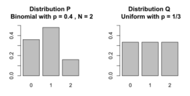
Autoencoders - An autoencoder is a specific type of a neural network, which is mainly designed to encode the input into a compressed and meaningful representation, and then decode it back such that the reconstructed input is similar as possible to the original one. NN-based autoencoders- Can learn nonlinear dependencies. Dropping the non-linear activation functions in the architecture, the AE would achieve the latent representation as PCA. Training a simple AE • Take a sample input data X $f1,f2,f3,,,,fn$ • Input layer will have n number of nodes (number of features in the data) • Encode the input vector $z = f(Wx + b)$ i.e. Regular neural network • Decode the encoded vector (from 2) into output vector • Output vector must contain identical number of nodes as in
the input vector • Compute reconstruction error $\|x - x`\|$ • Apply backpropagation until achieve optimality . What happens if we increase the number of latent hidden vector?
The AE may cheat – will not learn anything through the process. **Solution**: introduce new AE architecture, e.g., Sparce AE, Denoising AE. **Sparse Autoencoders -** A sparse autoencoder is a variant of the traditional autoencoder designed to create sparse representations of the input data during the encoding process. In a sparse autoencoder, the aim is not only to compress the input data but also to enforce sparsity in the learned feature representations. **Key character** - Sparsity Constraint, regularization Techniques, Applications, Interpretability. Sparse AE- • Uses sparsity to create information bottleneck • Uses regularization in the cost function to enforce sparsity
• $x - x` + L1\ Regularization.$ **Deep Autoencoders -** n general, Deep Autoencoders are a type of neural network used for learning efficient codings of unlabeled data. They typically consist of multiple layers of encoders and decoders, which allow them to learn more complex representations compared to simpler, shallow autoencoders. The deep structure (with more layers) enables them to capture multi-level hierarchical features of the input data **Convolutional Autoencoders** are a type of autoencoder that specifically utilizes convolutional layers. In these autoencoders, the encoder part is composed of a stack of Conv2D and MaxPooling2D layers, with max pooling being used for spatial down-sampling. On the other hand, the decoder part consists of a stack of Conv2D and UpSampling2D layers. **Upsampling2D vs. Conv2DTranspose: Upsampling2D**: This method is essentially the opposite of max-pooling but without the max value pooling. It is used to increase the size of the dimension of feature maps (arrays). Upsampling2D is a less computationally intensive operation. **Conv2DTranspose**: This method not only upsamples but also performs convolution. It is used for similar purposes as Upsampling2D, i.e., to increase the size of feature maps, but it adds the convolution process, making it a more computationally expensive operation compared to Upsampling2D.
Both methods are used to increase the dimensionality of feature maps in neural networks, particularly in the context of autoencoders and other deep learning models.
Denoising Autoencoders: are a type of autoencoder that are trained on noisy data with the objective of reconstructing the original, clean data. This training process enables the denoising autoencoder to learn how to remove noise from the input data. They are particularly effective for tasks where the input data is typically corrupted, such as in image denoising, audio signal enhancement, and text denoising. The key idea is that by exposing the autoencoder to corrupted versions of the data and training it to recover the uncorrupted data, it learns a robust representation that is useful for denoising applications. The encoder learns how to remove the noise while training, Lower risk of learning identity function, removing parts of the input rather than adding noise to the input so that the model can learn to predict the original image. **Variational Autoencoders (VAEs): Define an Encoder**: The first step is to define the encoder part of the VAE. This part of the network takes the input data and compresses it into a latent (hidden) space. **Flatten the Encoder's Last Layer**: After the data passes through the encoder, its last layer is flattened. This process transforms the multi-dimensional output of the encoder into a single-dimensional vector. **Define Parameters from the Encoded Input**: From this flattened layer, two critical parameters are derived: the mean and the standard deviation. These parameters are central to the VAE's ability to generate new data points and are used in the reparameterization trick. **Add a Custom Sampling Layer:** A custom sampling layer is then added. This layer uses the mean and standard deviation to sample points from the latent space. The reparameterization trick, a key concept in VAEs, is applied here.
The reparameterization trick allows for the backpropagation of gradients through the sampling process, making it possible to train the VAE effectively using standard stochastic gradient descent methods. This trick is crucial for training VAEs because it allows the model to randomly sample from the latent space while still maintaining differentiability of the mode.
Knowing the distribution of the latent space is important – Instead of encode the data into a fixed latent vector, we can map to a distribution – We can force the latent variables to be normally distributed $N(\mu, \sigma)$ • Now, we can use the mean and std for the decoder phase
– We can quantify the learned distribution while training and the standard normal distribution using KL divergence• Meaning that push the normal distribution as close as possible to standard normal distribution $N(0,1)$. •
Kullback-Leibler Divergence – Measure how one probability distribution p differs from the second q (is not distance between two distribution) – Is not symmetric (meaning that divergence from p to q distribution and divergence from q to p distribution is not identical) $\boldsymbol{DKL\ p\ q \neq DKL\ q \mid p}$)
– Interpretation • $\boldsymbol{DKL\ (p\ |q)}$ – is the information grain achieved when distribution p would be used instead of distribution q



x	0	1	2
Distribution $P(x)$	$\frac{9}{25}$	$\frac{12}{25}$	$\frac{4}{25}$
Distribution $Q(x)$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$

$$\begin{aligned} D_{\mathrm{KL}}(P \parallel Q) &= \sum_{x\in\mathcal{X}} P(x)\ln\left(\frac{P(x)}{Q(x)}\right) \\ &= \frac{9}{25}\ln\left(\frac{9/25}{1/3}\right) + \frac{12}{25}\ln\left(\frac{12/25}{1/3}\right) + \frac{4}{25}\ln\left(\frac{4/25}{1/3}\right) \\ &= \frac{1}{25}\left(32\ln(2) + 55\ln(3) - 50\ln(5)\right) \approx 0.0852996 \end{aligned}$$

$$\begin{aligned} D_{\mathrm{KL}}(Q \parallel P) &= \sum_{x\in\mathcal{X}} Q(x)\ln\left(\frac{Q(x)}{P(x)}\right) \\ &= \frac{1}{3}\ln\left(\frac{1/3}{9/25}\right) + \frac{1}{3}\ln\left(\frac{1/3}{12/25}\right) + \frac{1}{3}\ln\left(\frac{1/3}{4/25}\right) \\ &= \frac{1}{3}\left(-4\ln(2) - 6\ln(3) + 6\ln(5)\right) \approx 0.097455 \end{aligned}$$

There is no condition on the mean and standard deviation vector➔While training, the encoder may produce vary different means μ that are far away from each other for different classes with narrow std σ • However, we want overlapping of distribution of different classes • Regularize both the variance and means that are returned by the encoder • Therefore, we introduce KL divergence into the loss function➔works as a regularizes • If the encoder tries to clustering the encodings into different regions, then the model will be penalized using the KL divergence
Initially, the AE was in deterministic, however, the VAE becomes probabilistic (stochastic). Backpropagation cannot be performed due to the stochastic nature of the sampling layer: z. The reparameterization trick allows us to restructure how we take the derivative of the loss function in order to optimize our approximate distribution
1. Define an encoder 2. Flatten the encoder last layer 3. Define parameters from the encoded input 1. Mean 2. Standard deviation 4. Add a custom sampling layer 1. Define a function for Reparameterization trick 1. Input: mean and std 2. Output: $zi = \mu + \sigma * \epsilon$ 1. $\epsilon \sim N$ 0,1 5. Define your decoder 6. Define loss function 1. Reconstruction loss + KL divergence Loss 7. Train model
AE for Anomaly Detection: Steps: 1. Develop a AE model 2. Compute the reconstruction error 3. Select a threshold and find the anomalies based on the threshold
Generative Adversarial Network (GAN): GANs consist of two main components: a generator and a discriminator. These two components are in a competitive relationship, often described as a zero-sum game. The generator's role is to generate new data samples (e.g., images). It tries to create data that is indistinguishable from real data. The discriminator's role is to distinguish between real data and the fake data generated by the generator. Essentially, it tries not to be fooled by the generator. GANs can be built using Artificial Neural Networks (ANNs) or Convolutional Neural Networks (CNNs). The key distinction highlighted in your slide is between generative and discriminative models. Discriminative models can't model the probability of seeing a certain image and thus can't generate new samples. In contrast, generative models, like GANs, can model this probability, allowing them to generate new images. **Generative vs. Discriminative Algorithms:** Discriminative Models: These models focus on learning the boundary between different classes. They directly learn the conditional probability distribution p(y|x), which represents the probability of a label y given an observation x. Discriminative models are typically used for classification tasks where the goal is to accurately categorize incoming data into predefined classes. Generative Models: In contrast, generative models learn the distribution of individual classes. They aim to model the joint probability distribution p(x,y), capturing the distribution of both the features x and the labels y
y. Generative models are capable of generating new samples that are similar to the training data, hence the term 'generative'. They are used in tasks where the generation of new data instances is required, such as in image, text, or audio synthesis.
In essence, discriminative models are more concerned with distinguishing between different classes, while generative models focus on understanding and replicating the underlying data distribution.
• Train discriminator to discriminate between real vs fake data • Train the generator to fake data that can fool the discriminator • Continue the generator and discriminator process for many epochs
• Save generator model to generate new data (fake data) close to real data. When training discriminator, hold generator constant and while training generator hold the discriminator constant

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
        $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right].$
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Application
• Generating fake/synthetic data • Generate faces
• Image to image translation
• Text to image translation
• Super resolution
Conditional Generative Adversarial Network (cGAN): Control Over Output: While a standard GAN can generate random images from a given dataset, it doesn't offer control over the types of images generated. cGANs address this limitation by allowing for conditional generation of images. **Conditioning on Class Labels:** In cGANs, both the generator and the discriminator are conditioned during training. This conditioning is typically done using class labels. By conditioning on these labels, cGANs can target the generation of specific types of images. **Training Process**: During training, the generator in a cGAN tries to produce realistic examples for each label in the training dataset. This targeted approach allows for the generation of diverse and specific types of images, guided by the conditioning labels.
• For the dropout layer, specify a dropout probability of 0.75. • For the convolution layers, specify 5-by-5 filters with an increasing number of filters for each layer. Also specify a stride of 2 and padding of the output on each edge. • For the leaky ReLU layers, specify a scale of 0.2.
• For the final layer, specify a convolution layer with one 4-by-4 filter. **pix2pix image translation:** • cGANs are suitable for image-to-image translation • U-net based architecture is used as a generator
• Convolutional PatchGAN is used as a discriminator
• Training – alternating between one gradient descent step on D then one step on G
Word Embedding: • Word embeddings are dense vector representations of words in a high dimensional space – Each word is represented by a numerical values of vector that captures context of the word • Primary goal is to capture the semantic relationship between words – "A word is characterized by the company it keeps" (1957, John) – Popular word embeddings – Word2Vec, Glove, FastText– Shown significant improvement over traditional methods (numerical representation of text data) like BoWs, TFIDF,Word embeddings are learned from large corpora of text using DL – Can be pretrained and fine tuned on specific tasks to achive better performance. **Numerical representation of Text: • Bag of Words** – Binary BoW: Document-term incidence matrix – Frequency BoW: frequency of a word in a document. **Limitations of BoW** • Vocabulary size would increase if new documents contain new words- length of vectors will be increased as well • Curse of dimensionality •Will generate a highly sparce matrix •No grammar or sequential information is preserved • Lack of semantic relationship among words. • **N-grams** are contiguous sequence of n items (characters/ words/ other units) • Unigrams: 1- grams; Bigrams: 2-grams; trigrams: 3-grams • Can capture the context of the words in a sentence • Limitations – •Limited semantic understanding •Sparse representation • Limited context. • **TF-IDF** stands for Term Frequency-Inverse Document Frequency • It can tell how important a word is to a document in a corpus • It is calculated by multiplying the term frequency (TF) of a word by the inverse document frequency (IDF) of the same word • TF measures how frequently a word appears in a document • IDF measures how unique or rare the word is across the entire corpus
• The result is a numerical representation of the importance of a word in a document, which can be used as a feature for machine learning models . **Advantages of TF-IDF:** •Unlike BoW, which treats all words as equally important, TF-IDF considers the rarity and importance of each word in the document •TF-IDF can help to address the problem of stop words, which are common words that do not carry much meaning but can appear frequently in a document. **Limitations**: –It computes document similarity directly in the word-count space, which be slow for large vocabularies – It makes no use of semantic similarities between words – In addition, TF-IDF may not perform well for short documents, as the limited number of words may not provide sufficient information for calculating the TF-IDF values. **Text preprocessing:** Removing Punctuation, Removing stop words, Removing Contraction, Lower Casing, Stemming, Lemmatization. **Semantic relations among words: 1- Synonyms:** Words that share the same or highly similar meanings. While often interchangeable, they can have nuances in different contexts. For example, "big" and "large" are synonyms but may be used differently.
2- Hyponymy-Hypernymy: This refers to hierarchical relationships where hypernyms encompass specific hyponyms. An example is "fruit" as a hypernym for "apple" and "banana."
3- Meronymy-Holonymy: These are part-whole relationships where meronyms represent parts of a whole, and holonyms represent the entire entity. For example, "wheel" is a meronym of "car." **4-Antonymy:** Words with opposite meanings, like "hot" and "cold." **5-Polysemy:** This involves one word having multiple related senses, such as "bank," which can refer to a financial institution or a riverbank. Understanding these semantic relationships is crucial for various NLP tasks, including machine translation and sentiment analysis, as it aids in comprehending the complex and varied meanings of words in different contexts. **Dependency Relations: Words and Their Roles** • how words depend on one another within a sentence • Example: In the sentence "The cat chases the mouse," "cat" is the subject, and "chases" is the verb, representing a subject-verb dependency. **Word Order: Syntax and Sentence Structure** • Different languages exhibit distinct word order patterns. Understanding word order is essential in syntactic analysis. • Example: English follows a subject-verb-object (SVO) order, while Japanese uses a subject-object-verb (SOV) structure. **Application -** Understanding syntactic relations is fundamental for parsing, machine translation, and grammar correction tasks in NLP. **Word Embedding:** • Words can have different qualities or features that describe their meaning, context, or usage in language. • One common approach to represent words with their features is through a vector representation - word embedding • a numerical representation of words in a vector space, where each dimension of the vector corresponds to a feature or quality of the word • Each feature in the word embedding can have a different weight or importance, depending on the context or application of the embedding • These weights or importance in word embedding will be learned through NN training. **Word Embedding – Word2Vec:** – A neural network based approach to word embeddings – Continuous vector representation of words from very large datasets • Learns to represent each word in a high dimensional space by predicting its context in a training corpus – It generates two types of models: Continuous Bag of Words (CBoW) and Skip-gram • CBoW: predicts target word on its context words • Skip-gram: predicts context words given a target word – Learn high-quality word vectors from a 1.6 billion words of datasets – Provided good results for syntactic and semantic word similarities**. Advantages:** Word2Vec can capture semantic relationships between words, such as analogies (e.g., "king" is to "queen" as "man" is to "woman"). However, Word2Vec has **limitations** - unable to capture polysemy (words with multiple meanings) and being sensitive to training data biases • For instance, if the word "bank" appears frequently with words related to finance in the training data, then the resulting vector representation for "bank" may be biased towards its financial meaning and not capture its meaning related to a river • Association of certain words: nurse and women or doctor and man. Extensions and variations of Word2Vec: GloVe, FastText, and Doc2Vec • address some of these limitations and provide additional functionality for NLP tasks. GloVe (Global Vectors for Word Representation) is another popular technique for generating word embeddings. Similar to Word2Vec - generates dense vector representations; however, GloVe uses a different approach • The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus.
• The co-occurrence matrix is typically based on a large corpus of text, and it captures the distributional information of words in the context of the corpus. • This approach allows GloVe to capture the meaning of words based on their usage patterns across the entire corpus, rather than just in local contexts**. Recurrent Neural Network –** Motivation- - RNNs primarily used for time series analysis, sequential data analysis (NLP), – Sequential information is needed when we predict based on previous data along with current data – Stock price prediction • We not only need previous day data, but also require to observe trend of the data. – Text generation • Prediction next word – for that we not only need the previous word, but also need to look at the whole sentence to understand the context. • Feeding sequential data to regular NN encounter difficulties – Input and output length can be different for different examples – Does not share features learned across different position of text – Very large input layer➔enormous number of parameter to learn. **Recurrent Neural Network – Activations:** • Sigmoid – leads to vanishing gradient issue, • ReLU – may lead to exploding gradient issue.
Exploding gradient: – Applying Batch Normalization is tricky for

RNN • Solution: **Gradient Clipping** - Capping gradient values if it goes above a predefined minimum or maximum value. **Vanishing gradient** – Solution: LSTM / GRU. **RNNs** – only use information from the previous time steps, not from the future time steps **– Solution**: Bidirectional RNN. **Long Short-Term Memory (LSTM)** networks are a type of Recurrent Neural Network (RNN) specifically designed to address the problem of learning long-term dependencies. Traditional RNNs struggle with this due to the vanishing and exploding gradient problems. LSTMs tackle these issues with a unique architecture that includes components like memory cells, input gates, output gates, and forget gates. These elements work together to regulate the flow of information, allowing the network to remember important information over long sequences and forget irrelevant data. This design makes LSTMs particularly well-suited for tasks in NLP where understanding the context over long sequences is crucial, such as in language modeling, text generation, and machine translation. **The Gated Recurrent Unit (GRU**) is a variant of the LSTM designed to be simpler and more efficient. Key features of GRUs are: **Update Gate**: Determines how much past information is retained. It's a combination of the forget and input gates in an LSTM. **Reset Gate**: Controls how much past information is forgotten.**Candidate Hidden State**: Represents the current memory content, which is a blend of new information with the past memory. **Hidden State:** Decides what information to carry forward using the update gate. Unlike LSTMs, GRUs do not have a separate cell state and combine the forget and input gates, streamlining the process of information flow and decision-making about what to remember and what to forget. This design simplifies the architecture and often leads to faster training times, making GRUs a popular choice for tasks that require processing sequential data. **What happens if the value of update gate around 0 or 1?: if the update gate value is close to 0:** It means the gate is letting almost none of the past information through. This situation is akin to the model 'forgetting' what it has learned previously. The GRU behaves almost like it's starting fresh, with minimal influence from past data. **If the update gate value is close to 1**: In this case, the gate is allowing almost all the past information to pass through. This means the GRU retains much of the past data, effectively remembering more of the historical context. This can be useful when the sequence's historical information is crucial for understanding its future. **Embedding Layer in Deep Learning (Keras): Purpose**: The Embedding Layer is used for dimension expansion. It is a way to represent text data in neural networks, where each word/token is represented by a unique vector in a higher-dimensional space. **Learning Weights**: Weights of the embedding are learned during the training process. This means the layer automatically learns the best way to represent words in the vector space based on the training data. **Representation of Similar Words:** One of the key features of the Embedding Layer is its ability to represent similar words with similar encoding. This is achieved through the training process, which adjusts the vectors of words based on their context and usage in the training data. **Preparation of Text Data:** To utilize the Embedding Layer, each token (word) in the text data needs to be encoded to a unique integer. This is typically done through tokenization and indexing, where each unique word is assigned a specific integer.

The Embedding Layer is a crucial part of many natural language processing models as it allows for a more nuanced and effective representation of language compared to simpler methods like Bag of Words or one-hot encoding**. Sequence 2 Sequence Model:** is a type of model used in machine learning, particularly in the field of Natural Language Processing (NLP). It's designed to convert sequences from one domain (e.g., sentences in English) to sequences in another domain (e.g., sentences in French), making it highly effective for tasks like machine translation, text summarization, question answering, and more**. Components- Encoder**: The first part of a Seq2Seq model is the encoder. It processes the input sequence (e.g., a sentence in the source language) and compresses the information into a context vector (also known as the state vector). This vector aims to encapsulate the information from the input sequence for the decoder. **Decoder**: The second part is the decoder. It takes the context vector from the encoder and generates the output sequence (e.g., a sentence in the target language). The decoder generates the sequence step-by-step, often one word or token at a time. **Working Mechanism: Input Processing:** The input sequence is fed into the encoder, often after being converted into a suitable format like a sequence of word embeddings. **Context Vector Generation:** The encoder processes the entire input sequence and converts it into a context vector, a fixed-size representation that captures the essence of the input. **Sequence Generation:** The decoder then takes this vector and begins generating the output sequence, using the context vector to help in determining the appropriate output at each step. **Output**: The output sequence is constructed incrementally, with the decoder deciding when to stop (often marked by a special end-of-sequence token). **Applications**: Machine Translation, Text Summarization, question answering, speech recognition. Limilation: Context Vector Limitation, Long Sequences, Variability in Outputs. **a Sequence 2 Sequence Model for Image Captioning** typically involves using a convolutional neural network (CNN) to extract features from an image, which are then fed into a recurrent neural network (RNN) or a similar sequence model. This RNN generates a sequence of words to form a caption that describes the contents of the image. This process combines techniques from both computer vision and natural language processing to create meaningful descriptions for images**. Challenges with Long Sentences**: Memory Limitations, Vanishing and Exploding Gradients, Attention Mechanism Limitation, Contextual Understanding, Resource Intensiveness, Ambiguity and Noise. **Transformer: Key Features- Attention Mechanism:** The core of the Transformer is the attention mechanism, specifically the self-attention mechanism, which allows the model to weigh the importance of different parts of the input data differently.**No Recurrence or Convolution**: Unlike its predecessors (RNNs, LSTMs, GRUs), the Transformer does not use recurrent or convolutional layers. This allows for significantly more parallelization during training and thus faster processing.**Positional Encodings**: Since the model does not inherently process sequential data in order, it uses positional encodings to maintain the order of the input data.**Encoder-Decoder Architecture**: The Transformer follows an encoder-decoder structure. The encoder processes the input data and the decoder generates the output. Each consists of multiple identical layers.**Multi-Head Attention**: The model utilizes multi-head attention in both the encoder and decoder, allowing the model to jointly attend to information from different representation subspaces at different positions.**Feed-Forward Networks**: Each layer in the encoder and decoder contains a fully connected feed-forward network which is applied to each position separately and identically. **Applications**:**Machine Translation**: The Transformer's initial and most notable use case. **Text Summarization**: Generating concise summaries of longer texts.**Question Answering**: Answering questions based on a given context.**Text Generation**: Generating coherent and contextually relevant text. **Advantages:Parallelization:** Enables faster training compared to sequence-based models.**Handling Long-Range Dependencies**: More effective in capturing long-range dependencies in text due to the self-attention mechanism. **Challenges**: Resource intensive, Interpretability. **BERT: Based on Transformer Encoder Layers**: BERT is built upon multiple layers of the transformer encoder. This reflects its primary focus on processing and understanding input text, rather than generating text, which is the typical role of the decoder in a transformer model. **Absence of Transformer Decoder**: Unlike the full Transformer model, which includes both encoder and decoder components, BERT omits the transformer decoder. This design choice aligns with BERT's role as a model for encoding and interpreting text, rather than for sequence generation tasks. **Language Model for Generating Word Embeddings**: BERT is fundamentally a language model trained initially to produce word embeddings. These embeddings are rich, context-aware representations of words, capturing their meanings based on their contexts in sentences. **Context Awareness and Bidirectionality**: One of the hallmark features of BERT is its bidirectional nature. Traditional language models typically process text in a single direction (left-to-right or right-to-left), but BERT reads the entire sequence of words at once. This allows it to capture the context from both directions, leading to a more nuanced understanding of the language. **Vision Transformer (ViT): Pretraining and Fine-Tuning**: The ViT model is initially pretrained on a large dataset (Dataset A) and then fine-tuned and evaluated on a different dataset (Dataset B). This approach is common in deep learning, where a model is first trained on a comprehensive dataset to learn general features and then fine-tuned on a more specific dataset for particular tasks. **Performance on Different Datasets:** When pretrained on ImageNet (a smaller dataset), ViT performs slightly worse than ResNet, a popular convolutional neural network model used for image processing tasks. On ImageNet-21K, which is a medium-sized dataset, ViT's performance is comparable to that of ResNet. When pretrained on JFT, a large dataset, ViT slightly outperforms ResNet. These points suggest that the Vision Transformer's performance relative to traditional convolutional neural networks like ResNet varies depending on the size and nature of the dataset used for pretraining. Generally, ViT shows more promising results when trained on larger datasets. ViT represents an adaptation of the Transformer architecture, originally designed for NLP tasks, for computer vision. It processes images not as grids of pixels (like traditional CNNs) but as sequences of patches, allowing it to apply the self-attention mechanism of Transformers to visual data. This approach has been groundbreaking in showing that attention-based models can be highly effective in domains beyond natural language processing**. Siamese Network: Shared Weights**: A Siamese Neural Network (NN) uses the same weights for two different input vectors. This means that two separate inputs are processed by the same neural network to ensure that the same kind of transformations and feature extractions are applied to both. **Encoding Parameters**: The parameters of the neural network define encoding for the inputs. In the context of images, which Siamese Networks are often used for, this would mean encoding the visual features of the images in a way that can be compared. **Learning Parameters for Similarity Measurement**: The learning objective of a Siamese Network is to adjust its parameters so that the distance (often measured in terms of a metric like Euclidean distance) between the encodings is small if the inputs are similar and large if they are different. This makes Siamese Networks particularly useful for tasks like image verification or comparison, where the goal is to determine how similar or different two inputs are. **Triplet Loss:** is a technique used in machine learning, particularly in the context of training models to understand similarities and differences between inputs. It's commonly used in applications like face recognition and image retrieval. The key concept of Triplet Loss involves three parts: **Anchor**: This is a reference input. In the context of image processing, it could be a specific image. **Positive**: This is another input that is similar to the anchor. For instance, another image of the same person in face recognition. **Negative**: This is an input that is different from the anchor. In face recognition, this would be an image of a different person. The goal of Triplet Loss is to ensure that the distance (in terms of the features learned by a neural network) between the anchor and the positive is smaller than the distance between the anchor and the negative. This helps the model learn to distinguish between similar and dissimilar inputs. The loss function is typically formulated in a way that penalizes the model if the positive is too far from the anchor or if the negative is too close to the anchor. By minimizing this loss during training, the model learns to effectively encode similarities and differences between inputs.Triplet Loss is particularly effective in scenarios where the distinction between different classes (like different people in face recognition) is subtle and requires learning fine-grained differences. This approach helps in learning robust feature representations that are useful for comparison tasks. **Topic Modeling: Purpose of Topic Models**: They are used to discover latent topics in collections of documents. These models help uncover common themes and the underlying narrative in text data. **Powerful Unsupervised Tool**: Topic models are considered a powerful unsupervised tool in text analysis, meaning they do not require labeled data for training. **Conventional Models**: **Latent Dirichlet Allocation (LDA):** LDA is a widely used topic modeling technique that assumes each document is a mixture of a small number of topics and that each word in the document is attributable to one of the document's topics.**Non-Negative Matrix Factorization (NMF):** NMF is another method used in topic modeling. It factorizes the given document-term matrix into two lower-dimensional matrices, revealing the latent structure in the data. **Bag-of-Words Representation**: Both LDA and NMF usually describe a document as a bag-of-words, meaning they focus on the frequency of words in the document while disregarding the order of words. **Limitation**: A significant limitation of these models is that they ignore the semantic relationships among words due to the bag-of-words representations. Since these models do not account for the context of words within sentences, they may fail to accurately represent the true meaning or themes of the documents**. GPT Versions and Focus: GPT**-1: Aimed at improving language understanding. GPT-2: Described as unsupervised multitask learners. GPT-3: Characterized as few-shot learners, capable of learning from a minimal number of examples. **Architecture of GPT**:GPT models remove the transformer encoder and the cross-attention mechanism in the decoder. This is a significant departure from the typical Transformer architecture, which includes both an encoder and a decoder. **Training Process:** GPT models are trained in a self-supervised manner. This means they use their own data for learning, rather than relying on external labels.Specifically, they are pre-trained to predict the next word in a sequence, a task that helps the model learn contextual relationships between words. After pre-training, GPT models can be fine-tuned for specific downstream tasks such as question answering, chatbots, or sentence classification. **PT-2's Zero-Shot Prediction:** Unlike GPT-1, GPT-2 does not necessarily require fine-tuning for downstream tasks. It has the capability for zero-shot prediction, meaning it can perform certain tasks without specific task-focused training. The GPT series represents a significant development in the field of NLP, with each iteration bringing advancements in language understanding and the ability to handle a wide range of language-related tasks with minimal task-specific training. ---

1.Siamese Nets: Take the arguments of the fn for h1 and h2 to be z1 and z2, respectively. δ1 = ∇h1 J = (h1 − h2) => **δ2** = ∇h2 J = −(h1 − h2) => **δ3** = ∇z1 J = δ1 ◦ h1 ◦ (1 − h1) => δ4 = ∇z2 J = δ2 ◦ h2 ◦ (1 − h2) => **∇W J** = δ3x T 1 + δ4x T 2 + λW => ∇bJ = δ3 + δ4 2.(Vanilla) gradient descent update rule: Wt+1 = Wt − α∇W J => bt+1 = bt − α∇bJ 3. W ∈ R 10×5 and b ∈ R 10×1 W,b are the model parameters. 50 +10 = 60. (Note: we don't address hyperparameter tuning in this question, so λ is not updated.) 4. **Skip-gram**- Diff: Not classification, but regression problem - Diff: Compares word vectors with each other instead of predicting context words. - Sim: Tries to learn word vector representations (similar input) 5. **Find** ∇h1 J, ∇h2 J, and ∇xJ. Take the arguments of the fn for h1, h2, and yˆ to be z1, z2, and z3, respectively. δ1 = ∇z3 J = yˆ − y, δ2 = ∇h1 J = ∇h2 J = WT 3 δ1. Soln for the gradient w.r.t a single word vec or w.r.t. a batch of word vec were accepted. δ3 = ∇z1 J = δ2 ◦ h1 ◦ (1 − h1), δ4 = ∇z2 J = δ2 ◦ 1{z2 > 0}, ∇xJ = W1T δ3 + W2T δ4 5. Which is likely to train faster, W1 or W2? W2: since the ReLU function gradient does not saturate at high values of the input, vanishing gradient is not as much of an issue for this parameter. Since we are training these parameters together, computational efficiency has little effect on training efficiency. **Dropout** is a stochastic regularization technique where with probability p a neuron in a neural network is kept alive. Neurons are dropped during training (during both forward and back propagation). To be more thorough, if you have a hidden layer h => h = f(W x + b) then dropout is an element-wise multiplication of a binary mask matrix m where mj ∼ Bernoulli(p) => hˆ = m ◦ f(W x + b) 1. True since f(0) = 0 for tanh, ReLU and not for non-centered sigmoid. 3. L2-penalized update is equivalent to the SGD update. 5. If the update gate zt is close to 0=> True. In this case, ht ≈ ht−1 6. If the update gate zt is close to 1 => False. In this case, ht depends strongly on input xt and not on ht−1. 2. **dMatMul:** Because they may be used in the expression for gradient. case i) The i/p=> If h 0 = Wx, and we already know δ = ∂J/ ∂h'. Then ∂J/∂W' = δxT. So to calculate ∂J/∂W' we need the value of x, which is input to the MatMul node. These answers are also correct: ReLU, CE. Case ii) the o/p => Because σ'(x) = σ(x)(1 − σ(x)), calculating backpropagation of sigmoid function would require the output of sigmoid node. These answers are also correct: ReLU, tanh.

3. The CE node (softmax cross-entropy) in the graph expects unscaled inputs=> Recall that the gradient of J with respect to the unexponentiated inputs to CE node is ˆy−y. This is very easy to compute. We also give full credits to these reasons: 1. Don't have to first take exp and then take log. Improve numerical stability. 2. y is one hot so we only need to compute ˆyk where k is the correct label. 8. multi-layer neural network model trained using stochastic gradient…**False**, 9. parses a sentence.. RIGHT-ARC…**True**, 10. Stochastic gradient descent results in a smoother convergence plot….**False.** Q8. if you had class imbalance-> One would have to adjust the threshold at which one predicts true/**- false.** iii) Applying gradient clipping. C) If we are using the KNN-method with L1 distances….- **TRUE** || D) You train a model and you observe that the validation set…-**TRUE. different kind of architecture of lang model. 1**:1 - Image classification, sentiment analysis 1:M - Text generation M:M - Part-of-Speech Tagging(for same length), Machine Translation (Different length). **Face recognition using Saimes network: triplet loss function, including anchor positive and negative images. you are given 10k images each belonging to unique phase (?). What is minimum number of additional images you need two develop saimes network?** !0k extra total 20k images. **Lang model. What we want do? We want to find the max prob sentence. How we find?** Maximum Likelihood Estimation (MLE).

Advantages of Bidirectional Models: Capturing Context from Both Directions, Improved Performance in Sequence Tasks, Enhanced Feature Representation

In bi-directional what kind of information is used: Forward Information:The forward direction processes the input sequence from the beginning to the end.It captures the context and dependencies of each element based on the elements that precede it in the sequence.nformation from the past is utilized to understand the current state.**Backward Information**:The backward direction processes the input sequence from the end to the beginning.It captures the context and dependencies of each element based on the elements that succeed it in the sequence.Information from the future is utilized to understand the current state. **Padding** is used for maintaining fixed input size, batch processing, handling variable length sequences, avoid information loss, masking and attention mechanism,

The output shape of the **Embedding layer** is determined by the configuration:For an input sequence of length max_sequence_length, the **output shape** is (max_sequence_length , embedding_dim) The number of parameters in the Embedding layer is given by the formula:**Total Parameters** =vocabulary_size x embedding_di.

model.add(Embedding(input_dim=vocabulary_size, output_dim=embedding_dim, input_length=max_sequence_length))