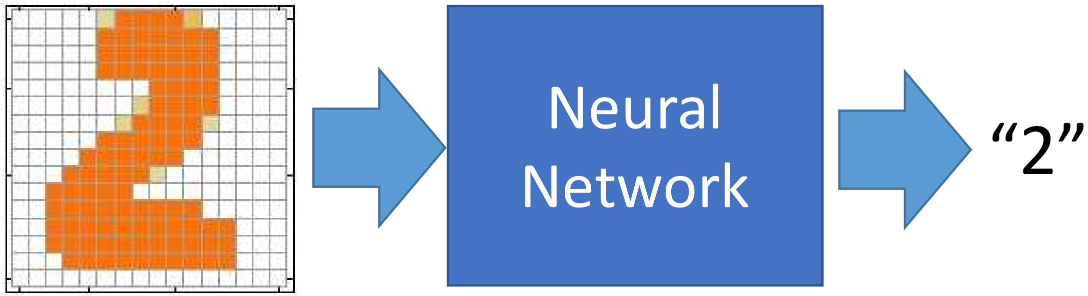
Neural Networks

Dr. Jameel Malik

muhammad.jameel@seecs.edu.pk

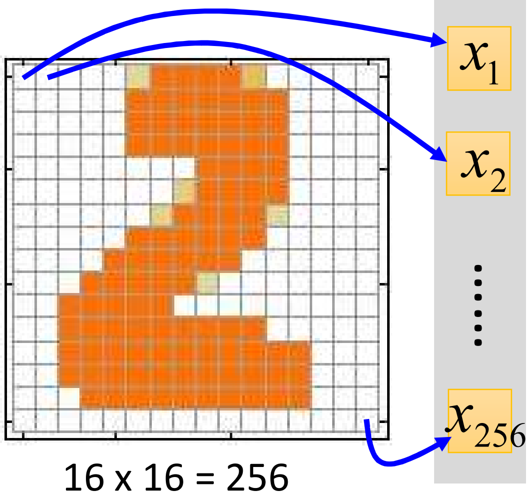
Example Application

- Handwriting Digit Recognition

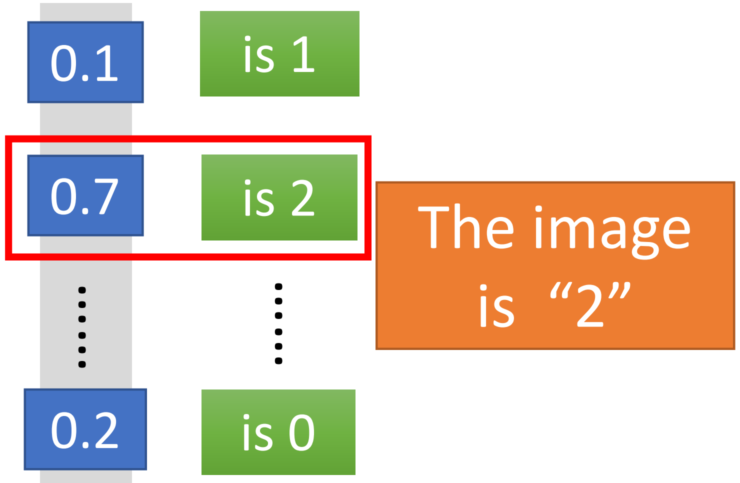


Handwriting Digit Recognition

Input



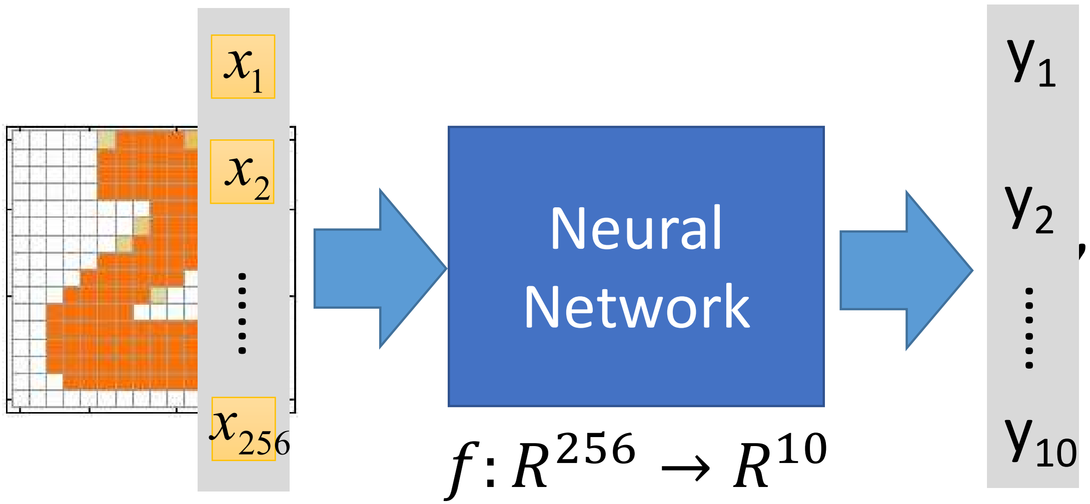
Output



Each dimension represents the confidence of a digit.

Example Application

- Handwriting Digit Recognition



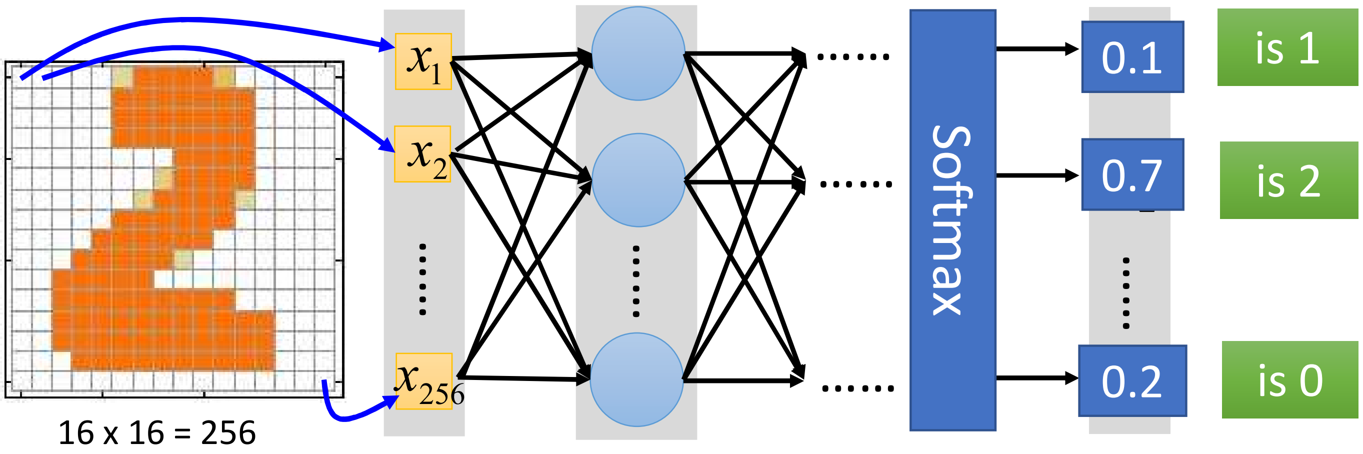
In deep learning, the function f is represented by neural network

Training NNs

- The network ***parameters*** θ include the weight matrices and bias vectors from all layers

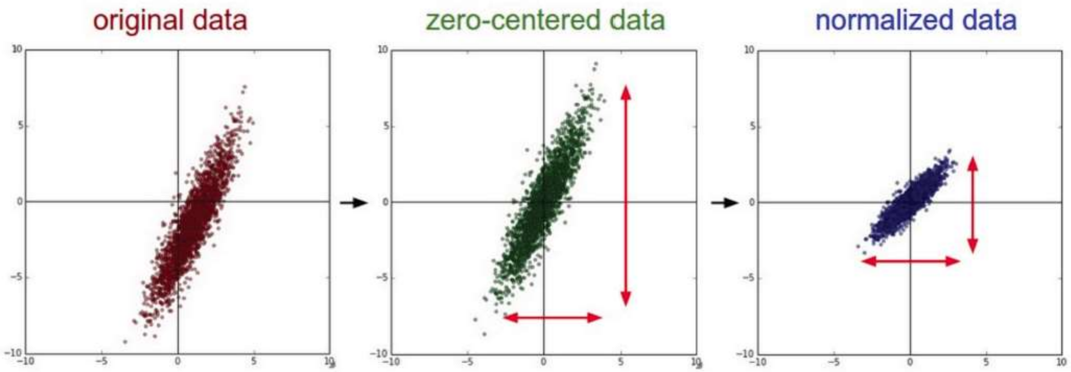
$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$

  - Often, the model parameters θ are referred to as weights
- Training a model to learn a set of parameters θ that are optimal (according to a criterion) is one of the greatest challenges in ML



Training NNs

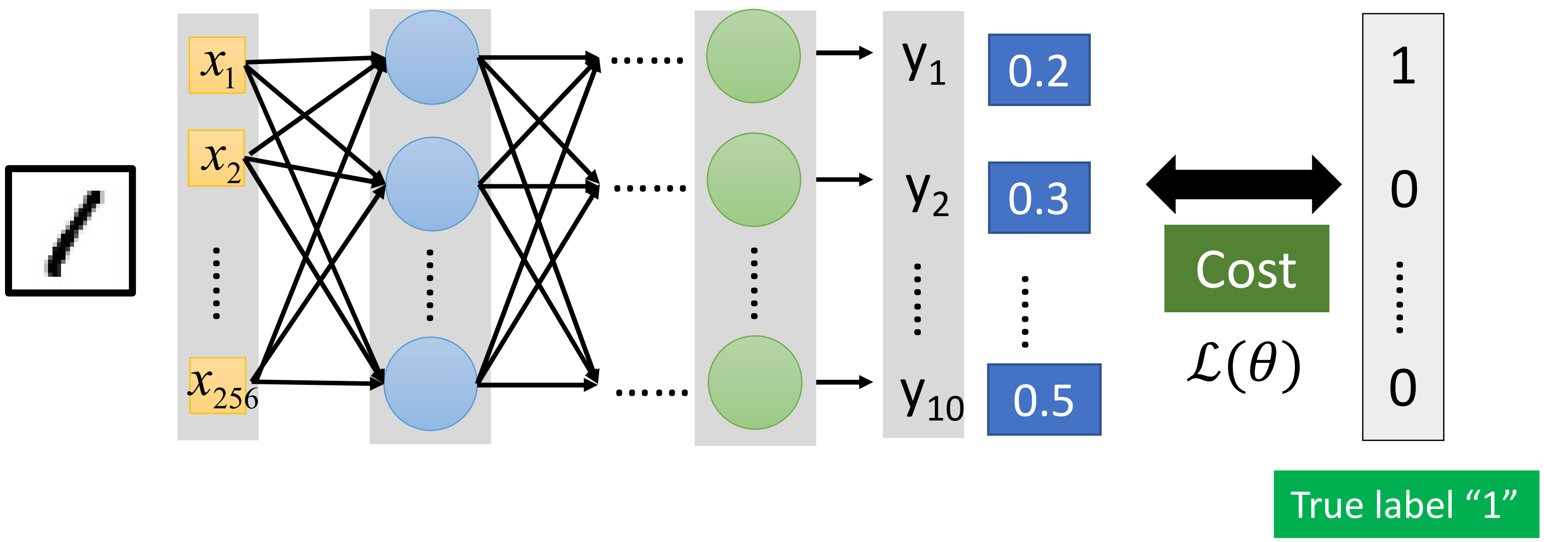
- *Data preprocessing* - helps convergence during training
  - Mean subtraction, to obtain zero-centered data
    - Subtract the mean for each individual data dimension (feature)
  - Normalization
    - Divide each feature by its standard deviation
      - To obtain standard deviation of 1 for each data dimension (feature)
    - Or, scale the data within the range [0,1] or [-1, 1]
      - E.g., image pixel intensities are divided by 255 to be scaled in the [0,1] range



Picture from: https://cs231n.github.io/neural-networks-2/

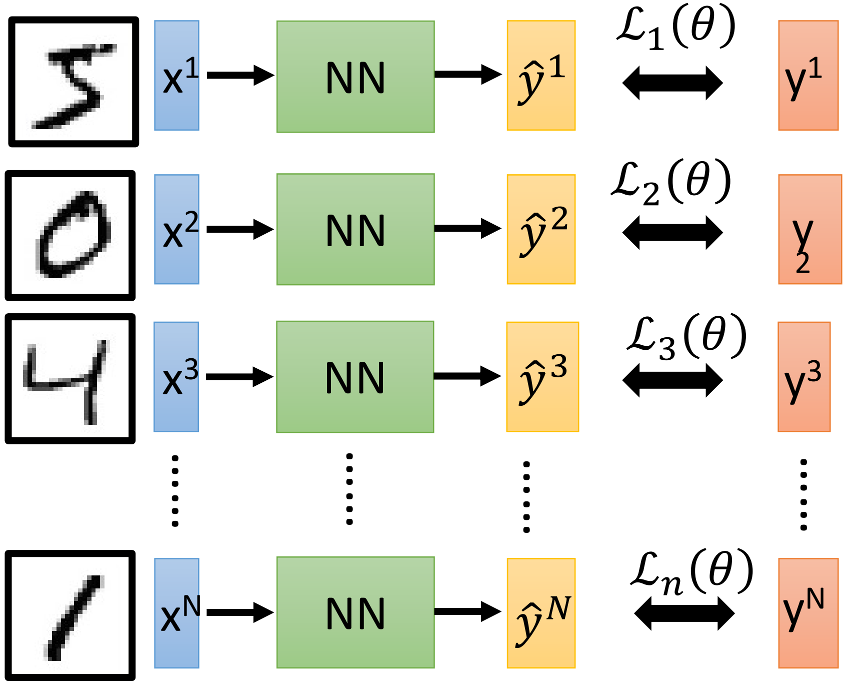
Training NNs

- Define a ***loss function***/objective function/cost function $\mathcal{L}(\theta)$ that calculates the difference (error) between the model prediction and the true label
  - E.g., $\mathcal{L}(\theta)$ can be mean-squared error, cross-entropy, etc.



Training NNs

- For a training set of N images, calculate the total loss overall all images: $\mathcal{L}(\theta) = \sum_{n=1}^{N} \mathcal{L}_n(\theta)$
- Find the optimal parameters θ^* that minimize the total loss $\mathcal{L}(\theta)$



Slide credit: Hung-yi Lee – Deep Learning Tutorial

Loss Functions

- ***Classification tasks***

Training examples	Pairs of N inputs x_i and ground-truth class labels y_i	
Output Layer	Softmax Activations [maps to a probability distribution]	$P(y=j \mid \mathbf{x}) = \dfrac{e^{\mathbf{x}^{\mathsf{T}}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^{\mathsf{T}}\mathbf{w}_k}}$
Loss function	***Cross-entropy***	$\mathcal{L}(\theta) = -\dfrac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left[y_k^{(i)} \log \hat{y}_k^{(i)} + \left(1 - y_k^{(i)}\right) \log\left(1 - \hat{y}_k^{(i)}\right)\right]$

Ground-truth class labels y_i and model predicted class labels $\hat{y}_i$

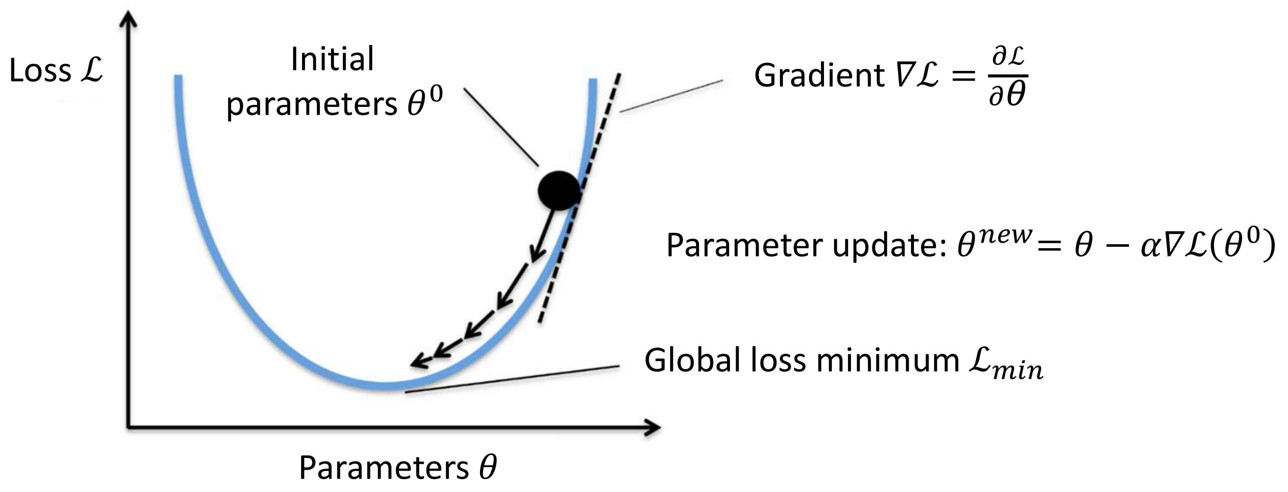
Loss Functions

- *Regression tasks*

Training examples	Pairs of N inputs x_i and ground-truth output values y_i
Output Layer	Linear (Identity) or Sigmoid Activation
Loss function	***Mean Squared Error*** $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$ ***Mean Absolute Error*** $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\|y^{(i)} - \hat{y}^{(i)}\right\|$

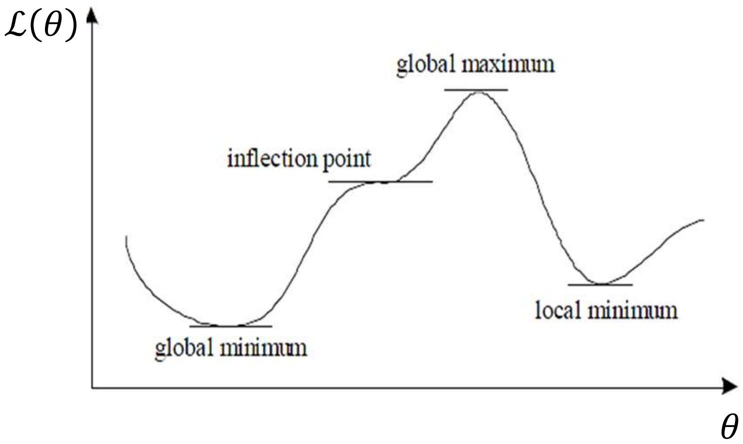
Gradient Descent Algorithm

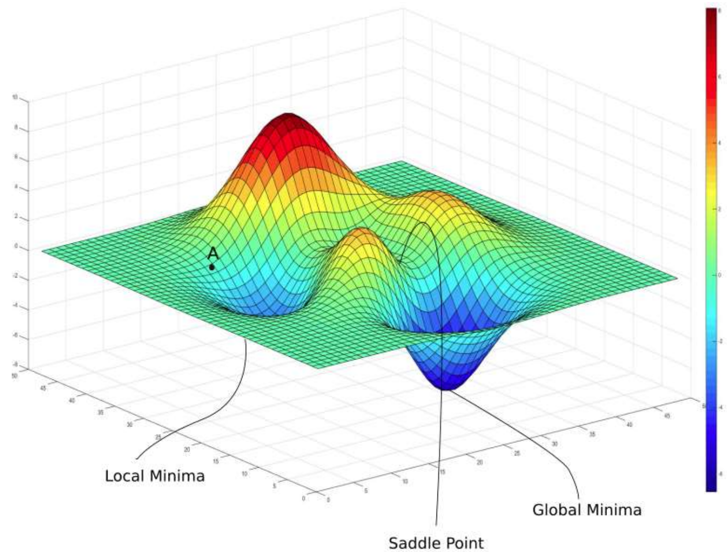
- Steps in the ***gradient descent algorithm***:
  1. Randomly initialize the model parameters, θ^0
  2. Compute the gradient of the loss function at the initial parameters θ^0: $\nabla\mathcal{L}(\theta^0)$
  3. Update the parameters as: $\theta^{new} = \theta^0 - \alpha\nabla\mathcal{L}(\theta^0)$
     - Where α is the learning rate
  4. Go to step 2 and repeat (until a terminating criterion is reached)



Gradient Descent Algorithm

- Gradient descent algorithm stops when a local minimum of the loss surface is reached
  - GD does not guarantee reaching a global minimum
  - However, empirical evidence suggests that GD works well for NNs



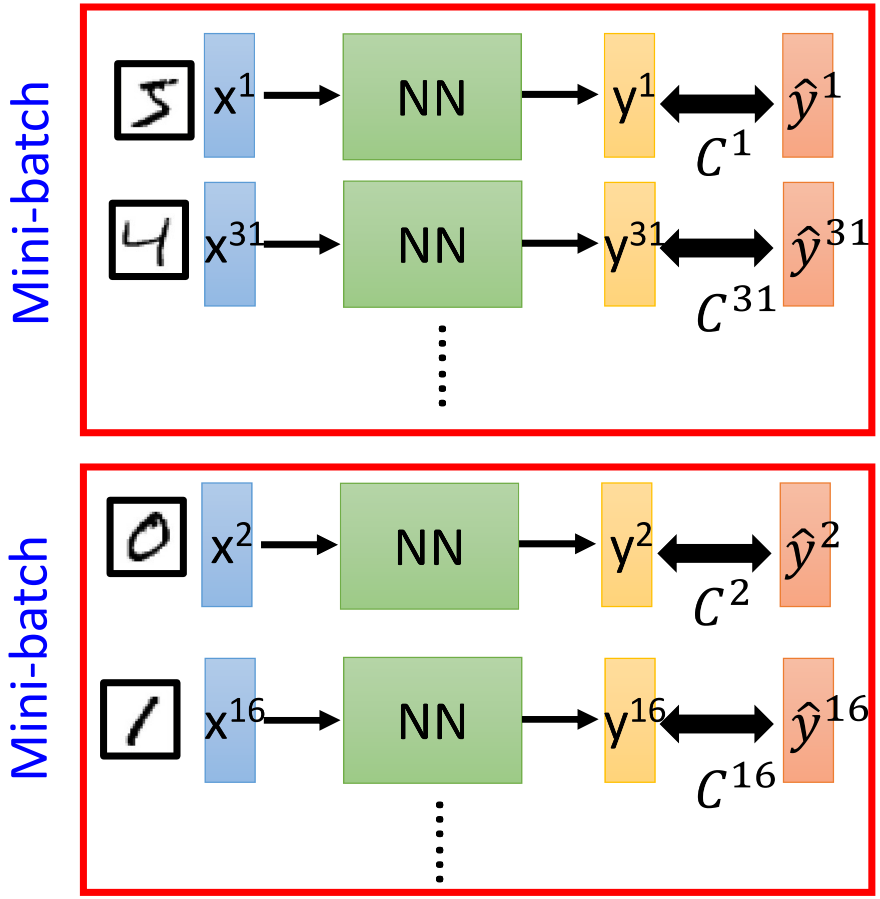


Picture from: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

Mini-batch Gradient Descent

- It is wasteful to compute the loss over the entire training dataset to perform a single parameter update for large datasets
  - E.g., ImageNet has 14M images
  - Therefore, GD (a.k.a. vanilla GD) is almost always replaced with mini-batch GD
- ***Mini-batch GD (or Stochastic GD)***
  - Approach:
    - Compute the loss $\mathcal{L}(\theta)$ on a mini-batch of images, update the parameters θ, and repeat until all images are used
    - At the next epoch, shuffle the training data, and repeat the above process
  - Mini-batch GD results in much faster training
  - Typical mini-batch size: 32 to 256 images
  - It works because the gradient from a mini-batch is a good approximation of the gradient from the entire training set

Mini-batch



- Randomly initialize θ^0
- Pick the 1st batch

  $C = C^1 + C^{31} + \cdots$

  $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$
- Pick the 2nd batch

  $C = C^2 + C^{16} + \cdots$

  $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

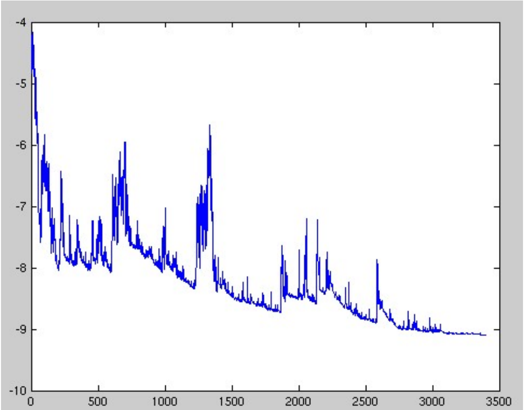
  ⋮
- Until all mini-batches have been picked

one epoch

Repeat the above process

Stochastic Gradient Descent

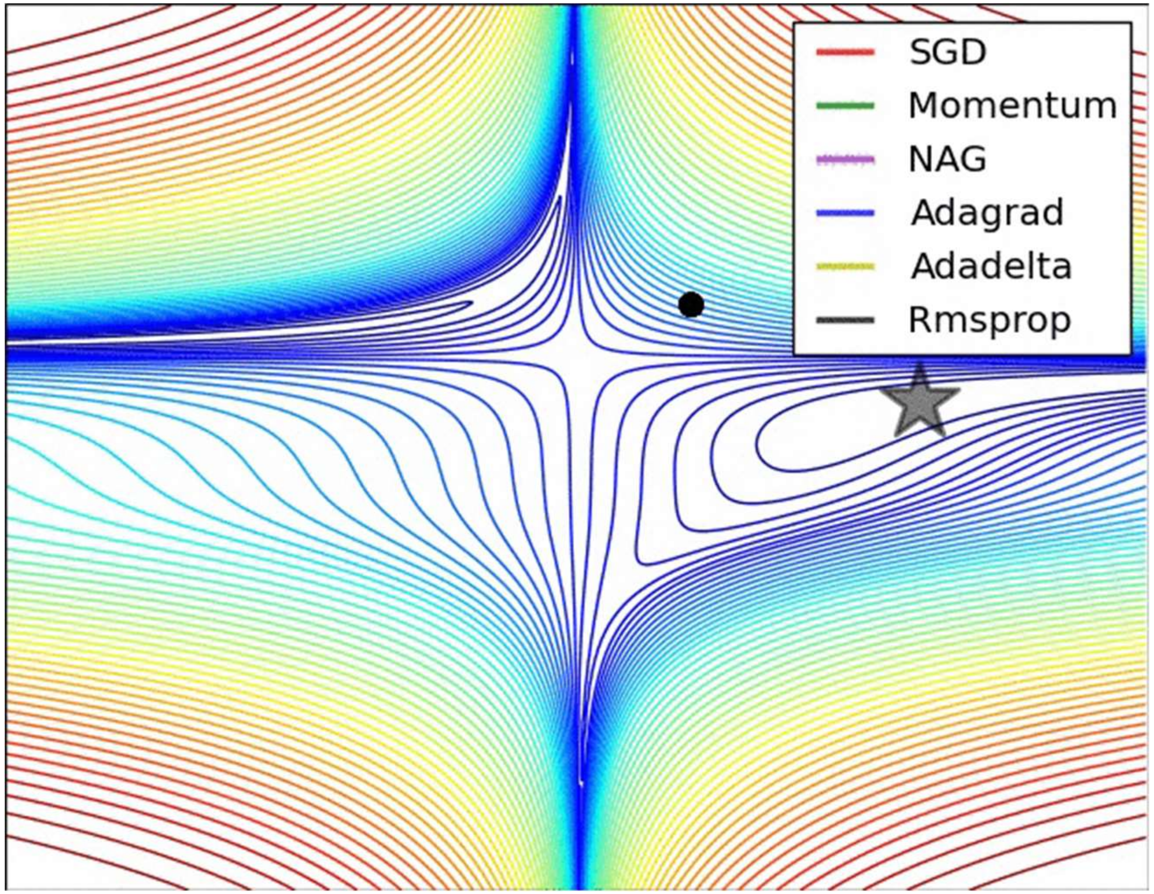
- ***Stochastic gradient descent***
  - SGD uses mini-batches that consist of a single input example
    - E.g., one image mini-batch
  - Although this method is very fast, it may cause significant fluctuations in the loss function
    - Therefore, it is less commonly used, and mini-batch GD is preferred
  - In most DL libraries, SGD typically means a mini-batch GD (with an option to add momentum)



Other Optimizers

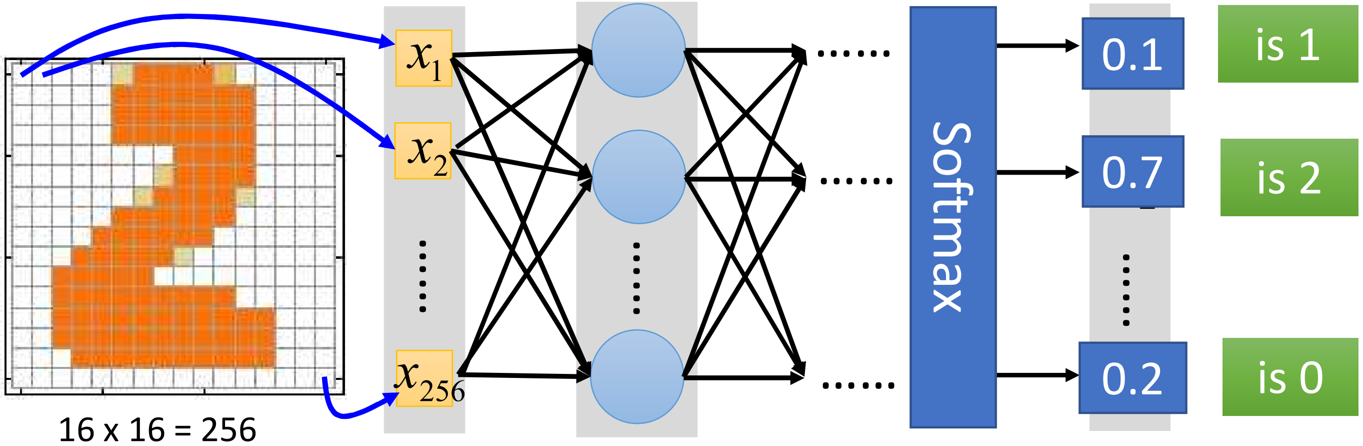
- Add momentum and/or velocity/acceleration terms
  - SGD with momentum
  - Adagrad
  - Adam
  - Adam

```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
  dx = # ... evaluate gradient
  m = beta1*m + (1-beta1)*dx # update first moment
  v = beta2*v + (1-beta2)*(dx**2) # update second moment
  mb = m/(1-beta1**t) # correct bias
  vb = v/(1-beta2**t) # correct bias
  x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```



Animation by Alec Radford

MNIST Digits Classification using MLP

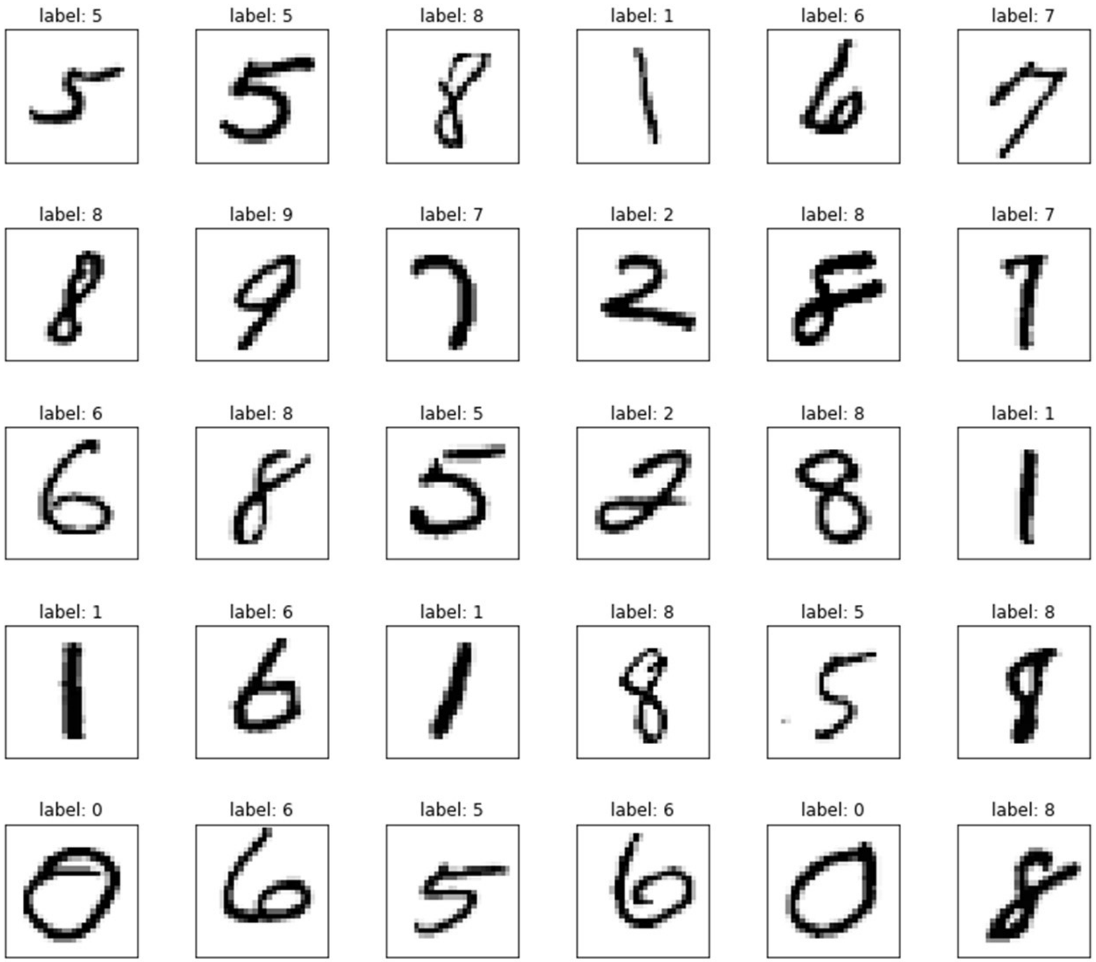


MNIST Dataset

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

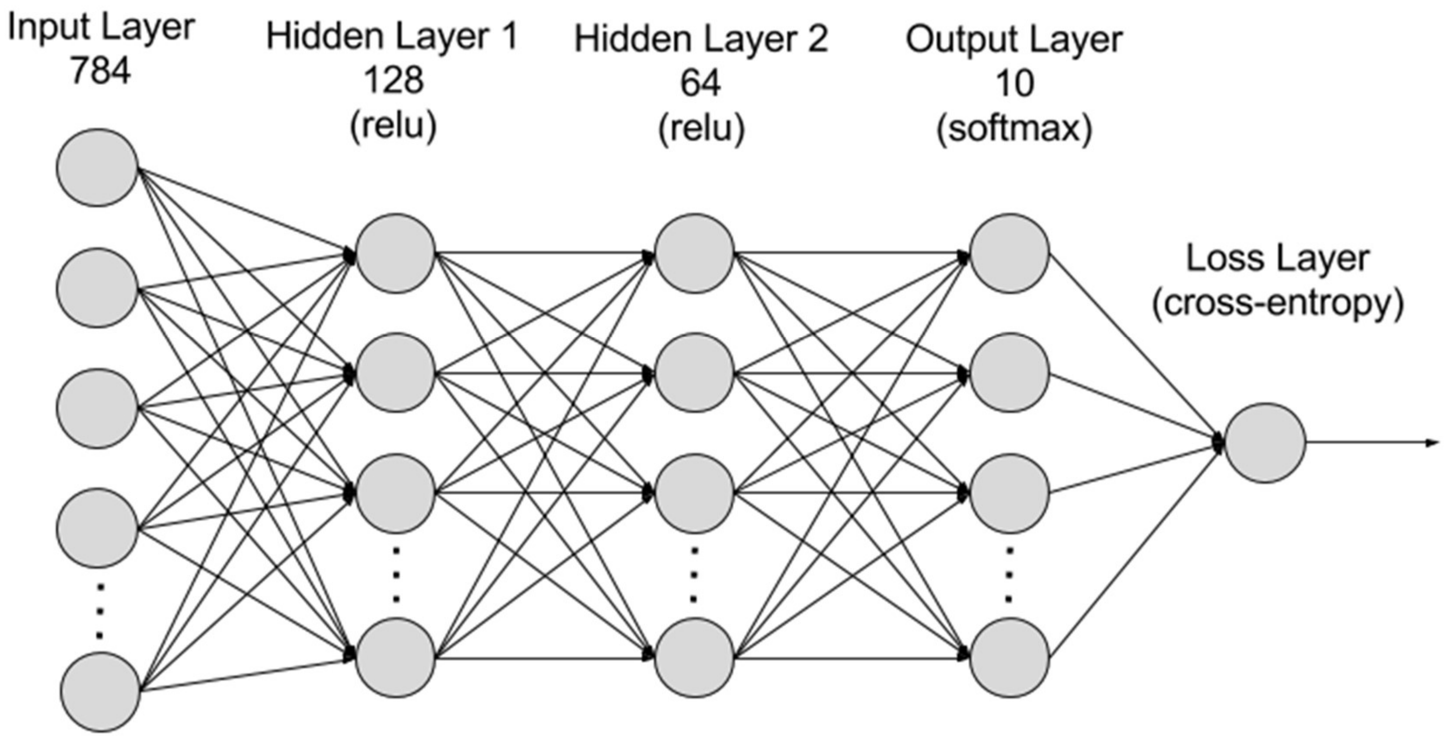
  - Image size: **28x28**
    - Pre-processed data(e.g., the digits are placed in the center of image).
    - Labels are provided.

- 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

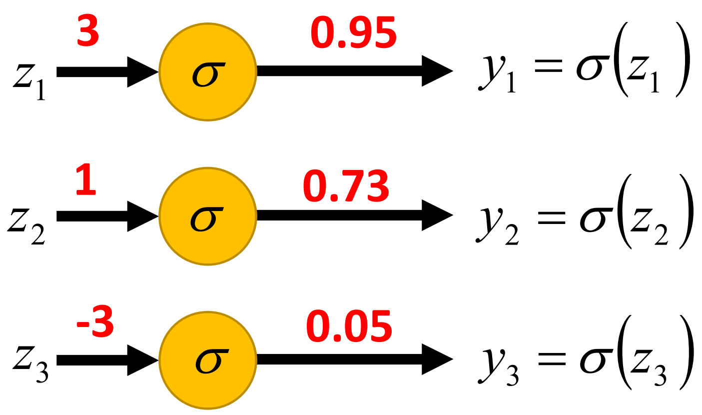


Softmax Layer

- 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:

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