Introduction to Neural Networks

Inspired from Prof. Xiangliang Zhang (KAUST)

---

by Rishikesh Yadav (Postdoctoral Research Fellow)

HEC Montréal, McGill University, Canada

Table of contents

References

- Christopher M. Bishop (2007). Pattern Recognition and Machine Learning
- Chollet, F., & Allaire, J. J. (2018). *Deep Learning with R*.

- There are some really good Coursera courses for deep learning, particularly with Python.

Goal

- The objectives of this short-course are:
  - Understanding and learning regression using gradient descent algorithm
  - Understand the basics of deep learning and neural networks.
  - Build and train simple feed-forward prediction and regression models using the R interface to Keras.
  - Perform conditional density estimation using neural networks.

Background

Linear Regression (1)

- **Linear Regression** is a statistical method for modeling the relationship between a dependent variable and one or more independent variables.
- The relationship is modeled using a linear predictor function whose unknown parameters are estimated from the data.
- In its simplest form, with one dependent variable y and one independent variable x, the linear regression model is:

$$y = w_0 + w_1 x + \epsilon.$$

  where:
    - y is the dependent variable (response variable).
    - x is the independent variable (predictor variable).
    - w_0 is the intercept term, which represents the expected value of y when $x = 0$.
    - w_1 is the slope term, which represents the change in y for a one-unit change in x.
    - ϵ is the error term, which accounts for the variability in y that cannot be explained by the linear relationship with x.

Linear Regression (1)

- **Linear Regression** is a statistical method for modeling the relationship between a dependent variable and one or more independent variables.
- The relationship is modeled using a linear predictor function whose unknown parameters are estimated from the data.
- In its simplest form, with one dependent variable y and one independent variable x, the linear regression model is:

$$y = w_0 + w_1 x + \epsilon.$$

  where:
  - y is the dependent variable (response variable).
  - x is the independent variable (predictor variable).
  - w_0 is the intercept term, which represents the expected value of y when $x = 0$.
  - w_1 is the slope term, which represents the change in y for a one-unit change in x.
  - ϵ is the error term, which accounts for the variability in y that cannot be explained by the linear relationship with x.

Linear Regression (1)

- **Linear Regression** is a statistical method for modeling the relationship between a dependent variable and one or more independent variables.
- The relationship is modeled using a linear predictor function whose unknown parameters are estimated from the data.
- In its simplest form, with one dependent variable y and one independent variable x, the linear regression model is:

$$y = w_0 + w_1 x + \epsilon.$$

where:

  - y is the dependent variable (response variable).
  - x is the independent variable (predictor variable).
  - w_0 is the intercept term, which represents the expected value of y when $x = 0$.
  - w_1 is the slope term, which represents the change in y for a one-unit change in x.
  - ϵ is the error term, which accounts for the variability in y that cannot be explained by the linear relationship with x.

Linear Regression (1)

- **Linear Regression** is a statistical method for modeling the relationship between a dependent variable and one or more independent variables.
- The relationship is modeled using a linear predictor function whose unknown parameters are estimated from the data.
- In its simplest form, with one dependent variable y and one independent variable x, the linear regression model is:

$$y = w_0 + w_1 x + \epsilon.$$

where:

- y is the dependent variable (response variable).
- x is the independent variable (predictor variable).
- w_0 is the intercept term, which represents the expected value of y when $x = 0$.
- w_1 is the slope term, which represents the change in y for a one-unit change in x.
- ϵ is the error term, which accounts for the variability in y that cannot be explained by the linear relationship with x.

Linear Regression (1)

- **Linear Regression** is a statistical method for modeling the relationship between a dependent variable and one or more independent variables.
- The relationship is modeled using a linear predictor function whose unknown parameters are estimated from the data.
- In its simplest form, with one dependent variable y and one independent variable x, the linear regression model is:

$$y = w_0 + w_1 x + \epsilon.$$

  where:
  - y is the dependent variable (response variable).
  - x is the independent variable (predictor variable).
  - w_0 is the intercept term, which represents the expected value of y when $x = 0$.
  - w_1 is the slope term, which represents the change in y for a one-unit change in x.
  - ϵ is the error term, which accounts for the variability in y that cannot be explained by the linear relationship with x.

Linear Regression (1)

- **Linear Regression** is a statistical method for modeling the relationship between a dependent variable and one or more independent variables.
- The relationship is modeled using a linear predictor function whose unknown parameters are estimated from the data.
- In its simplest form, with one dependent variable y and one independent variable x, the linear regression model is:

$$y = w_0 + w_1 x + \epsilon.$$

  where:

  - y is the dependent variable (response variable).
  - x is the independent variable (predictor variable).
  - w_0 is the intercept term, which represents the expected value of y when $x = 0$.
  - w_1 is the slope term, which represents the change in y for a one-unit change in x.
  - ϵ is the error term, which accounts for the variability in y that cannot be explained by the linear relationship with x.

Linear Regression (1)

- **Linear Regression** is a statistical method for modeling the relationship between a dependent variable and one or more independent variables.
- The relationship is modeled using a linear predictor function whose unknown parameters are estimated from the data.
- In its simplest form, with one dependent variable y and one independent variable x, the linear regression model is:

$$y = w_0 + w_1 x + \epsilon.$$

where:

  - y is the dependent variable (response variable).
  - x is the independent variable (predictor variable).
  - w_0 is the intercept term, which represents the expected value of y when $x = 0$.
  - w_1 is the slope term, which represents the change in y for a one-unit change in x.
  - ϵ is the error term, which accounts for the variability in y that cannot be explained by the linear relationship with x.

Linear Regression (1)

- **Linear Regression** is a statistical method for modeling the relationship between a dependent variable and one or more independent variables.
- The relationship is modeled using a linear predictor function whose unknown parameters are estimated from the data.
- In its simplest form, with one dependent variable y and one independent variable x, the linear regression model is:

$$y = w_0 + w_1 x + \epsilon.$$

where:

  - y is the dependent variable (response variable).
  - x is the independent variable (predictor variable).
  - w_0 is the intercept term, which represents the expected value of y when $x = 0$.
  - w_1 is the slope term, which represents the change in y for a one-unit change in x.
  - ϵ is the error term, which accounts for the variability in y that cannot be explained by the linear relationship with x.

Linear Regression (2)

- The parameters w_0 and w_1 are estimated from the data using methods such as **Ordinary Least Squares (OLS)**, which minimizes the sum of the squared differences between the observed values and the predicted values.
- The **OLS** estimate of the parameters can be obtained by solving the following normal equations:

$$\begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \end{pmatrix} = \left(X^T X\right)^{-1} X^T y.$$

  where:
  - X is the design matrix, including a column of ones for the intercept and a column of the predictor variable values.
  - y is the vector of observed values of the dependent variable.
- Linear regression can be extended to include multiple independent variables, resulting in **Multiple Linear Regression**:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p + \epsilon.$$

  where $x_1, x_2, \ldots, x_p$ are the predictor variables.

Linear Regression (2)

- The parameters w_0 and w_1 are estimated from the data using methods such as **Ordinary Least Squares (OLS)**, which minimizes the sum of the squared differences between the observed values and the predicted values.
- The **OLS** estimate of the parameters can be obtained by solving the following normal equations:

$$\begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \end{pmatrix} = \left(X^T X\right)^{-1} X^T y.$$

  where:
  - X is the design matrix, including a column of ones for the intercept and a column of the predictor variable values.
  - y is the vector of observed values of the dependent variable.
- Linear regression can be extended to include multiple independent variables, resulting in **Multiple Linear Regression**:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p + \epsilon.$$

  where $x_1, x_2, \ldots, x_p$ are the predictor variables.

Linear Regression (2)

- The parameters w_0 and w_1 are estimated from the data using methods such as **Ordinary Least Squares (OLS)**, which minimizes the sum of the squared differences between the observed values and the predicted values.
- The **OLS** estimate of the parameters can be obtained by solving the following normal equations:

$$\begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \end{pmatrix} = \left(X^T X\right)^{-1} X^T y.$$

  where:
  - X is the design matrix, including a column of ones for the intercept and a column of the predictor variable values.
  - y is the vector of observed values of the dependent variable.
- Linear regression can be extended to include multiple independent variables, resulting in **Multiple Linear Regression**:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p + \epsilon.$$

  where $x_1, x_2, \ldots, x_p$ are the predictor variables.

Linear Regression (2)

- The parameters w_0 and w_1 are estimated from the data using methods such as **Ordinary Least Squares (OLS)**, which minimizes the sum of the squared differences between the observed values and the predicted values.
- The **OLS** estimate of the parameters can be obtained by solving the following normal equations:

$$\begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \end{pmatrix} = \left(X^T X\right)^{-1} X^T y.$$

  where:
  - X is the design matrix, including a column of ones for the intercept and a column of the predictor variable values.
  - y is the vector of observed values of the dependent variable.
- Linear regression can be extended to include multiple independent variables, resulting in **Multiple Linear Regression**:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p + \epsilon.$$

  where $x_1, x_2, \ldots, x_p$ are the predictor variables.

Linear Regression (2)

- The parameters w_0 and w_1 are estimated from the data using methods such as **Ordinary Least Squares (OLS)**, which minimizes the sum of the squared differences between the observed values and the predicted values.
- The **OLS** estimate of the parameters can be obtained by solving the following normal equations:

$$\begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \end{pmatrix} = \left(X^T X\right)^{-1} X^T y.$$

  where:
  - X is the design matrix, including a column of ones for the intercept and a column of the predictor variable values.
  - y is the vector of observed values of the dependent variable.
- Linear regression can be extended to include multiple independent variables, resulting in **Multiple Linear Regression**:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p + \epsilon.$$

  where $x_1, x_2, \ldots, x_p$ are the predictor variables.

Gradient Descent Algorithm (1)

Gradient Descent Algorithm is an optimization technique used to minimize a loss function by iteratively moving towards the minimum of the function.

The main components of the gradient descent algorithm are:

- **Loss Function**:
  - The loss function measures the error or difference between the predicted values and the actual values. That means it measures how well the model's predictions match the actual data.
  - The goal of gradient descent is to minimize this loss function to improve the accuracy of the model.
  - Examples of loss functions include **Mean Squared Error (MSE)** for regression problems and **Cross-Entropy Loss** for classification problems.
  - Mathematically, for a set of predictions $\hat{y}$ and actual values y, the Mean Squared Error is given by:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Gradient Descent Algorithm (1)

Gradient Descent Algorithm is an optimization technique used to minimize a loss function by iteratively moving towards the minimum of the function.

The main components of the gradient descent algorithm are:

- **Loss Function**:
    - The loss function measures the error or difference between the predicted values and the actual values. That means it measures how well the model's predictions match the actual data.
    - The goal of gradient descent is to minimize this loss function to improve the accuracy of the model.
    - Examples of loss functions include **Mean Squared Error (MSE)** for regression problems and **Cross-Entropy Loss** for classification problems.
    - Mathematically, for a set of predictions $\hat{y}$ and actual values y, the Mean Squared Error is given by:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Gradient Descent Algorithm (1)

Gradient Descent Algorithm is an optimization technique used to minimize a loss function by iteratively moving towards the minimum of the function.

The main components of the gradient descent algorithm are:

- **Loss Function**:
  - The loss function measures the error or difference between the predicted values and the actual values. That means it measures how well the model's predictions match the actual data.
  - The goal of gradient descent is to minimize this loss function to improve the accuracy of the model.
  - Examples of loss functions include **Mean Squared Error (MSE)** for regression problems and **Cross-Entropy Loss** for classification problems.
  - Mathematically, for a set of predictions $\hat{y}$ and actual values y, the Mean Squared Error is given by:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Gradient Descent Algorithm (1)

Gradient Descent Algorithm is an optimization technique used to minimize a loss function by iteratively moving towards the minimum of the function.

The main components of the gradient descent algorithm are:

- **Loss Function**:
  - The loss function measures the error or difference between the predicted values and the actual values. That means it measures how well the model's predictions match the actual data.
  - The goal of gradient descent is to minimize this loss function to improve the accuracy of the model.
  - Examples of loss functions include **Mean Squared Error (MSE)** for regression problems and **Cross-Entropy Loss** for classification problems.
  - Mathematically, for a set of predictions $\hat{y}$ and actual values y, the Mean Squared Error is given by:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Gradient Descent Algorithm (1)

Gradient Descent Algorithm is an optimization technique used to minimize a loss function by iteratively moving towards the minimum of the function.

The main components of the gradient descent algorithm are:

- **Loss Function**:
  - The loss function measures the error or difference between the predicted values and the actual values. That means it measures how well the model's predictions match the actual data.
  - The goal of gradient descent is to minimize this loss function to improve the accuracy of the model.
  - Examples of loss functions include **Mean Squared Error (MSE)** for regression problems and **Cross-Entropy Loss** for classification problems.
  - Mathematically, for a set of predictions $\hat{y}$ and actual values y, the Mean Squared Error is given by:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Gradient Descent Algorithm (2)

- **Gradient**:
  - The gradient is a vector of partial derivatives of the loss function with respect to all the parameters of the model.
  - It points in the direction of the steepest ascent of the function.
  - In gradient descent, we move in the opposite direction of the gradient to find the minimum of the loss function.
  - For a parameter w, the gradient of the loss function L with respect to w is given by:

$$\nabla_w L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_p} \right).$$

- **Update Rule**:
  - In each iteration of gradient descent, the model parameters are updated using the gradient and the learning rate.
  - The update rule for a parameter w is given by:

$$w := w - \eta \nabla_w L,$$

where η is the learning rate and $\nabla_w L$ is the gradient of the loss function with respect to w.

Gradient Descent Algorithm (2)

- **Gradient**:
  - The gradient is a vector of partial derivatives of the loss function with respect to all the parameters of the model.
  - It points in the direction of the steepest ascent of the function.
  - In gradient descent, we move in the opposite direction of the gradient to find the minimum of the loss function.
  - For a parameter w, the gradient of the loss function L with respect to w is given by:

$$\nabla_w L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_p} \right).$$

- **Update Rule**:
  - In each iteration of gradient descent, the model parameters are updated using the gradient and the learning rate.
  - The update rule for a parameter w is given by:

$$w := w - \eta \nabla_w L,$$

where η is the learning rate and $\nabla_w L$ is the gradient of the loss function with respect to w.

Gradient Descent Algorithm (2)

- **Gradient**:
  - The gradient is a vector of partial derivatives of the loss function with respect to all the parameters of the model.
  - It points in the direction of the steepest ascent of the function.
  - In gradient descent, we move in the opposite direction of the gradient to find the minimum of the loss function.
  - For a parameter w, the gradient of the loss function L with respect to w is given by:

$$\nabla_w L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_p} \right).$$

- **Update Rule**:
  - In each iteration of gradient descent, the model parameters are updated using the gradient and the learning rate.
  - The update rule for a parameter w is given by:

$$w := w - \eta \nabla_w L,$$

    where η is the learning rate and $\nabla_w L$ is the gradient of the loss function with respect to w.

Gradient Descent Algorithm (2)

- **Gradient**:
  - The gradient is a vector of partial derivatives of the loss function with respect to all the parameters of the model.
  - It points in the direction of the steepest ascent of the function.
  - In gradient descent, we move in the opposite direction of the gradient to find the minimum of the loss function.
  - For a parameter w, the gradient of the loss function L with respect to w is given by:

$$\nabla_w L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_p} \right).$$

- **Update Rule**:
  - In each iteration of gradient descent, the model parameters are updated using the gradient and the learning rate.
  - The update rule for a parameter w is given by:

$$w := w - \eta \nabla_w L,$$

    where η is the learning rate and $\nabla_w L$ is the gradient of the loss function with respect to w.

Gradient Descent Algorithm (2)

- **Gradient**:
  - The gradient is a vector of partial derivatives of the loss function with respect to all the parameters of the model.
  - It points in the direction of the steepest ascent of the function.
  - In gradient descent, we move in the opposite direction of the gradient to find the minimum of the loss function.
  - For a parameter w, the gradient of the loss function L with respect to w is given by:

$$\nabla_w L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_p} \right).$$

- **Update Rule**:
  - In each iteration of gradient descent, the model parameters are updated using the gradient and the learning rate.
  - The update rule for a parameter w is given by:

$$w := w - \eta \nabla_w L,$$

    where η is the learning rate and $\nabla_w L$ is the gradient of the loss function with respect to w.

Gradient Descent Algorithm (2)

- **Gradient**:
  - The gradient is a vector of partial derivatives of the loss function with respect to all the parameters of the model.
  - It points in the direction of the steepest ascent of the function.
  - In gradient descent, we move in the opposite direction of the gradient to find the minimum of the loss function.
  - For a parameter w, the gradient of the loss function L with respect to w is given by:

$$\nabla_w L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \dots, \frac{\partial L}{\partial w_p} \right).$$

- **Update Rule**:
  - In each iteration of gradient descent, the model parameters are updated using the gradient and the learning rate.
  - The update rule for a parameter w is given by:

$$w := w - \eta \nabla_w L,$$

    where η is the learning rate and $\nabla_w L$ is the gradient of the loss function with respect to w.

Gradient Descent Algorithm (2)

- **Gradient**:
  - The gradient is a vector of partial derivatives of the loss function with respect to all the parameters of the model.
  - It points in the direction of the steepest ascent of the function.
  - In gradient descent, we move in the opposite direction of the gradient to find the minimum of the loss function.
  - For a parameter w, the gradient of the loss function L with respect to w is given by:

$$\nabla_w L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_p} \right).$$

- **Update Rule**:
  - In each iteration of gradient descent, the model parameters are updated using the gradient and the learning rate.
  - The update rule for a parameter w is given by:

$$w := w - \eta \nabla_w L,$$

where η is the learning rate and $\nabla_w L$ is the gradient of the loss function with respect to w.

Gradient Descent Algorithm (2)

- **Gradient**:
  - The gradient is a vector of partial derivatives of the loss function with respect to all the parameters of the model.
  - It points in the direction of the steepest ascent of the function.
  - In gradient descent, we move in the opposite direction of the gradient to find the minimum of the loss function.
  - For a parameter w, the gradient of the loss function L with respect to w is given by:

$$\nabla_w L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_p} \right).$$

- **Update Rule**:
  - In each iteration of gradient descent, the model parameters are updated using the gradient and the learning rate.
  - The update rule for a parameter w is given by:

$$w := w - \eta \nabla_w L,$$

where η is the learning rate and $\nabla_w L$ is the gradient of the loss function with respect to w.

Gradient Descent Algorithm (3)

- **Learning Rate η**:
  - The learning rate is a hyperparameter that determines the size of the steps taken towards the minimum of the loss function.
  - A smaller learning rate means smaller steps, leading to more accurate convergence but slower progress.
  - A larger learning rate can speed up the process but risks overshooting the minimum and potentially diverging.
  - It is important to choose an appropriate learning rate to ensure efficient and effective training of the model.
- **Convergence**:
  - The algorithm iteratively updates the parameters until the loss function converges to a minimum value.
  - Convergence can be determined by setting a threshold for the change in the loss function or by specifying a maximum number of iterations.

Gradient Descent Algorithm (3)

- **Learning Rate** η:
  - The learning rate is a hyperparameter that determines the size of the steps taken towards the minimum of the loss function.
  - A smaller learning rate means smaller steps, leading to more accurate convergence but slower progress.
  - A larger learning rate can speed up the process but risks overshooting the minimum and potentially diverging.
  - It is important to choose an appropriate learning rate to ensure efficient and effective training of the model.
- **Convergence**:
  - The algorithm iteratively updates the parameters until the loss function converges to a minimum value.
  - Convergence can be determined by setting a threshold for the change in the loss function or by specifying a maximum number of iterations.

Gradient Descent Algorithm (3)

- **Learning Rate** η:
  - The learning rate is a hyperparameter that determines the size of the steps taken towards the minimum of the loss function.
  - A smaller learning rate means smaller steps, leading to more accurate convergence but slower progress.
  - A larger learning rate can speed up the process but risks overshooting the minimum and potentially diverging.
  - It is important to choose an appropriate learning rate to ensure efficient and effective training of the model.
- **Convergence**:
  - The algorithm iteratively updates the parameters until the loss function converges to a minimum value.
  - Convergence can be determined by setting a threshold for the change in the loss function or by specifying a maximum number of iterations.

Gradient Descent Algorithm (3)

- **Learning Rate** η:
  - The learning rate is a hyperparameter that determines the size of the steps taken towards the minimum of the loss function.
  - A smaller learning rate means smaller steps, leading to more accurate convergence but slower progress.
  - A larger learning rate can speed up the process but risks overshooting the minimum and potentially diverging.
  - It is important to choose an appropriate learning rate to ensure efficient and effective training of the model.
- **Convergence**:
  - The algorithm iteratively updates the parameters until the loss function converges to a minimum value.
  - Convergence can be determined by setting a threshold for the change in the loss function or by specifying a maximum number of iterations.

Gradient Descent Algorithm (3)

- **Learning Rate** η:
  - The learning rate is a hyperparameter that determines the size of the steps taken towards the minimum of the loss function.
  - A smaller learning rate means smaller steps, leading to more accurate convergence but slower progress.
  - A larger learning rate can speed up the process but risks overshooting the minimum and potentially diverging.
  - It is important to choose an appropriate learning rate to ensure efficient and effective training of the model.
- **Convergence**:
  - The algorithm iteratively updates the parameters until the loss function converges to a minimum value.
  - Convergence can be determined by setting a threshold for the change in the loss function or by specifying a maximum number of iterations.

Gradient Descent Algorithm (3)

- **Learning Rate** η:
  - The learning rate is a hyperparameter that determines the size of the steps taken towards the minimum of the loss function.
  - A smaller learning rate means smaller steps, leading to more accurate convergence but slower progress.
  - A larger learning rate can speed up the process but risks overshooting the minimum and potentially diverging.
  - It is important to choose an appropriate learning rate to ensure efficient and effective training of the model.
- **Convergence**:
  - The algorithm iteratively updates the parameters until the loss function converges to a minimum value.
  - Convergence can be determined by setting a threshold for the change in the loss function or by specifying a maximum number of iterations.

Gradient Descent Algorithm (3)

- **Learning Rate** η:
  - The learning rate is a hyperparameter that determines the size of the steps taken towards the minimum of the loss function.
  - A smaller learning rate means smaller steps, leading to more accurate convergence but slower progress.
  - A larger learning rate can speed up the process but risks overshooting the minimum and potentially diverging.
  - It is important to choose an appropriate learning rate to ensure efficient and effective training of the model.
- **Convergence**:
  - The algorithm iteratively updates the parameters until the loss function converges to a minimum value.
  - Convergence can be determined by setting a threshold for the change in the loss function or by specifying a maximum number of iterations.

Gradient Descent Algorithm (3)

- **Learning Rate** η:
  - The learning rate is a hyperparameter that determines the size of the steps taken towards the minimum of the loss function.
  - A smaller learning rate means smaller steps, leading to more accurate convergence but slower progress.
  - A larger learning rate can speed up the process but risks overshooting the minimum and potentially diverging.
  - It is important to choose an appropriate learning rate to ensure efficient and effective training of the model.
- **Convergence**:
  - The algorithm iteratively updates the parameters until the loss function converges to a minimum value.
  - Convergence can be determined by setting a threshold for the change in the loss function or by specifying a maximum number of iterations.

Gradient Descent Algorithm: Algorithm Steps

- **Initialize**: Start with an initial guess for the parameters.
- **Compute Gradient**: Calculate the gradient of the loss function with respect to each parameter.
- **Update Parameters**: Adjust the parameters by moving them in the direction opposite to the gradient, scaled by the learning rate.
- **Iterate**: Repeat the process until the parameters converge to the minimum or a stopping criterion is met.

Gradient Descent Algorithm: Algorithm Steps

- **Initialize**: Start with an initial guess for the parameters.
- **Compute Gradient**: Calculate the gradient of the loss function with respect to each parameter.
- **Update Parameters**: Adjust the parameters by moving them in the direction opposite to the gradient, scaled by the learning rate.
- **Iterate**: Repeat the process until the parameters converge to the minimum or a stopping criterion is met.

Gradient Descent Algorithm: Algorithm Steps

- **Initialize**: Start with an initial guess for the parameters.
- **Compute Gradient**: Calculate the gradient of the loss function with respect to each parameter.
- **Update Parameters**: Adjust the parameters by moving them in the direction opposite to the gradient, scaled by the learning rate.
- **Iterate**: Repeat the process until the parameters converge to the minimum or a stopping criterion is met.

Gradient Descent Algorithm: Algorithm Steps

- **Initialize**: Start with an initial guess for the parameters.
- **Compute Gradient**: Calculate the gradient of the loss function with respect to each parameter.
- **Update Parameters**: Adjust the parameters by moving them in the direction opposite to the gradient, scaled by the learning rate.
- **Iterate**: Repeat the process until the parameters converge to the minimum or a stopping criterion is met.

Example 1: Gradient Descent for Linear Regression

- **Loss Function** (Mean Squared Error, MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_0 + w_1 x_i))^2.$$

- **Gradients**:

$$\frac{\partial \text{MSE}}{\partial w_1} = -\frac{2}{N}\sum_{i=1}^{N} x_i(y_i - (w_0 + w_1 x_i)),$$

$$\frac{\partial \text{MSE}}{\partial w_0} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - (w_0 + w_1 x_i)).$$

- **Parameter Updates**:

$$w_1 \leftarrow w_1 - \eta \frac{\partial \text{MSE}}{\partial w_1},$$

$$w_0 \leftarrow w_0 - \eta \frac{\partial \text{MSE}}{\partial w_0}.$$

where η is the learning rate.

Example 1: Gradient Descent for Linear Regression

- **Loss Function** (Mean Squared Error, MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_0 + w_1 x_i))^2.$$

- **Gradients**:

$$\frac{\partial \text{MSE}}{\partial w_1} = -\frac{2}{N}\sum_{i=1}^{N} x_i (y_i - (w_0 + w_1 x_i)),$$

$$\frac{\partial \text{MSE}}{\partial w_0} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - (w_0 + w_1 x_i)).$$

- **Parameter Updates**:

$$w_1 \leftarrow w_1 - \eta \frac{\partial \text{MSE}}{\partial w_1},$$

$$w_0 \leftarrow w_0 - \eta \frac{\partial \text{MSE}}{\partial w_0}.$$

where η is the learning rate.

Example 1: Gradient Descent for Linear Regression

- **Loss Function** (Mean Squared Error, MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_0 + w_1 x_i))^2.$$

- **Gradients**:

$$\frac{\partial \text{MSE}}{\partial w_1} = -\frac{2}{N}\sum_{i=1}^{N} x_i(y_i - (w_0 + w_1 x_i)),$$

$$\frac{\partial \text{MSE}}{\partial w_0} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - (w_0 + w_1 x_i)).$$

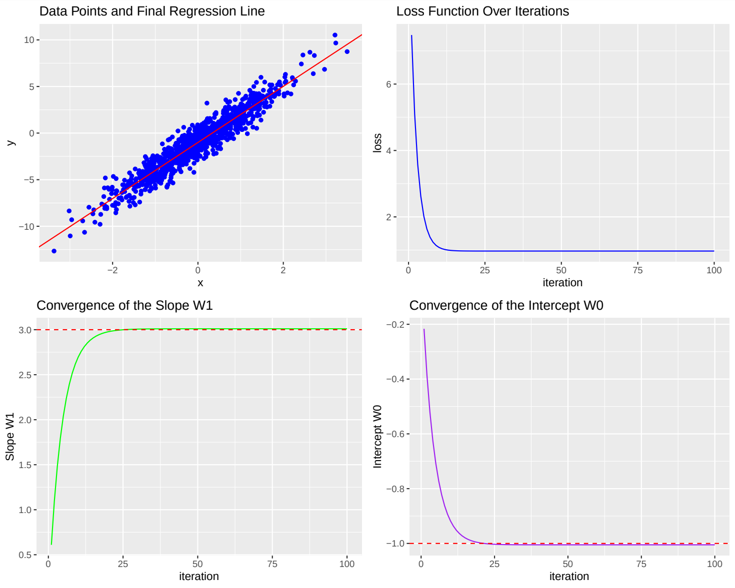
- **Parameter Updates**:

$$w_1 \leftarrow w_1 - \eta\frac{\partial \text{MSE}}{\partial w_1},$$

$$w_0 \leftarrow w_0 - \eta\frac{\partial \text{MSE}}{\partial w_0}.$$

where η is the learning rate.

Results Example 1



[1]

[1] See the R code **gradient-descent-LM.R**

Example 2: Gradient Descent for Polynomial Regression

- **Polynomial Regression** is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an Mth degree polynomial.

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M + \epsilon.$$

- **The loss function** for polynomial regression is similar to linear regression, typically using **Mean Squared Error (MSE)**.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad \hat{y}_i = w_0 + w_1 x_i + w_2 x_i^2 + \ldots + w_M x_i^M.$$

- The update rule is similar but applied to all polynomial coefficients.

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} MSE,$$

$$\frac{\partial}{\partial w_j} MSE = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)x_i^j, \qquad j = 0, \ldots, M.$$

Example 2: Gradient Descent for Polynomial Regression

- **Polynomial Regression** is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an Mth degree polynomial.

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M + \epsilon.$$

- **The loss function** for polynomial regression is similar to linear regression, typically using **Mean Squared Error (MSE)**.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad \hat{y}_i = w_0 + w_1 x_i + w_2 x_i^2 + \ldots + w_M x_i^M.$$

- The update rule is similar but applied to all polynomial coefficients.

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} MSE,$$

$$\frac{\partial}{\partial w_j} MSE = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) x_i^j, \qquad j = 0, \ldots, M.$$

Example 2: Gradient Descent for Polynomial Regression

- **Polynomial Regression** is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an Mth degree polynomial.

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M + \epsilon.$$

- **The loss function** for polynomial regression is similar to linear regression, typically using **Mean Squared Error (MSE)**.

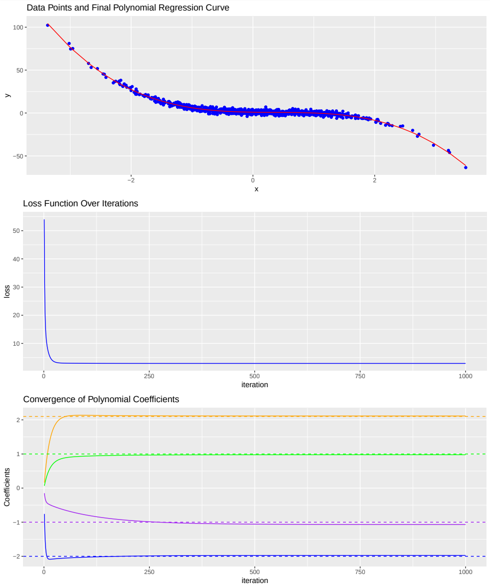
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad \hat{y}_i = w_0 + w_1 x_i + w_2 x_i^2 + \ldots + w_M x_i^M.$$

- The update rule is similar but applied to all polynomial coefficients.

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} MSE,$$

$$\frac{\partial}{\partial w_j} MSE = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) x_i^j, \qquad j = 0, \ldots, M.$$

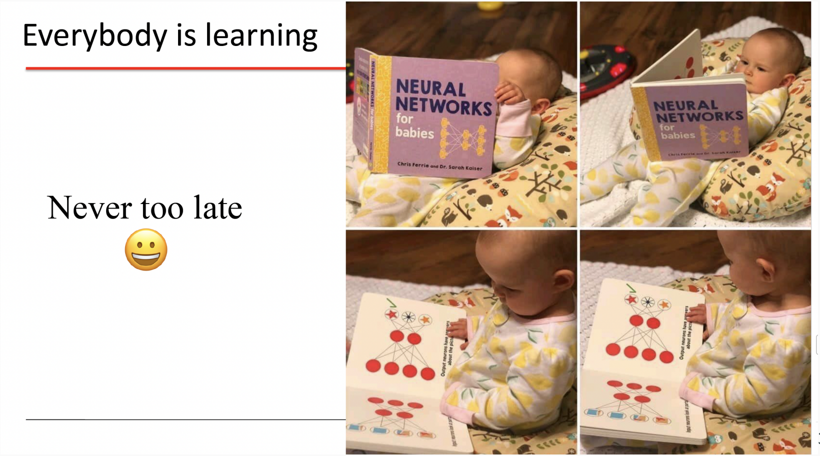
Results Example 2



[2]

[2] See the R code **gradient-decent-polynomial.R**

Neural Networks



[3]

[3] CS 229: Prof. Xiangliang Zhang (KAUST)

What is a Neural Network?

- A **Neural Network** is a computational model inspired by the way biological neural networks in the human brain process information.
- It consists of **layers** of interconnected **nodes**, or **neurons**, which work together to solve complex problems.
- Similarly, brain activity occurs when a **stimulus** enters the system; information is processed through the network via neurons that extract relevant information, and this information is passed to another area.
- Neural networks can **learn** from data through **training**, making them powerful tools for **pattern recognition** and **predictive modeling**.

What is a Neural Network?

- A **Neural Network** is a computational model inspired by the way biological neural networks in the human brain process information.
- It consists of **layers** of interconnected **nodes**, or **neurons**, which work together to solve complex problems.
- Similarly, brain activity occurs when a **stimulus** enters the system; information is processed through the network via neurons that extract relevant information, and this information is passed to another area.
- Neural networks can **learn** from data through **training**, making them powerful tools for **pattern recognition** and **predictive modeling**.

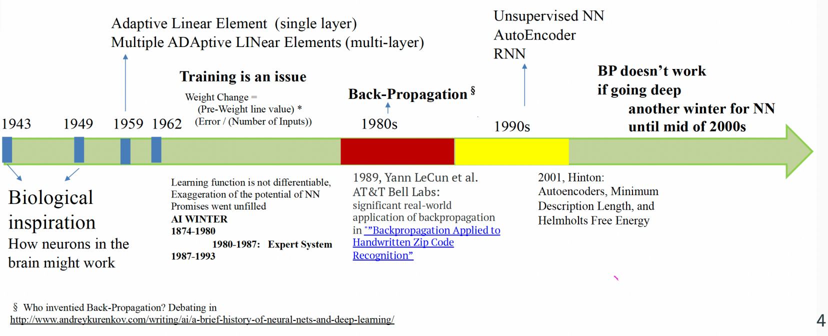
What is a Neural Network?

- A **Neural Network** is a computational model inspired by the way biological neural networks in the human brain process information.
- It consists of **layers** of interconnected **nodes**, or **neurons**, which work together to solve complex problems.
- Similarly, brain activity occurs when a **stimulus** enters the system; information is processed through the network via neurons that extract relevant information, and this information is passed to another area.
- Neural networks can **learn** from data through **training**, making them powerful tools for **pattern recognition** and **predictive modeling**.

What is a Neural Network?

- A **Neural Network** is a computational model inspired by the way biological neural networks in the human brain process information.
- It consists of **layers** of interconnected **nodes**, or **neurons**, which work together to solve complex problems.
- Similarly, brain activity occurs when a **stimulus** enters the system; information is processed through the network via neurons that extract relevant information, and this information is passed to another area.
- Neural networks can **learn** from data through **training**, making them powerful tools for **pattern recognition** and **predictive modeling**.

History of NNs



[4]

[4] CS 229: Prof. Xiangliang Zhang (KAUST)

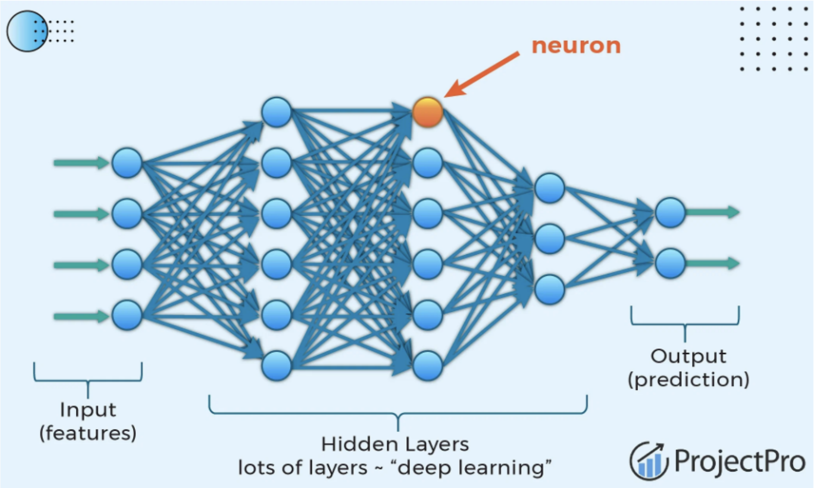
Components of Neural Networks

- Layers
- Neurons
- Weights
- Bias
- Activation Functions
- Forward Propagation
- Loss Function
- Backpropagation
- Learning Rate
- Training Data
- Validation and Test Data

A Basic Architectures of Neural Networks (1)

- A neural network consists of several key components:
- **Input Layer:** Each input represents a single predictor variable, which can be a scalar value, a sequence, an image, or even a sequence of images.
- **Output Layer:** For prediction or classification tasks, this layer typically has a single node. For Conditional Density Estimation (CDE), it contains multiple nodes.
- **Hidden Layers:** These layers, often more than one, perform calculations at each node. Each hidden layer's computations are parameterized by weights and biases.
- The calculations within a layer can vary in type, including standard, convolutional, and recurrent layers.
- The structure of a neural network, referred to as its **architecture**, defines the arrangement and connectivity of these layers.

A Basic Architectures of Neural Networks (2)



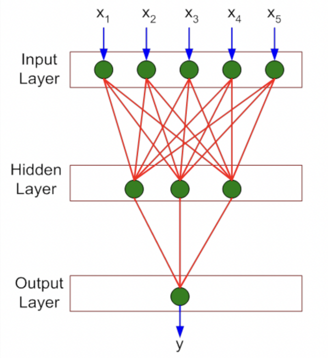
[5]

[5] ProjectPro

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).

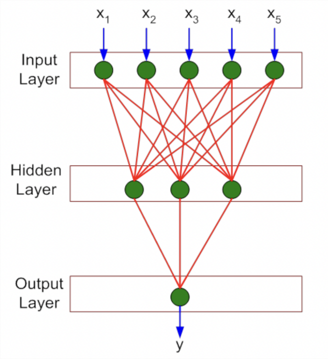


[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



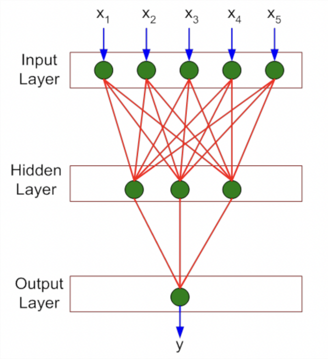
[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



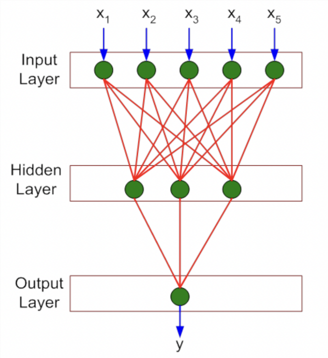
[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



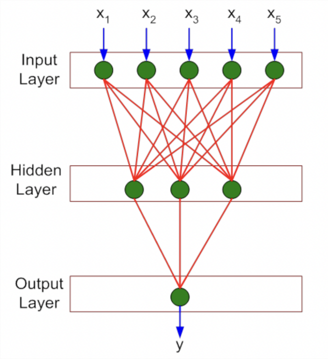
[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



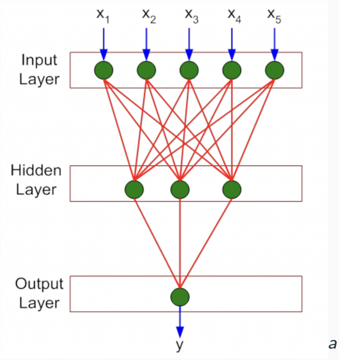
[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).

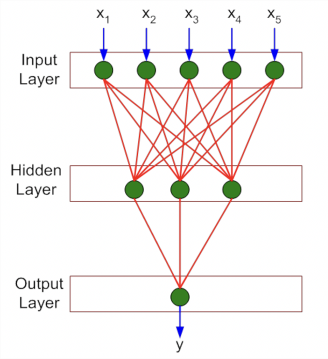


[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



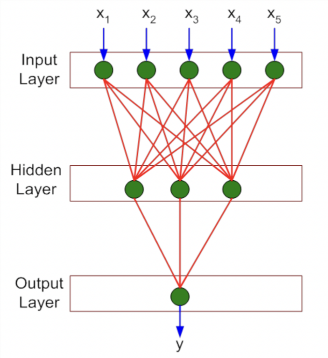
[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



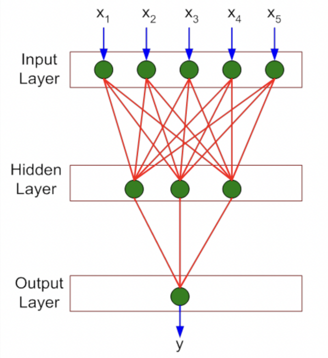
[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



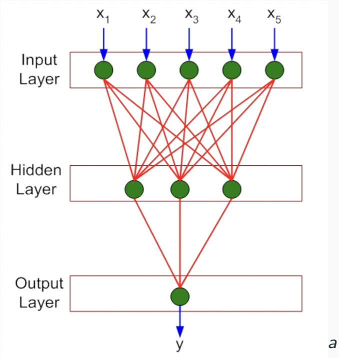
[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



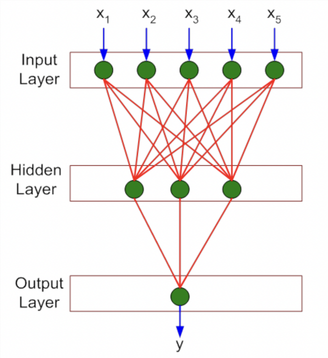
[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).

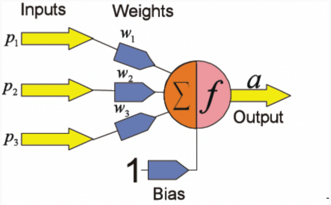


[a]

[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.

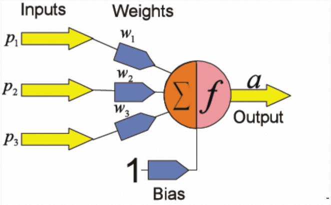


$$a = f(p_1 w_1 + p_2 w_2 + p_3 w_3 + b) = f\left(\sum p_i w_i + b\right)$$ [6]

[6] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.

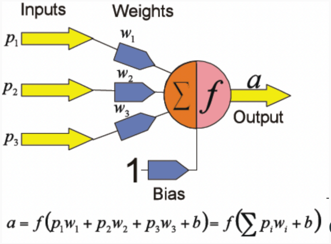


$$a = f(p_1 w_1 + p_2 w_2 + p_3 w_3 + b) = f\left(\sum p_i w_i + b\right)$$ [6]

[6] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.

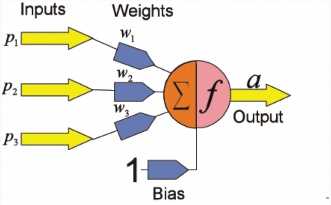


[6]

[6] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

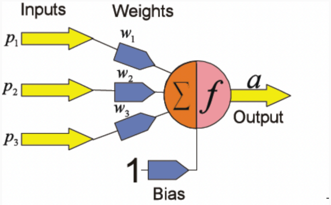
- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.



$$a = f(p_1 w_1 + p_2 w_2 + p_3 w_3 + b) = f\left(\sum p_i w_i + b\right)$$ [6]

[6] CS 229: Prof. Xiangliang Zhang (KAUST)

- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.

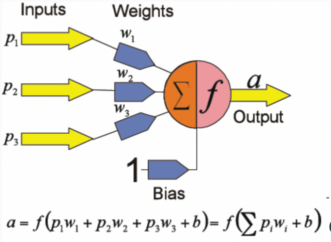


$$a = f(p_1 w_1 + p_2 w_2 + p_3 w_3 + b) = f\left(\sum p_i w_i + b\right)$$ [6]

[6] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.

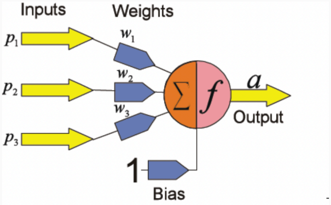


[6]

[6]CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
    - Basic units of a neural network.
    - Receive input, process it, and pass it to other neurons.
    - Apply a mathematical function to generate an output.
- **Weights**:
    - Parameters that transform input data within the network.
    - Each connection between neurons has an associated weight.
- **Biases**:
    - Additional parameters in a neuron that allow to fit the data better.
    - Provide ability to each neuron have output not 0 when inputs are 0.

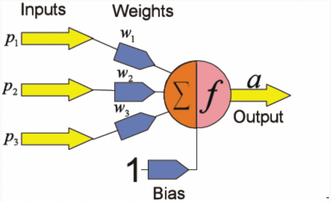


$$a = f(p_1w_1 + p_2w_2 + p_3w_3 + b) = f\left(\sum p_iw_i + b\right)$$ [6]

[6] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.

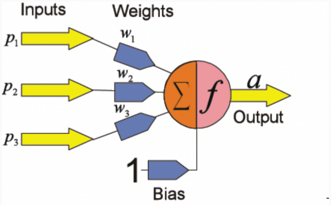


$$a = f(p_1 w_1 + p_2 w_2 + p_3 w_3 + b) = f\left(\sum p_i w_i + b\right)$$ [6]

[6] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
    - Basic units of a neural network.
    - Receive input, process it, and pass it to other neurons.
    - Apply a mathematical function to generate an output.
- **Weights**:
    - Parameters that transform input data within the network.
    - Each connection between neurons has an associated weight.
- **Biases**:
    - Additional parameters in a neuron that allow to fit the data better.
    - Provide ability to each neuron have output not 0 when inputs are 0.

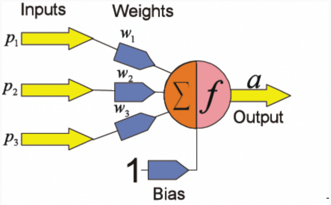


$$a = f(p_1w_1 + p_2w_2 + p_3w_3 + b) = f\left(\sum p_iw_i + b\right)$$ [6]

[6] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.



$$a = f(p_1 w_1 + p_2 w_2 + p_3 w_3 + b) = f\left(\sum p_i w_i + b\right)$$ [6]

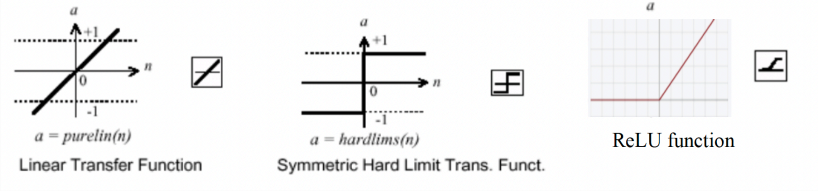
[6] CS 229: Prof. Xiangliang Zhang (KAUST)

Activation function

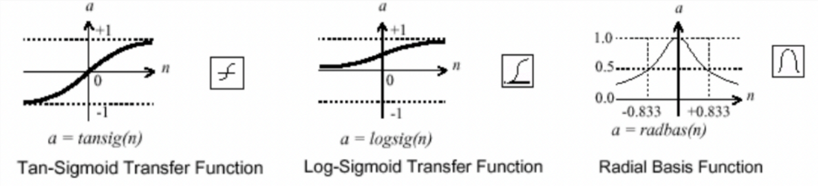
- The activation function defines the **output of the node**. There is two different functions that we can use, but it must satisfy bwlow two properties
  - **Nonlinear**: If all node has linear activation functions, this is just a linear regression model.
  - **Continuously differentiable:** Needed for gradient based optimization

For example, we can think of this activation functions as link function in generalized linear model. In fact a single layered NN is simply a generalized linear model.

Some Examples of Activation Function



[7]



[7] CS 229: Prof. Xiangliang Zhang (KAUST)

Rectified Linear Unit (ReLu) Activation Function

- Older neural networks relied on sigmoid or tanh activation functions that suffsred from vanishing gradient problems which restrict going to deep
- ReLu[8] somewhat solve this issue and turn out to be a huge help
- ReLu has the following forms

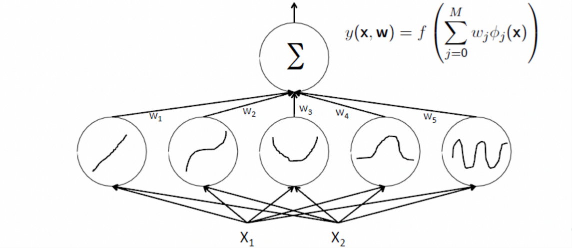
$$a(x) = \max\{0, x\}.$$

  That means

  - The neuron **activates** only when enough information passes through
  - One drawbacks: Not differentiable at zero.

[8] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Linear Regression as Neural Networks



[9]

$f(\cdot)$

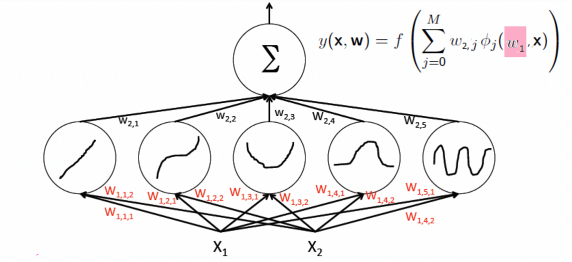
- The identity function in the case of regression.
- A nonlinear activation function in the case of classification.

Learning object:

- Learn $\boldsymbol{w}$'s so the function fits the data well.
- The basis functions are fixed functions of X.

[9] CS 229: Prof. Xiangliang Zhang (KAUST)

Non Linear Regression as Neural Networks

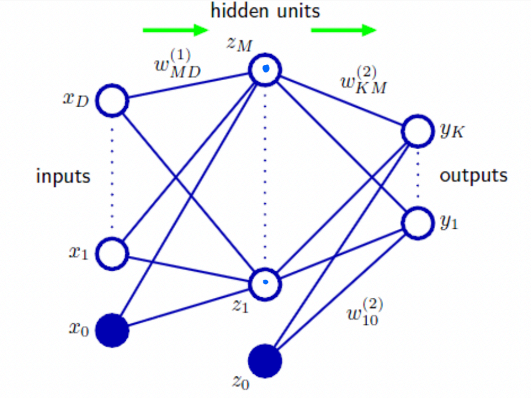


[10]

- Learning object: Learn $\boldsymbol{w}$ so the function fits the data well
- The basis functions are fixed functions of X and $\boldsymbol{w}_1$

[10] CS 229: Prof. Xiangliang Zhang (KAUST)

A Two-Layer Neural Network (1)



- D input variables
- M hidden neurons
- K outputs
- Use the two activation function on weights sum on hidden layers and output layers

A Two-Layer Neural Network (2)

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

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i, \qquad x_0 = 1.$$

- a_j are the activations, $j = 1, \ldots, M$.
- $w_{ji}^{(1)}$ are the layer one weights, $i = 1, \ldots, D$.
- $w_{j0}^{(1)}$ are the layer one biases.

Each linear combination a_j is transformed by a (non-linear differentiable) activation function

$$z_j = h(a_j).$$

A Two-Layer Neural Network (3)

The hidden output $z_j = h(a_j)$ are linearly combined in layer two:

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j, \qquad z_0 = 1.$$

- a_k are the output activations, $k = 1, \dots, K$.
- $w_{kj}^{(2)}$ are the layer two weights, $j = 1, \dots, M$.
- $w_{k0}^{(2)}$ are the layer one biases.

The output activations a_k are transformed by the output (non-linear differentiable) activation function

$$y_k = \sigma(a_k).$$

- y_k are the final outputs.
- $\sigma(\cdot)$ is like $h(\cdot)$, but often sigmoid function for classification and linear function for Regressions.

A Two-Layer Neural Network (4)

After substituting $y_k = \sigma(a_k)$ the definitions of a_j and a_k :

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

Evaluation of this is called **forward propagation**.

- $h(\cdot)$ and $\sigma(\cdot)$ are a sigmoid functions, e.g., the logistics function.

$$s(a) = \frac{1}{1 + \exp(-a)}, \quad s(a) \in [0, 1].$$

- If $\sigma(a)$ is the identity, then a regression model is obtained.

Properties & Generalizations

- Typically, $K < D < M$. If $M < D$ or $M < K$, information may be lost at the hidden units.
- A multilayer network of linear units (where all $h(\cdot)$ are linear) is not interesting and can be simplified to a network without hidden units.
- There may be more than one layer of hidden units.
- Individual units need not be fully connected to the next layer.
- Individual links may skip over one or more subsequent layers.
- There may be symmetries in the weight space, meaning that different choices of w may define the same mapping from input to output.

- The learning rule modifies the weights according to the input patterns that it is presented with.
- In a sense, artificial neural networks (ANNs) learn by examples, similar to their biological counterparts.
- When the desired outputs are known, the process is referred to as supervised learning.
- Each weight will be changed proportional to the error gradient.

Error (Loss) Function

Training a neural network involves estimating the weights and biases by optimizing a loss function. Let N represent the number of observations of the response $y_1, y_2, \ldots, y_N$, and let the neural network predict some output $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N$ (which could be a vector). We express the loss function as $l(y, \hat{y})$.

- Regression: Sum of squared

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

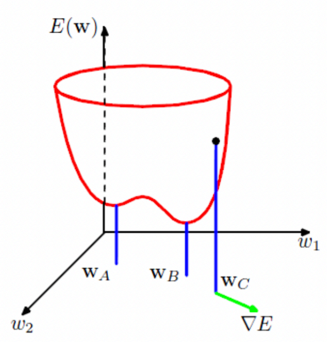
  where $t_n, n = 1, \ldots, N$ is the observed response. N is the total number if training samples (examples).
- Binary classification: NNs with only one output node whose activation function is a logistic sigmoid Cross-entropy error function

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

Gradients of Error Function

- $y(\boldsymbol{x}_n, \boldsymbol{w})$ is non-linear w.r.t $\boldsymbol{w}$
- Error function $E(\boldsymbol{w})$ is non-convex
- Gradient of $E(\boldsymbol{w})$, $\nabla E(\boldsymbol{w})$
  - Direction of greatest rate of increase
  - Minima, $E(\boldsymbol{w}) = 0$
  - No analytical solution
- Gradient descent: Moving through weight space in direction of $-\nabla E(\boldsymbol{w})$

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



Gradient Descent towards Global Minima

Learning by gradients decent algorithm: Require the evaluation of gradients wrt weight parameters.

How to Evaluate the Gradient

Need an efficient techniques for evaluating the gradient ∇E

Back Propogation Algorithm

Each iteration of the gradient decent algorithm has two stages:

- Evaluate derivatives of error w.r.t. weights
- Use derivatives to compute adjustments of the weights

Back Propagating

- For output units

$$\delta_k = y(\boldsymbol{x}_n, \boldsymbol{w}) - t_k.$$

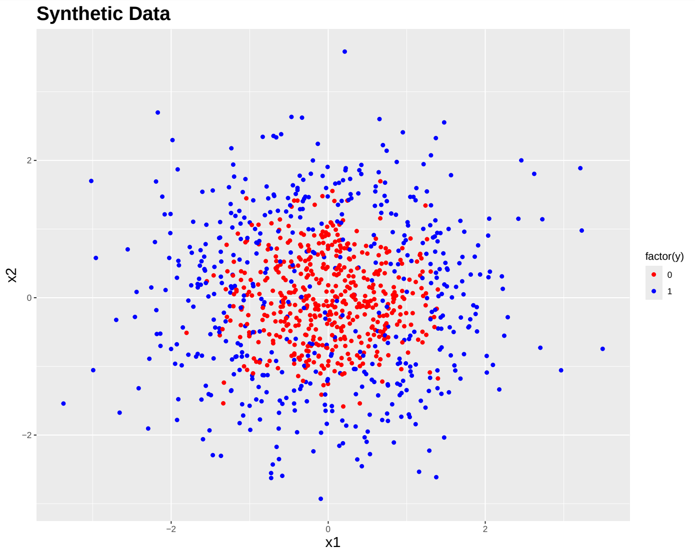
- For hidden units

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

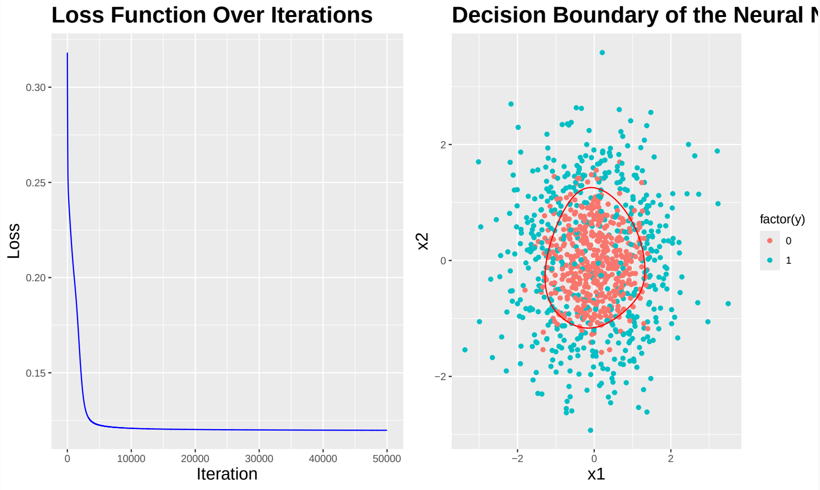
Summary

- Apply input x, and forward propagate to find the hidden and output activations.
- Evaluate δ_k directly for the outputs units
- Back propagate δ's to obtain a δ_j's for each hidden unit.
- Evaluate the derivatives $\frac{\partial E}{\partial a_j} = \delta_i Z_i$
- Sum these derivatives to over training cases to compute $\frac{\partial E}{\partial w_{ij}}$

Example: Classification Using two-Layer Neural Networks (1)



Example: Classification Using two-Layer Neural Networks (2)



Learning Issue

- Vanishing gradient problems due to propagated error coming from the other neurons and layers
    - Use of smart activation functions solve this issue
- Overfitting due to a lot number of hidden units the number of hidden units determines the complexity of the learned function (the number of parameters $\boldsymbol{w}$)
    -
    - We may allow regularizations
    - Drop of techniques
    - Sparse neural networks

Practical Rule of Thumb on deciding hidden units

- There is no theoretical results that determines the minimum number of hidden units.
- Practical rule of thumb
  - For binary data $M = 2D$
  - For real data $M >> 2D$
- Multiple hidden layers with fewer nodes maybe trained faster for similar quality in some applications