Deep Neural Network
Module # 1 : Fundamentals of Neural Network

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors
who made their course materials freely available online.

Table of Contents

1. Course Logistics
2. Introduction to Deep Learning
3. Applications of Deep Learning
4. Key Components of DL Problem
5. Artificial Neural Network
6. Perceptron
7. Single Perceptron for Linear Regression
8. Multi Layer Perceptron (MLP)

What we Learn...

1. Fundamentals of Neural Network and Multilayer Perceptron
2. Deep Feed-forward Neural Network
3. Improve its performance by Optimization
4. Improve its performance by Regularization
5. Convolutional Neural Networks
6. Sequence Models
7. Attention Mechanism
8. Neural Network search
9. Time series Modelling and Forecasting
10. Other Learning Techniques

Text and Reference Books

Text Books

T1	Dive into Deep Learning by Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola. `https://d2l.ai/chapter_introduction/index.html`
R1	Deep Learning by Ian Goodfellow, YoshuaBengio, Aaron Courville `https://www.deeplearningbook.org/`

Evaluation Schedule

No	Name	Type	Duration	Weight	Day, Date, Time
EC1	Quiz I	Online	0.5 hr	5 %	Check Canvas
	Quiz II	Online	0.5 hr	5 %	Check Canvas
	Assignment I	Online	4 weeks	15 %	Check Canvas
	Assignment II	Online	4 weeks	15 %	Check Canvas
EC2	Mid-sem Regular	Closed book	2 hrs	30%	Check Calendar
EC3	Compre-sem Regular	Open book	2.5 hrs	30%	Check Calendar

Lab Sessions

1. Python Libraries – Keras and Tensorflow

L1 Introduction to Tensorflow and Keras

L2 Deep Neural Network with Back-propagation and optimization

L3 CNN

L4 RNN

L5 LSTM

L6 Auto-encoders

LMS

Refer Canvas for the following

- Handout
- Schedule for Webinars
- Schedule of Quiz, and Assignments.
- Evaluation scheme
- Session Slide Deck
- Demo Lab Sheets
- Quiz-I, Quiz-II
- Assignment-I, Assignment-II
- Sample QPs

The Lecture Recordings are made available on Microsoft Teams.

Honor Code

All submissions for graded components must be the result of your original effort. It is strictly prohibited to copy and paste verbatim from any sources, whether online or from your peers. The use of unauthorized sources or materials, as well as collusion or unauthorized collaboration to gain an unfair advantage, is also strictly prohibited. Please note that we will not distinguish between the person sharing their resources and the one receiving them for plagiarism, and the consequences will apply to both parties equally.

In cases where suspicious circumstances arise, such as identical verbatim answers or a significant overlap of unreasonable similarities in a set of submissions, will be investigated, and severe punishments will be imposed on all those found guilty of plagiarism.

In case of Queries regarding the course ...

Step 1 : Post in the discussion forum.
Read through the existing posts and if you find any topic similar to your concern, add on to the existing discussion.
Avoid duplication of queries or issues. This is highly appreciated.

Step 2 : Email to the IC at seetha.p@pilani.bits-pilani.ac.in if the query or issue is not resolved within one weeks' time. Turn around for response to the email is 48hrs.
In the subject please mention the phrase **"DNN"** clearly.
Use BITS email id for correspondence. Emails from personal emails will be ignored without any reply.

PATIENCE is highly APPRECIATED :)

Table of Contents

1. Course Logistics
2. Introduction to Deep Learning
3. Applications of Deep Learning
4. Key Components of DL Problem
5. Artificial Neural Network
6. Perceptron
7. Single Perceptron for Linear Regression
8. Multi Layer Perceptron (MLP)

What is Deep Learning?

- Deep Learning is a type of **machine learning** based on **artificial neural networks** in which multiple layers of processing are used to extract progressively higher level features from data.
- Deep learning is a method in artificial intelligence (AI) that teaches computers to process data in a way that is **inspired by the human brain**.
- Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: **learn by example**.
- Deep learning is a subset of machine learning, which is essentially a neural network with **three or more layers**.
- Deep Learning gets its name from the fact that we add more **Layers** to learn from the data.

Where in AI sits DL?

- AI is a general field that encompasses machine learning and deep learning, but that also includes many more approaches that don't involve any learning.

AI – ML – DL

AI : Artificial intelligence is the **science** of making things smart. The aim is make machines perform human tasks. Eg: Robot cleaning a room.

ML : Machine learning is an **approach** to AI. The machine learns or perform tasks through learning by experience.

DL : Deep Learning is a **technique** for implementing machine learning to recognise patterns.

Deep (Machine) Learning

- Deep learning is a specific subfield of machine learning.
- Learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations.
- The **deep** in deep learning stands for this idea of successive layers of representations.
- The number of layers that contribute to model the data is called the **depth** of the model.
- In deep learning, the layered representations are learned via models called **neural networks**, structured in literal layers stacked on top of each other.

Why Deep Learning?

- Large amounts of data
- Lots and lots of unstructured data like images, text, audio, video
- Cheap, high-quality sensors
- Cheap computation - CPU, GPU, Distributed clusters
- Cheap data storage
- Learn by examples
- Automated feature generation
- Better learning capabilities
- Scalability
- Advance analytics can be applied

Why Deep Learning?

Deep Learning: Timeline

Table of Contents

1. Course Logistics
2. Introduction to Deep Learning
3. Applications of Deep Learning
4. Key Components of DL Problem
5. Artificial Neural Network
6. Perceptron
7. Single Perceptron for Linear Regression
8. Multi Layer Perceptron (MLP)

Breakthroughs with Neural Networks

TECHNEWSWORLD EMERGING TECH SEARCH

Computing Internet IT Mobile Tech Reviews Security Technology Tech Blog Reader Service

Microsoft AI Beats Humans at Speech Recognition

By Richard Adhikari
Oct 20, 2016 11:40 AM PT

Print
Email

Image: Adobe Stock

Microsoft's Artificial Intelligence and Research Unit earlier this week reported that its speech recognition technology had surpassed the performance of human transcriptionists.

Most Popular Newsletters News Aler

How do you feel about Black Friday and Cyber Monday?

- They're great -- I get a lot of bargains!
- The deals are too spread out -- I'd prefer just one day.
- They're a fun way to kick off the holiday season.
- I don't like the commercialization of Thanksgiving Day.
- They're crucial for the retail industry and the economy.
- The deals typically aren't that good.

Vote to See Results

E-Commerce Times

Black Friday Shoppers Hungry for New Experiences, New Tech

Pay TV's Newest Innovation: Giving Users Control

Apple Celebrates Itself in $300 Coffee Table Tome

AWS Enjoys Top Perch in IaaS, PaaS Markets

US Comptroller Gears Up for Blockchain and

Breakthroughs with Neural Networks

Image Segmentation and Recognition

Image Recognition

Breakthroughs with Neural Networks

Figure 1: Training *AlphaZero* for 700,000 steps. Elo ratings were computed from evaluation games between different players when given one second per move. **a** Performance of *AlphaZero* in chess, compared to 2016 TCEC world-champion program *Stockfish*. **b** Performance of *AlphaZero* in shogi, compared to 2017 CSA world-champion program *Elmo*. **c** Performance of *AlphaZero* in Go, compared to *AlphaGo Lee* and *AlphaGo Zero* (20 block / 3 day) (*29*).

Breakthroughs with Neural Networks

Applications of Deep Learning

Application	Input	Output	Neural Network
Real Estate	House features	House Price	Std NN
Photo Tagging	Image	Text	CNN
Object detection	Image	Bounding box	CNN
Speech Recognition	Audio	Text transcript	RNN
Translation	English Text	French Text	RNN
Autonomous driving	Image, Sensors Radars	Position of other cars, objects, signals	Hybrid NN

Many more applications...

- a program that predicts tomorrow's weather given geographic information, satellite images, and a trailing window of past weather.
- a program that takes in a question, expressed in free-form text, and answers it correctly.
- a program that given an image can identify all the people it contains, drawing outlines around each.
- a program that presents users with products that they are likely to enjoy but unlikely, in the natural course of browsing, to encounter.

Table of Contents

1. Course Logistics
2. Introduction to Deep Learning
3. Applications of Deep Learning
4. Key Components of DL Problem
5. Artificial Neural Network
6. Perceptron
7. Single Perceptron for Linear Regression
8. Multi Layer Perceptron (MLP)

Core Components of DL Problem

- The **data** that we can learn from.
- A **model** of how to transform the data.
- An **objective function** that quantifies how well (or badly) the model is doing.
- An **algorithm** to adjust the model's parameters to optimize the objective function.

1. Data

- Collection of examples.
- Data has to be converted to an useful and a suitable numerical **representation**.
- Each example (or data point, data instance, sample) typically consists of a set of attributes called **features** (or covariates), from which the model must make its predictions.
- In the supervised learning problems, the attribute to be predicted is designated as the **label** (or target).
- Mathematically, a set of m examples,

$$Data = \mathcal{D} = \{X, t\}$$

- We need right data.

1. Data

- Dimensionality of data
  - Each example has the same number of numerical values. This data consist of fixed-length vectors. Eg: Image
  - The constant length of the vectors as the dimensionality of the data.
  - Text data has varying-length data.

2. Model

- Model denotes the computational machinery for ingesting data of one type, and spitting out predictions of a possibly different type.
- Deep learning models consist of many successive transformations of the data that are chained together top to bottom, thus the name deep learning.

3. Objective Function

- **Learning means improving at some task over time.**
- A formal mathematical system of learning machines is defined using formal measures of how good (or bad) the models are. These formal measures are called as **objective functions**.
- By convention, objective functions are defined so that lower is better. Because lower is better, these functions are sometimes called **loss functions**.

3. Loss Functions

- To predict numerical values (regression), the most common loss function is squared error.
- For classification, the most common objective is to minimize error rate, i.e., the fraction of examples on which our predictions disagree with the ground truth.

3. Loss Functions

- Loss function is defined
  - with respect to the model's parameters
  - depends upon the dataset.
- We learn the best values of our model's parameters by **minimizing the loss** incurred on a set consisting of some number of examples collected for training. However, doing well on the training data does not guarantee that we will do well on unseen data. i.e **Model has to generalize better.**
- When a model performs well on the training set but fails to generalize to unseen data, we say that it is **overfitting**.

4. Optimization Algorithms

- Optimization Algorithm is an algorithm capable of **searching for the best possible parameters for minimizing the loss function**.
- Popular optimization algorithms for deep learning are based on an approach called **gradient descent**.

Example of the Framework

- We have to tell a computer explicitly how to map from inputs to outputs.
- We have to define the problem precisely, pinning down the exact nature of the inputs and outputs, and choosing an appropriate model family.
- Collect a huge dataset containing examples of audio and label those that do and that do not contain the wake word.

Example of the Framework

- Create a Model
  - Define a flexible program whose behavior is determined by a number of parameters.
  - To determine the best possible set of parameters, use the data. The parameters should improve the performance of the program with respect to some measure of performance on the task of interest.
  - After fixing the parameters, we call the program a model.
    Eg: The model receives a snippet of audio as input, and the model generates a selection among yes, no as output.
  - The set of all distinct programs (input-output mappings) that we can produce just by manipulating the parameters is called a family of models.
    Eg: We expect that the same model family should be suitable for ”Alexa” recognition and ”Hey Siri” recognition because they seem, intuitively, to be similar tasks.

Example of the Framework

- The meta-program that uses our dataset to choose the parameters is called a learning algorithm.
- In machine learning, the learning is the process by which we discover the right setting of the parameter coercing the desired behavior from our model.
- **Train the model with data.**

Fig. 1.1.2: A typical training process.

Table of Contents

1. Course Logistics
2. Introduction to Deep Learning
3. Applications of Deep Learning
4. Key Components of DL Problem
5. Artificial Neural Network
6. Perceptron
7. Single Perceptron for Linear Regression
8. Multi Layer Perceptron (MLP)

What are Neural Networks??

What are Neural Networks???

It begins with brain.

- Humans learn, solve problems, recognize patterns, create, think deeply about something, meditate and many many more.....
- Humans learn through association. [Refer to Associationism for more details.]

Observation: The Brain

- The brain is a mass of interconnected neurons.
- Number of neurons is approximately 10^{10}.
- Connections per neuron is approximately $10^{(4\ to\ 5)}$.
- Neuron switching time is approximately 0.001 second.
- Scene recognition time is 1 second.
- 100 inference steps doesn't seem like enough. Lot of parallel computation.

Brain: Interconnected Neurons

- Many neurons connect **in** to each neuron.
- Each neuron connects **out** to many neurons.

Biological Neuron

Connectionist Machines

[1]

- Network of processing elements, called artificial neural unit.
- The neurons are interconnected to form a network.
- All world knowledge is stored in the connections between these elements.
- Neural networks are connectionist machines.

[1] alanturing.net

Artificial Neuron

- Neuron is a processing element inspired by how the brain works.
- Similar to biological neuron, each artificial neuron will be do some computation. Each neuron is interconnected to other neurons.
- Similar to brain, the interconnections between neurons store the knowledge it learns. The knowledge is stored as parameters.

Properties of Artificial Neural Nets (ANNs)

- Many neuron-like threshold switching units.
- Many weighted interconnections among units.
- Highly parallel, distributed process.
- Emphasis on tuning parameters or weights automatically.

When to consider Neural Networks?

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input).
- Possibly noisy data. Data has lots of errors.
- Output is discrete or real valued or a vector of values.
- Form of target function is unknown.
- Human readability, in other words, explainability, of result is unimportant.
- Examples:
  - Speech phoneme recognition
  - Image classification
  - Financial prediction

Table of Contents

1. Course Logistics
2. Introduction to Deep Learning
3. Applications of Deep Learning
4. Key Components of DL Problem
5. Artificial Neural Network
6. Perceptron
7. Single Perceptron for Linear Regression
8. Multi Layer Perceptron (MLP)

Perceptron

- One type of ANN system is based on a unit called a perceptron.
- **A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.**

Representing Logic Gates using Perceptron

- Any linear decision can be represented by either a linear regression equation or a Boolean equation.
- The quest is to find out how to represent the above using Perceptron.
- For this we test whether each of the logic gates can be represented by a Perceptron.

NOT Gate

Question:

- How to represent NOT gate using a Perceptron?
- What are the parameters for the NOT Perceptron?
- Data is given below.

A	B
0	1
1	0

Rewrite as

x_1	t
-1	1
1	-1

Perceptron for NOT gate

- Perceptron equation is $\hat{y} = w_0 x_0 + w_1 x_1$.
- $x_0 = 1$ always.
- $h > 0$ for output to be 1.
- For each row of truth table, the equations are given.
- One solution is $w_0 = 1$ and $w_1 = -1$. (Intuitive solution)
- This give a beautiful linear decision boundary.

x_1	t_1	h	h
-1	1	$w_0 x_0 + w_1(-1) > 0$	$w_0 - w_1 > 0$
1	-1	$w_0 x_0 + w_1(1) < 0$	$w_0 + w_1 < 0$

AND Logic Gate

Question:

- How to represent AND gate using a Perceptron?
- What are the parameters for the AND Perceptron?
- Data is given below.

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Rewrite as

x_1	x_2	t
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Perceptron for AND gate

- Perceptron equation is $\hat{y} = w_0 x_0 + w_1 x_1 + w_2 x_2$.
- $h > 0$ for output to be 1.
- For each row of the truth table, the equations are given.
- Solve for the inequalities.
- One solution is $w_1 = w_2 = 2, w_0 = (-1)$.
- This give a beautiful linear decision boundary.

x_1	x_2	t	h
-1	-1	-1	$w_0 + w_1(-1) + w_2(-1) < 0$
-1	1	-1	$w_0 + w_1(-1) + w_2(1) < 0$
1	-1	-1	$w_0 + w_1(1) + w_2(-1) < 0$
1	1	1	$w_0 + w_1(1) + w_2(1) > 0$

Perceptron for AND gate

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Perceptron and Decision boundary for AND gate

Exercises

- Represent OR gate using Perceptron. Compute the parameters of the Perceptron.
- Represent NOR gate using Perceptron. Compute the parameters of the Perceptron.
- Represent NAND gate using Perceptron. Compute the parameters of the Perceptron.

OR Logic Gate

Question:

- How to represent OR gate using a Perceptron?
- What are the parameters for the OR Perceptron?
- Data is given below.

A	B	AB
0	0	0
0	1	1
1	0	1
1	1	1

Rewrite as

x_1	x_2	t
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

Perceptron for OR gate

- Perceptron equation is $\hat{y} = w_0 x_0 + w_1 x_1 + w_2 x_2$.
- $h > 0$ for output to be 1.
- For each row of the truth table, the equations are given.
- Solve for the inequalities.
- One solution is $w_0 = w_1 = w_2 = 2$.
- This give a beautiful linear decision boundary.

x_1	x_2	t	h
-1	-1	-1	$w_0 + w_1(-1) + w_2(-1) < 0$
-1	1	-1	$w_0 + w_1(-1) + w_2(1) > 0$
1	-1	-1	$w_0 + w_1(1) + w_2(-1) > 0$
1	1	1	$w_0 + w_1(1) + w_2(1) > 0$

Perceptron for OR gate

A	B	AB
0	0	0
0	1	1
1	0	1
1	1	1

Perceptron and Decision boundary for OR gate

Perceptron Learning Algorithm

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - \hat{y})x_i$$

where :

$$t = \text{target value}$$

$$\hat{y} = \text{perceptron output}$$

$$\eta = \text{learning rate}$$

Convergence of Perceptron Learning Algorithm

It can be proved that the algorithm will converge

- If training data is linearly separable.
- Learning rate is sufficiently small.
  - The role of the learning rate is to moderate the degree to which weights are changed at each step.
  - It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

Perceptron Learning for NOT gate

- Equations are

$$
\begin{aligned}
\text{Hypothesis} \quad & z = w_0 + w_1 x_1 \\
\text{Activation} \quad & h = sign(z) \\
\text{Compute output} \quad & \hat{y} = h = sign(w_0 + w_1 x_1) \\
\text{Compute } \Delta w & = \eta(t - \hat{y})x \\
\text{Update } w_{new} & \leftarrow w_{old} + \Delta w
\end{aligned}
$$

Perceptron Learning for NOT gate

- Assume $w_0 = w_1 = 0$. Let the learning rate$= \eta = 1$.
- Epoch 1

x_1	t	w_1	w_0	z	h	$isequal(t, h)$	Δw	New w
0	1	0	0	$0 + 0 = 0$	-1	Not equal	$\Delta w_1 = 1(1 - (-1))0 = 0$ $\Delta w_0 = 1(1 - (-1)) = 2$	$w_1 \leftarrow 0 + 0 = 0$ $w_0 \leftarrow 0 + 2 = 2$
1	-1	0	2	$2 + 0 = 2$	1	Not equal	$\Delta w_1 = 1(-1 - 1)1 = -2$ $\Delta w_0 = 1(-1 - 1) = -2$	$w_1 \leftarrow 0 + -2 = -2$ $w_0 \leftarrow 2 + -2 = 0$

- Epoch 2

x_1	t	w_1	w_0	z	h	$isequal(t, h)$	Δw	New w
0	1	-2	0	$0 + 0 = 0$	-1	Not equal	$\Delta w_1 = 1(1 - (-1))0 = 0$ $\Delta w_0 = 1(1 - (-1)) = 2$	$w_1 \leftarrow -2 + 0 = -2$ $w_0 \leftarrow 0 + 2 = 2$
1	-1	-2	2	$2 + -2 = 0$	-1	Equal		

- Algorithm Converges.

Demo Python Code

- https://colab.research.google.com/drive/1DUVcOoUIWhl8GQKc6AWR1wi0LaMeNkgD?usp=sharing

Student pl note:
The Python notebook is shared for anyone who has access to the link and the access is restricted to use BITS email id. So please do not access from non-BITS email id and send requests for access. Access for non-BITS email id will nto be granted.

Exercise

- Represent OR gate using Perceptron. Compute the parameters of the Perceptron using Perceptron learning algorithm.
- Represent AND gate using Perceptron. Compute the parameters of the Perceptron using Perceptron learning algorithm.

Representational Power of Perceptron

- A Perceptron represents a hyperplane decision surface in the n-dimensional space of examples.
- The Perceptron outputs a 1 for examples lying on one side of the hyperplane and outputs a -1 for examples lying on the other side.

(a) (b)

Linearly Separable Data

- Two sets of data points in a two dimensional space are said to be linearly separable when they can be completely separable by a single straight line.
- A straight line can be drawn to separate all the data examples belonging to class +1 from all the examples belonging to the class -1. Then the two-dimensional data are clearly linearly separable.
- An infinite number of straight lines can be drawn to separate the class +1 from the class -1.

Perceptron for Linearly Separable Data

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

$$\sum = w_0 + w_1 x_1 + \cdots + w_n x_n$$
$$= w^\top x$$

$$\hat{y} = \begin{cases} 1 & \text{if } w^\top x > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Challenge: How to learn these n parameters $w_0 \ldots w_n$?

Solution: Use Perceptron learning algorithm.

Perceptron Training

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Perceptron training rule will converge

- If training data is linearly separable
- η sufficiently small

Perceptron and its Learning - Review

- Data
  - Truth tables or set of examples
- Model
  - Perceptron
- Objective Function
  - Deviation of desired output t and the computed output $\hat{y}$
- Learning Algorithm
  - Perceptron Learning algorithm

Table of Contents

1. Course Logistics
2. Introduction to Deep Learning
3. Applications of Deep Learning
4. Key Components of DL Problem
5. Artificial Neural Network
6. Perceptron
7. Single Perceptron for Linear Regression
8. Multi Layer Perceptron (MLP)

Linear Regression Example

- Suppose that we wish to estimate the prices of houses (in dollars) based on their area (in square feet) and age (in years).
- The linearity assumption just says that the target (price) can be expressed as a weighted sum of the features (area and age):

$$price = w_{area} * area + w_{age} * age + b$$

  w_{area} and w_{age} are called weights, and b is called a bias.
- The weights determine the influence of each feature on our prediction and the bias just says what value the predicted price should take when all of the features take value 0.

Data

- The dataset is called a **training dataset or training set**.
- Each row is called an **example** (or data point, data instance, sample).
- The thing we are trying to predict is called a **label** (or target).
- The independent variables upon which the predictions are based are called **features** (or covariates).

$$
\begin{aligned}
m &- \text{ number of training examples} \\
i &- \text{ index of } i^{th} \text{ example} \\
x^{(i)} &- \text{ features of } i^{th} \text{ example} \\
&= [x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}] \\
y^{(i)} &- \text{ label corresponding to } i^{th} \text{ example} \\
&= [y^{(1)}, y^{(2)}, y^{(3)}, \ldots, y^{(m)}]
\end{aligned}
$$

Affine transformations and Linear Models

- Affine transformations are

$$\begin{aligned}\hat{y} &= w_1x_1 + w_2x_2 + \ldots + w_dx_d + b \\ &= w^Tx + b \\ \hat{y} &= WX + b\end{aligned}$$

- $\hat{y} = WX + b$ is an **affine transformation** of input features, which is characterized by a linear transformation of features via weighted sum, combined with a translation via the added bias.
- Models whose output prediction is determined by the affine transformation of input features are **linear models**.
- The affine transformation is specified by the chosen **weights (w)** and **bias (b)**.

Loss Function

- Loss function is a quality measure for some given model or a measure of fitness.
- The loss function quantifies the distance between the real and predicted value of the target.
- The loss will usually be a non-negative number where smaller values are better and perfect predictions incur a loss of 0.
- The most popular loss function in regression problems is the squared error.

Squared Error Loss Function

- The most popular loss function in regression problems is the squared error.
- For each example,

$$loss^{(i)}(w, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

- For the entire dataset of m examples average (or equivalently, sum) the losses

$$L(w, b) = \frac{1}{m}\sum_{i=1}^{m} loss^{(i)}(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left[\frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2\right]$$

- When training the model, find parameters (w_{opt}, b_{opt}) that minimize the total loss across all training examples.

$$w_{opt}, b_{opt} = argmin_{w,b} \quad L(w, b)$$

Single-Layer Neural Network

Linear regression is a single-layer neural network.

- Number of inputs (or feature dimensionality) in the input layer is d. The inputs are $x_1, \ldots, x_d$.
- Number of outputs in the output layer is 1. The output is $\hat{y}_1$.
- Number of layers for the neural network is 1. (conventionally we do not consider the input layer when counting layers.)
- Every input is connected to every output, This transformation is a **fully-connected layer or dense layer.**

Single-Layer Neural Network

Table of Contents

1. Course Logistics
2. Introduction to Deep Learning
3. Applications of Deep Learning
4. Key Components of DL Problem
5. Artificial Neural Network
6. Perceptron
7. Single Perceptron for Linear Regression
8. Multi Layer Perceptron (MLP)

Non-linearly Separable Data

- Two groups of data points are non-linearly separable in a 2-dimensional space if they cannot be easily separated with a linear line.

XOR Logic Gate

Question:

- How to represent XOR gate using a Perceptron?
- What are the parameters for the XOR Perceptron?
- Data is given below.

A	B	AB
0	0	0
0	1	1
1	0	1
1	1	0

Challenge: Data is non-linearly separable.

Decision boundary for XOR

MLP for XOR gate

Qn: How to represent XOR gate using a Perceptron?

- Use Multilayer Perceptron (MLP)
- Introduce another layer in between the input and output.
- This in-between layer is called hidden layer.

MLP for XOR gate

x_1 *xor* x_2

x_1	x_2	n_1	n_2	n_3
0	0	$0 \cdot 1 + 0 \cdot 1 = 0 \not\geq th$ $n_1 = 0$	$0 \cdot (-1) + 0 \cdot (-1) = 0 > th$ $n_2 = 1$	$0 \cdot 1 + 1 \cdot 1 = 1 \not\geq th$ $n_3 = 0$
0	1	$0 \cdot 1 + 1 \cdot 1 = 1 \geq th$ $n_1 = 1$	$0 \cdot (-1) + 1 \cdot (-1) = -1 \geq th$ $n_2 = 1$	$1 \cdot 1 + 1 \cdot 1 = 2 \geq th$ $n_3 = 1$
1	0	$1 \cdot 1 + 0 \cdot 1 = 1 \geq th$ $n_1 = 1$	$1 \cdot (-1) + 0 \cdot (-1) = -1 \geq th$ $n_2 = 1$	$1 \cdot 1 + 1 \cdot 1 = 2 \geq th$ $n_3 = 1$
1	1	$1 \cdot 1 + 1 \cdot 1 = 2 \geq th$ $n_1 = 1$	$1 \cdot (-1) + 1 \cdot (-1) = -2 \not\geq th$ $n_2 = 0$	$1 \cdot 1 + 0 \cdot 1 = 1 \not\geq th$ $n_3 = 0$

BITS-Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Solution of XOR Data

- Data
  - Truth table
- Model
  - Multi-layered Perceptron
- Challenge
  - How to learn the parameters and threshold?
- Solution for learning
  - Use gradient descent algorithm

Gradient Descent

To understand, consider simpler *linear unit*, where

$$\hat{y} = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - \hat{y}_d)^2$$

where D is set of training examples.
Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n} \right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}] \quad \text{i.e.,} \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradient Descent

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - \hat{y}_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - \hat{y}_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - \hat{y}_d)\frac{\partial}{\partial w_i}(t_d - \hat{y}_d) \\
&= \sum_d (t_d - \hat{y}_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - \hat{y}_d)(-x_{i,d})
\end{aligned}
$$

Gradient Descent Algorithm

Gradient-Descent(training examples $\langle \vec{x}, t \rangle$, learning rate η)

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output $\hat{y}$
    - For each linear unit weight w_i, Do

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - \hat{y}_d)(-x_{i,d})$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

Gradient Descent Algorithm

Incremental Gradient Descent Algorithm

- Do until satisfied
  - For each training example d in D
    - Compute the gradient $\nabla E_d[\vec{w}]$
    - $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_d[\vec{w}] \equiv \frac{1}{2}(t_d - \hat{y}_d)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough.

Minibatch Stochastic Gradient Descent (SGD)

- Apply Gradient descent algorithm on a random minibatch of examples every time we need to compute the update.
- In each iteration,
  1. Step 1: randomly sample a minibatch B consisting of a fixed number of training examples.
  2. Step 2: compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.
  3. Step 3: multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$$w \leftarrow w - \frac{\eta}{|B|} \sum_{i \in B} \partial_w loss^{(i)}(w, b) = w - \frac{\eta}{|B|} \sum_{i \in B} x^{(i)} \left(w^T x^{(i)} + b - y^{(i)} \right)$$

$$b \leftarrow b - \frac{\eta}{|B|} \sum_{i \in B} \partial_b loss^{(i)}(w, b) = b - \frac{\eta}{|B|} \sum_{i \in B} \left(w^T x^{(i)} + b - y^{(i)} \right)$$

Training SGD

- Initialize parameters (w, b)
- Repeat until done
  - compute gradient

$$g \leftarrow \partial_{(w,b)} \frac{1}{|B|} \sum_{i \in B} \left(w^T x^{(i)} + b - y^{(i)} \right)$$

  - update parameters

$$(w, b) \leftarrow (w, b) - \eta g$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyperparameters requires some adjustment by trial and error.

Numerical Example of SGD

- Equation is $y = (x+5)^2$.
- When will it be minimum?
- Use gradient descent algorithm .
- Assume starting point as 3 and Learning rate as 0.01.
- Equations:

$$\frac{dy}{dx} = \frac{d}{dx}((x+5)^2) = 2(x+5)$$

$$x \leftarrow x - \eta \cdot \frac{dy}{dx}$$

- Epoch 1:

$$x \leftarrow 3 - 0.01 * 2(3+5) = 2.84$$

- Epoch 2:

$$x \leftarrow 2.84 - 0.01 * 2(2.84+5) = 2.68$$

Further Reading

1. Chapter 1 of Dive into Deep Learning (T1)
2. Chapter 2 for Python Prelims, Linear Algebra, Calculus, Probability (T1)
3. Chapter 4 of Book: Machine Learning by Tom M. Mitchell

Thank You!

Deep Neural Networks
Module # 1 : MLP as Universal Approximators

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors
who made their course materials freely available online.

Table of Contents

1 Multi Layer Perceptron (MLP)

Power of MLP

MLP can be used for

- classification
  - binary
  - multiclass
- regression
  - real output
- representing complex decision boundaries
- representing boolean functions

MLP is called as Universal Approximator for the above reasons.

MLP for XOR gate

Perceptron with real inputs

- $x_1, \ldots, x_N$ are real valued.
- $W_1, \ldots, W_N$ are real valued.
- Unit "fires" if weighted input exceeds a threshold.

$$\hat{y} = \begin{cases} 1 & \text{if } \sum_i w_i x_i \geq T \\ 0 & \text{otherwise} \end{cases}$$

ani, D GC Act, 1956

Perceptron with real outputs

- $x_1, \ldots, x_N$ are real valued.
- $W_1, \ldots, W_N$ are real valued.
- The output can also be real valued. – Sometimes viewed as the ”probability” of firing.

$$y = sigmoid(\sum_i w_i x_i - b)$$

MLP for Complicated Decision Boundaries

- Build a network of units with a single output that fires if the input is in the coloured area.

Can now be composed into "networks" to compute arbitrary classification "boundaries"

MLP for Complicated Decision Boundaries

- The network must fire if the input is in the coloured area.

MLP for Complicated Decision Boundaries

- The network must fire if the input is in the coloured area.

MLP for Complicated Decision Boundaries

- The network must fire if the input is in the coloured area.

MLP for Complicated Decision Boundaries

- The network must fire if the input is in the coloured area.

MLP for Complicated Decision Boundaries

- The network must fire if the input is in the coloured area.

MLP for Complicated Decision Boundaries

- The network must fire if the input is in the coloured area.

MLP for Complicated Decision Boundaries

MLP for Complicated Decision Boundaries

- Network to fire if the input is in the yellow area.
  - ”OR” two polygons
  - A third layer is required

MLPs for Complex Decision Boundaries

- MLPs can compose arbitrarily complex decision boundaries with only one hidden layer.

MLPs for Arbitrary Classification Boundary

- Just fit in an arbitrary number of circles
  - More accurate approximation with greater number of smaller circles.
  - Can achieve arbitrary precision
- MLPs can capture any classification boundary.
- Deeper networks can require far fewer neurons.

MLP as Continuous-Valued Regression

- A simple 3-unit MLP with a ”summing” output unit can generate a ”square pulse” over an input.
  - Output is 1 only if the input lies between T1 and T2.
  - T1 and T2 can be arbitrarily specified

MLP as Continuous-Valued Regression

- A simple 3-unit MLP with a ”summing” output unit can generate a ”square pulse” over an input.
- An MLP with many units can model an arbitrary function over an input.
- This generalizes to functions of any number of inputs.

MLPs are Universal Boolean Functions

- MLPs can compute more complex Boolean functions.
- MLPs can compute any Boolean function; since they can emulate individual gates.

$$((A\&\bar{X}\&Z)|(A\&\bar{Y}))\&((X\ \&\ Y)|\overline{(X\&Z)})$$

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2X_3X_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + X_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2X_3\bar{X}_4X_5$$

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

How many layers for a Boolean MLP?

- Express any Boolean function in disjunctive normal form.

Truth table shows *all* input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

- Any truth table can be expressed in this manner.
- A one-hidden-layer MLP is a Universal Boolean Function.

Reducing a Boolean Function

This is a "Karnaugh Map"

It represents a truth table as a grid
Filled boxes represent input combinations for which output is 1; blank boxes have output 0

Adjacent boxes can be "grouped" to reduce the complexity of the DNF formula for the table

Reducing a Boolean Function

- Use K-map to reduce the DNF to find groups and express as reduced DNF.

$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

Reducing a Boolean Function

- Use K-map to reduce the DNF to find groups and express as reduced DNF.

$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

- *Reduced* DNF form:

Largest Irreducible DNF?

- What arrangement of ones and zeros simply cannot be reduced further?

- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function?

Width of a Single-Layer Boolean MLP

- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function of 6 variables?

Width of a Single-Layer Boolean MLP

- In general, XOR of N variables will require 2^{N-1} perceptrons in single hidden layer. **Exponential in N**

Width of a Single-Layer Boolean MLP

- In general, XOR of N variables will require 2^{N-1} perceptrons in single hidden layer.
- **How many units if we use multiple layers?**

Width of a Deep Boolean MLP

- An XOR needs 3 perceptrons.

$$O = W \oplus X \oplus Y \oplus Z$$

- This network will require 3x3 = 9 perceptrons.

Width of a Deep Boolean MLP

$$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$$

15 perceptrons

- This network will require 3x5 = 9 perceptrons.
- In general, the XOR of N variables will require $3(N-1)$ perceptrons.

Width of a Deep Boolean MLP

- Single hidden layer: Will require $2^{N-1} + 1$ perceptrons in all (including output unit). Exponential in N.
- Deep network: Will require $3(N-1)$ perceptrons in a deep network. Linear in N.
  This can be arranged in only $2log_2(N)$ layers.

Summary of MLP

- MLPs are connectionist computational models,
  - Individual perceptrons are computational equivalent of neurons.
  - The MLP is a layered composition of many perceptrons.
- MLP can model universal Boolean functions
  - Individual perceptrons can act as Boolean gates.
  - Networks of perceptrons are Boolean functions.
- MLPs are Boolean machines.
  - They represent Boolean functions over linear boundaries.
  - They can represent arbitrary decision boundaries.
  - They can be used to classify data.
- MLP can model continuous valued functions.
- MLPs as universal classifiers.
- MLPs as universal approximators.

Exercise

- **Note: Different Choices of weights and bias are possible.**
- Left hidden node implements x1=1 line
- Right hidden node implements x1=5 line
- 2nd node from left implements x2=x1+3 and 3rd from left implements x1+x2=9
- For + class, output of left hidden node = +1, for other nodes output = -1

Exercise

(x1, x2) are input features and target classes are either +1 or -1 as shown in the figure.

A. What is the minimum number of hidden layers and hidden nodes required to classify the following dataset with 100% accuracy using a fully connected multilayer perceptron network? Step activation functions are used at all nodes, i.e., output=+1 if total weighted input >= bias b at a node, else output = -1.

B. Show the minimal network architecture by organizing the nodes in each layer horizontally. Show the node representing x1 at the left on the input layer. Organize the hidden nodes in ascending order of bias at that node. Specify all weights and bias values at all nodes. Weights can be only -2.5, 2.5 or 0, and bias +ve/-ve multiples of 2.5.

Exercise - Solution

A. 2 hidden layers, 4 nodes in first hidden layer and 2 nodes in second hidden layer needed.

B.

Exercise

An XOR cannot be represented using a single perceptron. If this statement is true, demonstrate the alternative way to represent XOR using an ANN.

Exercise - Solution

An XOR cannot be represented using a single perceptron as the XOR has non-linear decision boundary. To represent non-linear decision boundary we need a multi-layer perceptron with one hidden layers.

$$a_k = sign(w_{k1}x_1 + w_{k2}x_2 + b)$$

x_1	x_2	a_1	a_2	y
0	0	-10 $\sim$ 0	20 $\sim$ 1	-10 $\sim$ 0
0	1	10 $\sim$ 1	5 $\sim$ 1	10 $\sim$ 1
1	0	10 $\sim$ 1	5 $\sim$ 1	10 $\sim$ 1
1	1	30 $\sim$ 1	-30 $\sim$ 0	-10 $\sim$ 0

Exercise

How many perceptrons are required to represent $W \oplus X \oplus Y \oplus Z$?

Exercise - Solution

An XOR cannot be represented using a single perceptron as the XOR has non-linear decision boundary. To represent non-linear decision boundary we need a multi-layer perceptron with one hidden layers.

- Traditional method:
  - Number of variables $n = 4$
  - Number of perceptrons = $2^n - 1 = 2^4 - 1 = 15$
- Alternate method:
  - Layer 1: $A = W \oplus X$
  - Layer 2: $B = Y \oplus Z$
  - Layer 3: $C = A \oplus B$
  - Number of perceptrons = $3(n-1) = 3 * (4-1) = 9$
  - Number of layers = Width = $2\log_2 n = 2\log_2 4 = 4$

Exercise - Solution

Further Reading

1. Dive into Deep Learning (T1)
2. http://mlsp.cs.cmu.edu/people/rsingh/docs/Chapter1_Introduction.pdf
3. http://mlsp.cs.cmu.edu/people/rsingh/docs/Chapter2_UniversalApproximators.pdf

Thank You!

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Deep Neural Network

Module # 2 : Deep Feedforward Neural Network

DNN Team, BITS Pilani

The author of this deck, Prof. Seetha Parameswaran,
is gratefully acknowledging the authors
who made their course materials freely available online.

Table of Contents

1. Single Neuron for Regression
2. Single Neuron for Binary Classification
3. Multi-class Classification
4. Deep FeedForward Neural Network
5. Activation Functions
6. Computational Graph
7. Training DNN

Linear Regression Example

- Suppose that we wish to estimate the prices of houses (in dollars) based on their area (in square feet) and age (in years).
- The linearity assumption just says that the target (price) can be expressed as a weighted sum of the features (area and age):

$y = m_1 x_1 + m_2 x_2 + \dots m_d x_d + b$

$$price = w_{area} * area + w_{age} * age + b$$

w_{area} and w_{age} are called weights, and b is called a bias.

d features.

- The weights determine the influence of each feature on our prediction and the bias just says what value the predicted price should take when all of the features take value 0.

Data

- The dataset comprises of training dataset and testing dataset. The split of DL is generally 99:1, as we are dealing with millions of examples.
- Each row is called an **example (or data point, data instance, sample)**.
- The thing we are trying to predict is called a **label (or target)**.
- The independent variables upon which the predictions are based are called **features (or covariates)**.

$$m - \text{ number of training examples}$$

$$i - i^{th} \text{ example}$$

$$x^{(i)} = [x_1^{(1)}, x_1^{(2)}, \ldots x_1^{(m)}] - \text{ features of } i^{th} \text{ example}$$

$$y^{(i)} = [y_1^{(1)}, y_1^{(2)}, \ldots y_1^{(m)}] - \text{ label of } i^{th} \text{ example}$$

Affine transformations and Linear Models

- The equation of the form

$$\hat{y} = w_1 x_1 + \ldots w_d x_d + b$$

$$\hat{\mathbf{y}} = \mathbf{w}^\top \mathbf{x} + b \qquad \mathbf{x} \in \mathcal{R}^d \quad \mathbf{w} \in \mathcal{R}^d$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b \qquad \mathbf{X} \in \mathcal{R}^{m \times d} \quad \hat{\mathbf{y}} \in \mathcal{R}^m$$

linear

real valued.

is an **affine transformation** of input features, which is characterized by a linear transformation of features via weighted sum, combined with a translation via the added bias.

- Models whose output prediction is determined by the affine transformation of input features are **linear models**.
- The affine transformation is specified by the chosen weights (w) and bias (b).

Loss Function

- Loss function is a quality measure for some given model or a measure of fitness.
- The loss function quantifies the distance between the real and predicted value of the target.
- The loss will usually be a non-negative number where smaller values are better and perfect predictions incur a loss of 0.
- The most popular loss function in regression problems is the **squared error**.

Squared Error Loss Function

- The most popular loss function in regression problems is the squared error.
- For each example,

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$= J$

- For the entire dataset of m examples, average (or equivalently, sum) the losses

Cost

Total loss

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{m}\sum_{i=1}^{m} l^{(i)}(\mathbf{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

- When training the model, find parameters $(\mathbf{w}_{opt}, b_{opt})$ that minimize the total loss across all training examples.

$$\boxed{(\mathbf{w}_{opt}, b_{opt}) = arg\min_{\mathbf{w},b} \mathcal{L}(\mathbf{w}, b)}$$

Minibatch Stochastic Gradient Descent (SGD)

$\eta = 0.01$

- Apply Gradient descent algorithm on a random minibatch of examples every time we need to compute the update.
- In each iteration,
  - Step 1: randomly sample a minibatch B consisting of a fixed number of training examples.
  - Step 2: compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.
  - Step 3: multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$|B|$

10 million

10^7

$B = 2^{\ }$

$= 1024$

1000 batches

$$\theta \leftarrow \theta - \frac{\eta \cdot g}{1024}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|B|} \sum_{i \in B} \partial_w loss^{(i)}(\mathbf{w}, b)$$

$$b \leftarrow b - \frac{\eta}{|B|} \sum_{i \in B} \partial_b loss^{(i)}(\mathbf{w}, b)$$

$$\frac{\partial \text{error}}{\partial \theta}$$

Training using SGD Algorithm

- Initialize parameters $(\mathbf{w}, b)$
- Repeat until done
  - Compute gradient

$$\mathbf{g} \leftarrow \partial_{(\mathbf{w},b)} \frac{1}{|B|} \sum_{i \in B} loss(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

  - Update parameters

$$\boxed{(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}}$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyper-parameters requires some adjustment by trial and error.

Prediction

$\hat{y} \rightarrow$ vector

1. FP $\rightarrow$ $\hat{y}^{(i)} = w^T x^{(i)} + b$

One value

2. Cost $\rightarrow$ $L(w,b) = \frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - \hat{y}^{(i)}\right)^2$

- Estimating targets given features is commonly called **prediction or inference**.
- Given the learned model, values of target can be predicted, for any set of features.

3. $\theta \leftarrow \theta - \eta \frac{\partial L(w,b)}{\partial \theta}$

vector

One layer

regression

Table of Contents

1. Single Neuron for Regression
2. Single Neuron for Binary Classification
3. Multi-class Classification
4. Deep FeedForward Neural Network
5. Activation Functions
6. Computational Graph
7. Training DNN

Binary Classification – Training Examples

$$\text{Single training example} = \{(x, y)\} \qquad \text{where } x \in \mathcal{R}^d \text{ and } y \in \{0, 1\}$$

$$\text{\# training examples} = m$$

$$m \text{ training examples} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \ldots (x^{(m)}, y^{(m)})\}$$

y discrete
0 or 1

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \ldots & x^{(m)} \\ | & | & & | \end{bmatrix} \qquad \text{where } \mathbf{X} \in \mathcal{R}^{d \times m}$$

$$\mathbf{Y} = \begin{bmatrix} y^{(1)} & y^{(2)} & \ldots & y^{(m)} \end{bmatrix} \qquad \text{where } \mathbf{Y} \in \mathcal{R}^{1 \times m}$$

Forward Propagation

(1) FP (a) $z = W^T x + b$

$a = \sigma(z)$

Given X find $\hat{y} = P(y = 1 \mid X)$

Input $X \in \mathcal{R}^{d \times m}$

Parameters $w \in \mathcal{R}^d$

$b \in \mathcal{R}$

Activation $z = w^\top x + b$

$$\boxed{\text{Activation function } \sigma(z) = \frac{1}{1 + e^{-z}}}$$

$$= \begin{cases} 1 \text{ if } z \text{ is large positive} \\ 0 \text{ if } z \text{ is large negative} \end{cases}$$

real 0 to 1.

Output $\hat{y} = \sigma(w^\top x + b)$ $\quad 0 \leq \hat{y} \leq 1$

log 0 = large value

$\hat{y}(0,1)$

Cost Function for Binary Classification

loss

$$\boxed{\text{Loss function } loss(\hat{y}, y) = -\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right]}$$

$0 + 1 \cdot \log 1 = 0$

If $y = 1 \quad loss(\hat{y}, y) = -\log \hat{y}$

$loss \approx 0 \implies \log \hat{y} \approx \text{large} \implies \hat{y} \approx \text{large}$

$1 \log 1 + 0 = 0$

If $y = 0 \quad loss(\hat{y}, y) = -\log(1-\hat{y})$

$loss \approx 0 \implies \log(1-\hat{y}) \approx \text{small} \implies \hat{y} \approx \text{small}$

$0 + 1 \log 0 = \text{large}$

$1 \log 0 + 0 = \text{large}$

$y \to 0, 1$

y	$\hat{y}$
0	0
1	1
0	1
1	0

correct

$$\text{Cost function } \mathcal{L}(w, b) = \frac{1}{m}\sum_{i=1}^{m} loss(\hat{y}^{(i)}, y^{(i)})$$

$$= \frac{1}{m}\sum_{i=1}^{m} -\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

How to Learn Parameters?

To Learn parameters use Gradient Descent Algorithm

$$\hat{y} = \sigma(w^\top x + b) \qquad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} - \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

x_1

x_d

Find $\mathbf{w}_{opt}, b_{opt}$ that minimize $\mathcal{L}(\mathbf{w}, b)$.

Gradient Descent

$\frac{\partial(w^T x + b)}{\partial w}$

$\hat{y}$

Find $\mathbf{w}_{opt}, b_{opt}$ that minimize $\mathcal{L}(\mathbf{w}, b)$

Repeat {

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial w}$$

$$b \leftarrow b - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial b}$$

}

$\theta \leftarrow \theta - \eta g$

Training using SGD Algorithm

- Initialize parameters $(\mathbf{w}, b)$
- Repeat until done
  - Compute gradient

$$\mathbf{g} \leftarrow \partial_{(\mathbf{w},b)} \frac{1}{|B|} \sum_{i \in B} loss(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

  - Update parameters

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$$

PS: The number of epochs and the learning rate are both hyperparameters. Setting hyper-parameters requires some adjustment by trial and error.

Exercise - MSE Loss

$loss = (y - \hat{y})^2$

$\frac{\partial loss}{\partial w} = +2(y - \hat{y}) * (-1)$

Consider the neural network with two inputs x_1 and x_2 and the initial weights are $w_0 = 0.5, w_1 = 0.8, w_2 = 0.3$. Draw the network, compute the output, mean squared loss function and weight updation when the input is $(1, 0)$, the learning rate is 0.01 and target output is 1. Assume any other relevant information.

$\Sigma = w_0 + w_1 x_1 + w_2 x_2$

$= 1.7$

$a = \sigma(1.7)$

$= 0.79$

$\hat{y} = \sigma(1.3) = 0.79$

$g = \frac{\partial loss}{\partial \theta}$

$= -2(y - \hat{y})$

$MSE = \frac{1}{1}\Sigma(y - \hat{y})^2 = (1 - 0.79)^2 =$

$w \leftarrow w - \eta g$

Exercise - MSE Loss Solution

$$\hat{y} = \sigma(w_0 + w_1 x_1 + w_2 x_2) = \sigma(0.5 + 0.8 * 1 + 0.3 * 0) = \sigma(1.3) = 0.7858$$

$$MSE = \frac{1}{2m}\sum_{i=1}^{m}(y - \hat{y})^2 = \frac{1}{2}(1 - 0.7858)^2 = \frac{0.04588}{2} = 0.02294$$

$$\mathbf{g} = \frac{2}{2m}\sum_{i=1}^{m}(y - \hat{y}) = (1 - 0.7858) = 0.2142$$

$$new\ w \leftarrow old\ w - \eta * \mathbf{g}$$

$$w_0 = 0.5 - 0.01 * 0.2142 = 0.4978$$

$$w_1 = 0.8 - 0.01 * 0.2142 = 0.7978$$

$$w_2 = 0.3 - 0.01 * 0.2142 = 0.2978$$

Exercise - BCE

Consider the neural network with two inputs x_1 and x_2 and the initial weights are $w_0 = 0.5, w_1 = 0.8, w_2 = 0.3$. Draw the network, compute the output, binary cross entropy loss function and weight updation when the input is $(1, 0)$, the learning rate is 0.01 and target output is 1. Assume any other relevant information.

Exercise - BCE Solution

$$\hat{y} = \sigma(w_0 + w_1 x_1 + w_2 x_2) = \sigma(0.5 + 0.8 * 1 + 0.3 * 0) = \sigma(1.3) = 0.7858$$

$$Loss = \frac{1}{m}\sum_{i=1}^{m} - \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right]$$

$$= -[1 \ln 0.7858 + (1 - 1) \ln(1 - 0.7858)] = 0.24$$

$$\mathbf{g} = \sum_{i=1}^{m}(y - \hat{y})z = (1 - 0.7858) * 1.3 = 0.278$$

$$w \leftarrow w - \eta * \mathbf{g}$$

$$w_0 = 0.5 - 0.01 * 0.278 = 0.497$$

$$w_1 = 0.8 - 0.01 * 0.278 = 0.797$$

$$w_2 = 0.3 - 0.01 * 0.278 = 0.297$$

Single-Layer Neural Network

- ~~Linear regression~~ is a single-layer neural network.
  - Number of inputs (or feature dimensionality) in the input layer is d. The inputs are $x_1, \ldots, x_d$.
  - Number of outputs in the output layer is 1. The output is $\hat{y}$.
  - Number of layers for the neural network is 1. (conventionally we do not consider the input layer when counting layers.)
  - Every input is connected to every output, This transformation is a **fully-connected layer or dense layer**.

Table of Contents

1. Single Neuron for Regression
2. Single Neuron for Binary Classification
3. Multi-class Classification
4. Deep FeedForward Neural Network
5. Activation Functions
6. Computational Graph
7. Training DNN

$$B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} \qquad W = \begin{bmatrix} 3 \times 4 \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}$$

$$Z = WX + B$$

$$\hat{Y} = \text{softmax}(Z)$$

Multi-class Classification Example

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

- Each input consists of a 2×2 grayscale image.
- Represent each pixel value with a single scalar, giving four features x_1, x_2, x_3, x_4.
- Assume that each image belongs to one among the categories "square", "triangle", and "circle".
- How to represent the labels?
  - Use label encoding. 0, 1, 2
    $y \in 1, 2, 3$, where the integers represent *circle*, *square*, *triangle* respectively.
  - Use one-hot encoding.
    $y \in (1, 0, 0), (0, 1, 0), (0, 0, 1)$. y would be a three-dimensional vector, with (1, 0, 0) corresponding to "circle', (0, 1, 0) to "square", and (0, 0, 1) to "triangle".

1 0 0
0 1 0
0 0 1

Single-Layer Neural Network

$\langle X, Y \rangle$

- A model with multiple outputs, one per class. Each output will correspond to its own affine function.
- 4 features and 3 possible output categories
- Every input is connected to every output, This transformation is a **fully-connected layer or dense layer.**

$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

$P(\hat{y}_1 | x)$ o/p

$P(\hat{y}_2 | x)$

$P(\hat{y}_3 | x)$ i/p

class 1

$P(\hat{y}_1 = 1 | x)$

$z_1 = w_{11} x_1 + w_{21} x_2 + w_{31} x_3 + w_{41} x_4 + w_{01}$

softmax

$\hat{y}_1 = f(z_1)$

Architecture

$P(y_i = 1 \mid x)$

$z = \begin{bmatrix} 1 \\ 0.6 \\ 0.4 \end{bmatrix}$ argmax $\hat{y} = \begin{bmatrix} 0.45 \\ 0.30 \\ 0.25 \end{bmatrix}$

0 to 1.

- 12 scalars to represent the weights and 3 scalars to represent the biases .
- Compute three logits, $\hat{y}_1$, $\hat{y}_2$, and $\hat{y}_3$, for each input.
- Weights is a 3×4 matrix and bias is 1×3 matrix.

assume bias as 0

$|w| = 4 \times 3 = 12$

$|b| = 3$

$\hat{y}_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$ norm

$e^{z_1} = 2.72$

$e^{z_2} = 1.82$

$e^{z_3} = 1.5$

denom = 6.03

$$z_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1 = 1 \qquad \hat{y}_1 = softmax(z_1)$$

$$z_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2 = 0.6 \qquad \hat{y}_2 = softmax(z_2)$$

$$z_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3 = 0.4 \qquad \hat{y}_3 = softmax(z_3)$$

0.1 0.2 0.3 0.4

0.3 -0.2 0.4 0.1

0.4 -0.3 0.1 -0.2

p1 p2 p3

Softmax Operation

- Interpret the outputs of our model as probabilities.
- Any output $\hat{y}_j$ is interpreted as the probability that a given item belongs to class j. Then choose the class with the largest output value as our prediction $argmax_j \hat{y}_j$.
- If $\hat{y}_1$, $\hat{y}_2$, and $\hat{y}_3$ are $0.1, 0.8$, and 0.1, respectively, then predict category 2.
- The softmax function transforms the outputs such that they become non-negative and sum to 1, while requiring that the model remains differentiable.

$$\hat{y} = softmax(Z) \quad where \quad \hat{y}_j = \frac{exp(z_j)}{\sum_k exp(z_k)}$$

- First exponentiate each logit (ensuring non-negativity) and then divide by their sum (ensuring that they sum to 1).
- Softmax is a non-linear function.

Log-Likelihood Loss / Cross-Entropy Loss

X → class 2

- The softmax function gives a vector $\hat{y}$, which can be interpreted as estimated conditional probabilities of each class given any input x,

$$\hat{y}_j = P(y = class_j \mid x)$$

argmax $\hat{y}$ = 0.45, 0.30, 0.25

- Compare the estimates with reality by checking how probable the actual classes are according to our model, given the features:

$$P(Y \mid X) = \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)})$$

→ cond prob

$P(\hat{y}_1 \mid X)$ $P(\hat{y}_2 \mid X)$ $P(\hat{y}_3 \mid X)$

$P(Y \mid X)$

class 2 $[0 \ 1 \ 0]^T$

Log-Likelihood Loss / Cross-Entropy Loss

$P(y \mid x) = \dfrac{e^{z_i}}{\sum e^{z_i}}$

- Maximize $P(Y \mid X)$ = Minimize the negative log-likelihood

m classes

$$-\log P(Y \mid X) = \sum_{i=1}^{m} -\log \underbrace{P(y^{(i)} \mid x^{(i)})}_{\text{conditional}} = \sum_{i=1}^{m} loss\, \boxed{P(y^{(i)}, \hat{y}^{(i)})}$$

joint

$$loss\, P(y^{(i)}, \hat{y}^{(i)}) = -\sum_{j=1}^{m} y_j \log \hat{y}_j$$

$$loss = -\left(\overset{0}{y_1} \log \overset{0.45}{\hat{y}_1} + \overset{1}{y_2} \log \overset{0.3}{\hat{y}_2} + \overset{0}{y_3} \log \overset{0.25}{\hat{y}_3} \right)$$

Softmax Example

- z_1 = Un-normalized log probabilities
- e^{z_1} = Un-normalized probabilities
- $\hat{y}_i = softmax(z_1)$ = normalized probabilities
- L_i = loss = $y_i \log \hat{y}_i$

class	z_1	e^{z_1}	$softmax(z_1)$	L_1
cat	3.2	24.5	0.13	0.89
car	5.1	164.0	0.87	0.06
dog	-1.7	0.18	0.00	∞

Table of Contents

1. Single Neuron for Regression
2. Single Neuron for Binary Classification
3. Multi-class Classification
4. Deep FeedForward Neural Network
5. Activation Functions
6. Computational Graph
7. Training DNN

Deep FeedForward Neural Network

- A neural network with more number of hidden layers is considered as a **deep neural network**. There are no feedback connections.
- Convolutional networks are examples of feedforward neural networks.

Figure: Shallow Neural Network

Figure: Deep Neural Network

Deep FeedForward Neural Network (DNN)

- With deep neural networks, use the data to jointly learn both a representation via hidden layers and a linear predictor that acts upon that representation.
- Add many hidden layers by stacking many fully-connected layers on top of each other. Each layer feeds into the layer above it, until we generate outputs.
- The first $(L-1)$ layers learns the representation and the final layer is the linear predictor.

DNN Architecture

- DNN has 4 inputs, 3 outputs, and its hidden layer contains 5 hidden units.
- Number of layers in this DNN is 2.

DNN Architecture

- The layers are fully connected.
- Every input influences every neuron in the hidden layer.
- Each of the hidden neurons in turn influences every neuron in the output layer.
- The outputs of the hidden layer are called as **hidden representations** or hidden-layer variable or a hidden variable.

DNN Architecture

Hidden layers – intermediate layers, desired output for each of these layers are not shown.

Depth – number of layers.

Width – dimensionality of the hidden layers

DNN – General Strategy

- Design a DNN architecture of the network. Architecture depends on the problem / application / complexity of the decision boundary.
- Choose the activation functions that will be used to compute the hidden layer activations. The activation function is the same for all neurons in a layer. Activation function can change between layers.
- Choose the cost function. It depends on whether regression, binary classification or multi-class classification.
- Choose the optimizer algorithm, depends on the complexity and variance of the input data.
- Train the feedforward network. Learning in deep neural networks requires computing the gradients using the back-propagation algorithm.
- Evaluate the performance of the network.

Non-linearity in DNN

- Input is $X \in \mathcal{R}^{m \times d}$ with m examples where each example has d features.
- Hidden layer has h hidden units $H \in \mathcal{R}^{m \times h}$.
- Hidden layer weights $W^{(1)} \in \mathcal{R}^{d \times h}$ and biases $b^{(1)} \in \mathcal{R}^{1 \times h}$
- Output layer weights $W^{(2)} \in \mathcal{R}^{d \times q}$ and biases $b^{(2)} \in \mathcal{R}^{1 \times q}$
- Output is $\hat{Y} \in \mathcal{R}^{m \times q}$
- A non-linear activation function σ has to be applied to each hidden unit following the affine transformation. The outputs of activation functions are called **activations**.

$$H = \sigma(XW^{(1)} + b^{(1)})$$
$$\hat{Y} = softmax(XW^{(2)} + b^{(2)})$$

Table of Contents

1. Single Neuron for Regression
2. Single Neuron for Binary Classification
3. Multi-class Classification
4. Deep FeedForward Neural Network
5. Activation Functions
6. Computational Graph
7. Training DNN

Activation Functions

- Activation function of a neuron defines the output of that neuron given an input or set of inputs.
- Introduces non-linearity to a neuron.
- A non-activated neuron will act as a linear regression with limited learning.
- Multilayered deep neural networks learn meaningful features from data.
- Artificial neural networks are designed as universal function approximators, they must have the ability to calculate and learn any non-linear function.

Sigmoid (Logistic) Activation Function

$$\text{Function: } \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\text{Derivative: } \sigma'(z) = \sigma(z)(1-\sigma(z))$$

$$\text{Range } : (0,1)$$

- Non-Linear function
- Small changes in z will be large in $f(z)$. This means it is a good classifier.
- Leads to vanishing gradient. So the learning is minimal.

Tanh Activation Function

$$\text{Function: } tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})}$$

$$\text{Derivative: } tanh'(z) = (1 - tanh^2(z))$$

$$\text{Range } : (-1, 1)$$

Tanh Activation Functions

- More efficient because it has a wider range for faster learning and grading.
- The tanh activation usually works better than sigmoid activation function for hidden units because the mean of its output is closer to zero, and so it centers the data better for the next layer.
- Issues with tanh
  - computationally expensive
  - lead to vanishing gradients
    If we initialize the weights to relative large values, this will cause the inputs of the tanh to also be very large, thus causing gradients to be close to zero. The optimization algorithm will thus become slow.

ReLU Activation Function

Rectified Linear Unit

Function: $g(z) = \max(0, z)$

Derivative: $g'(z) = \begin{cases} 0 & \text{for } z \leq 0 \\ 1 & \text{for } z > 0 \end{cases}$

Range : $[0, \infty]$

ReLU Activation Functions

- ReLU activation function is not differentiable at origin.
- If we combine two ReLU units, we can recover a piece-wise linear approximation of the Sigmoid function.
- ReLU activation can lead to exploding gradient.
- In zero value region, learning is not happening.
- Fast Learning
- Fewer vanishing gradient problems
- Sparse activation
- Efficient computation
- Scale invariant (max operation)
- Non Zero centered
- Non differentiable at Zero

Leaky ReLU Activation Function

Function: $g(z) = \max(0.01z, z)$

Derivative: $g'(z) = \begin{cases} 0.01 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$

Range : $[-\infty, \infty]$

Parametric ReLU (PReLU) Activation Function

$$\text{Function: } g(z) = \max(az, z)$$

$$\text{Derivative: } g'(z) = \begin{cases} a & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$$

$$\text{Range } : [-\infty, \infty]$$

ELU (Exponential Linear Units)

$$\text{Function: } g(z) = \begin{cases} z & \text{for } z < 0 \\ a(e^z - 1) & \text{for } z \geq 0 \end{cases}$$

$$\text{Range } : [-\infty, \infty]$$

Comparing Activation Functions

Activation Function	Sigmoid	Tanh	ReLU
Linearity	Non Linear	Non Linear	Non Linear
Activation Function	$\sigma(x) = \frac{1}{1+e^{-z}}$	$tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})}$	$g(x) = \max(0, z)$
Derivative	$\sigma(z)(1 - \sigma(z))$	$(1 - tanh^2(z))$	$1\ if\ z > 0\ else\ 0$
Symmetric Function	No	Yes	No
Range	$[0, 1]$	$[-1, 1]$	$[0, \infty]$
Vanishing Gradient	Yes	Yes	No
Exploding Gradient	No	No	Yes
Zero Centered	No	Yes	No

Comparing Activation Functions

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		f(x) = x	f'(x) = 1	(-∞, ∞)
Binary Step		f(x) = $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	{0, 1}
Sigmoid		f(x) = $\sigma(x) = \frac{1}{1+e^{-x}}$	f'(x) = $f(x)(1-f(x))$	(0, 1)
Hyperbolic Tangent(tanh)		f(x) = $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	f'(x) = $1 - f(x)^2$	(-1, 1)
Rectified Linear Unit(ReLU)		f(x) = $\begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$	f'(x) = $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	[0, ∞)
Softplus		f(x) = $\ln(1+e^x)$	f'(x) = $\frac{1}{1+e^{-x}}$	(0, 1)
Leaky ReLU		f(x) = $\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	(-1, 1)
Exponential Linear Unit(ELU)		f(x) = $\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	f'(x)=$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	[0, ∞)

Activation Functions Plots

Table of Contents

1. Single Neuron for Regression
2. Single Neuron for Binary Classification
3. Multi-class Classification
4. Deep FeedForward Neural Network
5. Activation Functions
6. Computational Graph
7. Training DNN

Computation Graphs

- Computations of a neural network are organized in terms of
  - a forward pass or a forward propagation step, in which we compute the output of the neural network,
  - followed by a backward pass or back propagation step, which we use to compute gradients or derivatives.
- The computation graph organizes a computation with this blue arrow, left-to-right computation.
- The backward red arrow, right-to-left shows computation of the derivatives.

Computation Graphs

- Each node in the graph indicate a variable.
- An operation is a simple function of one or more variable.
- An operation is defined such that it returns only a single output variable.
- If a variable y is computed by applying an operation to a variable x, then we draw a directed edge from x to y.

Computation Graphs Example

Forward Pass:

$$\text{Let } J(a, b, c) = 3(a + bc)$$
$$u = bc$$
$$v = a + u$$
$$\text{Then } J = 3v$$

Computation Graphs Example

Backward Pass / Back propagation of gradients:

$$\text{Let } J(a,b,c) = 3(a+bc)$$
$$u = bc$$
$$v = a+u$$
$$\text{Then } J = 3v$$

Computation Graph – Binary Classification

$$z = w^\top X + b$$
$$a = \hat{y} = \sigma(z)$$
$$\mathcal{L}(a, y) = -\left[y \log a + (1 - y) \log(1 - a)\right]$$

Computation Graph – Binary Classification

$$z = w^\top X + b$$
$$a = \hat{y} = \sigma(z)$$
$$\mathcal{L}(a, y) = -\left[y \log a + (1 - y) \log(1 - a)\right]$$
$$da = \frac{d\mathcal{L}(a, y)}{da} = \frac{-y}{a} + \frac{1 + y}{1 + a}$$

$$dz = \frac{d\mathcal{L}(a, y)}{dz} = a - y$$
$$dw_1 = \frac{d\mathcal{L}(a, y)}{dw_1} = x_1 dz$$
$$db = dz$$

Exercise

Draw the computational graph for the equation

$$f(x, y, z) = (x + y)z$$

Assume that $x = -2, y = 5$ and $z = -4$. Using the computation graph show the computation of the gradients also.

Exercise - Solution

Exercise

Draw the computational graph for the equation

$$f(u, V) = || \ V \cdot u \ ||^2 = \sum_{i=1}^{n} (V \cdot u)_i^2$$

Using the computation graph show the computation of the gradients also. Assume that

$$V = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

Exercise - Solution

Exercise - Solution

$$V = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

$$q = V \cdot u = \begin{bmatrix} 0.1 * 0.2 + 0.5 * 0.4 \\ -0.3 * 0.2 + 0.8 * 0.4 \end{bmatrix} = \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

$$f = \sum_{i=1} q_i^2 = 0.22^2 + 0.26^2 = 0.116$$

Exercise - Solution

$$\nabla q = \nabla_q f = 2q = 2\begin{bmatrix}0.22\\0.26\end{bmatrix} = \begin{bmatrix}0.44\\0.52\end{bmatrix}$$

$$\nabla V = \nabla_V f = 2q \cdot u^\top = 2\begin{bmatrix}0.22\\0.26\end{bmatrix}\begin{bmatrix}0.2 & 0.4\end{bmatrix} = \begin{bmatrix}0.088 & 0.176\\0.105 & 0.208\end{bmatrix}$$

$$\nabla u = \nabla_u f = 2V^\top \cdot q = 2\begin{bmatrix}0.1 & -0.3\\0.5 & 0.8\end{bmatrix}\begin{bmatrix}0.22\\0.26\end{bmatrix} = \begin{bmatrix}-0.112\\0.636\end{bmatrix}$$

Table of Contents

1. Single Neuron for Regression
2. Single Neuron for Binary Classification
3. Multi-class Classification
4. Deep FeedForward Neural Network
5. Activation Functions
6. Computational Graph
7. Training DNN

Two Layer Neural Network Architecture

Computing the Activations

For Hidden layer

$$z_1^{[1]} = w_1^{[1]} a^{[0]} + b^{[1]}$$
$$a_1^{[1]} = \sigma\left(z_1^{[1]}\right)$$
$$z_2^{[1]} = w_2^{[1]} a^{[0]} + b^{[1]}$$
$$a_2^{[1]} = \sigma\left(z_2^{[1]}\right)$$
$$z_3^{[1]} = w_3^{[1]} a^{[0]} + b^{[1]}$$
$$a_3^{[1]} = \sigma\left(z_3^{[1]}\right)$$
$$z_4^{[1]} = w_4^{[1]} a^{[0]} + b^{[1]}$$
$$a_4^{[1]} = \sigma\left(z_4^{[1]}\right)$$

In the matrix form

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \underbrace{\begin{bmatrix} \dots & w_1^{[1]T} & \dots \\ \dots & w_2^{[1]T} & \dots \\ \dots & w_3^{[1]T} & \dots \\ \dots & w_4^{[1]T} & \dots \end{bmatrix}}_{4 \times 3} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{3 \times 1} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{4 \times 1}$$

$$z^{[1]} = \begin{bmatrix} w_1^{[1]\top} a^{[0]} + b_1^{[1]} \\ w_2^{[1]\top} a^{[0]} + b_2^{[1]} \\ w_3^{[1]\top} a^{[0]} + b_3^{[1]} \\ w_4^{[1]\top} a^{[0]} + b_4^{[1]} \end{bmatrix} \leftarrow 4 \times 1 \text{ matrix}$$

$$a^{[1]} = \sigma\left(z^{[1]}\right) \leftarrow 4 \times 1 \text{ matrix}$$

Activations for Training Examples

Vectorizing for one training example

$$a^{[0]} = X$$
$$Z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$
$$a^{[1]} = \sigma\left(Z^{[1]}\right)$$
$$Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma\left(Z^{[2]}\right)$$
$$\hat{y} = a^{[2]}$$

Vectorizing for all training examples

$$\text{Example 1} : X^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$
$$\text{Example 2} : X^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$$
$$\text{Example 3} : X^{(2)} \longrightarrow a^{[2](3)} = \hat{y}^{(3)}$$
$$\vdots \qquad\qquad \vdots$$
$$\text{Example m} : X^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$$

Neural Network Training - Forward Pass

Step 1: Compute Loss on mini-batch [Forward pass]

Neural Network Training - Forward Pass

Step 1: Compute Loss on mini-batch [Forward pass]

Neural Network Training - Forward Pass

Step 1: Compute Loss on mini-batch [Forward pass]

Forward Propagation Algorithm

Algorithm 1: Forward Propagation

Initialize the weights and bias randomly.

$a^{[0]} = X^{(i)}$

$Z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = \sigma\left(Z^{[1]}\right)$

$Z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = \sigma\left(Z^{[2]}\right)$

$$\hat{y} = \begin{cases} 1 & \text{if} \quad a^{[2]} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Note: All training examples are considered in the vectorized form.

Example Neural Network

Computation Graph for Forward Pass

Cost Function

- The difference between the actual observation y and the computed activation $\hat{y}$ gives the error or the cost function.
- For binary classification,

$$\mathcal{L} = \frac{1}{m}\sum_{i=0}^{m} loss\left(\hat{y}^{(i)}, y^{(i)}\right)$$

$$\mathcal{L} = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - a^{[2](i)}\right)\right)$$

Computation Graph for Cost Function

Neural Network Training - Backward Pass

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]

Neural Network Training - Backward Pass

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]

Neural Network Training - Backward Pass

Step 1: Compute Loss on mini-batch [Forward pass]

Step 2: Compute gradients wrt parameters [Backward pass]

Back Propagation of Cost Function

Weights	$W^{[1]} + b^{[1]}$	$W^{[2]} + b^{[2]}$
Gradients	$da^{[1]} + dz^{[1]}$	$da^{[2]} + dz^{[2]}$

Computing the Gradients

For one training example

$$dz^{[2]} = a^{[2]} - y$$
$$dw^{[2]} = dz^{[2]} \cdot a^{[1]T}$$
$$db^{[2]} = dz^{[2]}$$
$$dz^{[1]} = \left(w^{[2]T} dz^{[2]}\right) * \sigma'\left(z^{[1]}\right)$$
$$dw^{[1]} = dz^{[1]} \cdot x^{\top}$$
$$db^{[1]} = dz1$$

For all training examples

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m}\left(dZ^{[2]} \cdot A^{[1]T}\right)$$
$$db^{[2]} = \frac{1}{m}\sum dZ^{[2]}$$
$$dZ^{[1]} = \left(W^{[2]T} dZ^{[2]}\right) * \sigma'\left(Z^{[1]}\right)$$
$$dW^{[1]} = \frac{1}{m}\left(dZ^{[1]} \cdot X^{\top}\right)$$
$$db^{[1]} = \frac{1}{m}\sum dZ1$$

Backward Propagation Algorithm

Algorithm 2: Backward Propagation

$dZ^{[2]} = A^{[2]} - Y$

$dW^{[2]} = \frac{1}{m}\left(dZ^{[2]} \cdot A^{[1]T}\right)$

$db^{[2]} = \frac{1}{m}\sum dZ^{[2]}$

$dZ^{[1]} = \left(W^{[2]T} dZ^{[2]}\right) * \sigma'\left(Z^{[1]}\right)$

$dW^{[1]} = \frac{1}{m}\left(dZ^{[1]} \cdot X^{\top}\right)$

$db^{[1]} = \frac{1}{m}\sum dZ^{[1]}$

Computation Graph for Backward Pass

Neural Network Training - Backward Pass

Step 1: Compute Loss on mini-batch [Forward pass]
Step 2: Compute gradients wrt parameters [Backward pass]
Step 3: Use gradients to update parameters

$$\theta \leftarrow \theta - \eta \frac{d\mathcal{L}}{d\theta}$$

Update the Weights and Bias

Algorithm 3: Weight Updation

$W^{[1]} = W^{[1]} - \text{learning rate} * dW^{[1]}$

$b^{[1]} = b^{[1]} - \text{learning rate} * db^{[1]}$

$W^{[2]} = W^{[2]} - \text{learning rate} * dW^{[2]}$

$b^{[2]} = b^{[2]} - \text{learning rate} * db^{[2]}$

DNN Architecture

Training DNN

Requires

1. **Forward Pass** through each layer to compute the output.
2. Compute the deviation or error between the desired output and computed output in the forward pass (first step). This morphs into **objective function**, as we want to minimize this deviation or error.
3. The deviation has to be send back through each layer to compute the delta or change in the parameter values. This is achieved using **back propagation algorithm**.
4. **Update** the parameters.

DNN Architecture

Notations

- L : Number of layers
- $n^{[l]}$: Number of units in layer l
- $w^{[l]}$: Weights for layer l
- $b^{[l]}$: Bias for layer l
- $z^{[l]}$: Hypothesis for layer l
- $g^{[l]}$: Activation Function used for layer l
- $a^{[l]}$: Activation for layer l

Matrix Dimensions

- $W^{[l]}$: $n^{[l]} \times n^{[l-1]}$
- $b^{[l]}$: $n^{[l]} \times 1$
- $z^{[l]}$: $n^{[l]} \times 1$
- $a^{[l]}$: $n^{[l]} \times 1$
- $dW^{[l]}$: $n^{[l]} \times n^{[l-1]}$
- $db^{[l]}$: $n^{[l]} \times 1$
- $dz^{[l]}$: $n^{[l]} \times 1$
- $da^{[l]}$: $n^{[l]} \times 1$

Forward Propagation

- The inputs **X** provide the initial information that then propagates up to the hidden units at each layer and finally produces $\hat{y}$. This is called forward propagation.
- Information flows forward through the network.
- During training, it produces a scalar cost $\mathcal{L}(\theta)$.

Forward Propagation Algorithm

Algorithm 4: Forward Propagation

for l *in range* $(1, L)$ **do**

  $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$

  $A^{[l]} = g^{[l]}\left(Z^{[l]}\right)$

$\hat{y} = A^{[L]}$

$J = L(\hat{y}, y) + \lambda \Omega(\theta)$

Backward Propagation

- Back-propagation algorithm or backprop, allows the information from the cost to then flow backwards through the network, in order to compute the gradient $\nabla_\theta J(\theta)$.
- Back-propagation refers only to the method for computing the gradient.

Backward Propagation Algorithm

Algorithm 5: Backward Propagation

$da^{[L]} = \frac{-y}{a} + \frac{1-y}{1-a}$

for l *in range* $(1, L)$ **do**

  $dZ^{[l]} = dA^{[l]} * g'^{[l]}\left(Z^{[l]}\right)$

  $dW^{[l]} = dZ^{[l]} \cdot A^{[l-1]T}$

  $db^{[l]} = dZ^{[l]}$

  $dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$

Update the Weights and Bias

Algorithm 6: Weight Updation

for l *in range* $(1, L)$ **do**

$\quad W^{[l]} = W^{[l]} - \eta * dW^{[l]}$

$\quad b^{[l]} = b^{[l]} - \eta * db^{[l]}$

Output Neurons

- The choice of how to represent the output determines the form of the cross-entropy function.
- The feedforward network provides a set of hidden features $h = f(x; \theta)$.
- The role of the output layer is to provide some additional transformation.
- Types of output transformation
  1. Linear
  2. Sigmoid
  3. Softmax

Loss Functions

- Loss function compares the actual output from data set, in case of supervised learning, to the predicted output of the output layer.
- Based on the task, the loss functions differ.

For Regression	For Classification
Mean Squared Error (MSE)	Binary Cross Entropy (Sigmoid)
Mean Absolute Error (MAE)	Categorical Cross Entropy (Softmax)
Huber loss (Smooth MAE)	KL Divergence (Relative Entropy)
Log Hyperbolic Cosine (log cosh)	Exponential Loss
Quantile Loss	Hinge Loss
Cosine Similarity	

Further Reading

1. Dive into Deep Learning (T1)

Thank You!

Deep Learning
Module # 3 : Optimization

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

The instructor is gratefully acknowledging the authors who made their course materials freely available online.

What we learn ...

1. 3.1 Challenges in Neural Network Optimization - saddle points and plateau
2. 3.2 Non-convex optimization intuition
3. 3.3 Overview of optimization algorithms
4. 3.4 Momentum based algorithms
5. 3.5 Algorithms with Adaptive Learning Rates

Variance vs Bias

Figure: Underfitting; High Bias

Figure: Low variance and bias

Figure: Overfitting; High variance

- Our aim is to achieve low variance and bias.

Deep Learning Recipe (Andrew Ng)

In This Segment

1. Optimization
2. Challenges in Optimization
3. Optimization Algorithms
   - Gradient Descent
   - Effect of Learning Rate on Gradient Descent
   - Stochastic Gradient Descent
   - Dynamic Learning Rate
   - Minibatch Stochastic Gradient Descent
   - Momentum
   - Adagrad
   - RMSPRop
   - ADAM

Optimization

- Training of DNN is an iterative process.
- We may be training the DNN on a very large dataset or big data.
- The aim of DNN is improve the performance measure evaluated on the training set, i.e. **reduce the bias**.
- The performance measure P is optimized indirectly. Train the DNN to find the parameters θ that significantly reduce the cost function $J(\theta)$, so that P improves.

Optimization Algorithms

- Optimization algorithms train deep learning models.
- Optimization algorithms are the tools that allow
  - continue updating model parameters
  - to minimize the value of the loss function, as evaluated on the training set.
- In optimization, a loss function is often referred to as the **objective function** of the optimization problem.
- By tradition and convention most optimization algorithms are concerned with **minimization**.

Optimization

- General problem of optimization: find the value of $\boldsymbol{x}$ where $\mathbf{f}(\boldsymbol{x})$ is minimum

Why Optimization Algorithms?

- The **performance** of the optimization algorithm directly affects the model's training efficiency.
- Understanding the principles of different optimization algorithms and the role of their hyperparameters will enable us to **tune the hyperparameters in a targeted manner** to improve the performance of deep learning models.
- **The goal of optimization is to reduce the training error.** The goal of deep learning is to reduce the generalization error, this requires reduction in overfitting also.

In This Segment

1. Optimization
2. Challenges in Optimization
3. Optimization Algorithms
   - Gradient Descent
   - Effect of Learning Rate on Gradient Descent
   - Stochastic Gradient Descent
   - Dynamic Learning Rate
   - Minibatch Stochastic Gradient Descent
   - Momentum
   - Adagrad
   - RMSPRop
   - ADAM

Challenges in Optimization

- Local minima
- Saddle points
- Vanishing gradients

Finding Minimum of a Function

- Find the of x at which $f'(x) = 0$. i.e $\frac{df(x)}{dx} = 0$.
- Solution is at a turning point.
- A turning point is a value where the derivatives turn from positive to negative or vice versa.
- Turning points occur at minima, maxima or inflection (saddle) points.

Derivatives of a Function

- Both maxima and minima are turning points.
- Both maxima and minima have zero derivatives.
- The second derivative $f''(x)$ is negative at maxima and positive at minima.

Derivatives of a Function

- All locations with zero derivative are **critical points.**
  - These can be local minima, local maxima or saddle points.

Derivatives of a Function

- All locations with zero derivative are **critical points.**
  - These can be local minima, local maxima or saddle points.
- The second derivative is
  - ≥ 0 at minima
  - ≤ 0 at maxima
  - $= 0$ at saddle point
- It is more complicated for functions of multiple variables.

Finding Minimum of a Function

- Find the of x at which $f'(x) = 0$. i.e $\frac{df(x)}{dx} = 0$.
- Solution x_{soln} is at a turning point.
- Check the double derivative $f''(x_{soln})$ i.e $f''(x_{soln})\frac{df'(x)}{dx}$.
- If $f''(x_{soln})$ is positive, x_{soln} is minimum.
- If $f''(x_{soln})$ is negative, x_{soln} is maximum.
- If $f''(x_{soln})$ is zero, x_{soln} is saddle point.

Local Minima

- For any objective function $f(x)$, if the value of $f(x)$ at x is smaller than the values of $f(x)$ at any other points in the vicinity of x, then $f(x)$ could be a local minimum.
- If the value of $f(x)$ at x is the minimum of the objective function over the entire domain, then $f(x)$ is the global minimum.
- In minibatch stochastic gradient descent, the natural variation of gradients over minibatches is able to dislodge the parameters from local minima.

Saddle Points

- In high dimensional space, the local points with gradients is closer to zero are known as saddle points.
- Saddle points tend to slow down the learning process.
- A saddle point is any location where all gradients of a function vanish but which is neither a global nor a local minimum.
  Eg: $f(x, y) = x^2 - y^2$
  saddle point at (0, 0); maximum wrt y and minimum wrt x

Gradient of a Scalar Function with Multiple Variables

- The gradient $\nabla f(X)$ of a mutli-variate input X is a multiplicative factor that gives us the change in $f(X)$ for tiny variations in X.

$$df(X) = \nabla f(X) d(X)$$

Gradient of a Scalar Function with Multiple Variables

- Consider $f(X) = f(x_1, x_2, \ldots, x_n))$
- The gradient is given by

$$\nabla f(X) = \left[\frac{\partial f(X)}{\partial x_1}, \frac{\partial f(X)}{\partial x_2}, \ldots, \frac{\partial f(X)}{\partial x_n}\right]$$

- The second derivative of the function is given by the Hessain.

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \cdots & & & \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Gradient of a Scalar Function with Multiple Variables

- The optimum point is the turning point. The gradient is zero.

Solution of Unconstrained Minimization

- Solve for X where the gradient equation equal to zero.

$$\nabla f(X) = 0$$

- Compute the Hessian matrix $\nabla^2 f(X)$ at the candidate solution and verify that
- Local Minimum
  - Eigenvalues of Hessian matrix are all positive.
- Local Maximum
  - Eigenvalues of Hessian matrix are all negative.
- Saddle Point
  - Eigenvalues of Hessian matrix at the zero-gradient position are negative and positive.

Example

- Minimize

$$f(x_1, x_2, x_3) = x_1^2 + x_1(1 - x_2) + x_2^2 - x_2 x_3 + x_3^2 + x_3$$

- Gradient

$$\nabla f(X) = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix}$$

Example

- Set Gradient = 0

$$\nabla f(X) = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- Solve the 3 equations in 3 unknowns.

$$X = (x_1, x_2, x_3) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

Example

- Compute the Hessian

$$\nabla^2 f(X) = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

- Evaluate the eigenvalues of the Hessain matrix

$$\lambda_1 = 3.414 \quad \lambda_2 = 0.586 \quad \lambda_3 = 2$$

- All Eigen vectors are positive.
- The obtained solution is minimum.

$$X = (x_1, x_2, x_3) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

Vanishing Gradients

- Function $f(x) = tanh(x)$
- $f'(x) = 1 - tanh2(x)$
- $f'(4) = 0.0013$.
- Gradient of f is close to nil.
- Vanishing gradients can cause optimization to stall.
- Reparameterization of the problem helps.
- Good initialization of the parameters.

In This Segment

1. Optimization
2. Challenges in Optimization
3. Optimization Algorithms
   - Gradient Descent
   - Effect of Learning Rate on Gradient Descent
   - Stochastic Gradient Descent
   - Dynamic Learning Rate
   - Minibatch Stochastic Gradient Descent
   - Momentum
   - Adagrad
   - RMSPRop
   - ADAM

How to find Global Minima?

- It is not possible to solve $\nabla f(X) = 0$.
  - The function to minimize / maximize may have a intractable form
- In these situation, iterative solutions are used.
  - Begin with a guess for the optimal X and refine it iteratively until the correct value is obtained.

Find Global Minima Iteratively

- Iterative solutions
  - Start with an initial guess X_0.
  - Update the guess towards a better value of $f(X)$.
  - Stop when $F(X)$ no longer decreases.
- Challenges
  - Which direction to step in?
  - How big the steps should be?

Gradient Descent

- Iterative Solution
  - Start at some initial guess.
  - Find the direction to shift this point to decrease error.
    - This can be found from the derivative of the function.
    - A positive derivative implies, move left to decrease error.
    - A negative derivative implies, move right to decrease error.
  - Shift the point in this direction.

Gradient Descent

Algorithm 1: Gradient Descent Algorithm

Initialize x_0
while $f'(x^k) \neq 0$ **do**
  $x^{k+1} = x^k - step_size f'(x^k)$

Gradient Descent

- First order Gradient Descent algos consider the first order derivatives to get the magnitude and direction of update.

```
def gd(eta, f_grad):
 x = 10.0
 results = [x]
 for i in range(10):
   x -= eta * f_grad(x)
   results.append(float(x))
 return results
```

Numerical Example

Consider an error function

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

Different variants of gradient descent algorithm can be used to minimize this error function w.r.t (w_1, w_2). Assume $(w_1, w_2) = (1, 1)$ at time $(t - 1)$ and after update $(w_1, w_2) = (1.5, 2.0)$ at time (t). Assume $\alpha = 1.5, \beta = 0.6, \eta = 0.3$.

Numerical Example

Compute the value that minimizes (w_1, w_2). Compute the minimum possible value of error.

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1 - 3)}{4} - \frac{(w_2 - 4)}{6} = 0$$

$$\frac{\partial E}{\partial w_2} = \frac{2(w_2 - 4)}{9} - \frac{(w_1 - 3)}{6} = 0$$

Solving $w_1 = 3$

$w_2 = 4$

Substituting $E = 0.05$

Numerical Example

Compute the value of (w_1, w_2) at time $(t+1)$ if standard gradient descent is used.

$$E(w_1, w_2) = 0.05 + \frac{(w_1-3)^2}{4} + \frac{(w_2-4)^2}{9} - \frac{(w_1-3)(w_2-4)}{6}$$

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1-3)}{4} - \frac{(w_2-4)}{6}$$

$$\frac{\partial E}{\partial w_2} = \frac{2(w_2-4)}{9} - \frac{(w_1-3)}{6}$$

at $(t+1)$

$$w_{1(t+1)} = w_{1(t)} - \eta \frac{\partial E}{\partial w_1} = 1.5 - 0.3 * \left[\frac{(1.5-3)}{2} - \frac{(2-4)}{6}\right] = 1.625$$

$$w_{2(t+1)} = w_{2(t)} - \eta \frac{\partial E}{\partial w_2} = 2 - 0.3 * \left[\frac{2(2-4)}{9} - \frac{(1.5-3)}{6}\right] = 2.05$$

Learning Rate

- The role of the learning rate is to moderate the degree to which weights are changed at each step.
- Learning rate η is set by the algorithm designer.
- If the learning rate that is too small, it will cause parameters to update very slowly, requiring more iterations to get a better solution.
- If the learning rate that is too large, the solution oscillates and in the worst case it might even diverge.

Gradient Descent

```
gd(eta = 0.05, f_grad)
```

- Small learning rate
  - Slow Learning
  - Definitely converge

```
gd(eta = 1.1, f_grad)
```

- Large learning rate
  - Oscillations
  - Worst case it might diverge

Example

Error surface is given by $E(x, y, z) = 3x^2 + 2y^2 + 4z^2 + 6$. Assume gradient descent is used to find the minimum of this error surface. What is the optimal learning rate that leads to fastest convergence to the global minimum?

$$
\begin{aligned}
E(x, y, z) &= 3x^2 + 2y^2 + 4z^2 + 6 \\
\eta_{xopt} &= 1/6 \\
\eta_{yopt} &= 1/4 \\
\eta_{zopt} &= 1/8 \\
\text{optimal learning rate for convergence} &= \min[\eta_{xopt}, \eta_{yopt}, \eta_{zopt}] = 1/8 \\
\text{Largest learning rate for convergence} &= \min[2\eta_{xopt}, 2\eta_{yopt}, 2\eta_{zopt}] = 0.33 \\
\text{Learning rate for divergence} &> 2\eta_{opt} = 2 * 1/8 = 0.25
\end{aligned}
$$

Stochastic Gradient Descent

- In deep learning, the objective function is the average of the loss functions for each example in the training dataset.
- Given a training dataset of m examples, let $f_i(\theta)$ is the loss function with respect to the training example of index i, where θ is the parameter vector.
- Computational cost of each iteration is O(1).

Stochastic Gradient Descent Algorithm

Algorithm 2: Stochastic Gradient Descent Algorithm

Data: Learning Rate η
Data: Initial Parameters θ
while *stopping criterion not met* **do**
  for *each example i* **do**
    Perform Forward prop on $X^{\{i\}}$ to compute $\hat{y}$ and cost $J(\theta)$
    Perform Backprop on $\left(X^{\{i\}}, Y^{\{i\}}\right)$ to compute gradient $\mathbf{g}$
    Update parameters as $\theta \leftarrow \theta - \eta \mathbf{g}$

Stochastic Gradient Descent

- Trajectory of the variables in the stochastic gradient descent is much more noisy. This is due to the stochastic nature of the gradient. Even near the minimum, uncertainty is injected by the instantaneous gradients.

```
def sgd(x1, x2, s1, s2, f_grad):
 g1, g2 = f_grad(x1, x2)
 # Simulate noisy gradient
 g1 += np.random.normal(0.0, 1, (1,))
 g2 += np.random.normal(0.0, 1, (1,))
 eta_t = eta * lr()
 return (x1-eta_t*g1, x2-eta_t*g2, 0, 0)
```

Dynamic Learning Rate

- Replace η with a time-dependent learning rate $\eta(t)$ adds to the complexity of controlling convergence of an optimization algorithm.
- A few basic strategies that adjust η over time.
  - Piecewise constant
    - decrease the learning rate, e.g., whenever progress in optimization stalls.
    - This is a common strategy for training deep networks

$$\eta(t) = \eta_i \qquad \textit{if} \quad t_i \leq t \leq t_{i+1}$$

  - Exponential decay
    - Leads to premature stopping before the algorithm has converged.

$$\eta(t) = \eta_0 e^{-\lambda t}$$

  - Polynomial decay

$$\eta(t) = \eta_0 (\beta t + 1)^{-\alpha}$$

Exponential Decay

- Variance in the parameters is significantly reduced.
- The algorithm fails to converge at all.

```
def exponential_lr():
 global t
 t += 1
 return math.exp(-0.1 * t)
```

Polynomial Decay

- Learning rate decays with the inverse square root of the number of steps.
- Convergence gets better after only 50 steps.

```
def polynomial_lr():
 global t
 t += 1
 return (1 + 0.1 * t) ** (-0.5)
```

Gap

- Gradient descent
  - Uses the full dataset to compute gradients and to update parameters, one pass at a time.
  - Gradient Descent is not particularly data efficient whenever data is very similar.
- Stochastic Gradient descent
  - Processes one observation at a time to make progress.
  - Stochastic Gradient Descent is not particularly computationally efficient since CPUs and GPUs cannot exploit the full power of vectorization.
  - For noisy gradients, choice of the learning rate is critical.
  - If we decrease it too rapidly, convergence stalls.
  - If we are too lenient, we fail to converge to a good enough solution since noise keeps on driving us away from optimality.
- Minibatch SGD
  - Accelerate computation, or better or computational efficiency.
  - A[illegible]dients reduce the amount of variance.

Shiv Nadar University Deemed to be University under Section 3 of UGC Act, 1956

Minibatch Stochastic Gradient Descent

- Suppose we have 4 million examples in the training set.

$$X = [X^{(1)}, X^{(2)}, \ldots, X^{(m)}]$$
$$Y = [Y^{(1)}, Y^{(2)}, \ldots, Y^{(m)}]$$

- The gradient descent algorithm process the entire dataset of $m = 4$ million examples before proceeding to the next step. This is cumbersome and time consuming.
- Hence split the training dataset.

Minibatch Stochastic Gradient Descent

- Let the mini-batch size be 1000.
- Then we have 4000 mini-batches, each having 1000 examples.

$$
\begin{aligned}
X &= [X^{(1)}, X^{(2)}, \ldots, X^{(m)}] \\
&= [X^{(1)}, X^{(2)}, \ldots, X^{(1000)}, X^{(1001)}, \ldots, X^{(2000)}, \ldots, X^{(m)}] \\
&= X^{\{1\}}, X^{\{2\}}, \ldots, X^{\{m/b\}} \\
Y &= [Y^{(1)}, Y^{(2)}, \ldots, Y^{(m)}] \\
&= [Y^{(1)}, Y^{(2)}, \ldots, Y^{(1000)}, Y^{(1001)}, \ldots, Y^{(2000)}, \ldots, Y^{(m)}] \\
&= Y^{\{1\}}, Y^{\{2\}}, \ldots, Y^{\{m/b\}}
\end{aligned}
$$

- The minibatch t is denoted as $X^{\{t\}}, Y^{\{t\}}$
- The Gradient descent algorithm will be repeated for t batches.

Minibatch Stochastic Gradient Descent Algorithm

- In each iteration, we first randomly sample a minibatch B consisting of a fixed number of training examples.
- We then compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.
- Finally, we multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$$\theta \leftarrow \frac{\eta}{|B|} \sum_{i \in B} \partial_\theta loss^{(t)}(\theta)$$

- $|B|$ represents the number of examples in each minibatch (the batch size) and η denotes the learning rate.

Minibatch Stochastic Gradient Descent Algorithm

- Gradients at time t is calculated as

$$g_{t,t-1} = \partial_{theta} \frac{1}{|B|} \sum_{i \in B_t} f(x_i, \theta_{t-1}) = \frac{1}{|B|} \sum_{i \in B_t} h_{i,t-1}$$

- $|B|$ represents the number of examples in each minibatch (the batch size) and *eta* denotes the learning rate.
- $h_{i,t-1} = \partial_\theta f(x_i, \theta_{t-1})$ is the stochastic gradient descent for sample i using the weights updated at time $t-1$.

Minibatch Stochastic Gradient Descent Algorithm

Algorithm 3: Minibatch Stochastic Gradient Descent Algorithm

Data: Learning Rate η
Data: Initial Parameters θ
while *stopping criterion not met* **do**
  for *t in range* $(1, m/batch_size)$ **do**
    Perform Forward prop on $X^{\{t\}}$ to compute $\hat{y}$ and cost $J(\theta)$
    Perform Backprop on $\left(X^{\{t\}}, Y^{\{t\}}\right)$ to compute gradient $\mathbf{g}$
    Update parameters as $\theta \leftarrow \theta - \eta \mathbf{g}$

Minibatch Size

- Typical mini batch size is taken as some power of 2; say 32, 64, 128, 256.
- Better utilization of multicore architecture.
- Amount of memory scales with batch size.

Leaky Average in Minibatch SGD

- Replace the gradient computation by a "leaky average" for better variance reduction.

$$\beta \in (0, 1)$$

- This effectively replaces the instantaneous gradient by one that's been averaged over multiple past gradients.
- v is called momentum.
- **Momentum accumulates past gradients.**
- Large β amounts to a long-range average and small β amounts to only a slight correction relative to a gradient method.
- The new gradient replacement no longer points into the direction of steepest descent on a particular instance any longer but rather in the direction of a weighted average of past gradients.

Momentum Method Example

- Consider a moderately distorted ellipsoid objective $f(x) = 0.1x_1^2 + 2x_2^2$
- f has its minimum at $(0, 0)$. This function is very flat in x_1 direction.

- For $\eta = 0.4$ Without momentum
- The gradient in the x_2 direction oscillates than in the horizontal x_1 direction.

- For $\eta = 0.6$ Without momentum
- Convergence in the x_1 direction improves but the overall solution quality is diverging.

Momentum Method Example

- Consider a moderately distorted ellipsoid objective $f(x) = 0.1x_1^2 + 2x_2^2$
- Apply momentum for $\eta = 0.6$

- For $\beta = 0.5$
- Converges well. Lesser oscillations. Larger steps in x_1 direction.

- For $\beta = 0.25$
- Reduced convergence. More oscillations. Larger magnitude of oscillations.

Momentum

- Compute an exponentially weighted average of the gradients and use it to update the parameters.
- The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.
- The momentum introduces a velocity v that denotes the direction and speed at which the parameters change.
- The velocity is set to an exponentially decaying average of the negative gradient.
- The Momentum parameter $\beta \in [0, 1)$ is a hyperparameter and typically takes a value of 0.9.

SGD with Momentum

SGD with Momentum Algorithm

Algorithm 4: Stochastic Gradient Descent with Momentum

Data: Learning Rate η, Momentum Parameter β

Data: Initial Parameters θ , Inital velocities v

```
while stopping criterion not met do
    for t in range (1, m/batch_size) do
        Perform Forward prop on X^{t} to compute ŷ and cost J(θ)
        Perform Backprop on (X^{t}, Y^{t}) to compute gradient g
        Update velocities as v = βv + (1 − β)g
        Update parameters as θ ← θ − ηv
```

Momentum method: Summary

- Momentum replaces gradients with a leaky average over past gradients. This accelerates convergence significantly.
- Momentum prevents stalling of the optimization process that is much more likely to occur for stochastic gradient descent.
- The effective number of gradients is given by $1/(1-\beta)$ due to exponentiated down weighting of past data.
- Implementation is quite straightforward but it requires us to store an additional state vector (momentum v).

Numerical Example

Consider an error function

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

Different variants of gradient descent algorithm can be used to minimize this error function w.r.t (w_1, w_2). Assume $(w_1, w_2) = (1, 1)$ at time $(t - 1)$ and after update $(w_1, w_2) = (1.5, 2.0)$ at time (t). Assume $\alpha = 1.5, \beta = 0.6, \eta = 0.3$. Compute the value of (w_1, w_2) at time $(t + 1)$ if momentum is used.

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1 - 3)}{4} - \frac{(w_2 - 4)}{6}$$

$$\frac{\partial E}{\partial w_2} = \frac{2(w_2 - 4)}{9} - \frac{(w_1 - 3)}{6}$$

Numerical Example

$$at \quad (t+1) \qquad w_{1(t+1)} = w_{1(t)} - \eta \frac{\partial E}{\partial w_1} + \beta \Delta w_1$$

$$= 1.5 - 0.3 * \left[\frac{(1.5-3)}{2} - \frac{(2-4)}{6} \right] + 0.9 * (1.5-1)$$

$$= 2.075$$

$$w_{2(t+1)} = w_{2(t)} - \eta \frac{\partial E}{\partial w_2} + \beta \Delta w_1$$

$$= 2 - 0.3 * \left[\frac{2(2-4)}{9} - \frac{(1.5-3)}{6} \right] + 0.9 * (2-1)$$

$$= 2.958$$

Adagrad

- Used for features that occur infrequently (sparse features).
- Adagrad uses aggregate of the squares of previously observed gradients.

Adagrad Algorithm

- Variable s_t to accumulate past gradient variance.
- Operation are applied coordinate wise. $\sqrt{1/v}$ has entries $\sqrt{1/v_1}$ and $u \cdot v$ has entries $u_i v_i$.
- η is the learning rate and ϵ is an additive constant that ensures that we do not divide by 0.
- Initialize $s_0 = 0$.

$$
\begin{aligned}
g_t &= \partial_w loss(y_t, f(x_t, w)) \\
s_t &= s_{t-1} g_t^2 \\
w_t &= w_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \cdot g_t
\end{aligned}
$$

Adagrad: Summary

- Adagrad decreases the learning rate dynamically on a per-coordinate basis.
- It uses the magnitude of the gradient as a means of adjusting how quickly progress is achieved - coordinates with large gradients are compensated with a smaller learning rate.
- If the optimization problem has a rather uneven structure Adagrad can help mitigate the distortion.
- Adagrad is particularly effective for sparse features where the learning rate needs to decrease more slowly for infrequently occurring terms.
- On deep learning problems Adagrad can sometimes be too aggressive in reducing learning rates.

RMSProp (Root Mean Square Prop)

- Adapts the learning rates of all model parameters by scaling the gradients into an exponentially weighted moving average.
- Performs better with non-convex setting.
- Adagrad use learning rate that decreases at a predefined schedule of effectively $O(t^{-1/2})$.
- RMSProp algorithm decouples rate scheduling from coordinate-adaptive learning rates. This is essential for non-convex optimization.

RMSProp Algorithm

- Use leaky average to accumulate past gradient variance.
- Parameter $\gamma > 0$.
- The constant ϵ is set to 10^{-6} to ensure that we do not suffer from division by zero or overly large step sizes.

$$s_t = \gamma s_{t-1} + (1-\gamma) g_t^2$$
$$w_t = w_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g_t$$

RMSProp Algorithm

Algorithm 5: RMSProp

Data: Learning Rate η, Decay Rate γ

Data: Small constant $\epsilon = 10^{-7}$ to stabilize division

Data: Initial Parameters θ

Initialize accumulated gradient $s = 0$

while *stopping criterion not met* **do**

  for t *in range* $(1, m/batch_size)$ **do**

    Perform Forward prop on $X^{\{t\}}$ to compute $\hat{y}$ and cost $J(\theta)$

    Perform Backprop on $\left(X^{\{t\}}, Y^{\{t\}}\right)$ to compute gradient $\mathbf{g}$

    Compute accumulated squared gradient $s = \gamma s + (1-\gamma)\mathbf{g}^2$

    Update parameters as $\theta \leftarrow \theta - \frac{\eta}{\sqrt{s+\epsilon}} \odot \mathbf{g}$

RMSProp Example

```
def rmsprop_2d(x1, x2, s1, s2):
 g1, g2, eps = 0.2*x1, 4*x2, 1e-6
 s1 = gamma*s1 +(1-gamma)*g1**2
 s2 = gamma*s2 +(1-gamma)*g2**2
 x1 -= eta/math.sqrt(s1 + eps)*g1
 x2 -= eta/math.sqrt(s2 + eps)*g2
 return x1, x2, s1, s2
```

RMSProp: Summary

- RMSProp is very similar to Adagrad as both use the square of the gradient to scale coefficients.
- RMSProp shares with momentum the leaky averaging. However, RMSProp uses the technique to adjust the coefficient-wise preconditioner.
- The learning rate needs to be scheduled by the experimenter in practice.
- The coefficient γ determines how long the history is when adjusting the per-coordinate scale.

Review of algorithms learned so far

1. Stochastic gradient descent
   - more effective than Gradient Descent when solving optimization problems, e.g., due to its inherent resilience to redundant data.
2. Minibatch Stochastic gradient descent
   - affords significant additional efficiency arising from vectorization, using larger sets of observations in one minibatch. This is the key to efficient multi-machine, multi-GPU and overall parallel processing.
3. Momentum
   - added a mechanism for aggregating a history of past gradients to accelerate convergence.

Review of algorithms learned so far

1. Adagrad
   - used per-coordinate scaling to allow for a computationally efficient preconditioner.
2. RMSProp
   - decoupled per-coordinate scaling from a learning rate adjustment.

Adam (Adaptive Moments)

- Adam combines all these techniques into one efficient learning algorithm.
- Very effective and robust algorithm
- Adam can diverge due to poor variance control. (disadvantage)
- Adam uses exponential weighted moving averages (also known as leaky averaging) to obtain an estimate of both the momentum and also the second moment of the gradient.

Adam Algorithm

- State variables

$$v_t \leftarrow \beta_1 v_{t-1} + (1 - \beta_1) g_t$$
$$s_t \leftarrow \beta_2 s_{t-1} + (1 - \beta_2) g_t^2$$

- β_1 and β_2 are nonnegative weighting parameters. Common choices for them are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. That is, the variance estimate moves much more slowly than the momentum term.
- Initialize $v_0 = s_0 = 0$.

Adam Algorithm

- Normalize the state variables

$$\hat{v}_t = \frac{v_t}{1 - \beta_1^t}$$

$$\hat{s}_t = \frac{s_t}{1 - \beta_2^t}$$

- Rescale the gradient

$$\hat{g}_t = \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t} + \epsilon}$$

- Compute updates

$$w_t \leftarrow w_{t-1} - \frac{\eta \hat{v}}{\sqrt{\hat{s}} + \epsilon}$$

Adam Algorithm

Algorithm 6: Adam

Data: Learning Rate $\eta = 0.001$, Momentum parameter $\beta_1 = 0.9$, Decay Rate $\beta_2 = 0.999$, Small constant $\epsilon = 10^{-7}$ to stabilize division

Data: Initial Parameters θ; Initialize first and second moments $v = 0, s = 0$

while *stopping criterion not met* **do**

  for *t in range* $(1, m/batch_size)$ **do**

    Perform Forward prop on $X^{\{t\}}$ to compute $\hat{y}$ and cost $J(\theta)$

    Perform Backprop on $\left(X^{\{t\}}, Y^{\{t\}}\right)$ to compute gradient $\mathbf{g}$

    Update first moment as $v \leftarrow \beta_1 v + (1 - \beta_1)\mathbf{g}$

    Update second moment as $s = \beta_2 s + (1 - \beta_2)\mathbf{g}^2$

    Perform bias correction as $\hat{v} = \frac{v}{1-\beta_1^t}$ and $\hat{s} = \frac{s}{1-\beta_2^t}$

    Update parameters as $\theta \leftarrow \theta - \frac{\eta \hat{v}}{\sqrt{\hat{s}}+\epsilon}$

Adam: Summary

- Adam combines features of many optimization algorithms into a fairly robust update rule.
- Adam uses bias correction to adjust for a slow startup when estimating momentum and a second moment.
- For gradients with significant variance we may encounter issues with convergence. They can be amended by using larger minibatches or by switching to an improved estimate for state variables.
- Yogi algorithm offers such an alternative.

Numerical Example

The training data $(x_1, y_1) = (3.5, 0.5)$ for a single Sigmoid neuron and initial values of $w = -2.0, b = -2.0, \eta = 0.10, \beta_1 = 0.90, \beta_2 = 0.99, \epsilon = 1e-8, s_W = 0$ and $s_B = 0$ are provided. Showing all the calculations, find out the values of w, b, s_W, s_B, r_W, r_B after one iteration of Adam. Use BCE as loss function. Apply bias-correction.

Numerical Example

- Forward Pass and Calculate Loss

$$\hat{y} = \sigma(wx + b) = \sigma(-2.0 \times 3.5 - 2.0) \approx 0.0180$$
$$L = -[y \times \log(\hat{y}) + (1 - y) \times \log(1 - \hat{y})$$
$$L = -[0.5 \times \log(0.0180) + (1 - 0.5) \times \log(1 - 0.0180)] \approx 4.0076$$

- Calculate Gradients

$$\nabla_w L = (\hat{y} - y) \times x = (0.0180 - 0.5) \times 3.5 \approx -0.493$$
$$\nabla_b L = \hat{y} - y = 0.0180 - 0.5 \approx -0.482$$

- Update First Moments

$$s_W = \beta_1 \times s_W + (1 - \beta_1) \times \nabla_w L = 0.90 \times 0 - 0.10 \times (-0.493) \approx 0.04437$$
$$s_B = \beta_1 \times s_B + (1 - \beta_1) \times \nabla_b L = 0.90 \times 0 - 0.10 \times (-0.482) \approx 0.04382$$

Numerical Example

- Update Second Moments

$$
\begin{aligned}
r_W &= \beta_2 \times r_W + (1-\beta_2) \times (\nabla_w L)^2 \\
&= 0.999 \times 0 - 0.001 \times (-0.493)^2 \approx 0.0001216 \\
r_B &= \beta_2 \times r_B + (1-\beta_2) \times (\nabla_b L)^2 \\
&= 0.999 \times 0 - 0.001 \times (-0.482)^2 \approx 0.0001168
\end{aligned}
$$

- Corrected First Moments

$$
\hat{s}_W = \frac{s_W}{1-\beta_1^1} \approx \frac{0.04437}{1-0.90^1} \approx 0.4437
$$

$$
\hat{s}_B = \frac{s_B}{1-\beta_1^1} \approx \frac{0.04382}{1-0.90^1} \approx 0.4382
$$

Numerical Example

- Corrected Second Moments

$$\hat{r}_W = \frac{r_W}{1-\beta_2^1} \approx \frac{0.0001216}{1-0.999^1} \approx 0.0608$$

$$\hat{r}_B = \frac{r_B}{1-\beta_2^1} \approx \frac{0.0001168}{1-0.999^1} \approx 0.0584$$

- Update Weights and Biases

$$w = w - \frac{\eta}{\sqrt{\hat{r}_W}+\epsilon} \times \hat{s}_W = -2.0 - \frac{0.10}{\sqrt{0.0608}+1e-8} \times 0.4437 \approx -2.081$$

$$b = b - \frac{\eta}{\sqrt{\hat{r}_B}+\epsilon} \times \hat{s}_B = -2.0 - \frac{0.10}{\sqrt{0.0584}+1e-8} \times 0.4382 \approx -2.079$$

References

1. Dive into Deep Learning by Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola
   `https://d2l.ai/chapter_introduction/index.html`
2. Deep Learning by Ian Goodfellow, YoshuaBengio, Aaron Courville
   `https://www.deeplearningbook.org/`

Thank You!

Deep Neural Network

AIML Module 5

Seetha Parameswaran

BITS Pilani

The author of this deck, Prof. Seetha Parameswaran, is gratefully acknowledging the authors who made their course materials freely available online.

Regularization Techniques

What we Learn….

4.1 Model Selection

4.2 Underfitting, and Overfitting

4.3 L1 and L2 Regularization

4.4 Dropout

4.5 Challenges - Vanishing and Exploding Gradients, Covariance shift

4.6 Parameter Initialization

4.7 Batch Normalization

Generalization in DNN

Generalization

- Goal is to discover patterns that generalize.
  - The goal is to discover patterns that capture regularities in the underlying population from which our training set was drawn.
  - Models are trained a sample of data.
  - When working with finite samples, we run the risk that we might discover apparent associations that turn out not to hold up when we collect more data or on newer samples.
- The trained model should predict for newer or unseen data. This problem is called generalization.

Training Error and Generalization Error

- **Training error** is the error of our model as calculated on the training dataset.
  - Obtained while training the model.
- **Generalization error** is the expectation of our model's error, if an infinite stream of additional data examples drawn from the same underlying data distribution as the original sample were applied on the model.
  - Cannot be computed, but estimated.
  - Estimate the generalization error by applying the model to an independent test set, constituted of a random selection of data examples that were withheld from the training set.

Model Complexity

- Simple models and abundant data
  - Expect the generalization error to resemble the training error.
- More complex models and fewer examples
  - Expect the training error to go down but the generalization gap to grow.
- Model complexity
  - A model with more parameters might be considered more complex.
  - A model whose parameters can take a wider range of values might be more complex.
  - A neural network model that takes more training iterations are more complex, and one subject to early stopping (fewer training iterations) are less complex.
-

Factors that influence the generalizability of a model

1. The number of tunable parameters.
   - When the number of tunable parameters, called the degrees of freedom, is large, models tend to be more susceptible to overfitting.
2. The values taken by the parameters.
   - When weights can take a wider range of values, models can be more susceptible to overfitting.
3. The number of training examples.
   - It is trivially easy to overfit a dataset containing only one or two examples even if your model is simple. But overfitting a dataset with millions of examples requires an extremely flexible model.
4.

Model Selection

- Model selection is the process of selecting the final model after evaluating several candidate models.
- With MLPs, compare models with
  - different numbers of hidden layers,
  - different numbers of hidden units
  - different activation functions applied to each hidden layer.
- Use Validation dataset to determine the best among our candidate models.

Validation dataset

- Never rely on the test data for model selection.
  - Risk of overfit the test data
- Do not rely solely on the training data for model selection
  - We cannot estimate the generalization error on the very data that we use to train the model.
- Split the data three ways, incorporating a validation dataset (or validation set) in addition to the training and test datasets.
- In deep learning, with millions of data available, the split is generally
  - Training = 98-99 % of the original dataset
  - Validation = 1-2 % of training dataset
  - Testing = 1-2 % of the original dataset

Just Right Model

- High Training accuracy
- High Validation accuracy
- Low Bias and Low Variance
- Usually care more about the validation error than about the gap between the training and validation errors.

```
844/844 [==============================] - 3s 3ms/step - loss: 0.0848 - accuracy: 0.9750 - val_loss: 0.0714 - val_
accuracy: 0.9790
Epoch 2/25
844/844 [==============================] - 2s 2ms/step - loss: 0.0821 - accuracy: 0.9761 - val_loss: 0.0730 - val_
accuracy: 0.9802
Epoch 3/25
844/844 [==============================] - 2s 2ms/step - loss: 0.0793 - accuracy: 0.9769 - val_loss: 0.0706 - val_
accuracy: 0.9798
Epoch 4/25
844/844 [==============================] - 2s 3ms/step - loss: 0.0769 - accuracy: 0.9771 - val_loss: 0.0753 - val_
accuracy: 0.9795
```

Underfitting

- Low Training accuracy and Low Validation accuracy.

```
Train on 2594 samples, validate on 495 samples
Epoch 1/200
2594/2594 [==============================] - 6s 2ms/step - loss: 2.8738 - acc: 0.1191 - val_loss: 2.2041 - val_a
cc: 0.1232
Epoch 2/200
2594/2594 [==============================] - 5s 2ms/step - loss: 2.1731 - acc: 0.1415 - val_loss: 2.1776 - val_a
cc: 0.1192
Epoch 3/200
2594/2594 [==============================] - 5s 2ms/step - loss: 2.0862 - acc: 0.1974 - val_loss: 2.1261 - val_a
cc: 0.1596
Epoch 4/200
2594/2594 [==============================] - 5s 2ms/step - loss: 1.9268 - acc: 0.2787 - val_loss: 2.0487 - val_a
cc: 0.2525
```

- Training error and validation error are both substantial but there is a little gap between them.
- The model is too simple (insufficiently expressive) to capture the pattern that we are trying to model.
- If generalization gap between our training and validation errors is small, a more complex model may be better.

Overfitting

- The phenomenon of fitting the training data more closely than fit the underlying distribution is called **overfitting.**
- High Training accuracy and Low Validation accuracy
- Training error is significantly lower than the validation error.
- The techniques used to combat overfitting are called **regularization**.

```
Training accuracy =  1.0
Validation accuracy =  0.43689320475152393
Testing accuracy =  0.5614035087719298
```

Underfitting or Overfitting?

Simple Model
Underfitting

Complex Model
Overfitting

Just right model

Polynomial degree and underfitting vs. overfitting

Model complexity and dataset size

- More data, fit a more complex model.
- More data, the generalization error typically decreases.

- Less data, simpler models may be more difficult to beat.
- Less data, more likely and more severely, models may over-fit.

The current success of deep learning owes to the current abundance of massive datasets due to Internet companies, cheap storage, connected devices, and the broad digitization of the economy.

Deep Learning Model Selection

Credit: Andrew Ng

Regularization

Regularization Techniques

- Weight Decay (L2 regularization)
- Dropout
- Early Stopping

Weight Decay

L2 Regularization

- Measure the complexity of a linear function $f(x) = w^{\top}x$ by some norm of its weight vector, e.g., $\| w \|^2$.
- Add the norm as a penalty term to the problem of minimizing the loss. This will ensure that the weight vector is small.
- The objective function becomes **minimizing the sum of the prediction loss and the penalty term.**
- L2-regularized linear models constitute the ridge regression algorithm.

L2 Regularization

- The trade off between standard loss and the additive penalty is given by regularization constant λ, a non-negative hyperparameter.

$$Objective function = \mathcal{L}(w, b) + \frac{\lambda}{2} \mid\mid w \mid\mid^2$$

$$Weight updation \qquad w \leftarrow (1 - \eta\lambda)w - \eta g$$

- For λ = 0, we recover the original loss function.
- For λ > 0, we restrict the size of // w // .
- Smaller values of λ correspond to less constrained w, whereas larger values of λ constrain w more considerably.
- By squaring the L2 norm, we remove the square root, leaving the sum of squares of each component of the weight vector.
- The derivative of a quadratic function ensure that the 2 and 1/2 cancel out.

L1 Regularization

- L1-regularized linear regression is known as lasso regression.
- L1 penalties lead to models that concentrate weights on a small set of features by clearing the other weights to zero. This is called feature selection.

$$Objective function = \mathcal{L}(w, b) + \lambda \sum_i | w_i |$$

$$Weight updation \qquad w \leftarrow (1 - \eta\lambda)w - \eta g$$

L2 Regularization	L1 Regularization
Sum of square of weights	Sum of absolute value of weights
Learn complex data patterns	Generates simple and interpretable models
Estimate mean of data	Estimate median of data
Not robust to outliers	Robust to outliers
Shrink coefficients equally	Shrink coefficients to zero
Non sparse solution	Sparse solution
One solution	Multiple solutions
No feature selection	Selects features
Useful for collinear features	Useful for dimensionality reduction

Dropout

Smoothness

- Classical generalization theory suggests that to close the gap between train and test performance, aim for a simple model.
- Simplicity can be achieved
  - Using weight decay
  - Smoothness, i.e., that the function should not be sensitive to small changes to its inputs.
- Injecting noise enforces smoothness
  - training with input noise
  - inject noise into each layer of the network before calculating the subsequent layer during training.

Dropout

- **Dropout** involves injecting noise while computing each internal layer during forward propagation.
- It has become a **standard** technique for training neural networks.
- The method is called **dropout** because we literally drop out some neurons during training.
- Apply dropout to a hidden layer, zeroing out each hidden unit with probability *p*.
- The calculation of the outputs no longer depends on dropped out neurons and their respective gradient also vanishes when performing backpropagation. (in that iteration)
- Dropout is disabled at test time.

Without Dropout

Dropout for first iteration

Keep probability = 0.5

Dropout for second iteration

Keep probability = 0.5

Dropout

- Dropout gives a smaller neural network, giving the effect of regularization.
- In general,
  - Vary keep probability (0.5 to 0.8) for each hidden layer.
  - The input layer has a keep probability of 1.0 or 0.9.
  - The output layer has a keep probability of 1.0.
-

Early Stopping

Before Early stopping

- When training large models, training error decreases steadily over time, but validation set error begins to rise again.
- Training objective decreases consistently over time.
- Validation set average loss begins to increase again, forming an asymmetric U-shaped curve.

Early stopping

- No longer looking for a local minimum of validation error, while training.
- Train until the validation set error has not improved for some amount of time.
- Every time the error on the validation set improves, store a copy of the model parameters. When the training algorithm terminates, return these parar

Early stopping

- Effective and simple form of regularization.
- Trains simpler models

Early Stopping code

```
>>> callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
>>> # This callback will stop the training when there is no improvement in
>>> # the loss for three consecutive epochs.
>>> model = tf.keras.models.Sequential([tf.keras.layers.Dense(10)])
>>> model.compile(tf.keras.optimizers.SGD(), loss='mse')
>>> history = model.fit(np.arange(100).reshape(5, 20), np.zeros(5),
...                     epochs=10, batch_size=1, callbacks=[callback],
...                     verbose=0)
>>> len(history.history['loss'])  # Only 4 epochs are run.
4
```

Numerical Stability and Initialization

Why Initialization is important?

- The choice of initialization is crucial for maintaining numerical stability.
- The choices of initialization can be tied up in interesting ways with the choice of the nonlinear activation function.
- Which function we choose and how we initialize parameters can determine how quickly our optimization algorithm converges.
- Poor choices can cause to encounter exploding or vanishing gradients while training.

Vanishing and Exploding Gradients

- Consider a deep network with **L** layers, input **x** and output **o**. With each layer **l** defined by a transformation $\mathbf{f_l}$ parameterized by weights **W(l)** , whose hidden variable is **h(l)**

$$\mathbf{h}^{(l)} = f_l(\mathbf{h}^{(l-1)}) \text{ and thus } \mathbf{o} = f_L \circ \ldots \circ f_1(\mathbf{x}).$$

- If all the hidden variables and the input are vectors, then the gradient of **o** with respect to any set of parameters **W(l)**

$$\partial_{\mathbf{W}^{(l)}} \mathbf{o} = \underbrace{\partial_{\mathbf{h}^{(L-1)}} \mathbf{h}^{(L)}}_{\mathbf{M}^{(L)} \stackrel{\text{def}}{=}} \cdot \ldots \cdot \underbrace{\partial_{\mathbf{h}^{(l)}} \mathbf{h}^{(l+1)}}_{\mathbf{M}^{(l+1)} \stackrel{\text{def}}{=}} \underbrace{\partial_{\mathbf{W}^{(l)}} \mathbf{h}^{(l)}}_{\mathbf{v}^{(l)} \stackrel{\text{def}}{=}}.$$

- Gradient is the product of **L − l** matrices M(L) · . . . · M(l+1) and the gradient vector v(l) .

Vanishing and Exploding Gradients

- The matrices M(l) may have a wide variety of eigenvalues. They might be small or large, and their product might be very large or very small.
- Gradients of unpredictable magnitude also threaten the stability of the optimization algorithms.
- Parameter updates may be either

  (i) excessively large, destroying our model (the **exploding gradient** problem);

  (ii) excessively small (the **vanishing gradient** problem), rendering learning impossible as parameters hardly move on each update.

Vanishing Gradients

- Activation function sigmoid σ can cause the vanishing gradient problem.
  - The sigmoid's gradient vanishes both when its inputs are large and when they are small.
  - When backpropagating through many layers, where the inputs to many of the sigmoids are close to zero, the gradients of the overall product may vanish.
- Solution: Use ReLU for hidden lakers. ReLU is more stable.

Parameter Initialization

1. Default Initialization
   - Used a normal distribution to initialize the values of the parameters.
2. Xavier Initialization
   - samples weights from a Gaussian distribution with zero mean and variance

$$\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}.$$

   - now-standard and practically beneficial

Batch Normalization

Why Batch Normalization?

1. Standardize the input features to each have a mean of zero and variance of one. This standardization puts the parameters a priori at a similar scale. Better optimization.
2. A MLP or CNN, as we train, the variables in intermediate layers may take values with widely varying magnitudes: both along the layers from the input to the output, across units in the same layer, and over time due to our updates to the model parameters. This drift in the distribution of such variables could hamper the convergence of the network.
3. Deeper networks are complex and easily capable of overfitting. This means that regularization becomes more critical.

Batch Normalization

- Batch normalization is a popular and effective technique that consistently accelerates the convergence of deep networks.
- Batch normalization is applied to individual layers.
- It works as follows:
    - In each training iteration, first normalize the inputs (of batch normalization) by subtracting their mean and dividing by their standard deviation, where both are estimated based on the statistics of the current minibatch.
    - Next, apply a scale coefficient and a scale offset.
- Due to the normalization based on batch statistics that batch normalization derives its name.
- Batch normalization works best for moderate minibatches sizes in the 50 to 100 range.

Batch Normalization

- Denote by **x** ∈ B an input to batch normalization (BN) that is from a minibatch B, batch normalization transforms **x** as

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}.$$

- $\hat{\mu}_B$ is the sample mean and $\hat{\sigma}_B$ is the sample standard deviation of the minibatch B.
- After applying standardization, the resulting minibatch has zero mean and unit variance.

Batch Normalization

- Denote by **x** ∈ B an input to batch normalization (BN) that is from a minibatch B, batch normalization transforms **x** as

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}.$$

- $\hat{\mu}_B$ is the sample mean and $\hat{\sigma}_B$ is the sample standard deviation of the minibatch B.
- After applying standardization, the resulting minibatch has zero mean and unit variance.

Batch Normalization

- Elementwise scale parameter γ and shift parameter β that have the same shape as x. γ and β are parameters are learned jointly with the other model parameters.
- Batch normalization actively centers and rescales the inputs to each layer back to a given mean and size.
- Calculate $\hat{\mu}_B$ and $\hat{\sigma}_B$

$$\hat{\boldsymbol{\mu}}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x},$$

$$\hat{\boldsymbol{\sigma}}_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} (\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}})^2 + \epsilon.$$

Batch Normalization Layers

- Batch normalization implementations for fully-connected layers and convolutional layers are slightly different.
  - Fully-Connected Layers
    - Insert batch normalization after the affine transformation and before the nonlinear activation function.
  - Convolutional Layers
    - Apply batch normalization after the convolution and before the nonlinear activation function.
    - Carry out each batch normalization over the $m \cdot p \cdot q$ elements per output channel simultaneously.
- It operates on a full minibatch at a time.

Batch Normalization During Prediction

- After training, use the entire dataset to compute stable estimates of the variable statistics and then fix them at prediction time.

Ref TB Dive into Deep Learning

- Sections 5.4, 5.5, 5.6 and 8,5 (online version)

Next Session:
CNN

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Learning
Module # 5 : Convolutional Neural Network [CNN]

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging the authors who made their course materials freely available online.

In This Segment

1. Computer Vision Tasks
2. Image Basics
3. Image Classification Challenges
4. Convoltional Neural Network
5. Convolution
6. Putting it Together

Image Classification

64x64 x3

→ Cat? (0/1)

→ Cat? (0/1)

Classify into 3 classes

Face Detection

Object Recognition

"SIFT" & Object Recognition, David Lowe, 1999

Image is public domain

Image is CC BY-SA 2.0

Image Captioning

Andrej Karpathy, Fei-Fei Li

In This Segment

1. Computer Vision Tasks
2. Image Basics
3. Image Classification Challenges
4. Convoltional Neural Network
5. Convolution
6. Putting it Together

Digital Grayscale Image

Pixel intensities = $I(x, y)$

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

https://mozanunal.com/2019/11/img2sh/

Digital Color Image

Colour Image

This image is composed of many colors and almost all colors can be generated from the three primary colors- **Red, Green, and Blue.** We can say that each colored image is composed of these three colors or 3 channels- Red, Green, and Blue-

Colour Image Red Green Blue

https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/

Digital Color Image

https://www.mathworks.com/help/matlab/creating_plots/image-types.html

In This Segment

1. Computer Vision Tasks
2. Image Basics
3. Image Classification Challenges
4. Convoltional Neural Network
5. Convolution
6. Putting it Together

Image Classification - Challenges

Image Classification: A core task in Computer Vision

This image by Nikita is
licensed under CC-BY 2.0

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

Image Classification - Challenges

The Problem: Semantic Gap

This image by Nikita is
licensed under CC-BY 2.0

```
[[105 112 108 111 104  93 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [105  91  61  64  65  91  88  85 101 107 105  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  68  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 105 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 138 115  87]
 [ 62  65  82  80  78  71  88 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 130 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [184 146 112  80  82 120 124 104  76  48  45  66  88 101 102 100]
 [157 170 157 120  93  86 114 132 112  97  69  55  78  82  99  94]
 [150 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  88 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 110 130 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Image Classification - Challenges

Challenges: Viewpoint variation

All pixels change when the camera moves!

This image by Nikita is licensed under CC-BY 2.0

Challenges: Illumination

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Challenges: Deformation

This image by Umberto Salvagnin
is licensed under CC-BY 2.0

This image by Umberto Salvagnin
is licensed under CC-BY 2.0

This image by sare bear is
licensed under CC-BY 2.0

This image by Tom Thai is
licensed under CC-BY 2.0

Challenges: Occlusion

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

Challenges: Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Image Classification - Challenges

Challenges: Intraclass variation

This image is CC0 1.0 public domain

Image Classification - Challenges

Pixelated Images

Larger Images

- A 50x50 Image requires 2500 input dimensions.
- A HD image 1920x1080 is about 2 million inputs. (2 Mega Pixels)

Image Invariance - Challenges

Object can appear anywhere in the image.

Image A

Image B

Model predicts “cat” for both images!

Source: https://penkovsky.com/neural-networks/day5/

Image Equivariance - Challenges

Changes to the input predictably (equivalently) should not affect the output.

Source: https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a

Image Classification - Solution

In This Segment

1. Computer Vision Tasks
2. Image Basics
3. Image Classification Challenges
4. Convoltional Neural Network
5. Convolution
6. Putting it Together

Convolutional Neural Networks (CNN)

- Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
- Special kind of neural network for processing data that has a known, grid-like topology.
- Eg: Time-series data – 1D grid taking samples at regular time intervals.
- Eg: Image data – 2D grid of pixels.
- Network employs a mathematical operation called **convolution**.

Example of CNN – LeNet-5

- Handwritten Character recognition
- 10-categories classification problem

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Courtesy: Yann LeCun

Why CNN for Images

- Some patterns are much smaller than the whole image.

Courtesy: Akka Zemmari

Why CNN for Images

- Some patterns are much smaller than the whole image.

A neuron does not have to see the whole image to discover the pattern.

Courtesy: Akka Zemmari

Why CNN for Images

- Same pattern may appear in different regions.

Courtesy: Akka Zemmari

Why CNN for Images

- Sub-sampling the pixels will not change the object.

- We can sub-sample the image to make it smaller.
- Less parameters for the network to process the image.

Courtesy: Akka Zemmari

The Whole CNN

The whole CNN

Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

The Whole CNN

The Whole CNN

In This Segment

1. Computer Vision Tasks
2. Image Basics
3. Image Classification Challenges
4. Convoltional Neural Network
5. Convolution
6. Putting it Together

Convolution Layer

- Perform Convolution operation.
- In mathematics, convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other. [Wikipedia]
- The term convolution refers to both the result function and to the process of computing it. [Wikipedia]
- We will assume that the kernel is square matrix.
- We will assume that the kernel is centered on the pixel of interest. (One reason why kernel is odd matrix.)
- Convolution operation

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a,\frac{n}{2}+b}$$

1-D Convolution

- Suppose we track position of an object using a sensor or GPS device at discrete time intervals.
- Assume that the sensor is noisy.
- To obtain a less noisy estimate, we would like to take a weighted average.
- We would only sum over a small window or a weight array (filter) (w).

$$s_t = \sum_{a=0}^{k} x_{t-a} w - a$$

- Slide the filter over the input and convolve.
- The input and the kernel are one dimensional.

1-D Convolution

$$s = x_5 w_0 + x_4 w_1 + x_3 w_2 + x_2 w_3 + x_1 w_4 + x_0 w_5$$

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50
W	0.01	0.02	0.03	0.04	0.4	0.5					
S						1.80					
W		0.01	0.02	0.03	0.04	0.4	0.5				
S						1.80	1.96				
W			0.01	0.02	0.03	0.04	0.4	0.5			
S						1.80	1.96	2.11			

2D Convolution Operation

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

These values inside the matrix are to be learned by the network. We call these as parameters or weights.

1	-1	-1
-1	1	-1
-1	-1	1

Kernel / Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Kernel / Filter 2

Property 1: Each filter detects a small pattern (3×3)

Convolution Operation

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Start at top left corner.
Perform element wise product and add all the values.

Output

3

Convolution Operation

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Slide by one pixel towards right. Perform element wise product and add all the values.

Output

3 -1

Convolution Operation

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Slide by one pixel towards right. Perform element wise product and add all the values.

Output

3 -1 -3

Convolution Operation

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Slide by one pixel towards right. Perform element wise product and add all the values.

Output

3 -1 -3 -1

Convolution Operation

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Jump to next row, left most pixel. The filter has to completely superimpose the pixels of input. NOT PARTIALLY.

Output

3 -1 -3 -1

-3

Convolution Operation

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

After sliding through the entire image.

3	-1	-3	-1
-3	-1	0	-3
-3	-2	0	1
3	-2	-2	-1

Convolution Operation

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

3	-1	-3	-1
-3	-1	0	-3
-3	-2	0	1
3	-2	-2	-1

Property 2 is satisfied.

Convolution Operation

stride $s = 1$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

Feature Maps

Convolution Operation

stride $s = 3$

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input Image 6×6

Filter1

1	-1	-1
-1	1	-1
-1	-1	1

Output

3	-1
3	-1

Convolution with Striding

- Stride means skip by how many pixels.

$$\begin{aligned} \text{Input Size} &: n \times n \\ \text{Stride} &: s \\ \text{Kernel Size} &: f \times f \\ \text{Output Size} &: \left\lfloor \frac{n-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n-f}{s} + 1 \right\rfloor \end{aligned}$$

- The filter must lie completely inside the input.

Padding

- When filter is convolved with the input image
  - Image shrinks – 6×6 image became a 4×4 image.
  - Pixels in the corners are used only once, when compared the pixels in the middle. The data in the corners are thrown away.
- To resolve, apply padding to the input image before convolution.
- Padding means append zeros around the boundary of the input.

Convolution with Padding

Input Image with padding 8×8

0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0

*

Filter1

1	-1	-1
-1	1	-1
-1	-1	1

Padding

- Two kinds of padding
  - Valid Padding
    - Padding is not applied.

$$p = 0$$

    - Input size $\neq$ output size.
    - Output shrinks when compared to the input.
  - Same Padding
    - Padding is applied.

$$p = \frac{f-1}{2}$$

    - Input size $=$ output size.
    - We maintain the size of input and output as the same.

Padding

- How much to pad?

$$\text{Padding} \ : p = \frac{f-1}{2}$$
$$\text{Kernel } 3 \times 3 : p = 1$$
$$\text{Kernel } 5 \times 5 : p = 2$$
$$\text{Kernel } 7 \times 7 : p = 3$$

- Padding preserves the input size.

$$\text{Input Size} : n \times n$$
$$\text{Padding} : p$$
$$\text{Kernel Size} : f \times f$$
$$\text{Output Size} : (n + 2p - f + 1) \times (n + 2p - f + 1)$$

Convolutional Layer

Courtesy: Akka Zemmari

Convolutional Layer

Less parameters!

Even less parameters!

Courtesy: Akka Zemmari

Activations or Receptive Fields

- Effective area "seen" by a neuron
- What is the effective number of pixels seen at each layer?
  - Input image x
  - $h_1 = Conv2d(x, k = 3)$
  - $h_2 = Conv2d(h_1, k = 3)$
  - $h_3 = Conv2d(h_2, k = 3)$
  - …
  - $h_n = Conv2d(h_{n-1}, k = 3)$
- Homework: What happens if the kernel size was 5?

Local to Global

Yann LeCun's presentations on ConvNets

Parameter Size in Convolution Layer

- Input image: $B \times W_{in} \times H_{in} \times C_{in}$
  - Input data has B samples
  - Input or Image size is $W_{in} \times H_{in} \times C_{in}$
- Convolution filter
  - Weight parameters: $C_{in} \times C_{out} \times K_W \times K_H$
  - Bias parameters: C_{out}
- Output activation: $B \times W_{out} \times H_{out} \times C_{out}$
  - C_{out} filters of shape $C_{in} \times K_W \times K_H$ operate on each sample

Parameter Size in Convolution Layer

- Each filter gives one 2D output.
- K filter will give K such 2D outputs.
- The number of parameters depend on the number of kernels and the kernel size.

$$\text{Number of kernels in layer } (l-1) : K_{l-1}$$
$$\text{Kernel size} : f \times f$$
$$\text{Number of kernels in layer } (l) : K_l$$
$$\text{Number of Parameters in layer } (l) : K_{l-1} * f * f * K_l$$

Cross-Correlation vs Convolution

- Convolution in Deep Learning is actually cross-correlation.
- Cross-correlation is sliding dot product over the images.
- Cross-correlation

$$Z[i,j] = \sum_{u=-k}^{k} \sum_{-k}^{k} K[u,v] A[i+u, j+v]$$

Cross-Correlation vs Convolution

Cross-Correlation

$$Z[i,j] = \sum_{u=-k}^{k} \sum_{-k}^{k} K[u,v]A[i+u, j+v]$$

(Note the looping direction given in red.)

1) -1,-1	2) -1,0	3) -1,1
4) 0,-1	5) 0,0	6) 0,1
7) 1,-1	8) 1,0	9) 1,1

Convolution

$$Z[i,j] = \sum_{u=-k}^{k} \sum_{-k}^{k} K[u,v]A[i-u, j-v]$$

(Flipping the kernel.)

9) -1,-1	8) -1,0	7) -1,1
6) 0,-1	5) 0,0	4) 0,1
3) 1,-1	2) 1,0	1) 1,1

Cross-Correlation vs Convolution

- Convolution in Deep Learning is actually cross-correlation.
- Convolution (as in Signal Processing) has nice association property.

$$(A * B) * C = A * (B * C)$$

- In DL, we usually don't care about that (as opposed to traditional computer vision and signal processing applications.)
- Cross-correlation is easier to implement.
- Maybe the term "convolution" for cross-correlation because popular, because " Cross-correlation Neural Network" sounds weird.

Transposed Convolution

Regular Convolution:

Figure 2.1: (No padding, unit strides) Convolving a $3 * 3$ kernel over a $4 * 4$ input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

Transposed Convolution (emulated with direct convolution):

Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

Dilated Convolution

Figure 4.5: The transpose of convolving a $3 * 3$ kernel over a $5 * 5$ input using $2 * 2$ strides (i.e., $i = 5, k = 3, s = 2$ and $p = 0$). It is equivalent to convolving a $3 * 3$ kernel over a $2 * 2$ input (with 1 zero inserted between inputs) padded with a $2 * 2$ border of zeros using unit strides

Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

Remember Image Invariance Challenge

Object can appear anywhere in the image.

Image A

Image B

Model predicts “cat” for both images!

Source: https://penkovsky.com/neural-networks/day5/

Pooling (Pool) Layer

- Helps to achieve image invariance. Objects can appear anywhere in the Image and still get detected.
- Used for sub-sampling
- Reduce the size of representation
- Speed up computation

Pooling

- Apply a filter of size f and stride s. Padding is not applied.
- The output dimensions will be

$$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n-f}{s} + 1 \right\rfloor$$

- Fixed computation
- Gradient descent is not applied as there are not parameters to be learned.
- Pooling is applied on each of the channels.

Pooling Layer

- Two types
  1. Max pooling
  2. Average pooling

Max Pooling

- Take the maximum of the sub region that is considered.

Input 4×4

2	4	7	4
6	6	9	8
3	4	8	3
7	5	3	6

$\rightarrow$ Max pooling $f = 2$ $s = 2$ $\rightarrow$

Output

2×2

6	9
7	8

Average Pooling

- Compute the average of the sub region that is considered.

Input 4×4

2	4	7	4
6	6	9	8
3	4	8	3
7	5	3	6

$\rightarrow$ Average pooling

$f = 2$

$s = 2$

$\rightarrow$ Output

2×2

4.5	7
4.75	5

Convolution + Max Pooling

Input Image 6×6

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

Convolve s=1

3	-1	-3	-1
-3	-1	0	-3
-3	-2	0	1
3	-2	-2	-1
-1	-1	-1	-1
-1	-1	-2	-1
-1	-1	-2	1
-1	0	-4	3

Maxpool f=2, s=2

3	0
3	1
-1	-1
0	3

Convolution + Max Pooling

Flatten

-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

Image is Highly Non-Linear

Rectified Linear Unit (ReLU)

- Very fast
- Gradients
- Piece-wise linear approximation

ReLU in Images

Ground truth image

Standard fully-connected net

With Positional Encoding

Jon Barron's talk at the MIT Vision and Graphics Seminar 2021 on Neural Radiance Fields

Loss for Training

- Mean Squared Error
- Binary Cross Entropy
- Multi class Cross Entropy
- Class-weighted cross-entropy losses (as a way to counter imbalance)
- L1 (MAE), L2 (MSE) losses
- Focal, Dice, Huber loss
- Margin ranking losses
- Generalized Intersection-over-Union (GIoU) loss
- Connectionist Temporal Classification (CTC) loss

Regularisation and Optimization

- Drop out
- Batch Normalisation
- RMSProp, Adam or any other optimizer can be used.

Classification

- $p \in [0, 1]$ - use Sigmoid activation function
- k classes - use softmax activation function

In This Segment

1. Computer Vision Tasks
2. Image Basics
3. Image Classification Challenges
4. Convoltional Neural Network
5. Convolution
6. Putting it Together

Putting Everything Together

1. Identify the task as classification or regression. This helps in identify the output size.
2. Collect the N images. This helps in identifying input size as $W \times H \times C$.
3. Feature extraction
   - Decide on the number of Convolution layers. This can be derived by looking at how large or small pattern we are interested in.
   - Decide on the kernel size and number of kernels in each layer. Smaller patterns in images require smaller kernels. Number of kernels can be derived from the non-linearity.
   - Use ReLU activation function for better approximation.
4. Classification
   - Use Fully connected layers for classification.
   - The number of (dense) layers depends on the non-linearity.
   - The last (output) layer activation is decided based on Step 1.
5. Train and Test on the available dataset.

Convolutional Neural Network

- Input: 1 or 3 images
  - Grey scale or color
  - Will assume color to be generic

Convolutional Neural Network

- Input is convolved with a set of K_1 filters or kernels.
  - Typically, K_1 is a power of 2 e.g 2,4,8,16,32,...
  - Filters are typically 5x5x3, 3x3x3 or 1x1x3 (better notation).
  - Typical stride is 1 or 2.

Convolutional Neural Network

- Input is convolved with a set of K_1 filters or kernels.
  - Typically, K_1 is a power of 2 e.g 2,4,8,16,32,...
  - Filters are typically 5x5, 3x3 or 1x1.
  - Filters should be small enough to capture fine features, particularly important for scaled down images.

K_1 total filters
Filter size: $L \times L \times 3$

$I \times I$ *image*

Parameters to choose: K_1, L and S
1. Number of filters K_1
2. Size of filters $L \times L \times 3 + bias$
3. *Stride* of convolution S

Total number of parameters: $K_1(3L^2 + 1)$

Convolutional Neural Network

- First Convolutional layer:
  - Several convolutional filers, which are 3D.
  - Convolution is followed typically by a ReLU activation.
  - Each filter creates a singly 2D feature map.

The layer includes a convolution operation followed by an activation (typically RELU)

Convolutional Neural Network

- First Pooling layer:
  - For each $P \times P$ block of each map, pool down to a single value.

Convolutional Neural Network

- Better drawing

Convolutional Neural Network

- Second Convolutional layer" K_2 3D filters resulting in K_2 feature maps.

Convolutional Neural Network

- Second Convolutional layer

Parameters to choose: K_3, L_3 and S_3

1. Number of filters K_3
2. Size of filters $L_3 \times L_3 \times K_2 + bias$
3. *Stride* of convolution S_3

Total number of parameters: $K_3(K_2 L_3^2 + 1)$
All these parameters must be learned

Convolutional Neural Network

- Second Pooling layer: K_2 pooling operations results in K_2 reduced 2D maps.

Convolutional Neural Network

- Second Pooling layer

Convolutional Neural Network

- Fully connected layer: This continues for several layers until the final convolved feature maps are fed to MLP.

Convolutional Neural Network

Size of Layers

- Each convolution layer with stride 1 typically maintains the size of the image.
  - With appropriate zero padding.
  - If performed without zero padding it will decrease the size of the input.
- Each convolution layer will generally increase the number of maps from the previous layer.
  - Increasing layers reduces the amount of information lost by subsequent downsampling.
- Each pooling layer with stride D decreases the size of the maps by a factor of D.
- Filters within a layer must all be the same size, but sizes may vary with layer. Similarly for pooling D may vary with layer.
- In general the number of convolutional filters increases with layers.

Convolutional Neural Network

Parameters to choose (design choices)

- Number of convolutional and downsampling layers
  - And arrangement (order in which they follow one another)
- For each convolution layer:
  - Number of filters K_i
  - Spatial extent of filter $L_i \times L_i$
    - ★ The "depth" of the filter is fixed by the number of filters in the previous layer K_{i-1}.
  - The stride S_i
- For each downsampling/pooling layer:
  - Spatial extent of filter $P_i \times P_i$
  - The stride D_i
- For the final MLP:
  - Number of layers
  - Number of neurons in each layer

CNN-Recap

- Neural network with specialized connectivity structure
- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Supervised training
- Train convolutional filters by back-propagating error
- Convolution over time

References

1. Deep Learning with Python by Francois Chollet.
   https://livebook.manning.com/book/deep-learning-with-python/
2. Deep Learning by Ian Goodfellow, YoshuaBengio, Aaron Courville
   https://www.deeplearningbook.org/

Thank You!

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Learning
Module # 5 : Convolutional Neural Network [CNN]

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging the authors who made their course materials freely available online.

In This Segment

1. CNN Architectures
2. Transfer Learning
3. Further Reading

LeNet 5

- Every convolutional layer includes three parts: convolution, pooling, and nonlinear activation functions.
- Using convolution to extract spatial features.
- Conv filters were 5x5, applied at stride 1.
- Subsampling average pooling layer. Subsampling (Pooling) layers were 2x2 applied at stride 2.
- tanh activation function.
- Using MLP as the last classifier.
- Architecture is [CONV-POOL-CONV-POOL-FC-FC]
- Sparse connection between layers to reduce the complexity of computation.

LeNet-5

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

ImageNet Large Scale Visual Recognition

ImageNet Data

Figure 3: 96 convolutional kernels of size 11*11*3 learned by the first convolutional layer on the 224*224*3 input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU

Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

ImageNet Data

The ImageNet set that was used has ~1.2 million images and 1000 classes

Accuracy is measured as top-5 performance: Correct prediction if the true label matches one of the top 5 predictions of the model

AlexNet

- AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012.[3] The network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up.
- Details/Retrospectives:
  - first use of ReLU
  - used Local Response Normalisation (LRN) layers (not common anymore)
  - Approx. 60 million parameters are learned.
  - heavy data augmentation
  - dropout 0.5
  - batch size 128
  - SGD Momentum 0.9
  - Learning rate $1e-2$, reduced by 10 manually when val accuracy plateaus
  - L2 weight decay $5e-4$
  - 7 CNN ensemble: $18.2\% -> 15.4\%$

AlexNet Architecture

227x227x3 INPUT
55x55x96 CONV1 : 96 11x11 filters at stride 4, pad 0
27x27x96 MAX POOL1 : 3x3 filters at stride 2
27x27x96 NORM1 : Normalization layer
27x27x256 CONV2 : 256 5x5 filters at stride 1, pad 2
13x13x256 MAX POOL2 : 3x3 filters at stride 2
13x13x256 NORM2 : Normalization layer
13x13x384 CONV3 : 384 3x3 filters at stride 1, pad 1
13x13x384 CONV4 : 384 3x3 filters at stride 1, pad 1
13x13x256 CONV5 : 256 3x3 filters at stride 1, pad 1
6x6x256 MAX POOL3 : 3x3 filters at stride 2
4096 FC6 : 4096 neurons
4096 FC7 : 4096 neurons
1000 FC8 : 1000 neurons (class scores)

AlexNet

[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

Andrew Ng

VGG 16

- VGG stands for Visual Geometry Group with 16 Layers
- Plug and play in Caffe
- Deeper the better
- Details
  - ILSVRC'14 2nd in classification, 1st in localization
  - Similar training procedure as Krizhevsky 2012
  - Approx. 138 million parameters are learned.
  - No Local Response Normalisation (LRN)
  - Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
  - Use ensembles for best results
  - All convolutions with a 3×3 kernel
  - All max-pooling layers with a 2×2 kernel
  - FC7 features generalize well to other tasks

VGG 16

[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

Andrew Ng

Network in Network

- A convolution kernel can be thought of as a generalized linear model (GLM).
- Using a sophisticated non-linear function approximator like MLP, enhance the abstraction ability of the local model.
- Replace GLM by **micro network**.
- Replace the MLP micro structure via convolutions.
- Replace fully connected layers in the last layers by global average pooling.

Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

Network in Network

Using micro-networks allow to extract more sophisticated features and may need fewer extractors and can avoid learning too simple or redundant features.

(a) Linear convolution layer (b) Mlpconv layer

Figure 1: Comparison of linear convolution layer and mlpconv layer. The linear convolution layer includes a linear filter while the mlpconv layer includes a micro network (we choose the multilayer perceptron in this paper). Both layers map the local receptive field to a confidence value of the latent concept.

Network in Network

Fully connected layers have a lot of learnable parameters and may cause overfiting. Replace FC by global average pooling for better generalization.

Figure 16: Global average pooling layer replacing the fully connected layers. The output layer implements a Softmax operation with $p_1, p_2, \ldots, p_n$ the predicted probabilities for each class.

Figure Source: Singh, Anshuman Vikram. "Content-based image retrieval using deep learning." (2015)

GoogLeNet / Inception Net

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers with weights
- Only 5 million parameters (12x less than AlexNet and 27x less than VGG-16)
- Inception Module - convolutional "blocks" – efficient
  - Design a good local network topology (network within a network) and then stack these modules on top of each other.
- Linear layers at the end
- Max pooling in between, multiple Conv layers between pooling
- Great ideas for data augmentation
- Deeper networks, with computational efficiency
- No FC layers.
- After the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer.

Inception Net

Inception Net Module

Inception Net Module

- Apply parallel filter operations on the input from previous layer.
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5) and Pooling operation (3x3).
- Concatenate all filter outputs together channel-wise.
- Very expensive compute
- 1x1 convolutions to reduce feature channel size.
  - Assume an input of WxHx64.
  - Each filter has size 1x1x64, and performs a 64-dimensional dot product.
  - Preserves spatial dimensions, reduces depth.
  - Projects depth to lower dimension (combination of feature maps).

Residual Net (ResNet)

- Very deep networks using residual connections
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15.
- Add residual connections between convolution blocks
- Easy gradient flow for deep models
- Enable very deep models (from 20 layers to 150 and even 1000 layers!)
- No linear layers except classifier.
- Solve the problems of vanishing gradients using Residual connections.

ResNet Block

- What happens when we continue stacking deeper layers on a convolutional neural network?
- The deeper model performs worse, but it's not caused by overfitting.

ResNet Block

- Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping.

Identity mapping:
$H(x) = x$ if $F(x) = 0$

Use layers to fit **residual** $F(x) = H(x) - x$ instead of $H(x)$ directly

ResNet

relu
F(x) + x
3x3 conv
relu
3x3 conv
X
Residual block
X
identity
Softmax
FC 1000
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
Pool
7x7 conv, 64, /2
Input

ResNet Modules

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet

[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

ResNet architecture

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2. Reduce the activation volume by half.
- Additional conv layer at the beginning (stem).
- No FC layers at the end (only FC 1000 to output classes)
- Total depths of 18, 34, 50, 101, or 152 layers for ImageNet.
- Training ResNet in practice:
  - Batch Normalization after every CONV layer
  - Xavier initialization from He et al.
  - SGD + Momentum (0.9)
  - Learning rate: 0.1, divided by 10 when validation error plateaus
  - Mini-batch size 256
  - Weight decay of 1e-5
  - No dropout used

Efficacy of ResNet

Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

https://arxiv.org/abs/1512.03385

Comparing complexity

Alfredo Canziani, Adam Paszke, Eugenio Culurciello, An Analysis of Deep Neural Network Models for Practical Applications, 2017

Comparing complexity

- Inception-v4: Resnet + Inception – more accuracy
- VGG: most parameters, most operations
- GoogLeNet: most efficient
- AlexNet: Smaller compute, still memory heavy, lower accuracy
- ResNet: Moderate efficiency depending on model, highest accuracy

In This Segment

1. CNN Architectures
2. Transfer Learning
3. Further Reading

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning with CNNs

1. Train on Imagenet

Transfer Learning

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning with CNNs

Transfer Learning

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning with CNNs

Transfer Learning for Image Data

- Use a deep learning model that is pre-trained on large dataset like ImageNet or MS Coco.
- Oxford VGG Model
- Google Inception Model
- Microsoft ResNet Model

Transfer Learning for Text Data

- Embedding is the mapping of words to a high-dimensional continuous vector space where different words with similar meanings have similar vector representations.
- Google's word2vec Model
- Stanford's GloVe Model
- FastText
- Gensim

Transfer Learning - when to use?

- You need a lot of a data if you want to train/use CNNs / RNNs.
- Task A and Task B have the same type of input. Eg: Input is images for both tasks.
- We have lot of data for training Task A and relatively low data for training in Task B.
- Low level features obtained from Task A could be more helpful for learning Task B.

Transfer Learning Example

Object Detection
(Fast R-CNN)

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

CNN pretrained on ImageNet

Image Captioning: CNN + RNN

Word vectors pretrained with word2vec

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

In This Segment

1. CNN Architectures
2. Transfer Learning
3. Further Reading

Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways ("cardinality")
- Parallel pathways similar in spirit to Inception module

Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet

MobileNets: Efficient Convolutional Neural Networks for Mobile Applications *[Howard et al. 2017]*

- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- ShuffleNet: Zhang et al, CVPR 2018

EfficientNet: Smart Compound Scaling

[Tan and Le. 2019]

- Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.
- Search for optimal set of compound scaling factors given a compute budget (target memory & flops).
- Scale up using smart heuristic rules

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

References

Refer Chapter 8 of Dive into Deep Learning book.

LeNet5 : Yann Lecun and Leon Bottou and Yoshua Bengio and Patrick Haffner, Gradient-based learning applied to document recognition, IEEE-1998

AlexNet : Imagenet classification with deep convolutional neural networks, Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey E, In: Advances in neural information processing systems, NIPS 2012

VGG16 : Simonyan, Karen, and Andrew Zisserman. ”Very deep convolutional networks for large-scale image recognition.”

Inception v1 : Going deeper with convolutions, Szegedy, Christian et.al CVPR 2015

ResNet : Deep residual learning for image recognition, He et.al CVPR-2016

Thank You!

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Learning
Module # 6 : Recurrent Neural Network [RNN]

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging the authors who made their course materials freely available online.

Issues of Feedforward Neural Networks

In feedforward and convolutional neural networks

- The size of the input is always fixed.
- Each input to the network is independent of the previous or future inputs.
- The computations, outputs and decisions for two successive inputs / images are completely independent of each other.

Issues of Feedforward Neural Networks

This is not true in many applications.

- The size of the input is not always fixed.
- Successive inputs may not be independent of each other.
- Each network (blue - orange - green structure) is performing the same task – input : character output : character.

Example: Auto-completion.

In This Segment

1. Sequence Learning
2. Recurrent Neural Network (RNN)
3. Types of RNN
4. Learning in RNN
5. Issues in RNN
6. Long Short Term Memory Unit (LSTM)
7. Gated Recurrent Unit (GRU)
8. Bidirectional RNN

Sequence Learning Problems

To model a sequence we need

- Process an input or sequence of inputs.
- The inputs may have be dependent.
- We may have to maintain the sequence order.
- Each input corresponds to one time step.
- Keep track of long term dependencies.
- Produce an output or sequence of outputs.
- Supervised Learning.
- Share parameters across the sequences.

Sequence Model

Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation	∅	→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

Courtesy: Andrew Ng

Part of Speech Tagging

- Task is predicting the part of speech tag (noun, adverb, adjective, verb) of each word in a sentence.
- When we see an adjective we are almost sure, the next word should be a noun.
- The current output depends on the current input as well as the previous input.
- The size of the input is not fixed. Sentences have any number of words.
- An output is produced at end of each time step.
- Each network is performing the same task – input : word, output : tag.

Sentiment Analysis

- Task is predicting the sentiment of a whole sentence.
- Input is the entire sequence of inputs.
- An output is **not** produced at end of each time step.
- Each network is performing the same task – input : word, output : polarity $+/-$.

In This Segment

1. Sequence Learning
2. Recurrent Neural Network (RNN)
3. Types of RNN
4. Learning in RNN
5. Issues in RNN
6. Long Short Term Memory Unit (LSTM)
7. Gated Recurrent Unit (GRU)
8. Bidirectional RNN

Recurrent Neural Network (RNN)

Andrew Ng

- Accounts for variable number of inputs.
- Accounts for dependencies between inputs.
- Accounts for variable number of outputs.
- Ensures that the same function executed at each time step.
- The features learned across the inputs at different time step has to be shared.

RNN I

- The function learned at each time step.

$$
\begin{aligned}
t &= \text{ time step} \\
x_t &= \text{ input at time step } t \\
s_t &= \sigma(Ux_t + b) \\
y_t &= g(Vs_t + c)
\end{aligned}
$$

- Since the same function has to be executed at each time step we should share the same network i.e., same parameters at each time step.

RNN II - Unrolled Network

- The parameter sharing ensures that
  - the network becomes invariant to the length of the input.
  - the number of time steps doesn't matter.
- Create multiple copies of the network and execute them at each timestep.
  - i.e. create a loop effect.
  - i.e. add recurrent connection in the network.

RNN III - Rolled Network

- Parameter sharing across time steps
- Self loop or recurrent connection

Schematic of RNN

In This Segment

1. Sequence Learning
2. Recurrent Neural Network (RNN)
3. Types of RNN
4. Learning in RNN
5. Issues in RNN
6. Long Short Term Memory Unit (LSTM)
7. Gated Recurrent Unit (GRU)
8. Bidirectional RNN

Types of RNN

Courtesy: Andrej Karpathy

Types of RNN

Courtesy: Andrej Karpathy

Types of RNN and Applications

- One to one – Generic neural network, Image classification
- One to many – Music generation, Image Captioning
- Many to one – Movie review or Sentiment Analysis
- Many to many – Machine translation
- Synced Many to many – Video classification

In This Segment

1. Sequence Learning
2. Recurrent Neural Network (RNN)
3. Types of RNN
4. Learning in RNN
5. Issues in RNN
6. Long Short Term Memory Unit (LSTM)
7. Gated Recurrent Unit (GRU)
8. Bidirectional RNN

Forward Propagation in RNN

- s_t is the **state** of the network at time step t.

$$s_0 = 0$$
$$s_t = \sigma(Ux_t + Ws_{t-1} + b)$$
$$\hat{y}_t = g(Vs_t + c)$$
or
$$\hat{y}_t = f(x_t, s_{t-1}, W, U, V, b, c)$$

- The parameters W, U, V, b, c are shared across time steps.

Back Propagation in RNN

Loss function

$$\mathcal{L}_t(\hat{y}_t, y_t) = \prod_{t=1}^{T_y} P(\hat{y}_t \mid \hat{y}_{t-1}, \ldots, \hat{y}_1)$$

Overall Loss

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}_t(\hat{y}_t, y_t)$$

Back Propagation Through Time (BPTT) in RNN

In This Segment

1. Sequence Learning
2. Recurrent Neural Network (RNN)
3. Types of RNN
4. Learning in RNN
5. Issues in RNN
6. Long Short Term Memory Unit (LSTM)
7. Gated Recurrent Unit (GRU)
8. Bidirectional RNN

Issue of Maintaining States

- The old information gets morphed by the current input at each new time step.
- After t steps the information stored at time step $t - k$ (for some $k < t$) gets completely morphed so much that it would be impossible to extract the original information stored at time step $t - k$.
- It is very hard to assign the responsibility of the error caused at time step t to the events that occurred at time step $t - k$.
- Basically depends on the size of memory that is available.

Strategy to Maintain States

- Selectively write on the states.
- Selectively read the already written content.
- Selectively forget (erase) some content.

Sentiment Analysis

Review: The first half of the movie was dry but the second half really picked up pace. The lead actor delivered an amazing performance

- RNN reads the document from left to right and after every word updates the state.
- By the time we reach the end of the document the information obtained from the first few words is completely lost.
- Ideally we want to
  - forget the information added by stop words (a, the, etc.).
  - selectively read the information added by previous sentiment bearing words (awesome, amazing, etc.)
  - selectively write new information from the current word to the state.

Selective Write

- Recall that in RNNs we use s_{t-1} to compute s_t.

$$s_t = \sigma(Ws_{t-1} + Ux_t + b)$$

Selective Write

- Introduce a vector o_{t-1} which decides what fraction of each element of s_{t-1} should be passed to the next state.
- Each element of o_{t-1} gets multiplied with the corresponding element of s_{t-1}.
- Each element of o_{t-1} is restricted to be between 0 and 1.
- The RNN has to learn o_{t-1} along with the other parameters (W, U, V).

Selective Write

- Compute o_{t-1} and h_{t-1} as

$$o_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$$
$$h_{t-1} = o_{t-1} \odot \sigma(s_{t-1})$$

- The parameters (W_o, U_o, b_o) are learned along with the existing parameters (W, U, V).
- The sigmoid function ensures that the values are between 0 and 1.
- o_t is called the **output gate** as it decides how much to pass (write) to the next time step.

Compute State

- h_{t-1} and x_t are used to compute the new state at the next time step.

$$\tilde{s}_t = \sigma(Wh_{t-1} + Ux_t + b)$$

Selective Read

- $\tilde{s}_t$ captures all the information from the previous state h_{t-1} and the current input x_t.
- To do selective read, introduce another gate called the **input gate**.

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$\text{Selectively Read } = i_t \odot \tilde{s}_t$$

Selective Read

- $\tilde{s}_t$ captures all the information from the previous state h_{t-1} and the current input x_t.
- To do selective read, introduce another gate called the **input gate**.

$$s_t = s_{t-1} + i_t \odot \tilde{s}_t$$

Selective Forget

- To do selective forget, introduce another gate called the **forget gate**.

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

Full LSTM

- 3 gates

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

- 3 states

$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$
$$h_t = o_t \odot \sigma(s_t)$$
$$\hat{y}_t = g(V s_t + c)$$

In This Segment

1. Sequence Learning
2. Recurrent Neural Network (RNN)
3. Types of RNN
4. Learning in RNN
5. Issues in RNN
6. Long Short Term Memory Unit (LSTM)
7. Gated Recurrent Unit (GRU)
8. Bidirectional RNN

Long Short Term Memory Unit (LSTM)

- Another representation
- 3 gates are used – Update gate Γ_u, Forget gate Γ_f and Output gate Γ_o.

$$
\begin{aligned}
\tilde{c}^{<t>} &= \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\
\Gamma_u &= \sigma(W_u[a^{<t-1>}, x^{<t>} + b_u]) \\
\Gamma_f &= \sigma(W_f[a^{<t-1>}, x^{<t>} + b_f]) \\
\Gamma_o &= \sigma(W_o[a^{<t-1>}, x^{<t>} + b_o]) \\
c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\
a^{<t>} &= \Gamma_o * \tanh(c^{<t>})
\end{aligned}
$$

LSTM

c^{t-1}
c^t
tanh
f^t
u^t
$\tilde{c}^t$
o^t
$a^{<t>}$
softmax
$a^{<t-1>}$
Forget
Update
tanh
Output
$y^{<t>}$
$x^{<t>}$

LSTM

c^{t-1}
c^t
tanh
f^t
u^t
$\tilde{c}^t$
o^t
$a^{<t>}$
softmax
$a^{<t-1>}$
Forget
Update
tanh
Output
$y^{<t>}$
$x^{<t>}$

c^{t-1}
c^t
tanh
f^t
u^t
$\tilde{c}^t$
o^t
$a^{<t>}$
softmax
$a^{<t-1>}$
Forget
Update
tanh
Output
$y^{<t>}$
$x^{<t>}$

In This Segment

1. Sequence Learning
2. Recurrent Neural Network (RNN)
3. Types of RNN
4. Learning in RNN
5. Issues in RNN
6. Long Short Term Memory Unit (LSTM)
7. Gated Recurrent Unit (GRU)
8. Bidirectional RNN

Gated Recurrent Unit (GRU)

- Introduce a memory cell $c^{<t>} = a^{<t>}$.
- Candidate for replacing $c^{<t>}$ is given as $\tilde{c}^{<t>}$.
- The decision whether to update $c^{<t>}$ with $\tilde{c}^{<t>}$ is given by the **update gate** Γ_u. Γ_u takes the value of 0 or 1.

$$
\begin{aligned}
\tilde{c}^{<t>} &= \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c) \\
\Gamma_u &= \sigma(W_u[c^{<t-1>}, x^{<t>} + b_u]) \\
\Gamma_r &= \sigma(W_r[c^{<t-1>}, x^{<t>} + b_r]) \\
c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>} \\
a^{<t>} &= c^{<t>}
\end{aligned}
$$

Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU)

In This Segment

1. Sequence Learning
2. Recurrent Neural Network (RNN)
3. Types of RNN
4. Learning in RNN
5. Issues in RNN
6. Long Short Term Memory Unit (LSTM)
7. Gated Recurrent Unit (GRU)
8. Bidirectional RNN

Bidirectional RNN (BRNN)

- Forward and backward connections.
- The blocks can be RNN, GRU, LSTM.
- Mostly used in the NLP.
- Acyclic graph
- Example: Name entity recognition
  He said “Teddy bear is soft.”
  He said “Teddy Roosevelt was a President.”

BRNN ARCHITECTURE

$$\hat{y}^{<t>} = g(W_y[\overrightarrow{a}^{<t>}\overleftarrow{a}^{<t>}x^{<t>}] + b_y)$$

Summary

- Use GRU, when dependency is short. Eg: Weather forecasting
- Use LSTM, when dependency is long. Eg: NLP Translation
- Use BRNN, dependency is in both direction. Eg: Stock prediction

References

1. Dive into Deep Learning by Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola
   `https://d2l.ai/chapter_introduction/index.html`
2. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
   `https://www.deeplearningbook.org/`

Thank You!

Attention Models & Transformers

S. P. Vimal, CSIS Department (WILP Division)
vimalsp@pilani.bits-pilani.ac.in

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Topics

- ★ Brief review of Sequence Models (RNN, LSTMS)
- ★ Encoder - Decoder Models
- ★ Attention
- ★ Transformers
- ★ Vision Transformers

References and Recommendation for Readings: Draft Chapter 9 and Draft Chapter 10 , Speech and Language Processing. Daniel Jurafsky & James H. Martin.

RNN Architectures for NLP Tasks

a) sequence labeling

Ex: POS Tagging, Named Entity Tagging

b) sequence classification

Ex: Sentiment Analysis

c) language modeling

Ex: Predict Next Word

d) encoder-decoder

Ex: Language Translation

Encoder - Decoder

- **Goal:** Develop an architecture capable of generating *contextually appropriate*, *arbitrary length*, output sequences

- **Applications**:
  - Machine translation
  - Summarization
  - Question answering
  - Dialogue modeling.

Encoder - Decoder

Encoder:

- Input sequence ($x_{1...n}$) $\longrightarrow$ Sequence of contextualized representations, ($h_{1...n}$)
- Ex: LSTM, CNN, Transformers etc.

Context:

- c, a function of ($h_{1...n}$)

Decoder:

- c $\longrightarrow$ arbitrary length sequence of hidden states ($h_{1...m}$) $\longrightarrow$ sequence of output states ($y_{1...m}$)

Encoder - Decoder

Encoder - Decoder for Language Translation

Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

Encoder - Decoder for Language Translation

<u>Weakness of this approach :</u> Influence of the context vector, c, will wane as the output sequence is generated.

<u>Solution:</u> Make the context vector c available at each step in the decoding process by adding it as a parameter to the computation of the current hidden state

Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

Encoder - Decoder for Language Translation

<u>Weakness of this approach :</u> Influence of the context vector, c, will wane as the output sequence is generated.

<u>Solution:</u> Make the context vector c available at each step in the decoding process by adding it as a parameter to the computation of the current hidden state

Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

Encoder - Decoder for Language Translation

Encoder - Decoder

$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}
$$

Encoder - Decoder for Language Translation

Encoder - Decoder

Training

Encoder - Decoder

Teaching Forcing

- Force the system to use the gold target token from training as the next input x_{t+1}, rather than allowing it to rely on the (possibly erroneous) decoder output ˆyt .
- Speeds up training

Attention !

In an encoder-decoder arch, the final hidden state acts as a bottleneck:

- It must represent absolutely everything about the meaning of the source text
- The only thing the decoder knows about the source text is what's in this context vector

Attention !

Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

Attention !

Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

With attention, decoder to sees a different, dynamic, context, which is a function of all the encoder hidden states

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

Attention !

Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

With attention, decoder to sees a different, dynamic, context, which is a function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

With attention, decoder gets information from all the hidden states of the encoder, not just the last hidden state of the encoder

Each context vector is obtained by taking a weighted sum of all the encoder hidden states.

The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing

Attention !

Step -1 : Find out how relevant each encoder state is to the present decoder state $\mathbf{h}_{i-1}^{d}$

Compute a score of similarity between $\mathbf{h}_{i-1}^{d}$ and all the encoder states : $score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e})$

Dot Product Attention : $\text{score}(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}) = \mathbf{h}_{i-1}^{d} \cdot \mathbf{h}_{j}^{e}$

Step -2 : Normalize all the scores with softmax to create a vector of weights, $\alpha_{i,j}$

$\alpha_{i,j}$ indicates the proportional relevance of each encoder hidden state j to the prior hidden decoder state, $\mathbf{h}_{i-1}^{d}$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}) \ \forall j \in e)$$

$$= \frac{\exp(\text{score}(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e})}{\sum_{k} \exp(\text{score}(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{k}^{e}))}$$

Attention !

Step -3 : **Given the distribution in α, compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states**

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

Attention !

Step -3 : Given the distribution in α, compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

Plus : In step-1, we can get a more powerful scoring function by parameterizing the score with its own set of weights, W_s:

$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{t-1}^d \mathbf{W}_s \mathbf{h}_j^e$$

W_s , is trained during normal end-to-end training,
W_s , gives the network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

Attention !

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

Transformers

- 2017, NIPS, Vaswani et. al., **Attention Is All You Need** !!!
- Transformers **map sequences** of input vectors $(x_1,\ldots,x_n)$ to sequences of output vectors $(y_1,\ldots,y_n)$ **of the same length**
- Made up of **transformer blocks** in which the key component is **self-attention** layers

  [Self-attention allows a network to directly extract and use information from arbitrarily large contexts directly !!!]

- Transformers are **not based on recurrent connections** ⇒ Parallel implementations possible ⇒ Efficient to scale (comparing LSTM)

Self-Attention | Transformers

- Attention ⇒ Ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context.
- Self-attention ⇒

  > Set of ***comparisons*** are to other elements ***within a given sequence***

  > Use these comparisons to compute an output for the current input

Self-Attention | Transformers

In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one.

Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Self-Attention | Transformers

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \ \forall j \leq i$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

Self-Attention | Transformers

- Let us understand how transformers uses self-attention !

Self-Attention | Transformers

$$\mathbf{W}^V, \mathbf{W}^K, \mathbf{W}^Q \in \mathbb{R}^{d \times d}$$

In Vaswani et al., 2017, d was 1024.

- Let us understand how transformers uses self-attention !

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

Query, Q	Key, K	Value, V
As the current focus of attention when being compared to all of the other preceding inputs.	In its role as a preceding input being compared to the current focus of attention.	As a value used to compute the output for the current focus of attention

Three different roles each x_i (input embedding) , in the computation of self attention

Self-Attention | Transformers

$$\mathbf{W}^V, \mathbf{W}^K, \mathbf{W}^Q \in \mathbb{R}^{d \times d}$$

In Vaswani et al., 2017, d was 1024.

- Let us understand how transformers uses self-attention !

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

The simple dot product can be an arbitrarily large; scaled dot-product is used in transformers;

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \ \forall j \leq i \end{aligned}$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

Self-Attention | Transformers

- Each output, y_i , is computed independently
- Entire process can be parallelized

Calculating the value of y3, the third element of a sequence using causal (left-to-right) self-attention

Self-Attention | Transformers

- Pack the input embeddings of the N input tokens into a single matrix

$$\mathbf{X} \in \mathbb{R}^{N \times d}$$

> Each row of X is the embedding of one token of the input

- Multiply X by the key, query, and value (dxd) matrices

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \;\; \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \;\; \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

Self-Attention | Transformers

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}\right)\mathbf{V}$$

$\mathbf{QK}^\intercal$ Matrix

N

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

Note: Upper-triangle portion of the comparisons matrix zeroed out (set to −∞, which the softmax will turn to zero)

Transformer Blocks | Transformers

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x}))$$
$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z}))$$

Transformer Blocks | Transformers

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x}))$$
$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z}))$$

LayerNorm:

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i - \mu)^2}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

$$\text{LayerNorm} = \gamma\,\hat{\mathbf{x}} + \beta$$

Multihead-Attention | Transformers

- Different words in a sentence can relate to each other in many different ways simultaneously

  >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.

- Multihead self-attention layers

  >> Heads ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.

  >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

Multihead-Attention | Transformers

- Different words in a sentence can relate to each other in many different ways simultaneously

  >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.

- Multihead self-attention layers

  >> Heads ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.

  >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

$$\text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q \; ; \; \mathbf{K} = \mathbf{X}\mathbf{W}_i^K \; ; \; \mathbf{V} = \mathbf{X}\mathbf{W}_i^V$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

Multihead-Attention | Transformers

Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected down to d, thus producing an output of the same size as the input so layers can be stacked.

Positional Embeddings | Transformers

A simple way to model position:
Add an embedding representation of the absolute position to the input word embedding to produce a new embedding of the same dimensionality.

A combination of sine and cosine functions with differing frequencies was used in the original transformer work.

Transformers as Language Models

$$= \frac{1}{T}\sum_{t=1}^{T} L_{CE}$$

Vision Transformers (ViT)

- Ref: Alexey et. al.(2021) , ICLR, An Image Is Worth 16x16 Words: Transformers For Image Recognition At Scale
-

Readings

1. Draft Chapter 9 and Draft Chapter 10 , Speech and Language Processing. Daniel Jurafsky & James H. Martin.
2. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
3. Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

Thank you

Neural Network Search

(Learning how to learn)

Mounika Marreddy

What is Deep Learning

Deep Learning

- Explosion of interest since 2012
- Very powerful machine learning technique
- Huge variety of neural networks for different tasks
- Key ingredients took years to develop
- Algorithms are getting increasingly more specialized and complicated

Machine and Deep Learning: a story of automation

```
from torchvision.models import resnet50, ResNet50_Weights


# Best available weights (currently alias for IMAGENET1K_V2)
# Note that these weights may change across versions
resnet50(weights=ResNet50_Weights.DEFAULT)
```

Deep Learning (cont'd)

- Algorithms are getting increasingly more specialized and complicated
- GoogLeNet (2014)

Neural Architectures

- Search models are replaced by human designed models

Definition

- Neural network search (NAS) is a technique for automating the design of artificial neural networks (ANNs) for a given dataset.
- NAS automates the process of designing ANNs by searching through a space of possible architectures.
- The search space is defined by the set of possible nodes, connections, and layers that can be used to construct an ANN.
- The NAS algorithm then explores the search space to find an architecture that performs well on a given task

NAS: Growing Field!

NAS can be applied for:

Spherical	Omnidirectional Vision
NinaPro DB5	Prosthetics Control
FSD50K	Audio Classification
Darcy Flow	PDE Solver
PSICOV	Protein Folding
Cosmic	Astronomy Imaging
ECG	Medical Diagnostics
Satellite	Earth Monitoring
DeepSEA	Genetic Prediction

Graph neural networks . . . . .

Generative adversarial network

Dense prediction tasks . . . . .

Adversarial robustness . . . . .

Self-supervised learning for NAS

Basic Math Definition

- Define a search space A,

$$\min_{a \in \mathcal{A}} \quad \mathcal{L}_{\text{val}}\left(w^*(a), a\right)$$

$$\text{s.t.} \quad w^*(a) = \text{argmin}_w \quad \mathcal{L}_{\text{train}}\left(w, a\right)$$

Three Pillars of NAS

- Search Space
- Search Strategy
- Performance Estimation Strategy

Search Space

- The search space defines which architectures can be represented in principle.
- Incorporating prior knowledge about typical properties of architectures well-suited for a task can reduce the size of the search space and simplify the search.
- However, this also introduces a human bias, which may prevent finding novel architectural building blocks that go beyond the current human knowledge.

Search Strategy

- The search strategy details how to explore the search space (which is often exponentially large or even unbounded).
- It encompasses the classical exploration-exploitation trade-off since, on the one hand, it is desirable to find well-performing architectures quickly, while on the other hand, premature convergence to a region of suboptimal architectures should be avoided.

Performance Estimation

- The objective of NAS is typically to find architectures that achieve high predictive performance on unseen data.
- Performance Estimation refers to the process of estimating this performance: the simplest option is to perform a standard training and validation of the architecture on data, but this is unfortunately computationally expensive and limits the number of architectures that can be explored.
- Much recent research therefore focuses on developing methods that reduce the cost of these performance estimations.

Search Space

- The search space defines which neural architectures a NAS approach might discover in principle.
- Some of the recent search spaces are:
  - Chain-structured neural networks
  - Multi-branch networks
  - Cell or block search space

Chain structured and Complex Search Spaces

Advantages of cell or block search

- The size of the search space is drastically reduced since cells usually consist of significantly less layers than whole architectures.
  - For example, Zoph et al. (2018) estimate a seven-times speed-up compared to their previous work (Zoph and Le, 2017) while achieving better performance.
- Architectures built from cells can more easily be transferred or adapted to other data sets by simply varying the number of cells and filters used within a model.
  - Indeed, Zoph et al. (2018) transfer cells optimized on CIFAR-10 to ImageNet and achieve state-of-the-art performance.
- Creating architectures by repeating building blocks has proven a useful design principle in general, such as repeating an LSTM block in RNNs or stacking a residual block.

Search Spaces by Stacking the Cells

Search Strategy

- Many different search strategies can be used to explore the space of neural architectures:
  - Random search
  - Bayesian optimization
  - Evolutionary methods
  - Reinforcement learning (RL), and
  - Gradient-based methods

Search Strategy - Types

- Some of the most common algorithms are:
  - Reinforcement learning-based NAS: In reinforcement learning, an agent learns to perform a task by trial and error. The agent receives rewards for taking actions that lead to successful outcomes, and punishments for taking actions that lead to unsuccessful outcomes.
  - Bayesian optimization-based NAS: Bayesian optimization is a technique for finding the best value of a function by iteratively querying the function and using the results of the queries to update the estimate of the function's true value.
  - Genetic algorithm-based NAS: Genetic algorithms are a type of evolutionary algorithm that uses principles of natural selection to search for solutions to problems. NAS algorithms use genetic algorithms to search for ANN architectures that are likely to be fit for a given task.

Performance Estimation - NAS

- The search strategies finds a neural architecture A that maximizes some performance measure, such as accuracy on unseen data.
- To guide their search process, these strategies need to estimate the performance of a given architecture A they consider.
- The simplest way of doing this is to train A on training data and evaluate its performance on validation data.
- However, training each architecture to be evaluated from scratch frequently yields computational demands in the order of thousands of GPU days for NAS.
- This naturally leads to developing methods for speeding up performance estimation.

Performance Estimation - NAS

Speed-up method	How are speed-ups achieved?	References
Lower fidelity estimates	Training time reduced by training for fewer epochs, on subset of data, downscaled models, downscaled data, ...	Li et al. (2017), Zoph et al. (2018), Zela et al. (2018), Falkner et al. (2018), Real et al. (2019), Runge et al. (2019)
Learning Curve Extrapolation	Training time reduced as performance can be extrapolated after just a few epochs of training.	Swersky et al. (2014), Domhan et al. (2015), Klein et al. (2017a), Baker et al. (2017b)
Weight Inheritance/ Network Morphisms	Instead of training models from scratch, they are warm-started by inheriting weights of, e.g., a parent model.	Real et al. (2017), Elsken et al. (2017), Elsken et al. (2019), Cai et al. (2018a,b)
One-Shot Models/ Weight Sharing	Only the one-shot model needs to be trained; its weights are then shared across different architectures that are just subgraphs of the one-shot model.	Saxena and Verbeek (2016), Pham et al. (2018), Bender et al. (2018), Liu et al. (2019b), Cai et al. (2019), Xie et al. (2019)

Performance Estimation – Lower Fidelity

- Performance can be estimated based on lower fidelities of the actual performance after full training (also denoted as proxy metrics).
- Such lower fidelities include shorter training times, training on a subset of the data, on lower-resolution images, or with less filters per layer and less cells.
- While these low-fidelity approximations reduce the computational cost, they also introduce bias in the estimate as performance will typically be underestimated.

Performance Estimation – Learning Curve Extrapolation

- Authors propose to extrapolate initial learning curves and terminate those predicted to perform poorly to speed up the architecture search process.
- Some consider architectural hyperparameters for predicting which partial learning curves are most promising.
- Training a surrogate model for predicting the performance of novel architectures is also proposed, do not employ learning curve extrapolation but support predicting performance based on architectural/cell properties and extrapolate to architectures/cells with larger size than seen during training.

Performance Estimation – Network Morphisms

- Another approach to speed up performance estimation is to initialize the weights of novel architectures based on weights of other architectures that have been trained before.
- One way of achieving this, dubbed network morphisms, allows modifying an architecture while leaving the function represented by the network unchanged, resulting in methods that only require a few GPU days.
- This allows increasing the capacity of networks successively and retaining high performance without requiring training from scratch.

Performance Estimation – One-Shot Architecture Search

- One-Shot Architecture Search treats all architectures as different subgraphs of a supergraph (the one-shot model) and shares weights between architectures that have edges of this super graph in common.
- Only the weights of a single one-shot model need to be trained (in one of various ways), and architectures (which are just subgraphs of the one-shot model) can then be evaluated without any separate training by inheriting trained weights from the one-shot model.
- This greatly speeds up performance estimation of architectures, since no training is required (only evaluating performance on validation data), again resulting in methods that only require a few GPU days.

One-Shot Architecture Search Example

References

- Neural Architecture Search: A Survey: https://arxiv.org/abs/1808.06993 by Chen et al.
- Neural Architecture Search with Reinforcement Learning: https://arxiv.org/abs/1606.03657 by Zoph et al.
- Neural Architecture Search with Bayesian Optimization: https://arxiv.org/abs/1802.03268 by Real et al.

Deep Neural Network

AIML Module 9

Seetha Parameswaran

BITS Pilani

The author of this deck, Prof. Seetha Parameswaran, is gratefully acknowledging the authors who made their course materials freely available online.

Time Series Modelling and Forecasting

What we Learn….

9.1 Univariate, Multivariate and Multi-step CNN Models

9.2 Univariate, Multivariate and Multi-step LSTM Models

Time Series Data

Time series Data

- General definition: “A time series is a collection of observations made
- sequentially through time, whose dynamics is often characterized by short/long period fluctuations (seasonality and cycles) and/or long period direction (trend)”.
- Such observations may be denoted by X_1 , X_2 , X_3 , ... X_t , ... , X_T since data are usually collected at discrete points in time.
  - The interval between observations can be any time interval (seconds, minute, hours, days, weeks, months, quarters, years, etc.) and we assume that these time **periods are equally spaced**.
  - One of the most distinctive characteristics of a time series is the mutual dependence between the observations, generally called SERIAL CORRELATION OR **AUTOCORRELATION.**

Time series example 1

Numbers of Doctorates Awarded in US, annual data – Engineering Vs. Education

At a glance

Annual data

Different «directions»

No big fluctuations

Time series example 2

Monthly carbon dioxide concentration (globally averaged over marine surface sites)

At a glance

Monthly basis data

Regular pattern

Constant fluctuations

Average value increases year by year

Time series example 3

LinkedIn daily stock market closing price

At a glance

Daily basis data

Very irregular dynamic

Many sudden changes

Time series example 4

Number of photos uploaded on the Instagram every minute (regional sub-sample)

At a glance

Minute basis data

Almost regular daily pattern but with some anomalies and spikes

Time series example 5

Acceleration detected by a smartphone sensors during a workout session (10 seconds)

At a glance

Milliseconds basis data

Each sensor has its own dynamics

Time Series Data Properties: Main Elements

1. **TREND**

- The general direction in which the series is running during a long period.
- A TREND exists when there is a long-term increase or decrease in the data.
- It does not have to be necessarily linear (could be exponential or others functional form).

Time Series Data Properties: Main Elements

2. CYCLE

- Long-term fluctuations that occur regularly in the series.
- A CYCLE is an oscillatory component (i.e. Upward or Downward swings) which is repeated after a certain number of years, so:
  - May vary in length and usually lasts several years (from 2 up to 20/30)
  - Difficult to detect, because it is often confused with the trend component.

Time Series Data Properties: Main Elements

3. SEASONAL EFFECTS

- Short-term fluctuations that occur regularly - often associated with months or quarters.
- A SEASONAL PATTERN exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, day of the week).
- Seasonality is always of a fixed and known period.

Time Series Data Properties: Main Elements

4. RESIDUAL

- Whatever remains after the other components have been taken into account.
- The residual/error component is everything that is not considered in previous components.
- Typically, it is assumed to be the sum of a set of random factors (e.g. a white noise series) not relevant for describing the dynamics of the series.

Seasonal effect: Additive Seasonality

- When the seasonality in **Additive**, the dynamics of the components are independents from each other; for instance, an increase in the trend-cycle will not cause an increase in the magnitude of seasonal dips.
- The difference of the trend and the raw data is **roughly constant in similar periods of time** (months, quarters) irrespectively of the tendency of the trend.

EXAMPLES OF ADDITIVE SEASONALITY

Seasonal effect: Multiplicative Seasonality

- In the multiplicative model the amplitude of the seasonality increase (decrease) with an increasing (decreasing) trend, therefore, on the contrary to the additive case, the **components are not independent from each other.**
- When the variation in the seasonal pattern (or the variation around the trend-cycle) **appears to be proportional** to the level of the time series, then a multiplicative model is more appropriate.

EXAMPLES OF MULTIPLICATIVE SEASONALITY

Seasonal effect: Frequency

- According to the data granularity and to the type of seasonality you want to model, it is important to consider the right seasonal frequency (i.e. how many observations you have for every seasonal cycle)
  - For annual seasonality, the data points are years, quarters, months or weeks (in this case you will face only annual seasonality), but if the frequency of observations is smaller than a week, things get more com

Frequency		Cycle type			
		Hour	Day	Week	Year
Data granularity	Annual				1
	Quarterly				4
	Monthly				12
	Weekly			1	52.18
	Daily		1	7	365.25
	Hourly	1	24	168	8766
	Minutes	60	1440	10080	525960

*Every year, on average, is made up of 365 days and 6 hours → so 365.25 days and 365.25/7=52.18 weeks

Numerical and Graphical description of Time Series

- The first step in Time Series Analysis is to produce a detailed exploratory analysis of the data to get some insights about the distribution of the series over time.

Graphical descriptive analyses

- Time plot
- Seasonal plot
- Box plot analysis
- Scatterplots (Lag plots)
- Plotting auto-correlation and cross-correlation functions

Numerical descriptive analyses

- Sampling period evaluation (start, end, data points features)
- Number of data available
- Missing value and outlier evaluation
- Frequency distribution analysis
- Summary descriptive statistics (overall and by season)

Time Series Data and Analysis

Time Series Analysis

- **Time series analysis** is a specific way of analyzing a sequence of data points collected over an interval of time.
- **Time** is a crucial variable because it shows how the data adjusts over the course of the data points as well as the final results. It provides an additional source of information and a set order of dependencies between the data.
- **Time series analysis** involves developing models to gain an understanding of the data to understand the underlying causes. Analysis can provide the “why” behind the outcomes you are seeing.
- **Forecasting** then takes the next step of what to do with that knowledge and the predictable extrapolations of what might happen in the future.

Main Objectives of Time Series Analysis

- Summary description (graphical and numerical) of data point vs. time
- Interpretation of specific series features (e.g. seasonality, trend, relationship with other series)
- Forecasting (e.g. predict the series values in (t + 1), (t + 2), ... , (t + k))
- Hypothesis testing and Simulation (comparing different scenarios)

Time Series Forecasting

Time Series Forecasting

- **Time series forecasting** is the process of analyzing time series data using statistics and modeling to make predictions and inform strategic decision-making.
- Time series forecasting is part of **predictive** analytics.
- It can show likely changes in the data, like **seasonality or cyclic behavior**, which provides a better understanding of data variables and helps forecast better.

Challenges in Time Series Forecasting

- At the time of the work, the future outcome is completely unavailable and can only be estimated through careful analysis and evidence-based priors.
- Likelihood of forecasts can vary wildly, especially when dealing with the commonly fluctuating variables in time series data as well as factors outside our control.

Applications of Time Series Forecasting

- Weather forecasting
- Climate forecasting
- Economic forecasting
- Healthcare forecasting
- Engineering forecasting
- Finance forecasting
- Retail forecasting
- Business forecasting
- Environmental studies forecasting
- Social studies forecasting

Time series forecasting considerations

1. Time horizons
   - The time frame of your forecast is known as a time horizon.
   - Time horizon is a fixed point in time where the forecast ends.
   - It’s much easier to forecast a shorter time horizon with fewer variables than it is a longer time horizon.
2. Dynamic and static states
   - If the forecast is static, it is set in stone once it is made, so make sure your data is adequate for a forecast.
   - Dynamic forecasts can be constantly updated with new information as it comes in. This means you can have less data at the time the forecast is made, and then get more accurate predictions as data is added.

Time series forecasting considerations

3. Data quality
   - Typical guidelines for data quality apply.
     - Make sure data is complete,
     - Data is not duplicated or redundant,
     - Data was collected in a timely and consistent manner,
     - Data is in a standard and valid format,
     - Data is accurate for what it is measuring,
     - Data is uniform across sets.
     - Data is collected at consistent intervals over the period of time being tracked. This helps account for trends in the data, cyclic behavior, and seasonality. It also can help identify if an outlier is truly an outlier or if it is part of a larger cycle. Gaps in the data can hide cycles or seasonal variation, skewing the forecast as a result.

Limitations of Classical Methods for TS Forecasting

1. **Focus on complete data**: missing or corrupt data is generally unsupported.
2. **Focus on linear relationships**: assuming a linear relationship excludes more complex joint distributions.
3. **Focus on fixed temporal dependence**: the relationship between observations at different times, and the number of lag observations provided as input, must be diagnosed and specified.
4. **Focus on univariate data**: many real-world problems have multiple input variables.
5. **Focus on one-step forecasts**: many real-world problems require forecasts with a long time horizon.

Strengths of DNN for TS Forecasting

1. **Robust to Noise**. Neural networks are robust to noise in input data and in the mapping function and can support learning and prediction in the presence of missing values.
2. **Nonlinear**. Neural networks do not make strong assumptions about the mapping function and readily learn linear and nonlinear relationships.
3. **Multivariate Inputs**. An arbitrary number of input features can be specified, providing direct support for multivariate forecasting.
4. **Multi-step Forecasts**. An arbitrary number of output values can be specified, providing direct support for multi-step and even multivariate forecasting.

Strengths of DNN for TS Forecasting

5. **Feature Learning in CNN**. Automatic identification, extraction and distillation of salient features from raw input data that pertain directly to the prediction problem that is being modeled.
6. **Native Support for Sequences.** Recurrent neural networks directly add support for input sequence data.
7. **Learned Temporal Dependence.** The most relevant context of input observations to the expected output is learned and can change dynamically.

Limitations of DNN for TS Forecasting

1. The mapping function is fixed or static.
2. Fixed Inputs. The number of lag input variables is fixed, in the same way as traditional time series forecasting methods.
3. Fixed Outputs. The number of output variables is also fixed; although a more subtle issue, it means that for each input pattern, one output must be produced.

Framework for Time Series Forecasting

Framework

Time series forecasting involves **developing and using a predictive model** on data where there is an **ordered relationship between observations.**

Framework

1. What are the inputs and outputs for a forecast?
    a. Inputs: Historical data provided to the model in order to make a single forecast.
    b. Outputs: Prediction or forecast for a future time step beyond the data provided as input.
2. What are the endogenous and exogenous variables?
    a. Endogenous: Input variables that are influenced by other variables in the system and on which the output variable depends.
    b. Exogenous: Input variables that are not influenced by other variables in the system and on which the output variable depends.
3. Are you working on a regression or classification predictive modeling problem?
    a. ˆRegression: Forecast a numerical quantity.
    b. ˆClassification: Classify as one of two or more labels.

Framework

4. Are the time series variables unstructured or structured?
    a. Unstructured: No obvious systematic time-dependent pattern in a time series variable.
    b. Structured: Systematic time-dependent patterns in a time series variable (e.g. trend and/or seasonality).
5. Are you working on a univariate or multivariate time series problem?
    a. Univariate: One variable measured over time.
    b. Multivariate: Multiple variables measured over time.
    c. Univariate and Multivariate Inputs: One or multiple input variables measured over time.
    d. Univariate and Multivariate Outputs: One or multiple output variables to be predicted.

Framework

6. Do you require a single-step or a multi-step forecast?
    a. One-step: Forecast the next time step.
    b. Multi-step: Forecast more than one future time steps.
7. Do you require a static or a dynamically updated model?
    a. Static. A forecast model is fit once and used to make predictions.
    b. Dynamic. A forecast model is fit on newly available data prior to each prediction.
8. Are your observations contiguous or discontiguous?
    a. Contiguous. Observations are made uniform over time.
    b. Discontiguous. Observations are not uniform over time.

Steps to Develop a Forecasting Model

1. Define Problem.
   a. Inputs vs. Outputs: What are the inputs and outputs for a forecast?
   b. Endogenous vs. Exogenous: What are the endogenous and exogenous variables?
   c. Unstructured vs. Structured: Are the time series variables unstructured or structured?
   d. Regression vs. Classification: Are you working on a regression or classification predictive modeling problem? What are some alternate ways to frame your time series forecasting problem?
   e. Univariate vs. Multivariate: Are you working on a univariate or multivariate time series problem?
   f. Single-step vs. Multi-step: Do you require a single-step or a multi-step forecast?
   g. Static vs. Dynamic: Do you require a static or a dynamically updated model?
   h. Contiguous vs. Discontiguous: Are your observations contiguous or discontiguous?

   Some useful tools to help get answers include:

   a. ˆData visualizations (e.g. line plots, etc.).
   b. ˆStatistical analysis (e.g. ACF/PACF plots, etc.).

Steps to Develop a Forecasting Model

2. Design Test Harness
   a. Split the dataset into a train and test set.
   b. Fit a candidate approach on the training dataset.
   c. Make predictions on the test set.
   d. Calculate a metric that compares the predictions to the expected values.

Steps to Develop a Forecasting Model

3. Test Models.: Test many models using your test harness.
   a. Baseline. Simple forecasting methods such as persistence and averages.
   b. Autoregression. The Box-Jenkins process and methods such as SARIMA.
   c. Exponential Smoothing. Single, double and triple exponential smoothing methods.
   d. Linear Machine Learning. Linear regression methods and variants such as regularization.
   e. Nonlinear Machine Learning. kNN, decision trees, support vector regression and more.
   f. Ensemble Machine Learning. Random forest, gradient boosting, stacking and more.
   g. Deep Learning. MLPs, CNNs, LSTMs, and Hybrid models.

Steps to Develop a Forecasting Model

4. Finalize Model.
    a. List of the top 5 to 10 candidate models that are skillful on the problem.
    b. Pick one or multiple models and finalize them. This involves training a new final model on all available historical data (train and test). The model is ready for use.

Prepare Time Series Data for DNNs

Prepare Time Series Data for DNNs

Objectives

- Transform a time series dataset into a two-dimensional supervised learning format.
- Transform a two-dimensional time series dataset into a three-dimensional structure suitable for CNNs and LSTMs.
- Split a very long time series into subsequences ready for training a CNN or LSTM model.

Time Series Data to 2D Supervised Learning Data

Univariate Time Series Data

time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

Univariate Time Series Data for supervised learning

X,	y
?,	100
100,	110
110,	108
108,	115
115,	120
120,	?

Using sliding window method.

Time Series Data to 2D Supervised Learning Data

Multivariate Time Series Data

time,	measure1,	measure2
1,	0.2,	88
2,	0.5,	89
3,	0.7,	87
4,	0.4,	88
5,	1.0,	90

Multivariate Time Series Data for supervised learning

X1,	X2,	X3,	y
?,	?,	0.2,	88
0.2,	88,	0.5,	89
0.5,	89,	0.7,	87
0.7,	87,	0.4,	88
0.4,	88,	1.0,	90
1.0,	90,	?,	?

Using sliding window method.

Time Series Data to 2D Supervised Learning Data

Multivariate Time Series Data

time,	measure1,	measure2
1,	0.2,	88
2,	0.5,	89
3,	0.7,	87
4,	0.4,	88
5,	1.0,	90

Multivariate Time Series Data for sequence prediction (multi-step)

X1,	X2,	y1,	y2
?,	?,	0.2,	88
0.2,	88,	0.5,	89
0.5,	89,	0.7,	87
0.7,	87,	0.4,	88
0.4,	88,	1.0,	90
1.0,	90,	?,	?

Using sliding window method.

Time Series Data to 3D Supervised Learning Data

- First convert time series data to 2D data for supervised learning.
- Then convert 2D data to 3D data.
  - The input layer for CNN and LSTM models is specified by the input shape argument on the first hidden layer of the network.
  - The input to every CNN and LSTM layer must be three-dimensional.
    - The three dimensions of this input are [samples, timesteps, features]
      - **Samples**. One sequence is one sample.
      - **Time Steps**. One time step is one point of observation in the sample. One sample is comprised of multiple time steps.
      - **Features**. One feature is one observation at a time step. One time step is comprised of one or more features.

Example: Helper function

```
# transform univariate time series to supervised learning problem
from numpy import array

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
  X, y = list(), list()
  for i in range(len(sequence)):
    # find the end of this pattern
    end_ix = i + n_steps
    # check if we are beyond the sequence
    if end_ix > len(sequence)-1:
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    y.append(seq_y)

  return array(X), array(y)
```

Example: Convert TS data to 2D data

```
# define univariate time series
series = array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
print(series.shape)
# transform to a supervised learning problem
X, y = split_sequence(series, 3)
print(X.shape, y.shape)
# show each sample
for i in range(len(X)):
  print(X[i], y[i])
```

```
(10,)

(7, 3) (7,)

[1 2 3] 4
[2 3 4] 5
[3 4 5] 6
[4 5 6] 7
[5 6 7] 8
[6 7 8] 9
[7 8 9] 10
```

Example

```
# define univariate time series
series = array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
print(series.shape)
# transform to a supervised learning problem
X, y = split_sequence(series, 3)
print(X.shape, y.shape)
# show each sample
for i in range(len(X)):
  print(X[i], y[i])
```

```
(10,)
(7, 3) (7,)
(7, 3, 1)
```

CNNs for Time Series Forecasting

Univariate CNN Models

- Univariate time series are datasets comprised of a single series of observations with a temporal ordering and a model is required to learn from the series of past observations to predict the next value in the sequence.
- The CNN model will learn a function that maps a sequence of past observations as input to an output observation.
- A one-dimensional CNN is a CNN model that has a convolutional hidden layer that operates over a 1D sequence.

Demo Code for CNN

- Univariate CNN
  - https://colab.research.google.com/drive/1VXCnwxtRdFkDYlxGqhnEuQeXTjQNc0uM#scrollTo=Ifms-u8BBraF
- Multivariate CNN
  - https://colab.research.google.com/drive/1aADi0xKTExjZSReSFmwTke_Ws4yAwp4k#scrollTo=8kDtsv_uJygm

PS: Use BITS email ID to access files.

RNNs for Time Series Forecasting

Univariate RNN Models

- The LSTM model will learn a function that maps a sequence of past observations as input to an output observation.
- The sequence of observations must be transformed into multiple examples from which the LSTM can learn.

Demo Code for RNN

- Univariate RNN
  - https://colab.research.google.com/drive/1b-ORe5WeYHpTHgcpFKPIoWRQcWNLXBq_#scrollTo=vfZ36VoM9ZIh
- Multivariate RNN
  - https://colab.research.google.com/drive/1s5bsuW-wmxyuPkIK_C4mOwc8JPqlMwwS

PS: Use BITS email ID to access files.

Ref

- Deep Learning for Time Series Forecasting by Jason Brownlee (R4)
- https://www.tableau.com/learn/articles/time-series-forecasting

Next Session:
CNN

Deep Neural Network

AIML Module 10

Seetha Parameswaran

BITS Pilani

The author of this deck, Prof. Seetha Parameswaran, is gratefully acknowledging the authors who made their course materials freely available online.

What we Learn….

10.1 Federated learning

10.2 Meta learning

10.3 Online (incremental) learning

Federated Learning

Federated Learning

- Federated learning is a way to train AI models without anyone seeing or touching your data.
- Google introduced the term federated learning in 2016.
- AI models are being trained on the edge, on data that never leave the source of the data like laptop, server or mobile phone or wearable device.
- Decentralized approach.

Federated Learning

- Federated Learning is a machine learning approach that allows a model to be trained across multiple decentralized edge devices (such as smartphones, IoT devices, or local servers) holding local data samples, without exchanging them centrally.
- Instead of sending all data to a central server for model training, federated learning keeps data localized, thus addressing privacy, security, and bandwidth concerns.

Federated Learning

- Read world performance of ML model depends on **relevance** of the data used to train it.
- The best data is available at the **source** , ie the devices used **every day**.

Image: https://blog.ml.cmu.edu/2019/11/12/federated-learning-challenges-methods-and-future-directions/

Federated Learning

- Federated Learning = train a centralized model on decentralized data.
  - On-device data is used to train a central model. Data resides on the device and is not brought to the server on which training is performed.
  - Training is brought to the device. (Some challenges: Battery, wifi, idle, compute)
  - Devices participate when they are eligible. A subset of devices are selected to receive a training model. The model may be few megs.

Federated Learning

- Federated Learning = train a centralized model on decentralized data.
  - Training on the data on the device in few minutes, sends the training results to the server. The results are encrypted with a key that the server does not have.
  - Uses **Secure aggregation and Differential privacy**
  - ML models are trained on device after multiple rounds of training.
  - Test the ML model on the user devices.
  - New model is **static**. It learned from thousands of users who may have rare unique data.

Types of Federated Learning

1. Centralized federated learning
   - Centralized federated learning requires a central server.
   - It coordinates the selection of client devices in the beginning and gathers the model updates during training.
   - The communication happens only between the central server and individual edge devices.
   - Straightforward approach
   - Generates accurate models
   - Central server poses a bottleneck problem

https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

Types of Federated Learning

2. Decentralized federated learning
   - Decentralized federated learning does not require a central server to coordinate the learning.
   - The model updates are shared only among the interconnected edge devices.
   - The final model is obtained on an edge device by aggregating the local updates of the connected edge devices.
   - Prevents the possibility of a single-point failure.
   - The model's accuracy is completely dependent on the network topology of the edge devices.

https://www.researchgate.net/figure/Decentralized-Federated-Learning-Environments_fig1_344386575

Types of Federated Learning

3. Heterogeneous federated learning
   - Heterogeneous federated learning involves having heterogeneous clients such as mobile phones, computers, or IoT (Internet of Things) devices. These devices may differ in terms of hardware, software, computation capabilities, and data types.
   - HeteroFL can generate a single global model for inference after training over multiple varied local models.

Federated Learning Algorithms

1. Federated stochastic gradient descent (FedSGD)
   - In FedSGD, the central model is distributed to the clients, and each client computes the gradients using local data. These gradients are then passed to the central server, which aggregates the gradients in proportion to the number of samples present on each client to calculate the gradient descent step.
2. Federated averaging (FedAvg)
   - Clients can perform more than one local gradient descent update. Instead of sharing the gradients with the central server, weights tuned on the local model are shared. Finally, the server aggregates the clients' weights (model parameters).
   - If all the clients begin from the same initialization, averaging the gradients is equal to averaging the weights. Therefore, Federated Averaging leaves room for tuning the local weights before sending them to the central server for averaging.
3. Federated learning with dynamic regularization (FedDyn)
   - In federated learning, the global loss must be computed based on local losses generated from heterogeneous devices.
   - Due to the heterogeneity of clients, minimizing global loss is different than minimizing local losses. Therefore, FedDyn method aims to generate the regularization term for local losses by adapting to the data statistics, such as the amount of data or communication cost. This modification of local losses through dynamic regularization enables local losses to converge to the global loss.

Federated Learning

- Secure aggregation enables server to combine the encrypted results and only decrypt the aggregate.
  - On each device, before anything is sent, the secure aggregation protocol adds zero-sum masks to scramble training results.
  - Add up all the training results to cancel out the masks.
- Unique rare data.
  - ML model should not memorize data specific to a device.
  - Add noise to obscure rare data. This is known as differential privacy.
- Differential privacy
  - Deal with the risk of model memorization, where a shared model's parameters may be influenced heavily by a single data.
- Federated Learning = ML model learns from everyone without learning about anyone.

Steps in Federated Learning

Step 1	Step 2	Step 3	Step 4
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data	Central server pools model results and generate one global mode without accessing any data

Steps in Federated Learning

1. Initialization:
   a. A global model is initialized on a central server. This model serves as a starting point for training.
   b. A ML model (e.g., linear regression, neural network, boosting) is chosen to be trained on local nodes and initialized.
2. Client selection
   a. A fraction of local nodes are selected to start training on local data.
   b. The selected nodes acquire the current statistical model while the others wait for the next federated round.
3. Local Training:
   a. Each edge device performs local model updates using its own local data but does not send data samples to the central server.
   b. The local model is trained on the device using a portion of the data.
   c. The device computes the gradients of the local model with respect to its data but only shares these gradients with the central server, not the data itself.
4. Aggregation:
   a. The central server collects gradient updates from all participating devices.
   b. It aggregates these updates to update the global model.
   c. Popular aggregation methods include federated averaging, weighted averaging, or other secure and privacy-preserving aggregation techniques.
5. Global Model Update:
   a. The updated global model is sent back to the participating devices.
6. Local Model Fine-Tuning:
   a. Devices further fine-tune the global model locally using their data to adapt it to their specific characteristics or changes in data distribution.
7. Repeat:
   a. Steps 2 to 6 are repeated iteratively for several rounds, allowing the global model to improve based on local data without centralizing it.

Federated Learning - Advantages

- Privacy:
  - User data remains on their devices, reducing privacy concerns associated with centralized data storage and processing.
- Efficiency:
  - Bandwidth and computational resources are saved because only model updates (gradients) are communicated, not raw data.
- Security:
  - The decentralized nature of federated learning makes it more resilient to certain attacks, such as data breaches during centralized data aggregation.
- Personalization:
  - Local fine-tuning allows the global model to adapt to individual device characteristics or data distributions, enabling personalized recommendations or predictions.

Federated Learning

- Use cases for federated learning
  - Transportation: self-driving cars
    - Training self-driving cars on aggregated real-word driver behaviour.
  - Medicine: digital health
    - Help hospitals improve diagnostics while maintaining patient privacy.
  - Industry 4.0: smart manufacturing
    - Improve the efficiency and effectiveness of industrial process while guaranteeing a high level of safety and privacy of sensitive data.
  - Robotics
    - from perception and decision-making to control.
    - Federated Learning is applied to improve multi-robot navigation under limited communication bandwidth scenarios, which is a current challenge in real-world learning-based robotic tasks.
    - Federated Learning is used to learn Vision-based navigation, helping better sim-to-real transfer.

Federated Learning - Challenges

- Dealing with non-IID (non-Independently and Identically Distributed) data
- Communication overhead
- Synchronization issues
- Heterogeneity between the different local datasets: each node may have some bias with respect to the general population, and the size of the datasets may vary significantly
- Temporal heterogeneity: each local dataset's distribution may vary with time
- Interoperability of each node's dataset is a prerequisite
- Each node's dataset may require regular curations
- Hiding training data might allow attackers to inject backdoors into the global model
- Lack of access to global training data makes it harder to identify unwanted biases entering the training e.g. age, gender, sexual orientation
- Partial or total loss of model updates due to node failures affecting the global model
- Lack of annotations or labels on the client side
- Heterogeneity between the processing platform

Ref: Federated Learning

a. https://federated.withgoogle.com/#:~:text=The%20federated%20learning%20approach%20for,and%20one%20driven%20by%20AI.
b. https://en.wikipedia.org/wiki/Federated_learning
c. https://arxiv.org/pdf/1902.04885.pdf
d. https://arxiv.org/pdf/1903.10635.pdf
e. Code: https://proandroiddev.com/federated-learning-e79e054c33ef

Meta Learning

Meta-Learning

- **Learning to learn**
- Focus on developing algorithms and techniques that enable models to learn and adapt quickly and efficiently from a limited amount of data or across different tasks.
- The core idea behind meta-learning is to **improve the learning process** itself by leveraging prior learning experiences to tackle new, unseen tasks effectively.
- A model not only learns from a handful of examples, but also learns to classify novel classes during model inference.
- Learning with Limited Labeled Data

Meta-Learning

	Shallow Learning	Deep Learning	Meta-Learning
Classifier	**Learned**	**Learned**	**Learned**
Feature	Hand-crafted	**Learned**	**Learned**
Learner	Hand-crafted	Hand-crafted	**Learned**

Meta-Learning

- Humans learn quickly.
- Humans adapt to new environments quickly.
- Humans leverage knowledge acquired from prior experience to solve novel tasks.
- Meta-learning leverages previous knowledge acquired from data to solve novel tasks quickly and more efficiently.

The problem of Generalization

- Supervised learning through deep learning methods requires massive amounts of labeled training data.
- The datasets are expensive to create, especially when one needs to involve a domain expert.
- Pre-training or transfer learning become less effective for domain-specific problems, which still require large amounts of task-specific labeled data to achieve good performance.
- Certain real world problems have long-tailed and imbalanced data distributions.
    - search engines: a few keywords are commonly searched for, whereas a vast majority of keywords are rarely searched for.

Meta-Learning

Meta-Learning

- **Learner** trains a model such as a neural network to solve a task such as visual object recognition.
- **Meta-learner** then wraps the learning process and trains the learner to better solve learning tasks.
- The **learner** typically minimizes a loss function that measures the difference between the true label of an input and the label predicted by a neural network model.
- The **meta-learner** minimizes a meta-loss function such as the generalization error of the trained neural network on novel data.

Why Meta-Learning

1. Ability to learn from a handful of examples
2. Learning or adapting to novel tasks quickly
3. Capability to build more generalizable systems.
4. Applications that require data-efficient approaches
   a. robots are tasked with learning new skills in the real world, and are often faced with new environments.

Key Aspects of Meta-Learning

1. Learning to Adapt:
   a. In meta-learning, the goal is to train a model to be highly adaptable, so it can rapidly adapt to new tasks with minimal additional training.
   b. In traditional machine learning, models are trained for specific tasks and require a substantial amount of labeled data.
2. Few-shot or Zero-shot Learning:
   a. Meta-learning often deals with scenarios where a model needs to learn from very few examples (few-shot learning) or even no examples (zero-shot learning) for a new task.
   b. The model leverages knowledge gained from previous tasks to generalize effectively.
3. Task Similarity:
   a. Meta-learning assumes that there is some degree of similarity or structure among the tasks encountered.
   b. Models are designed to capture this task similarity and transfer knowledge accordingly.

Meta-Learning - Three Approaches

1. Model-based
   - using (cyclic) networks with external or internal memory
   - models updates its parameters rapidly with a few training steps, which can be achieved by its internal architecture or controlled by another meta-learner model.
   - Memory-Augmented Neural Networks - encode new information quickly
   - Meta Networks - learns a meta-level knowledge across tasks
2. Metrics-based
   - learning effective distance metrics
   - Convolutional Siamese Neural Network - twin networks whose output is jointly trained to learn the relationship between input data sample pairs.
   - Matching Networks - learn a network that maps a small labelled support set and an unlabelled example to its label
3. Optimization-based
   - explicitly optimizing model parameters for fast learning
   - LSTM Meta-Learner
   - Model-Agnostic Meta-Learning (MAML) - general optimization algorithm
   - Reptile - meta-learning optimization algorithm

Meta-Data

Two levels of data:

- Meta-Training Data:
  - This data consists of multiple tasks with associated training data.
  - The model learns from these tasks.
- Meta-Test Data:
  - After meta-training, the model is evaluated on new tasks for which it has not seen the training data.

Steps in Meta-Learning

1. Meta-Training:
   - Task Generation:
     - Create a dataset of tasks, each defined by a task description and a few examples.
   - Task Sampling:
     - Randomly select a task from the dataset of tasks.
   - Training:
     - Train a base model (often called the "learner" or "base learner") on the examples provided by the selected task.
2. Meta-Testing:
   - Task Adaptation:
     - Select a new task from the dataset of tasks for which the model was not directly trained.
   - Fine-Tuning:
     - Use the few examples provided by the new task to fine-tune the base model.
   - Inference:
     - Make predictions or perform the task using the adapted model.

Steps in Meta-Learning

3. Evaluation:
   - Measure the performance of the adapted model on the new task.
   - Repeat the meta-testing process for multiple tasks to assess the model's ability to adapt across a range of tasks.
4. Meta-Learning Objective:
   - Define an objective or loss function that encourages the model to learn useful representations or adaptation strategies during the meta-training process.
   - This objective typically includes both the base model's training loss on each task and a regularization term that encourages the model to adapt well to new tasks during meta-testing.

Steps in Meta-Learning

5. Optimization:
   - Use optimization algorithms, such as gradient descent or reinforcement learning techniques, to update the model's parameters during meta-training.
   - During meta-testing, employ optimization algorithms like gradient descent to fine-tune the base model on new tasks.
6. Hyperparameter Tuning:
   - Tune hyperparameters of the meta-learning algorithm to achieve better adaptation performance.
   - These hyperparameters might include learning rates, regularization strengths, or architectural choices.

Steps in Meta-Learning

7. Regularization Techniques:
   - Apply regularization techniques to prevent overfitting during meta-training and encourage model generalization during meta-testing.
   - Common regularization methods include weight decay, dropout, and episodic training.
8. Model Architecture:
   - Choose an appropriate base model architecture that can be effectively adapted to various tasks.
   - Architectures can range from simple models like linear regressors to complex neural networks.

Steps in Meta-Learning

9. Dataset Preparation:
   - Curate or generate datasets of tasks and examples that are representative of the problem domain.
   - Ensure that tasks exhibit sufficient diversity and complexity to challenge the meta-learning model.
10. Iterative Process:
   - Meta-learning is often an iterative process where the model is trained on a sequence of tasks, gradually improving its adaptation capabilities.

Transfer Learning vs. Meta-Learning

- Both transfer learning and meta-learning involve leveraging prior knowledge
- Transfer learning fine-tunes a pre-trained model on a related task.
- Meta-learning trains a model to be able to adapt to various tasks from scratch, often with a minimal number of examples.

Meta Learning Libraries

- Torch-meta
  - https://github.com/tristandeleu/pytorch-meta
- learn2learn
  - https://github.com/learnables/learn2learn
- Meta-datasets
  - https://github.com/google-research/meta-dataset

Applications of Meta-Learning

1. Computer Vision:
   a. Few-Shot Image Classification: Meta-learning can be used to train models that can recognize new object categories with just a few examples.
   b. Object Detection and Segmentation: It can help in adapting object detection or segmentation models to new classes or domains.
2. Natural Language Processing (NLP):
   a. Transfer Learning for Text: Meta-learning can be applied to tasks such as sentiment analysis, language modeling, and named entity recognition to adapt models to specific domains or languages.
   b. Machine Translation: Meta-learning can facilitate adapting translation models to new language pairs.
3. Robotics:
   a. Robotic Control: Meta-learning can enable robots to adapt their control policies quickly to handle new environments or tasks.
   b. Object Manipulation: Robots can use meta-learned skills to manipulate objects with different shapes and sizes.
   c. Autonomous Vehicles:
   d. Adaptive Driving: Meta-learning can help autonomous vehicles adapt to changing road conditions or new traffic scenarios with limited real-world data.

Applications of Meta-Learning

6. Healthcare:
   a. Disease Diagnosis: Meta-learning can assist in building diagnostic models that adapt to the characteristics of specific patient populations.
   b. Drug Discovery: It can facilitate the adaptation of drug discovery models to new chemical compounds or biological targets.
7. Recommendation Systems:
   a. Personalization: Meta-learning can be used to personalize recommendation algorithms, adapting them to individual user preferences.
8. Finance:
   a. Financial Forecasting: Meta-learning models can adapt to changing market conditions for improved financial forecasting.
   b. Fraud Detection: Meta-learned fraud detection models can quickly adapt to new types of fraudulent activities.
9. Education:
   a. Adaptive Learning: In educational technology, meta-learning can adapt tutoring systems to individual student learning styles and needs.
10. Game Playing:
   a. Reinforcement Learning: Meta-learned reinforcement learning agents can adapt to new game environments or strategies efficiently.

Applications of Meta-Learning

11. Drug Discovery:
    a. Molecule Generation: Meta-learning can be applied to generate novel molecules with desired properties by adapting generative models to specific chemical spaces.
12. Meta Reinforcement Learning:
    a. Robotics Control: In robotics, meta-RL agents can adapt to various tasks or environments by quickly learning new policies.
13. Anomaly Detection:
    a. Network Security: Meta-learning models can adapt to changing cybersecurity threats and identify anomalies in network traffic.
14. Language Generation:
    a. Text Generation: Meta-learning can help text generation models adapt to specific writing styles or content domains.
15. Health Monitoring:
    a. Patient-Specific Monitoring: In healthcare, meta-learning can adapt monitoring models to individual patients' health data.
16. Human-Robot Collaboration:
    a. Cobotics: Robots can adapt their behavior to collaborate more effectively with humans in various contexts.

Ref: Meta Learning

a. https://arxiv.org/pdf/2004.05439.pdf
b. https://meta-learning.fastforwardlabs.com/
c. https://en.wikipedia.org/wiki/Meta-learning_(computer_science)
d. https://research.samsung.com/blog/Meta-Learning-in-Neural-Networks

Online / Incremental Learning

Incremental Learning

Ref: Online Learning

a. https://arxiv.org/pdf/1802.07569.pdf
b. https://arxiv.org/pdf/1904.07734v1.pdf

Thank you

All the best for the future :)