Seoul
National
University

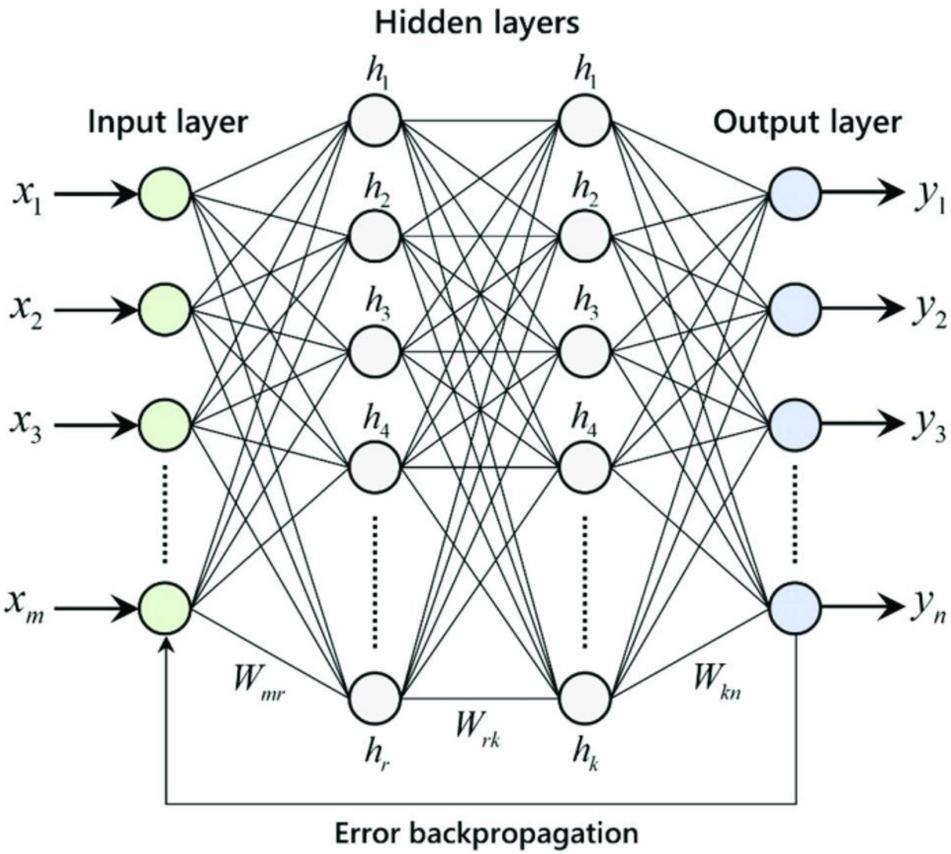
# Introduction to Deep Learning - part2

Taewook Ko

SCONE
Lab.

# Recap- Neural Network

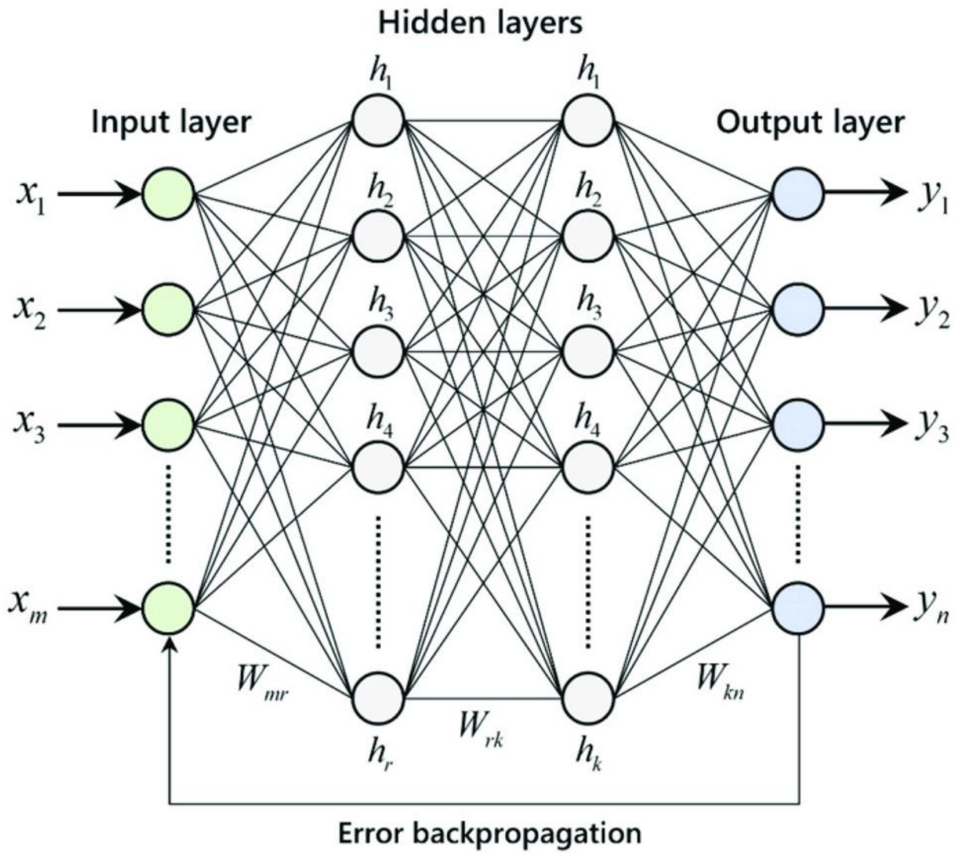
- Neural Network Structure
  - Neuron / Node
  - Layers
    - Input, Hidden, Output
  - Parameters
    - Weights
    - Bias



# Recap- Neural Network



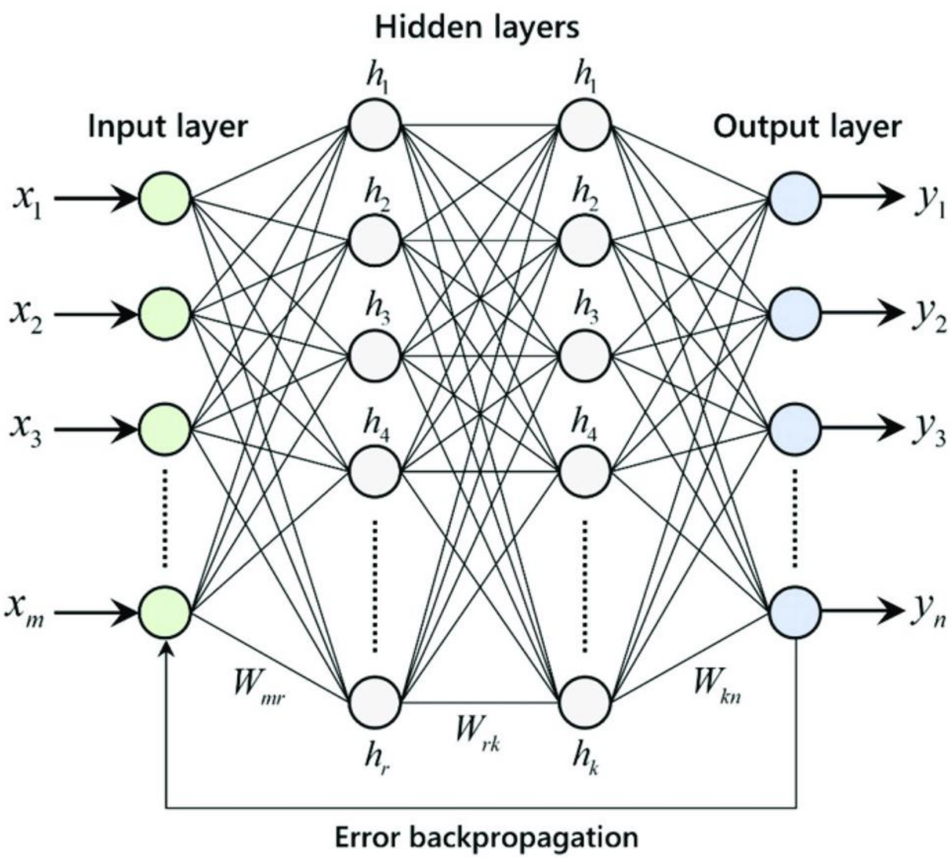
- Forward Propagation
  - $Input = X \in \mathbb{R}^{1 \times m}$
    - $m$ feature vector input
  - $H^1 = f(XW^1 + b^1) \in \mathbb{R}^{1 \times r}$
    - $W^1 \in \mathbb{R}^{m \times r}$, $b^1 \in \mathbb{R}^{1 \times r}$
  - $H^2 = f(H^1W^2 + b^2) \in \mathbb{R}^{1 \times k}$
    - $W^2 \in \mathbb{R}^{r \times k}$, $b^2 \in \mathbb{R}^{1 \times k}$
  - $Output = f(H^2W^O + b^O) \in \mathbb{R}^{1}$
    - $W^O \in \mathbb{R}^{k \times n}$, $b^O \in \mathbb{R}^{1 \times n}$
  - $f(\cdot)$ : Activation function
    - Sigmoid, Softmax, Tanh, ReLU and so on



# Recap- Neural Network



- Back Propagation
  - Initialization
    - Sampling $\theta \sim N(0, \sigma)$
  - Loss functions
    - $L(\hat{y}, y)$
    - MSE, Cross Entropy
  - Parameter Gradient
    - With Chain rule
  - Update Rule
    - SGD, RMSProp, Adam



# Recap- Neural Network



- Pseudo Code

**Algorithm 1** Neural Network Train

Define a model, $M$
Initialize parameters, $M_\theta$
**for** *Epochs* **do**
  **for** *mini batches* **do**
    $\hat{y} = M_\theta(x)$
    $Loss = \mathcal{L}(\hat{y}, y)$
    $\theta \leftarrow \theta - \alpha \times \frac{\partial Loss}{\partial \theta}$
  **end for**
**end for**

# Contents

- Data preparation
  - Data split
  - Data normalization
- Under- and Over-fitting
  - Underfitting
  - Overfitting
- Regularization
  - L1 and L2
  - Dropout
- Exercise with MNIST

# Data preparation

- Dataset
  - Deep learning is known to require a lot of data
  - Data is a set of instances
  - Instance
    - Tuple of (Input, Label)

(  , Dog) (  , Cat)

(  , Dog) (  , Cat)

⋮ ⋮

(  , Dog) (  , Cat)

# Data preparation



- Dataset
  - Expect the model learns hidden patterns of input
    - Some patterns which dog images commonly have
    - The model maps input to the output
      - Want the output similar to label
  - Not only input, but also label is required
    - Label is the right answer, also called ground-truth
    - Usually manually set the labels
      - Requires a lot of labor
      - Hurdles for applying the NN to real-world problems
    - But it is them most easiest way
      - Because we know the right answer
      - Call it supervised learning method

# Data preparation



- Split dataset
  - Train Dataset
    - Dataset for train the model
  - Validation Dataset
    - Dataset for check whether the training is going well
  - Test Dataset
    - Dataset for check the performance of the model

  - Split ratio
    - No theoretically best ratio
      - Train:Valid:Test = 80:10:10
      - Train:Valid:Test = 60:20:20

# Data preparation



- Split dataset
  - Purpose of the train and validation set
    - Train model
    - Learn hidden patterns of input
    - Learn how to map input to label
  - Purpose of the test set
    - Instances never seen during train/valid
    - If the model learn hidden patterns from train/valid input properly
      - The model can find the patterns with the new instances (test-set)
      - Check the performance of the trained model

# Data preparation

- Normalization
  - Data features have different scale
  - Parameters related with large scale feature are huge or tiny
  - Gradients will be dominated by some parameters
    - Training is not properly done for all features
  - Re-scaling data features

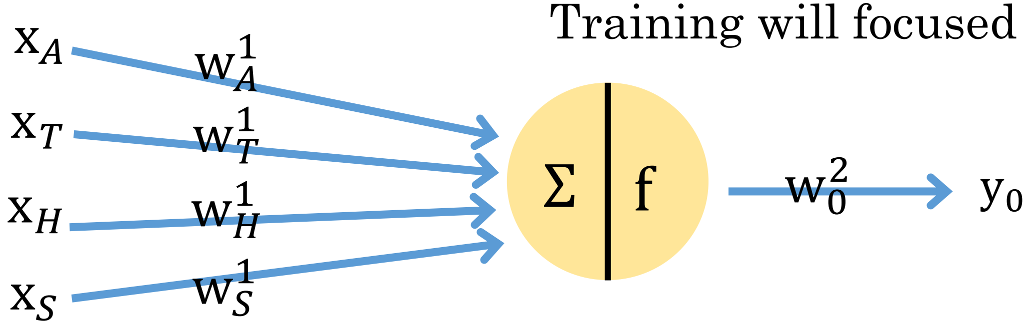
  - Examples.
    - Prediction batting average of a baseball player
    - Input of four features
      - [Age, Service Time, Height, Salary]
        - Age : 20~50 years old
        - Service Time : 0~20 years
        - Height : 160~200 cm
        - Salary : Hundreds of millions ₩

# Data preparation

- **Normalization**
  - For a simple neural network
    - Input = [Age, Service Time, Height, Salary]
    - $\mathrm{y_o} = \mathrm{w}_0^2 f(\mathrm{x}_A \mathrm{w}_A^1 + \mathrm{x}_T \mathrm{w}_T^1 + \mathrm{x}_H \mathrm{w}_H^1 + \mathrm{x}_S \mathrm{w}_S^1)$
      - $\frac{\partial y}{\partial w_T} = \frac{\partial y}{\partial \mathrm{w}_\mathrm{o}^2} \times \cdots \times \frac{\partial In_1}{\partial w_T^1}$
      - $\frac{\partial y}{\partial w_S} = \frac{\partial y}{\partial \mathrm{w}_\mathrm{o}^2} \times \cdots \times \frac{\partial In_1}{\partial w_S^1}$
    - $In_1 = \mathrm{x}_0 \mathrm{w}_A^1 + \mathrm{x}_1 \mathrm{w}_T^1 + \mathrm{x}_2 \mathrm{w}_H^1 + \mathrm{x}_3 \mathrm{w}_S^1$
      - $\frac{\partial In_1}{\partial w_S^1} >>> \frac{\partial In_1}{\partial w_\mathrm{A}^1}, \frac{\partial In_1}{\partial w_T^1}, \frac{\partial In_1}{\partial w_\mathrm{H}^1}$
      - $\because \mathrm{x}_S >>> \mathrm{x_A}, \mathrm{x}_T, \mathrm{x_H}$

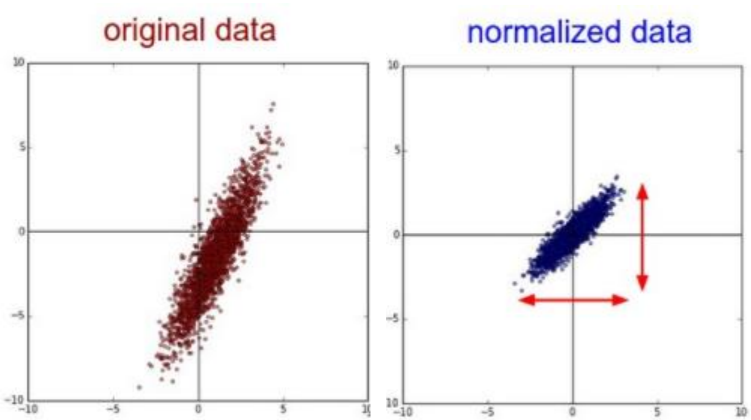
Gradients will be dominated by some parameters

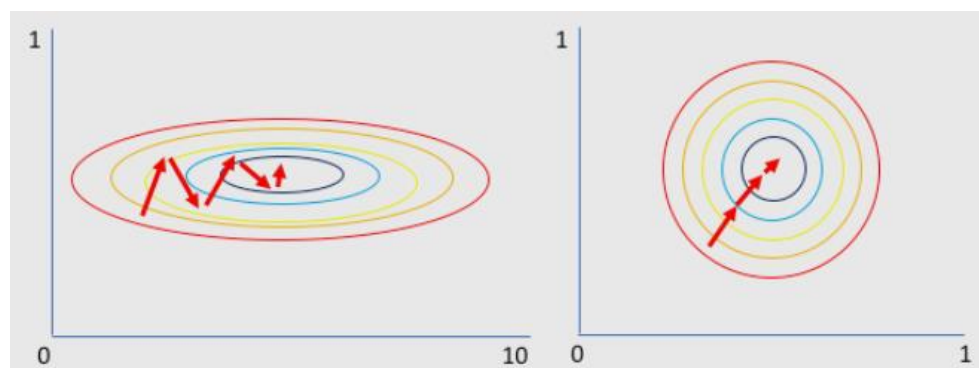
Training will focused on some features



# Data preparation

- Normalization
  - Min-Max normalization
    - $x_{norm} = \frac{x - Min}{Max - Min}$
    - Max, Min: maximum and minimum feature values
    - Convert maximum value to 1 and minimum value to 0
  - Z-score normalization
    - $x_{norm} = \frac{x - \mu}{\sigma}$
    - $\mu$, $\sigma$: mean and standard deviation of feature values



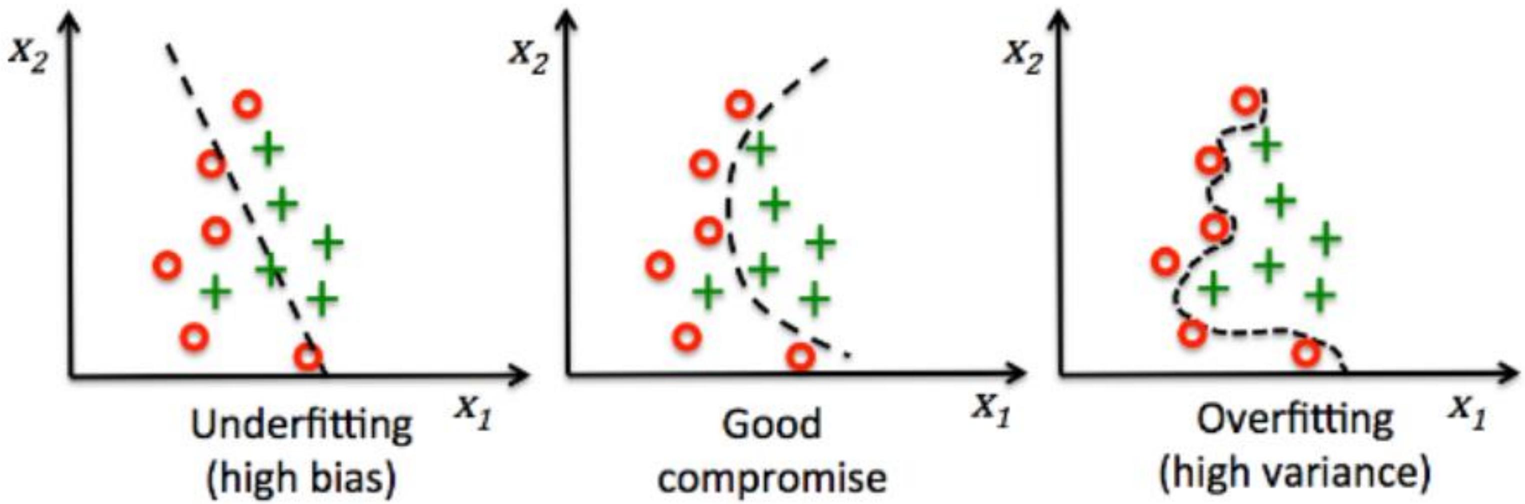




[1] Stanford cs231n lecture note

# Under- and Over-fitting



- During training process,
  - Parameters are updated and decision boundary are changing
- Overfitting and Underfitting
  - Underfitting : Not trained properly, with high bias
  - Overfitting:  Too much train, with high variance




[2] Andrew Ng, Coursera Machin Learning

# Under- and Over-fitting



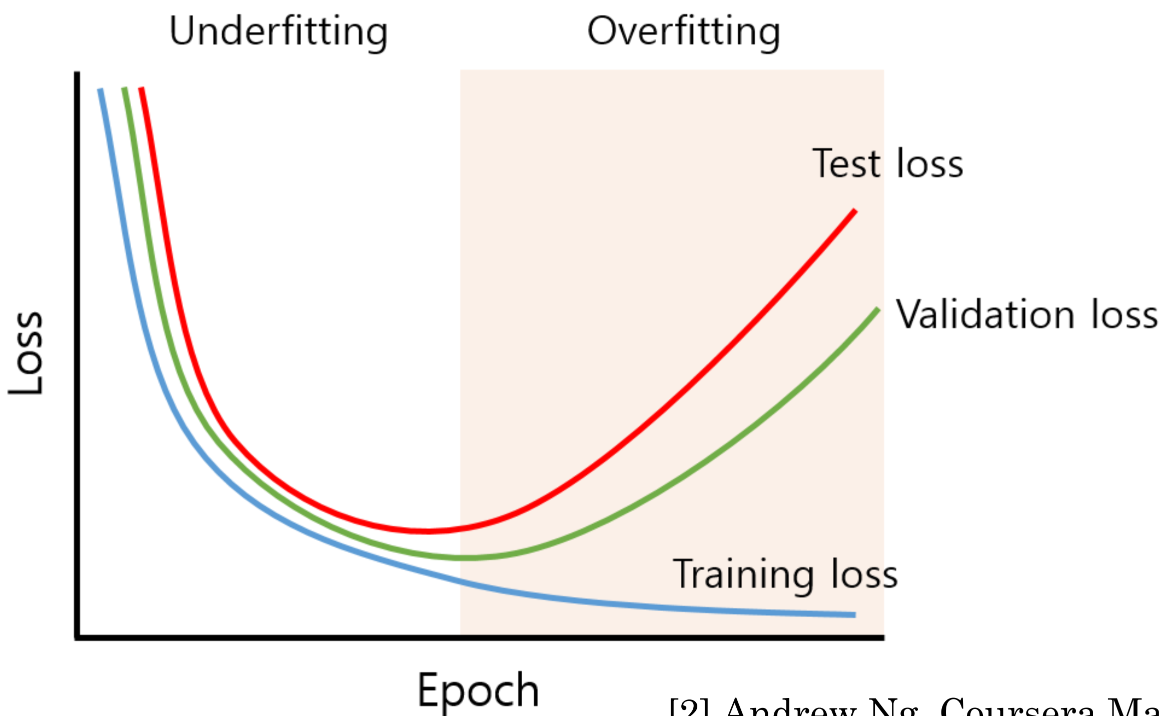
- Effect of parameter size
  - Each neuron contributes on a non-linear transformation
  - For a large size model
    - Means large number of layers or hidden neurons
    - Has more non-linear transformation process
    - Make the decision boundary more flexible

  - Adjusting model size is one of the solution to over- and under-fitting
    - For underfitting issue
      - Need the boundary more flexible
    - For overfitting issue
      - Need the boundary make flatter

# Under- and Over-fitting



- Early Stopping
  - Validation shows whether the train goes well
  - If boundary line goes too much flexible
    - The model starts overfitting
    - Validation loss will goes up



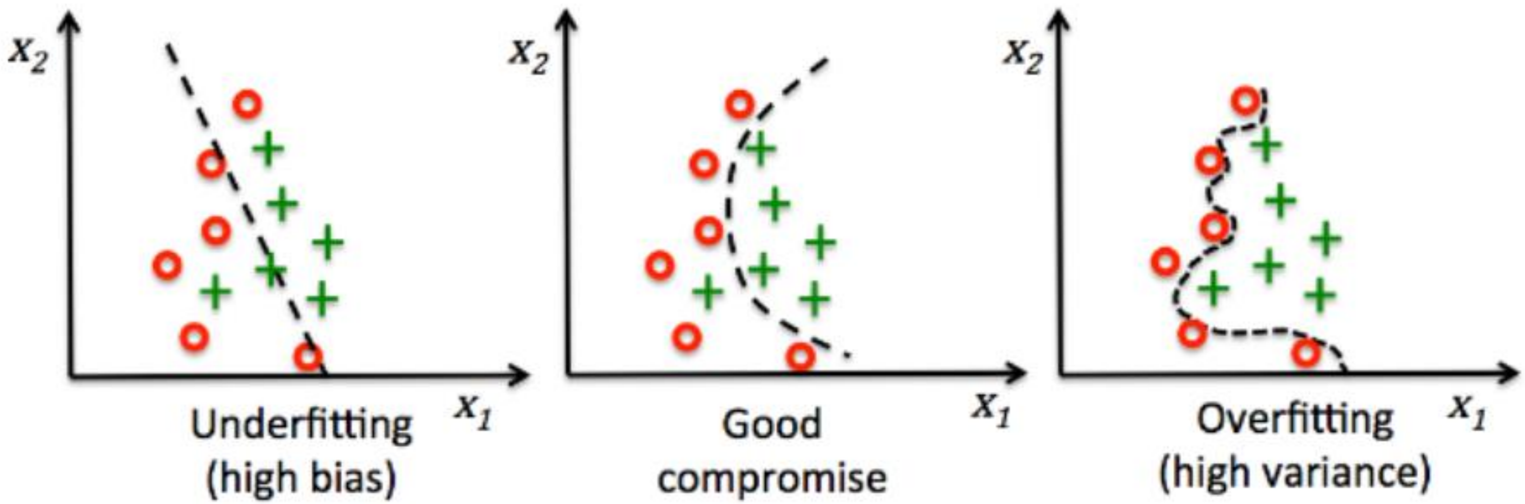

[2] Andrew Ng, Coursera Machin Learning

# Under- and Over-fitting

- What does too much flexible mean?
- What is good compromise boundary line?
  - As training runs,
    - The boundary line changes and goes flexible
  - At some point,
    - The boundary line too much fit to the train dataset
    - They overly flexed to fit the train dataset
  - If the flexibility is also good for validation or test dataset
    - In other words, both train loss and valid loss goes down
    - It is not overfitting, it is under training for general data
  - But the flexibility is only good for train dataset
    - In other words, train loss goes down but valid loss goes up
    - It is overfitting, it is under training only for train data not for general
    - They learn 'noise data' of train dataset // Stop training

# Underfitting

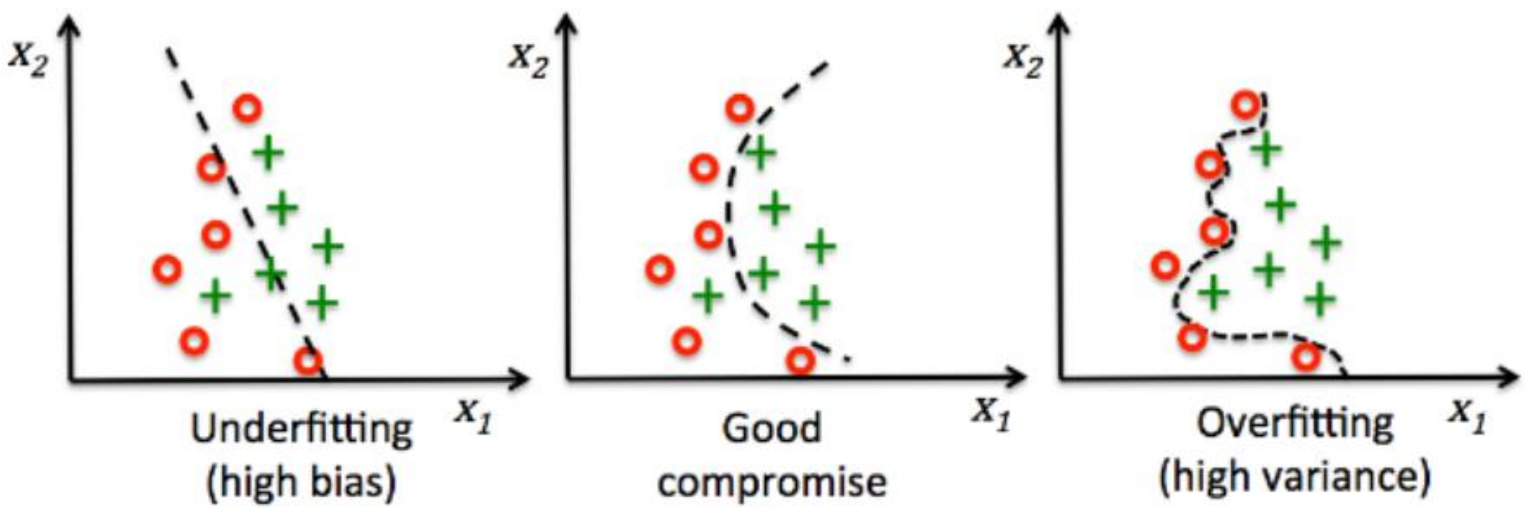
- Decision boundary is too flat
  - Make the boundary flexible
    - Increase training epoch
    - Increase the parameter size (model size)
      - The number of layers or hidden size




[2] Andrew Ng, Coursera Machin Learning

# Overfitting

- Decision curve is to flexible
  - Make the boundary flat
    - Reduce training epoch or apply early stop
    - Reduce the parameter size




[2] Andrew Ng, Coursera Machin Learning

# Regularization



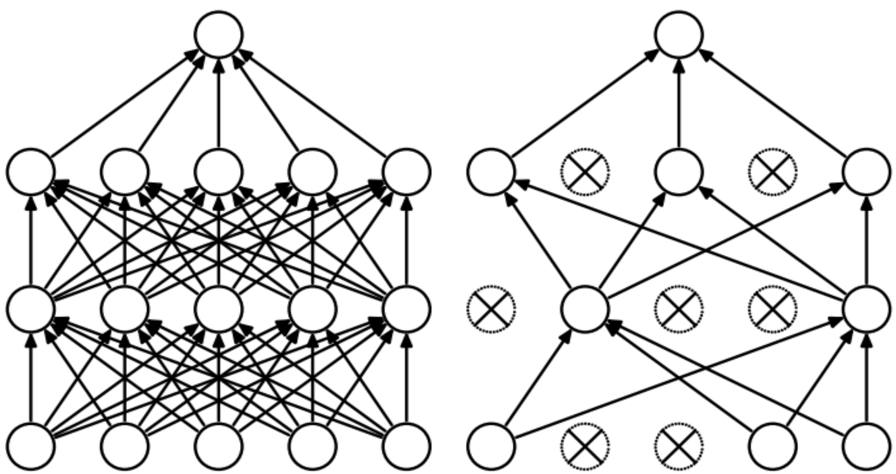
- L1 and L2
  - Add penalty on loss function when parameter value is high
  - Loss $= \mathrm{L}(\hat{y} - y) + \lambda|\theta|$
    - $\mathrm{L}_1$ regularization
      - $\theta = \sum |\text{parameters}|$
    - $\mathrm{L}_2$ regularization
      - $\theta = \sum |\text{parameters}|^2$
    - $\lambda$: regularize weight
      - $0 < \lambda < 1$
  - Avoid having high parameter values
    - Reduce the flexibility of decision boundary
    - Can avoid overfitting problem without reducing model size

[3] Ng, Andrew, 2004. ICML

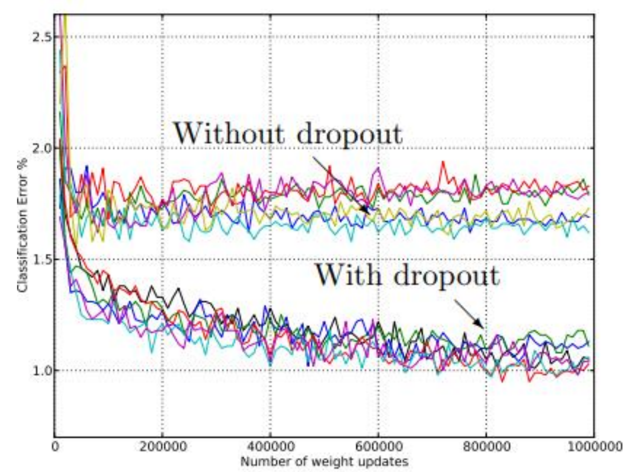
# Regularization



- Dropout
  - Randomly drops (inactivates) nodes for an iteration
    - Drop node
      - The weights related the dropped nodes are not contributes to forward/back propagation
      - No parameter updates
      - Prevent overly train on specific features
        - Learn on overall features



(a) Standard Neural Net (b) After applying dropout.




[4]Srivastava, N., Hinton, G. ,2014. JMLR

# Regularization

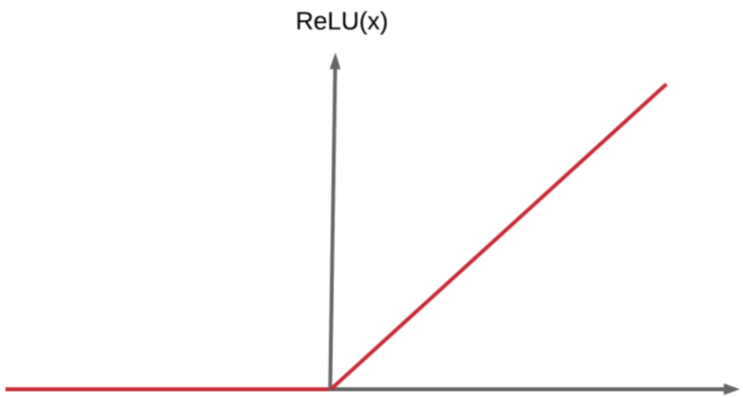


- Goal of regularization
  - Make the model utilizes as many parameters as possible
  - L1 and L2
    - Instead having high values for some parameters
    - Let multiple small valued parameters to replace a single large parameter
    - Avoid training on specific features
    - Avoid decision boundary too flexible
  - Dropout
    - Training is easy to focus on some specific features and parameters
    - Randomly drop some nodes, force to learn on different features for different iteration
      - Let sparse learning
    - In the beginning of the training, dropout may be detrimental
      - But it is helpful after some training has progressed.

# Regularization



- Another reason why ReLU is good
  - ReLU
    - $R(x) = \max(0, x)$
    - $R'(x) = 0\ or\ 1$
  - ReLU for negative input is 0
    - 0 output, no parameter update
    - Makes some neurons die
    - Called 'Dying ReLU problem'
    - Similar effect of Dropout
      - Sparse learning
  - ReLU
    - Reduce gradient vanishing issue
    - Easy to compute gradients
    - Give sparsity on dense hidden layer



# Exercise



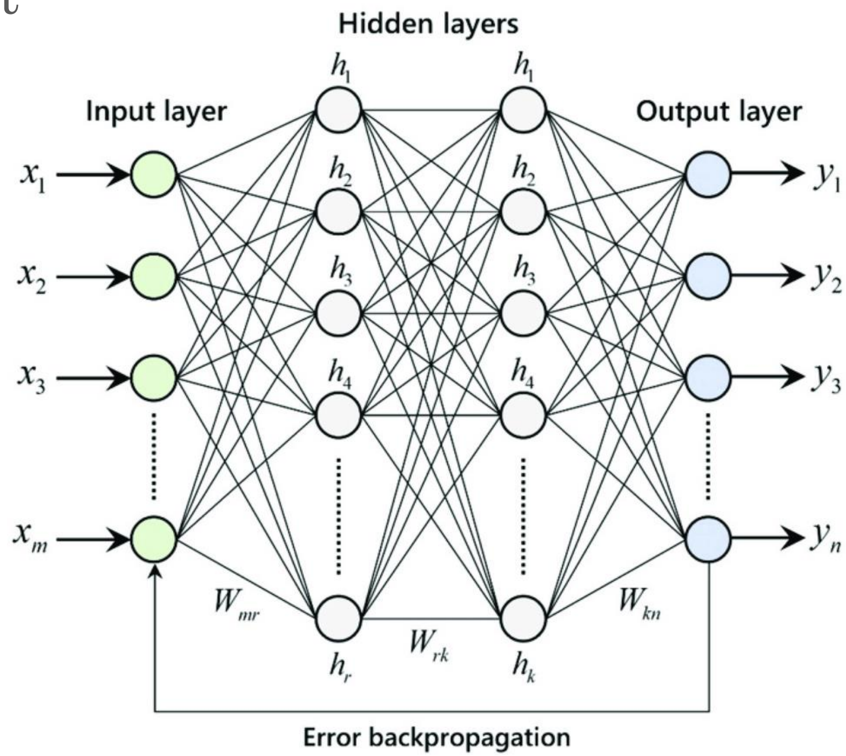
- Data processing
- Neural network modeling
- Training and testing

- MNIST Dataset
  - Handwritten digits dataset
    - 60,000 images for train
    - 10,000 images for test
    - 28x28 image
  - Images are labeled

[5] http://yann.lecun.com/exdb/mnist/

# Exercise

- Design input and output
  - Input image is 28x28
  - But NN takes $1-D$ vector as input
    - $Input = X \in \mathbb{R}^{1 \times m}$
    - Make the images flatten
      - $28 \times 28 \rightarrow 1 \times 784$
      - Our input layer size should be
        - $1 \times 784$
      - 784 featured vector input



# Exercise



- Design input and output
  - Output
    - Goal : Predict the number
    - Ten possible outputs *0~ 9*
    - There are two possible output shape
      - One hot vector shape
        - shape of $1 \times 10$
        - Each elements values indicates probability
      - A scalar shape
        - Shape of 1
        - The output directly shows the number

    - Which output shape will be better?

# Exercise

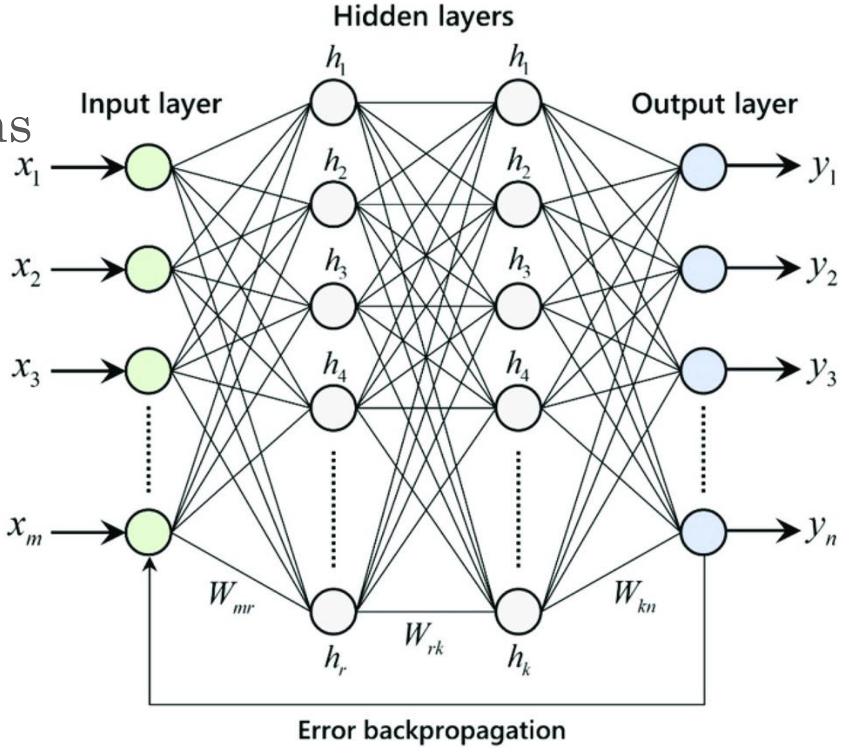
- Design input and output
  - Let's say if a model is confusing whether a image is '3' or '9'
    - If the output is one hot vector shape
      - The output will be

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.01 | 0.02 | 0.45 | 0 | 0 | 0.02 | 0 | 0 | 0.50 |

      - And the model predicts it as '9' (highest probability)
      - It may wrong, but the model will reduce the loss by update rule
      - If the label is '3' then increase probability of '3' and reduce '9'
      - The model will make right prediction '3' in the next iteration
    - However if the output is scalar shape
      - The output will be
        - (3+9)/2 = 6
      - And the model predicts it as '6'
      - Even the update rule works properly
      - The model will make prediction '6' again
        - Still confusing between '3' and '9'

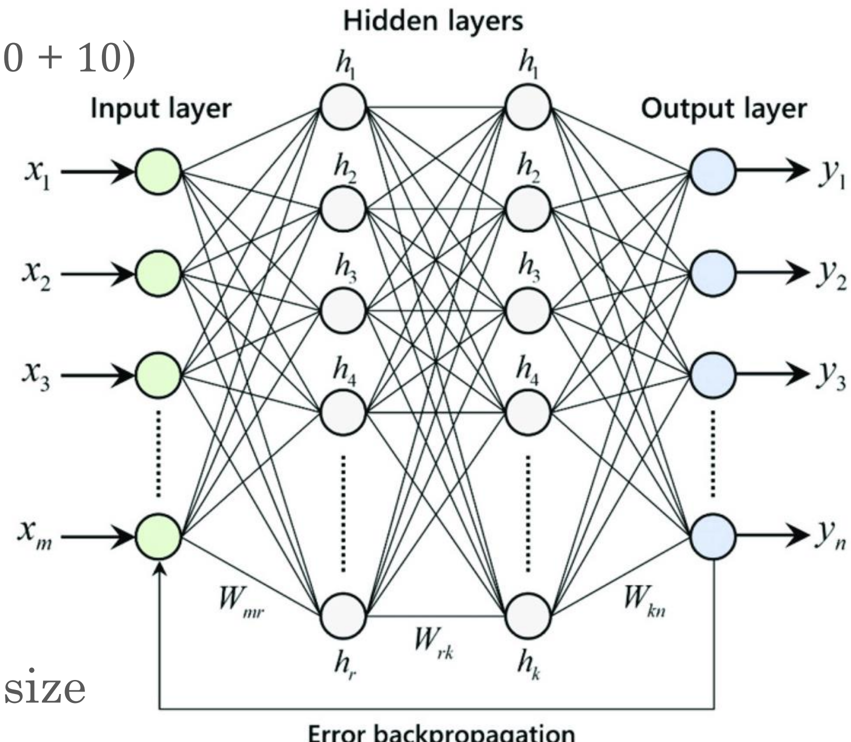
# Exercise

- Design hidden layers
  - First hidden layer with $r$ neurons
    - Parameters for the first layer
      - $W^1 \in \mathbb{R}^{784\times r}$, $b^1 \in \mathbb{R}^{1\times r}$
  - Second hidden layer with $k$ neurons
    - Parameters for the second layer
      - $W^2 \in \mathbb{R}^{r\times k}$, $b^2 \in \mathbb{R}^{1\times k}$
  - Output Parameters
    - $W^O \in \mathbb{R}^{k\times 10}$, $b^o \in \mathbb{R}^{1\times 10}$



# Exercise

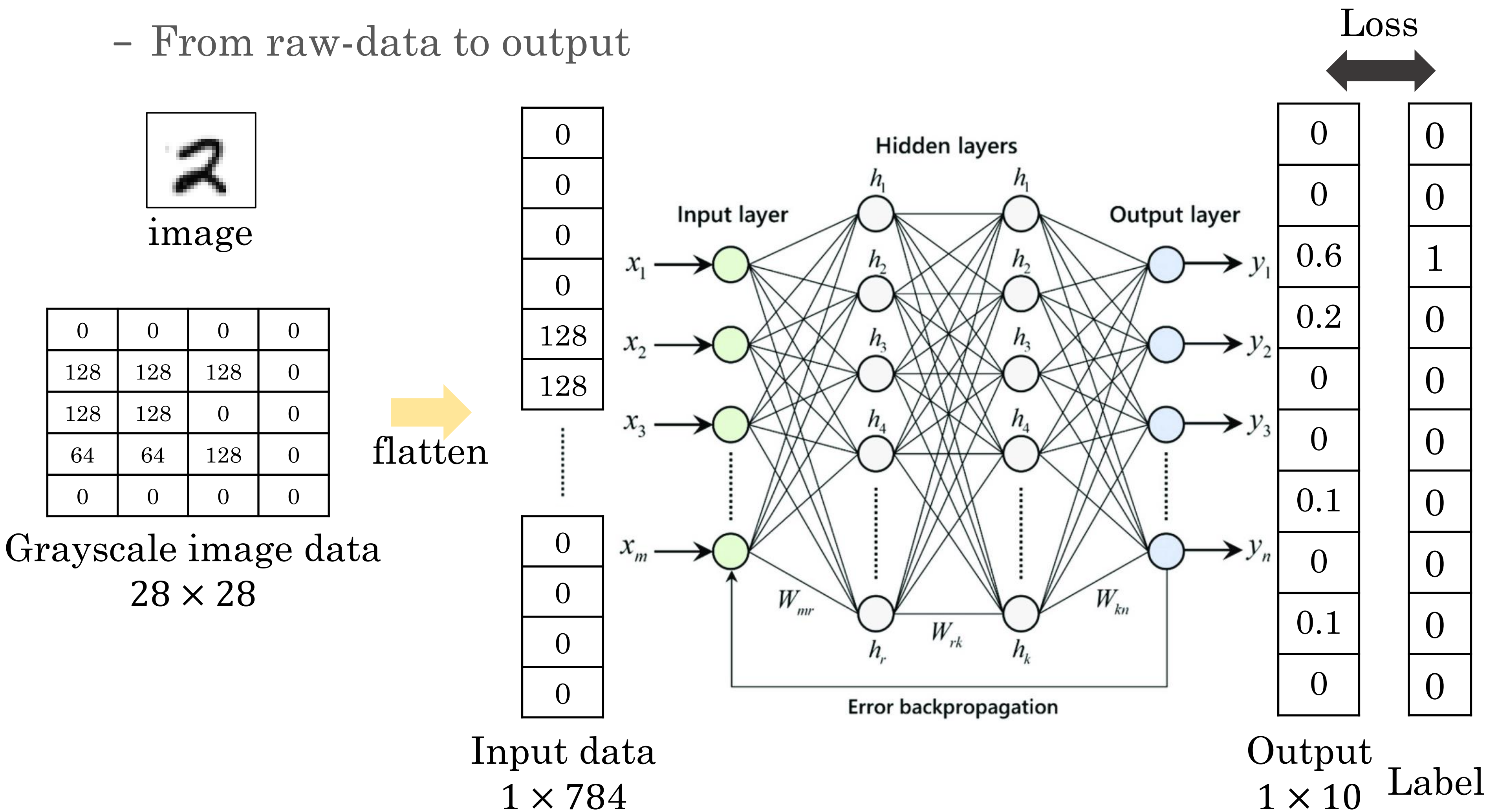
- Design hidden layers
  - For the three layer NN
    - The number of parameters
      - $(784 \times r + r) + (r \times k + k) + (k \times 10 + 10)$
  - When r, k = 512, 512
    - 670k parameters
  - When r, k = 512, 256
    - 530k parameters
  - When r, k = 256, 128
    - 240k parameters
  - There is no right answer
    - For the # of layers or # of hidden size
    - Smoothly increase layers and nodes
    - Depends on the dataset size, over- under- fitting issue, data complexity, model structure, and other model components



# Exercise

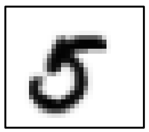


- Model overview
  - From raw-data to output



# Exercise

- Model overview
  - From raw-data to output



image

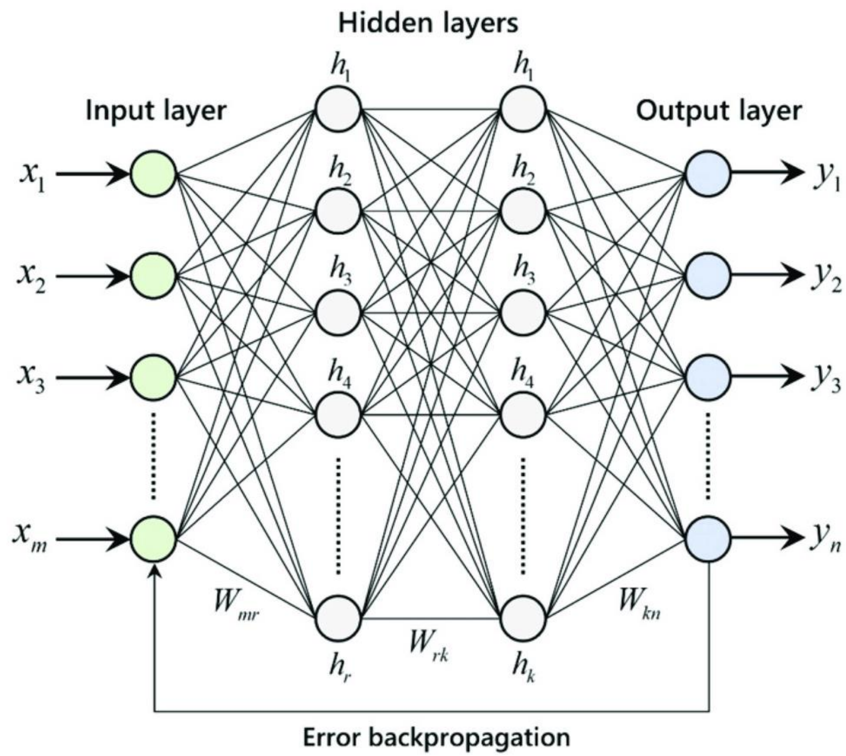
| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 128 | 128 | 128 |
| 0 | 128 | 64 | 0 |
| 64 | 64 | 128 | 0 |
| 0 | 32 | 32 | 0 |

Grayscale image data
$28 \times 28$

flatten

| |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 128 |
| ⋮ |
| 0 |
| 32 |
| 32 |
| 0 |

Input data
$1 \times 784$




Loss

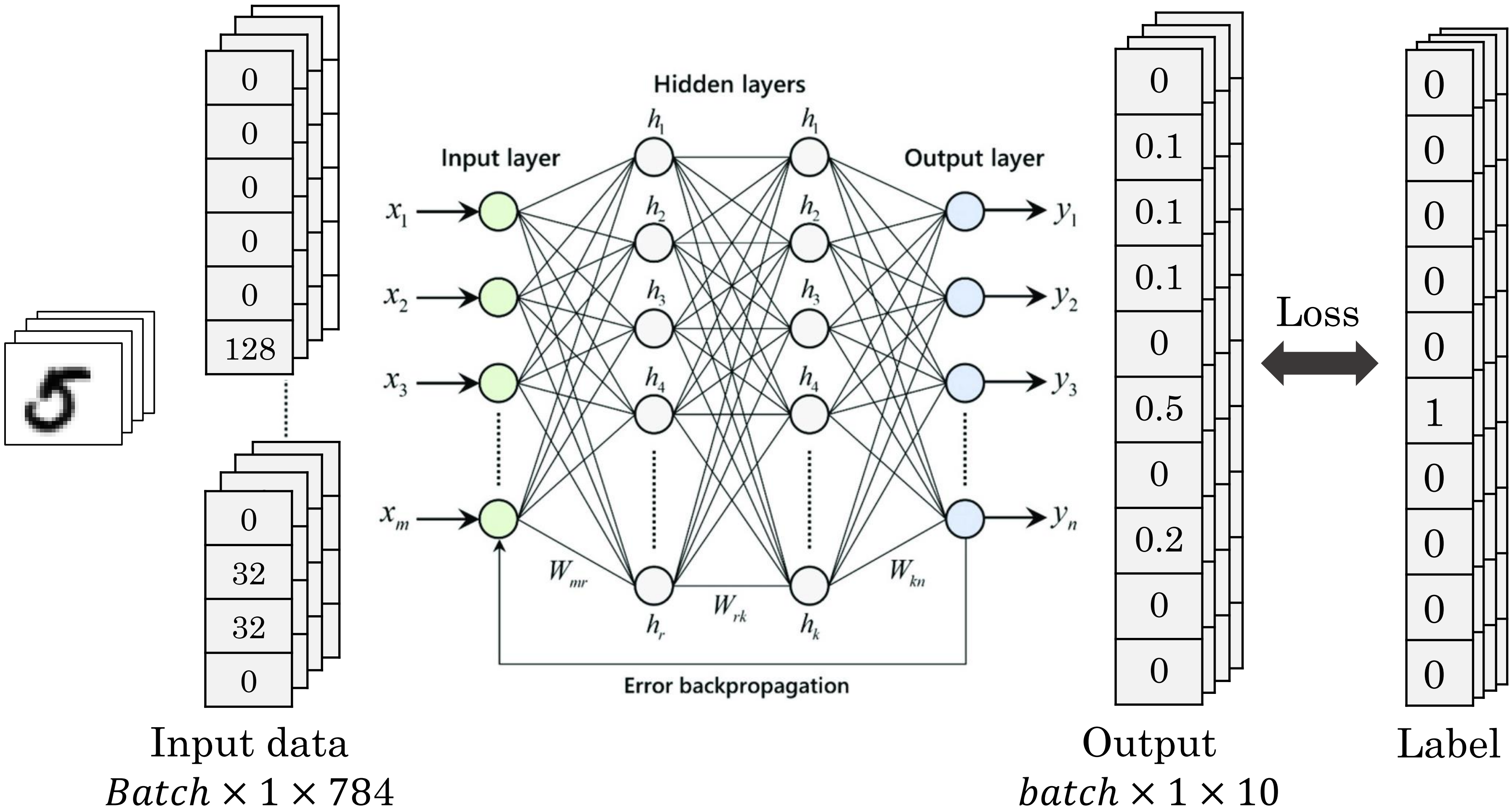
| Output | Label |
|---|---|
| 0 | 0 |
| 0.1 | 0 |
| 0.1 | 0 |
| 0.1 | 0 |
| 0 | 0 |
| 0.5 | 1 |
| 0 | 0 |
| 0.2 | 0 |
| 0 | 0 |
| 0 | 0 |

Output
$1 \times 10$

Label

# Exercise

- Model overview
  - Mini-batch learning



# Exercise

- Implementation examples
  - Split dataset
    - Train/Validation/Test = 50,000/10,000/10,000
    - Spare 10k samples for validation from train data
  - Hidden layers and neuron size
    - Three layer and $r, k$ = 512, 256
  - Data normalization
    - Min-Max normalizer
  - Batch-size
    - 64
  - Parameter Initialization
    - Xavier initialization
  - Activation function
    - ReLU for hidden layer, Softmax for output layer

# Exercise



- Implementation examples
  - Loss function
    - Cross-entropy + L1 regularization
    - With $\lambda = 0.001$
  - Dropout
    - With 10% drop rate
  - Update Rule
    - ADAM Optimizer
  - Learning rate
    - $\alpha = 0.01$
  - Number of epochs
    - Early stop
    - If there is no validation loss reduction during ten iterations

# Exercise



- MNIST prediction with MLP

| Model | # of Layers | # of Hidden Size | Loss function | Accuracy |
|---|---|---|---|---|
| 1 | 2 | 300 | MSE | 95.3% |
| 2 | 2 | 100 | MSE | 95.5% |
| 3 | 3 | 300, 100 | MSE | 96.95% |
| 4 | 3 | 500, 150 | MSE | 97.05% |
| 5 | 2 | 800 | Cross Entropy | 98.4% |
| 6 | 3 | 500, 300 | Cross Entropy + L1 | 98.47% |
| 7 | 6 | 2500, 2000, 1500, 1000, 500 | Cross Entropy + L1 | 99.65% |

[5] http://yann.lecun.com/exdb/mnist/

# Reference



[1] http://cs231n.stanford.edu/

[2] https://www.coursera.org/learn/machine-learning\

[3] Ng, Andrew Y. "Feature selection, L 1 vs. L 2 regularization, and rotational invariance." *Proceedings of the twenty-first international conference on Machine learning*. 2004.

[4] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15.1 (2014): 1929-1958.

[5] http://yann.lecun.com/exdb/mnist/