Announcements

[7 Jan 2025] Part I Begins: Basics of Applied Machine Learning

The course begins

Part I: Basics of Applied Machine Learning (~1 month)
Recommended Readings

- Case Studies: https://www.evidentlyai.com/ml-system-design (browse and get familiar with the case studies of classification, regression, and ranking with numerical, text, image, and multi-modal data)
- Basic ML: https://www.statlearning.com/ (first 3 chapters)

[14 Jan 2025] No Lecture: Happy Pongal

[16 Jan 2025] Assignment 1: Prototype (due 30 Jan 2025)

Build a **prototype** for *sms spam classification*

1. in prepare.ipynb write the functions to
    - load the data from a given file path
    - preprocess the data (if needed)
    - split the data into train/validation/test
    - store the splits at train.csv/validation.csv/test.csv
2. in train.ipynb write the functions to
    - fit a model on train data
    - score a model on given data
    - evaluate the model predictions
    - validate the model
        i. fit on train
        ii. score on train and validation
        iii. evaluate on train and validation
        iv. fine-tune hyper-params using train and validation (if necessary)
    - score three benchmark models on test data and select the best one

Notes:

- You may download sms spam data from https://archive.ics.uci.edu/ml/datasets/sms+spam+collection
- You may refer to https://radimrehurek.com/data_science_python/ for building a prototype
- You may refer to first 3 chapters of https://www.statlearning.com/ for basic ML concepts
- You may refer to the Solution Design example covered in the class as a guideline for experiment design

[30 Jan 2025] No Lecture

[6 Feb 2025] Part II Begins: Basics of MLOps

The Part II begings

Part II: Machine Learning Operations (MLOps) (~1 month)
Recommended Readings

- Case Studies: https://www.evidentlyai.com/ml-system-design (browse and get familiar with the case studies involving MLOps)
- MLOps: https://madewithml.com/ (browse and get familiar with different aspects of MLOps)

[11 Feb 2025] Assignment 2: Experiment Tracking (due 25 Feb 2024) (extended to 4th Mar 2025)

1. **data version control**
   in prepare.ipynb track the versions of data using dvc
   - load the raw data into raw_data.csv and save the split data into train.csv/validation.csv/test.csv
   - update train/validation/test split by choosing different random seed
   - checkout the first version (before update) using dvc and print the distribution of target variable (number of 0s and number of 1s) in train.csv, validation.csv, and test.csv
   - checkout the updated version using dvc and print the distribution of target variable in train.csv, validation.csv, test.csv
   - **bonus:** (decouple compute and storage) track the data versions using google drive as storage
2. **model version control and experiment tracking**
   in train.ipynb track the experiments and model versions using mlflow
   - build, track, and register 3 benchmark models using MLflow
   - checkout and print the model selection metric AUCPR for each of the three benchmark models

References
Data Version Control
https://dvc.org/doc/start/data-management/data-versioning
https://realpython.com/python-data-version-control/
https://towardsdatascience.com/how-to-manage-files-in-google-drive-with-python-d26471d91ecd
https://madewithml.com/courses/mlops/versioning/

ML Experiment Tracking
https://mlflow.org/docs/latest/tracking.html
https://mlflow.org/docs/latest/getting-started/intro-quickstart/index.html
https://www.datarevenue.com/en-blog/how-we-track-machine-learning-experiments-with-mlflow
https://towardsdatascience.com/experiment-tracking-with-mlflow-in-10-minutes-f7c2128b8f2c
https://madewithml.com/courses/mlops/experiment-tracking/

[27 Feb 2025] No Lecture

[7 Mar 2025] Part III Begins: Basics of Deep Learning

The Part III begings

Part III: Deep Learning (~1 month)
Recommended Readings

- Case Studies: https://www.evidentlyai.com/ml-system-design (browse and get familiar with the case studies of deep learning)
- Deep Learning: https://www.deeplearningbook.org/ (Book Part II, first 3 chapters)

[18 Mar 2025] Assignment 3: Testing & Model Serving (due 1 Apr 2025)

1. **unit testing**
    - In score.py, write a function with the following signature that scores a trained model on a text:

      def score(text:str,
                model:sklearn.estimator,
                threshold:float) -> prediction:bool,
                                    propensity:float
    - In test.py, write a unit test function test_score(...) to test the score function. You may reload and use the best model saved during experiments in train.ipynb (in joblib/pkl format) for testing the score function.
      You may consider the following points to construct your test cases:
        - does the function produce some output without crashing (smoke test)
        - are the input/output formats/types as expected (format test)
        - is prediction value 0 or 1
        - is propensity score between 0 and 1
        - if you put the threshold to 0 does the prediction always become 1
        - if you put the threshold to 1 does the prediction always become 0
        - on an obvious spam input text is the prediction 1
        - on an obvious non-spam input text is the prediction 0
2. **flask serving**
    - In app.py, create a flask endpoint /score that receives a text as a POST request and gives a response in the json format consisting of prediction and propensity
    - In test.py, write an **integration test** function test_flask(...) that does the following:
        - launches the flask app using command line (e.g. use os.system)
        - test the response from the localhost endpoint
        - closes the flask app using command line

In coverage.txt produce the coverage report output of the unit test and integration test using pytest

https://docs.pytest.org/en/8.0.x/
https://flask.palletsprojects.com/en/2.3.x/quickstart/

[18 Mar 2025] Assignment 4: Containerization & Continuous Integration (due 8 Apr 2025)

1. **containerization**

- create a docker container for the flask app created in Assignment 3
- create a Dockerfile which contains the instructions to build the container, which include
    - installing the dependencies
    - copying app.py and score.py
    - launching the app by running "python app.py" upon entry
- build the docker image using Dockerfile
- run the docker container with appropriate port bindings
- in test.py write test_docker(..) function which does the following
    - launches the docker container using commandline (e.g. os.sys(..), docker build and docker run)
    - sends a request to the localhost endpoint /score (e.g. using requests library) for a sample text
    - checks if the response is as expected
    - close the docker container

In coverage.txt, produce the coverage report using pytest for the tests in test.py

2. **continuous integration**

- write a pre-commit git hook that will run the test.py automatically every time you try to commit the code to your local 'main' branch
- copy and push this pre-commit git hook file to your git repo

References

https://docker-curriculum.com/
https://www.tutorialspoint.com/docker/docker_overview.htm

https://www.freecodecamp.org/news/how-to-dockerize-a-flask-app/

https://githooks.com/
https://www.atlassian.com/git/tutorials/git-hooks

https://www.giacomodebidda.com/posts/a-simple-git-hook-for-your-python-projects/

[27 Mar 2025] Assignment 5: Transfer Learning (due 17 April 2025) [extended to 19 April 2025]

1. Transfer Learning for image data using CNN
    - Download about 100 images of chickens and 100 images of ducks from the internet
    - In a google colab notebook, fine-tune a pre-trained convolutional neural network to classify duck vs chicken and output the classification report

    https://www.analyticsvidhya.com/blog/2021/07/step-by-step-guide-for-image-classification-on-custom-datasets/

    https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

    https://www.learnpytorch.io/

2. Transfer Learning for text data using Transformer

- Download the sentiment analysis dataset from https://www.kaggle.com/datasets/abhi8923shriv/sentiment-analysis-dataset

- Build a sentiment analysis classifier to classify the sentiment into positive, neutral, and negative by fine-tuning a pre-trained transformer model and print your classification report

https://alvinntnu.github.io/NTNU_ENC2045_LECTURES/temp/sentiment-analysis-using-bert-keras-movie-reviews.html

https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/

[7 Apr 2025] Part IV Begins: Emerging Methods

The Part IV begings

Part IV: Emerging Case Studies and Final Project (~1 month)
Recommended Readings

- Case Studies: https://www.evidentlyai.com/ml-system-design (browse and get familiar with the case studies of latest techniques such as LLM, RAG etc)
- Project Ideas: https://machinelearningprojects.net/ (browse and get familiar with different project ideas)

[25 Apr 2025] Final Project Presentations (6,7,8,9 May: 5pm to 7pm IST)(10 May, 4pm to 7pm)

6th, 7th, 8th, and 9th May 2025
5pm to 7pm IST

May 6
https://cmi-ac-in.zoom.us/j/86204786751?pwd=EpIXawjIh2PFwqKMbe8VAhkAnP2NyQ.1

May 7
https://cmi-ac-in.zoom.us/j/88475377727?pwd=lojgaGU9PaGqy4okZrvaPL0z80DGC3.1

May 8
Please use the last link if first ones are not working
https://cmi-ac-in.zoom.us/j/86204786751?pwd=EpIXawjIh2PFwqKMbe8VAhkAnP2NyQ.1

https://cmi-ac-in.zoom.us/j/87444245523?pwd=aybLolq39sVau53BX2PKFIgDKOUBaF.1

https://cmi-ac-in.zoom.us/j/81436335825?pwd=pnP8MTTWSfuIDTmQ1w4PokjT1BapLf.1

https://cmi-ac-in.zoom.us/j/87248382028?pwd=xiLlj8lKLyMbcxo8dKSKJkr43jd6cB.1

May 9

Please use the last link if first ones are not working

https://cmi-ac-in.zoom.us/j/83319817523?pwd=e9Xv6fmWCxwYafgdS9J5H2wW1cabpR.1

https://cmi-ac-in.zoom.us/j/82869555100?pwd=3Aoamaai4mJbYPmRqBaJS1zBMaa4KF.1

https://cmi-ac-in.zoom.us/j/82047485552?pwd=AFpP4A6XbiGEAR8ww6ZbRXtsfl6N6g.1

https://cmi-ac-in.zoom.us/j/84897634171?pwd=ENfdj5N7tbYwG0cAMRQoksicC4r4wY.1

May 10

https://cmi-ac-in.zoom.us/j/86304378531?pwd=dxLK6bOjhs1MJYwr5DsnzfSQptUhyB.1

https://app.zoom.us/wc/84093945615/start?fromPWA=1&pwd=Oy14n5DJIzUYW7A9FNKn1M3hze53iH.1

https://cmi-ac-in.zoom.us/j/83561099308?pwd=PTKGEq3YnaAwMIq7QKiGRPKHvuQrMN.1

https://cmi-ac-in.zoom.us/j/88298281919?pwd=5LybzNYDcnbnTiTfZoLbphuQYkINQp.1

https://cmi-ac-in.zoom.us/j/87946417398?pwd=TBzNNCqo5qmlladtGQmv7I5xa2ji2G.1

Course Overview

Logistics

- course: Applied Machine Learning (AML)
- instructor: Raghav Kulkarni
- email: kulraghav@gmail.com
- time: Tue/Thu 5pm to 6 15pm IST
- prerequisites: python, basic ML
- zoom link: https://cmi-ac-in.zoom.us/j/86204786751?pwd=EpIXawjIh2PFwqKMbe8VAhkAnP2NyQ.1

Learning Objective

To be able to methodically **navigate the problem space and the solution space** of a typical end-to-end application of ML

- **problem space:** understand the end-to-end framing of typical use cases of ML in practice
- **solution space**: understand the basic building blocks used for the end-to-end ML application

Course Structure

- 4 parts: Applied ML Basics, MLOps, Deep Learning, emerging case studies
- running use cases: spam filter, fraud detection, diabetic retinopathy detection, ads recommendation, relevance ranking
- 5 assignments: 2 jupyter notebooks (Applied ML Basics), 2 python codes (MLOps), 1 google colab (Deep Learning)
- 1 mid term + 1 final project with presentation

References

- **Text books:**
  - An Introduction to Statistical Learning (authors: Gareth M. James, Trevor Hastie, Daniela Witten, Robert Tibshirani)
  - Deep Learning: (authors: Ian Goodfelow, Yoshua Bengio (F.R.S.) , and Aaron Courville)
- **Websites:**
  - Basic ML: https://www.statlearning.com/
  - MLOps: https://madewithml.com/
  - Deep Learning: https://www.deeplearningbook.org/
  - Case Studies: https://www.evidentlyai.com/ml-system-design

Grading Structure

- **Assignments:** 5 assignments, one every two weeks: 50%
- **Midterm:** 1 exam in person: 20%
- **Final Project:** 1 project presentation online: 20%
- **Class Participation:** regular quizzes on case studies, in online class: 10%

Basics of Applied Machine Learning

Guiding Questions

Case Study: SMS Spam
What is the bigger picture for the problem

- Which product/service is relevant to the problem
  mobile phone sms service offered by a telecom company XYZ
- How does the product/service make money and add value?
  Typically a business generates *revenue* in exchange of providing some *value* to its customers/users:
  - revenue: charge per sms (incoming and outgoing)
  - value: instant and easy text communication with large network of participants

What is the current situation that needs to be bettered

- too many SMS spams
- (quantitative) 20%+ messages are spam

Why is it problematic

- too much spam => bad user experience => users churn/stop using sms service => less engagement => less revenue for the company => topline comes down
- x% spam => f(x)% users churn => g(x)% revenue down

What is the desired state to be reached

- only reasonable amount of spam messages
- (quantitative) < 10% spam messages

Why is it desired

- reasonably low spam => minimal impact on user experience => minimal degradation of topline
- (quantitative) < 1% churn => < 1% revenue down due to spam

What treatment can possibly to achieve the desired state

- what remedy: automatically take down suspected (90% confidence) spam messages
- when to apply remedy: when messages are initiated, before they are sent to the user
- how to administer the remedy: instantaneously

What are the expected effects

- reduction in spam
- better engagement

- better revenue

What are the expected side effects
- genuine sms gets classified as spam
- delay in receiving message due to extra processing

What is the goal to be achieved
- result: reduce spam (from 20% -> 10%)
- consequences: better engagement

What are the constraints
- limited side effects
    - do not take down too many (more than 10% of) genuine messages
    - user complaints for taking down genuine messages < 1%
- sufficient latency/throughput
    - automatic take down happens in real-time (within 3 seconds, 99% of times)
    - supports high peak traffic (100k messages per minute, 99% of times)

What are the trade-offs
- cost of misclassification
    - classify spam as ham :
    - classify ham as spam : higher (10x more chance of churn)
- benefit of correct classification
    - classify spam as spam : higher (10x more valuable)
    - classify ham as ham

What are the (simple) non-ML alternatives
- rule based spam filter less accurate, might be complicated to maintain with evolving spam

What is the ML value proposition
- more accurate, easy to retrain and maintain

What is the evidence that ML solution is feasible
- similar solutions/prototypes are available/doable for text classification based on company blogs/research papers

ML Design Framework

Problem Framing

To **navigate the problem space**
- understand the **problem** and its **context** (bigger picture/what&why)
- understand the relevant **cause and effects** (itermediate smaller pieces and their connections)
- understand the **quantifications** of the causal implications (impact/importance)

- understand the proposed **treatment**, its **effects, side effects**, and **trade-offs**
- understand the **alternatives** and **opportunity costs**

This will help you formulate (qualitatively and quantitatively) the following:

Bigger picture: current_state – improve_to→ desired_state (why)

current	desired	why

Treatment options: proposed_treatment => desired_effects (and limited side-effects)

treatment (what/when/how)	effets	side-effects

Goal: improve product_metrics while maintaining hard_constraints (and balancing trade-offs)

improve/optimize	constraints	trade-offs/priorities

Why ML: alternatives have painpoints which ML solution can overcome (and ML is feasible)

alternatives	ML value proposition	ML feasibility

Solution Design

To **navigate the solution space:**

- understand the **choices** you will have to make for **data, model, action, and impact**
- understand how would you **measure** the quality of these choices
- understand the **trade-offs** between different choices of **tools and techniques**

This will help you formulate the following via typical **causal flow** of ML application:
data => model => action => impact (=> feedback loop to data/model/action in RL)

	choices	metrics	experiments
data			
model			

action			
impact			

System Design

Requirements

functional	non-functional

Estimations

	back-of-envelope calculation
storage	
compute	
network	

Process Flows

process	steps/flow

System Design Diagram

component	diagram

Case Study: SMS Spam

Problem Framing

Bigger picture: current_state – improve_to→ desired_state (why)

current	desired	context
too much spam => bad user experience	lesser spam => more engagement	relevant product/service/industry:

=> less engagement (churn) => less revenue	=> more revenue	sms service in a company in telecom industry revenue: pay per sms value: instant, easy, and reliable text communication on large network of users
10 + x% spam => f(x)% drop in engagement => g(x)% drop in revenue	x% lesser spam => f(x)% more engagement => g(x)% more revenue	- revenue of 10 cents per sms - sms is received within 5 seconds of sending 99% of times
- what is the current situation? - why is it problematic?	- what is the desired outcome or success criteria? - why is it desired?	- what industry/company/product/service is relevant? - how does it make money? - what value does it provide? - what are the pieces of the puzzle and relationship between them?

Treatment options: proposed_treatment => desired_effects (and limited side-effects)

treatment (what/when/how)	effets	side-effects
- automatically remove/label suspected spam messages - after creation before reaching user - instantaneously in real-time	- actual spam is reduced	- genuine messages might get removed mistakenly as spam
- latency 5 seconds	- 20% reduction in spam	- more than 5% false positives might cause user churn
- what is the proposed remedy - when is the remedy to be applied - how is the remedy expected to be	- what is the main effect of the treatment	- what are the potential side effects

applied		

Goal: improve product_metrics while maintaining hard_constraints (and balancing trade-offs)

improve/optimize	constraints	trade-offs/priorities
reduce spam	- limited side effects due to removing genuine messages as spam - the solution must be real-time	- mistakenly removing genuine messages as spam is much more damaging - delaying message delivery too much is more damaging
- reduce spam from 20% to 10% - take down when confidence in suspected spam is 90%+	- less than 1% complains for genuine messages flagged as spam - latency 5 seconds, 99% of times	- false positive 10x more damaging than false negative
- what is the result we want to achieve?	- what are the constraints? - which side effects must be kept under control?	- which errors (e.g. FP/FN) are more costly? - what are the priorities?

Why ML: alternatives have painpoints which ML solution can overcome (and ML is feasible)

alternatives	ML value proposition	ML feasibility
rule-based filter - less accurate - too many false positives - too many rules might be complex to maintain	- more accurate - less falses positives - easier to maintain	highly accurate text classification models seem possible with small to medium size labelled data
- 50% false positives	- expected less than 10% false positives	- around 10k manually labelled samples - simple (e.g. tfidf + logistic regression type) models with low latency
- what are the simpler (possibly non-ML) alternatives? - what are the pros and cons?	- what are pros and cons of ML solution? - what is the advantage over non-ML?	- what is the evidence for the feasibility of ML solution?

- what are their major pain points?		

Solution Design

	choices	metrics	experiments
data	sample historic text sms data stratified by region of origin manually label spam/ham	- class imbalance - message length distribution - word distribuion	(generate hypotheses via **data profiling**) - cleaning (e.g. missing values, special symbols in messages etc) - EDA (e.g. target distributions, word counts, lengths etc) - eyeball say 10 samples from each class
model	text->prob(spam)	- AUCPR - precision/recall curve	(select model via **benchmarking** model metrics on train/val/test) - rule based - tfidf + logistic - BERT (neural network)
action	if prob(spam) > threshold then automatically label and take down as soon as text message is created before it is received	- precision - recall - confusion matrix for the selected threshold	(select action via **analyzing and optimizing trade-offs**) pick threshold that maximizes recall subject to precision above 90% using precision/recall curve

impact	(effects) decrease in spam => increase in engagement => increase in revenue (side effects) limited user complaints for auto take-down of genuine messages	effects - % decrease in spam - % increase in daily active users - % increase in revenue 2 weeks after launch side effects - % user complaints - % churn	(evaluate real impact via **online test**) A/B test, segment by region of origin

Case Study: ETA prediction

https://www.uber.com/blog/deepeta-how-uber-predicts-arrival-time
(we focus on uber ride service for this use case and omit the delivery drop service)

Problem Framing

Bigger picture: current_state – improve_to→ desired_state (why)

current	desired	context
current ETA models are less accurate => customer not happy when pickup arrival time exceeds ETA => likely churn/cancellations => loss of revenue due to churn current ETA models have high latency => bad user experience + challenging to scale and serve increasing demand + not suitable for incorporating real time data	more accurate ETA models => pickup arrival time within ETA => better user experience => less churn/cancellations => less loss of revenue due to churn low latency ETA models => real time scalability => better user experience	uber rideshare pickup service value: reliable, quickly available, and fairly priced cab ride on demand money: % commission from driver per ride

Treatment options: proposed_treatment => desired_effects (and limited side-effects)

treatment (what/when/how)	effets	side-effects
model based on real time network data in addition to pre-computed network data	more accuracy: less underprediction of ETA => customer happy	more complex models and real-time infrastruction

		might result in ETA overpredictions => cab drivers unhappy

Goal: improve product_metrics while maintaining hard_constraints (and balancing trade-offs)

improve/optimize	constraints	trade-offs/priorities
decrease the underpredictions of ETA (from 5min to 3min)	low latency for realtime solution side effect: overprediction should not exceed by 5min => cab drivers are not going to be happy otherwise	arrival time after ETA is much worse underprediction => customer might cancel and churn => loss of revenue

Why ML: alternatives have painpoints which ML solution can overcome (and ML is feasible)

alternatives	ML value proposition	ML feasibility
statistical rules => less accurate + more complex to manage as rules evolve over time precomputed network graph => less accurate	more accurate for complex relations + dynamic + scalable	depends on data + infra + skills similar solutions seem available in R&D works and prototypes

Solution Design

	choices	metrics	experiments
data	segment + driver + traffic labels: historic ETA and actual arrival time	stratification percentage across different regions, driver rating, time of day, special events historic ETA and arrival time distributions	data profile by segments, distributions of ETA, errors
model	segment + driver + traffic -> segment travel time	MAE 80th quantile of Absolute Error asymmetric MAE underpredictions are 5x worse than overprediction	benchmarks: - directly model ETA - residual to static model loss: asymmetric Huber features:

			historic segment travel time + driver rating + current traffic condition
action	combine segment predictions to obtain the fastest path	% times overall error exceeds 3min (asymmetric) % times pick up arrival time exceeds ETA by 3min % times pick up arrival time is lagging ETA by 9min	A* search shortest path using model predicted travel times for segments
impact	lesser churn due to arrival after pickup ETA lower ETA errors on an average	% cancellations due to arrival after pickup ETA average absolute ETA error	A/B test

Case Study: Job Recommendation

https://engineering.linkedin.com/blog/2022/near-real-time-features-for-near-real-time-personalization

Problem Framing

Bigger picture: current_state – improve_to→ desired_state (why)

current	desired	context
irrelevant stale recommendations => dismisses of recommendations => less engagement => less revenue	more relevant timely recommendations reflecting latest activities of users => less dismisses of recommendations => more engagement => more revenue	linkedin job recommendation service value: relevant recommendations for job seekers access to large pool of active jobseekers for the job posters money: talent solutions for employers + premium subscriptions for job seekers

Treatment options: proposed_treatment => desired_effects (and limited side-effects)

treatment (what/when/how)	effets	side-effects
incorporating near real time features together with batch features as soon as user logs in or resuming	less stale and more timely and relevant recommendations less dismisses	increased latency due to extra layer increased load on infrastructure (batch -> streaming)

Goal: improve product_metrics while maintaining hard_constraints (and balancing trade-offs)

improve/optimize	constraints	trade-offs/priorities
improve the relevance of top recommendations (decrease the dismisses)	low latency high latency => long time to load recommendations => bad user experience	irrelevant recommendation in top results is worse than missing a relevant recommendation in top results

Why ML: alternatives have painpoints which ML solution can overcome (and ML is feasible)

alternatives	ML value proposition	ML feasibility
batch processing delay in capturing user activity => irrelevant recommendations	near real time features => timely and relevant recommendations	simulated delay gives evidence that such real time features might improve relevance real time feature pipeline is feasible with modern technology

Solution Design

	choices	metrics	experiments
data	label: historic clicks signals: user + job + interaction	class imbalance percentage of sampes by user location, industry	train: positive samples: clicked jobs of user negative samples: (user, random job) pairs validation: 10% of train samples test:

			last week data
model	user + job + interaction -> prob(click)	ROC AUC	benchmarks: - logistic regression - neural network loss: log loss features: user profile text embedding job description text embedding user skill job skill historic success ratio
action	rank based on prob(click) and show top-5 recommendations	precision@5	choose top-k where k is decided based on user experience filter based on location, industry before ranking initial filter based on simple ML model can also be used
impact	increase in engagement decrease in dismiss rate	% clicks % dismisses	A/B test

Other Use Cases

Classification

Binary Classification

All these use cases can possibly be modeled as: input -> prob(output is positive).

However the “action” taken could be different in different use cases. For example:

use case	action

email spam filter to improve user engagement	choose a threshold to maximize recall subject to high precision
fraud detection to improve user trust and reduce loss	choose a threshold to maximize precision subject to very high recall
diabetic retinopathy detection to improve operational efficiency	prioratize based on prob score and bandwidth
ads recommendation to improve revenue and customer value	prioratize based on calibrated prob score, bid value, and bandwidth
relevance ranking to improve feed	prioritize based on prob score and diversity

Regression

use case	action
ETA prediction to improve user experience and reduce churn	aggregate ETA predictions of each segment using a shortest path algorithm such as A*
estimate task completion time to improve scheduling and planning	use estimated upper/lower bounds of tasks in order to optimize scheduling
energy consumption prediction to optimize energy usage	use estimated prediction in order to dynamically (say hourly) change the setpoint configurations based on weather condition, occupancy etc in order to optimize the total energy consumption
demand forecasting to improve inventory planning	use upper/lower bounds on forecasted demand in order to optimize the cost of inventory

Ranking

use case	action
video recommendation to improve user engagement and content discovery	show top-k videos based on score

search ranking to improve user experience	show top-k search results based on score
ads recommendation to improve revenue and customer value	prioratize based on calibrated prob score, bid value, and bandwidth
relevance ranking to improve feed	prioritize based on prob score and diversity
diabetic retinopathy detection to improve operational efficiency	prioratize based on prob score and bandwidth

Basics of Statistical Learning

Holy Grail: Generalization

generalization: the holy grail of statistical modeling

to train a model that achieves the statistical guarantee under the statistical assumption below

assumption	guarantee
test distribution is close to train distribution	test performance is close to train performance

train/test split

- test distribution is close to train distribution
- model does not see test data during training

- stratified sample can be obtained during train/test split to maintain the target distributions
- part of train data can be kept as validation data for the purpose of hyper-parameter tuning

bias variance tradeoff

expected model error on unseen test data = variance of model + bias of model + irreducible error

** expectation is taken over different samples of training data

in practice:

low bias + high variance => easy to achieve

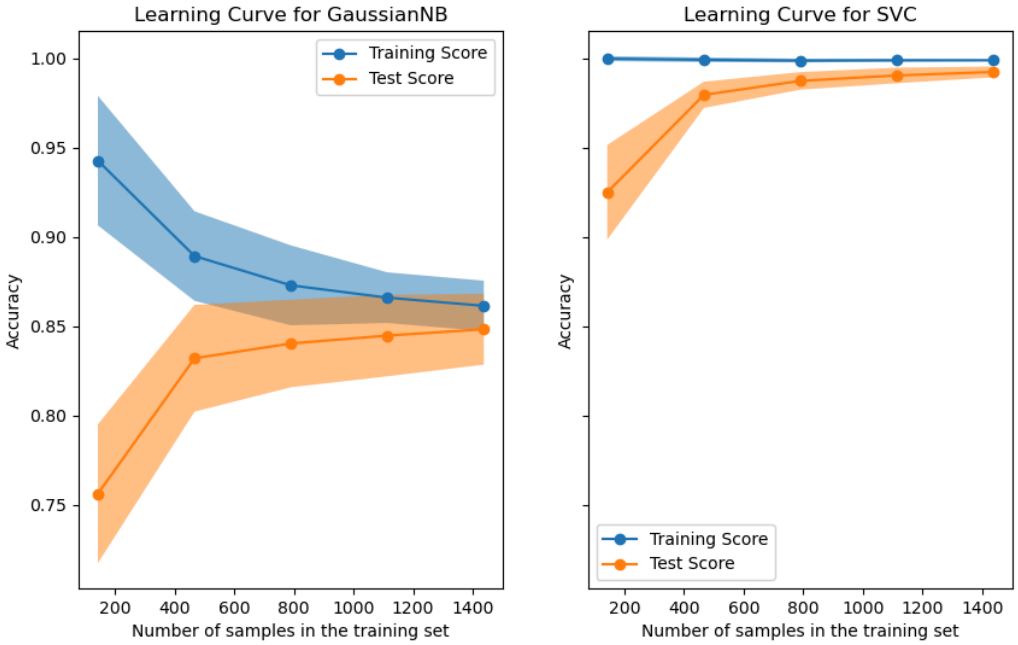
high bias + low variance => easy to achieve

low bias + low variance => challenging

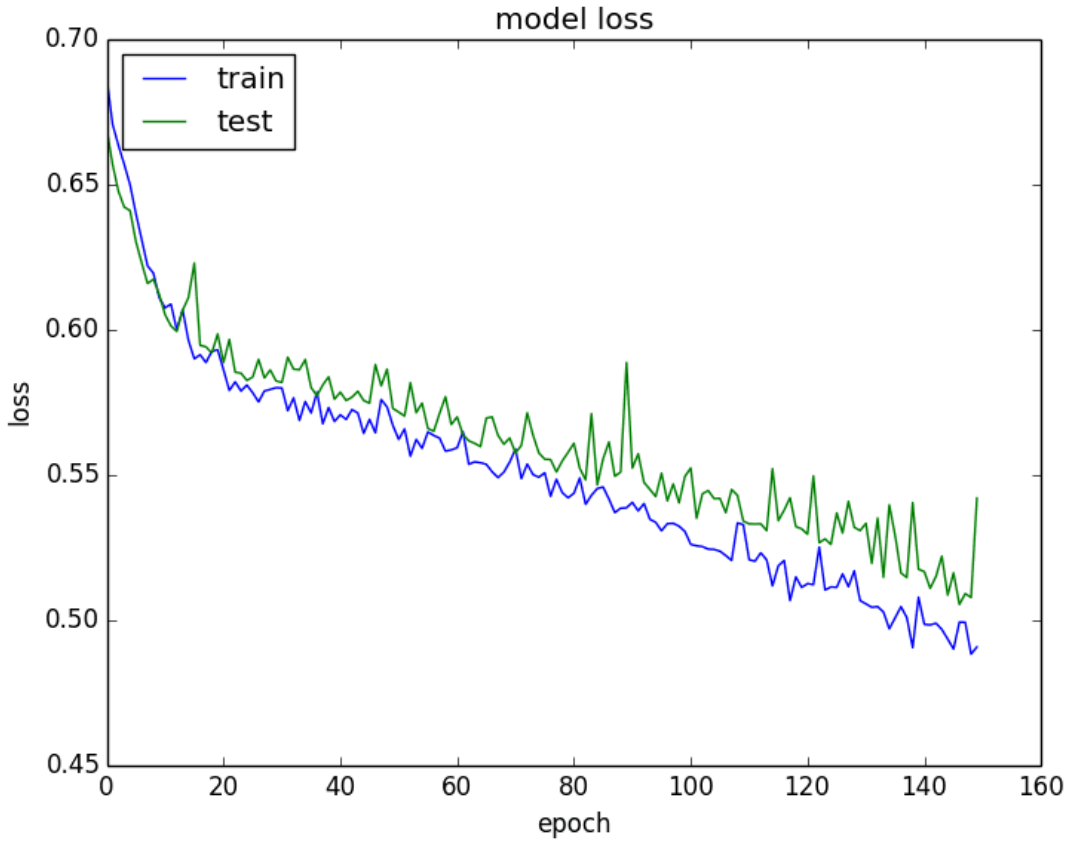
as we try to decrease the bias the variance tends to increase and vice versa

learning curves

number of samples vs accuracy



number of epochs vs loss



model sanity checks: overfitting/underfitting

	overfitting	underfitting
how to detect	test performance much lower than train performance learning curve shows divergence between train and test performance after some point	both train and test performance low learning curve shows potential for increase in train and test performance with more data or more training
root cause	high variance	high bias
remedy	regularization, more data, feature selection, simpler model, bagging	more data, more features, more training, more complex model, boosting

prediction vs inference

prediction	inference
black-box approach used when explainability might not be as important as correct prediction	white-box approach used when explainability (e.g. relation between predictor and target) is important
focus on getting as good model performance metrics as possible while maintaining good generalization capabilities	focus on statistical significance of the coefficients, multicollinearity etc while maintaining good generalization capabilities
sophisticated ML models might help	simple, interpretable statistical models preferred

Typical Modelling Process

- **data preparation**
  - sampling/stratification/imbalance
  - labeling
  - train/val/test split
  - cleaning (imputation, replacement, outliers)
- **exploratory data analysis**
  - target distribution/class imbalance
  - predictors distributions (categorical: boxplot, numerical: scatterplot)
  - correlations/synergies
  - clusters/segments
- **feature engineering**
  - standardization/normalization
  - grouping/bucketing/binning
  - log transforms/capping etc
  - synergies/interactions
  - embeddings/vectorization
- **hyperparameter tuning**

  SGD based models

  - loss
  - regularization
  - learning rate
  - number of epochs

  Tree-based models

  - split criterion
  - max depth
  - min number of samples at leaf
  - number of trees in forest

Neural network models

- loss
- optimization algorithm
- number of layers/number of nodes in each layer
- activation function
- dropout/batch normalization

- **model debugging**
  - overfit (test performance worse than train)
  - underfit (both train and test performance are low and show sign of improving with more data, features, or model complexity)
  - data drift (test distribution different from train)
  - concept drift (relation between predictor and target is changing)
  - data leakage (some information about test data is present in train data)

References

https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

https://developers.google.com/machine-learning

https://airbyte.com/data-engineering-resources/what-is-data-leakage

https://towardsdatascience.com/avoiding-data-leakage-in-timeseries-101-25ea13fcb15f

https://medium.com/@kylejones_47003/data-leakage-lookahead-bias-and-causality-in-time-series-analytics-76e271ba2f6b

MLOps

Basics of System Design

Requirements

functional	non-functional

Estimations

	back-of-envelope calculation
storage	
compute	
network	
others	

Process Flows

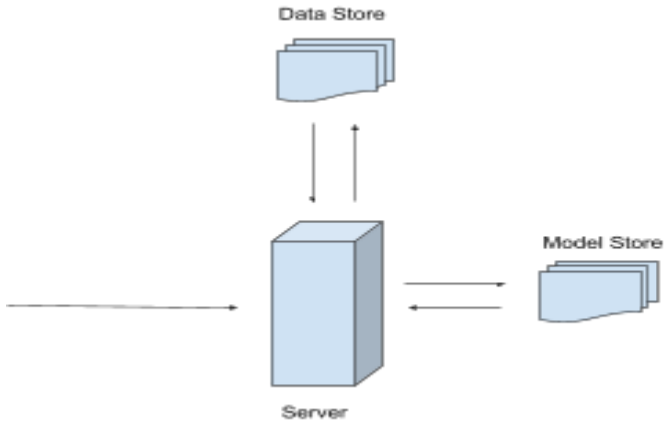
process	steps/flow

System Design Diagram

component	diagram

loosely coupled system => easy to build, maintain, debug, extend, scale, evolve

decouple compute and storage



fundamental building blocks of cloud based systems:

- compute (vm, cluster, serverles functions, serverless containers etc)
- storage (object/block/file, row vs column oriented, sql vs no-sql database etc)
- network (API endpoint, API gateway, load balancer, CDN etc)

- others (database servers, cache servers, distributed message queues, batch schedulers, ML platform & services, other specialized platforms & services like IoT etc)

webservice: request => response

example of ML webservice

	endpoint	flow
data	/prepare	1. request arrives at the server endpoint 2. server program pulls the data from Data Store to RAM 3. server program processes the data 4. server program stores the processed data back in Data Store 5. server program responds with success message
model	/train	1. request arrives at the server endpoint 2. server program pulls the data from Data Store 3. server program trains the model in RAM 4. server program logs the results in the Data Store 5. server program registers the model in Model Store 6. server program responds with success message
action	/score	1. server program pulls the model from Model Store to RAM and keeps it ready for scoring 2. request arrives at the server endpoint 3. server program pulls the data from Data Store to RAM 4. server program obtains the predictions and propensity scores of the model on the data in RAM 5. server program stores the scoring results in Data Store 6. server program responds with success message 7. downstream program takes appropriate action (e.g. auto take down) based on the stored results
impact	/evaluate	1. request arrives at the server endpoint 2. server pulls the data from Data Store to RAM 3. server computes evaluation metrics on the data 4. server stores the results back to Data Store 5. server program responds with success message

Batch
- trigger
- scheduler
- batch queue

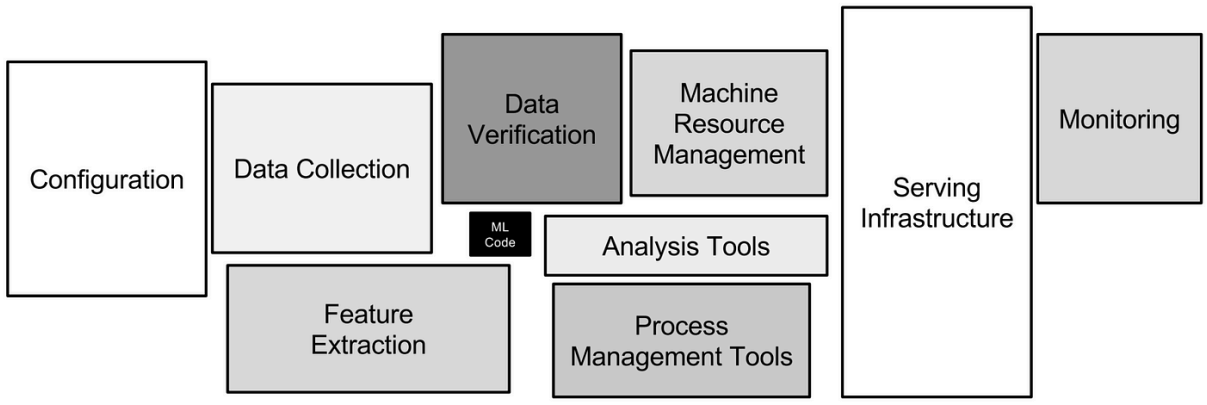
Stream
- publishers
- subscribers

- message queue

Core Functions

function name	description	input	output	file path
prepare_training_data				prepare.ipynb
prepare_scoring_data				
prepare_train_validation_test_split				
load_data				
save_data				
validate				train.ipynb
fit				
score				
evaluate				
load_model				
save_model				

Principles of MLOps



	Data	Model	Code
version control	dvc	mlflow	git

testing	• target distribution • key predictor distributions • out of range value checks	• overfit check • eyeballing sample positive/negative predictions	• unit tests • integration tests • automation using pytest • debug using pdb • profiling time and memory of different steps
modularity	• raw data • processed data • feature store • predictions	• separate propensity prediction and threshold	• prepare • train • score • evaluate
automation	• automatic data profiling upon change	• mlflow experiment tracking	• CI using git hook
deployment	• backup	• web-serving using flask	• CD using docker
monitoring	• distribution drift	• performance metrics • concept drift • bias	• test coverage
documentation	• wiki: data model	• wiki: experiment design • wiki: production models (params, training data, algo etc)	• sphinx • system design diagram
reproducibility	dvc logs	log model params + random seed + data version + code version + docker file	docker + git

Modularization

Version Control

Experiment Tracking

Testing

Deployment/Serving

Containerization

Monitoring

Batch Job

Streaming

Cloud Architecture

API

endpoint

https//www.xyz.com/recommendation/predict

webservice

request → response

request:
{
text: ‘hi how are you’
}

response:
{
is_spam: 0
propensity: 0.2
threshold: 0.5
}

json: nested dictionary + list + string + numericals
{
text: ‘hi how are you’
}

References

https://flask.palletsprojects.com/en/stable/quickstart/
https://www.freecodecamp.org/news/how-to-dockerize-a-flask-app/
https://githooks.com/

Deep Learning

Neural Network Basics

Neural networks are a cornerstone of machine learning and artificial intelligence, inspired by the structure and function of the human brain. Understanding them involves grasping a variety of fundamental concepts, which are crucial for designing, training, and deploying these models effectively. Here are some key concepts:

1. Neurons and Layers
- **Neurons:** The basic unit of computation in a neural network, modeled after biological neurons. Each neuron receives input, processes it, and passes on its output to the next layer.
- **Layers:** Neurons are organized in layers. A typical network has an input layer, one or more hidden layers, and an output layer. The complexity and capacity of the model increase with more layers and neurons.

2. Activation Functions
Activation functions decide whether a neuron should be activated or not. They introduce non-linearity into the model, enabling it to learn complex patterns. Common examples include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax.

3. Forward Propagation
The process of input data passing through the network layers to produce an output. Each neuron applies a weighted sum of its inputs, adds a bias, and then passes the result through an activation function.

4. Backpropagation and Gradient Descent
- **Backpropagation:** A method for computing gradients of the loss function with respect to the weights in the network, enabling learning. It involves propagating the error back through the network, from the output towards the input.

- **Gradient Descent:** An optimization algorithm used to minimize the loss by adjusting the weights and biases. The learning rate is a key parameter that controls how much the weights are adjusted at each step.

5. Loss Functions
Loss functions measure how well the model's predictions match the expected output. The choice of loss function depends on the type of problem (e.g., regression, classification). Common examples include Mean Squared Error (MSE) for regression and Cross-Entropy for classification.

6. Overfitting and Regularization
- **Overfitting:** Occurs when a model learns the training data too well, including the noise, leading to poor performance on unseen data.
- **Regularization:** Techniques like L1 and L2 regularization, dropout, and early stopping are used to prevent overfitting by penalizing large weights or simplifying the model.

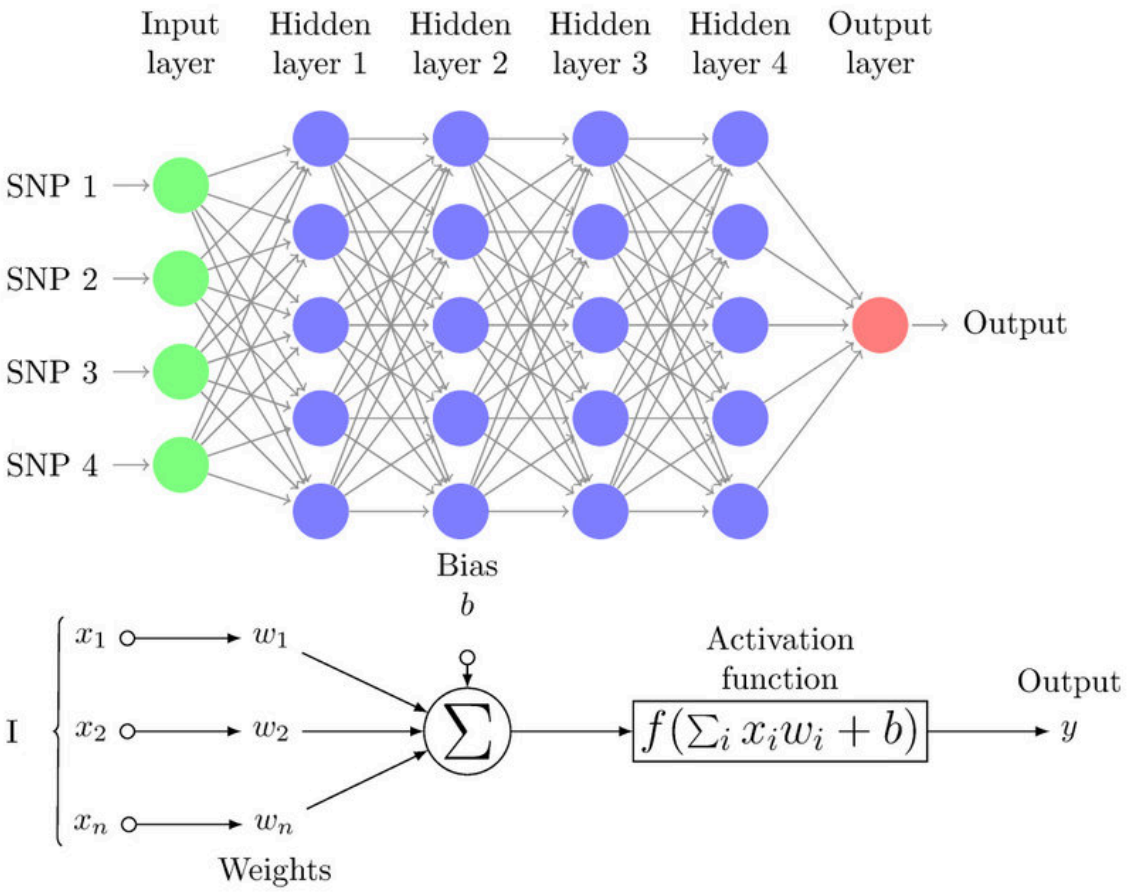
7. Learning Rate
The step size at each iteration while moving toward a minimum of the loss function. Choosing the right learning rate is crucial, as too small a rate can lead to slow convergence, while too large can cause overshooting the minimum.

8. Epochs, Batch Size, and Iterations
- **Epoch:** One forward pass and one backward pass of all the training examples.
- **Batch Size:** The number of training examples in one forward/backward pass. Smaller batch sizes require less memory and can train faster, but with more updates, it might lead to more noise per update.
- **Iteration:** One forward pass and one backward pass using a batch of data.

9. Model Evaluation Metrics
Metrics like accuracy, precision, recall, and F1 score for classification; MSE, RMSE, and MAE for regression; and more specialized ones for tasks like object detection or segmentation.

10. Deep Learning Frameworks
Frameworks such as TensorFlow, PyTorch, Keras, and others provide pre-built functions, layers, and optimizers, making it easier to design, train, and deploy neural networks.

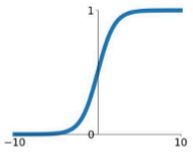
Grasping these concepts provides a strong foundation for delving deeper into neural network architectures, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and more advanced models like Transformers. Each of these architectures is designed to handle specific types of data and learning tasks, from image and video processing to sequence prediction and natural language processing.



Activation Functions

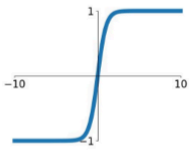
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



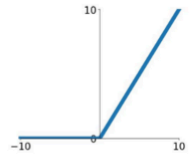
tanh

$\tanh(x)$



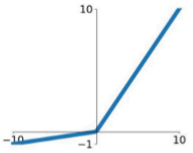
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

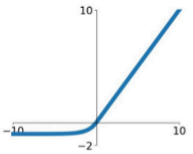


Maxout

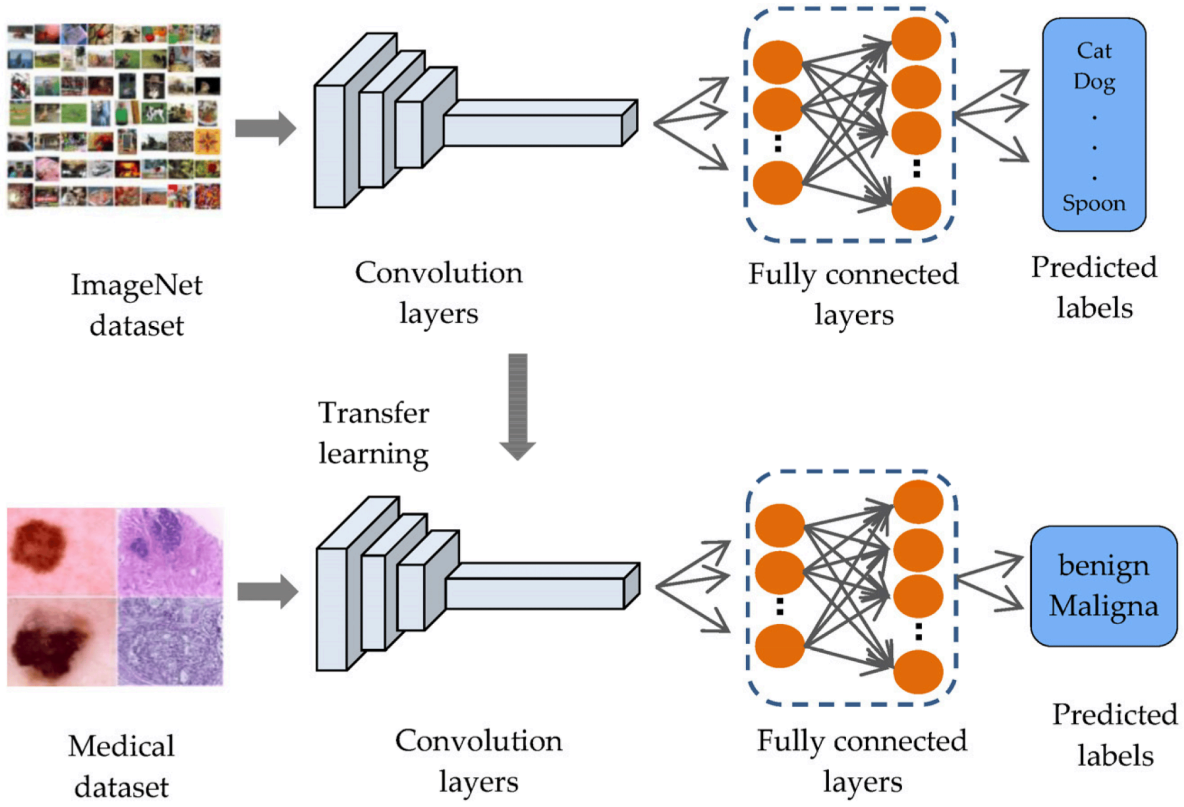
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Transfer Learning



Transfer learning is important for several reasons:

1. **Reduced Data Requirements:**

Transfer learning allows models to be trained effectively with smaller datasets by leveraging knowledge from pre-trained models. This is particularly useful in scenarios where labeled data is scarce or expensive to obtain.

2. **Faster Training:**
By starting with pre-trained models that have already learned useful features or representations, transfer learning can significantly reduce the training time for new models. This is especially beneficial when training deep neural networks, which can be computationally intensive.

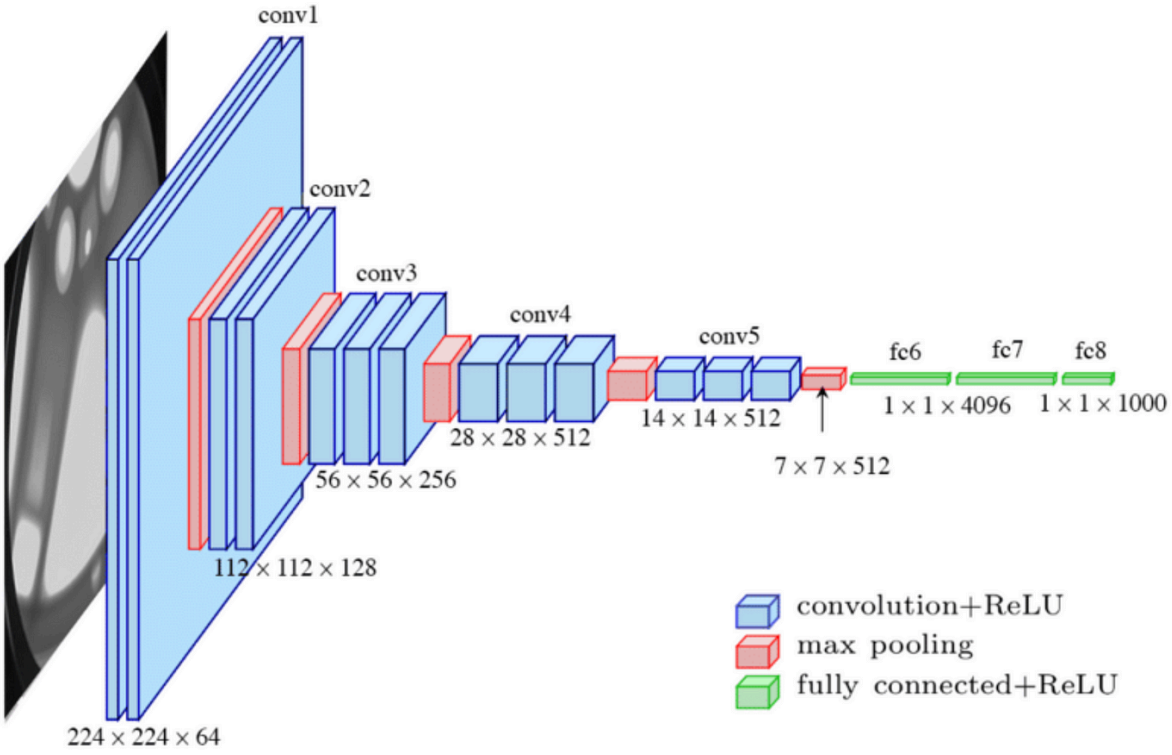
3. **Improved Performance:**
Transfer learning often leads to better performance on target tasks compared to training models from scratch. Pre-trained models have learned rich representations from large-scale datasets, which can capture general patterns and structures in the data. By fine-tuning these representations on target tasks, models can achieve better performance than if they were trained solely on the target data.

4. **Domain Adaptation:**
Transfer learning allows models to adapt to new domains or tasks by leveraging knowledge from related domains or tasks. This is particularly useful when the distribution of the target data differs from the distribution of the data used to train the pre-trained model.

5. **Addressing Overfitting:**
Pre-trained models are often trained on large, diverse datasets, which helps prevent overfitting. By fine-tuning these models on smaller, task-specific datasets, transfer learning can mitigate the risk of overfitting on the target task.

6. **Robustness and Generalization:**
Transfer learning can improve the robustness and generalization of models by transferring knowledge learned from one task to another. This can lead to models that are more adept at handling variations and uncertainties in the data.

Overall, transfer learning is a powerful technique that enables the reuse of learned knowledge across tasks, leading to faster training, improved performance, and more efficient use of resources. It plays a crucial role in advancing the state-of-the-art in machine learning and facilitating the development of AI systems for a wide range of applications.



Rule of thumb for transfer learning

1. closeness of original task to end task
2. availability of data

- end task close to original task and large data not available => use as featurizer without finetuning
- end task close to original task and large data available => try tuning last few layers
- end task not close and large data not available => finetune last few layers freeze the rest
- end task not close and large data available => finetune entire network

Transfer learning is applicable to a wide range of tasks and domains. Here are some example use cases where transfer learning can be particularly beneficial:

1. **Image Classification:**
   - **Use Case:** Training a model to classify images into different categories (e.g., animals, vehicles).

- **Transfer Learning:** Leveraging pre-trained convolutional neural networks (CNNs) trained on large image datasets (e.g., ImageNet) and fine-tuning them on target tasks with smaller labeled datasets.

2. **Object Detection:**
   - **Use Case:** Detecting and localizing objects within images or videos.
   - **Transfer Learning:** Utilizing pre-trained CNNs as feature extractors for object detection frameworks (e.g., Faster R-CNN, YOLO), then fine-tuning them on specific object detection tasks.

3. **Natural Language Processing (NLP):**
   - **Use Case:** Sentiment analysis of text data (e.g., product reviews, social media posts).
   - **Transfer Learning:** Fine-tuning pre-trained language models (e.g., BERT, GPT) on target sentiment analysis tasks with domain-specific text data.

4. **Medical Image Analysis:**
   - **Use Case:** Diagnosing medical conditions from medical images (e.g., X-rays, MRIs).
   - **Transfer Learning:** Using pre-trained CNNs trained on large medical image datasets to extract features from medical images, then fine-tuning them on specific diagnostic tasks with labeled medical image data.

5. **Speech Recognition:**
   - **Use Case:** Converting spoken language into text (e.g., voice commands, transcriptions).
   - **Transfer Learning:** Adapting pre-trained speech recognition models (e.g., DeepSpeech, wav2vec) on target speech recognition tasks with domain-specific audio data.

6. **Autonomous Driving:**
   - **Use Case:** Detecting and classifying objects on the road (e.g., pedestrians, vehicles).
   - **Transfer Learning:** Utilizing pre-trained CNNs for object detection and classification in autonomous driving systems, then fine-tuning them on specific driving environments and conditions.

7. **Recommendation Systems:**
   - **Use Case:** Personalizing recommendations for users in e-commerce platforms or content streaming services.
   - **Transfer Learning:** Leveraging pre-trained recommendation models (e.g., collaborative filtering, matrix factorization) trained on large user-item interaction datasets, then fine-tuning them on specific recommendation tasks with user behavior data.

8. **Biometric Identification:**
   - **Use Case:** Recognizing individuals based on biometric data (e.g., facial images, fingerprints).
   - **Transfer Learning:** Utilizing pre-trained CNNs for feature extraction from biometric data, then fine-tuning them on specific identification tasks with labeled biometric data.

These are just a few examples of how transfer learning can be applied across different domains and tasks to improve model performance, reduce training time, and address data

scarcity issues. Transfer learning has become a standard practice in machine learning and deep learning, enabling the development of more accurate and efficient AI systems for various real-world applications.

https://www.datacamp.com/tutorial/fine-tuning-llama-2

Architectures

CNN
https://medium.com/@senanahmedli89/convolutional-neural-networks-from-scratch-2bd6a421fcc8
https://medium.com/@prathammodi001/convolutional-neural-networks-for-dummies-a-step-by-step-cnn-tutorial-e68f464d608f
https://medium.com/@myringoleMLGOD/simple-convolutional-neural-network-cnn-for-dummies-in-pytorch-a-step-by-step-guide-6f4109f6df80
https://www.kaggle.com/code/iamsouravbanerjee/convolutional-neural-network-for-dummies
https://towardsdatascience.com/a-comprehensible-explanation-of-the-dimensions-in-cnns-841dba49df5e/

Unet: https://towardsdatascience.com/cook-your-first-u-net-in-pytorch-b3297a844cf3/
Encoder-Decoder

(dropout)
https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9/

RNN and LSTM
https://colah.github.io/posts/2015-08-Understanding-LSTMs/
https://dagshub.com/blog/rnn-lstm-bidirectional-lstm/

Transformer
https://jalammar.github.io/illustrated-transformer/
https://towardsdatascience.com/foundations-of-nlp-explained-visually-beam-search-how-it-works-1586b9849a24/

Emerging Methods

LLM

(Discriminative)
BERT
https://jalammar.github.io/illustrated-bert/

(Generative)
https://jalammar.github.io/illustrated-gpt2/
https://jalammar.github.io/illustrated-stable-diffusion/

- GPT (chatgpt)
- diffusion

RAG

https://realpython.com/build-llm-rag-chatbot-with-langchain/
https://python.langchain.com/docs/tutorials/rag/
https://realpython.com/chromadb-vector-database/
https://medium.com/data-science/rag-how-to-talk-to-your-data-eaf5469b83b0
https://aravindkolli.medium.com/unlocking-the-power-of-retrieval-augmented-generation-rag-an-end-to-end-guide-8ee1e58c11c0

https://realpython.com/practical-prompt-engineering/

- vector DB
- chunking
- retrieval
    - meta filter (e.g. using regexes, sql, classifiers etc)
    - query (semantic search)

Knowledge Graphs
https://realpython.com/langgraph-python/
https://towardsdatascience.com/how-to-build-a-graph-rag-app-b323fc33ba06/

- graph RAG
- agentic framework

Efficient AI

(e.g. DeepSeek, Lama2, Gpt4All, Distillation, Quantization, LoRA finetuning)

https://www.mindset.ai/blogs/in-the-loop-ep1-whats-so-great-about-deepseek

https://www.techtarget.com/whatis/feature/DeepSeek-explained-Everything-you-need-to-know
https://www.reddit.com/r/LLMDevs/comments/1ibhpqw/how_was_deepseekr1_built_for_dummies/
https://www.datacamp.com/tutorial/fine-tuning-llama-2
https://blog.streamlit.io/how-to-build-a-llama-2-chatbot/
https://www.reddit.com/r/LocalLLaMA/comments/1dtp71h/gpt4all_30_the_opensource_local_llm_desktop/
https://github.com/nomic-ai/gpt4all
https://achimoraites.medium.com/lightweight-roberta-sequence-classification-fine-tuning-with-lora-using-the-hugging-face-peft-8dd9edf99d19

Final Project Guidelines

- Max 4 people in one group
- Each person has a primary responsibility

For projects, please submit the tentative plan by **20 Mar 2025**

- problem statement
- data (e.g. Kaggle datasets, open source ML datasets, manual curation)
- models
- responsibilities (each person has a primary responsibility)

Final presentation: **6th, 7th, 8th, 9th May 2025 5pm IST - 7pm**

- 15min + 5min for question answers

May 7
https://cmi-ac-in.zoom.us/j/88475377727?pwd=lojgaGU9PaGqy4okZrvaPL0z80DGC3.1

May 9
https://cmi-ac-in.zoom.us/j/83319817523?pwd=e9Xv6fmWCxwYafgdS9J5H2wW1cabpR.1

References:

https://www.projectpro.io/projects/data-science-projects
http://cs230.stanford.edu/past-projects/
https://ocw.mit.edu/courses/15-097-prediction-machine-learning-and-statistics-spring-2012/pages/projects/

Here are some ideas for applied machine learning projects for a class, covering a range of domains and complexity levels:

1. **Image Classification:**
   - Build a model to classify images into different categories, such as recognizing different types of fruits, animals, or vehicles.
   - Create a project to classify X-ray images for medical diagnosis, such as detecting pneumonia or bone fractures.

2. **Natural Language Processing (NLP):**
   - Develop a sentiment analysis tool to analyse customer reviews or social media posts.
   - Build a text summarization tool to summarise news articles or research papers.
   - Create a chatbot for answering questions related to a specific domain or topic.

3. **Time Series Forecasting:**
   - Predict stock prices or cryptocurrency values using historical data.
   - Forecast energy consumption for a building or city based on past usage patterns.
   - Predict weather conditions, such as temperature or precipitation, using historical weather data.

4. **Anomaly Detection:**
   - Build a fraud detection system to detect fraudulent transactions in financial data.
   - Develop a system to detect anomalies in IoT sensor data, such as detecting equipment failures in manufacturing plants.
   - Create a model to detect anomalies in network traffic for cybersecurity applications.

5. **Recommendation Systems:**
   - Build a movie recommendation system based on user preferences and viewing history.
   - Develop a product recommendation system for an e-commerce platform based on past purchases and browsing behaviour.
   - Create a music recommendation system based on user preferences and listening history.

6. **Healthcare Applications:**
   - Predict patient readmission to the hospital using electronic health record (EHR) data.

- Develop a model to predict the risk of developing certain diseases based on genetic information and lifestyle factors.
- Create a system to segment medical images (e.g., MRI or CT scans) for tumour detection or organ segmentation.

7. **Computer Vision:**
- Build an object detection model to detect and localise objects in images or videos.
- Develop a facial recognition system for identity verification or access control.
- Create a project to detect and classify different types of road signs for autonomous driving applications.

8. **Social Good Projects:**
- Develop a model to predict the severity and impact of natural disasters based on historical data.
- Build a system to identify and classify endangered species in wildlife photos for conservation efforts.
- Create a project to detect and classify hate speech or misinformation in social media posts for online safety.

9. **Education and Learning:**
- Develop a model to predict student performance and identify at-risk students based on academic data.
- Build a personalized learning platform that adapts to individual student needs and preferences.
- Create a system to automatically grade student assignments or exams using machine learning algorithms.

10. **IoT and Smart Devices:**
- Develop a smart home system that learns user preferences and adjusts settings accordingly (e.g., temperature, lighting).
- Build a predictive maintenance system for industrial equipment to detect and prevent failures before they occur.
- Create a project to analyze sensor data from wearable devices for health monitoring and fitness tracking.

When choosing a project idea, consider your interests, the available data, and the complexity level appropriate for your class. Additionally, try to select a project that has real-world applications and can make a positive impact in its domain.