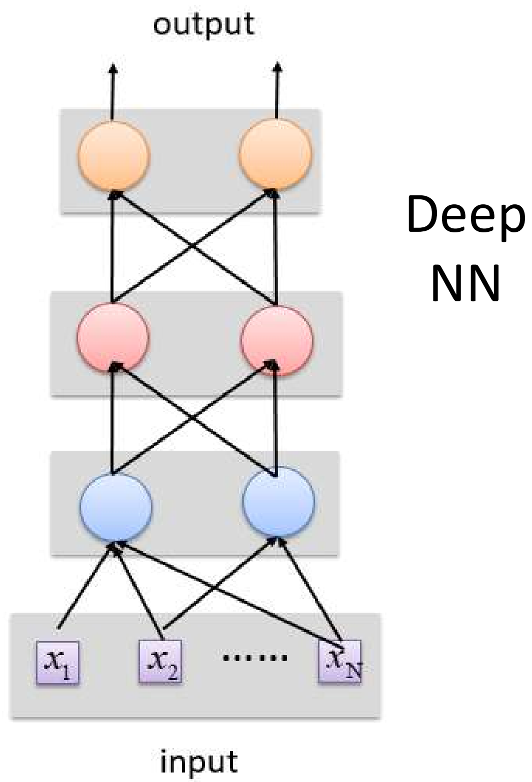
Hyper-parameter Tuning

- Training NNs can involve setting many ***hyper-parameters***
- The most common hyper-parameters include:
  - Number of layers, and number of neurons per layer
  - Initial learning rate
  - Learning rate decay schedule (e.g., decay constant)
  - Optimizer type
- Other hyper-parameters may include:
  - Regularization parameters (ℓ_2 penalty, dropout rate)
  - Batch size
  - Activation functions
  - Loss function
- Hyper-parameter tuning can be time-consuming for larger NNs

Deep vs Shallow Networks

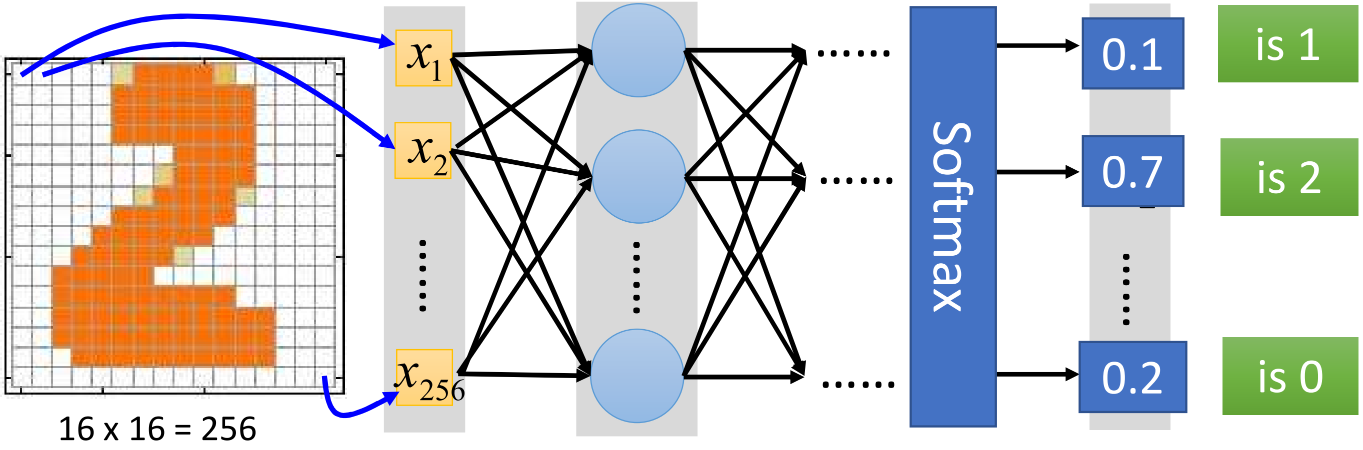
- Deeper networks perform better than shallow networks
  - But only up to some limit: after a certain number of layers, the performance of deeper networks plateaus





Slide credit: Hung-yi Lee – Deep Learning Tutorial

MNIST Digits Classification using MLP

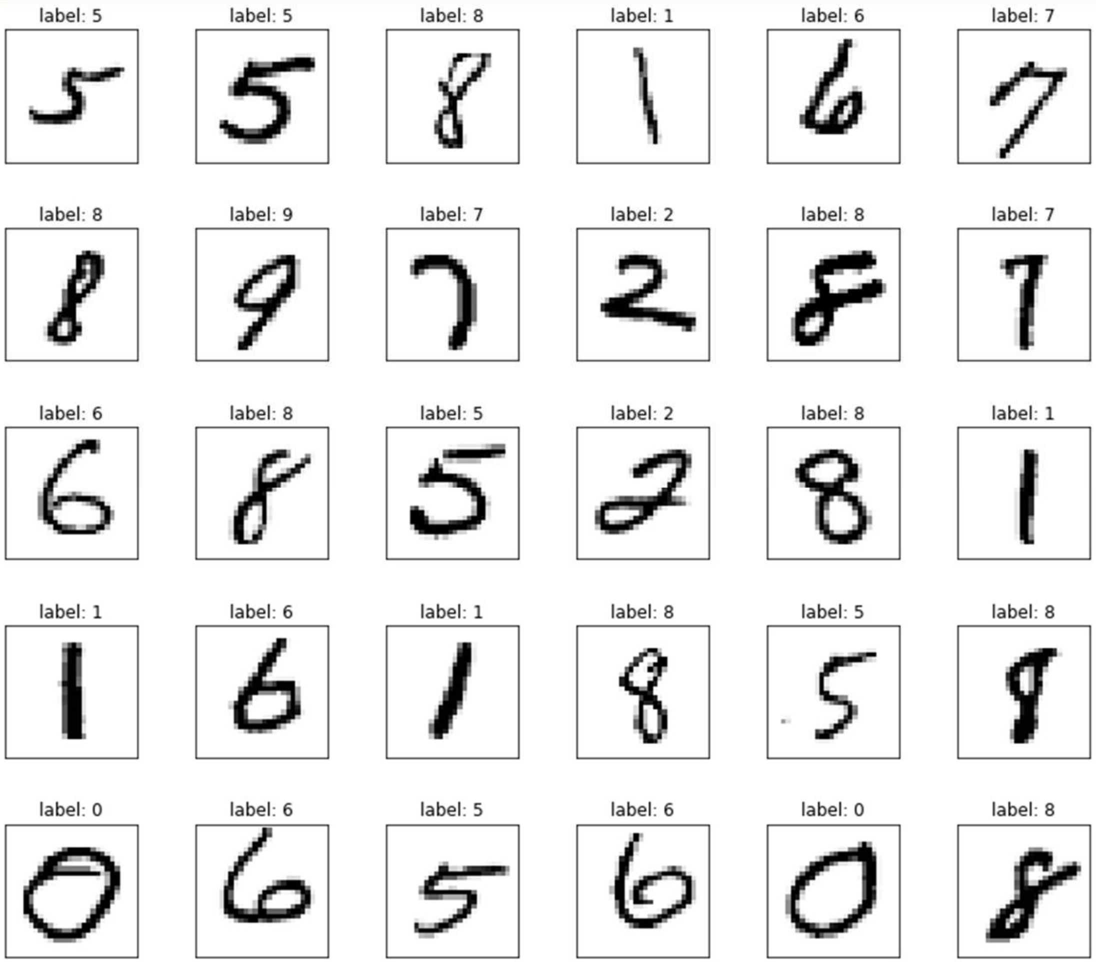


MNIST Dataset

- A database of handwritten digits
  - Training set: 60,000 examples
  - Test set: 10,000 examples

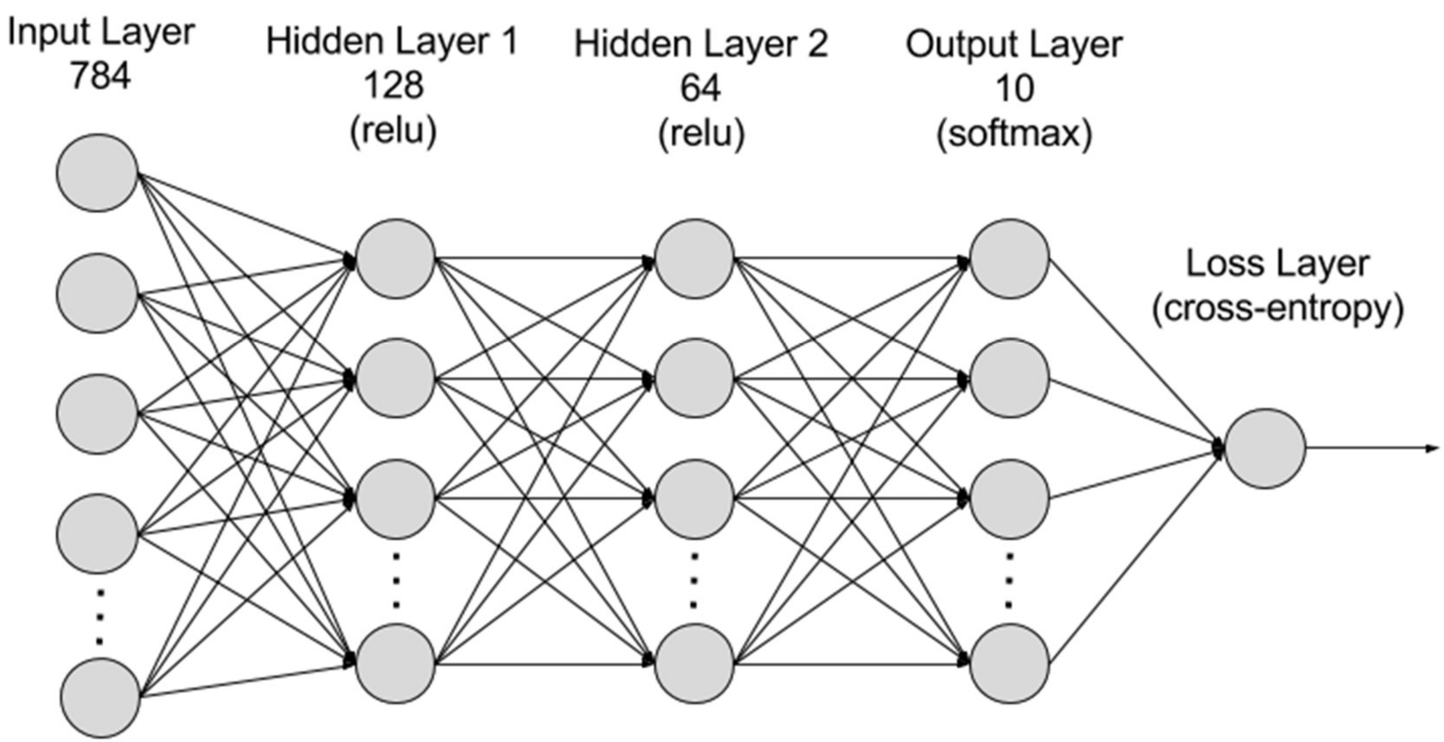
  - Image size: **28x28**
    - Pre-processed data(e.g., the digits are placed in the center of image).
    - Labels are provided.

- Input representation:
  - 28x28 = 784 values (x_1, x_2,… ,x_{784})
- Output representation
  - One-hot encoding (y_1, y_2, … ,y_{10})



MNIST Digits Classification using MLP

- One possible model

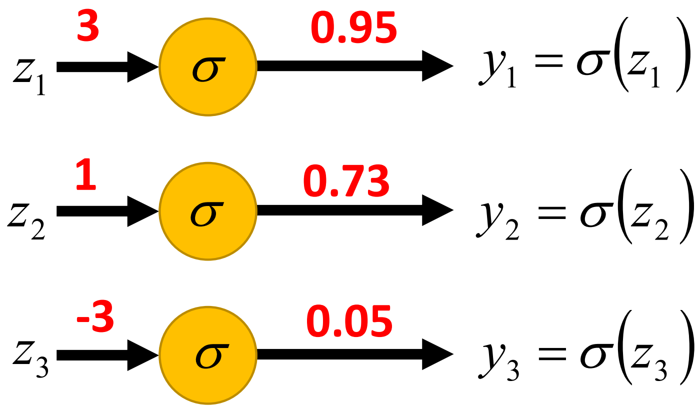


Softmax Layer

- In multi-class classification tasks, the output layer is typically a ***softmax layer***
  - i.e., it employs a ***softmax activation function***

  - If a layer with a sigmoid activation function is used as the output layer instead, the predictions by the NN may not be easy to interpret
    - Note that an output layer with sigmoid activations can still be used for binary classification

A Layer with Sigmoid Activations



Softmax Operation

- The softmax layer applies softmax activations to output a probability value in the range [0, 1]
  - The values z inputted to the softmax layer are referred to as ***logits***

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer