Feedforward Neural Networks

Dit-Yan Yeung

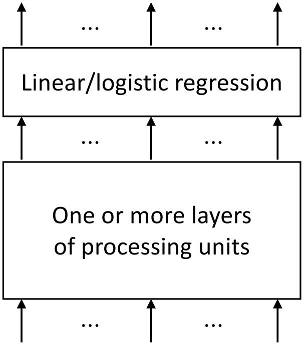
Department of Computer Science and Engineering
Hong Kong University of Science and Technology

COMP 4211: Machine Learning (Fall 2022)

Artificial Neural Networks

- Early research in artificial neural networks was inspired by findings from neuroscience, but subsequent development has mostly been guided by mathematical and computational considerations.
- Machine learning researchers and practitioners regard artificial neural networks as computational models for machine learning.
- Two major types of artificial neural networks:
  - Feedforward neural networks: networks without loops
  - Recurrent neural networks: networks with loops
- We will consider feedforward neural networks in this topic and recurrent neural networks in a later topic.

Layered Extension of Linear or Logistic Regression

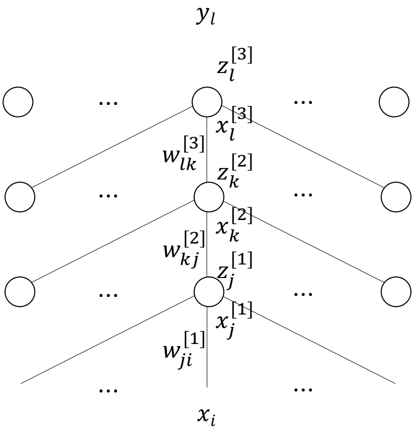


- A feedforward neural network, a.k.a. multilayer perceptron (MLP), may be considered as an extension of linear or logistic regression.
- The input is transformed by one or more layers of processing units (a.k.a. neurons) before it is fed into the linear or logistic regression model which corresponds to the output layer of the feedforward neural network.

Universal Approximation

- An informal way of stating the universal approximation theorem is that, a feedforward neural network with sufficiently many sigmoid hidden units in only one layer can approximate any well-behaved function to arbitrary precision.
- Nevertheless, using more than one hidden layer may give a network that can approximate the same function using (exponentially) fewer parameters due to the high nonlinearity that a deeper network can induce. This higher parameter efficiency also makes deeper networks much faster to train.
- Deeper networks also mimic better the hierarchical organization of data in many real-world applications.

Notation for a 3-Layer Network



Notation for a 3-Layer Network (2)

- We consider here an illustrative example with two hidden layers to simplify the notation.
- The superscripts, e.g., $[1], [2]$, refer to the corresponding network layers.
- For each processing unit, its (summed) input is denoted by $x^{[*]}$ (such as $x_j^{[1]}$, $x_k^{[2]}$, $x_\ell^{[3]}$ for units of different layers) and its output is denoted by $z^{[*]}$ (such as $z_j^{[1]}$, $z_k^{[2]}$, $z_\ell^{[3]}$).
- The function relating the input and output of a processing unit is called the activation function or transfer function of the unit:

$$z_j^{[1]} = g_j^{[1]}(x_j^{[1]}), \qquad z_k^{[2]} = g_k^{[2]}(x_k^{[2]}), \qquad z_\ell^{[3]} = g_\ell^{[3]}(x_\ell^{[3]}).$$

- All the activation functions are nonlinear except for those in the output layer (i.e., $g_\ell^{[3]}(\cdot)$) when the network is for solving regression problems.
- No processing is done at the input so no processing units are needed.

Forward Computation

- Input to first hidden layer:

$$x_j^{[1]} = \sum_i w_{ji}^{[1]} x_i, \qquad z_j^{[1]} = g_j^{[1]}(x_j^{[1]}).$$

- First hidden layer to second hidden layer:

$$x_k^{[2]} = \sum_j w_{kj}^{[2]} z_j^{[1]}, \qquad z_k^{[2]} = g_k^{[2]}(x_k^{[2]}).$$

- Second hidden layer to output layer:

$$x_\ell^{[3]} = \sum_k w_{\ell k}^{[3]} z_k^{[2]}, \qquad z_\ell^{[3]} = g_\ell^{[3]}(x_\ell^{[3]}).$$

Loss Functions

- Squared loss function for regression problems:

$$L(\mathbf{W}; \mathcal{S}) = \frac{1}{2} \sum_{q=1}^{N} \sum_{\ell} \left(z_\ell^{[3](q)} - y_\ell^{(q)} \right)^2 = \frac{1}{2} \sum_{q=1}^{N} \sum_{\ell} \left(x_\ell^{[3](q)} - y_\ell^{(q)} \right)^2 .$$

  The constant $1/2$ is introduced to simplify the subsequent derivation.

- Cross-entropy loss function for classification problems:

$$L(\mathbf{W}; \mathcal{S}) = - \sum_{q=1}^{N} \sum_{\ell} y_\ell^{(q)} \log z_\ell^{[3](q)} = - \sum_{q=1}^{N} \sum_{\ell} y_\ell^{(q)} \log \mathrm{softmax}(x_\ell^{[3](q)}).$$

Backpropagation Learning Algorithm

- Gradient descent based on the gradients computed recursively in the backward direction starting from the output layer:

$$\Delta w_{\ell k}^{[3]} \propto -\frac{\partial L}{\partial w_{\ell k}^{[3]}}$$

$$\Delta w_{kj}^{[2]} \propto -\frac{\partial L}{\partial w_{kj}^{[2]}}$$

$$\Delta w_{ji}^{[1]} \propto -\frac{\partial L}{\partial w_{ji}^{[1]}}.$$

- This recursive way of gradient computation for gradient descent is called the backpropagation (BP) learning algorithm.
- More advanced gradient-based learning algorithms may also make use of the gradients computed this way.

Algorithm Sketch

Initialize network weights
repeat
    for each training example **x** **do**
        # Forward propagation
        predicted-output = neural-network-output(**x**)
        actual-output = label(**x**)
        Compute error terms (actual-output - predicted-output) at output units
        # Backward propagation
        Compute weight changes for last layer of weights
        Compute weight changes for second layer of weights
        Compute weight changes for first layer of weights
    end for
    Update network weights for all layers
until some stopping criterion is satisfied

Gradients of the Last Layer of Weights

- Gradients w.r.t. $\{w_{\ell k}^{[3]}\}$:

$$\frac{\partial L}{\partial w_{\ell k}^{[3]}} = \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial w_{\ell k}^{[3]}} = \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial x_{\ell}^{[3](q)}} \frac{\partial x_{\ell}^{[3](q)}}{\partial w_{\ell k}^{[3]}} = -\sum_{q=1}^{N} \delta_{\ell}^{[3](q)} \frac{\partial x_{\ell}^{[3](q)}}{\partial w_{\ell k}^{[3]}},$$

  where

$$L^{(q)} = \begin{cases} \frac{1}{2} \sum_{\ell} \left(z_{\ell}^{[3](q)} - y_{\ell}^{(q)} \right)^2 & \text{for regression} \\ -\sum_{\ell} y_{\ell}^{(q)} \log z_{\ell}^{[3](q)} & \text{for classification.} \end{cases}$$

- Derivation of $\delta_{\ell}^{[3](q)}$ for regression:

$$\delta_{\ell}^{[3](q)} = -\frac{\partial L^{(q)}}{\partial x_{\ell}^{[3](q)}} = -\sum_{m} \frac{\partial L^{(q)}}{\partial z_{m}^{[3](q)}} \frac{\partial z_{m}^{[3](q)}}{\partial x_{\ell}^{[3](q)}} = y_{\ell}^{(q)} - z_{\ell}^{[3](q)}.$$

Gradients of the Last Layer of Weights (2)

- Derivation of $\delta_\ell^{[3](q)}$ for classification:

$$\delta_\ell^{[3](q)} = -\frac{\partial L^{(q)}}{\partial x_\ell^{[3](q)}} = -\sum_m \frac{\partial L^{(q)}}{\partial z_m^{[3](q)}} \frac{\partial z_m^{[3](q)}}{\partial x_\ell^{[3](q)}}$$
$$= \sum_m \frac{y_m^{(q)}}{z_m^{[3](q)}} z_m^{[3](q)} (\delta_{m\ell} - z_\ell^{[3](q)})$$
$$= \sum_m y_m^{(q)} (\delta_{m\ell} - z_\ell^{[3](q)}) = y_\ell^{(q)} - z_\ell^{[3](q)}.$$

- $$\frac{\partial x_\ell^{[3](q)}}{\partial w_{\ell k}^{[3]}} = z_k^{[2](q)}.$$

- We regard $\{\delta_\ell^{[3](q)}\}$ as the error terms computed at the output layer.

Gradients of the Second Layer of Weights

- Gradients w.r.t. $\{w_{kj}^{[2]}\}$:

$$\frac{\partial L}{\partial w_{kj}^{[2]}} = \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial w_{kj}^{[2]}} = \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial x_k^{[2](q)}} \frac{\partial x_k^{[2](q)}}{\partial w_{kj}^{[2]}} = -\sum_{q=1}^{N} \delta_k^{[2](q)} \frac{\partial x_k^{[2](q)}}{\partial w_{kj}^{[2]}},$$

where

$$\delta_k^{[2](q)} = -\frac{\partial L^{(q)}}{\partial x_k^{[2](q)}} = -\sum_{\ell} \frac{\partial L^{(q)}}{\partial x_\ell^{[3](q)}} \frac{\partial x_\ell^{[3](q)}}{\partial z_k^{[2](q)}} \frac{\partial z_k^{[2](q)}}{\partial x_k^{[2](q)}} = \sum_{\ell} \delta_\ell^{[3](q)} \, w_{\ell k}^{[3]} \, g_k^{[2]\prime}(x_k^{[2](q)})$$

$$\frac{\partial x_k^{[2](q)}}{\partial w_{kj}^{[2]}} = z_j^{[1](q)}.$$

- The error terms $\{\delta_k^{[2](q)}\}$ of the second hidden layer are computed based on $\{\delta_\ell^{[3](q)}\}$.

Gradients of the First Layer of Weights

- Gradients w.r.t. $\{w_{ji}^{[1]}\}$:

$$\frac{\partial L}{\partial w_{ji}^{[1]}} = \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial w_{ji}^{[1]}} = \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial x_j^{[1](q)}} \frac{\partial x_j^{[1](q)}}{\partial w_{ji}^{[1]}} = -\sum_{q=1}^{N} \delta_j^{[1](q)} \frac{\partial x_j^{[1](q)}}{\partial w_{ji}^{[1]}},$$

where

$$\delta_j^{[1](q)} = -\frac{\partial L^{(q)}}{\partial x_j^{[1](q)}} = -\sum_k \frac{\partial L^{(q)}}{\partial x_k^{[2](q)}} \frac{\partial x_k^{[2](q)}}{\partial z_j^{[1](q)}} \frac{\partial z_j^{[1](q)}}{\partial x_j^{[1](q)}} = \sum_k \delta_k^{[2](q)} \, w_{kj}^{[2]} \, g_j^{[1]'}(x_j^{[1](q)})$$

$$\frac{\partial x_j^{[1](q)}}{\partial w_{ji}^{[1]}} = x_i^{[1](q)}.$$

- The error terms $\{\delta_j^{[1](q)}\}$ of the first hidden layer are computed based on $\{\delta_k^{[2](q)}\}$.

Weight Update Rules

- Last layer:

$$\delta_\ell^{[3](q)} = y_\ell^{(q)} - z_\ell^{[3](q)}$$

$$\Delta w_{\ell k}^{[3]} = -\eta \frac{\partial L}{\partial w_{\ell k}^{[3]}} = \eta \sum_{q=1}^{N} \delta_\ell^{[3](q)} z_k^{[2](q)}.$$

- Second layer:

$$\delta_k^{[2](q)} = \sum_\ell \delta_\ell^{[3](q)} w_{\ell k}^{[3]} g_k^{[2]\prime}(x_k^{[2](q)})$$

$$\Delta w_{kj}^{[2]} = -\eta \frac{\partial L}{\partial w_{kj}^{[2]}} = \eta \sum_{q=1}^{N} \delta_k^{[2](q)} z_j^{[1](q)}.$$

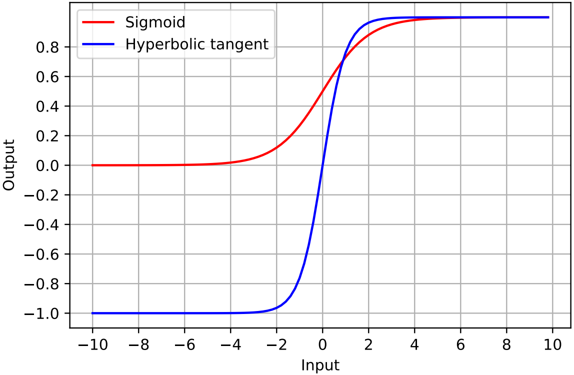
Weight Update Rules (2)

- First layer:

$$\delta_j^{[1](q)} = \sum_k \delta_k^{[2](q)} \, w_{kj}^{[2]} \, g_j^{[1]'}(x_j^{[1](q)})$$

$$\Delta w_{ji}^{[1]} = -\eta \frac{\partial L}{\partial w_{ji}^{[1]}} = \eta \sum_{q=1}^{N} \delta_j^{[1](q)} x_i^{(q)}.$$

Common Activation Functions



Vanishing Gradient Problem

- Both the sigmoid function and the hyperbolic tangent function are saturating activation functions in that their output does not change much when the input becomes very large or very small, i.e., the derivative or gradient $g^{[*]'}(\cdot)$ becomes very close to 0.
- Since the error terms in the BP algorithm make use of the derivatives of the activation functions in the hidden layers, the vanishing gradient problem may arise especially when there are many hidden layers.
- Some techniques for overcoming the vanishing gradient problem in deep neural networks will be discussed in the next topic.

Stochastic Gradient Descent

- While (batch) gradient descent computes the gradients by summing over all N examples in the training set, stochastic gradient descent (SGD) sums over a (usually much smaller) mini-batch of training examples at a time.
- SGD can be regarded as a stochastic approximation of (batch) gradient descent with faster convergence.
- SGD is a more favorable alternative when the training set is large.
- SGD is also good at avoiding being trapped in a local minimum because SGD has more randomness than (batch) gradient descent, making SGD more likely to jump out of a local minimum.

Regularization

- Regularized loss function based on L_2 regularization:

$$L_\lambda(\mathbf{W}; \mathcal{S}) = L(\mathbf{W}; \mathcal{S}) + \frac{\lambda}{2} \sum_{w \text{ except bias terms}} w^2.$$

- Weight update rule for w (except the bias term):

$$\Delta w = -\eta \frac{\partial L_\lambda}{\partial w} = -\eta \frac{\partial L}{\partial w} - \eta \lambda w,$$

where the second term is also called the weight decay term because it moves the weight towards 0.

Momentum

- A momentum term can be added to the weight update rule for w to improve the speed of convergence:

$$\Delta w = -(1-\beta)\,\eta \frac{\partial L}{\partial w} + \beta\, \Delta w^{prev},$$

  where L refers to the loss for a mini-batch, the momentum parameter β is generally taken to be between 0.5 and 1, and Δw^{prev} refers to the previous weight update.
- It has been shown mathematically that the momentum term plays a role similar to the mass in damped harmonic oscillators by bringing the system closer to critical damping.