CS-618
Deep Learning
LateSpring 2023

Lecture 6

Major Architectures of Deep Network (1of 2)

1. Convolutional Neural Networks (CNNs): CNNs are primarily used for image and video recognition tasks. They consist of multiple layers of convolutional and pooling operations, followed by fully connected layers. CNNs have achieved remarkable success in image classification, object detection, and image segmentation.
2. Recurrent Neural Networks (RNNs): RNNs are designed to process sequential data by introducing connections with feedback loops. This architecture enables RNNs to maintain internal memory, making them well-suited for tasks such as language modeling, machine translation, and speech recognition. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are popular variants of RNNs that address the vanishing gradient problem.
3. Generative Adversarial Networks (GANs): GANs are composed of two neural networks, a generator and a discriminator, which are trained in an adversarial setting. The generator generates synthetic data samples, while the discriminator tries to distinguish between real and synthetic samples. GANs have been successfully used for tasks such as image generation, style transfer, and data augmentation.

Major Architectures of Deep Network (1of 2)

1. Transformer Networks: Transformers have gained significant attention, especially in natural language processing tasks. They rely on self-attention mechanisms to capture relationships between different elements in a sequence. Transformers have achieved state-of-the-art results in machine translation, text summarization, and language understanding. The popular model "BERT" (Bidirectional Encoder Representations from Transformers) belongs to this architecture.
2. Autoencoders: Autoencoders are unsupervised learning models that aim to learn efficient representations of input data. They consist of an encoder network that maps the input to a lower-dimensional representation (latent space) and a decoder network that reconstructs the original input from the latent space. Autoencoders have been used for tasks like data compression, denoising, and anomaly detection.
3. Deep Reinforcement Learning (DRL): DRL combines deep learning with reinforcement learning techniques. Deep neural networks are used to approximate the action-value or policy functions in reinforcement learning algorithms. DRL has achieved remarkable success in game playing, robotics control, and autonomous driving.

Unsupervised Pretrained Networks

- An unsupervised pretrained network, also known as a pretraining or unsupervised pretraining method, is a technique used in deep learning to initialize the weights of a neural network using unsupervised learning. It involves training a model on a large amount of unlabeled data to learn meaningful representations or features of the input data. These learned representations can then be used as a starting point for training a supervised model on a smaller labeled dataset.
- In this group, we cover three specific architectures:
- Autoencoders
- Deep Belief Networks (DBNs)
- Generative Adversarial Networks (GANs)

The motivation behind unsupervised pretraining

- is to leverage the vast amounts of unlabeled data available, which is often easier to obtain than labeled data.
- By pretraining a network on this unlabeled data, the model can learn to extract high-level features that are relevant to the task at hand.
- This initialization can help the model converge faster and achieve better performance when fine-tuned on a smaller labeled dataset.

An autoencoder

- is a neural network architecture that is trained to reconstruct its input data from a compressed latent representation.
- By training the autoencoder on unlabeled data, the encoder network learns to extract meaningful features that capture the underlying structure of the data.
- These learned features can then be used for initializing the weights of a supervised model.

GAN

- Another popular method for unsupervised pretraining is generative modeling, such as training a generative adversarial network (GAN) or a variational autoencoder (VAE).
- These models learn to generate new samples that resemble the training data distribution.
- The pretrained generator network can then be used as an initialization for a supervised model.

Autoencoders

- We use autoencoders to learn compressed representations of datasets.
- we use them to reduce a dataset’s dimensionality.
- The output of the autoencoder network is a reconstruction of the input data in the most efficient form.

Defining features of autoencoders

- Autoencoders differ from multilayer perceptrons in a couple of ways:
  - They use unlabeled data in unsupervised learning.
  - They build a compressed representation of the input data.

- *Unsupervised learning of unlabeled data*
  - The autoencoder learns directly from unlabeled data. This is connected to the second major difference between multilayer perceptrons and autoencoders.
- *Learning to reproduce the input data*
  - The goal of a multilayer perceptron network is to generate predictions over a class (e.g., fraud versus not fraud). An autoencoder is trained to reproduce its own input data.

Training autoencoders

- Autoencoders rely on backpropagation to update their weights.
- The main difference between RBMs and the more general class of autoencoders is in how they calculate the gradients.

- The gradient is the generalization of the derivative to multivariate functions. It captures the local slope of the function, allowing us to predict the effect of taking a small step from a point in any direction

Common variants of autoencoders

- Two important variants of autoencoders to note are *compression autoencoders* and *denoising autoencoders*:
- *Compression autoencoders*
  - The network input must pass through a bottleneck region of the network before being expanded back into the output representation.
- *Denoising autoencoders*
  - The denoising autoencoder is the scenario in which the autoencoder is given a corrupted version (e.g., some features are removed randomly) of the input and the network is forced to learn the uncorrupted output.

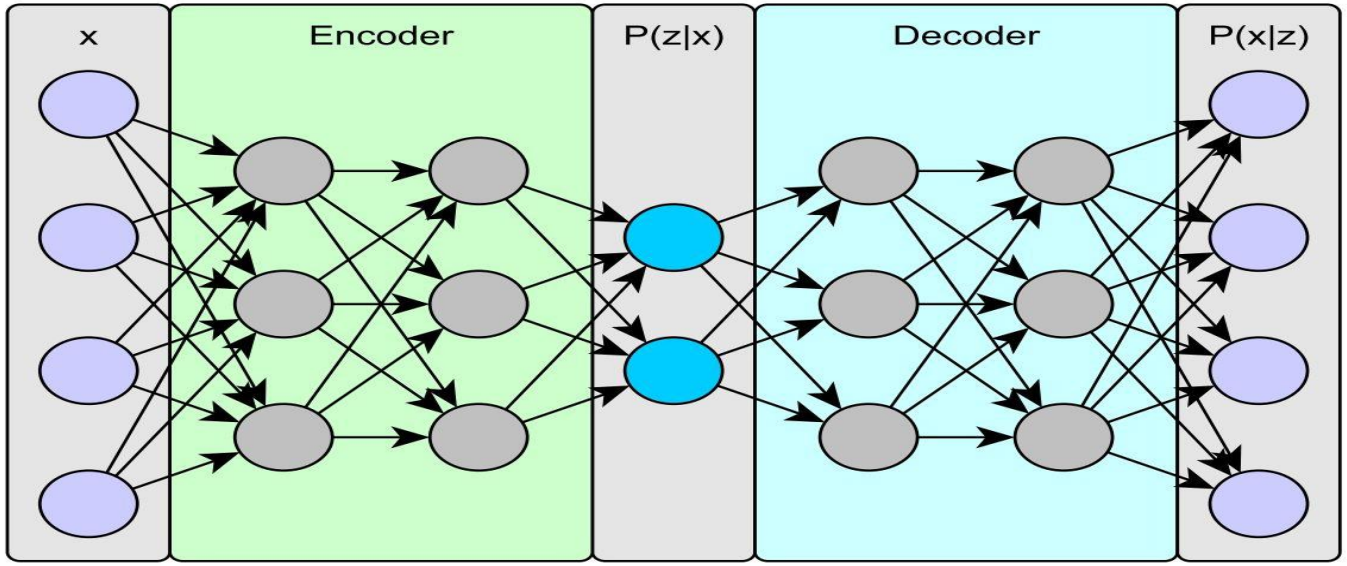
Applications of autoencoders

- AUTOENCODERS AS ANOMALY DETECTORS
- Autoencoders are commonly used in systems in which we know what the normal data will look like, yet it’s difficult to describe what is anomalous. Autoencoders are good at powering anomaly detection systems.

- Anomaly detection is the identification of rare events, items, or observations which are suspicious because they differ significantly from standard behaviors or patterns. Anomalies in data are also called standard deviations, outliers, noise, novelties, and exceptions.

Variational Autoencoders

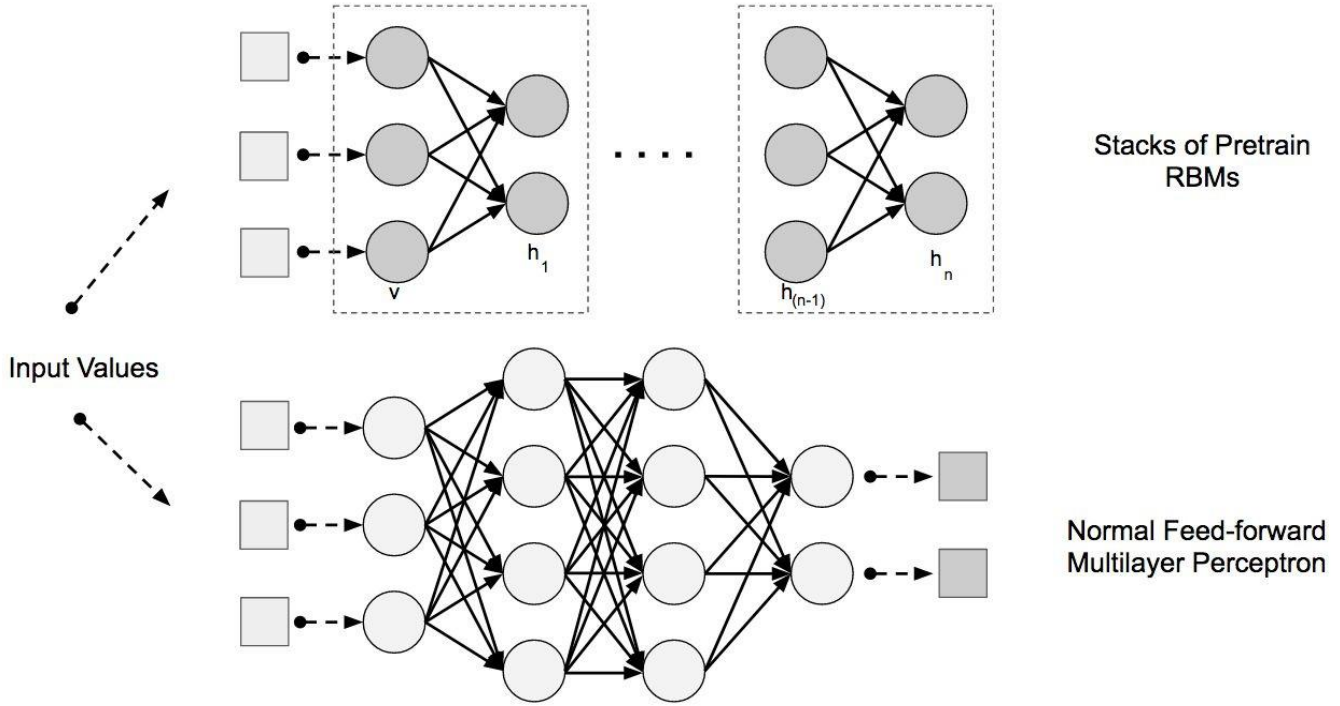
- A more recent type of autoencoder model is the variational autoencoder (VAE) introduced by Kingma and Welling



- The VAE is similar to compression and denoising autoencoders as they are all trained in an unsupervised manner to reconstruct inputs.
- However, the mechanisms that the VAEs use to perform training are quite different.
- In a compression/denoising autoencoder, activations are mapped to activations throughout the layers, as in a standard neural network; comparatively, a VAE uses a probabilistic approach for the forward pass.

Deep Belief Networks

- DBNs are composed of layers of Restricted Boltzmann Machines (RBMs) for the pretrain phase and then a feed-forward network for the fine-tune phase.



Deep Belief Networks (DBNs)

- are a type of generative deep learning model that combines the power of deep architectures with the probabilistic approach of belief networks.
- DBNs are composed of multiple layers of latent variables (hidden units) that form a hierarchical structure.

Generative Adversarial Networks

- GANs are an example of a network that uses unsupervised learning to train two models in parallel.
- A key aspect of GANs (and generative models in general) is how they use a parameter count that is significantly smaller than normal with respect to the amount of data on which we're training the network.
- The network is forced to efficiently represent the training data, making it more effective at generating data similar to the training data.

A generative adversarial network (GAN)

- is a class of deep learning models that consists of two neural networks: a generator and a discriminator.
- GANs are used for generative modeling tasks, such as generating new samples that resemble a given training dataset.
- The generator network in a GAN takes random noise or a latent vector as input and attempts to generate realistic samples that resemble the training data.
- The generator is trained to produce samples that can fool the discriminator into classifying them as real.
- The discriminator network, on the other hand, is trained to distinguish between real samples from the training dataset and fake samples generated by the generator.
- It tries to correctly classify whether a given sample is real or fake.

Generative Adversarial Networks (GANs) have found numerous applications(1of3)

1. Image Generation: GANs have been widely used for generating realistic images. They can learn the underlying distribution of a training dataset and generate new samples that resemble the training data. GANs have been used to generate high-resolution images, create artistic images, and even generate images based on textual descriptions.
2. Image-to-Image Translation: GANs have been employed for image-to-image translation tasks, where the goal is to transform an input image from one domain to another while preserving the important characteristics. For example, GANs can convert images from day to night, transform sketches into photorealistic images, or change the style of an image.
3. Text-to-Image Synthesis: GANs have been used to generate images from textual descriptions. By conditioning the generator network on text input, GANs can generate images that correspond to the provided descriptions. This application has implications for generating images based on textual prompts, improving accessibility, and aiding in virtual reality environments.

Generative Adversarial Networks (GANs) have found numerous applications(2of 3)

1. Video Generation: GANs have been extended to generate videos by capturing temporal dependencies. They can generate new video frames based on preceding frames, enabling applications such as video synthesis, video prediction, and video editing.
2. Style Transfer: GANs have been employed for transferring the style of one image to another while preserving the content. This enables artistic transformations, where the style of a famous painting, for example, can be applied to a real photograph.
3. Data Augmentation: GANs can be used to augment training data by generating additional synthetic samples. This can help increase the size of training datasets and improve the generalization and performance of machine learning models.

Generative Adversarial Networks (GANs) have found numerous applications(3 of 3)

1. Anomaly Detection: GANs can be utilized for anomaly detection by training on normal samples and then identifying deviations from the learned data distribution. GANs can learn to generate typical samples and flag unusual or anomalous samples as outliers.
2. Domain Adaptation: GANs have been applied to adapt models trained on one domain to perform well on a different domain. By learning the underlying mapping between the source and target domains, GANs can help transfer knowledge and reduce the need for labeled data in the target domain.

Training generative models, unsupervised learning, and GANs

- If we had a large corpus of training images
- we could build a generative neural network that outputs images (as opposed to classifications).
- We'd consider these generated output images to be samples from the model.
- The generative model in GANs generates such images while a secondary "discriminator" network tries to classify these generated images.
- This secondary discriminator network attempts to classify the output images as real or synthetic.
- When training GANs, we want to update the parameters such that the network will generate more believable output images based on the training data.
- The goal here is to make images realistic enough that the discriminator network is fooled to the point that it cannot distinguish the difference between the real and the synthetic input data.
- An example of efficient model representation in GANs is how they typically have around 100 million parameters when modeling a dataset such as ImageNet.

The discriminator network

- When modeling images, the discriminator network is typically a standard CNN.
- Using a secondary neural network as the discriminator network allows the GAN to train both networks in parallel in an unsupervised fashion.
- These discriminator networks take images as input, and then output a classification.
- The gradient of the output of the discriminator network with respect to the synthetic input data indicates how to make small changes to the synthetic data to make it more realistic.

The generative network

- The generative network in GANs generates data (or images) with a special kind of layer called a *deconvolutional layer*
- During training, we use backpropagation on both networks to update the generating network's parameters to generate more realistic output images.
- The goal here is to update the generating network's parameters to the point at which the discriminating network is sufficiently "fooled" by the generating network because the output is so realistic as compared to the training data's real images.

LSTM Networks

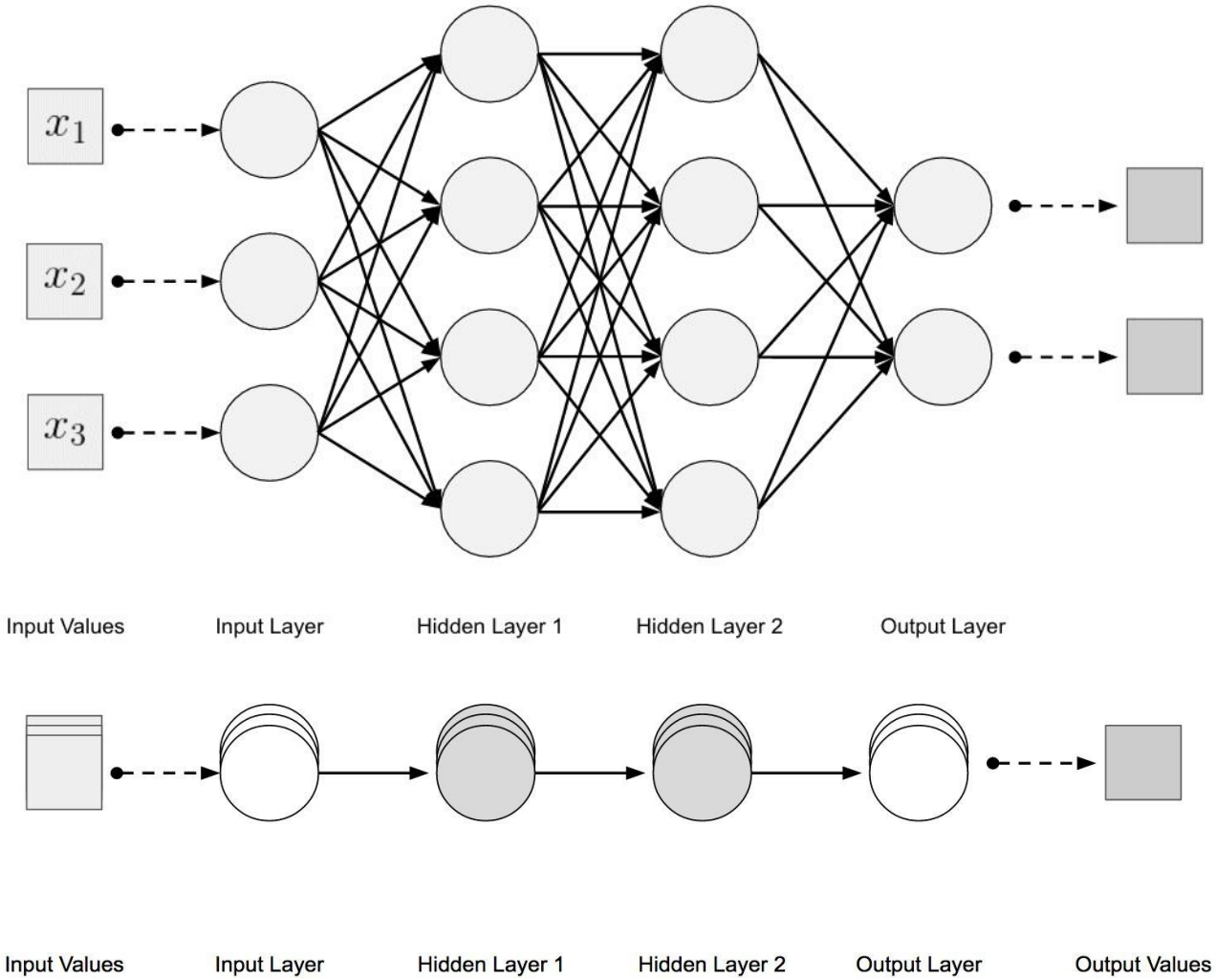
- LSTM networks are the most commonly used variation of Recurrent Neural Networks.
- LSTM networks were introduced in 1997 by Hochreiter and Schmidhuber.
- The critical component of the LSTM is the memory cell and the gates (including the forget gate, but also the input gate).
- The contents of the memory cell are modulated by the input gates and forget gates.
- Assuming that both of these gates are closed, the contents of the memory cell will remain unmodified between one time-step and the next.
- The gating structure allows information to be retained across many time-steps, and consequently also allows gradients to flow across many time-steps.
- This allows the LSTM model to overcome the vanishing gradient problem that occurs with most Recurrent Neural Network models.

Properties of LSTM networks

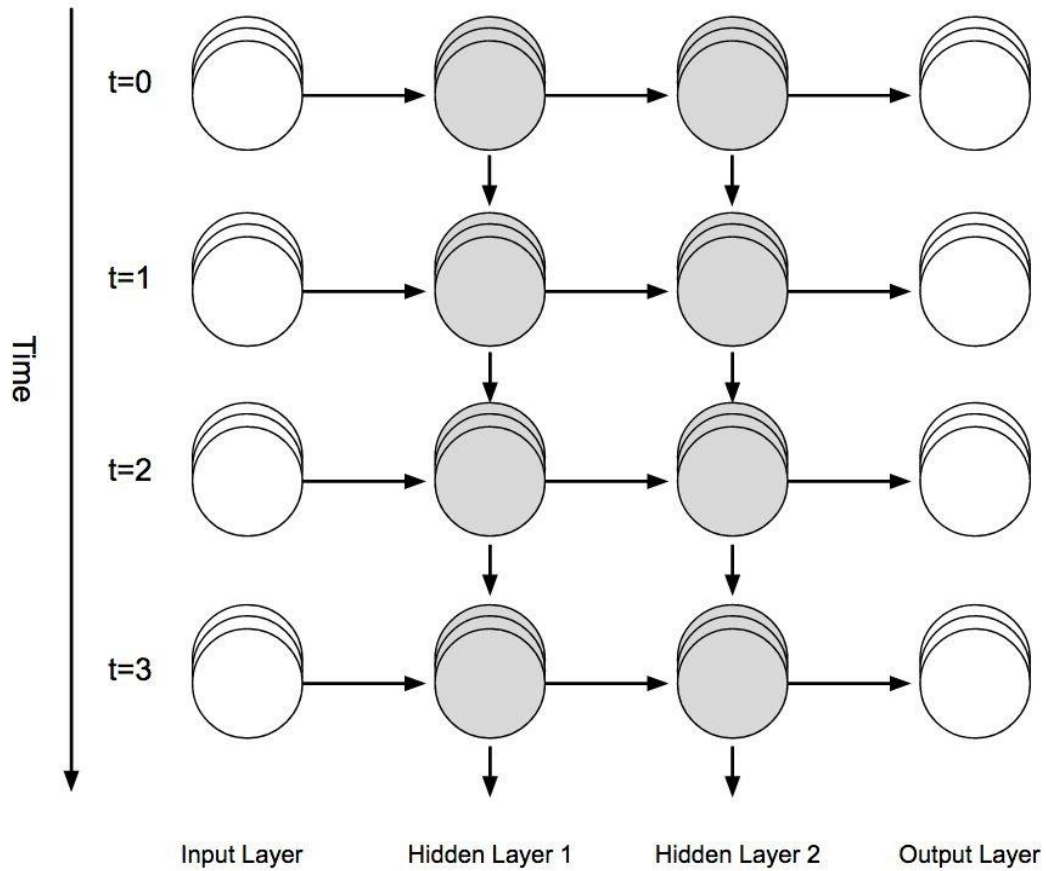
- LSTMs are known for the following:
  - Better update equations
  - Better backpropagation
- Here are some example use cases of LSTMs:
  - Generating sentences (e.g., character-level language models)
  - Classifying time-series
  - Speech recognition
  - Handwriting recognition
  - Polyphonic music modeling

- LSTM and Bidirectional Recurrent Neural Networks (BRNN) architectures have shown industry-leading benchmarks in recent years on tasks such as the Following:
  - Image captioning
  - Language translation
  - Handwriting recognition
- LSTM networks consist of many connected LSTM cells and perform well in how efficient they are during learning.

LSTM network architecture

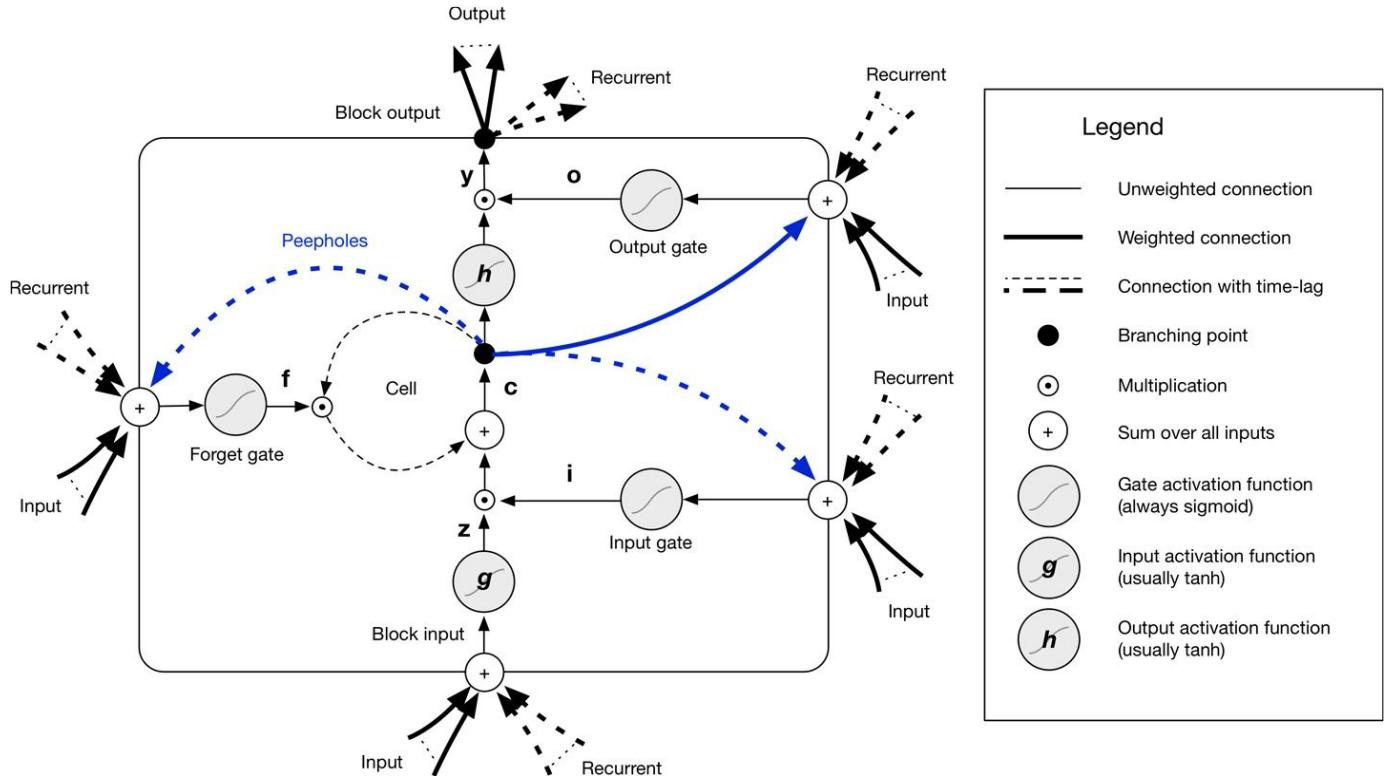


Recurrent Neural Network unrolled along the time axis



LSTM units

- The units in the layers of Recurrent Neural Networks are a variation on the classic artificial neuron.
- Each LSTM unit has two types of connections:
  - Connections from the previous time-step (outputs of those units)
  - Connections from the previous layer
- the components in an LSTM unit:
- Three gates
  - input gate (input modulation gate)
  - forget gate
  - output gate
- Block input
- Memory cell (the constant error carousel)
- Output activation function
- Peephole connections



The components in an LSTM unit

- Three gates
  - input gate (input modulation gate)
  - forget gate
  - output gate
- Block input
- Memory cell (the constant error carousel)
- Output activation function
- Peephole connections

- There are three gate units, which learn to protect the linear unit from misleading signals:
  - The input gate protects the unit from irrelevant input events.
  - The forget gate helps the unit forget previous memory contents.
  - The output gate exposes the contents of the memory cell (or not) at the output of the LSTM unit.

Recursive Neural Networks

- Recursive Neural Networks, like Recurrent Neural Networks, can deal with variable length input.
- The primary difference is that Recurrent Neural Networks have the ability to model the hierarchial structures in the training dataset.
- Images commonly have a scene composed of many objects.
- Deconstructing scenes is often a problem domain of interest yet is nontrivial.
- The recursive nature of this deconstruction challenges us to not only identify the objects in the scene, but also how the objects relate to form the scene.

Network Architecture

- A Recursive Neural Network architecture is composed of a shared-weight matrix and a binary tree structure that allows the recursive network to learn varying sequences of words or parts of an image.
- It is useful as a sentence and scene parser. Recursive Neural Networks use a variation of backpropagation called *backpropagation through structure* (BPTS).
- The feed-forward pass happens bottom-up, and backpropagation is top-down.

Varieties of Recursive Neural Networks

- Recursive Neural Networks come in a few varieties.
- One is the recursive autoencoder. Just like its feed-forward cousin, recursive autoencoders learn how to reconstruct the input. In the case of NLP, it learns how to reconstruct contexts.
- A semisupervised recursive autoencoder learns the likelihood of certain labels in each context.
- Another variation of this is a supervised neural network, called a Recursive Neural Tensor Network, which computes a supervised objective at each node of the tree.
- The tensor part of this means that it calculates the gradient a little differently, factoring in more information at each node by taking advantage of another dimension of information using a tensor (a matrix of three or more dimensions).

Applications of Recursive Neural Networks

- Both Recursive and Recurrent Neural Networks share many of the same use cases.
- Recurrent Neural Networks are traditionally used in NLP because of their ties to binary trees, contexts, and natural-language-based parsers.
- For example, constituency parsers are able to break up a sentence into a binary tree, segmenting it by the linguistic properties of the sentence.
- In the case of Recursive Neural Networks, it is a constraint that we use a parser that builds the tree structure (typically constituency parsing).
- Recursive Neural Networks can recover both granular structure and higher-level hierarchical structure in datasets such as images or sentences.
- Applications for recursive neural networks include the following:
  - Image scene decomposition
  - NLP
  - Audio-to-text transcription

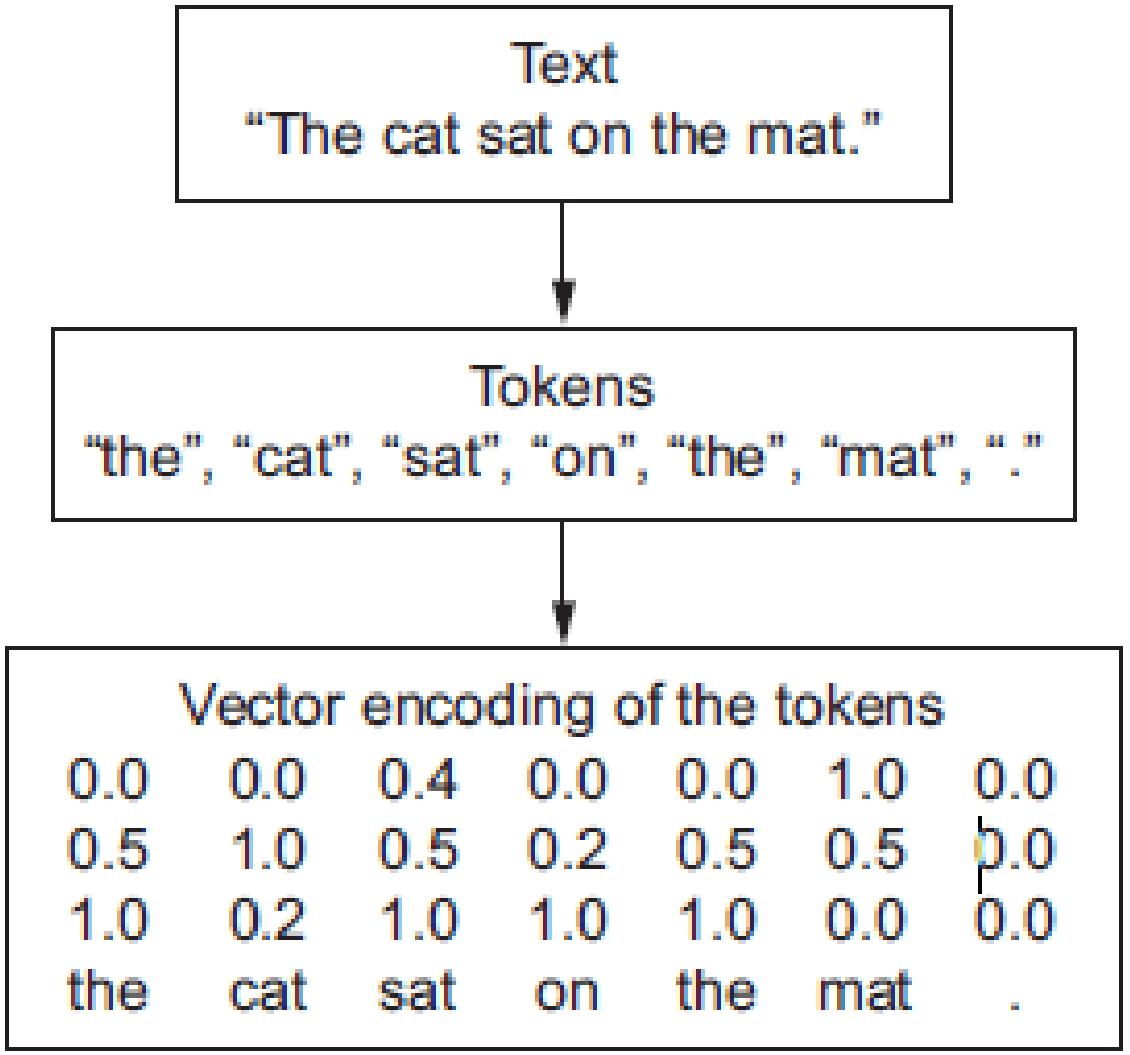
Deep learning for text and sequences

- Text is one of the most widespread forms of sequence data.
- It can be understood as either a sequence of characters or a sequence of words, but it's most common to work at the level of words.
- The deep-learning sequence-processing models introduced in the following sections can use text to produce a basic form of natural-language understanding, sufficient for applications including document classification, sentiment analysis, author identification, and even question-answering (QA) (in a constrained context).
- Deep learning models truly understand text in a human sense; rather, these models can map the statistical structure of written language, which is sufficient to solve many simple textual tasks.
- Deep learning for natural-language processing is pattern recognition applied to words, sentences, and paragraphs, in much the same way that computer vision is pattern recognition applied to pixels.
- Like all other neural networks, deep-learning models don't take as input raw text:
  - they only work with numeric tensors
  - *Vectorizing* text is the process of transforming text into numeric tensors.

- This can be done in multiple ways:
  - Segment text into words, transform each word into a vector.
  - Segment text into characters, transform each character into a vector.
  - Extract n-grams of words or characters, transform each n-gram into a vector.

Tokens

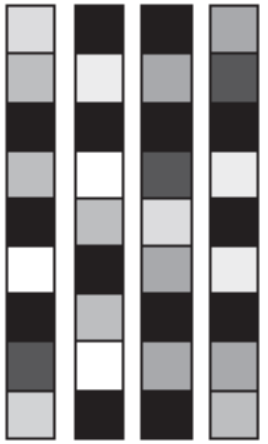
- the different units into which you can break down text (words, characters, or n-grams) are called *tokens*,
- breaking text into such tokens is called *tokenization*.
- All text-vectorization processes consist of applying some tokenization scheme and then associating numeric vectors with the generated tokens.
- These vectors, packed into sequence tensors, are fed into deep neural networks.
- There are multiple ways to associate a vector with a token.
- In this section, I'll present two major ones:
- *one-hot encoding* of tokens, and *token embedding* (typically used exclusively for words, and called *word embedding*).



word embeddings

- Another popular and powerful way to associate a vector with a word is the use of dense *word vectors*, also called *word embeddings*
- Whereas the vectors obtained through one-hot encoding are binary, sparse (mostly made of zeros), and very high-dimensional (same dimensionality as the number of words in the vocabulary), word embeddings are lowdimensional floating-point vectors (that is, dense vectors, as opposed to sparse vectors)

One-hot word vectors:
- Sparse
- High-dimensional
- Hardcoded



Word embeddings:
- Dense
- Lower-dimensional
- Learned from data

There are two ways to obtain word embeddings:

- Learn word embeddings jointly with the main task you care about (such as document classification or sentiment prediction). In this setup, you start with random word vectors and then learn word vectors in the same way you learn the weights of a neural network.
- Load into your model word embeddings that were precomputed using a different machine-learning task than the one you're trying to solve. These are called *pretrained word embeddings*.

Deep Learning Techniques for Text Classification

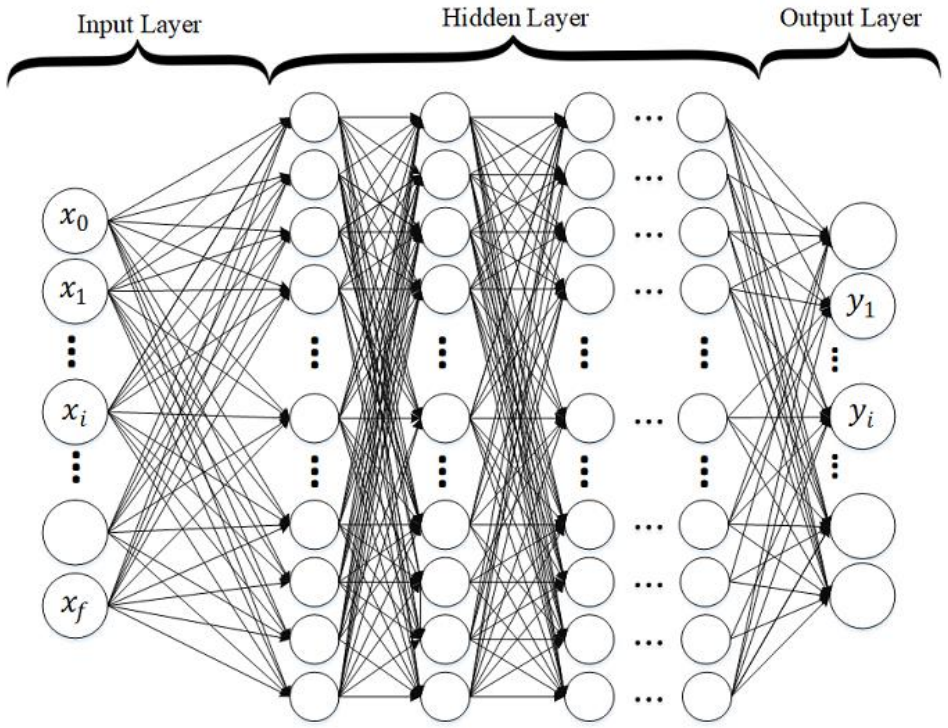
- The exponential growth in the number of complex datasets every year requires more enhancement in machine learning methods to provide robust and accurate data classification.
- Lately, deep learning approaches are achieving better results compared to previous machine learning algorithms on tasks like image classification, natural language processing, face recognition, etc.
- The success of these deep learning algorithms relies on their capacity to model complex and non-linear relationships within the data.

Text classification methods

- Deep Neural Networks
- Recurrent Neural Networks (RNN)
- Gated Recurrent Unit (GRU)
- Long Short-Term Memory (LSTM)
- Convolutional Neural Networks (CNN)
- Hierarchical Attention Networks
- Recurrent Convolutional Neural Networks (RCNN)
- Random Multimodel Deep Learning (RMDL)
- Hierarchical Deep Learning for Text (HDLTex)

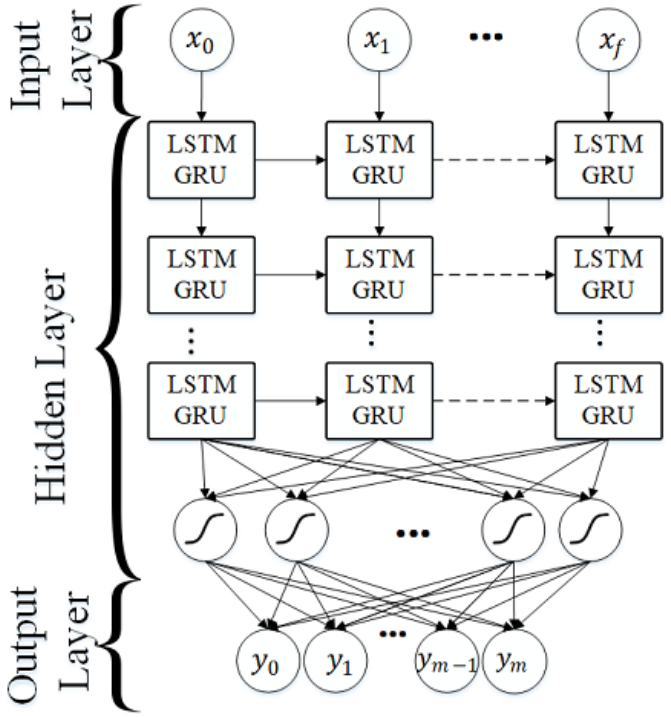
1- Deep Neural Networks

- The implementation of Deep Neural Network (DNN) is basically a discriminatively trained model that uses the standard back-propagation algorithm and sigmoid or ReLU as activation functions. The output layer for multi-class classification should use Softmax.



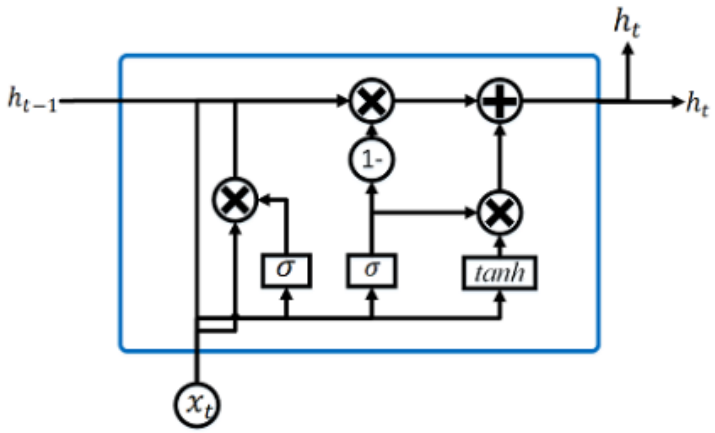
2- Recurrent Neural Networks (RNN)

- this technique is a powerful method for text, string, and sequential data classification. In RNN, the neural net considers the information of previous nodes in a very sophisticated method which allows for better semantic analysis of the structures in the dataset.



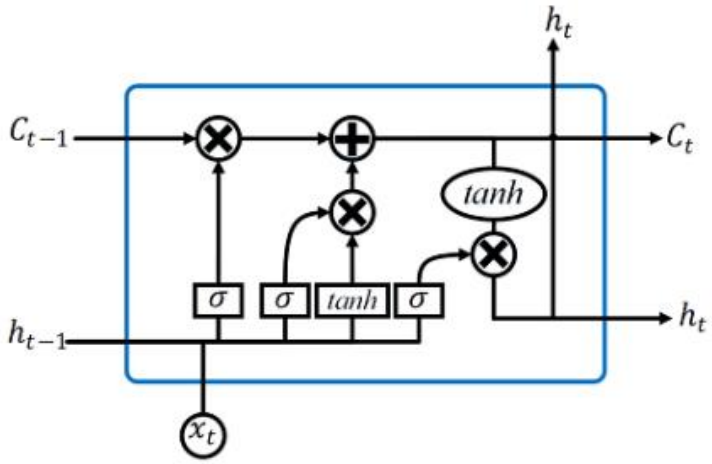
3- **Gated Recurrent Unit (GRU)**

Gated Recurrent Unit (GRU) is a gating mechanism for RNN GRU is a simplified variant of the LSTM architecture, but there are differences as follows: GRU contains two gates and does not possess any internal



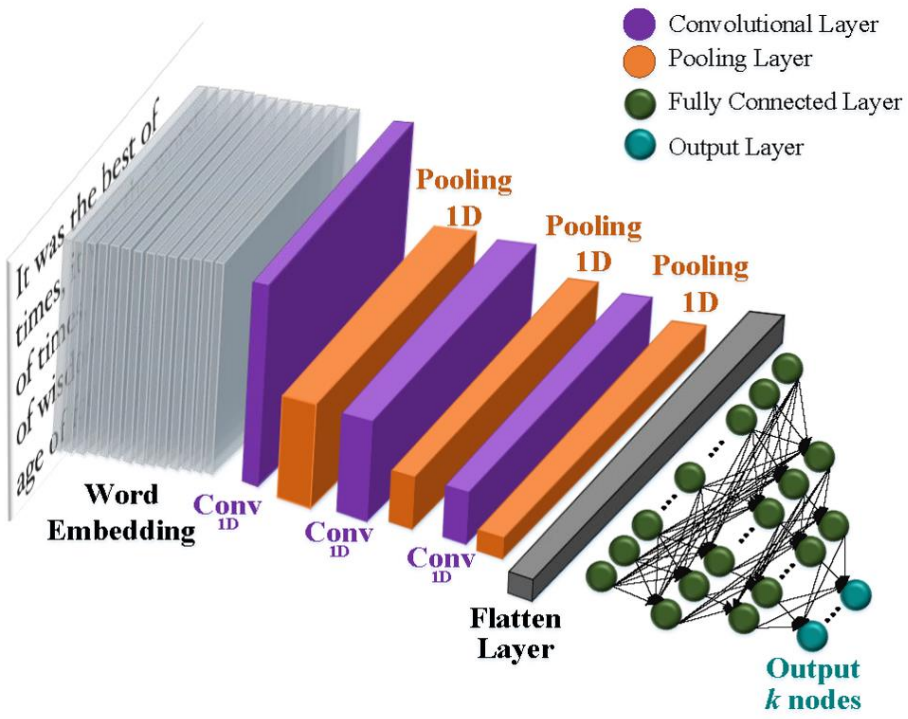
4- **Long Short-Term Memory (LSTM)**

- Long Short-Term Memory (LSTM) is a special type of RNN that preserves long term dependency in a more effective way compared to the basic RNNs. This is particularly useful to overcome vanishing gradient problem as LSTM uses multiple gates to carefully regulate the amount of information that will be allowed into each node state.



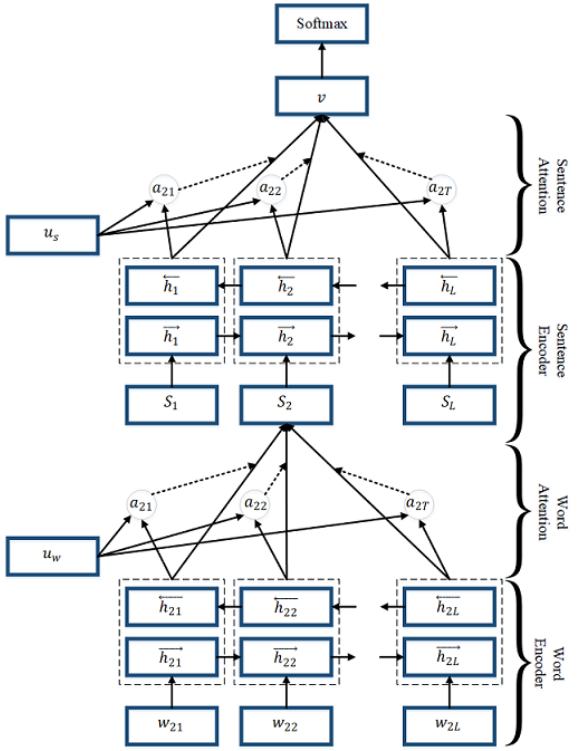
5- **Convolutional Neural Networks (CNN)**

Convolutional Neural Networks (CNN) is Another deep learning architecture that is employed for hierarchical document classification. Although originally built for image processing with architecture similar to the visual cortex, CNNs have also been effectively used for text classification. In a basic CNN for image processing, an image tensor is convolved with a set of kernels of size *d by d*. These convolution layers are called feature maps and can be stacked to provide multiple filters on the input. To reduce the computational complexity, CNNs use pooling which reduces the size of the output from one layer to the next in the network. Different pooling techniques are used to reduce outputs while preserving important features.



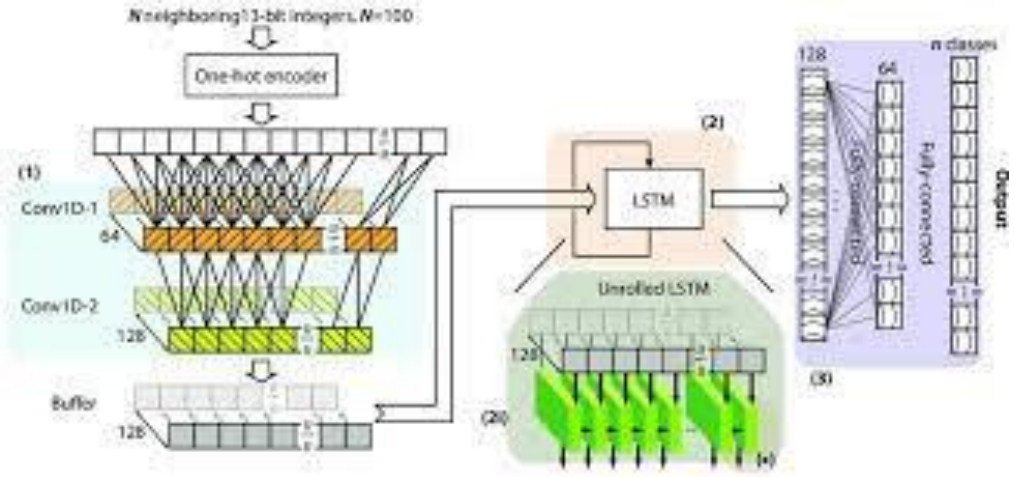
6- **Hierarchical Attention Networks**

Hierarchical Attention Network **uses stacked recurrent neural networks on word level, followed by an attention network**. The goal is to extract such words that are important to the meaning of the entire sentence and aggregate these instructional words to form a vector of the sentence



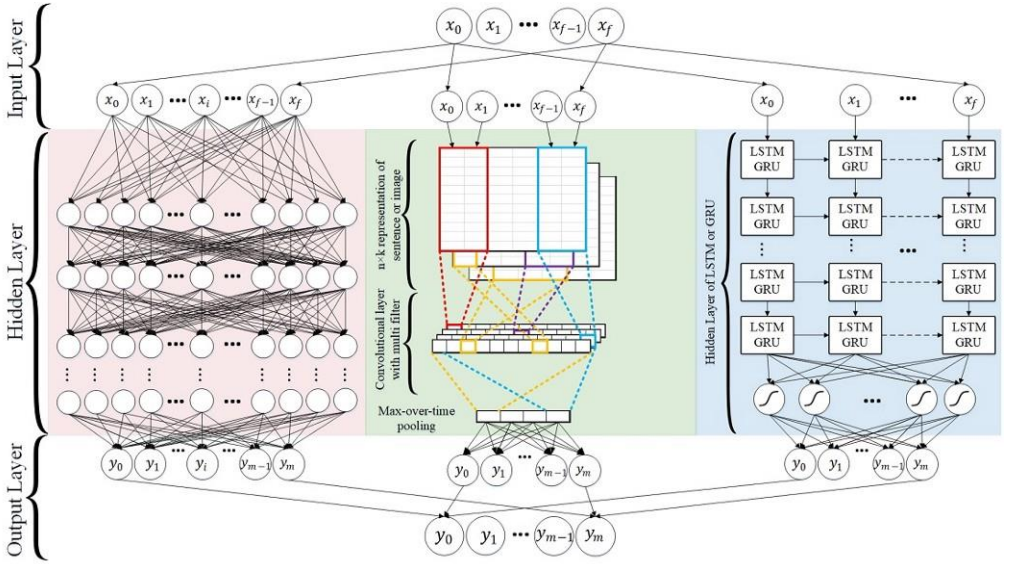
7- Recurrent Convolutional Neural Networks (RCNN)

- Recurrent Convolutional Neural Networks (RCNN) is also used for text classification. The main idea of this technique is capturing contextual information with the recurrent structure and constructing the representation of text using a convolutional neural network. This architecture is a combination of RNN and CNN to use the advantages of both technique in a model.



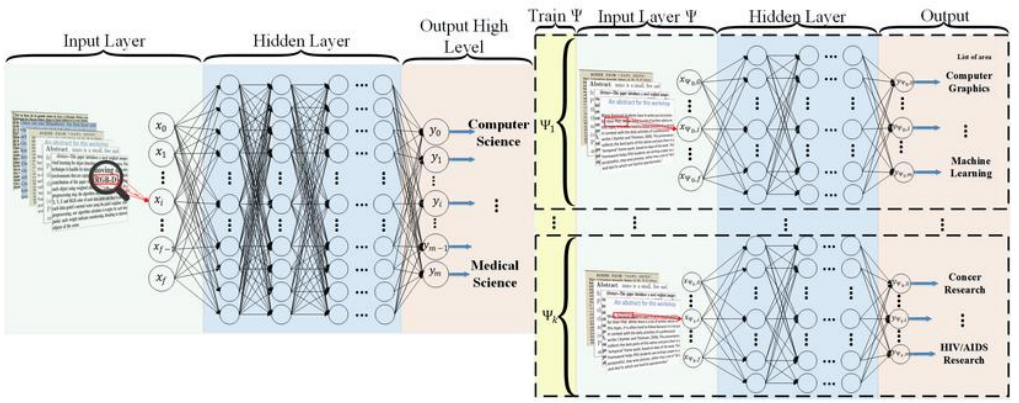
8- **Random Multimodel Deep Learning (RMDL)**

RMDL solves the problem of finding the best deep learning structure and architecture while simultaneously improving robustness and accuracy through ensembles of different deep learning architectures. RDMLs can accept a variety of data as input including text, video, images, and symbols.



9- **Hierarchical Deep Learning for Text (HDLTex)**

- perform hierarchical classification using an approach we call Hierarchical Deep Learning for Text classification (HDLTex). HDLTex employs stacks of deep learning architectures to provide a hierarchical understanding of the documents.



Thank You