Basic Concept

AI, NI, Congitive Function

- AI(artificial intelligence)
  - the intelligence exhibited by machines
- NI(natural intelligence)
  - the intelligence exhibited by humans and animals
- Congitive Function
  - mental capabilities that enable individuals to process information, reason, learn, and solve problems

Elements of NN

- Neurons
  - the basic unit of a neural network
- Activation Function
  - a function that determines the output of a neuron

4 branches of ML

- Supervised Learning

Train a model on **labeled** or annotated examples.

- Unsupervised Learning:

Find patterns in a dataset without **label**

- Self-supervised Learning:

Does not use human-labeled training data but generating labels.

- Reinforcement Learning:

choose action to maximize some reward function base on the env

Division of dataset

- 100 ~ 10k data:
  - 70/30 (70% training, 30% testing)
  - 60/20/20 (60% training, 20% dev, 20% testing)
- 100K data: 90/5/5 (90% training, 5% dev, 5% testing)
- 1M labeled data: 98/1/1 (98% training, 1% dev, 1% testing)
- n millions labeled data, can use 99.5/0.4/0.1

Overfitting vs. Underfitting

- Bias refers to the error that is due to **overly simplistic assumptions** in the learning algorithm.
- **High bias** can lead to **underfitting**

$$\text{bias} = \text{TrainingSet Error}$$

- Variance refers to the error that is due to **excessive complexity** in the learning algorithm.
- **High variance** can lead to **overfitting**

$$\text{variance} = \text{DevSet Error} - \text{TrainingSet Error}$$

Vanishing / Exploding gradients

This occurs in training a deep neural network especially when dealing with **very deep layers**.

- Vanishing gradients can lead to a **slow convergence**

- Exploding gradients can result in very large and **unpredictable updates to the weights**

Weights Regularization

L_1 Regularization: $\frac{\lambda}{m}\ \|w\|_1$ where $\|w\|_1 = \sum(|w_i|)$

L_2 Regularization: $\frac{\lambda}{2m}\ \|w\|_2^2$ where $\|w\|_2^2 = \sum(w_i^2)$

The purpose of this term is to **constrain the model's complexity** by forcing the weights to take on smaller values, which prevent the model from fitting the noise in the training data.

Dropout Regularization

- Randomly set some neurons to zero in each iterations of **training**
- prevent overfitting by **discouraging reliance** on any single feature
- The dropout rate refers to the **fraction of output features that are randomly set to zero** during the training of a neural network

Data Augmentation

- images: flipping, rotating, randomly cropping
- text: cropping, back translation

Data Normalization

- form a **standard normal distribution** (mean = 0, variance = 1) using $x \leftarrow \frac{x-\mu}{\sigma}$

Algorithms for finding the minimum of the cost function

- Gradient Descent

- Gradient Descent with Momentum

$$V_{dw} \leftarrow \beta V_{dW} + (1-\beta)dW$$

$$V_{db} \leftarrow \beta V_{db} + (1-\beta)db$$

$$W \leftarrow W - \alpha V_{dW}$$

$$b \leftarrow b - \alpha V_{db}$$

- Root mean square propogation

- Stochastic gradient descent

Small updates for **large** oscillations, and **large** updates for **small** oscillations.

Learning rate decay

- Same rate for all iterations → wander around the minimum
- LR decay → allow taking smaller steps as it approaches the minimum
- Taking big steps at the beginning and small steps at the end

Optima

In most optima, we have **saddle points** rather than min or max points, because the training data are in high dimensions, you may **descending in one dimensions but ascending in another dimensions**.

Problems near saddle points:

- slope is small, causing the optimization algorithm to take small steps
- solution: **stochastic gradient descent**

Explain the difference between parameters and hyperparameters. Give examples of each.

When do we need to retune hyperparameters?

- When applying a Model to a different application
- When new data is introduced or the model performance degraded

Explain the "panda" versus the "caviar" approach in tuning hyperparameters. In which situation would you use panda? Use caviar?

- "Panda" approach: Train only one model, ajust hyperparameter each day
  - You have lots of data but not much computational resources
- "Caviar" approach: Train many models in **parallel** with different hyperparameters
  - You have lots of computational resources

Briefly explain the main idea in batch normalization. How is batch norm similar to normalizing inputs (C2M1L09)? How are they different?

How does batch norm improve the calculations? Under what circumstance would your use batch norm?

How:
- Speeds up learning (training)
- Slight regularization effect (prevent overfitting)

When:
- new data with different distribution is introduced

Briefly explain softmax. What is it use for?

- Softmax is a mathematical function that transforms a vector of raw scores (logits) into probabilities that sum to one.
- Softmax is typically used in **classification problems**, where it enables models to predict the likelihood of each class given an input.

How do we to calculate softmax?

$$\mathrm{softmax}(x) = \frac{e^{x_i}}{\sum(e^{x_i})}$$

Name some of the deep learning frameworks presented in this class. Which two are used the most today?

- Pytorch, developed by Facebook
- Tensorflow / Keras, developed by Google

C3M1

What is the meaning of perfect precision? What is the meaning of perfect recall?

- Perfect precision means **no false positives**
- Perfect recall means **no false negatives**

When you combine precision and recall, what is that metric called? Give the formula.

The metric that combines precision and recall is called the **F1 Score**.

$$\text{F1 Score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

What is the meaning for “dev set is like setting the target”? What is the consequence of this statement?

- It means the dev set is served as the benchmark to evaluate the performance of model
- Biased dev set $\rightarrow$ biased model performance (overfitting in dev set, poor real-world performance)

How should we divide our labeled data into training / dev / testing if we have a) 1K labeled data b) 100K labeled data c) 1 million labeled data?

Amount labeled data	**Train/Dev/Test %**
100 to 10K	60/20/20
100K	90/5/5
1M	98/1/1

What is Bayes optimal error?

Bayes optimal error is the **lowest possible error rate** for any classifier on a given classification problem with known class distributions.

How do we compute avoidable bias? How do we compute the variance?

$$\text{Avoidable Bias} = \text{Bayes Error} - \text{Training Error}$$
$$\text{Variance} = \text{Dev Error} - \text{Training Error}$$

C3M2

What is the purpose of error analysis? Briefly describe how you would use it? Draw a table to illustrate.

Purpose: understand the underlying causes of the errors by examing the misclassified examples

- Manually examine 100 mistakes (takes 2 hours)
- Categorize errors into meaningful groups
- Count frequency of each error type
- Focus on categories with highest potential impact

Supposed we have 2 difference sets of labeled data from different distributions. One large set is downloaded from the Internet, and a smaller set is specifically made for the app we want to build. How should be divide the data for training, dev and testing?

Total Data:
- 200,000 internet images
- 10,000 app images

Split:
- Training: 200,000 internet + 5,000 app images
- Dev: 2,500 app images
- Test: 2,500 app images

Key Principle:
- Dev/test must reflect future real-world data you want to perform well on
- Don’t randomly mix distributions, even if tempting

C4M1

Know how to calculate convolutions, compute output sizes and the number of parameters in each layer.

image ($W \times H$), kernel ($K_w \times K_h$), padding (P_w and P_h), stride (S_w and S_h)

$$W_{\text{out}} = \left\lfloor \frac{W - K_w + 2P_w}{S_w} + 1 \right\rfloor \qquad H_{\text{out}} = \left\lfloor \frac{H - K_h + 2P_h}{S_h} + 1 \right\rfloor$$

Parameters calculation:

- Conv layer: $K_w \times K_h \times C_{\text{in}} \times C_{\text{out}} + C_{\text{out}} (\text{bias for each filter})$
- Fully connected layer: $N_{\text{in}} \times N_{\text{out}} + N_{\text{out}} (\text{bias for each neuron})$

Why do CONV layers has so few parameters compared to densely connected layers?

- Shared parameters: the same filter is applied to every position in the input
- Sparse connections: each output value depends only on a small number of input values
- Reduce the risk of overfitting: fewer parameters $\rightarrow$ less likely to overfit

Is convolution linear or nonlinear? Is maxpool linear or nonlinear?

Why do we place a maxpool layer between conv layers?

In the architecture for VGG, ResNet, etc., we always place a maxpool layer between conv layers. Why?