NN

Architecture keeps during the training to fit the data

require the use of **ReLU** activation

beginning of training process the parameters are initialized to small random values.

- *NLLLoss*: log probabilities as inputs, softmax as final layer
- *CrosEntropywithLogits*: raw logits as input, softmax applied internally, output the raw scores without activation function called.
- BCELoss: for binary classification tasks and the inputs are probabilities, ideally between 0 and 1, which aligns with a Bernoulli distribution

Activation function:

- f(x) = tanh(x), f(x) = (1 + e^−x)^−1, f(x) = max(0, x)
- LeakyReLU
- ReLU
- Softmax can be used as an activation function in the output layer. Sum(p)=1.
- Linea: x^3

Adam

- optimization techniques: Root Mean Square Propagation, Momentum （Uncentered Variance）
- two moving-average vectors
- adapts the learning rate by scaling it with the moving averages.
- Momentum-like behavior, Efficient memory usage, Robustness to hyperparameters
- each parameter is updated by a custom learning rate.

Adversarial Attacks:

Choose a small perturbation ε on an image x so that a neural network f misclassifies x + ε. (x + ε) = correct class

Non-targeted attack:

Minimize the probability that f (x + ε) = correct class

Targeted attack:

Maximize the probability that f (x + ε) = target class

White-box attacks:

model is known, need to know the architectures and weights of f to optimize ε

Black-box attacks:

Don’t know the architectures and weights of f to optimize ε

Substitute model mimicking target model with known, differentiable function

Adversarial attacks often transfer across models!

ANN

- ***Neurons***

  All neurons in the same layer of an ANN own unique bias.

Autoencoder - learn low dimensional encoding

Backpropagation

- Backpropagation algorithm is used to compute *gradients* of loss function with weight and bias that minimize the loss function.
- Requires the derivative (gradient) of the activation function to update the weights.

- Have regions with zero derivatives can cause issues with backpropagation
- update to a weight connected to an input variable is proportional the value of that variable.
- activation functions: f(x) = tanh(x), f(x) = (1 + e^−x)^−1, f(x) = max(0, x)

Batch gradient Descent

- In batch gradient descent, the weights are updated after the entire training set has been processed.

Bias & Inductive Bias:

- Offset from Origin
- Each kernel will have one bias term
- Modeling Non-zero-centered Data
- Generalization to Higher Dimensions
- *Inductive Bias* (algorithm) uses to predict outputs given unseen data.

CNN - Convolution -> Activation -> Pooling

Parameters = input_channels × output_channels × kernel_size × kernel_size+(output_chan bias)

- kernel : Convolving an input image/feature map having any number of channels with a single kernel produces a feature map with one channel. Kernels of size 3×3 can approximate bigger kernel sizes
- convolutional layer(feed-forward)
  the parameters (weights and biases) are shared(efficient for processing grid-like data like images)
  Adding convolutional layer increases trainable parameters.
  applying a set of learnable filters (convolution kernels) to neighboring patches of the input image.
  don't depend on the input image size
  use **geometric** information within the image
  find features in the image
- maxpooling layer
  Adding **maxpooling** layer decrease trainable parameters. **No parameters**
- Fully connect layer(feed-forward): map features to predictions
- Drop out/Weight decay(L2)/data augumentation used
- **deeper** layers: Kernel access high level, dimension of channels increase, feature map **resolution**(stride) is decreased. height and width decrease.
- **bias** is shared across all the spatial locations within that feature map
- **reuses** convolutional filters for every pixel
- **Pooling layers** reduce the **spatial** resolution of the image
- can use **proxy losses** over intermediate layers to tackle the gradient vanishing problem

---

RNN - Fully-connected NN(token* dim *dim)

use a shared neural network to update hidden state
Reuse the RNN module for every token in the sequence
Keep the context of the previous tokens encoded in the hidden state (h)

Cross entropy

- For classification tasks, output is a probability distribution, This loss combines **nn.LogSoftmax() and nn.NLLLoss()** in one single class.

- The inputs are logits before any activation function like **softmax**.
- Using two probability distributions.
- *Depth* = Length of **sequence**

Drawbacks:

Not good at modelling **long-term** dependencies

Hard to train due to **vanishing/exploding gradients**

Gradient *clipping* → *Exploding* Gradient:

If **gradient** is greater than a threshold, set the gradient to threshold

Skip-connection → *Vanishing* Gradient:

Ideally we want **skip** connections to all previous states → Too expensive

We could preserve the hidden state/context over the long term

Gating Mechanism(Sigmoid or Tanh / A neural network):

skip-connections to all previous states by learning to weight previous states differently instead (soft skip-connections)

use gates that learn to update the **context** selectively

controls how much **informations** flows through.

control symbols to define the Beginning of Sequence <BOS> and End of Sequence <EOS>

RNNs for prediction (Encoder)

- Process tokens one at a time
- Hidden state represents all the tokens read thus far

RNNs for generating sequences (Decoder)

- Generate tokens one at a time
- Hidden state is a representation of all the tokens to be generated

We do this by passing the ground-truth label as the **next input** instead of current prediction.

This trick is known as ***Teacher Forcing - let the model be trained by gradient descent***

diversity not deterministic behavior

greedy approach results in lots of **grammatical** errors

Greedy Search:

selects the token with highest probability

Beam Search:

highest probability within a window

Softmax Temperature Scaling:

Low Temperature (larger logits, more confident)

LSTM - long-term memory (cell state) and a short-term memory (context or hidden state)

Gates: Forget, input, output - emulates skip-connections

Forget : How much of the past memory should we forget

Input: How much the current input should contribute to the memory

updated long-term memory is the amount of past that is remembered (decided by forget gate) combined with the memory that was just created (decided by input gate)

Output Gate: How much of the updated long-term memory should construct the short-term memory

Gated Recurrent Unit (GRU) - more efficient than LSTMs

Combine forget and input gates into an *update gate*

Deep RNNs

stack RNN layers to learn more abstract representations

first layers are better for *syntactic tasks* while representations in last layers perform better on *semantic tasks*.

Word Embeddings - static embeddings(non-contextual)

word2vec model:

Encoder: one-hot embedding→ low-dim embedding

Decoder: low-dim embedding→ nearby words

Skipgram → Predict context from target:

Skip-Gram: Predict context words from target word Skip-Gram components need **not be consecutive** in the text Can be skipped over or **randomly selected** from many documents.

An n-Gram is a **contiguous** sequence of n items from a given text. A k-Skip n-Gram is an n-gram that can involve a skip operation of size k or smaller.

Neighboring words are defined by the window size — a hyper-parameter

- Works well with small datasets
- Better semantic relationships (cat & dog)
- Better representation of less frequent words

CBOW → Predict target from context

Predicts the center word from a fixed window size of context words

- Trains faster than Skip-Gram as the task is simpler
- Better syntactic relationships (cat & cats)
- Better representation of more frequent words

GloVe vectors - enforces global information into the embeddings

co-occurrence **frequency** counts for each word

Optimization: Inner product of word vectors should be a **good predictor** of co-occurrence frequency

! retraining the word vectors may hurt our model if our dataset for the specific task is too small:

Word vectors in training data move around; word vectors not in training data don't move around. Destroys structure of word vector space. Could also phrase as an overfitting or generalization problem.

Evaluate word vectors:

Word Vector Analogies; Word vector distances

and their correlation with human judgments, performance on a downstream NLP task

e.g. name entity recognition: finding a person, location or organization

---

Transformers- self-attention, contextual embeddings(token* token*dim)

Advantage:

every token has a direct connection to every other toke O(1)

Less likely to have gradient vanishing and explosion problem

Fewer training steps, Facilitate long range dependencies

No recurrence, facilitates parallel computation

learns the **edge** weights
focus (attend) on some regions within the input → high **resolution**
learns an **attention score -** importance of different parts of the data, **aggregate** the data
Attention **score:** Dot product, Cosine similarity, Bilinear, MLP

- It retrieves a **value v**i for a query q based on a **key k**i
- Values, queries, and keys are d-dimensional embeddings
- However, rather than retrieving a single value for a query, it uses a **soft retrieval**
- It retrieves all the values but then computes their importance w.r.t query based on the similarity between the query and their keys

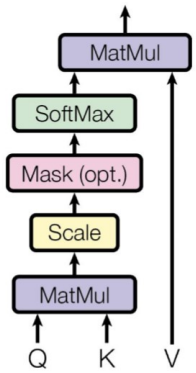
Input: passed through three distinct linear layers (fully connected layers) to produce the keys, queries, and values. Each of these layers has its own set of weights and biases.(**random**)
Simple Attention - **single** score for each embedding

- normalize these scores across all words within the tweet using a **softmax**
- trained end-to-end with the classifier

Pair-wise attention score between all tokens within the input sequence, $\in \mathbb{R}^{n \times n}$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Hidden dimension of Q, K is used to keep the scores from blowing up for large d_k



Multi-Head Attention:

- Divide the representation space to h sub-spaces
- Run parallel linear layers and attentions
- Concatenate them back to form the original space
- Different heads learn different features

Transformer Encoders: Each encoder layer consists of:
1. A multi-head self-attention sub-layer
2. A fully-connected sub-layer
3. A residual connection around each of the two sub-layers followed by layer normalization

Positional Encoding
does not have recurrent or convolutional layers so it **doesn't** into account the **order of sequence**

Attention Taxonomy
Cross-Attention - Between two sequences
Self-Attention - Compute attention of input with respect to itself: for a given token of the input, compute attention weight for all other tokens in the sequence.

BERT - contextual
Input Embeddings
[CLS]→start of the text, and is specific to classification tasks

[SEP]→end of a sentence, or the separation between two sentences
Sentence Embeddings:
belongs to which sentence
Encoder: Softmax
Task:
Masked Word Prediction - Replace **15%** of words, at random, with [MASK]
Next **Sentence** Prediction - Given two sentences, predict if they appear together (50% positive and 50% negative pairs)
Transfer Learning
Vision Transformers (ViT) Compared to CNNs, they achieve higher accuracies on large datasets due to their:

- Higher modeling capacity
- Lower inductive biases
- Global receptive fields

Data augmentations – Rotation,scale,brightness,mirro

- preserve the semantic content
- help with generalization
- increase the data diversity
- given our knowledge of the distribution/domain
- when using MNIST- Mirror results in images that are invalid

Delta rule
gradient descent learning rule for updating the weights of the inputs to artificial neurons in a single-layer neural network.

Encoders
can be used in dimensionality reduction/ create new images..

- Autoencoders construct lower dimensional embeddings, generate outputs similar to the input sample, we throw out the decoder after pre-training for transfer learning.
- Variational autoencoders constrain the network to generate a gaussian distribution (vectors of mean and standard deviation) as its embeddings, impose a distribution prior over the embedding space.
- contrastive loss encoder does not generate embeddings, helpful for discriminations between, loss function is defined over the embedding space

samples.

Dropout

- Used in fully connected layer
- randomly setting weights to 0 and prevent overfitting

Discriminator
Input:image – output: Boolean label
Should be be too good, otherwise can't incrementally improve the generator (no gradients!)

Gans

- LeakyReLU Activations instead of ReLU
- Batch Normalization
- Regularizing discriminator weights & adding noise to discriminator inputs

Cycle GAN:

Cycle loss is reconstruction loss between input to cyclegan andoutput of cyclegan to ensure consistency.

Generator network

- A generative model learns the distribution of the training data
- Allows sampling new samples from the distribution

Input → A noise vector

Output → A generated image

loss function defined by discriminator.(trained)

Unconditional Generative Models:

Only get random noise as input

No control over what category they generate

Conditional Generative Models:

One-hot encoding of the target category + random noise, or

An embedding generated by another model (e.g., from CNN)

User have a high-level control over what the model will generate

---

GNN - predict node/graph classes, links between node

Sets - omit the Positional Encoding from Transformers:

- Learned representation won't change if you randomly shuffle the input tokens
- property is known as order-invariance or permutation-invariance

Deep Sets:

- Learn embeddings for each item → Use a **shared** **neural network** ψ (e.g., MLP, Transformer, etc.) to project each item to a shared space.
- Use an **order-invariant aggregation** function such as sum, mean, max, etc. to aggregate the embeddings into a single embedding
- another neural network ϕ (e.g., MLP) to project the embedding to the final space

- **Degree** of a node is number of edges connecting to that node
- based on Message-Passing → communicate with neighbors to update embeddings

Message-Passing:

1. Aggregate embeddings of its neighbor nodes
2. Combine the aggregated embedding with the node embedding
3. Update the node embedding

Read-out **(graph pooling)** function:

- Aggregates all node embeddings into a graph embedding
- Must be an order invariant function such as sum, mean, max, attention, etc

Different **instantiations of aggregation** function define different GNN models

---

Graph Convolutional Networks

A layer of a GNN is basically a **nonlinear** function over node features and adjacency matrix

$$H = \sigma(AXW)$$

non-linearity — Weight matrix

Limitation:

- Multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself - > **Add self-loops(A= A+I)**
- A is not normalized and therefore the multiplication with A will completely change the scale of the feature vectors -> symmetrically normalize A using diagonal degree matrix D

$$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$$

=> $$H = \sigma\left(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}XW\right)$$

- can influence the embeddings from further neighborhood by **stacking** GCN layers

---

Graph Attention Networks (GAT)

learn an **attention score** between two nodes, learn the **contribution weight** of neighbor nodes:

- Use a **shared** neural network to compute an attention score between two nodes

$$e_{ij} = \mathrm{NN}(h_i, h_j)$$

- Normalize the attention scores

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}$$

- Update the node embeddings based on the **attention score**

$$h_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}h_j\right)$$

DataLoader & Dataset:

- **Dense** implementation: batching is done by creating a diagonal matrix of adjacency matrice
- **Sparse** implementation: uses an index vector which maps each node to its respective graph in the batch

---

Global average pooling

reduce the spatial dimensions (width and height) to 1x1 for each channel while keeping the same depth(1*1*depth)

Gradients

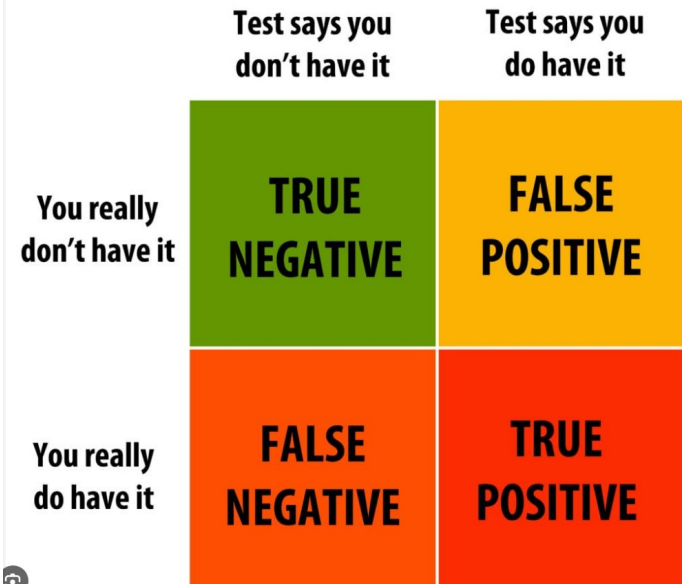
loss.backward() computes the gradients

Loss Function

- Derivative of the loss function with respect to a weight indicates how much the weight is changing the loss.
- Derivative of the loss function with respect to a weight indicates if the weight is increasing or decreasing the loss.
- Derivative of the loss function with respect to an input feature indicates how much the network is paying attention to that feature.
- Binary Cross-Entropy : Learn discriminator weights to maximize the probability that it labels a real image as real and a generated image as fake
- Discriminator: Learn generator weights to maximize the probability that the discriminator labels a generated image as real

Metric

- Recall, or related metrics that emphasize the importance of tracking (minimizing) false negatives, tells us what percentage of positive samples are correctly classified as positive
- Precision focuses on the accuracy of the positive predictions made(actual positive samples out of all the samples)
- F1-score consider both precision and recall - ignore the mistakes made on minority class



Mode Collapse

- generator starts producing the same output – should reject avoid trapped in local optimum
- discriminator is too good/learns faster than generator, mall changes in the generator weights won't change discriminator output,
  and so generator doesn't learn

MSE

- standard loss function for regression
- combined with weight decay (also known as L2 regularization) prevent overfitting

Regularization

- Drop-out
- Adding some noise to gradients
- Weight decay
- Data Augmentation()
- Batch Normalization(Moving averages)
- layer normalization(ineffective for convolutional layers)(feature dimension)(size=1 allowed)

regression model

- output with one neuron
- regression loss function: Mean Squared Error (MSE) / Quadratic Loss / L2 Loss, Mean Absolute Error (MAE) / L1 Loss, Root Mean Squared Error (RMSE), Mean Bias Deviation (MBD)

Skip connection (residual connection)
- commonly used in CNN
- vanishing gradient

Softmax
- Outputs between 0 and 1
- Softmax can be used as an activation function in the output layer. Sum(p)=1.
- Sensitive to Differences

Stochastic gradient descent (SGD)
- estimates the gradient using a single or a small batch of samples
- minimize or maximize an objective function
- additional memory to keep track of the momentum term， utilizes memory at least twice the amount
- stores both the current parameters and the momentum term
- weights are updated for every single training example

Transpose convolution
Used in tasks that require upsampling such as image generation and semantic segmentation.
Padding has the effect of decreasing the output size.
reduces the spatial dimensions of an input, the transpose convolution does the opposite and increases the spatial dimensions.

Unsupervised learning
- data without explicit labels
- learning with a contrastive loss(*Contrastive learning*):
  loss function is defined over the embedding space

variational autoencoder
- It can be used for reducing the dimensionality of the input data
  It can be used to generate new data
  The decoder is used to sample the latent (embedding) code space
- Not deterministic

Vanishing gradient problem
- ReLU non-linearity
- Auxiliary losses
- Skip-connections

Weight:
Number of weights = input channels x output channels x kernel width x kernel height

- **Euclidean Distance** → L2-norm of embeddings

$$D(\vec{X},\vec{Y}) = \| \vec{X} - \vec{Y} \| = \sqrt{\sum_{i=0}^{d} (x_i - y_i)^2}$$

- **Cosine Similarity** → cosine of the angle between embeddings (invariant to magnitude)

$$Sim(\vec{X},\vec{Y}) = \cos(\theta) = \frac{\vec{X}.\vec{Y}}{\| \vec{X} \| \| \vec{Y} \|} = \frac{\sum_{i=0}^{d} x_i y_i}{\sqrt{\sum_{i=0}^{d} x_i^2} \sqrt{\sum_{i=0}^{d} y_i^2}}$$