Deep Learning: Basics

S. Sumitra
Department of Mathematics
Indian Institute of Space Science and Technology

- Linear regression
  - $f(x) = w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_n x^{(n)}, x = (x^{(1)}, x^{(2)}, \ldots x^{(n)})$
- Linear Models of Regression
  - $f(x) = w_0 + w_1 \phi_1(x) + w_2 \phi_2(x) + \ldots w_M \phi_M(x)$
- Linear Models of Classification
  - $f(x) = h(w_0 + w_1 \phi_1(x) + w_2 \phi_2(x) + \ldots w_M \phi_M(x))$ where h is a non linear activation function
    - Perceptron $f(x) = sgn(w^T x + b)$
    - Logistic Regression: $f(x) = \frac{1}{1 + \exp(-w^T x)}$
- h is a nonlinear activation in the case of classification and identity in the case of regression

Perceptron

- $\{(x_1, y_1), (x_2, y_2) \ldots (x_N, y_N)\}, x_i \in \mathbb{R}^n, y_i \in \{1, -1\}$ be the given data
- Binary Classification
- Decision boundary: Hyperplane

Perceptron: Decision Boundary

- Decision Boundary

$$w^T x_i + w_0 = 0$$

- Prediction

$$f(x_i) = sgn(w^T x_i + w_0)$$

$$f(x_i) = \begin{cases} 1 & \text{if } w^T x_i + w_0 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Seperable Data

Introduction to Signum Function

- The signum function, denoted as $\text{sgn}(x)$, extracts the sign of a real number x.
- It is commonly used in mathematics to determine if a number is positive, negative, or zero.

Definition of Signum Function

The signum function is defined as:

$$\operatorname{sgn}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0 \end{cases}$$

In many practical implementations, to avoid the output being zero, the case where $x = 0$ is often defined to resolve to one of the binaries, typically $\operatorname{sign}(0) = 1$.

Cost Function

- Given a training set, how to choose the values of $w_i's$?
- Loss function
  - $\mathcal{L}_i(w, w_0)$ measures the discrepancy between the given output (y_i) and the predicted output ($f(x_i)$).
- Cost Function

$$J(w) = \frac{1}{N} \sum_i \mathcal{L}_i$$

Loss Function: Perceptron

- For Correct Prediction

$$y_i(w^T x_i + w_0) \geq 0$$

- Hinge loss function

$$\mathcal{L}_i(w, w_0) = \max(0, -y_i(w^T x_i + w_0))$$

Finding the Parameters

- Minimize the number of training errors
  - Optimization Problem

$$w^*, w_0^* = \arg\min J(w, w_0)$$

Hinge Loss Function

The hinge loss function for a single sample is defined as:

$$\begin{aligned} \mathcal{L}_i(w, w_0) &= \max(0, -y_i(w^T x_i + w_0)) \\ &= \xi \end{aligned}$$

where $\xi = \begin{cases} -y_i(w^T x_i + w_0) & \text{if } f(x_i) \neq y_i \\ 0 & \text{if } f(x_i) = y_i \end{cases}$

- Hinge loss function is not differentiable. Hence sub gradient has to be found.

Definition

A subgradient is a generalization of the concept of a derivative for convex functions that may not be differentiable at certain points.
Subgradients are particularly useful in optimization problems involving non-differentiable functions.

Key Concepts

- Subgradient of a Function: For a convex function f, a vector g is a subgradient at a point x_0 if:

$$f(x) \geq f(x_0) + g \cdot (x - x_0) \text{ for all } x.$$

- Subdifferential: The set of all subgradients at a point x_0 is called the subdifferential $\partial f(x_0)$.
- Convex Functions: Subgradients are relevant for convex functions, as they ensure the existence of subgradients.

Example: Subgradient of the Absolute Value Function

Function:

$$f(x) = |x|$$

- **For** $x > 0$:

$$\text{Subgradient} = 1$$

- **For** $x < 0$:

$$\text{Subgradient} = -1$$

- **At** $x = 0$:

$$\partial f(0) = [-1, 1]$$

Summary of Subgradients for $f(x) = |x|$

- For $x > 0 \Rightarrow$ Subgradient = 1
- For $x < 0 \Rightarrow$ Subgradient = -1
- For $x = 0 \Rightarrow$ Subgradient = any value in the interval $[-1, 1]$

Subgradient of Hinge Loss

- **Subgradient with respect to w:**

$$\frac{\partial \mathcal{L}_i}{\partial w} = \begin{cases} 0, & \text{if } y_i(w^T x_i + w_0) \geq 0 \\ -y_i x_i, & \text{if } y_i(w^T x_i + w_0) < 0 \end{cases}$$

- **Subgradient with respect to w_0:**

$$\frac{\partial \mathcal{L}_i}{\partial w_0} = \begin{cases} 0, & \text{if } y_i(w^T x_i + w_0) \geq 0 \\ -y_i, & \text{if } y_i(w^T x_i + w_0) < 0 \end{cases}$$

$$\frac{\partial \mathcal{L}_i}{\partial w} = \begin{cases} -y_i x_i & \text{if } f(x_i) \neq y_i \\ 0 & \text{if } f(x_i) = y_i \end{cases}$$

$$\frac{\partial \mathcal{L}_i}{\partial w_0} = \begin{cases} -y_i & \text{if } f(x_i) \neq y_i \\ 0 & \text{if } f(x_i) = y_i \end{cases}$$

Parameter Updation: Stochastic Gradient Descent

If $f(x_i) \neq y_i$

$$w := w + \alpha y_i x_i$$

$$w_0 := w_0 + \alpha y_i$$

Algorithm 1 Perceptron algorithm

Initialize w, w_0, $\alpha \in (0, 1)$

Iterate until covergence:

for $i = 1$ to N **do**

$w := w + \alpha y_i x_i$ if $y_i \neq f(x_i)$

$w_0 := w_0 + \alpha y_i$ if $y_i \neq f(x_i)$

end for

Randomly shuffle the data

end

Neural Networks

- Extension of linear models of classification and regression
  - $f(x) = h(\sum_{j=1}^{M} w_j \phi_j(x))$
- This model is extended by constructing the basis functions $\phi_j(x)$ to depend on parameters. Find the value of these parameters along with the coefficients $\{w_j\}$ during training.
  - Each basis function is itself a nonlinear function of a linear combination of inputs, where the coefficients in the linear combination has to be found from the data.

Linear Models

$x = (x^{(1)}, x^{(2)}, \ldots x^{(n)})^T$

- Linear Regression:
  - $f(x) = w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_n x^{(n)}$
  - $f(x) = I(w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_n x^{(n)})$
- Logistic Regression:
  - $w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_n x^{(n)}$
  - $f(x) = \sigma(w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_n x^{(n)}) = \frac{1}{1 + exp - (w_0 + w^T x)}$
- Perceptron:
  - $w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_n x^{(n)}$
  - $f(x) = sgn(w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_n x^{(n)})$

Composite Functions

- Function of functions: $f^{(n)} \ldots f^{(2)} f^{(1)}(x)$
- Layers of functions
  - $sincos(x), tansin(x+1)^2$

Neural Network Architecture

- Prediction function is a composite function
- Two layers: two functions
  - $f(x) = f^{(2)} f^{(1)}(x)$
- Construction of function in each layer
- The argument of each basic function consists of linear combination of the input of the current layer and the bias

Neural Network Terminologies

Model $f(x) = f^{(2)} f^{(1)}(x)$

- $f^{(1)}$ is called hidden function (hidden layer), $f^{(2)}$ is called output function (output layer)
- The basis used to represent the layers are called neurons
  - Neuron: h(linear combination of inputs + bias) or O(linear combination of inputs + bias)

Hyperparameters

- The number of neurons in first layer is a hyperparameter
- The number of neurons in second layer depends on the length of the prediction vector

Data

$\{(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)\}, x_m \in \mathbb{R}^n,$
$x_m = (x_{m1}, x_{m2}, \ldots x_{mn})^T$

Construction of First Layer ($f^{(1)}$)

- Data: x_m
- Input to the first layer: $x_m = (x_{m1}, x_{m2}, \ldots x_{mn})^T$
- $N^{(1)}$ basis
  - Number of neurons: $N^{(1)}$

Terminologies

- j^{th} neuron, $j = 1, 2, \ldots N^{(1)}$
- $w_{ji}^{(1)}$: j^{th} neuron's i^{th} component in the first layer

Construction of Neuron: First Layer

- Input to the first layer: $x_m = (x_{m1}, x_{m2}, \ldots x_{mn})^T$
- Linear combination of the inputs
  - $$z_j^{(1)} = w_{j1}^{(1)} x_{m1} + w_{j2}^{(1)} x_{m2} + \ldots w_{jn}^{(1)} x_{mn} + w_{j0}^{(1)} = \sum_{i=1}^{n} w_{ji}^{(1)} x_{mi} + w_{j0}^{(1)} \quad (1)$$
- Application of activation function
  - $$a_j^{(1)} = h(z_j^{(1)}) \quad (2)$$

Construction of Neuron: First Layer

- $z_j^{(1)}, j = 1, 2, \ldots N^{(1)}$
- $a_j^{(1)}, j = 1, 2, \ldots N^{(1)}$

Second Layer: Output layer

- Input: Output of the first layer
- Input: $a_1^{(1)}, a_2^{(1)}, \ldots a_{N^{(1)}}^{(1)}$
- The number of neurons (basis) depends on the length of the prediction vector
  - $N^{(2)}$ number of basis

Terminologies: Outputlayer

- k^{th} neuron, $k = 1, 2, \ldots N^{(2)}$
- $w_{kj}^{(2)}$: k^{th} neuron's j^{th} component in layer 2

Construction of Neuron: Second Layer

- Linear combination of the inputs
  - $$z_k^{(2)} = \sum_{j=1}^{N^{(1)}} w_{kj}^{(2)} a_j^{(1)} + w_{k0}^{(2)} \tag{3}$$
- Application of activation function
  - $a_k^{(2)} = O(z_k^{(2)})$

Construction of Neuron: Second Layer

- $z_k^{(2)}, k = 1, 2, \ldots N^{(2)}$
- $a_k^{(2)}, k = 1, 2, \ldots N^{(2)}$

Multilayer Perceptron

$$a_k^{(2)} = O\left(\sum_{j=1}^{N^{(1)}} w_{kj}^{(2)} a_j^{(1)} + w_{k0}^{(2)}\right)$$

$$a_k^{(2)} = O\left(\sum_{j=1}^{N^{(1)}} w_{kj}^{(2)} \left(h\left(\sum_{i=1}^{n} w_{ji}^{(1)} x_{mi} + w_{j0}^{(1)}\right)\right) + w_{k0}^{(2)}\right) \quad (4)$$

- The neural network has two stages of processing and each of that resembles the perceptron model and hence the neural network is known as multilayer perceptron or MLP.

Output Vector

$$\begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ . \\ . \\ . \\ a_{N^{(2)}}^{(2)} \end{bmatrix}$$

Multilayer Perceptron: Composite Function

- $f^{(2)}(f^{(1)}) = O(f^{(1)})$
- MLP model: $f^{(2)} f^{(1)}(x)$
- MLP (neural network) is a nonlinear function from a set of input data to a set of output variables controlled by a vector w of adjustable parameters.

Example

Data: $\{(x_1, y_1), (x_2, y_2) \ldots (x_N, y_N)\}, x_m \in \mathbb{R}^3, y_m \in \mathbb{R}$. Construct a MLP

- Hidden Layer: No of neurons: 2 (hyperparameters)
- Output Layer: one neuron

$z_1^{(1)} = w_{11}^{(1)} x_{m1} + w_{12}^{(1)} x_{m2} + w_{13}^{(1)} x_{m3} + w_{10}^{(1)}$ and $a_1^{(1)} = h(z_1^{(1)})$

$z_2^{(1)} = w_{21}^{(1)} x_{m1} + w_{22}^{(1)} x_{m2} + w_{23}^{(1)} x_{m3} + w_{20}^{(1)}$ and $a_2^{(1)} = h(z_2^{(1)})$

In output layer one neuron has to be constructed:

$z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)}$

$a_1^{(2)} = O(z_1^{(2)})$

$f(x_m) = O\left(\sum_{j=1}^{2} w_{1j}^{(2)} h(\sum_{i=1}^{3} w_{ji}^{(1)} x_{mi} + w_{j0}^{(1)}) + w_{10}^{(2)}\right)$

Representation Using Matrices and Vectors

$$z^{[1]} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} x_{m1} + w_{12}^{(1)} x_{m2} + w_{13}^{(1)} x_{m3} + w_{10}^{(1)} \\ w_{21}^{(1)} x_{m1} + w_{22}^{(1)} x_{m2} + w_{23}^{(1)} x_{m3} + w_{20}^{(1)} \end{bmatrix}$$

$$W^{[1]} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}.$$

$$b^{[1]} = \begin{bmatrix} w_{10}^{(1)} \\ w_{20}^{(1)} \end{bmatrix}$$

$$z^{[1]} = W^{[1]} x_m + b^{[1]}$$

$$a^{[1]} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \end{bmatrix} = \begin{bmatrix} h(z_1^{(1)}) \\ h(z_2^{(1)}) \end{bmatrix}$$

$$a^{[1]} = h(z^{[1]}) (\ h \text{ operates element wise})$$

$$z^{[2]} = \begin{bmatrix} z_1^{(2)} \end{bmatrix}$$

$$\begin{aligned} z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\ a^{[2]} &= O(z^{[2]}) = f(x_m) \end{aligned}$$

Composite Function

$$f(x_m) = O\left(W^{[2]}h(W^{[1]}x_m + b^{[1]}) + b^{[2]}\right)$$

$$f^{(1)}(x_m) = h(W^{[1]}x_m + b^{[1]})$$

,

$$f^{(2)}(f^{(1)}(x_m)) = O(W^{[2]}f^{(1)}(x_m) + b^{[2]})$$

$$f(x_m) = f^{(2)}f^{(1)}(x_m))$$

Hidden Activation Functions

- sigmoid : $h(z) = \frac{1}{1 + \exp(-z)}$ The range of this is $(0, 1)$
- hyperbolic tangent: $h(z) = tanhz = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. The range of this is $[-1, 1]$
- Rectified linear unit: Relu: $f(z) = \max(0, z)$

The rectified Linear Activation Function (ReLU)

- Relu(z) $= \max(0, z)$
- Default one
- This is a nonlinear transformation. As it has two linear pieces, it is very close to hyperplane. So they preserve many properties that make linear models easy to optimize with a gradient-based methods.

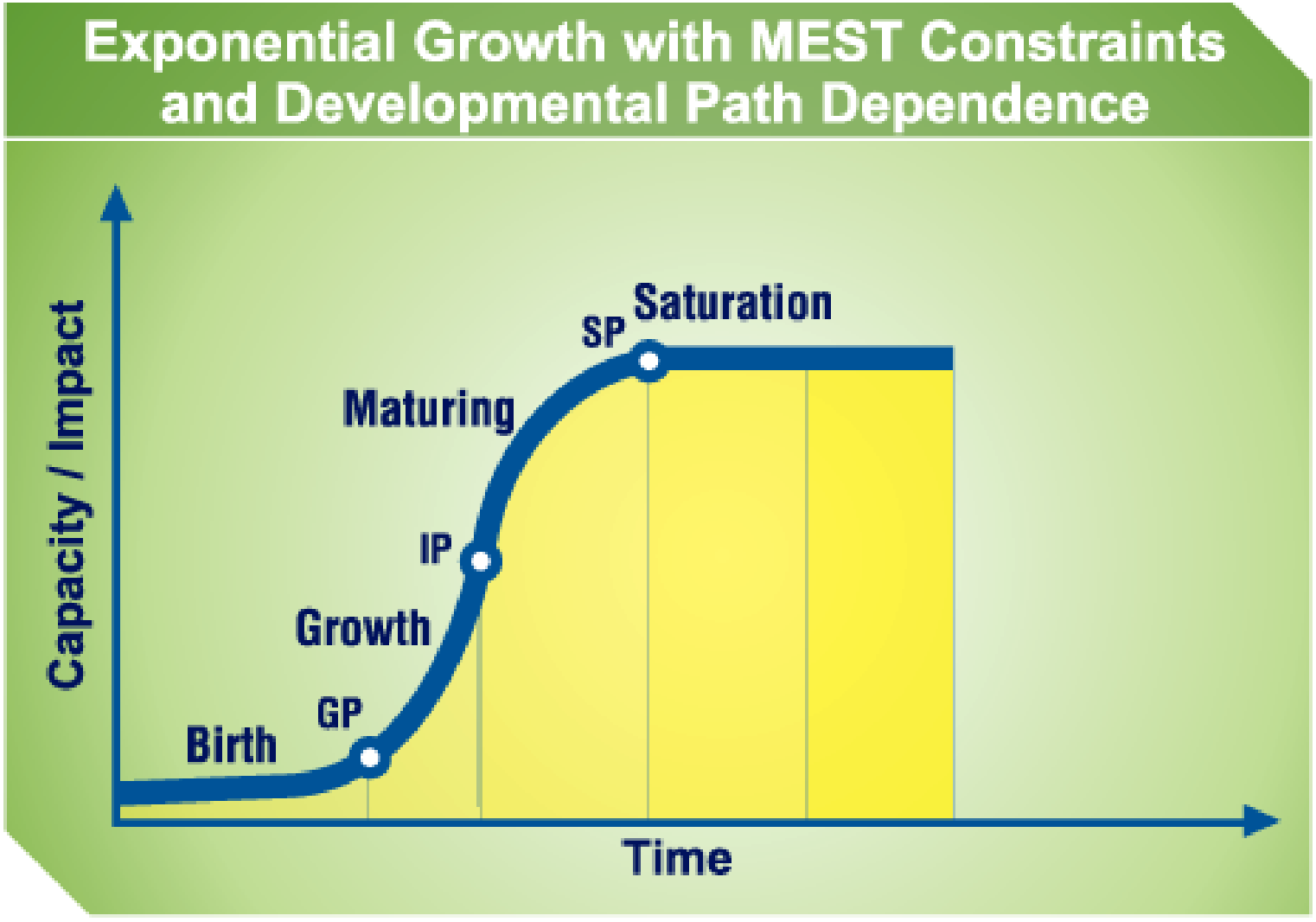
Variations of ReLU

- A drawback of ReLU is that they cannot learn via gradient based methods on examples for which their activation is zero.
- Various generalizations of ReLU guarantee that they receive gradient everywhere.
- $h_i = h(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$. Absolute value rectification fixes $\alpha_i = -1$ to obtain $h(z) = |z|$. A leaky ReLU fixes α_i to a small value like 0.01, while a parametric ReLU treats α_i as a learnable parameter.

Logistic Activation functions

- Logistic: $\sigma(z) = \frac{1}{1 + \exp(-z)}$
- Saturate across most of their domain- they saturate to a high value when z is very positive, saturate to a low value when z is very negative and are only strongly sensitive to their input when z is near 0. The widespread saturation of sigmoidal units can make gradient-based learning very difficult. So their usage is discouraged.
- RNN, many probabilistic models and some autoencoders have additional requirements that rule out the use of piecewise linear activation functions amd make sigmoidal units more appealing despite the drawbacks of saturation.

Logistic Sigmoid Function



Hyperbolic tangent Activation functions

- The hyperbolic tangent activation funtcion performs better than the logistic sigmoid
- It resembles the identity function more closely, since $\mathit{tanh}(0) = 0$ while $\sigma(0) = 1/2$

Other Hidden Activation Functions

- Radial basis function: $h(\mathbf{x}) = h(\|\mathbf{x} - \mathbf{c}\|)$.
- Softplus: $h(a) = \log(1 + e^a)$
- Hard tanh: $h(a) = \max(-1, \min(1, a))$

Output Actvation Functions

- Regression : Identity function
- Classification
  - Two class Classification: Sigmoid Units
  - Multiclass Classiffication: Softmax Units

Softmax Units

$$softmax(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Exponential Function in Softmax

Softmax Function: Converts logits to probabilities, ensuring outputs are non-negative and sum to 1.

Role of Exponential:

- **Non-negativity:** Guarantees positive values.
- **Sensitivity:** Amplifies differences between scores.
- **Normalization:** Orders scores and produces valid probabilities.

Example:

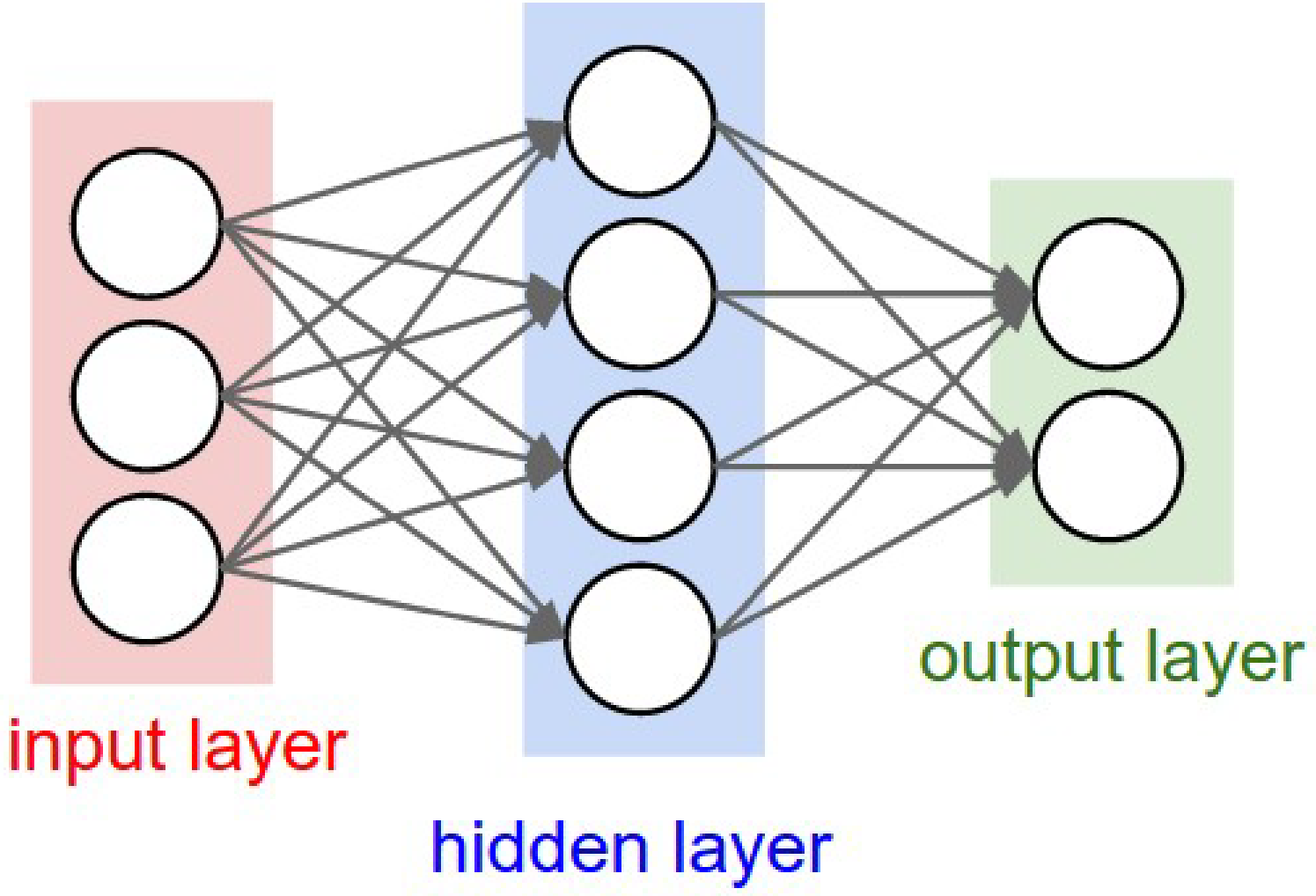
$$z_1 = 1,\ z_2 = 2,\ z_3 = 0;$$
$$e^{z_1} \approx 2.72,\ e^{z_2} \approx 7.39,\ e^{z_3} = 1;$$
$$S \approx 11.11;$$
$$\sigma(z_1) \approx 0.24,\ \sigma(z_2) \approx 0.66,\ \sigma(z_3) \approx 0.09$$

Class 2 has highest probability (66.4%).

Output Actvation Functions

- step function $f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$
- sgn function: $f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$

Neural Network



Feedforward Neural Networks

- A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle.
  - Perceptron
  - MLP

Characteristics

- Neural network with at least one hidden layer is a universal approximator (can represent any function).
- The capacity of the network increases with more hidden units and more hidden layers.
- The network we constructed above is a two layer network. A two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has sufficiently large number of hidden units. This result holds for a wide range of activation functions but excluding polynomials.
- Multilayer feed-forward networks, given enough hidden units and enough training samples can closely approximate any function.

Universal Approximation Theorem

- A feed-forward neural network with a single hidden layer can approximate any continuous function.
- Given sufficient neurons, the network can achieve any desired level of accuracy.

Formally:

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function on a compact subset of $\mathbb{R}^n$. For any $\varepsilon > 0$, there exists a neural network function $\hat{f}$ such that:

$$\sup_{x \in K} \left| f(x) - \hat{f}(x) \right| < \varepsilon$$

where K is a compact subset of $\mathbb{R}^n$.

No Free Lunch Theorem

- No single optimization algorithm is superior for all possible problems.
- An algorithm's effectiveness depends on the specific problem.
- If one algorithm performs better on one class of problems, it performs worse on another.

Contrast: Universal Approximation Theorem and No Free Lunch Theorem

- Universal Approximation Theorem:
  - States that feedforward neural networks can approximate any continuous function, given sufficiently many neurons and layers.
- No Free Lunch Theorem:
  - Asserts that no single machine learning algorithm is best for all problems; different tasks may favor different algorithms.
- Contrast:
  - Although neural networks can model any function, determining the right model configuration (including hyperparameters and training data selection) to generalize effectively beyond the training data is not universally solved.

Deep Learning Model

- $f(x) = f^{(n)}(f^{(n-1)} \dots f^{(3)}(f^{(2)}(f^{(1)}(x))))$.
- $f^{(1)}$ is called the first layer, $f^{(2)}$ is called the second layer and so on.
- The overall length of the chain is called the depth of the model. The term deep learning arose from this. The final layer is called output layer.
- The training points specify what the output layer should do: it has to produce a value close to y. The behavior of other layers are not specified by the training data. The learning algorithm must decide how to use them. So they are called hidden layers.
- Each hidden layer of the network is vector valued. The dimensionality of these hidden layers determines the width of the model. Each element of the vector can be considered as neuron.

Deep Learning Model: Depth 3

- $x_m \in \mathbb{R}^n$
- $N^{(l)}$: No of neurons in layer l, $l = 1, 2, 3$

Deep Learning Model: Depth 3

- Layer 1
  - $W^{[1]} : N^{(1)} \times n$
  - $z^{[1]} : N^{(1)} \times 1$
  - $b^{[1]} : N^{(1)} \times 1$
  - $a^{[1]} : N^{(1)} \times 1$
- Layer 2
  - $W^{[2]} : N^{(2)} \times N^{(1)}$
  - $z^{[2]} : N^{(2)} \times 1$
  - $b^{[2]} : N^{(2)} \times 1$
  - $a^{[2]} : N^{(2)} \times 1$
- Layer 3
  - $W^{[3]} : N^{(3)} \times N^{(2)}$
  - $z^{[3]} : N^{(3)} \times 1$
  - $b^{[3]} : N^{(3)} \times 1$
  - $a^{[3]} : N^{(3)} \times 1$

Deep Learning Model: Depth 3

$$
\begin{aligned}
z^{[1]} &= W^{[1]} x_i + b^{[1]} \\
a^{[1]} &= h(z^{[1]}) \\
z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\
a^{[2]} &= h(z^{[2]}) \\
z^{[3]} &= W^{[3]} a^{[2]} + b^{[3]} \\
f(x_i) &= a^{[3]} = O(z^{[3]})
\end{aligned}
$$

Deep Learning Model: Depth 3

$$f(x_m) = O\left(W^{[3]}h\left(W^{[2]}\left(h\left(W^{[1]}x_m + b^{[1]}\right)\right) + b^{[2]}\right) + b^{[3]}\right)$$

Deep Learning Model: Depth L

$$f(x_m) = O\left(W^{[L]}h\left(W^{[L-1]}\left(h(\ldots)\right) + b^{[L-1)}\right) + b^{[L)}\right)$$

Deep Learning Model: Depth L

$$z^{(l)} = [z_1^{(l)}, z_2^{(l)}, \ldots z_{N^{(l)}}^{(l)}]^T,\ a^{(l)} = [a_1^{(l)} a_1^{(l)}, a_2^{(l)}, \ldots, a_{N^l}^{(l)}]^T$$

$$W^{(l)} = \begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \cdots & w_{1,N^{(l-1)}}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \cdots & w_{2,N^{(l-1)}}^{(l)} \\ \vdots & \vdots & \cdots & \vdots \\ w_{N^{(l)},1}^{(l)} & w_{N^{(l)},2}^{(l)} & \cdots & w_{N^{(l)},N^{(l-1)}}^{(l)} \end{pmatrix}$$

Deep Learning Model

A supervised deep learning model of depth L is a composite function:

$$f : \mathcal{X} \to \mathbb{R}^{N(L)}$$

where

$$f = f^L \circ f^{L-1} \circ \cdots \circ f_1$$

- $f_1 : \mathcal{X} \to \mathbb{R}^{N(1)}$ where $f_1(x) = h(W^{(1)}x + b^{(1)})$
- $f_i : \text{Range}(f_{i-1}) \to \mathbb{R}^{N(i)}$ for $i = 2, 3, \ldots, L-1$ where $f_i(\tilde{x}) = h(W^{(i)}\tilde{x} + b^{(i)})$ and h is an appropriate hidden activation function
- $f^L : \text{Range}(f^{L-1}) \to \mathbb{R}^{N(L)}$ where $f_L(\hat{x}) = O(W^L\hat{x} + b^{(L)}$ and O is an appropriate output activation function

What happens if all the activation functions are identity functions?

$$
\begin{aligned}
z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\
&= W^{[2]} h(z^{[1]}) + b^{[2]} \\
&= W^{[2]} z^{[1]} + b^{[2]} [h \equiv I] \\
&= W^{[2]} W^{[1]} x \\
&= \tilde{W} x + b^{[2]}
\end{aligned}
$$

Without non-linear activation functions, the neural network behaves as a linear transformation of the input space, which limits its ability to represent complex relationships within the data.

Cost Function

- Given a training set, how to choose the values of $w_i's$?
- Loss function ($L(y_i, f(x_i))$ measures the discrepancy between the given output (y_i) and the predicted output ($f(x_i)$).
- Cost Function

$$J(w) = \frac{1}{N} \sum_i L(y_i, f(x_i))$$

Cost function

- Least square cost function $\frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2$
- Hinge loss function $\sum_{i=1}^{N} \max(0, -y_i(w^T x_i + w_0))$
- Cross entropy loss function
  - The cross entropy formula takes in two distributions, $p(x)$, the true distribution, and $q(x)$, the estimated distribution, defined over the discrete variable X and is given by $H(p, q) = -\sum_x p(x) log(q(x))$

Cross entropy loss function:Two Class Classification

- Logistic Regression: Two class
  - $(x_i, y_i), y_i \in 1, 0)$
  - $f(x_i)$ is the probability x_i belongs to the positive class.
    $\mathcal{L}(y_i, \tilde{y_i}) = -(y_i \log f(x_i) + (1 - y_i) \log(1 - f(x_i))$

Cross Entropy Loss Function: Dot Product

- For a positive class, let $y = (1, 0)^T$
- For a negative class, let $y = (0, 1)^T$
- The cross-entropy loss for a single data point is given by:

$$\mathcal{L}(y, \hat{y}) = -\langle y, \log \tilde{y} \rangle$$

where y is the ground truth label vector, and $\tilde{y}$ is the predicted probability vector, with the logarithm applied to each element.

Cross entropy loss function: k class Classification

- Ground truth: $y_m \in \mathbb{R}^k$
- $y_m = (y_{m1}, y_{m2}, \ldots y_{mk})^T$, where $y_{mj} = 1$, if x_m belongs to the j^{th} class else $y_{mj} = 0$
- Predicted probability $a^{(L)} = \sigma(z^{(L)})$, where σ is the softmax function
- $L(y_m, \tilde{y_m}) = -y_m^T log a^{(L)} = -\sum_{j=1}^{k} y_{mj} log a_j^{(L)}$