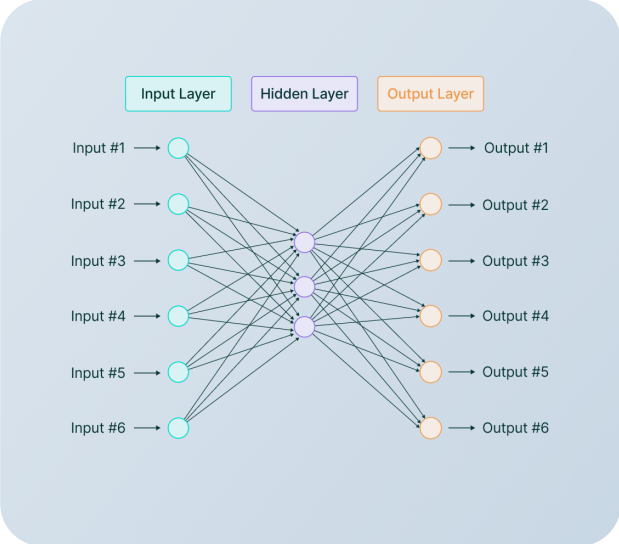
BLOG > DEEP LEARNING

The Essential Guide to Neural Network Architectures

What are Neural Networks and how do they work? Learn about different Artificial Neural Networks architectures, their characteristics, and limitations.

16 min read · July 8, 2021

Pragati Baheti
Microsoft



Here's the fact—

Deep learning, specifically Neural Networks, is a boiling hot area of research.

There are countless new Neural Network architectures proposed and updated every single day.

Earlier, the use of Neural Networks was restricted to simple classification problems, like spam messages, but they have since advanced to domains like visual search engines, recommendation engines, chatbots, and the medical field.

The evolution of small Artificial Neural Networks that could handle fewer data samples has evolved into architectures consisting of millions of parameters trained on tons of data.

Here's what we'll cover:

1. What are Neural Networks?
2. Standard Neural Networks
3. Recurrent Neural Networks (RNNs)
4. Convolutional Neural Networks (CNNs)
5. Generative Adversarial Network (GAN)

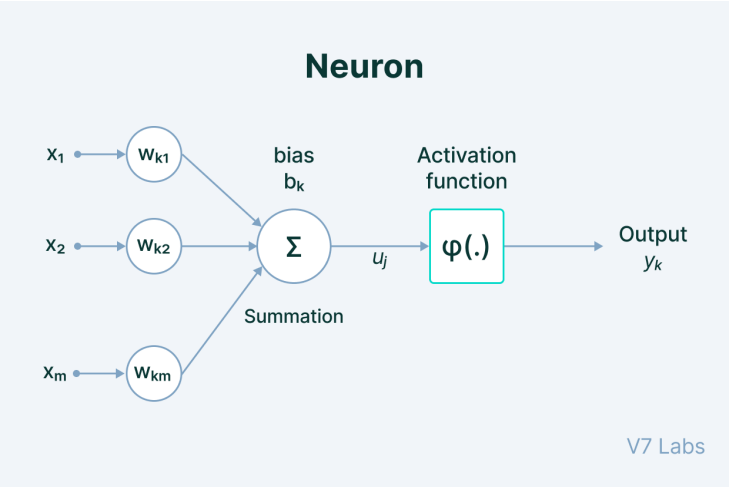
What are Neural Networks?

Neural Networks are the functional unit of Deep Learning and are known to mimic the behavior of the human brain to solve complex data-driven problems.

The input data is processed through different layers of artificial neurons stacked together to produce the desired output.

From speech recognition and person recognition to healthcare and marketing, Neural Networks have been used in a varied set of domains.

Key Components of the Neural



Neuron in Artificial Neural Network

Input - It is the set of features that are fed into the model for the learning process. For example, the input in object detection can be an array of pixel values pertaining to an image.

Weight - Its main function is to give importance to those features that contribute more towards the learning. It does so by introducing scalar multiplication between the input value and the weight matrix. For example, a negative word would impact the decision of the sentiment analysis model more than a pair of neutral words.

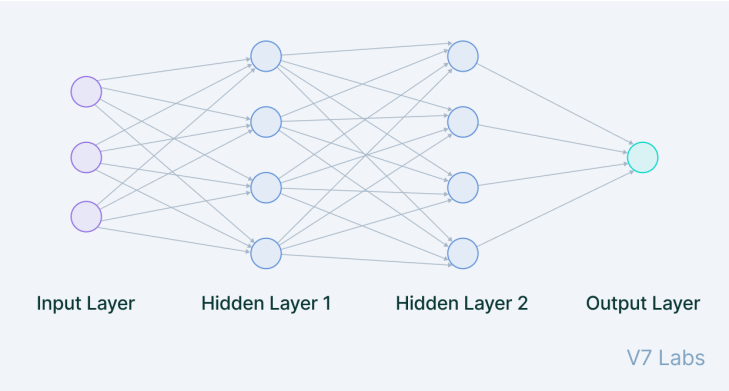
Transfer function - The job of the transfer function is to combine multiple inputs into one output value so that the activation function can be applied. It is done by a simple summation of all the inputs to the transfer function.

Activation Function—It introduces non-linearity in the working of perceptrons to consider varying linearity with the inputs. Without this, the output would just be a linear combination of input values and would not be able to introduce non-linearity in the network.

> 💡 **Pro tip:** Looking for a perfect source for a recap of activation functions? Check out Types of Neural Networks Activation Functions.

Bias - The role of bias is to shift the value produced by the activation function. Its role is similar to the role of a constant in a linear function.

When multiple neurons are stacked together in a row, they constitute a layer, and multiple layers piled next to each other are called a multi-layer neural network.



Multi-layer neural network

Input Layer

The data that we feed to the model is loaded into the input layer from external sources like a CSV file or a web service. It is the only visible layer in the complete Neural Network architecture that passes the complete information from the outside world without any computation.

Hidden Layers

The hidden layers are what makes deep learning what it is today. They are intermediate layers that do all the computations and extract the features from the data.

There can be multiple interconnected hidden layers that account for searching different hidden features in the data. For example, in image processing, the first hidden layers are responsible for higher-level features like edges, shapes, or boundaries. On the other hand, the later hidden layers perform more complicated tasks like identifying complete objects (a car, a building, a person).

Output Layer

The output layer takes input from preceding hidden layers and comes to a final prediction based on the model's learnings. It is the most important layer where we get the final result.

In the case of classification/regression models, the output layer generally has a single node. However, it is completely problem-specific and dependent on the way the model was built.

Standard Neural Networks

The Perceptron

Perceptron is the simplest Neural Network architecture.

It is a type of Neural Network that takes a number of inputs, applies certain mathematical operations on these inputs, and produces an output. It takes a vector of real values inputs, performs a linear combination of each attribute with the

Feed-Forward Networks

Perceptron represents how a single neuron works.

But—

What about a series of perceptrons stacked in a row and piled in different layers? How does the model learn then?

It is a multi-layer Neural Network, and, as the name suggests, the information is passed in the forward direction—from left to right.

In the forward pass, the information comes inside the model through the input layer, passes through the series of hidden layers, and finally goes to the output layer. This Neural Networks architecture is forward in nature—the information does not loop with two hidden layers.

The later layers give no feedback to the previous layers. The basic learning process of Feed-Forward Networks remain the same as the perceptron.

Residual Networks (ResNet)

Now that you know more about the Feed-Forward Networks, one question might have popped up in your head—how to decide on the number of layers in our neural network architecture?

A naive answer would be: *The greater the number of hidden layers, the better is the learning process.*

More layers enrich the levels of features.But—

Is that so?

Very deep Neural Networks are extremely difficult to train due to vanishing and exploding gradient problems.

ResNets provide an alternate pathway for data to flow to make the training process much faster and easier.

This is different from the feed-forward approach of earlier Neural Networks architectures.

The core idea behind ResNet is that a deeper network can be made from a shallow network by copying weight from the shallow counterparts using identity mapping.

The data from previous layers is fast-forwarded and copied much forward in the Neural Networks. This is what we call *skip connections* first introduced in Residual Networks to resolve vanishing gradients.

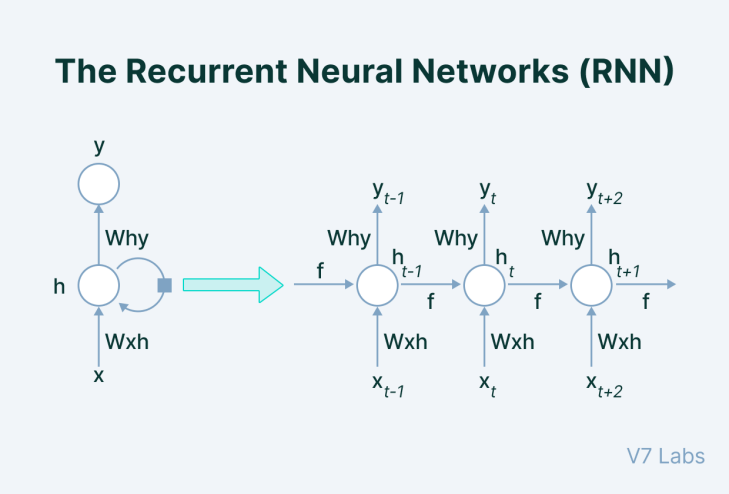
Recurrent Neural Networks (RNNs)

The basic deep learning architecture has a fixed input size, and this

Recurrent Neural Networks work very well with sequences of data as input. Its functionality can be seen in solving NLP problems like sentiment analysis, spam filters, time series problems like sales forecasting, stock market prediction, etc.

The Recurrent Neural Networks (RNN)

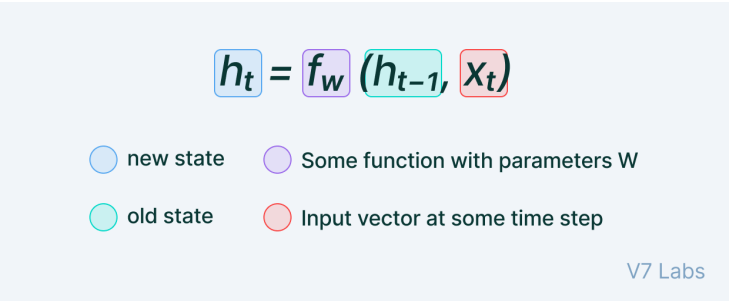
Recurrent Neural Networks have the power to remember what it has learned in the past and apply it in future predictions.



The input is in the form of sequential data that is fed into the RNN, which has a hidden internal state that gets updated every time it reads the following sequence of data in the input.

The internal hidden state will be fed back to the model. The RNN produces some output at every timestamp.

The mathematical representation is given below:

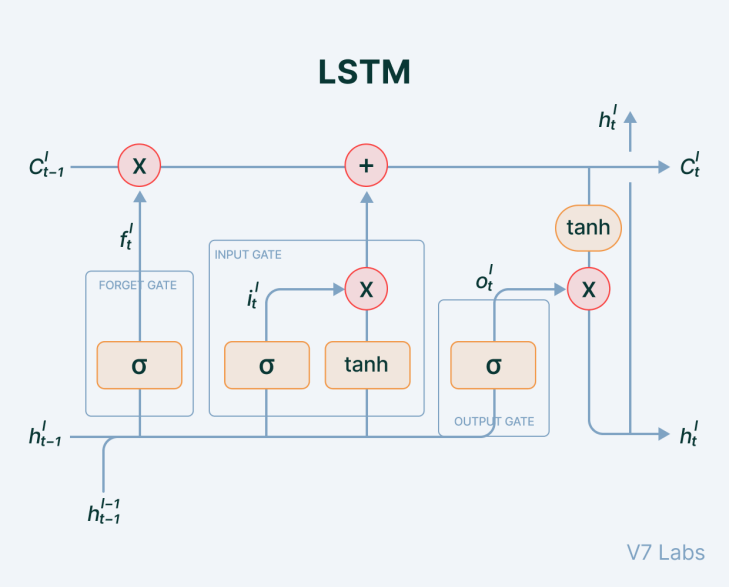


💡 **Note:** We use the same function and parameters at every timestamp.

The Long Short Term Memory Network (LSTM)

To rectify this, we can take our Recurrent Neural Networks structure and expand it by adding some more pieces to it.

The critical part that we add to this Recurrent Neural Networks is **memory**. We want it to be able to remember what happened many timestamps ago. To achieve this, we need to add extra structures called *gates* to the artificial neural network structure.



- **Cell state (c_t):** It corresponds to the long-term memory content of the network.
- **Forget Gate:** Some information in the cell state is no longer needed and is erased. The gate receives two inputs, x_t (current timestamp input) and h_t-1 (previous cell state), multiplied with the relevant weight matrices before bias is added. The result is sent into an activation function, which outputs a binary value that decides whether the information is retained or forgotten.
- **Input gate:** It decides what piece of new information is to be added to the cell state. It is similar to the forget gate using the current timestamp input and previous cell state with the only difference of multiplying with a different set of weights.
- **Output gate:** The output gate's job is to extract meaningful information from the current cell state and provide it as an output.

Echo State Networks (ESN)

Echo state Networks is a RNN with sparsely connected hidden layers with typically 1% connectivity.

The connectivity and weight of hidden neurons are fixed and randomly assigned. The only weight that needs to be learned is that of the output layer. It can be seen as a linear model of the weighted input passed through all the hidden layers and the

The training becomes uncomplicated, assuming linear output units. The only thing to keep in mind is to set the random connections very carefully.

Convolutional Neural Networks (CNNs)

Convolutional Neural Networks is a type of Feed-Forward Neural Networks used in tasks like image analysis, natural language processing, and other complex image classification problems.

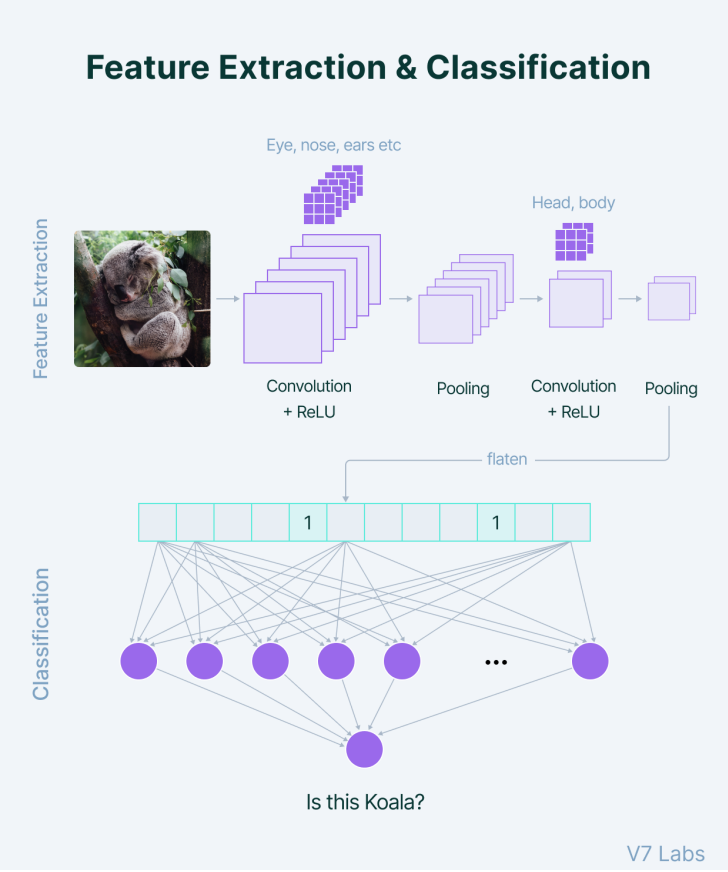
A CNN has hidden layers of convolutional layers that form the base of ConvNets.

Features refer to minute details in the image data like edges, borders, shapes, textures, objects, circles, etc.

At a higher level, convolutional layers detect these patterns in the image data with the help of filters. The higher-level details are taken care of by the first few convolutional layers.

The deeper the network goes, the more sophisticated the pattern searching becomes.

For example, in later layers rather than edges and simple shapes, filters may detect specific objects like eyes or ears, and eventually a cat, a dog, and what not.



A *filter* can be thought of as a relatively small matrix for which we decide the number of rows and columns this matrix has.

The value of this feature matrix is initialized with random numbers. When this convolutional layer receives pixel values of input data, the filter will convolve over each patch of the input matrix.

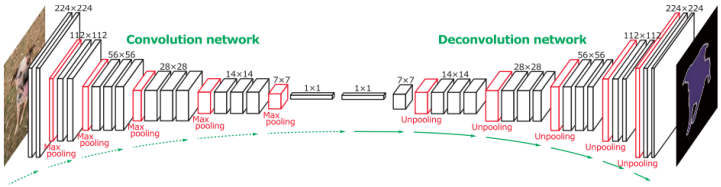
The output of the convolutional layer is usually passed through the ReLU activation function to bring non-linearity to the model. It takes the feature map and replaces all the negative values with zero.

Pooling is a very important step in the ConvNets as it reduces the computation and makes the model tolerant towards distortions and variations. A Fully Connected Dense Neural Networks would use a flattened feature matrix and predict according to the use case.

The Deconvolutional Neural Networks (DNN)

Deconvolutional Neural Networks are CNNs that work in a reverse manner.

When we use convolutional layers and max-pooling, the size of the image is reduced. To go to the original size, we use upsampling and transpose convolutional layers. Upsampling does not have trainable parameters—it just repeats the rows and columns of the image data by its corresponding sizes.



Transpose Convolutional layer means applying convolutional operation and upsampling at the same time. It is represented as Conv2DTranspose (number of filters, filter size, stride). If we set stride=1, we do not have any upsampling and receive an output of the same input size.

AlexNet

AlexNet was trained on the Imagenet dataset with 15 million high-resolution images with 256*256*3. It has multiple convolutional layers and is deeper than the LeNet artificial neural network.

Here are the characteristics of AlexNet:

- GPU learning was carried out for the first time
- Overlapping pooling was done in order to prevent information loss.

It had five convolutional-pooling layer blocks followed by three fully connected dense layers for classification.

Overfeat

This Neural Networks architecture explores three well-known vision tasks of classification, localization, and detection using a single framework.

It trains the models on all three tasks simultaneously to boost up the accuracy.

It is a modification of AlexNet. It predicts bounding boxes at each spatial location and scale. For localization, the classification head is replaced by a regression network.

VGG

VGG stands for Visual Geometry Group.

The thought behind VGG was that if AlexNet performed better than LeNet by being bigger and deeper, why not keep pushing further?

One of the paths that we could take was to add more dense layers. This would bring with it more computations.

The next possible approach was to have more convolutional layers. But this didn't work out as it was very tiring to define each convolutional layer separately.

So—

The best of all the solutions was to group convolutional layers into blocks.

The question was: *Is it better to use fewer wider convolutional blocks or more narrow ones?*

Eventually, the researchers concluded that more layers of narrow convolutions were more powerful than smaller numbers of wider convolutions.

A VGG-block had a bunch of 3x3 convolutions padded by 1 to keep the output size the same as that of input, followed by max-pooling to half the resolution. The architecture had *n* number of VGG blocks followed by three fully connected dense layers.

Network-in-network

Convolutional layers need fewer parameters. It is the last few layers of fully connected neurons that bring a huge spike in the

Convolutions and pooling reduce the resolutions, but at some point, we still need to map it to corresponding classes. Therefore, the idea was to reduce the resolution as we go deeper and increase the number of channels by using 1*1 convolutions. This gives us high-quality information per channel.

In network-in-network architecture, the last fully connected layer is replaced by a global max-pooling layer making the model light.

GoogLeNet and Inception

Inception Neural Networks architecture has three convolutional layers with different size filters and max-pooling. Every layer has different size filters for parallel learning.

There are different size filters to take care of huge variations in the location of information, which makes it very difficult to choose the right size filter.

The small filter size convolutional layer takes care of a small information area.

A bigger filter size captures a bigger unit of information.

GoogleNet architecture consists of inception blocks that have 1x1, 3x3, 5x5 convolutional layers followed by 3x3 max pooling with padding (to make the output of the same shape as the input) on the previous layer, followed by concatenations of their output.

SqueezeNet

It aims for smaller CNNs so that there is less communication across servers during distributed training.

The changes it performs on the AlexNet architecture are as follows:

- Replace 3*3 filters with 1*1 filters to reduce the number of parameters.
- Downsample later in the architecture so that the convolutional layers have large activation maps
- They squeeze the features with squeeze layers consisting of 1*1 convolutional layers and then they expand it with a combination of 1*1 and 3*3 convolutional layers. Each squeeze-expand block is placed together and is known as a *fire module*.

Xception

The convolutional layer that is the basic building block of all CNN's involves a convolution operation. Each convolution operation involves the sliding of a filter through all the patches of the input pixel array.

separable convolutions that are proposed in Xception architecture break down this operation into two parts:

- Depthwise convolution
- Pointwise convolution

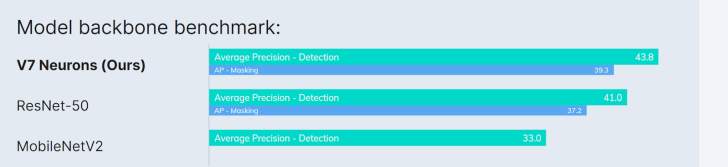
MobileNets

MobileNets use depth-wise separable convolutions to build lightweight deep Neural Networks. They develop very small, low latency models that are used for applications like robots, self-driving cars, etc. These are considered best for mobile devices, and hence their name—MobileNets.

In a simple CNN structure, a filter is a block that is superimposed on the input image block, and the dot product is calculated between the two overlapping components. The details inside one channel are calculated along with the relationship between different channels.

Instead of having one large filter, MobileNets have two filters:

- One goes through one channel at a time to detect how all the pixels in a channel are related
- The other goes through all the channels at the same time to see how one pixel is related to every other pixel that comes behind it.



> 💡 **Pro tip:** Check out V7 pre-trained model (VoVNet) for object detection that outperforms many state-of-art architectures.

Capsule Networks

There were some problems with Convolutional Neural Networks—

They were trained to learn on images: in the lower layers and learn about edges and curvatures, and as we go up the hierarchy, it learns more complex features.

Sub-sampling or pooling loses spatial relationships.

To help you understand it better—

It's not enough for the model to learn that the image contains a

You see, Convolutional Neural Networks perform poorly in detecting an image in a different position, for example, rotated. It has to be in a position similar to the images they were trained on.

And this is a problem.

Instead of invariance, the network should strive for equivariance. It means that no matter what position or rotation a subsampled image is in, the neural network responds in the same way. It should also change accordingly to adapt to such sub-images.

In simple words: We need a network that is easier to generalize.

Here's the main idea—

Artificial Neural Networks must achieve translation rotation and invariance in a much more efficient way. These networks should have local capsules that perform complex internal computations on their inputs and then encapsulate the results into a small vector of highly informative outputs.

Now, try to keep this in mind and start thinking about using a capsule instead of a neuron. Sounds interesting, right?

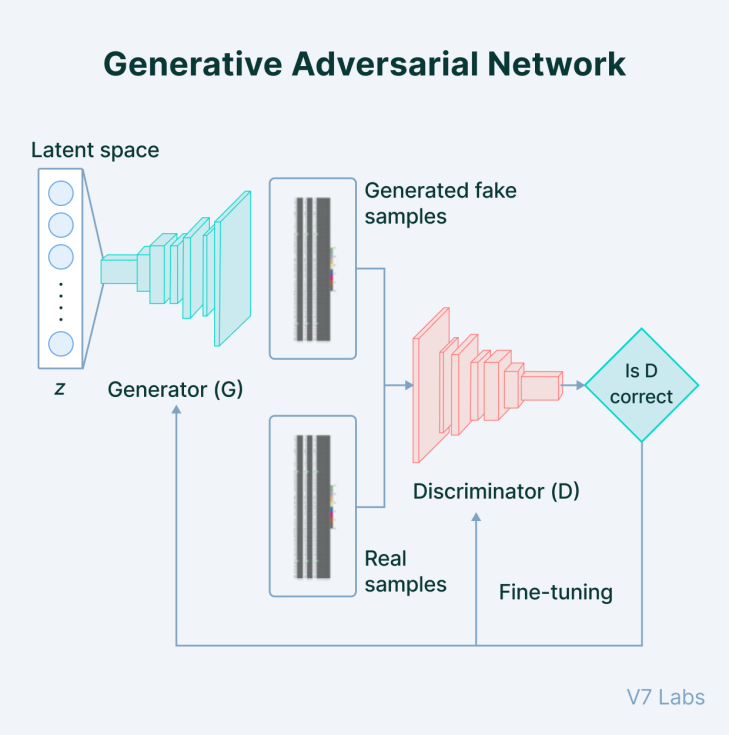
Neural Networks where instead of adding a layer, it nests a new layer inside a layer. This nested layer is called a *capsule* which is a group of neurons. Instead of making the structure deeper in terms of layers, a Capsule Network nests another layer within the same layer.

This makes the model more robust.

Generative Adversarial Network (GAN)

Generative modeling comes under the umbrella of unsupervised learning, where new/synthetic data is generated based on the patterns discovered from the input set of data.

GAN is a generative model and is used to generate entirely new synthetic data by learning the pattern and hence is an active area of AI research.



They have two components—a generator and a discriminator that work in a competitive fashion.

The generator's job is to create synthetic data based on the model's features during its learning phase. It takes in random data as input and returns a generated image after performing certain transformations.

The discriminator acts as a critic and has an overall idea of the problem domain with a clear understanding of generated images.

These generated images are classified into fake/genuine images by the discriminator.

The discriminator returns a probabilistic prediction for the images to be noisy/free-of-noise by a value in the range of 0 to 1, where 1 is an authentic image and 0 a fake image.

The generator network produces samples based on its learning.

Its adversary, the discriminator, strives to distinguish between samples from the training data and samples produced from the generator. There is feedback from the discriminator fed to the generator to improve the performance.

When the discriminator successfully distinguishes between real and fake examples, the component is working well and no changes need to be applied to its parameters.

The generator is given a penalty when it fails to generate an image as real such that it could fool the discriminator. However, if it succeeds in making the discriminator categorize the generated

It is used in scenarios like predicting the next frame in a video, text to image generation, image to image translation like style transfer, denoising of the image, etc.

Transformer Neural Networks

The truth is—

RNNs are slow and take too much time in training.

They are not very good with large sequenced data and lead to vanishing gradients. LSTMs that were introduced to bring memory in the RNN became even slower to train.

For both RNN and LSTM, we need to feed the data sequentially or serially. This does not make use of GPUs.

How to parallelize the training on sequential data?

The answer is Transformers.

These networks employ an encoder-decoder structure with a difference that the input data can be passed parallelly.

In RNN structure, one word at a time was passed through the input layer. But in Transformers, there is no concept of timestamps for passing the input. We feed the complete sentence together and get the embeddings for all the words together.

How do these Transformer Neural Networks do this?

Consider an example of English-French translation.

Positional Encoder: Vector gives context based on the position of the word in the sentence.

So, Input Embeddings + Positional Encoder = Input Embeddings with context information

We pass this into an encoder block where it goes to a multi-head attention layer and a Feed-Forward layer.

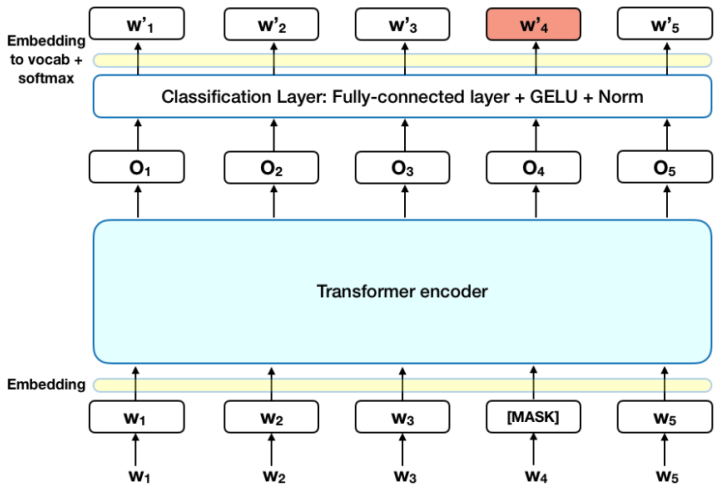
The attention layer determines what part of the input sentence the model should focus on. During the training, the corresponding French sentence embeddings are fed to the decoder that has three main components.

Self-attention blocks generate attention vectors for every word in the sentence to represent how much each word is related to every word in the same sentence. These attention vectors and encoder's vectors are passed into another attention block called—"encoder-decoder attention block." This attention block determines how related each word vector is with respect to each other, and this is where English to French mapping occurs.

A big change in the architecture was proposed -

RNNs had a drawback of not using parallel computing and loss of critical information through the sequenced time stamped data. In contrast, Transformers are based on Attention that require a single step to feed all the sequential data and have a self-attention mechanism working in the core architecture to preserve important information.

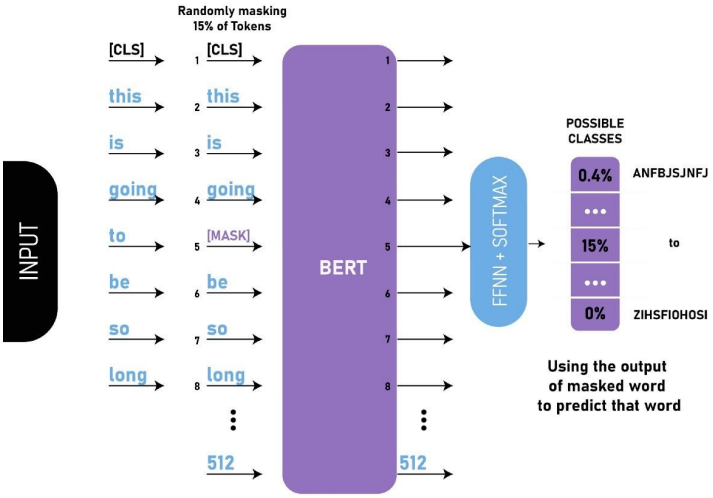
BERT



BERT (*Bidirectional Encoder Representations from Transformer*s) outperform LSTM.

These models are faster as the words can be processed simultaneously. The context of words is better learned as they can

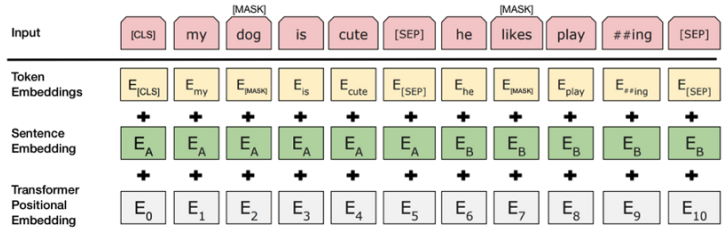
Masked Language Modeling: BERT takes input sentences and replaces some random words with [MASK] tokens. The goal of the model is to predict the original word of the masked words based on the context provided by the other, non-masked, words in the sequence.



The model calculates the probability of each word in the vocabulary with the Softmax squashing function. It helps BERT understand the bidirectional context within a sentence.

Next Sentence Prediction: In this case, BERT take the input of two sentences, and it determines if the second sentence follows the first sentence.

This helps BERT understand context across different sentences.



To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

- A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
- A sentence embedding indicating Sentence A or Sentence B is added to each token. It is created to understand the correlation between the sentences.
- A positional embedding is added to each token to indicate its position in the sequence. This helps the model to understand

function of the two strategies and get a good understanding of the language.

GPT; GPT2; GPT3

GPT (Generative PreTraining) is a language model used to predict the probability of the sequence of words.

Language models involving generative training do not require human-labeled data.

GPT-1 has two steps of training—unsupervised pre-training using unlabeled data with language model objective function followed by supervised fine-tuning of the model without a task-specific model. GPT uses transformer decoder architecture.

With GPT2, the purpose of the model shifted more to the text generation side. It is an autoregressive language model. It is trained on an input sequence and its target is predicting the next token at each point of the sequence.

It consists of a single stack of transformer blocks with an attention mechanism. It has slightly lower dimensionality than BERT, with more transformer blocks(48 blocks) and a larger sequence length.

The basic structure of GPT3 is similar to that of GPT2, with the only difference of more transformer blocks(96 blocks) and is trained on more data. The sequence size of input sentences also doubled as compared to GPT2. It is by far the largest neural network architecture containing the most number of parameters.

Momentum Contrast (MoCo)

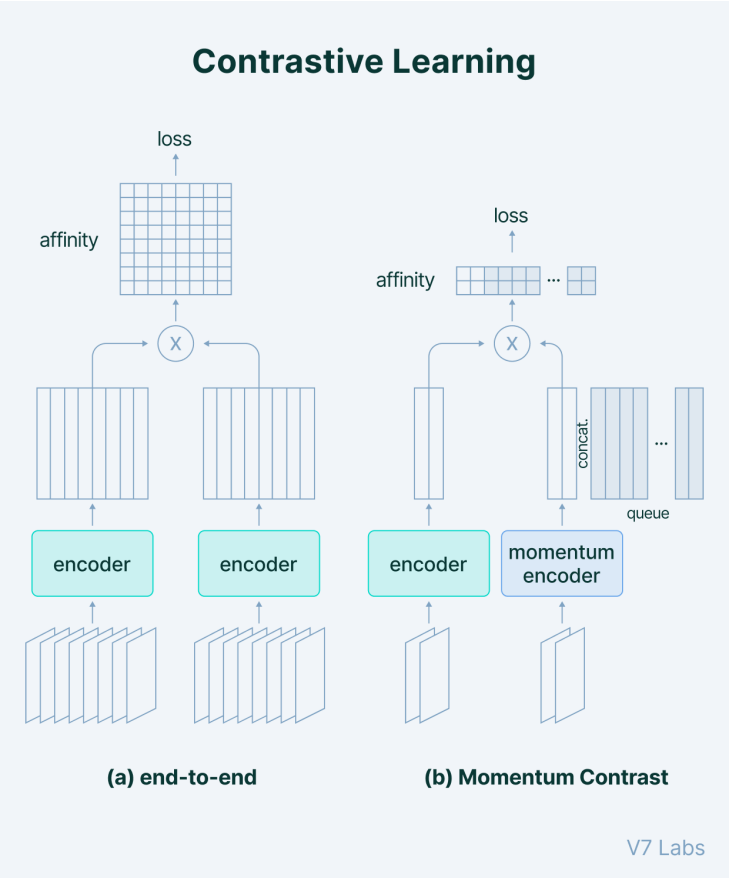
The idea behind this model was that unsupervised pre-training could surpass the supervised counterpart in computer vision tasks like detection or segmentation.

In the past, we have already seen that models like BERT, GPT that are based on unsupervised learning have been a huge success in the NLP domain.

In natural language processing related tasks, a model is given an input sentence and the model is required to predict one or multiple following words. Lets say we have a dictionary of all proposed words. Using such a dictionary allows us to define loss as a simple dictionary look-up problem.

Let's say an image is passed through an encoder; the encoded feature of the image can be called a *query*.

The dictionary in this case is a set of features of a large set of images. Such a dictionary is very hard to create as images and corresponding features are not readily available. A dynamic dictionary is prepared by applying the encoder model to a set of images.



The image above represents a batching perspective of two optimization mechanisms for contrastive learning. Images are encoded into a representation space, in which pairwise affinities are computed.

MoCo addresses two challenges in Contrastive Learning:

- How to make the dynamic dictionary large enough?
- How to make the dynamic dictionary consistent when the encoder is being updated?

To make a large dictionary in the Contrastive Learning framework, we maintain the features of the previous batch of images as a queue. The dictionary consists of current and prior batches and is not limited by the batch size.

The features in this dictionary are produced by an encoder that is being constantly updated, hence reducing the overall consistency of the dictionary. To solve this consistency problem, a momentum encoder is suggested that gets slowly updated,

SimCLR

Contrastive Learning is combined with data augmentation, larger batch size, more training epochs, and wider networks.

SimCLR strongly augmented the unlabeled training data and feed them to series of standard ResNet architecture and a small neural

cross-entropy is used that says that similarity between two embeddings should be close if they form image and augmented image pair.

In other words, the embeddings should attract. On the other hand, the similarity between images that do not belong to the same class should repel.

Summary

Each Neural Networks architecture has its own set of pros and cons.

Standard Neural Networks like Feed-Forward Neural Networks are most commonly used in solving classification and regression problems related to simple structured data.

Recurrent Neural Networks are much more powerful in terms of remembering information for a long time and are used in sequential data like text, audio, video etc.

Recent research shows that Transformers based on Attention Mechanism outperform RNNs and have almost replaced RNNs in every field.

For complex data like images we can use ConvNets in classification tasks and for generation of images or style transfer related tasks Generative Adversarial Networks performs the best.

💡 **Read more:**

The Beginner's Guide to Self-Supervised Learning

Overfitting vs. Underfitting: What's the Difference?

The Beginner's Guide to Deep Reinforcement Learning

The Complete Guide to Ensemble Learning

A Newbie-Friendly Guide to Transfer Learning

The Essential Guide to Zero-Shot Learning

Supervised vs. Unsupervised Learning: What's the Difference?

The Complete Guide to CVAT—Pros & Cons

5 Alternatives to Scale AI

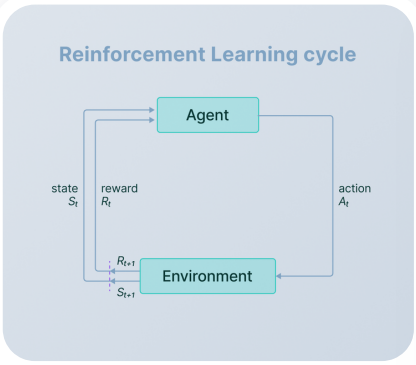
YOLO: Real-Time Object Detection Explained

Knowledge Distillation: Principles & Algorithms [+Applications]

Pragati Baheti
Microsoft

Pragati is a software developer at Microsoft, and a deep learning enthusiast. She writes about the fundamental mathematics behind deep neural networks.

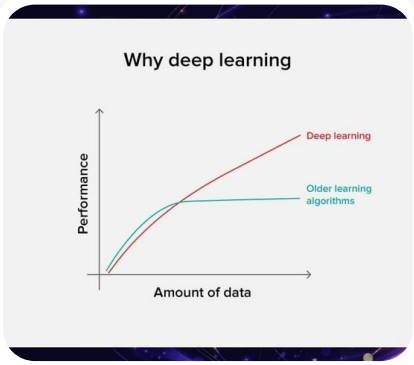
Related articles



DEEP LEARNING

The Beginner's Guide to Deep Reinforcement Learning [2023]

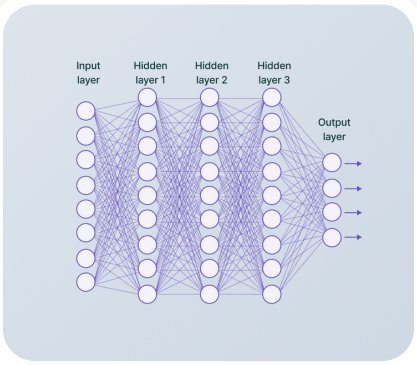
Pragati Baheti — 10 min read



DEEP LEARNING

A Gentle Introduction to Deep Learning—the ELI5 Way

Nilesh Barla — 13 min read



DEEP LEARNING

A Comprehensive Guide to Convolutional Neural Networks

Pragati Baheti — 9 min read