Introduction to Machine Learning

Neural Networks

林彥宇 教授
Yen-Yu Lin, Professor
國立陽明交通大學 資訊工程學系
Computer Science, National Yang Ming Chiao Tung University

Some slides are modified from S.-J. Wang, H.-T. Chen, V. Khalidov, and M. Hansard

Linear model for regression or classification

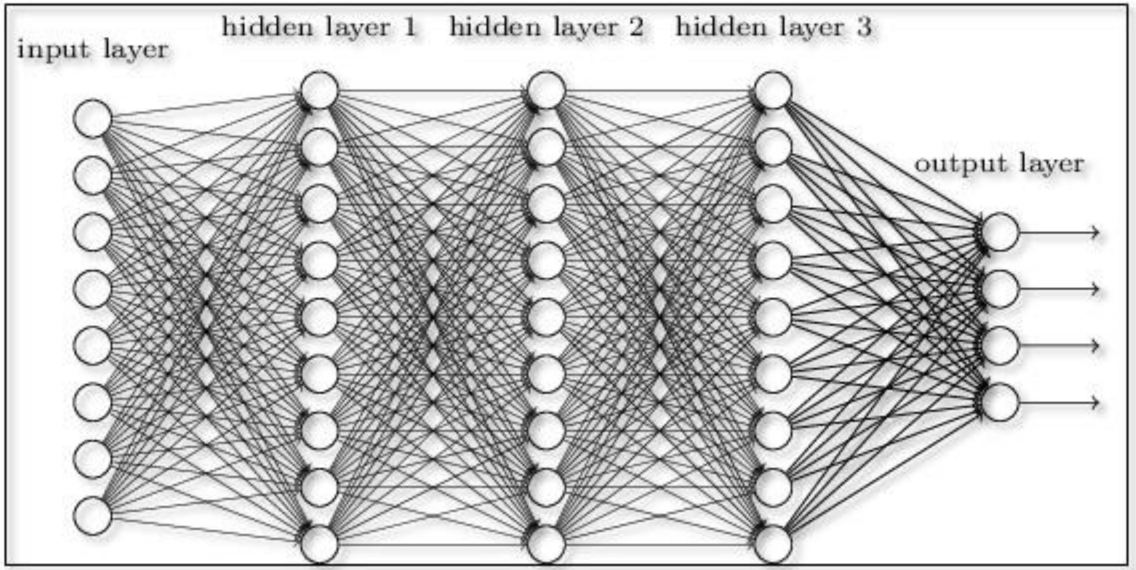
- A linear model for regression or classification

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

  - Decision based on a linear combination of fixed nonlinear basis functions
  - f is an identity function for regression
  - f is a nonlinear activation function for classification
    - Logistic sigmoid or softmax function

Linear model and neural networks

- Our goal is to extend the linear model by making
  - 1. The basis functions depend on parameters
    - Parametric basis functions
  - 2. Their parameters learnable during training
- The goal leads to the basic neural network model



國立交通大學 National Chiao Tung University

https://www.houseofbots.com/news-detail/1442-1-what-is-deep-learning-and-neural-network

Activations

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \boxed{\phi_j(\mathbf{x})}\right)$$

Examples of basis functions

- Polynomial basis function: taking the form of powers of x

$$\phi_j(x) = x^j$$

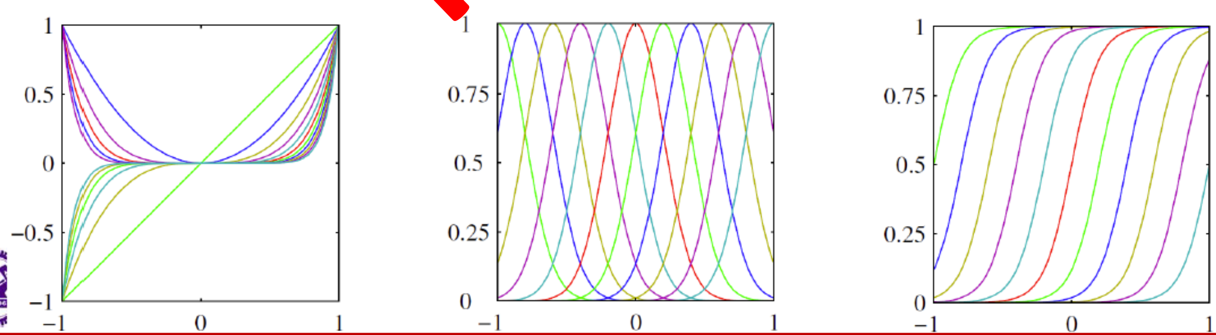
- Gaussian basis function: governed by μ_j and s
  - μ_j governs the location while s governs the scale

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

- Sigmoidal basis function: governed by μ_j and s

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right) \quad \text{where} \quad \sigma(a) = \frac{1}{1+\exp(-a)}$$

Fixed!



國立交通大學
National Chiao Tung University

Activations

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \boxed{\phi_j(\mathbf{x})}\right)$$

- Construct M linear combinations of the inputs $x_1, \ldots, x_D$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

  - where a_j is the activation for $j = 1, 2, \ldots, M$
  - $\{w_{ji}^{(1)}\}_{i=1}^{D}$ are the weights. Superscript (1) indicates that these parameters are in the first layer of neural networks
  - $w_{j0}^{(1)}$ is the bias
  - Each activation is nonlinearly transformed by using a differentiable, nonlinear activation function h, i.e.,

$$z_j = h(a_j)$$

- $\{z_j = h(a_j)\}_{j=1}^{M}$ are called hidden units

國立交通大學
National Chiao Tung University

Output unit activation

$$y(\mathbf{x}, \mathbf{w}) = f\left(\boxed{\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})}\right)$$

- The hidden units $\{z_j = h(a_j)\}_{j=1}^{M}$ are linearly combined in the second layer of neural networks
- Suppose there are K outputs in the neural networks. We have

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

  - where a_k is the output activation for $k = 1, 2, \ldots, K$
  - $\{w_{kj}^{(2)}\}_{j=1}^{M}$ are the weights. Superscript (2) indicates that these parameters are in the second layer of neural networks
  - $w_{k0}^{(2)}$ is the bias
- a_k is further transformed by output activation function

$$y_k = \sigma(a_k)$$

Neural networks for regression and classification

- Output activation a_k is further transformed by output activation function

$$y_k = \sigma(a_k)$$

- $\{y_k\}_{k=1}^{K}$ are the final outputs of the neural networks
- For regression, $\sigma(\cdot)$ is the identity function
- For two-class classification, $\sigma(\cdot)$ is the logistic sigmoid function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

- For multiclass classification, $\sigma(\cdot)$ is the softmax function

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

國立交通大學
National Chiao Tung University

Two-layer neural networks

- The two-layer neural network model

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

a_j

z_j

a_k

y_k

  - where $\mathbf{w}$ is the set of all weight and bias parameters

- The bias parameters can be absorbed into weight parameters by using one additional input $x_0 = 1$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

Feed-forward neural networks

- Evaluating the following equation is called forward propagation

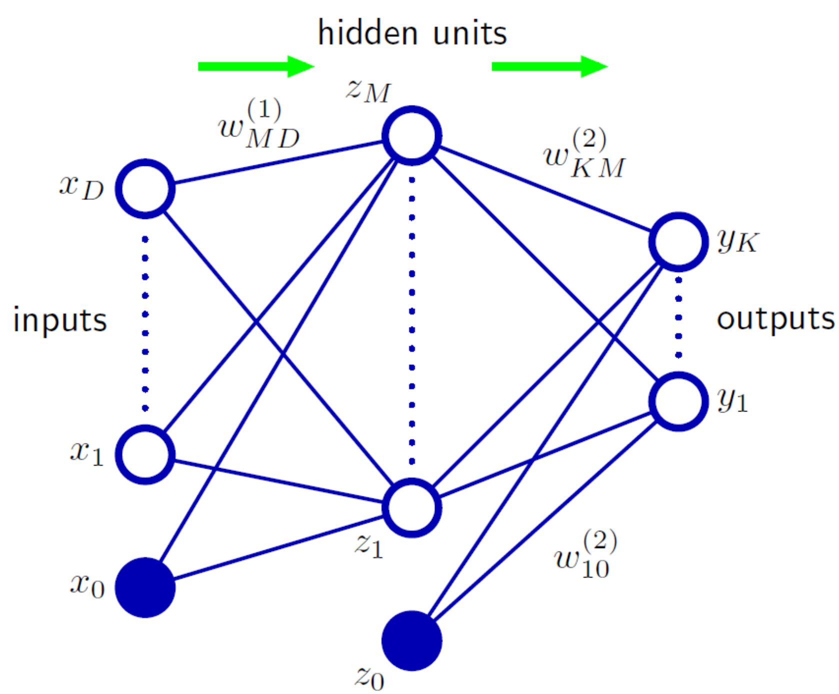
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

Network Diagram

Nodes: Input, hidden, and output variables

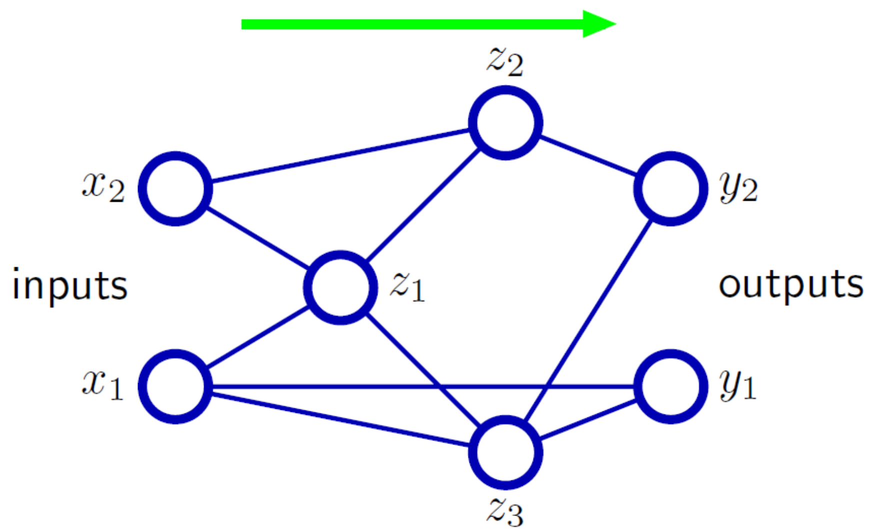
Links: Weights and biases

Arrows: Propagation direction



Generalizations

- There may be more than one layer of hidden units
  - Deep learning
- Individual units need not be fully connected to the next layer
  - Convolutional neural networks
- Individual links may skip over one or more subsequent layers
  - Skip connections



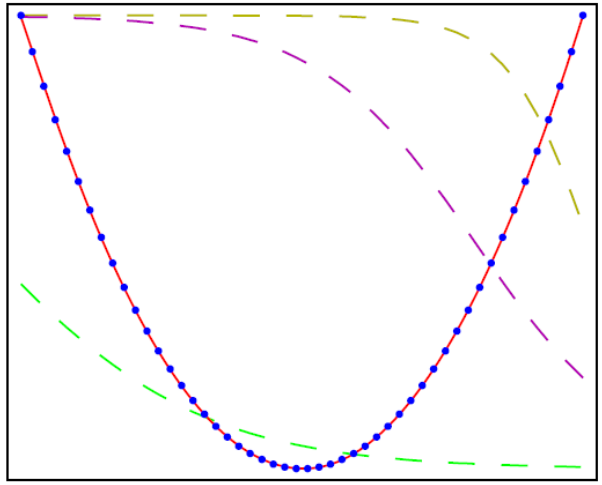
國立交通大學
National Chiao Tung University

Neural networks as universal approximators

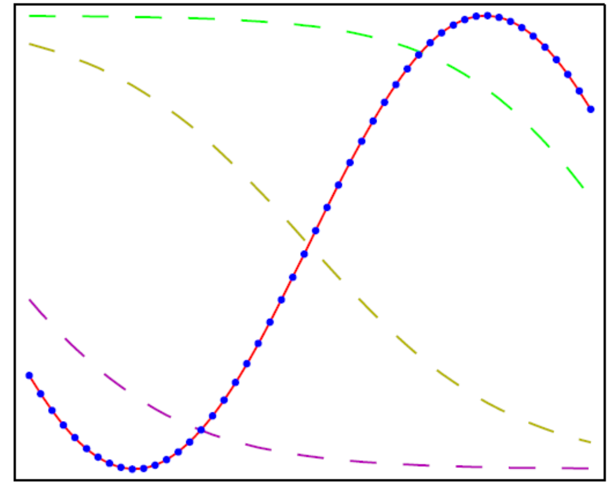
Points: training data

Dashed curves: Outputs of three hidden units

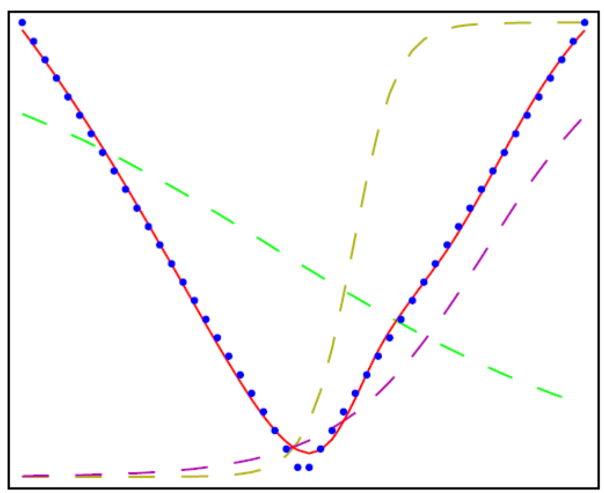
Curve: Prediction by the NN



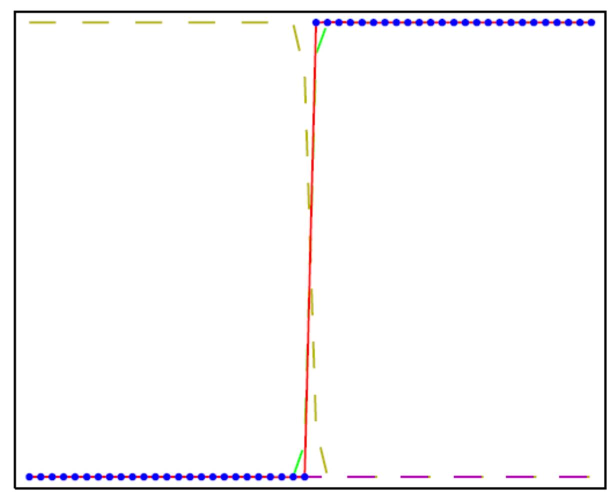
$f(x) = x^2$



$f(x) = \sin(x)$



$f(x) = |x|$

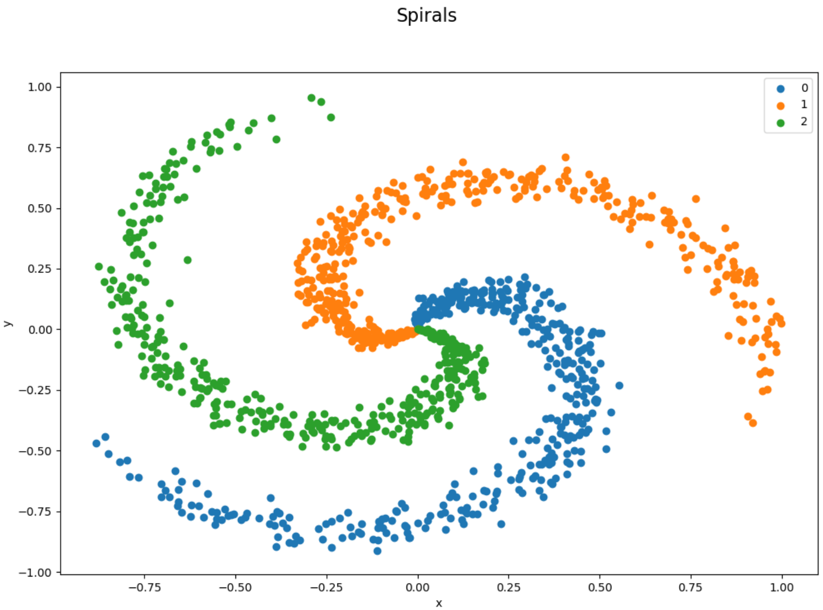


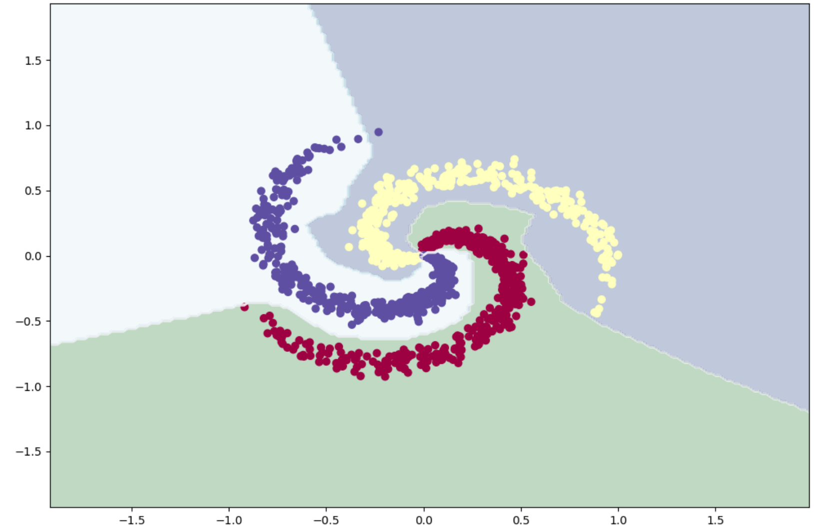
Heaviside step function

National Chiao Tung University

Neural networks for classification

- 3-class classification
- 2-layer neural networks with 64 hidden units





https://www.annytab.com/neural-network-classification-in-python/

Network training

- Given a set of training data $\{\mathbf{x}_n\}$ where $n = 1, 2, \ldots, N$, together with a corresponding set of target vectors $\{\mathbf{t}_n\}$, we can learn the neural networks by minimizing

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2$$

- Let's consider how to train the networks by giving a probabilistic interpretation to the network output

Neural networks for 1D regression

- We aim to minimize the error between $y(\mathbf{x}_n, \mathbf{w})$ and t_n
- We assume that the target is a scalar-valued function, which is normally distributed around the prediction

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

  - where $y(\mathbf{x}, \mathbf{w})$ is the prediction by neural networks and β^{-1} is the variance

- Suppose data are i.i.d. The likelihood is

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

  - where $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and $\mathbf{t} = \{t_1, \ldots, t_N\}$

ML solution for 1D regression

- Taking the negative logarithm, we get negative log likelihood

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w})-t_n\}^2-\frac{N}{2}\ln\beta+\frac{N}{2}\ln(2\pi)$$

- The maximum likelihood solution for $\mathbf{w}$ is equivalent to minimizing the sum-of-squares error

$$E(\mathbf{w})=\frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w})-t_n\}^2$$

- Does setting the gradient of $E(\mathbf{w})$ to zero work?
  - No closed-form solution

ML solution for 1D regression

- Optimization by using gradient descent, stochastic gradient descent, or Newton-Raphson iterative optimization scheme
- The nonlinearity of $y(\mathbf{x}_n, \mathbf{w})$ makes $E(\mathbf{w})$ to be nonconvex
- In practice, local minima of the negative log likelihood may be found
- After having found $\mathbf{w}_{\mathrm{ML}}$, the value of β can be found by minimizing the negative log likelihood

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{ML}}) - t_n\}^2$$

國立交通大學
National Chiao Tung University

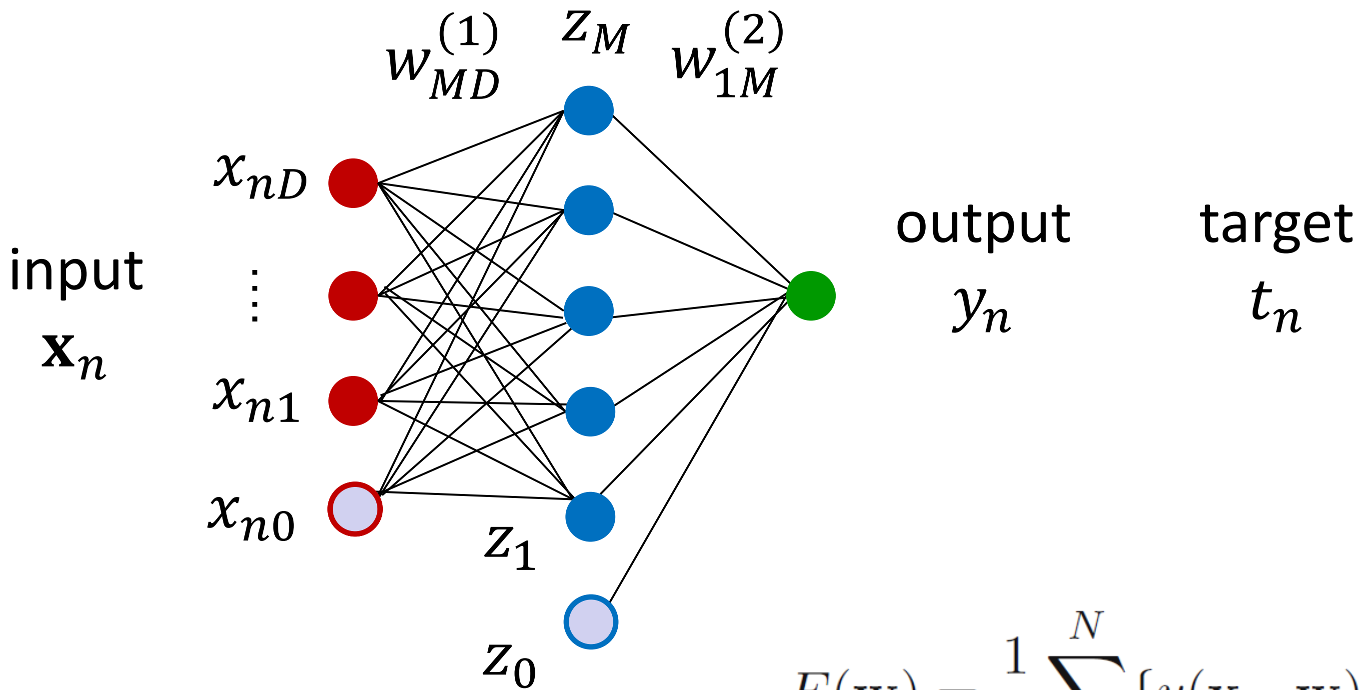
ML solution for 1D regression

- After getting $\mathbf{w}_{\mathrm{ML}}$ and β_{ML}, we can predict the distribution of the target value t for an input testing data point $\mathbf{x}$ via

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

ML solution for 1D regression

- Two-layer neural networks for one-dimensional regression



$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

Neural networks for multi-dimensional regression

- Neural networks can be used for K-dimensional regression
- Construct neural networks with K outputs
- Make the following assumption

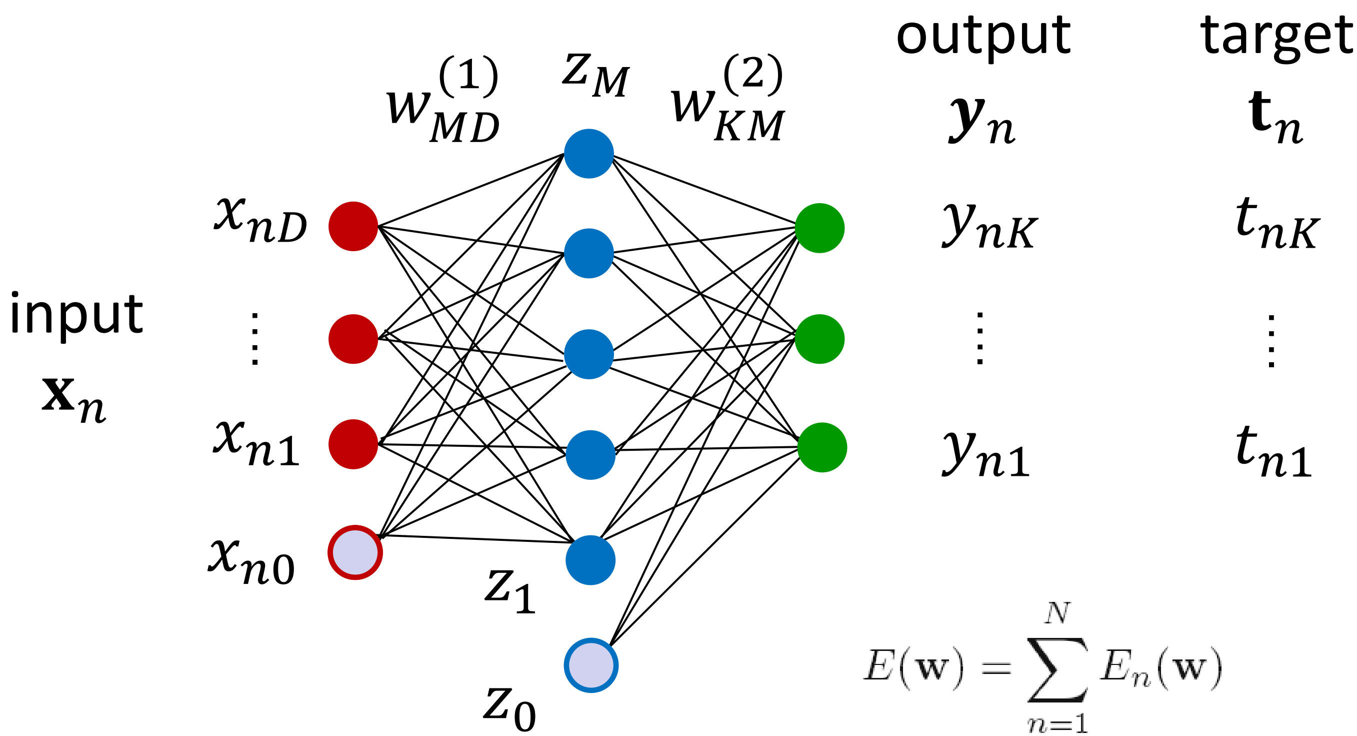
$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \beta^{-1}\mathbf{I}\right)$$

- We can use maximum likelihood solution, which is equivalent to minimizing the sum-of-squares errors, to get $\mathbf{w}_{\mathrm{ML}}$
- Similarly given $\mathbf{w}_{\mathrm{ML}}$, the optimal β_{ML} is obtained

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{NK} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}_{\mathrm{ML}}) - \mathbf{t}_n\|^2$$

Neural networks for multi-dimensional regression

- Two-layer neural networks for K-dimensional regression



$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

$$E_n(\mathbf{w}) = \frac{1}{2}\|\mathbf{y}_n - \mathbf{t}_n\|^2 = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2$$

Neural networks for binary classification

- Neural networks can be used for classification
- Given a set of training data $\{\mathbf{x}_n\}$ where $n = 1, 2, \dots, N$, together with a corresponding set of target labels $\{t_n\}$, where $t_n = 1$ denotes class C_1 and $t_n = 0$ denotes class C_2
- Construct (two-layer) neural networks having a single output whose activation function is a logistic sigmoid

$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)}$$

  - where $0 \leqslant y(\mathbf{x}, \mathbf{w}) \leqslant 1$
  - $y(\mathbf{x}, \mathbf{w})$ is the conditional probability $p(\mathcal{C}_1|\mathbf{x})$
  - The conditional probability $p(\mathcal{C}_2|\mathbf{x})$ is given by $1 - y(\mathbf{x}, \mathbf{w})$

ML solution for binary classification

- Regression: the target is a real-valued function, which is normally distributed around the prediction

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

- Classification: the conditional distribution of a target given its input is a Bernoulli distribution of the form

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \left\{1 - y(\mathbf{x}, \mathbf{w})\right\}^{1-t}$$

ML solution for binary classification

- When using ML optimization, we minimize the negative log likelihood, here called cross-entropy error

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

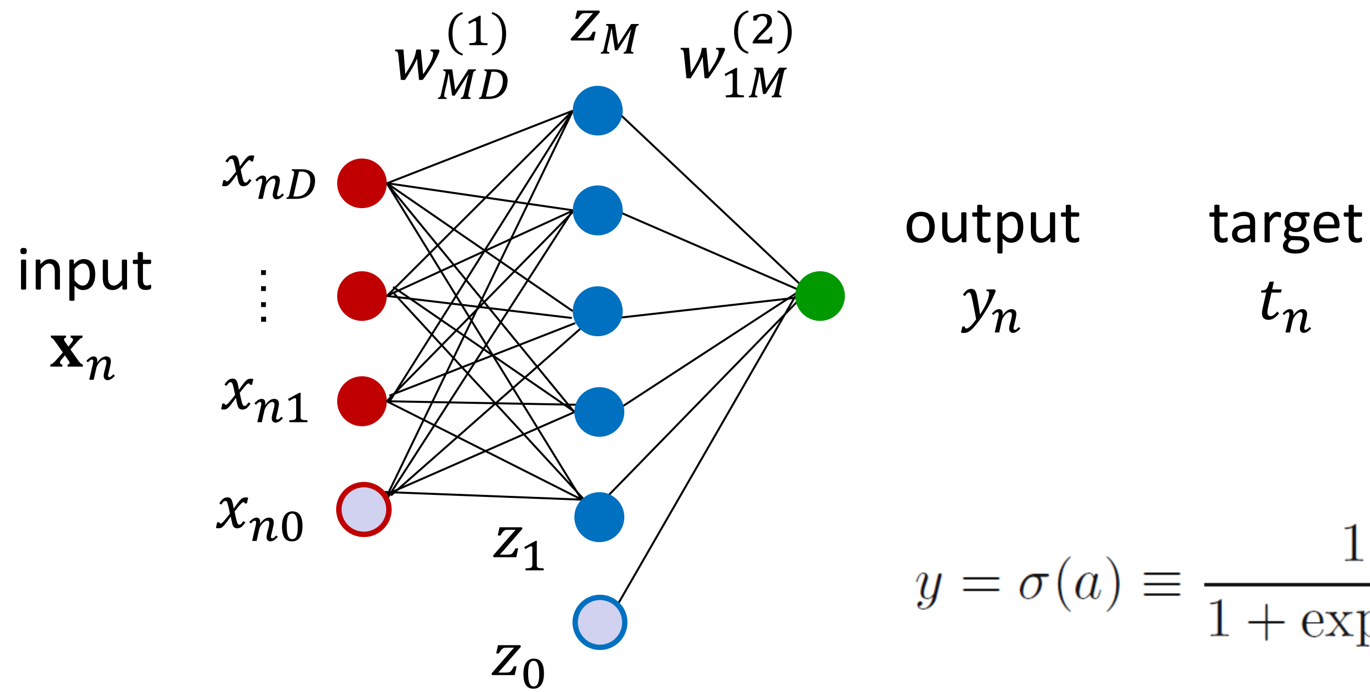
  - where y_n denotes $y(\mathbf{x}_n, \mathbf{w})$

- Optimize $\mathbf{w}$ by using gradient descent or its variant

- After getting $\mathbf{w}_{\mathrm{ML}}$, binary classification is carried out by

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}$$

ML solution for binary classification



$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)}$$

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

国立交通大學
National Chiao Tung University

Neural networks for multi-class classification

- Neural networks can be extended to K-class classification
- Given a set of training data $\{\mathbf{x}_n\}$ where $n = 1, 2, \ldots, N$, together with a corresponding set of target vectors $\{\mathbf{t}_n\}$, where $\mathbf{t}_n$ is encoded by using 1-of-K coding scheme
- Construct (two-layer) neural networks having K outputs and use softmax as the activation function

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

  - where $0 \leqslant y_k \leqslant 1$ and $\sum_k y_k = 1$

ML solution for multi-class classification

- The negative log likelihood or the cross-entropy error is

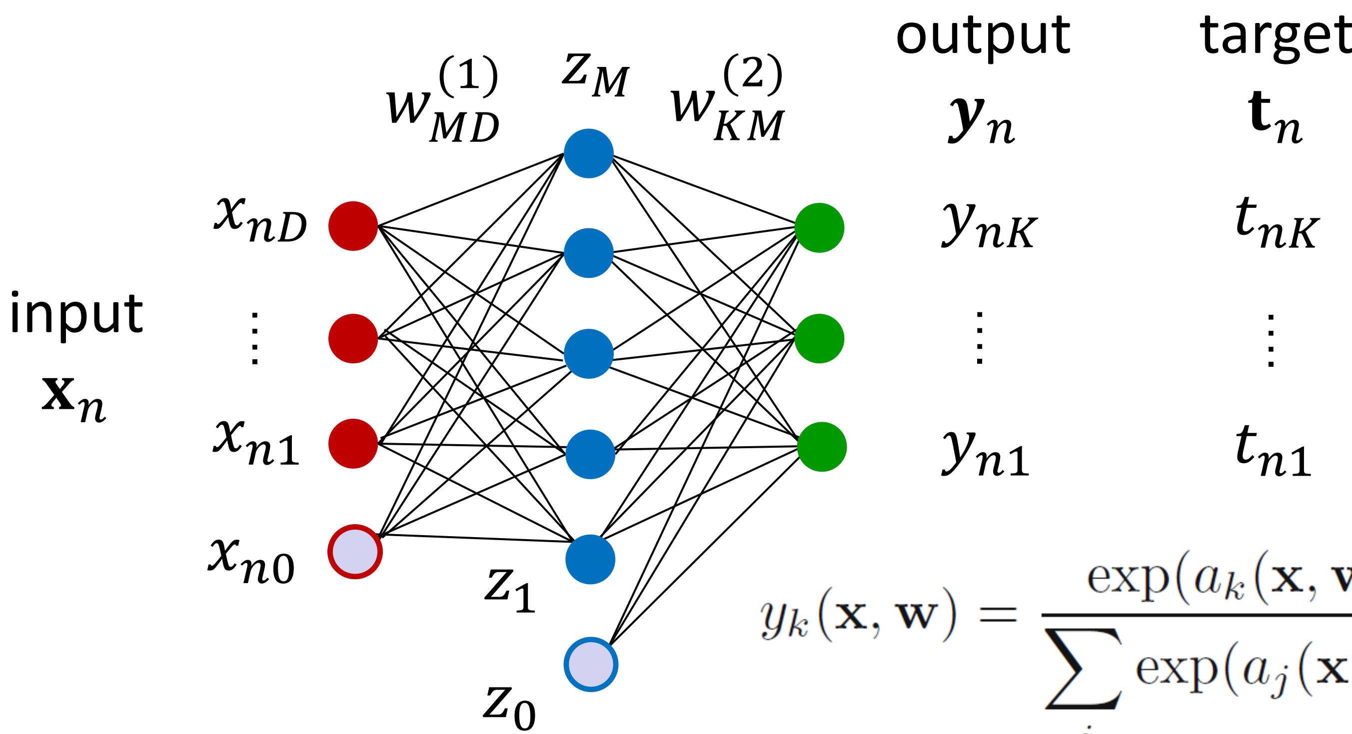
$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

- Optimize $\mathbf{w}$ by using gradient descent or its variant
- After getting $\mathbf{w}_{\mathrm{ML}}$, multi-class classification is carried out by using the softmax function

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

ML solution for multi-class classification

- Two-layer neural networks for K-class classification



$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

國立交通大學 National Chiao Tung University

Gradient descent

- The simplest approach is to update $\mathbf{w}$ by a displacement in the negative gradient direction

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

  - This is a steepest descent algorithm
  - $\eta > 0$ is the learning rate
  - This is a batch method, as evaluation of ∇E involves the entire data set
  - A range of starting points $\{\mathbf{w}^{(0)}\}$ may be needed, in order to find a satisfactory minimum

Stochastic gradient descent

- Stochastic gradient descent (or called sequential gradient descent) has proved useful in practice when training neural networks on a large data set
- The error function needs to comprise a sum of terms, one for each data point, i.e.,

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

  - Sum-of-squares error for regression

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

  - Cross-entropy error for classification

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$$

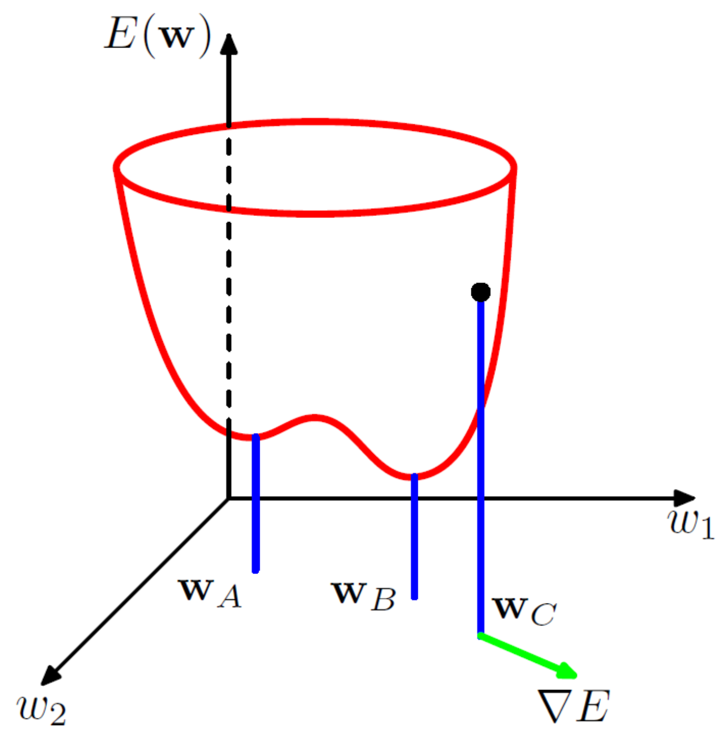
Stochastic gradient descent

- Stochastic gradient descent makes an update to the weight vector based on one data point at a time

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

Geometric view of gradient descent

- The error function $E(\mathbf{w})$ is a surface sitting over the weight space
- $\mathbf{w}_A$ is a local minimum
- $\mathbf{w}_B$ is a global minimum
- At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector ∇E



国立交通大學
National Chiao Tung University

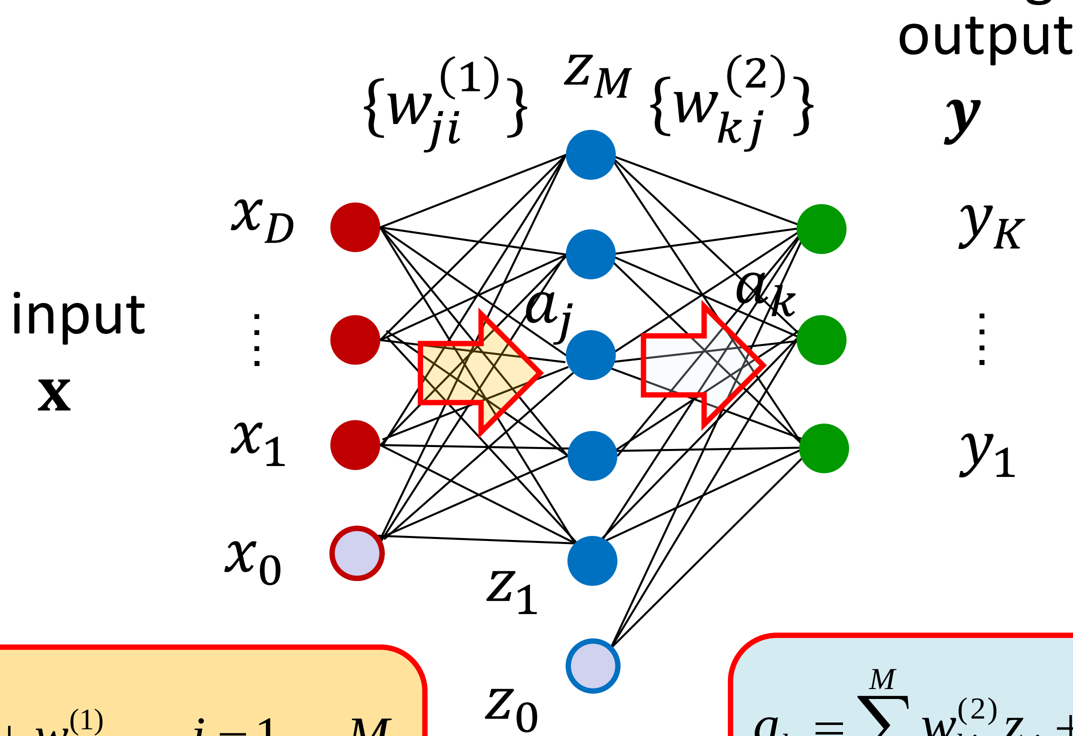
Error backpropagation

- The computational cost of gradient descent mainly lies in the evaluation of gradient at each iteration
  - $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$
  - The dimension of gradient is the number of learnable parameters

- In feed-forward neural networks, the gradient of an error function $E(\mathbf{w})$ can be efficiently evaluated via an algorithm called error backpropagation

Feed-forward neural networks

- Two-layer feed-forward neural networks for regression



$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \qquad j = 1,\ldots,M$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \qquad k = 1,\ldots,K$$

$$y_k = a_k$$

Error backpropagation

- Variables/Activations dependency:

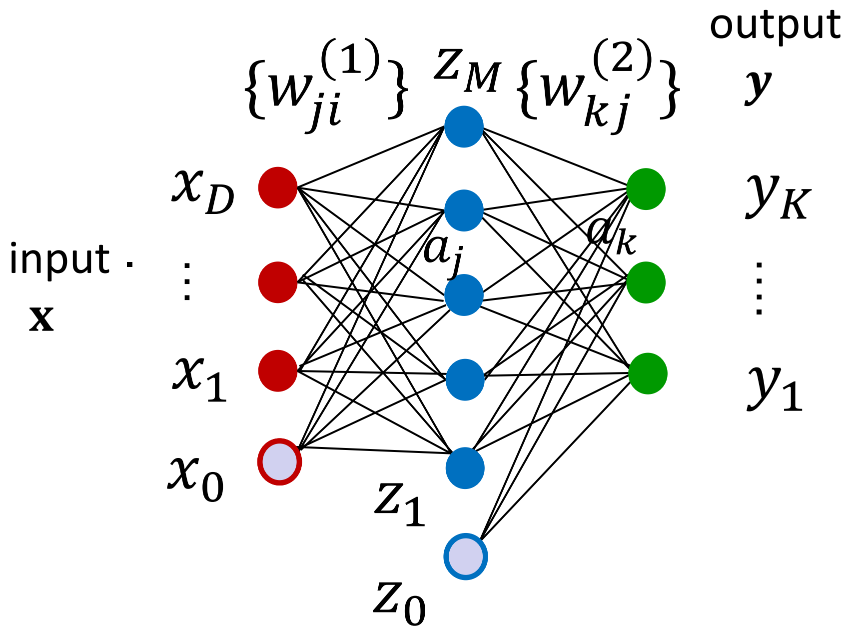
$$\{x_i\} \rightarrow \{w_{ji}^{(1)}\} \rightarrow \{a_j\} \rightarrow \{z_j\} \rightarrow \{w_{kj}^{(2)}\} \rightarrow \{a_k\} \rightarrow \{y_k\} \rightarrow E$$

- Our goal in gradient computation:

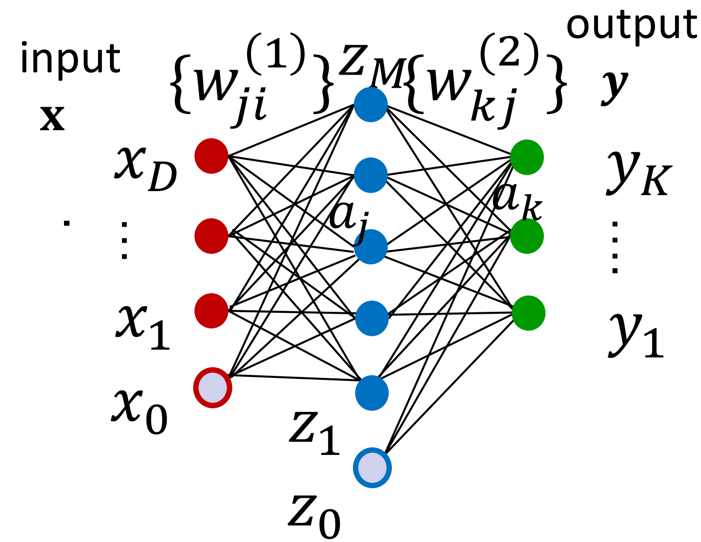
$\dfrac{\partial E}{\partial w_{kj}^{(2)}}$ and $\dfrac{\partial E}{\partial w_{ji}^{(1)}}$

- In backpropagation, we also need to compute

$\delta_k = \dfrac{\partial E}{\partial a_k}$ and $\delta_j = \dfrac{\partial E}{\partial a_j}$



Error backpropagation



- Stochastic gradient descent

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

- Multi-dimensional regression

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \qquad j = 1, \ldots, M$$

$$z_j = h(a_j)$$

Hidden layer

$$\delta_j \equiv \frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j} = h'(a_j) \sum_k w_{kj}^{(2)} \delta_k$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \quad k = 1, \ldots, K$$

$$y_k = a_k$$

Output layer

$$\delta_k \equiv \frac{\partial E}{\partial a_k} = y_k - t_k$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Error function

Error backpropagation

- Variables/Activations dependency:

$$\{x_i\} \rightarrow \{w_{ji}^{(1)}\} \rightarrow \{a_j\} \rightarrow \{z_j\} \rightarrow \{w_{kj}^{(2)}\} \rightarrow \{a_k\} \rightarrow \{y_k\} \rightarrow E$$

$$\delta_j = h'(a_j) \sum_k w_{kj}^{(2)} \delta_k$$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \quad j = 1, \ldots, M$$

$$z_j = h(a_j)$$

Hidden layer

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \quad k = 1, \ldots, K$$

$$y_k = a_k$$

Output layer

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Error function

$$\delta_k = y_k - t_k$$

Error backpropagation

- Variables/Activations dependency:

$$\{x_i\} \rightarrow \{w_{ji}^{(1)}\} \rightarrow \{a_j\} \rightarrow \{z_j\} \rightarrow \{w_{kj}^{(2)}\} \rightarrow \{a_k\} \rightarrow \{y_k\} \rightarrow E$$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \qquad j = 1, \ldots, M$$

$$z_j = h(a_j)$$

Hidden layer

$$\delta_j = h'(a_j) \sum_k w_{kj}^{(2)} \delta_k$$

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \qquad k = 1, \ldots, K$$

$$y_k = a_k$$

Output layer

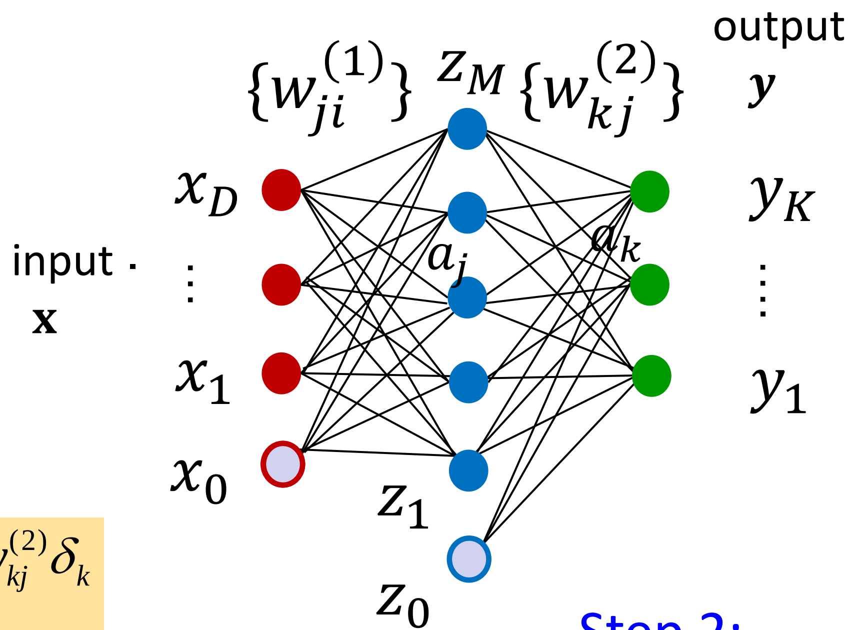
$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Error function

$$\delta_k = y_k - t_k$$

A review of error backpropagation



Step 1:

$$\delta_k = y_k - t_k$$

Step 2:

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Step 3:

$$\delta_j = h'(a_j) \sum_k w_{kj}^{(2)} \delta_k$$

Step 4:

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

Error backpropagation for other tasks

- Step 1: $\delta_k \equiv \dfrac{\partial E}{\partial a_k} = \dfrac{\partial E}{\partial y_k}\dfrac{\partial y_k}{\partial a_k}$

$$E(\mathbf{w}) = \begin{cases} \dfrac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2 & \text{regression} \\ -\{t \ln y(\mathbf{x},\mathbf{w}) + (1-t)\ln(1-y(\mathbf{x},\mathbf{w}))\} & \text{binary classification} \\ -\sum_{k=1}^{K} t_k \ln y_k(\mathbf{x},\mathbf{w}) & \text{multi-calss classification} \end{cases}$$

$$y_k = a_k \qquad \text{regression}$$

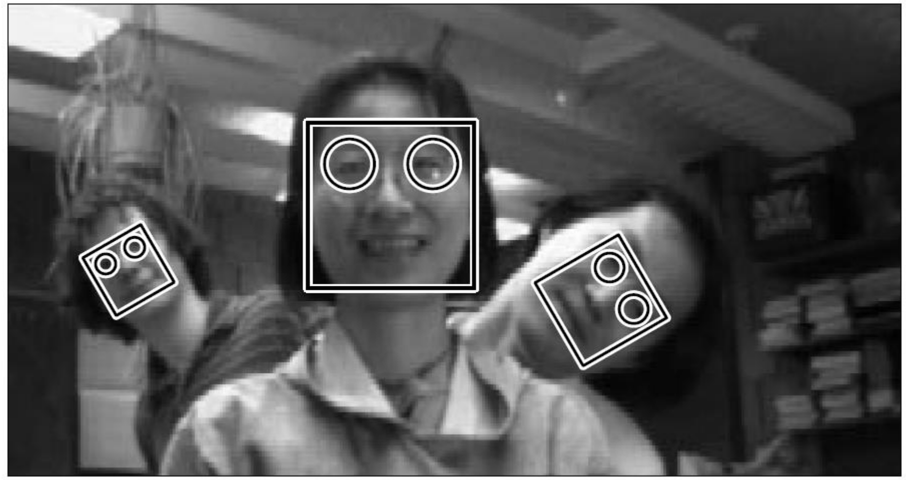
$$y = \frac{1}{1+e^{-a}} \text{ binary classification}$$

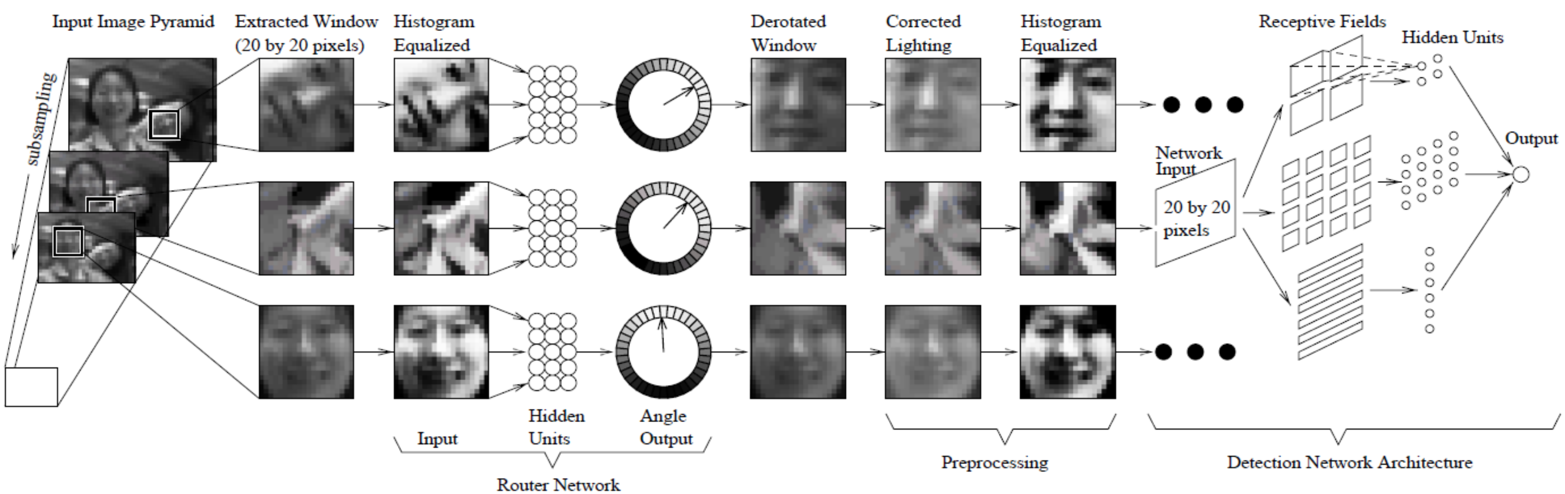
$$y_k = \frac{e^{a_k}}{\sum_j e^{a_j}} \qquad \text{multi}-\text{class classification}$$

- Steps 2 ~ 4 remain unchanged

Neural networks' applications

- Face detection

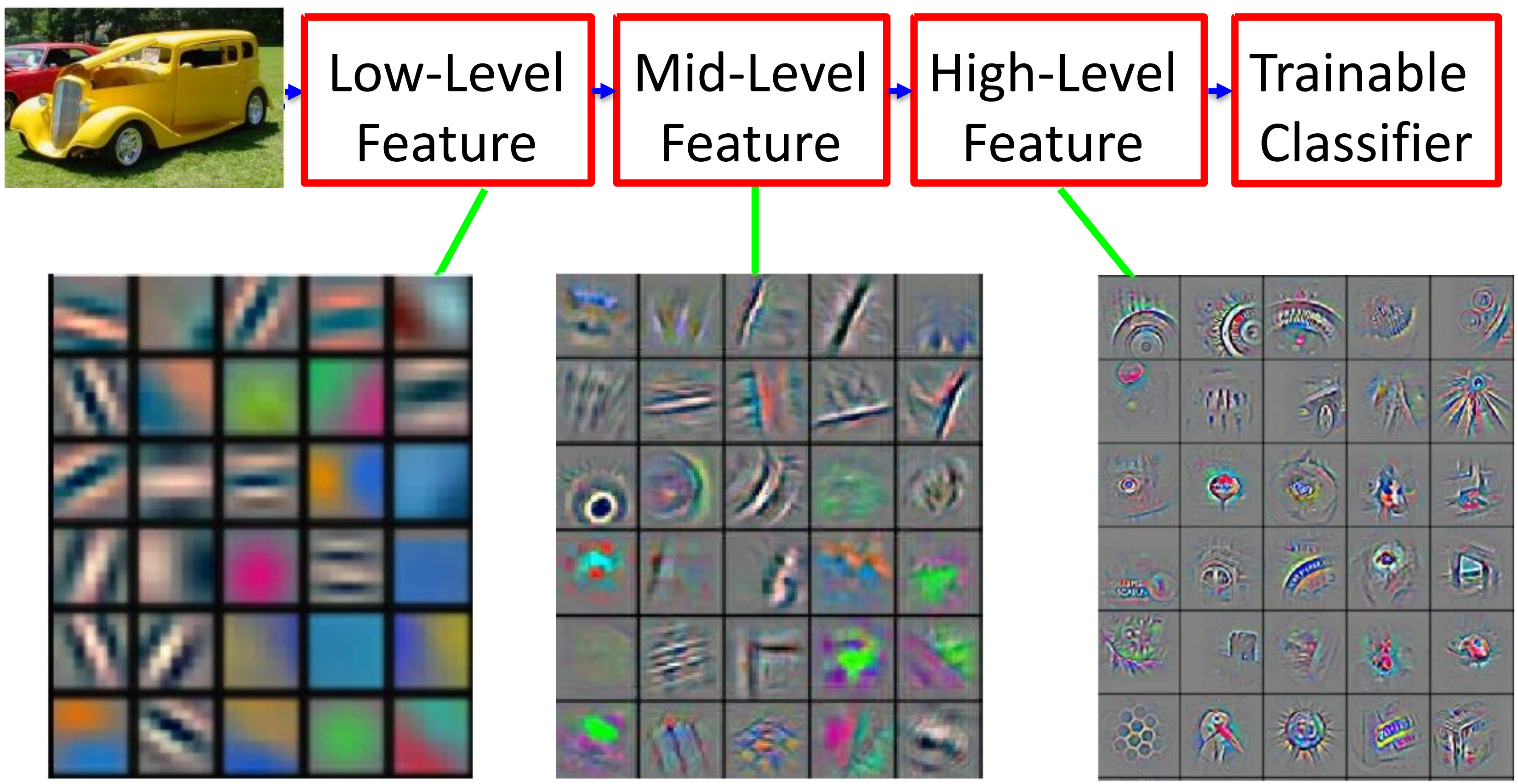




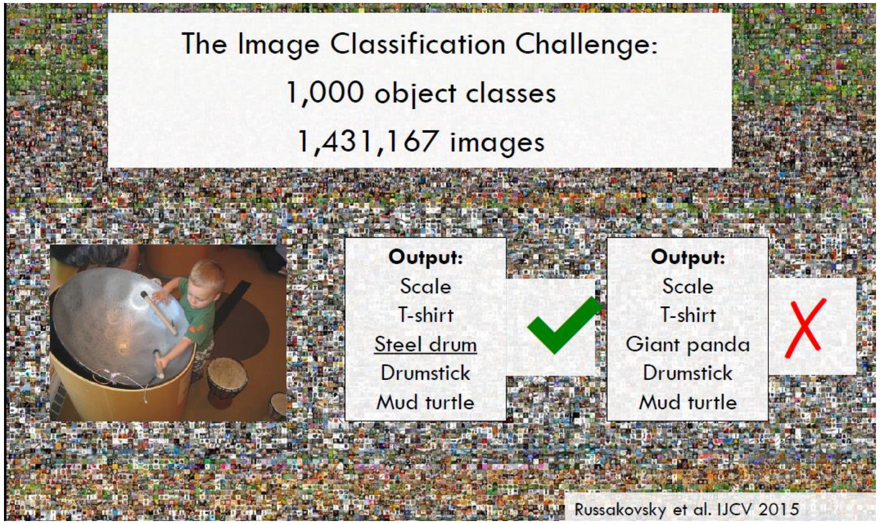
Rowley et al.

國立交通大學 National Chiao Tung University

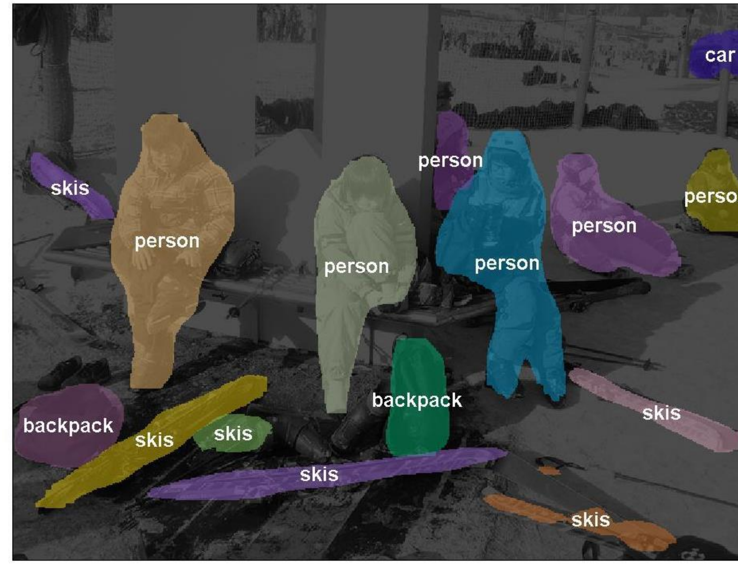
Convolutional neural networks



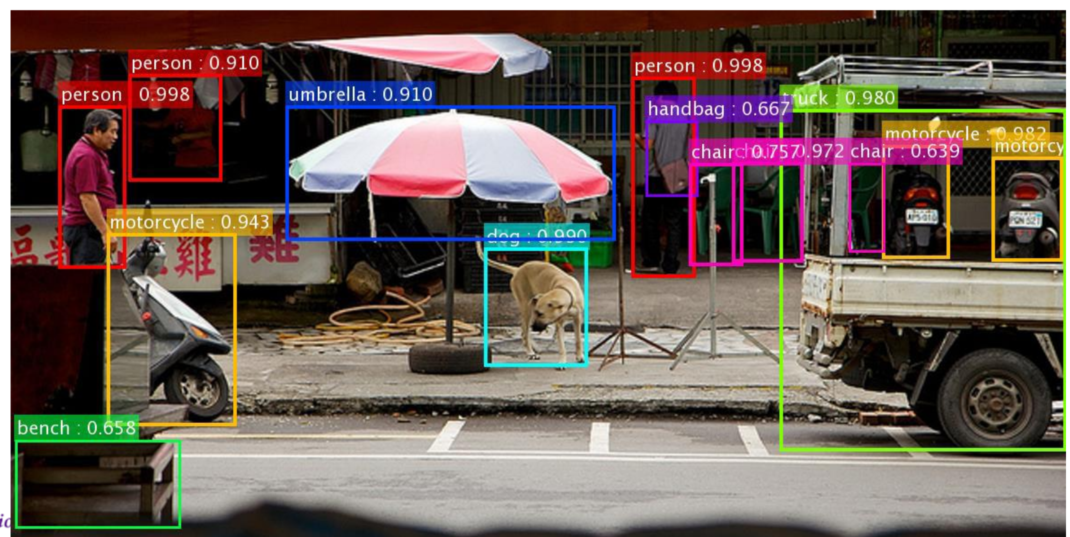
Convolutional neural networks’ applications



object recognition



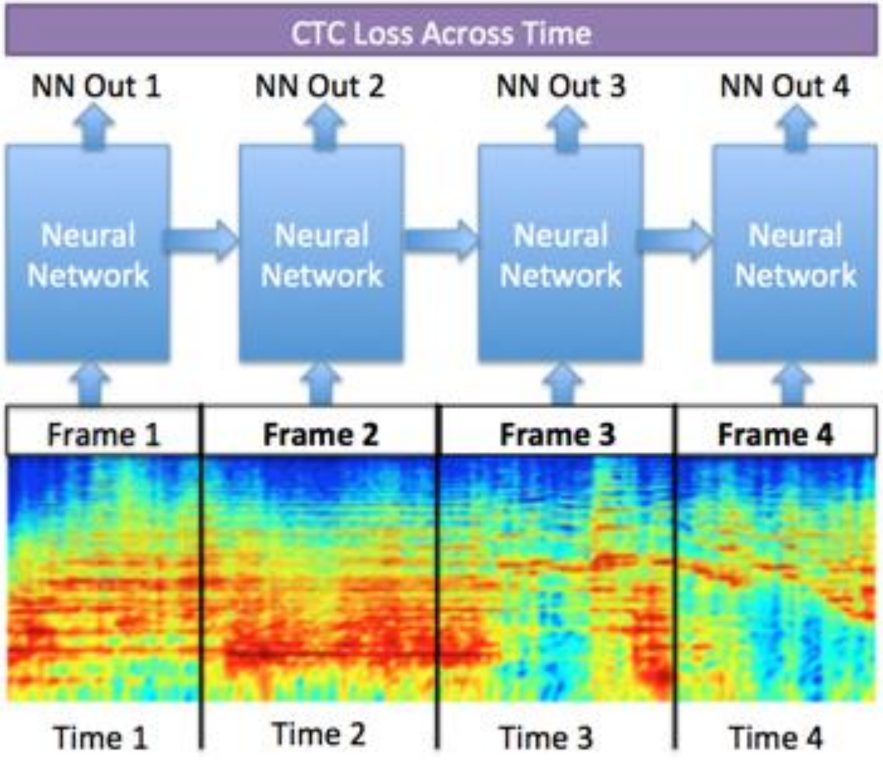
object segmentation



object detection

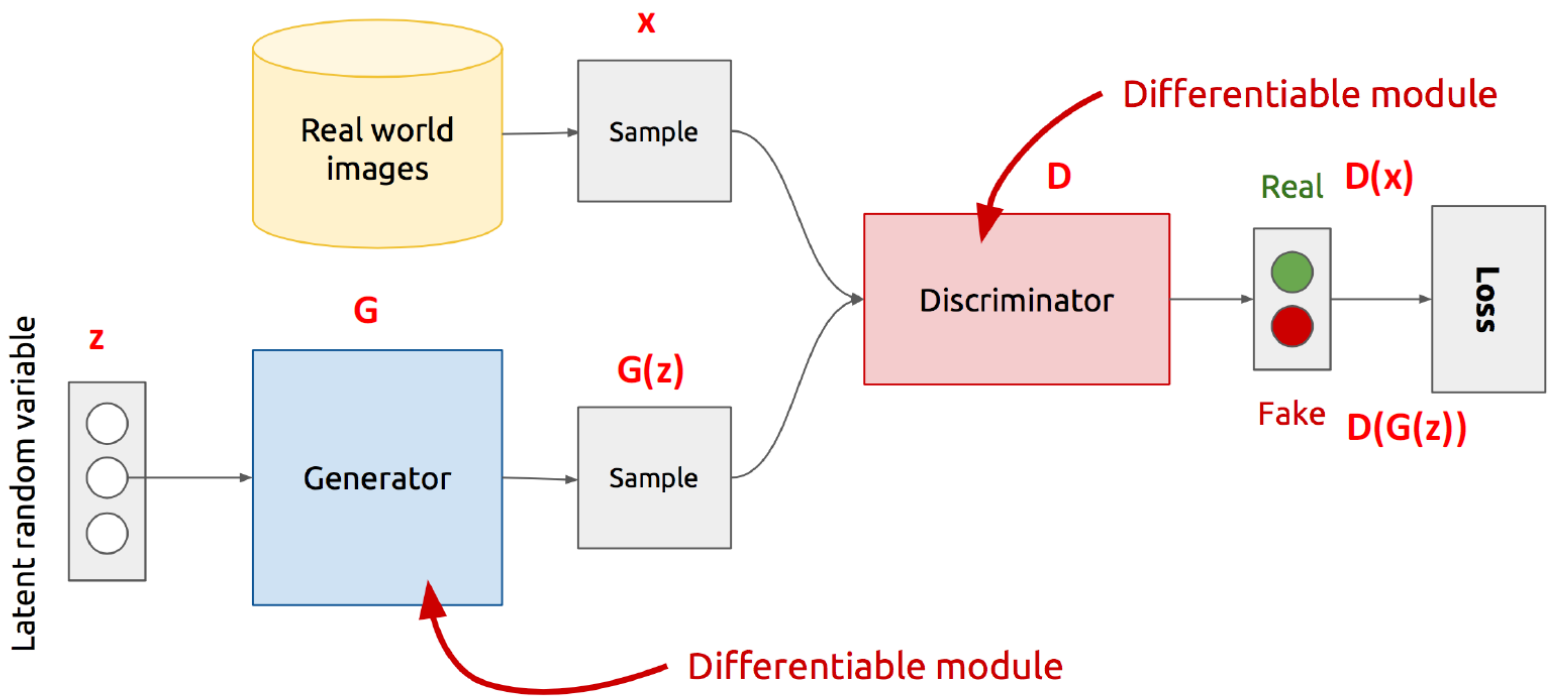
Recurrent neural networks

- Speech recognition



https://gab41.lab41.org/speech-recognition-you-down-with-ctc-8d3b558943f0

Generative adversarial networks



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

國立交通大學 National Chiao Tung University

Generative adversarial networks' applications

Karras et al.

Wang et al.

References

- Chapters 5.1, 5.2, and 5.3 in the PRML textbook

Thank You for Your Attention!

Yen-Yu Lin (林彥宇)

Email: lin@cs.nycu.edu.tw

URL: https://www.cs.nycu.edu.tw/members/detail/lin

國立交通大學
National Chiao Tung University