

Optimization for Deep Learning

Lecture 2: Basics in Machine Learning

Kun Yuan

Peking University

Main contents in this lecture

- Linear regression
- Logistic regression
- Multi-classification
- Neural network

Motivation

- You consider renting an apartment
- You don't know whether the price the agent offered is good or not
- You collect a dataset

Table: Collected dataset

Size (x_1)	Location (x_2)	Green rate (x_3)	Decoration (x_4)	Price (y)
80 m ²	8	20%	6	10000
60 m ²	10	30%	8	9000
100 m ²	5	20%	5	9000
⋮	⋮	⋮	⋮	⋮
70 m ²	10	25%	9	12000

Motivation

- Below is your target apartment's description. What should be the reasonable price for this apartment?

Table: Your target apartment

Size (x_1)	Location (x_2)	Green rate (x_3)	Decoration (x_4)	Price (y)
100 m ²	8	35%	8	?

- You need to learn how (x_1, x_2, x_3, x_4) will map to y from your dataset
- This is a typical task in machine learning: linear regression

Linear regression

- Consider a set of data $\{(x_i, y_i)\}_{i=1}^N$ where

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id}) \in \mathbb{R}^d$$

is the feature vector, e.g., x_{i1} = "Size" and x_{i2} = "Location", etc., and y is the label, e.g., y = "Price"

- We assume the mapping between \mathbf{x}_i and y_i is in the **linear** form

$$y_i \approx \mathbf{x}_i^\top \mathbf{w} \tag{1}$$

where $\mathbf{w} \in \mathbb{R}^d$ is the unknown parameter to learn

- If the parameter \mathbf{w} is known, given a new feature vector \mathbf{x} (e.g., the data for your target apartment), you can estimate its label y according to (1)

Linear regression

- How to get the parameter w ? We can calculate it with $\{(x_i, y_i)\}_{i=1}^N$
- A good w will incur the minimum estimation error

$$w^* = \arg \min_{w \in \mathbb{R}^d} \left\{ \frac{1}{2N} \sum_{i=1}^N (x_i^\top w - y_i)^2 \right\} \quad (2)$$

where is called the linear regression problem

- If we introduce

$$X = [x_1^\top; \dots; x_N^\top] \in \mathbb{R}^{N \times d} \quad y = [y_1; y_2; \dots; y_N] \in \mathbb{R}^N$$

problem (2) becomes

$$w^* = \arg \min_{w \in \mathbb{R}^d} \left\{ \frac{1}{2} \|Xw - y\|^2 \right\}$$

Solve the linear regression problem

- Consider the linear regression problem

$$\mathbf{w}^* = \arg \min_{\mathbf{w} \in \mathbb{R}^d} \left\{ \frac{1}{2} \|X\mathbf{w} - y\|^2 \right\}$$

- Let $f(\mathbf{w}) = \frac{1}{2} \|X\mathbf{w} - y\|^2$, the gradient is given by

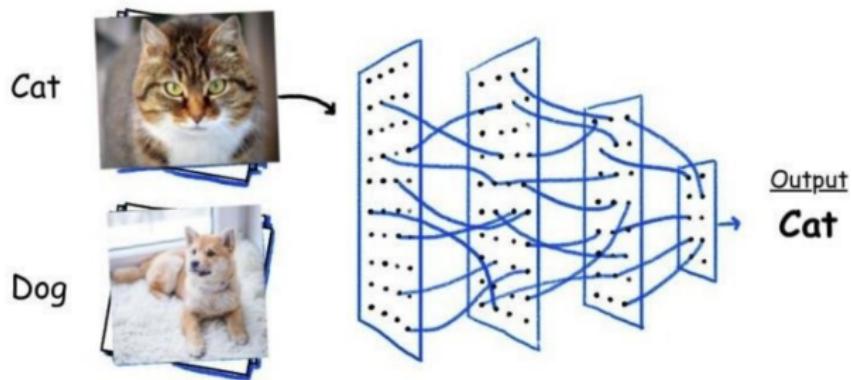
$$\nabla f(\mathbf{w}) = X^\top (X\mathbf{w} - y)$$

- The gradient descent is

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \gamma X^\top (X\mathbf{w}_k - y)$$

Logistic regression

- Another important machine learning task is classification



Logistic regression

- Again, we collect the dataset

Size (x_1)	Ear shape (x_2)	Tail length (x_3)	Color (x_4)	Label (y)
100 cm	round	30cm	yellow	dog
40 cm	triangle	20cm	white	cat
:	:	:	:	:

- We need to establish the mapping between (x_1, x_2, x_3, x_4) and the discrete label $y \in \{0, 1\}$ in which 1 indicates dog while 0 indicates cat

An intuitive approach

- We associate each feature item x_i with a weight w_i
- An intuitive hard classification approach is

$$(x_1, x_2, \dots, x_d) \longrightarrow y = \begin{cases} 1 & \text{if } \sum_{i=1}^d x_i w_i > c \\ 0 & \text{otherwise} \end{cases}$$

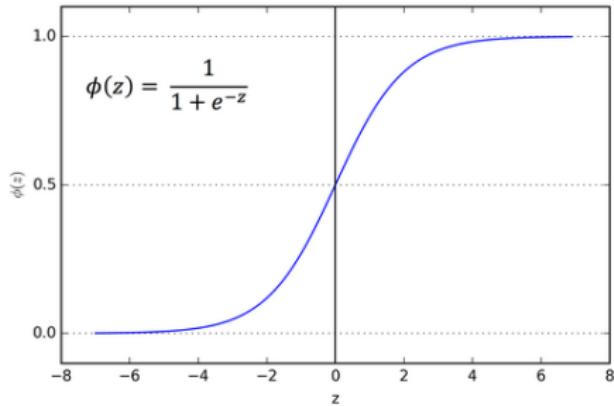
where c is a pre-defined threshold

- While intuitive, it is hard to construct smooth loss functions that facilitate to learn the weights (parameters) $\{w_i\}_{i=1}^d$

Sigmoid function

- Now we consider a different approach
- Sigmoid function maps $[-\infty, +\infty]$ to $[0, 1]$

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



Predicted probability

- With sigmoid function, we can map (x_1, \dots, x_d) to a probability

$$p(z) = \frac{1}{1 + e^{-z}} \in (0, 1) \quad \text{where} \quad z = \sum_{i=1}^d w_i x_i \quad (3)$$

- With (3), we map (x_1, \dots, x_d) to a probability distribution

$$(x_1, \dots, x_d) \longrightarrow \begin{bmatrix} p(z) \\ 1 - p(z) \end{bmatrix} \in \mathbb{R}^2$$

where $p(z)$ is the probability that (x_1, \dots, x_d) belongs to class 1

Real probability

- Given the label y , the real probability distribution is

$$\begin{bmatrix} y \\ 1 - y \end{bmatrix} \in \mathbb{R}^2$$

where label $y \in \{0, 1\}$ can be regarded as the probability of class 1

- We need to measure the difference between

$$(\text{Predicted prob.}) \quad \begin{bmatrix} p(z) \\ 1 - p(z) \end{bmatrix} \quad \text{and} \quad (\text{Real prob.}) \quad \begin{bmatrix} y \\ 1 - y \end{bmatrix}$$

Cross entropy

- Cross entropy can measure the difference between two probability distributions $p \in \mathbb{R}^d$ and $q \in \mathbb{R}^d$

$$H(p, q) = - \sum_{i=1}^d p_i \log(q_i)$$

Smaller cross entropy indicates smaller difference between p and q .

- Examples:

$$p = (1, 0, 0, 0) \quad q = (0.25, 0.25, 0.25, 0.25) \quad \rightarrow \quad H(p, q) = 2$$

$$p = (1, 0, 0, 0) \quad q = (0.91, 0.03, 0.03, 0.03) \quad \rightarrow \quad H(p, q) = 0.136$$

Loss function

- Given a data pair (\mathbf{x}, y) where $\mathbf{x} \in \mathbb{R}^d$ is the feature vector and $y \in \{0, 1\}$ is the label. Using the sigmoid function, we can predict the probability:

$$\begin{bmatrix} \frac{1}{1+\exp(-\mathbf{x}^\top \mathbf{w})} \\ \frac{\exp(-\mathbf{x}^\top \mathbf{w})}{1+\exp(-\mathbf{x}^\top \mathbf{w})} \end{bmatrix} \in \mathbb{R}^2$$

- The difference between the predicted and real probability is given by

$$\ell(\mathbf{x}, y; \mathbf{w}) = -y \log\left(\frac{1}{1 + \exp(-\mathbf{x}^\top \mathbf{w})}\right) - (1-y)\log\left(\frac{\exp(-\mathbf{x}^\top \mathbf{w})}{1 + \exp(-\mathbf{x}^\top \mathbf{w})}\right) \quad (4)$$

- Given the dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the loss function is to measure the averaged difference

$$L(\{(\mathbf{x}_i, y_i)\}_{i=1}^N; \mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{x}_i, y_i; \mathbf{w})$$

where $\ell(\mathbf{x}_i, y_i; \mathbf{w})$ is in (4).

Logistic regression

- By solving the following optimization problem

$$\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{x}_i, y_i; \mathbf{w}) \quad (5)$$

where $\ell(\mathbf{x}_i, y_i; \mathbf{w})$ is defined as

$$\ell(\mathbf{x}_i, y_i; \mathbf{w}) = -y_i \log \left(\frac{1}{1 + \exp(-\mathbf{x}_i^\top \mathbf{w})} \right) - (1 - y_i) \log \left(\frac{\exp(-\mathbf{x}_i^\top \mathbf{w})}{1 + \exp(-\mathbf{x}_i^\top \mathbf{w})} \right),$$

we can achieve the model parameters \mathbf{w}^* .

- Given \mathbf{w}^* and a new feature vector \mathbf{x} , we can decide its label by

$$y = \begin{cases} 1 & \text{if } p \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \quad \text{where} \quad p = \frac{1}{1 + \exp(-\mathbf{x}^\top \mathbf{w})}$$

Logistic regression: simplified loss

- The loss in (4) can be written as

$$\ell(\mathbf{x}, y; \mathbf{w}) = \begin{cases} \log(1 + \exp(-\mathbf{x}^\top \mathbf{w})) & \text{if } y = 1 \\ \log(1 + \exp(\mathbf{x}^\top \mathbf{w})) & \text{if } y = 0 \end{cases} \quad (6)$$

- If we modify the label as follows:

$$y \leftarrow \begin{cases} 1 & \text{if } y = 1 \\ -1 & \text{if } y = 0 \end{cases}$$

the loss in (6) becomes

$$\ell(\mathbf{x}, y; \mathbf{w}) = \log(1 + \exp(-y \mathbf{x}^\top \mathbf{w})) \quad (7)$$

Logistic regression: simplified loss

- Substituting (7) to (5), logistic regression becomes

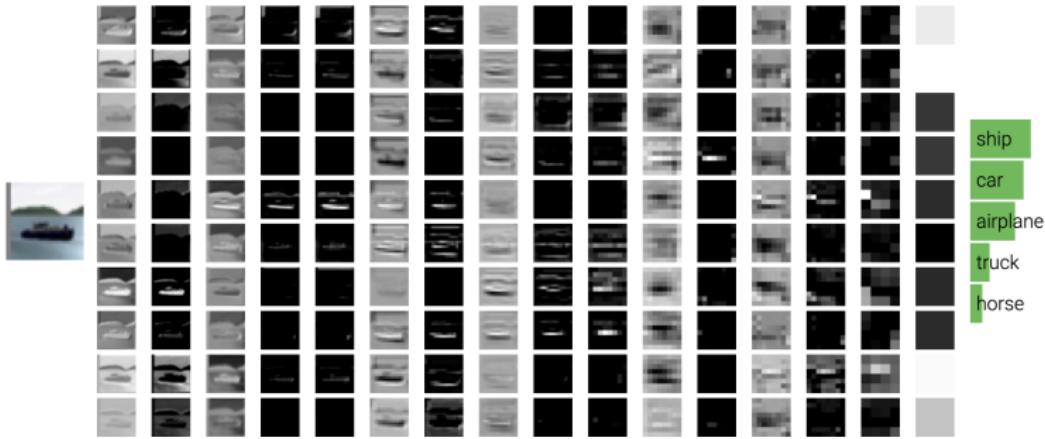
$$\min_{\mathbf{w} \in \mathbb{R}^d} \quad \frac{1}{N} \sum_{i=1}^N \ln(1 + \exp(-y_i \mathbf{x}_i^\top \mathbf{w}))$$

where $y \in \{+1, -1\}$ is the modified label

- Exercise: the gradient descent recursion to solve the above problem

Multi-classification

- Multi-classification is more common in real life



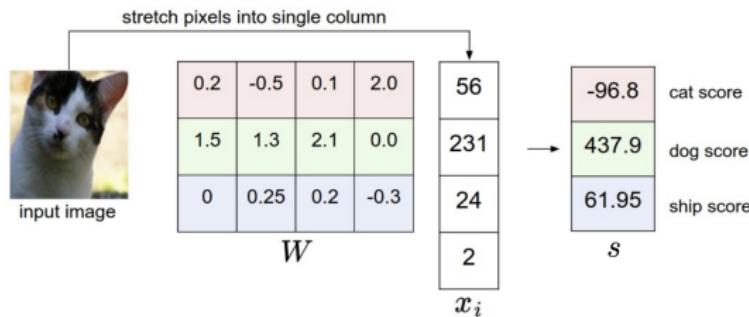
- Face recognition is one of the most successful multi-classification tasks

Multi-classification

- We first collect the dataset $\{\mathbf{x}_i, y_i\}_{i=1}^N$ where $\mathbf{x}_i \in \mathbb{R}^d$ is the feature vector and $y_i \in \{1, \dots, C\}$
- We consider the linear model to predict the score of each class

$$s = W\mathbf{x} \in \mathbb{R}^C$$

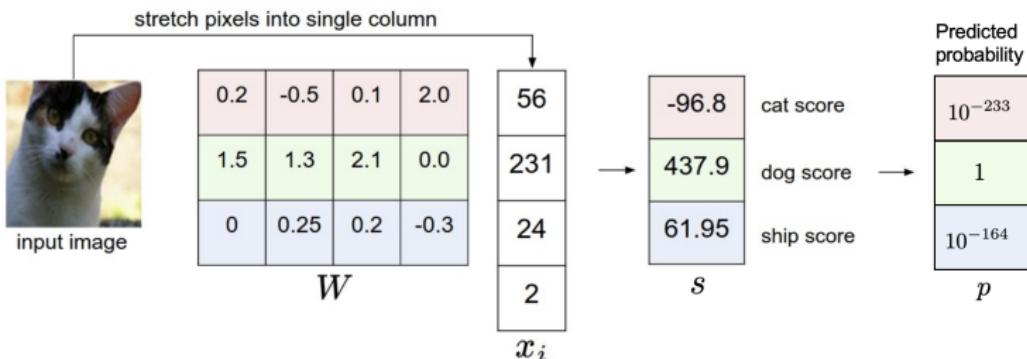
where $W \in \mathbb{R}^{C \times d}$ is the model parameter to learn and s_i indicates the score of the i -th class



Multi-classification

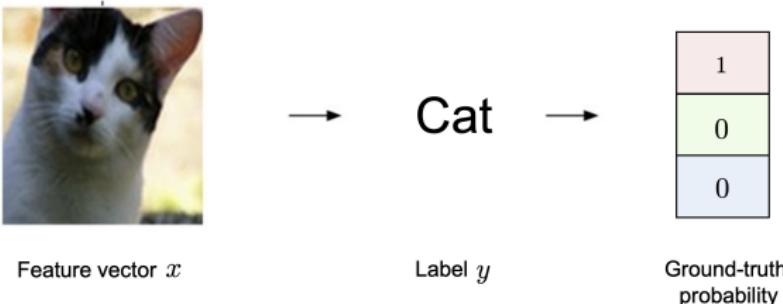
- Given the score vector s , we calculate the probability of each class with the **softmax** function as follows

$$p_i = \frac{\exp(s_i)}{\sum_{j=1}^C \exp(s_j)} \in (0, 1)$$



Multi-classification

- Recall the ground truth probability



- With cross-entropy function, we can measure the discrepancy between the predicted and real probability as

$$\ell(\mathbf{x}, y; W) = -\log \left(\frac{\exp(s_y)}{\sum_{j=1}^C \exp(s_j)} \right) \quad \text{where} \quad s = W\mathbf{x}$$

Multi-classification: loss function

- Now we achieve the loss function in multi-classification

$$\min_{W \in \mathbb{R}^{C \times d}} -\frac{1}{N} \sum_{i=1}^N \log \left(\frac{\exp(s_{y_i})}{\sum_{j=1}^C \exp(s_j)} \right) \quad \text{where } s = W\mathbf{x}_i$$

- Solving it with gradient descent, we can achieve the model parameter W^*
- Given the model W^* and a new feature vector \mathbf{x} , we can predict its class as

$$y = \arg \max_i \left\{ \frac{\exp(s_i)}{\sum_{j=1}^C \exp(s_j)} \right\} \quad \text{where } s = W^*\mathbf{x}$$

Neural network

- In the above machine learning tasks, we directly feed data into the regression or classification models
- A better way is to first transform data into more useful representations, and then feed the representation into the regression or classification models



Face representation

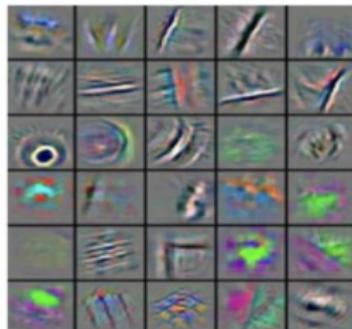
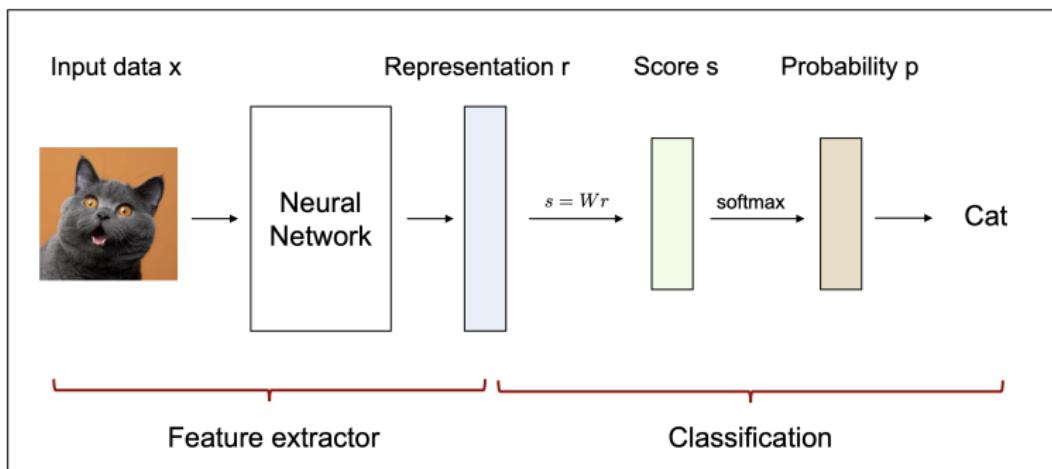


Image representation

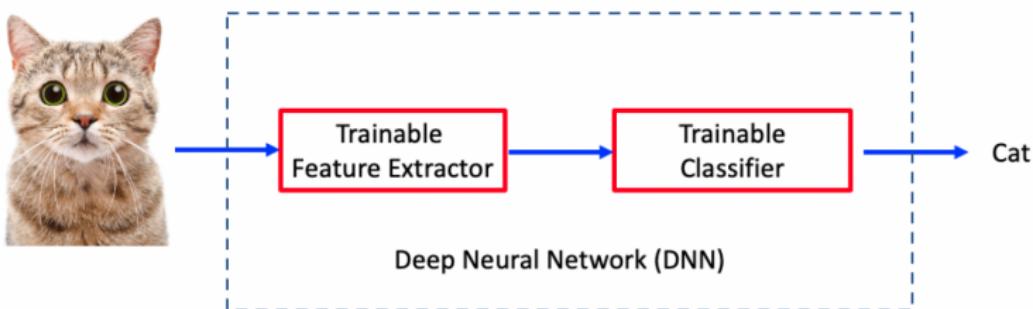
Neural network

- It is difficult to transform data to representations manually
- Deep neural network can do it **automatically**

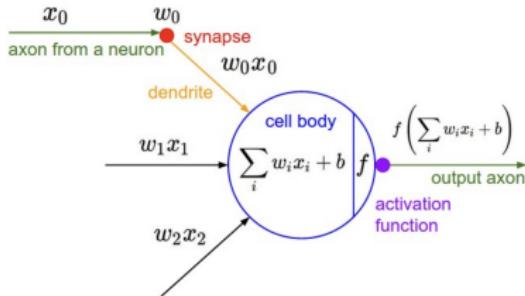


Neural network

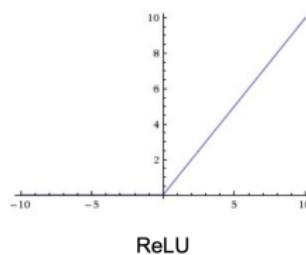
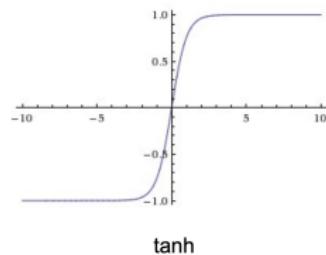
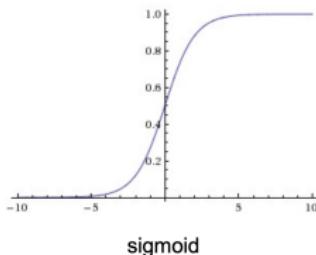
- Narrowly speaking, neural network targets to extract features
- But in convention, we also regard classification as part of neural network
- In other words, neural network includes a **feature extractor** and a **classifier**



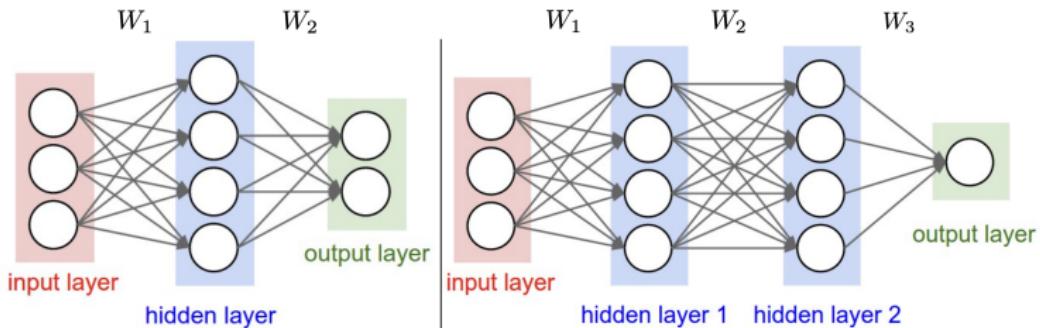
Neuron



- A neuron = affine mapping + nonlinear activation
- Various activations are used in real practice



Fully-connected neural network



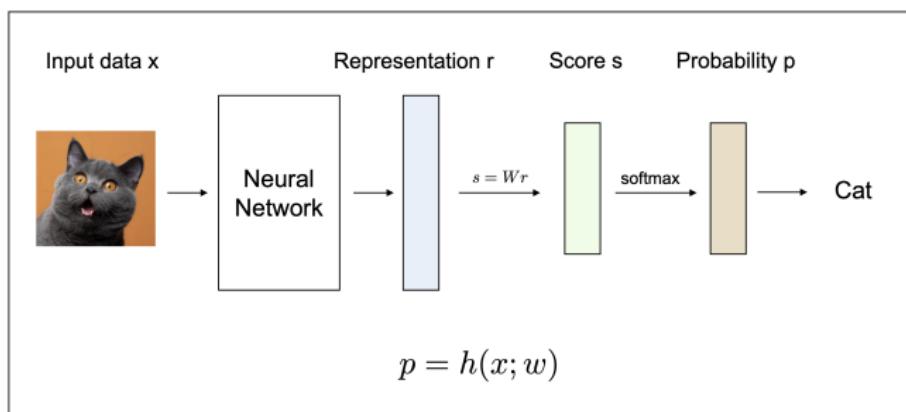
$$r = \sigma(W_2\sigma(W_1x))$$

$$r = \sigma(W_3\sigma(W_2\sigma(W_1x)))$$

- Fully-connected neural network can **approximate any continuous function**
- Given any continuous function $f(x)$ and some $\epsilon > 0$, there exists a Neural Network $g(x)$ with one hidden layer (with reasonable non-linearity, e.g. sigmoid) such that $\|f(x) - g(x)\| \leq \epsilon$ (Cybenko, 1989)

DNN model

- We model DNN as $h(\mathbf{x}; \mathbf{w}) : \mathbb{R}^d \rightarrow \mathbb{R}^c$
 - $\mathbf{w} \in \mathbb{R}^d$ is the DNN model parameter to be trained
 - \mathbf{x} is a random input data sample
 - c is the number of classes
- Given model parameter w , DNN outputs prediction probability p for input x



How to train a DNN model?

- Given good model w , DNN $h(\mathbf{x}; w)$ can make precise predictions
- But how to train/achieve the model parameter x ?
- Given a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^N$ where y_i is the ground-truth label for data \mathbf{x}_i
- Define $L(p_i, y_i) = L(h(\mathbf{x}_i; w), y_i)$ as a loss function to measure the difference/mismatch between predictions and ground-truth labels
- DNN training is to find a model parameter x such that the mismatch (between pred and real) are minimized across the entire dataset:

$$w^* = \arg \min_{x \in \mathbb{R}^d} \left\{ \frac{1}{N} \sum_{i=1}^N L(h(\mathbf{x}_i; w), y_i) \right\}$$

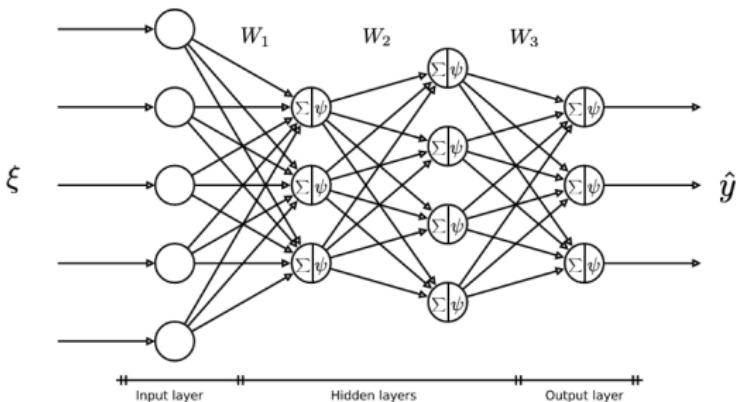
DNN model is notoriously difficult to train

- DNN model $L(h(x; w), y)$ is highly **non-convex**, and probably non-smooth

$$h(x; w) = \psi(\dots \psi(W_2 \cdot \psi(W_1 x + h_1) + h_2) \dots)$$

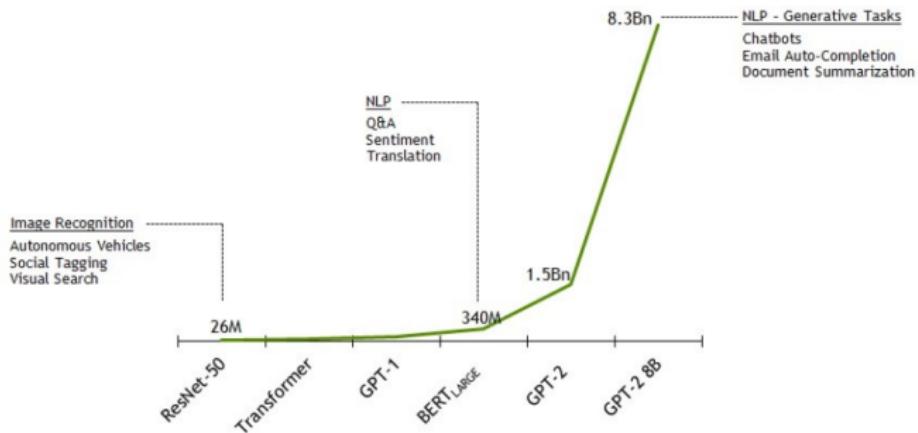
$$L(p; y) = \frac{1}{2} \|p - y\|^2 \text{ or } - \sum_i y_i \log(p_i) \text{ or others}$$

where $x = \{W_i, h_i\}$ and $\psi(\cdot)$ is a non-linear activation function



DNN model is notoriously difficult to train

- Cannot find global minima; trapped into local minima and saddle points
- The dimension of model parameter $w = \{W_i, h_i\}$ (or model size) is huge¹



¹Image source: neowin.net

DNN model is notoriously difficult to train

- Cannot find global minima; trapped into local minima and saddle points
- The dimension of model parameter $w = \{W_i, h_i\}$ (or model size) is huge
- The size of the dataset $\{x_i, y_i\}_{i=1}^m$ is huge

DNN Trainig = Non-convexity training + Huge dimension + Huge dataset

References I

G. Cybenko, "Approximation by superpositions of a sigmoidal function,"
Mathematics of control, signals and systems, vol. 2, no. 4, pp. 303–314,
1989.