ELEG 5491: Introduction to Deep Learning

Machine Learning Basics

Prof. LI Hongsheng

Office: SHB 428
e-mail: hsli@ee.cuhk.edu.hk
web: https://dl.ee.cuhk.edu.hk

Department of Electronic Engineering
The Chinese University of Hong Kong

Jan. 2022

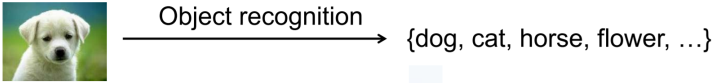
Outline

The objective of machine learning

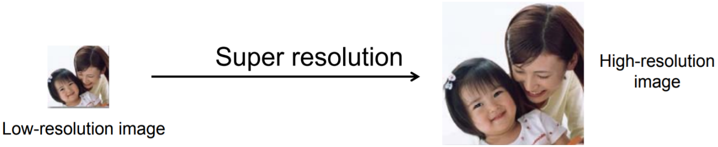
Learn a mapping function from training data

$$x \longrightarrow \boxed{F(x)} \longrightarrow y \begin{cases} \text{Class label (Classification)} \\ \text{Vector (Estimation)} \\ \text{Other things} \end{cases}$$

- Example of classification

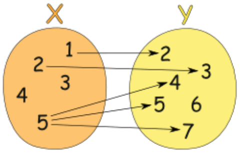


- Example of regression

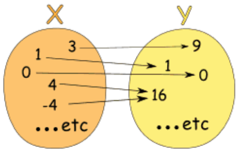


Machine learning aims at learning a function

- Given an input value or vector, a **function** assigns it with a value or vector
- "One-to-many" mapping is **not a function**. "Many-to-one" mapping is **a function**.



Not a function



A function

- Note that a function can have a vector output or matrix output. For instance, the following formula is still a function

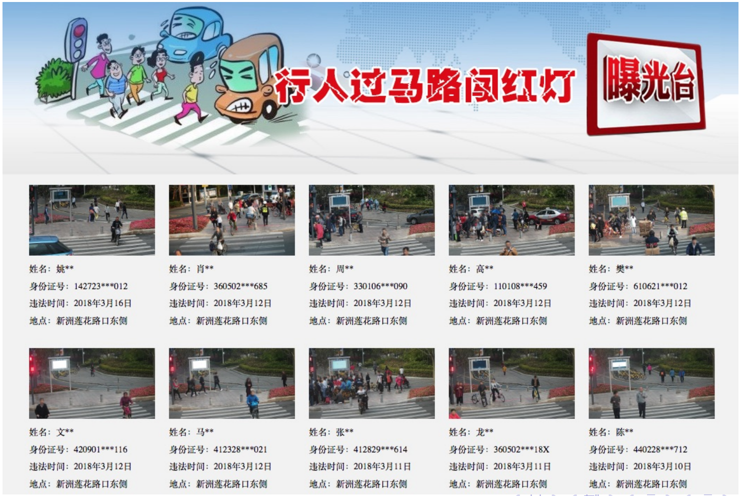
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 + x_2 \\ x_1 x_2 \end{bmatrix}$$

Function estimation

- We are interested in predicting y from input $\mathbf{x}$ and assume there exists a function that describes the relationship between y and $\mathbf{x}$, e.g., $y = f(x)$
- If the function f's parametric form is fixed, prediction function f can be parametrized by a parameter vector θ
- Estimating $\hat{f}$ from a training set $\mathcal{D} = \{(\mathbf{x}_1^{\text{train}}, y_1), (\mathbf{x}_2^{\text{train}}, y_2), \cdots, (\mathbf{x}_n^{\text{train}}, y_n)\}$
- With a better design of the parametric form of the function, the learner could achieve better performance
- This design process typical involves domain knowledge

Example of machine learning

- Face recognition in smart surveillance for crossing at a red light in China



姓名：姚**
身份证号：142723***012
违法时间：2018年3月16日
地点：新洲莲花路口东侧

姓名：肖**
身份证号：360502***685
违法时间：2018年3月12日
地点：新洲莲花路口东侧

姓名：周**
身份证号：330106***090
违法时间：2018年3月12日
地点：新洲莲花路口东侧

姓名：高**
身份证号：110108***459
违法时间：2018年3月12日
地点：新洲莲花路口东侧

姓名：樊**
身份证号：610621***012
违法时间：2018年3月12日
地点：新洲莲花路口东侧

姓名：文**
身份证号：420901***116
违法时间：2018年3月12日
地点：新洲莲花路口东侧

姓名：马**
身份证号：412328***021
违法时间：2018年3月12日
地点：新洲莲花路口东侧

姓名：张**
身份证号：412829***614
违法时间：2018年3月11日
地点：新洲莲花路口东侧

姓名：龙**
身份证号：360502***18X
违法时间：2018年3月11日
地点：新洲莲花路口东侧

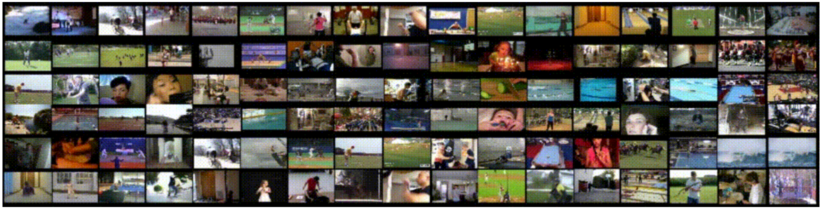
姓名：陈**
身份证号：440228***712
违法时间：2018年3月10日
地点：新洲莲花路口东侧

Example of machine learning

- Object detection for autonomous driving

Example of machine learning

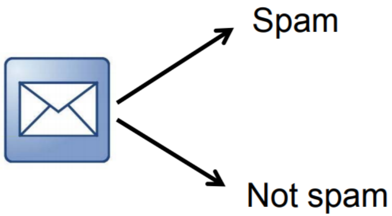
- Action recognition



Sample video frames from UCF-101 dataset

Example of machine learning

- Email spam classification



Example of machine learning

- Speech recognition

Recognizing the content

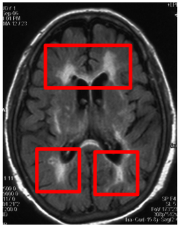
how
are
you
...

Recognizing the identity

Person 1
Person 2
Person 3
...

Example of machine learning

- Computer-aided medical diagnosis

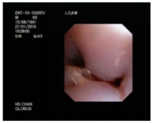


Normal?

Diseased?

Video of upper aerodigestive tract taken by endoscopy

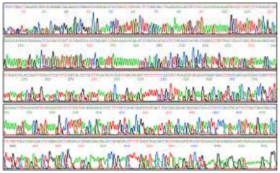
abnormal?



normal?

Example of machine learning

- Function of gene sequence classification



$\overset{?}{\longrightarrow}$

Trucarboxylic acid (TCA) cycle

Respiration

Cytoplasmic ribosomes

Proteasome

Histones

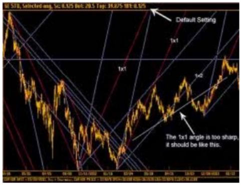
Helix-turn-helix proteins

Genes

Functional classes

Example of machine learning

- Financial time series prediction



$\xrightarrow{?}$

Sell

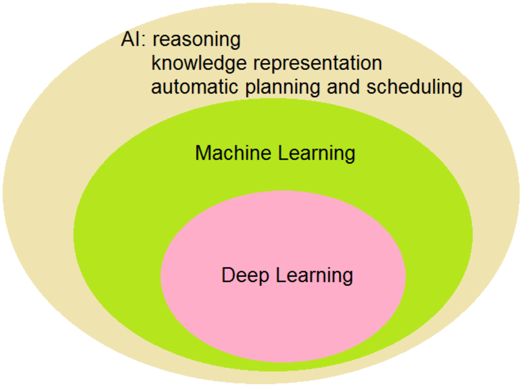
Buy

Wait

Share price time series **Investment decisions**

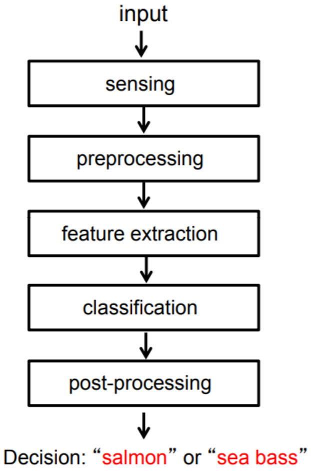
Machine learning is a sub-field of artificial intelligence

Artificial intelligence: general reasoning
Machine learning: learn to obtain a function with expected outputs
Deep learning: machine learning with deep neural networks



Machine learning systems

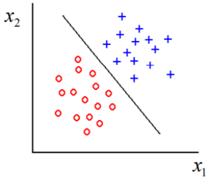
Classification of types of fishes



Classification model

- Each sample is represented by a d-dimensional feature vector.
- The goal of classification is to establish decision boundaries in the feature space to separate samples belonging to difference classes patterns belonging to difference classes

$$\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} = \vec{x} \longrightarrow \boxed{f(\vec{x})} \longrightarrow y \in \{1,\dots L\}$$



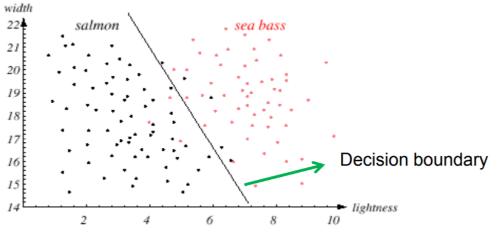
Feature matters

- Properly choose features for different classification/regression problems is one of the key problems in machine learning applications
- Computer vision applications
  - Histogram of Oriented Gradients (HOG) features
  - Scale-invariant Feature Transform (SIFT) features
  - Oriented FAST and rotated BRIEF (ORB) features
- Speech recognition
  - Linear Predictive Codes (LPC) features
  - Perceptual Linear Prediction (PLP) features
  - Mel Frequency Cepstral Coefficients (MFCC) features
- If discriminative (good) enough features exist, even a very simple linear classifier can perform well
- Back in 1970s to early 2010s, features are mostly manually designed by humans according to experience

Multi-dimensional feature vectors

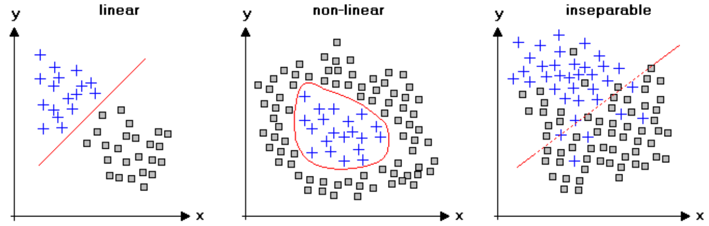
- Jointly use two features (lightness and width)
- Each sample can be considered as a 2-dimensional point in the feature space
- The classification error on the training data becomes lower than using only one feature

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{matrix} \longrightarrow \textbf{Lightness} \\ \longrightarrow \textbf{Width} \end{matrix}$$



Linearly separable features

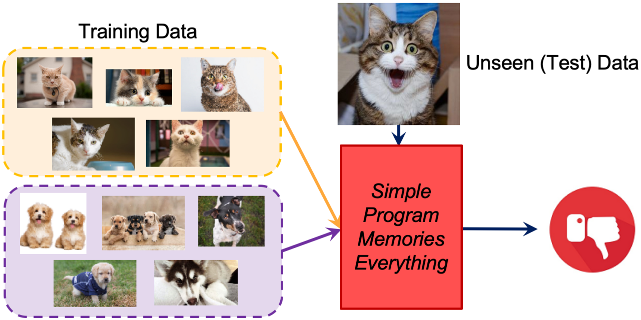
- What makes good features: linearly separable features
- A linear classifier (decision boundary) that correctly classifies all training samples



- However, such a property cannot be met for most scenarios

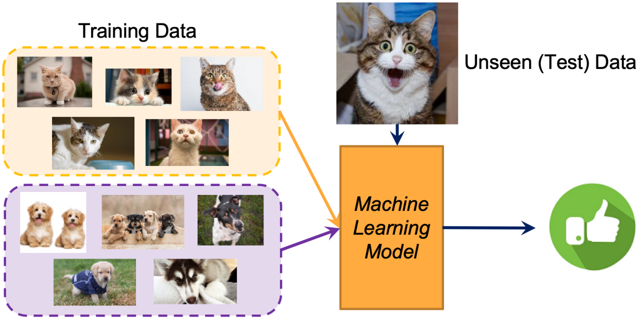
Training set and testing set

- The data with annotations should be separated into training set, validation set (optional), and testing set
- Reaching 100% accuracy on the training set cannot guarantee good performance to general unseen samples



Training set and testing set

- The data with annotations should be separated into training set, validation set (optional), and testing set
- Reaching 100% accuracy on the training set cannot guarantee good performance to general unseen samples
- The model must have the capability of **generalize** to unseen (test) samples



Model (learner) capacity

- **Capacity.** The ability of the learner (or called model) to discover a function taken from a family of functions. Examples:
  - Linear predictor
    $$y = wx + b$$
  - Quadratic predictor
    $$y = w_2x^2 + w_1x + b$$
  - Degree-10 polynomial predictor
    $$y = b + \sum_{i=1}^{10} w_i x^i$$
  - The latter family is richer, allowing to capture more complex functions
- Capacity can be measured by the number of training examples $\{x^{(\mathrm{i})}, y^{(\mathrm{i})}\}$ that the learner **could always fit**, no matter how to change the values of $x^{(\mathrm{i})}$ and $y^{(\mathrm{i})}$

Underfitting

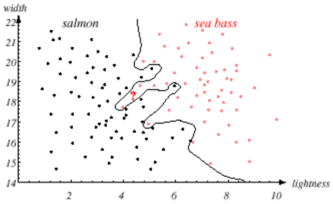
- The learner cannot find a solution that fits training examples well
  - For example, use linear regression to fit training examples $\{x^{(\mathsf{i})}, y^{(\mathsf{i})}\}$ where $y^{(\mathsf{i})}$ is an quadratic function of $x^{(\mathsf{i})}$
- Underfitting means that the learner cannot capture some important aspects of the data
- Reasons for underfitting happens
  - Model is not rich enough
  - Difficult to find the global optimum of the objective function on the training set or easy to get stuck at local minimum
  - Limitation on the computation resources (not enough training iterations of an iterative optimization procedure)
- Underfitting commonly happens in non-deep learning approaches with large scale training data and could be even a more serious problem than overfitting in some cases

Overfitting

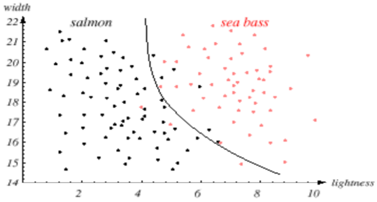
- The learner fits the training data well, but loses the ability to generalize well, i.e. it has small training error but larger generalization error
- A learner with large capacity tends to overfit
  - The family of functions is too large (compared with the size of the training data) and it contains many functions which all fit the training data well.
  - Without sufficient data, the learner cannot distinguish which one is most appropriate and would make an arbitrary choice among these apparently good solutions
  - A separate validation set helps to choose a more appropriate one
  - In most cases, data is contaminated by noise. The learner with large capacity tends to describe random errors or noise instead of the underlying models of data (classes)

Model complexity (capacity)

- The goal is to classify novel examples not seen yet, but not the training examples!
- Generalization. The ability to correctly classify new examples that differ from those used for training

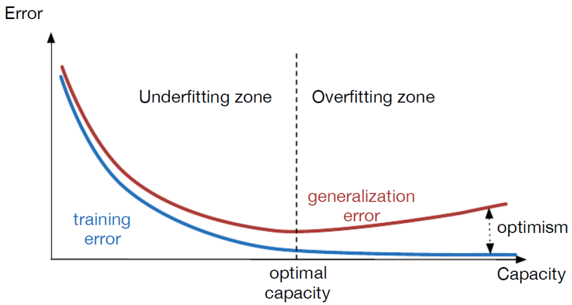


Overly complex models lead to complicated decision boundaries. It leads to perfect classification on the training examples, but would lead to poor performance on new patterns.



The decision boundary might represent the optimal tradeoff between performance on the training set and simplicity of classifier, therefore giving highest accuracy on new patterns.

Optimal capacity



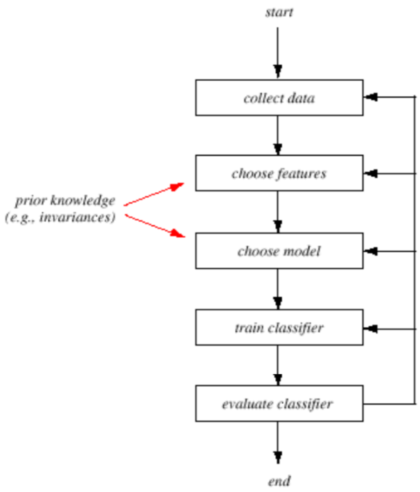
Typical relationship between capacity and both training and generalization (or test) error. As capacity increases, training error can be reduced, but the optimism (difference between training and generalization error) increases. At some point, the increase in optimism is larger than the decrease in training error (typically when the training error is low and cannot go much lower), and we enter the overfitting regime, where capacity is too large, above the optimal capacity. Before reaching optimal capacity, we are in the underfitting regime.

(Bengio et al. Deep Learning 2014)

Machine learning systems

- Feature extraction
  - Discriminative features
  - Invariant features with respect to certain transformation
  - A small number of features
- Classifier/regressor
  - Tradeoff of classification errors on the training set and the model complexity
  - Decide the form of the classifier
  - Tune of the parameters of the classifiers by training
- Post-processing
  - Risk: the cost of mis-classifying sea bass is different than that of mis-classifying salmon
  - Prior: it is more likely for a fish to be the same class as its previous one
  - Integrating multiple classifiers: classifiers are based on different sets of features

Training cycle



Data collection

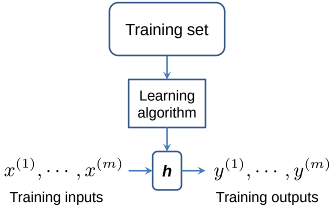
- Collect both training data, validation data, and test data
- Label the ground truth annotations
- Is the training set large enough?
- Is the training set representative enough?
  - Are the training data and the testing data collected under the same condition?
- Initial examination of the data to get a feel of data structure
  - Summary of statistics
  - Producing plots
- The analysis of the evaluation results may require further data collection

Problem setup for supervised learning

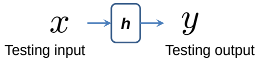
- Given pairs of inputs and outputs, learn a function to map inputs to outputs
  - Function inputs: **features** $x^{(i)}$ ($x^{(i)} \in \mathbb{R}^d$ for general problems)
  - Function outputs: target outputs $y^{(i)} \in \mathbb{R}$
  - One training sample: $(x^{(i)}, y^{(i)})$
  - Training set of m samples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$
  - Hypothesis $h : \mathbb{R}^d \to \mathbb{R}$: the function to be learned to map a general input x to expected output y

Training and testing

- The parametric form of h is fixed
- Training: find the optimal **parameters** θ of function h based on the training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$, usually by minimizing some **cost function**
- Testing: **fix** the found optimal parameters θ, given the input features of one unseen example x, predict the output value y



Training Stage

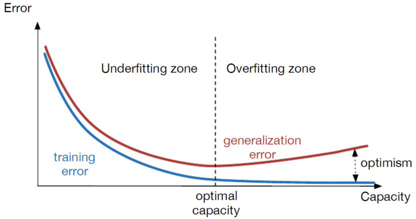


Testing Stage

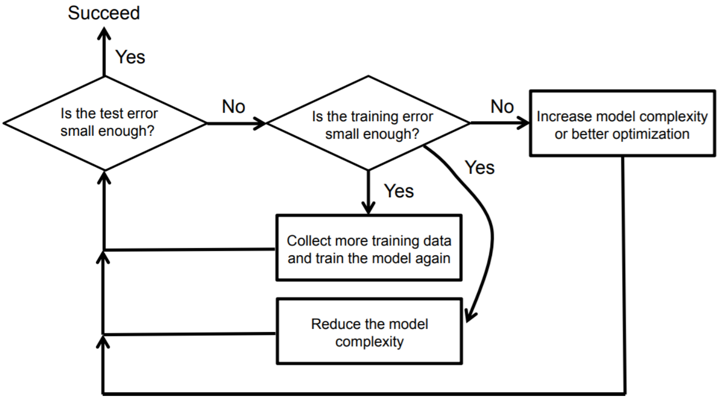
- If target variables y are continuous, the learning is a **regression** problem
- If target variables y can only take a small number of discrete values (classes), it is a **classification** problem

Evaluation

- Apply the trained classifier to an independent validation set of labeled samples
- It is important to both measure the performance of the system and to identify the need for improvements in its system and to identify the need for improvements in its components
- Compare the error rates on the training set and the validation set to decide if it is overfitting or underfitting
  - High error rates on both the training set and the validation set: underfitting
  - Low error rate on the training set and high error rate on the validation set: overfitting



Design cycle

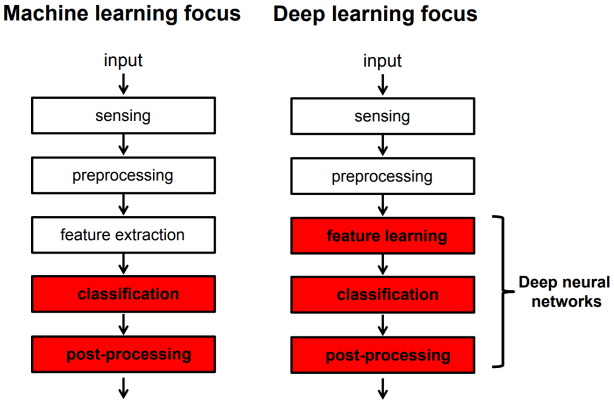

Succeed
Yes
Is the test error small enough?
No
Is the training error small enough?
No
Increase model complexity or better optimization
Yes
Yes
Collect more training data and train the model again
Reduce the model complexity

Learning schemes

- Supervised learning
  - An "expert" provides a category label for each pattern in the training set
  - It may be cheap to collect patterns but expensive to obtain the labels
- Unsupervised learning
  - The system automatically learns feature transformation from the training samples without any annotation to best represent them
- Weakly supervised learning
  - The supervisions are not exact or rough
  - Example: learning image segmentation by providing image-level annotations
- Semi-supervised learning
  - Some samples have labels, while some do not

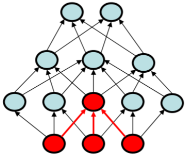
Deep learning

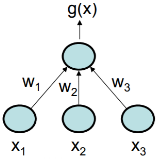
- Deep learning aims at learning better feature representations

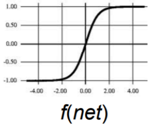


Neural networks

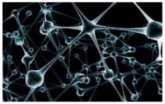
- Deep learning is based on neural networks
- Neural networks originates back to 1970s-1980s





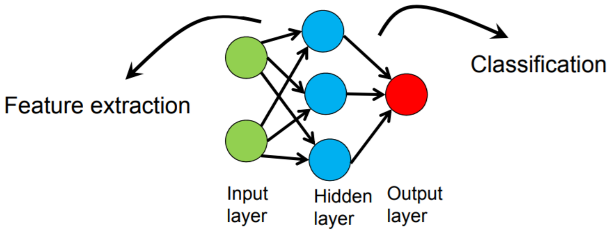


$$g(\mathbf{x}) = f(\sum_{i=1}^{d} x_i w_i + w_0) = f(\mathbf{w}^t\mathbf{x})$$



Neural networks

- A network of interconnecting artificial neurons
    - It simulates some properties of biological neural networks: learning generalization adaptivity fault networks: learning, generalization, adaptivity, fault tolerance, distribution computation
    - Low dependence on domain-specific knowledge

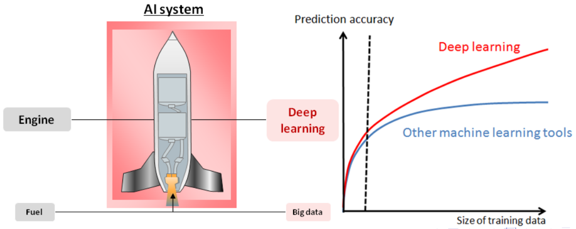


What makes the difference?

- Deep learning becomes popular again in 2010s
- Large-scale training data
- Super parallel computing power (e.g. GPU and TPU)

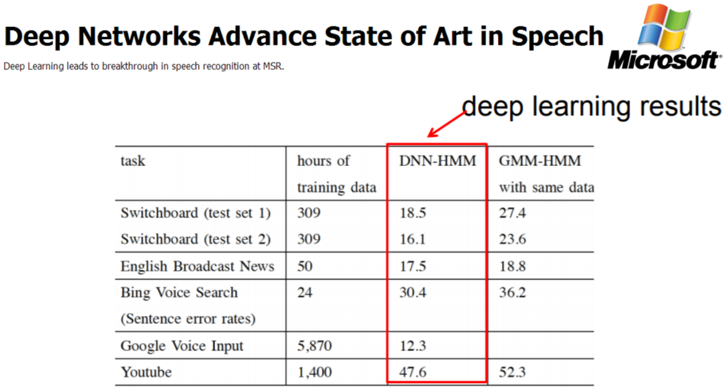
Deep learning in neural networks

- Become hot since 2006
  - Hinton et al., “A Fast Learning Algorithm for Deep Belief Nets,” Neural Computation, 2006
- Other famous researchers in deep learning
  - Yann LeCun (NYU), Yoshua Bengio (U of Montreal)
- MIT Technology Review lists deep learning as MIT Technology Review lists deep learning as one of the top-10 breakthrough technologies in 2013
- Neural networks with more hidden layers
- Many existing statistical models can be approximated as neural networks with one or two hidden layers



Success of deep learning in 2011

- Speech recognition breakthrough in 2011



Deep Networks Advance State of Art in Speech

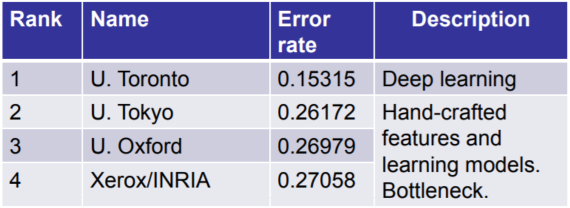
Deep Learning leads to breakthrough in speech recognition at MSR.

Microsoft
deep learning results

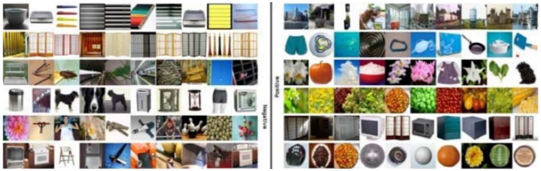
task	hours of training data	DNN-HMM	GMM-HMM with same data
Switchboard (test set 1)	309	18.5	27.4
Switchboard (test set 2)	309	16.1	23.6
English Broadcast News	50	17.5	18.8
Bing Voice Search (Sentence error rates)	24	30.4	36.2
Google Voice Input	5,870	12.3	
Youtube	1,400	47.6	52.3

Success of deep learning

- Object classification over 1 million images of 1000 classes
  - ImageNet Challenge 2012

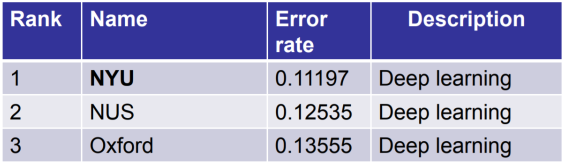


Rank	Name	Error rate	Description
1	U. Toronto	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	



Success of deep learning

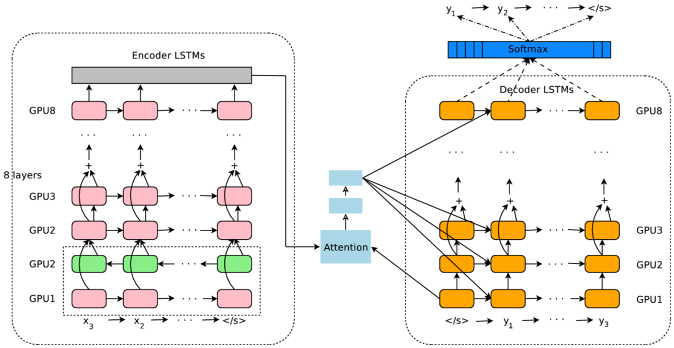
- ImageNet Challenge 2013
    - All teams used deep learning
    - MSRA, IBM, Adobe, NEC, Clarifai, Berkley, U. Tokyo, UCLA, UIUC, Toronto, etc.



Rank	Name	Error rate	Description
1	**NYU**	0.11197	Deep learning
2	NUS	0.12535	Deep learning
3	Oxford	0.13555	Deep learning

Success of deep learning

- Google's neural machine translation system in 2016

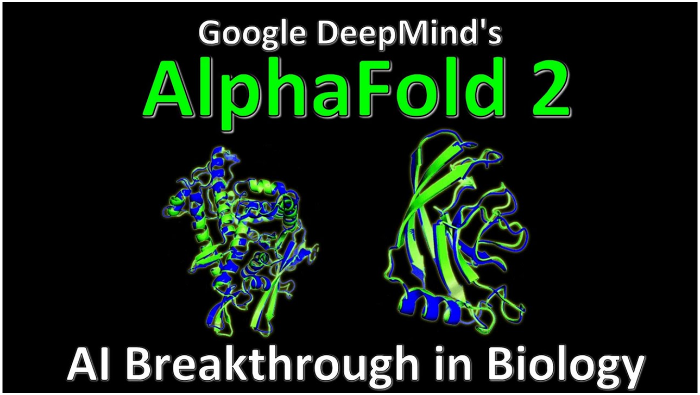


Success of Go playing in 2016

- DeepMind's AlphaGo beat Go master Lee Sedol in 2016

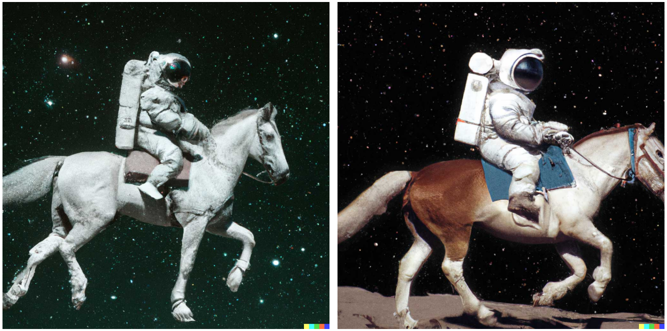
Success of deep learning in 2020

- DeepMind's AlphaFold beats humans in protein folding estimation



Success of text-based image generation in 2022

- OpenAI's DALL-E2



An astronaut riding a horse in a photo realistic style

Success of text-based image generation in 2022

- Google's Imagen

A brain riding a rocketship heading towards the moon.

A dragon fruit wearing karate belt in the snow.

A small cactus wearing a straw hat and neon sunglasses in the Sahara desert.

A photo of a Corgi dog riding a bike in Times Square. It is wearing sunglasses and a beach hat.

A marble statue of a Koala DJ in front of a marble statue of a turntable. The

A robot couple fine dining with Eiffel Tower in the background.

A bucket bag made of blue suede. The bag is decorated with intricate golden

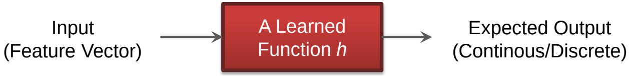
Android Mascot made from bamboo.

Different types of deep learning

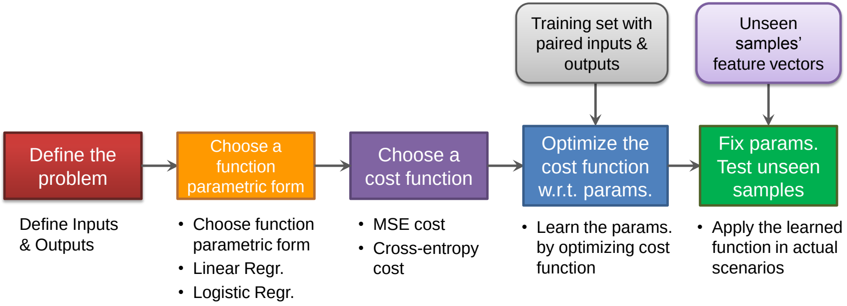
- Back in 2006, the name “deep learning” proposed by G. Hinton mostly describe deep neural networks trained in the unsupervised learning setting
  - Restricted Boltzmann Machine
  - Deep Belief Network
  - Auto-encoder
- In the past a few years, deep learning research is dominated by supervised learning approaches
  - Multi-layer perceptron (MLP)
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
- Since the recent two years, unsupervised deep learning and semi-supervised learning have re-gained much attention from the research community

Overview of supervised learning

- Supervised learning aims to learn a function that maps input feature vectors to expected output values



- Process map for supervised learning


Training set with paired inputs & outputs
Unseen samples' feature vectors
Define the problem
Choose a function parametric form
Choose a cost function
Optimize the cost function w.r.t. params.
Fix params. Test unseen samples
Define Inputs & Outputs
• Choose function parametric form
• Linear Regr.
• Logistic Regr.
• MSE cost
• Cross-entropy cost
• Learn the params. by optimizing cost function
• Apply the learned function in actual scenarios

The blood fact content example

- We enrich the example

Weight (kilograms)	Age (years)	Blood fat content
84	46	354
73	20	190
65	52	405
⋮	⋮	⋮
70	30	263

- Input features of each sample would be $x^{(i)} = [x_1^{(i)}, x_2^{(i)}]^T \in \mathbb{R}^2$, two-dimensional vectors
- $x_1^{(i)}$ is the weight of the ith person in the training set
- $x_2^{(i)}$ is the age of the ith person
- Linear regression
  - Approximate y as a linear function of feature vectors $x = [x_1, x_2]^T \in \mathbb{R}^2$

$$y = h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

  - For more general n-dimensional feature vectors $x \in \mathbb{R}^{n+1}$, where we assume $x_0 = 1$ is a constant

$$y = h(x) = \theta_0 + \theta_1 x_1 + \cdots \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

Linear regression

- $\theta = [\theta_0, \cdots, \theta_n]^T \in \mathbb{R}^{n+1}$ are the **parameters** (or **weights**) for learning
- Given a training set, how do we pick, or learn, the parameters θ?
- Given the training set, make $h(x) = \theta^T x$ close to y in the training set as much as possible
- How to measure the closeness of $h(x)$'s prediction?
- Use **cost function** or (**lost function** or just simply **loss**). Note that here we don't normalize w.r.t. the number of samples m for simplicity. The actual loss generally conducts normalization

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$$

- This function is called Mean Squared Error (MSE) / L2 cost function
- Our goal would be to learn θ to minimize the cost function

$$\min_{\theta} J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$$

- This specific minimization problem (when $m > n$) with MSE cost function is called **ordinary least squares** problem

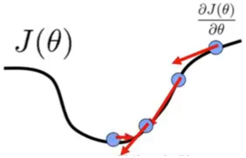
Gradient descent for optimization

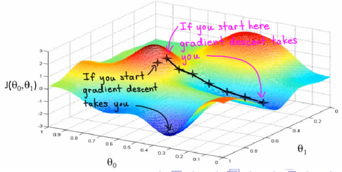
- Use a search algorithm that starts with some "initial guess" for θ and that iteratively changes θ to make $J(\theta)$ smaller
- To recover local minimum, we could utilize the gradient descent algorithm with initial parameter $\theta^{(0)}$

Gradient descent algorithm

```
For iteration $k = 1, 2, 3, \cdots$
    For parameter $\theta_j$, where $j = 1, 2, \cdots, n$
        $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$,
        Terminate if $k$ is large enough or $\|\nabla_\theta J(\theta)\|$ is small enough
```

α is called the *learning rate* (or *step size*), $-\nabla J(\theta_i)$ is the negative gradient direction.





Matrix calculus

- Suppose that $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ is a function that takes as input a matrix $A \in \mathbb{R}^{m\times n}$ and returns a scalar real value
- The **gradient** of f with respect to $A \in \mathbb{R}^{m\times n}$ is the matrix of partial derivatives,

$$\nabla_A f(A) \in \mathbb{R}^{m\times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

where $(\nabla_A f(A))_{ij} = \dfrac{\partial f(A)}{\partial A_{ij}}$

Matrix calculus

- The **gradient** of f with respect to $x \in \mathbb{R}^n$ is the vector of partial derivatives,

$$\nabla_x f(x) \in \mathbb{R}^n = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

- Properties
  - $\nabla_x(f(x) + g(x)) = \nabla_x f(x) + \nabla_x g(x)$
  - For $t \in \mathbb{R}$, $\nabla_x(tf(x)) = t\nabla_x f(x)$

Gradient descent for linear regression

- When we only have one training sample $(x^{(i)}, y^{(i)})$,

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y\right)^2 \\ &= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(h_\theta(x) - y\right) \\ &= \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y\right) \\ &= \left(h_\theta(x) - y\right) x_j\end{aligned}$$

- For a single training sample $(x^{(i)}, y^{(i)})$, we have

$$\theta_j := \theta_j + \alpha(y^{(i)} - h(x^{(i)}) x_j^{(i)}$$

- For the whole training set, we have the following **batch gradient descent** algorithm

```
Repeat until convergence {
    $\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$        (for every $j$).
}
```

Gradient descent for linear regression

- One key disadvantage remains, at each parameter update iteration, we need to sum over all training samples

$$\theta_j := \theta_j + \alpha \boxed{\sum_{i=1}^{m}} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

- This could be very time-consuming, when the number of samples is very large (for example, $m > 1$ million)
- Stochastic gradient descent: at each parameter update iteration, sample only 1 training sample to accelerate the parameter updating

Stochastic gradient descent for linear regression

```
For iteration $k = 1, 2, 3, \cdots$
    Randomly (or sequentially) sample one training sample from the training set $(x^{(i)}, y^{(i)})$
    $\theta_j := \theta_j + \alpha(y^{(i)} - h(x^{(i)}))x_j^{(i)}$ (for every $j$)
    Terminate if $k$ is large enough or $\|\nabla_\theta J(\theta)\|$ is small enough
```

Stochastic gradient descent for linear regression

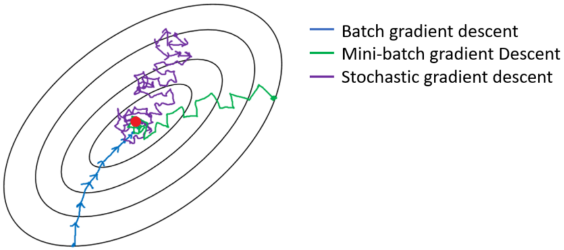
- Often, stochastic gradient descent gets θ "close" to the minimum much faster than batch gradient descent
- However that it may never "converge" to the minimum, and the parameters θ will keep oscillating around the minimum of $J(\theta)$
- In practice most of the values near the minimum will be reasonably good approximations to the true minimum
- Balance between gradient descent and stochastic gradient descent? Mini-batch gradient descent!

Mini-batch stochastic gradient descent for linear regression

```
For iteration $k = 1, 2, 3, \cdots$
    Randomly (or sequentially) sample a mini-batch of training samples $\mathcal{P}$ from the training set $(x^{(i)}, y^{(i)})$
    $\theta_j := \theta_j + \sum_{\text{for } i \text{ in } \mathcal{P}} \alpha(y^{(i)} - h(x^{(i)}))x_j^{(i)}$ (for every $j$)
    Terminate if $k$ is large enough or $\|\nabla_\theta J(\theta)\|$ is small enough
```

Mini-batch stochastic gradient descent

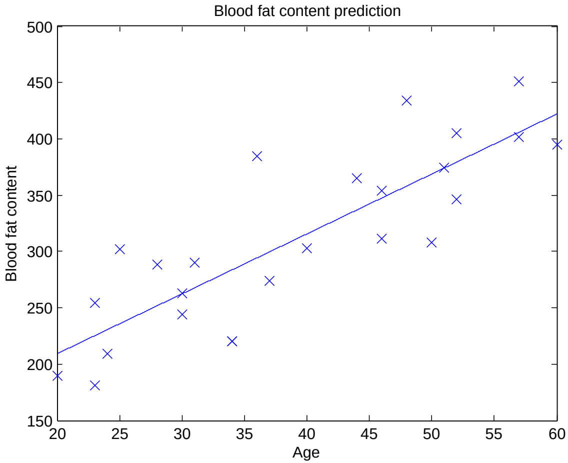
- **Mini-batch stochastic gradient descent** is between gradient descent and stochastic gradient descent
- Much faster than gradient descent
- More stable than stochastic gradient descent



- NOTE: those gradient descent algorithms are also utilized to optimize other machine learning models (e.g., deep neural networks)

Linear regression for 1D features

- Solving for our blood fat content prediction problem with only one input feature "age" yields $\theta_0 = 102.5751$ and $\theta_1 = 5.32$



Binary classification

- We first look at the binary classification problem with linear model
- Outputs y of the learned function h:
  - 0 – **negative class**
  - 1 – **positive class**

- Naturally, the possible range of values that h could output must be $\{0, 1\}$
- We could first relax the discrete constraints of the outputs to be in the continuous range $[0, 1]$

- We choose the following form for **hypothesis** h

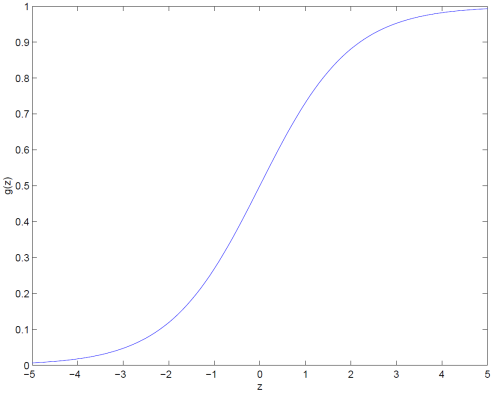
$$h(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where

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

Logistic regression

- $\theta^T x$ is a linear function. Its range of possible values is in $[-\infty, \infty]$
- $g(z)$ is called **logistic function** or **sigmoid function**
- $g(z)$ maps $[-\infty, \infty] \rightarrow [0, 1]$
- $g(z) \rightarrow 1$ when $z \rightarrow \infty$, and $g(z) \rightarrow 0$ when $z \rightarrow -\infty$



Derivative of the sigmoid function

- Derivative of the sigmoid function $g(z)$ is easy to compute

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\ &= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)) \end{aligned}$$

- NOTE: We could view the **continuous value** $h(\theta^T x)$ as **confidence** (or **probability**) that x belongs to the **positive** class

- Given a series of training samples $(x^{(1)}, 0), (x^{(2)}, 1), \cdots, (x^{(m)}, 0)$, how can we fit the parameters θ for the linear model $\theta^T x$?

Probabilistic modeling of binary classification

- Based on our probabilistic view on the **continuous value** $h(\theta^T x)$, we assume that

$$P(y=1|x;\theta) = h(x) = g(\theta^T x)$$
$$P(y=0|x;\theta) = 1 - h(x) = 1 - g(\theta^T x)$$

- Given the input features x, it has the probability $P(y=1|x;\theta)$ that x belongs to **positive** class; it has the probability $P(y=0|x;\theta)$ that x belongs to **negative** class

- Since $y=0$ or $y=1$, we could write the formula for a single training sample in a compact way

$$p(y|x;\theta) = (h(x))^y (1-h(x))^{1-y}$$

Probabilistic modeling of binary classification

- We are interested in maximize the joint probability (likelihood) over the **entire** training set $X = \{x^{(1)}, \cdots, x^{(m)}\}$ and $\vec{y} = \{y^{(1)}, \cdots, y^{(m)}\}$

$$
\begin{aligned}
\max_{\theta} L(\theta) = p(\vec{y}|X;\theta) \\
&= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\
&= \prod_{i=1}^{m} (h(x^{(i)}))^{y^{(i)}} (1 - h(x^{(i)}))^{1-y^{(i)}}
\end{aligned}
$$

- Convert the production to summation by taking $\log$ function over the likelihood L to obtain log likelihood ℓ

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))
\end{aligned}
$$

NOTE: $\log$ is a monotonically increasing function. θ and $\log L(\theta)$ would all reach their maximum values at the same parameters θ

Gradient ascent for maximizing log likelihood

- Similar to our optimization scheme for linear regression, we could use gradient **ascent to maximize** the log likelihood ℓ

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta)$$

- Consider only one training sample (x, y)

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g\left(\theta^T x\right)} - (1-y) \frac{1}{1 - g\left(\theta^T x\right)} \right) \frac{\partial}{\partial \theta_j} g\left(\theta^T x\right) \\
&= \left(y \frac{1}{g\left(\theta^T x\right)} - (1-y) \frac{1}{1 - g\left(\theta^T x\right)} \right) g\left(\theta^T x\right) \left(1 - g\left(\theta^T x\right)\right) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left(y \left(1 - g\left(\theta^T x\right)\right) - (1-y) g\left(\theta^T x\right) \right) x_j \\
&= \left(y - h_\theta(x)\right) x_j
\end{aligned}$$

Using the fact that $g'(z) = g(z)(1 - g(z))$

- Mini-batch stochastic gradient ascent for logistic regression

$$\theta_j := \theta_j + \alpha \sum_{i \in \mathcal{B}} (y^{(i)} - h(x^{(i)})) x_j^{(i)}$$

Other g choices: perceptron learning algorithm

- Linear regression fails to handle classification problem because its outputs are not bounded
- Logistic regression utilizes a "soft" bounding function (sigmoid function) to map the values into the range $[0, 1]$

- Perceptron learning: use the following $g(z)$ to bound the outputs of $\theta^T x$

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

  The hypothesis would then be $h(x) = g(\theta^T x)$
- The hypothesis h is not differentiable. Specific parameter updating scheme was designed

Softmax regression for multi-class classification

- The outputs are expected to be of one of the k pre-defined classes
- For each class i, a conditional probability $\phi_i = p(y = i|x; \Theta)$ of the input feature vector x belonging to class $y = i$ is estimated, which satisfy

$$\sum_{j=1}^{k} \phi_j = 1$$

- For the ith class, a linear model is first utilized to calculate the linear mapping $\Theta_i^T x$, where $\Theta_i^T \in \mathbb{R}^{n+1}$ and can stored as rows in a matrix $\Theta \in \mathbb{R}^{k\times(n+1)}$
- The conditional probabilities of each class is then calculated as

$$p(y = i|x; \Theta) = \phi_i = \frac{e^{\Theta_i^T x}}{\sum_{j=1}^{k} e^{\Theta_j^T x}}$$

- This function involving confidences of multiple classes is called **softmax function**. It can be easily check that $\sum_{j=1}^{k} \phi_j = 1$

Softmax regression for multi-class classification

- The **hypothesis** h for softmax regression outputs a k-dimensional vector

$$h_\Theta(x) = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_k \end{bmatrix} = \begin{bmatrix} \frac{\exp(\Theta_1^T x)}{\sum_{j=1}^k \exp(\Theta_j^T x)} \\ \frac{\exp(\Theta_2^T x)}{\sum_{j=1}^k \exp(\Theta_j^T x)} \\ \vdots \\ \frac{\exp(\Theta_k^T x)}{\sum_{j=1}^k \exp(\Theta_j^T x)} \end{bmatrix}$$

- The learning targets for different classes

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Cost function of softmax regression

- We use cross-entropy cost function for softmax regression

$$
\begin{aligned}
H(\Theta) = -\ell(\Theta) &= -\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)};\Theta) \\
&= -\sum_{i=1}^{m} \log \prod_{l=1}^{k} \left(\frac{\exp(\Theta_l^T x^{(i)})}{\sum_{j=1}^{k} \exp(\Theta_j^T x^{(i)})} \right)^{\mathbf{1}\{y^{(i)}=l\}} \\
&= -\sum_{i=1}^{m} \sum_{l=1}^{k} \mathbf{1}\{y^{(i)} = l\} \log \left(\frac{\exp(\Theta_l^T x^{(i)})}{\sum_{j=1}^{k} \exp(\Theta_j^T x^{(i)})} \right)
\end{aligned}
$$

- Gradients for softmax regression, considering one training sample $(x^{(i)}, y^{(i)})$, would be

$$
\begin{aligned}
\nabla_{\Theta_j} H(\theta) &= -\nabla_{\Theta_j} \log \left(\frac{\exp(\Theta_{y^{(i)}}^T x^{(i)})}{\sum_{l=1}^{k} \exp(\Theta_l^T x^{(i)})} \right) \\
&= -\nabla_{\Theta_j} \left(\Theta_{y^{(i)}}^T x^{(i)} \right) + \nabla_{\Theta_j} \log \left(\sum_{l=1}^{k} \exp(\Theta_l^T x^{(i)}) \right)
\end{aligned}
$$

Gradient descent for softmax regression

- If $j = y^{(i)}$, the gradients with respect to $\Theta_{y^{(i)}}$:

$$\nabla_{\Theta_j} = -x^{(i)} + \frac{\exp(\Theta_j^T x^{(i)})}{\sum_{l=1}^{k} \exp(\Theta_l^T x^{(i)})} x^{(i)} = -(1 - \frac{\exp(\Theta_j^T x^{(i)})}{\sum_{l=1}^{k} \exp(\Theta_l^T x^{(i)})}) x^{(i)}$$

- If $j \neq y^{(i)}$, the gradients with respect to Θ_j:

$$\nabla_{\Theta_j} = \frac{\exp(\Theta_j^T x^{(i)})}{\sum_{l=1}^{k} \exp(\Theta_l^T x^{(i)})} x^{(i)}$$

- The parameters could be updated via gradient descent

$$\Theta_j := \Theta_j - \alpha \nabla_{\Theta_j}$$

- NOTE: Even for a single sample of a class, parameters of all classes would be updated
  - The weights of the ground-truth class $\Theta_{y^{(i)}}$ would approach the input features $x^{(i)}$. If the confidence $\Theta_{y^{(i)}}^T x^{(i)}$ is **high**, the approaching would be **small**
  - The weights of other classes Θ_j would be **pushed away** from the input features $x^{(i)}$. If the confidence $\Theta_j^T x^{(i)}$ is **high**, i.e., the magnitude of pushing away would be **high**

Relation to logistic regression

- In the special case $k = 2$, one can show that softmax regression reduces to logistic regression

$$h(x) = \frac{1}{\exp(\Theta_1^T x) + \exp(\Theta_2^T x)} \begin{bmatrix} \exp(\Theta_1^T x) \\ \exp(\Theta_2^T x) \end{bmatrix}$$

- NOTE: This hypothesis is overparameterized. We can subtract Θ_1 from each parameter (equivalent to multiplying the same value on the numerator and denominator)

$$\begin{aligned} h(x) &= \frac{1}{\exp(0^T x) + \exp((\Theta_2 - \Theta_1)^T x)} \begin{bmatrix} \exp(0^T x) \\ \exp((\Theta_2 - \Theta_1)^T x) \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1 + \exp((\Theta_2 - \Theta_1)^T x)} \\ \frac{\exp((\Theta_2 - \Theta_1)^T x)}{1 + \exp((\Theta_2 - \Theta_1)^T x)} \end{bmatrix} = \begin{bmatrix} \frac{1}{1 + \exp((\Theta_2 - \Theta_1)^T x)} \\ 1 - \frac{1}{1 + \exp((\Theta_2 - \Theta_1)^T x)} \end{bmatrix} \end{aligned}$$

- Substitute $\Theta_2 - \Theta_1$ with a single parameter vector $-\theta$, gives us the same probability as logistic regression