**Recurrent neural network**

## What is RNN?

A recurrent neural network (RNN) is a deep learning model that is trained to process and convert a sequential data input into a specific sequential data output. Sequential data is data—such as words, sentences, or time-series data—where sequential components interrelate based on complex semantics and syntax rules. An RNN is a software system that consists of many interconnected components mimicking how humans perform sequential data conversions, such as translating text from one language to another. RNNs are largely being replaced by transformer-based artificial intelligence (AI) and large language

## How does a recurrent neural network work?

**The following image shows a diagram of an RNN.**

RNNs are made of neurons: data-processing nodes that work together to perform complex tasks. The neurons are organized as input, output, and hidden layers. The input layer receives the information to process, and the output layer provides the result. Data processing, analysis, and prediction take place in the hidden layer.

### **Hidden layer**

RNNs work by passing the sequential data that they receive to the hidden layers one step at a time. However, they also have a self-looping or recurrent workflow: the hidden layer can remember and use previous inputs for future predictions in a short-term memory component. It uses the current input and the stored memory to predict the next sequence.

For example, consider the sequence: Apple is red. You want the RNN to predict red when it receives the input sequence Apple is. When the hidden layer processes the word Apple, it stores a copy in its memory. Next, when it sees the word is, it recalls Apple from its memory and understands the full sequence: Apple is for context. It can then predict red for improved accuracy. This makes RNNs useful in speech recognition, machine translation, and other language modeling tasks.

### **Training**

Machine learning (ML) engineers train deep neural networks like RNNs by feeding the model with training data and refining its performance. In ML, the neuron's weights are signals to determine how influential the information learned during training is when predicting the output. Each layer in an RNN shares the same weight.

ML engineers adjust weights to improve prediction accuracy. They use a technique called backpropagation through time (BPTT) to calculate model error and adjust its weight accordingly. BPTT rolls back the output to the previous time step and recalculates the error rate. This way, it can identify which hidden state in the sequence is causing a significant error and readjust the weight to reduce the error margin.

## What are the types of recurrent neural networks?

RNNs are often characterized by one-to-one architecture: one input sequence is associated with one output. However, you can flexibly adjust them into various configurations for specific purposes. The following are several common RNN types.

### **One-to-many**

This RNN type channels one input to several outputs. It enables linguistic applications like image captioning by generating a sentence from a single keyword.

### **Many-to-many**

The model uses multiple inputs to predict multiple outputs. For example, you can create a language translator with an RNN, which analyzes a sentence and correctly structures the words in a different language.

### **Many-to-one**

Several inputs are mapped to an output. This is helpful in applications like sentiment analysis, where the model predicts customers’ sentiments like positive, negative, and neutral from input testimonials.

## How do recurrent neural networks compare to other deep learning networks?

RNNs are one of several different neural network architectures.

### **Recurrent neural network vs. feed-forward neural network**

Like RNNs, feed-forward neural networks are artificial neural networks that pass information from one end to the other end of the architecture. A feed-forward neural network can perform simple classification, regression, or recognition tasks, but it can’t remember the previous input that it has processed. For example, it forgets Apple by the time its neuron processes the word is. The RNN overcomes this memory limitation by including a hidden memory state in the neuron.

### **Recurrent neural network vs. convolutional neural networks**

Convolutional neural networks are artificial neural networks that are designed to process spatial data. You can use convolutional neural networks to extract spatial information from videos and images by passing them through a series of convolutional and pooling layers in the neural network. RNNs are designed to capture long-term dependencies in sequential data

## What are some variants of recurrent neural network architecture?

The RNN architecture laid the foundation for ML models to have language processing capabilities. Several variants have emerged that share its memory retention principle and improve on its original functionality. The following are some examples.

### **Bidirectional recurrent neural networks**

A bidirectional recurrent neural network (BRNN) processes data sequences with forward and backward layers of hidden nodes. The forward layer works similarly to the RNN, which stores the previous input in the hidden state and uses it to predict the subsequent output. Meanwhile, the backward layer works in the opposite direction by taking both the current input and the future hidden state to update the present hidden state. Combining both layers enables the BRNN to improve prediction accuracy by considering past and future contexts. For example, you can use the BRNN to predict the word trees in the sentence Apple trees are tall.

## What are the limitations of recurrent neural networks?

Since the RNN’s introduction, ML engineers have made significant progress in natural language processing (NLP) applications with RNNs and their variants. However, the RNN model family has several limitations.

### **Exploding gradient**

An RNN can wrongly predict the output in the initial training. You need several iterations to adjust the model’s parameters to reduce the error rate. You can describe the sensitivity of the error rate corresponding to the model’s parameter as a gradient. You can imagine a gradient as a slope that you take to descend from a hill. A steeper gradient enables the model to learn faster, and a shallow gradient decreases the learning rate.

Exploding gradient happens when the gradient increases exponentially until the RNN becomes unstable. When gradients become infinitely large, the RNN behaves erratically, resulting in performance issues such as overfitting. Overfitting is a phenomenon where the model can predict accurately with training data but can’t do the same with real-world data.

### **Vanishing gradient**

The vanishing gradient problem is a condition where the model’s gradient approaches zero in training. When the gradient vanishes, the RNN fails to learn effectively from the training data, resulting in underfitting. An underfit model can’t perform well in real-life applications because its weights weren’t adjusted appropriately. RNNs are at risk of vanishing and exploding gradient issues when they process long data sequences.

### **Slow training time**

An RNN processes data sequentially, which limits its ability to process a large number of texts efficiently. For example, an RNN model can analyze a buyer’s sentiment from a couple of sentences. However, it requires massive computing power, memory space, and time to summarize a page of an essay.

## How do transformers overcome the limitations of recurrent neural networks?

Transformers are deep learning models that use self-attention mechanisms in an encoder-decoder feed-forward neural network. They can process sequential data the same way that RNNs do.

### **Self-attention**

Transformers don’t use hidden states to capture the interdependencies of data sequences. Instead, they use a self-attention head to process data sequences in parallel. This enables transformers to train and process longer sequences in less time than an RNN does. With the self-attention mechanism, transformers overcome the memory limitations and sequence interdependencies that RNNs face. Transformers can process data sequences in parallel and use positional encoding to remember how each input relates to others.

### **Parallelism**

Transformers solve the gradient issues that RNNs face by enabling parallelism during training. By processing all input sequences simultaneously, a transformer isn’t subjected to backpropagation restrictions because gradients can flow freely to all weights. They are also optimized for parallel computing, which graphic processing units (GPUs) offer for generative AI developments. Parallelism enables transformers to scale massively and handle complex NLP tasks by building larger models.

## How can AWS support your RNN requirements?

[Generative AI](https://aws.amazon.com/generative-ai/) on Amazon Web Services (AWS) provides services, tools, and resources that you can use to build, manage, and scale traditional AI applications with advanced transformer-based technology. For example:

* [Amazon SageMaker](https://aws.amazon.com/sagemaker/) is a fully managed service to prepare data and build, train, and deploy ML models for any use case. It has fully managed infrastructure, tools, and workflows.
* [Amazon Bedrock](https://aws.amazon.com/bedrock/) simplifies generative AI development by enabling the customization and deployment of industry-leading [foundation models](https://aws.amazon.com/what-is/foundation-models/) (FM) securely and efficiently.
* [AWS Trainium](https://aws.amazon.com/machine-learning/trainium/) is an ML accelerator that you can use to train and scale deep learning models affordably in the cloud.

Get started with generative AI on AWS by [signing up for an account](https://aws.amazon.com/?nc2=h_lg) today.

**1.long short-term memory**

**What is long short-term memory?**

In the realm of artificial intelligence, *Long Short-Term Memory (LSTM)* refers to a recurrent neural network architecture designed to overcome the limitations of standard neural networks in effectively capturing and utilizing sequential data. Unlike traditional neural networks, LSTM is specifically engineered to retain information over extended durations, making it particularly well-suited for tasks involving temporal dependencies and long-range interactions within data sequences. The primary components of LSTM units, such as the cell state, input gate, forget gate, and output gate, collectively enable the network to preserve and selectively process information, thereby enhancing its ability to model and interpret complex patterns in sequential data. This unique attribute positions LSTM as a foundational element in advancing the capabilities of AI systems, particularly in domains requiring context-aware learning and predictive analysis.

**Background / history of long short-term memory**

Uncover the origins and historical evolution of LSTM, delving into its development and refinement over the years. Highlight key milestones and breakthroughs that have shaped the concept of LSTM as a pivotal component of AI technologies.

The evolutionary trajectory of *Long Short-Term Memory (LSTM)* can be traced back to its inception in the early 1990s, emerging as a response to the inherent limitations of standard recurrent neural networks in effectively capturing and preserving long-term dependencies within sequential data. Spearheaded by the pioneering work of Hochreiter and Schmidhuber in 1997, LSTM introduced a novel architecture equipped with specialized mechanisms for mitigating the issues of vanishing and exploding gradients, which plagued traditional recurrent neural networks during the training process. This foundational refinement marked a significant turning point in the field of AI, laying the groundwork for the development of advanced models capable of effectively processing time-dependent data and facilitating the seamless integration of contextual information into AI-driven applications.

**Significance of long short-term memory**

Explore the crucial role of LSTM in AI and its significance in enabling machines to learn and retain information over extended periods. Discuss the implications of LSTM in enhancing the efficiency and accuracy of AI systems.

The significance of *Long Short-Term Memory (LSTM)* within the paradigm of artificial intelligence resonates deeply with its ability to address the challenges associated with processing and interpreting sequential data. By preserving long-term dependencies and contextual information, LSTM empowers AI systems to sustainably accumulate knowledge, effectively manage varying input sequences, and make informed predictions based on the accumulated insights. This inherent capability underpins the successful deployment of LSTM across diverse AI applications, including language modeling, sentiment analysis, time series forecasting, and speech recognition, amplifying the overall utility and adaptability of AI-enabled solutions.

**How long short-term memory works**

Provide a detailed breakdown of the working principles of LSTM, highlighting its unique features and mechanisms that facilitate the retention and utilization of sequential data in AI applications.

*Long Short-Term Memory (LSTM)* operates on a distinctive set of mechanisms, allowing it to retain and leverage sequential information effectively. The core components of LSTM units – the cell state, input gate, forget gate, and output gate – collectively contribute to the network's ability to process and retain sequential data over extended time spans. The input gate regulates the inflow of new information, the forget gate controls the retention or removal of existing information, and the output gate governs the dissemination of useful knowledge to subsequent layers of the network. Collectively, these components enable LSTM to address the challenges of capturing long-range dependencies and temporal patterns, making it an indispensable asset in AI tasks necessitating contextual understanding and sequential analysis.

**Real-world examples and applications of long short-term memory**

**Example 1 - natural language processing**

In the domain of natural language processing, LSTM serves as a foundational model for various tasks, including language modeling, machine translation, and sentiment analysis. By leveraging the network's ability to retain context and linguistic nuances over extensive textual sequences, AI systems equipped with LSTM can generate coherent responses in conversational interfaces, analyze and categorize sentiment in textual content, and even facilitate the translation of text across different languages, thereby enhancing the efficacy and naturalness of language-based interactions.

**Example 2 - speech recognition**

The application of LSTM in speech recognition showcases its proficiency in processing and deciphering audio sequences, enabling AI systems to accurately transcribe spoken content, facilitate voice commands in smart devices, and support speech-to-text conversion in diverse applications. Through its capacity to capture and interpret subtle variations and nuances in speech patterns, LSTM significantly enhances the accuracy and reliability of speech recognition systems, elevating the overall user experience and accessibility of voice-driven interfaces.

**Example 3 - financial forecasting**

In the realm of financial forecasting and predictive analytics, LSTM plays a critical role in processing time-series data, enabling AI systems to model and predict complex financial trends and market behavior. By retaining and analyzing long-term dependencies within financial data sequences, LSTM empowers AI applications to generate valuable insights for investment decision-making, risk assessment, and trend analysis, thereby augmenting the precision and foresight of financial services and investment strategies.

**Pros & cons of long short-term memory**

Present an analysis of the advantages and limitations of LSTM in AI, addressing its effectiveness, computational complexity, and potential constraints in certain contexts.

The benefits of *Long Short-Term Memory (LSTM)* in AI span a spectrum of advantages:

Pros:

* Capability to preserve long-term dependencies
* Enhances sequential data processing in AI applications
* Mitigates the vanishing and exploding gradient issues
* Facilitates context-aware learning and predictive modeling

Cons:

* Computational complexity in training and deployment
* Vulnerability to overfitting in certain circumstances
* Requirement for extensive parameter tuning and optimization

Despite its inherent computational demands and susceptibility to overfitting, LSTM remains a cornerstone of AI-driven technologies, leveraging its unparalleled retention and utilization of sequential data to navigate complex scenarios and drive transformative outcomes across diverse domains.

**What are the primary advantages of integrating long short-term memory in ai systems?**

The integration of *Long Short-Term Memory (LSTM)* in AI systems confers several advantages, including:

* Enhanced capability to capture and retain long-term dependencies within sequential data
* Facilitation of context-aware learning and prediction in AI applications
* Mitigation of vanishing and exploding gradient issues encountered in traditional recurrent neural networks

**How does long short-term memory differ from conventional neural network architectures?**

Unlike conventional neural network architectures, *Long Short-Term Memory (LSTM)* is specifically designed to retain and process sequential data over extended durations, allowing it to effectively capture long-range dependencies and temporal patterns within data sequences. This pivotal attribute positions LSTM as a foundational model for sequential data processing in AI.

**Can long short-term memory be applied to time-series data analysis in ai?**

Yes, *Long Short-Term Memory (LSTM)* is widely applied to time-series data analysis within the domain of AI, leveraging its capacity to capture and interpret temporal patterns and dependencies. This makes it a valuable asset in tasks such as financial forecasting, stock market analysis, and sensor data processing.

**What are the potential challenges or drawbacks associated with the implementation of long short-term memory in ai applications?**

The implementation of *Long Short-Term Memory (LSTM)* in AI applications may encounter challenges such as:

* Computational complexity during training and deployment
* Susceptibility to overfitting in certain contexts
* Requirement for meticulous parameter tuning and optimization

**Are there any prominent alternatives to long short-term memory for sequential data processing in ai?**

While *Long Short-Term Memory (LSTM)* remains a prominent model for sequential data processing in AI, alternative approaches such as Gated Recurrent Units (GRUs) and Transformer architectures are also utilized for similar applications, offering distinct advantages and trade-offs based on specific contextual requirements.

**Do's and dont's table**

| **Do's** | **Dont's** |
| --- | --- |
| Utilize Long Short-Term Memory for sequence tasks | Over-rely on Long Short-Term Memory for all AI tasks |
| Regularly update Long Short-Term Memory parameters | Neglect the computational cost of Long Short-Term Memory |
| Implement efficient data preprocessing for LSTM | Ignore the potential overfitting issues with LSTM |

This comprehensive guide has illuminated the expansive landscape of *Long Short-Term Memory (LSTM)* in artificial intelligence, underscoring its profound significance, operational intricacies, and far-reaching applications across diverse domains. As AI continues to evolve, the enduring impact of LSTM is poised to unfold new frontiers in intelligent technology, driving unprecedented advancements and reshaping the future of AI-driven innovation.

**2.backpropagation**



## What Is Backpropagation?

Backpropagation is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration.) Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

So how does this process with vast simultaneous mini-executions work? Let’s explore some examples.

In order to make this example as useful as possible, we’re just going to touch on related concepts like [loss functions](https://builtin.com/data-science/loss-functions-deep-learning-python), optimization functions, etc., without explaining them, as these topics require their own articles.

## Advantages of Using the Backpropagation Algorithm in Neural Networks

Before getting into the details of backpropagation in neural networks, let’s review the importance of this [algorithm](https://builtin.com/software-engineering-perspectives/algorithm). Besides improving a neural network, below are a few other reasons why backpropagation is a useful approach:

* No previous knowledge of a neural network is needed, making it easy to implement.
* It’s straightforward to program since there are no other parameters besides the inputs.
* It doesn’t need to learn the features of a function, speeding up the process.
* The model is flexible because of its simplicity and applicable to many scenarios.

## Limitations of Using the Backpropagation Algorithm in Neural Networks

That said, backpropagation is not a blanket solution for any situation involving neural networks. Some of the potential limitations of this model include:

* Training data can impact the performance of the model, so high-quality data is essential.
* [Noisy data](https://builtin.com/data-science/dangers-of-too-much-data) can also affect backpropagation, potentially tainting its results.
* It can take a while to train backpropagation models and get them up to speed.
* Backpropagation requires a matrix-based approach, which can lead to other issues.

Although backpropagation has its flaws, it’s still an effective model for testing and refining the performance of neural networks. Now that we understand the pros and cons of this algorithm, let’s take a deeper look at the ins and outs of backpropagation in neural networks.

## How to Set the Model Components for a Backpropagation Neural Network

Imagine that we have a deep neural network that we need to train. The purpose of training is to build a model that performs the exclusive OR (XOR) functionality with two inputs and three hidden units, such that the training set (truth table) looks something like the following:

X1 | X2 | Y

0 | 0 | 0

0 | 1 | 1

1 | 0 | 1

1 | 1 | 0

We also need an [activation function](https://builtin.com/machine-learning/activation-functions-deep-learning) that determines the activation value at every node in the neural net. For simplicity, let’s choose an identity [activation function](https://builtin.com/machine-learning/relu-activation-function):f(a) = a

We also need a hypothesis function that determines the input to the activation function. This function is going to be the ever-famous:

h(X) = W0.X0 + W1.X1 + W2.X2

            or

h(X) = sigma(W.X) for all (W, X)

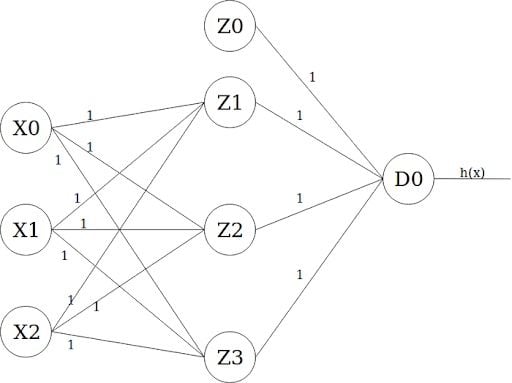
Let’s also make the loss function the usual cost function of [logistic regression](https://builtin.com/data-science/what-is-logistic-regression). It looks a bit complicated, but it’s actually fairly simple:

cost function of linear logistic regression equationCost function of logistic regression equation. | Image: Anas Al-Masri

We’re going to use the batch [gradient descent optimization function](https://builtin.com/data-science/gradient-descent) to determine in what direction we should adjust the weights to get a lower loss than our current one. Finally, we’ll set the learning rate to 0.1 and all the weights will be initialized to one.

## Building a Neural Network

Let’s finally draw a diagram of our long-awaited neural net. It should look something like this:

Model of a neural network. | Image: Anas Al-Masri

The leftmost layer is the [input layer](https://builtin.com/machine-learning/fully-connected-layer), which takes X0 as the bias term of value one, and X1 and X2 as input features. The layer in the middle is the first hidden layer, which also takes a bias term Z0 value of one. Finally, the output layer has only one output unit D0 whose activation value is the actual output of the model (i.e. h(x).)

## How Forward Propagation Works

It is now the time to feed-forward the information from one layer to the next. This goes through two steps that happen at every node/unit in the network:

1. Getting the weighted sum of inputs of a particular unit using the h(x) function we defined earlier.
2. Plugging the value we get from step one into the activation function, we have (f(a)=a, in this example) and using the activation value we get the output of the activation function as the input feature for the connected nodes in the next layer.

Units X0, X1, X2 and Z0 do not have any units connected to them providing inputs. Therefore, the steps mentioned above do not occur in those nodes. However, for the rest of the nodes/units, this is how it all happens throughout the neural net for the first input sample in the training set:

Unit Z1:

h(x) = W0.X0 + W1.X1 + W2.X2

= 1 . 1 + 1 . 0 + 1 . 0

= 1 = a

z = f(a) = a => z = f(1) = 1

Same goes for the remaining units:

Unit Z2:

h(x) = W0.X0 + W1.X1 + W2.X2

= 1 . 1 + 1 . 0 + 1 . 0

= 1 = a

z = f(a) = a => z = f(1) = 1

Unit Z3:

h(x) = W0.X0 + W1.X1 + W2.X2

= 1 . 1 + 1 . 0 + 1 . 0

= 1 = a

z = f(a) = a => z = f(1) = 1

Unit D0:

h(x) = W0.Z0 + W1.Z1 + W2.Z2 + W3.Z3

= 1 . 1 + 1 . 1 + 1 . 1 + 1 . 1

= 4 = a

z = f(a) = a => z = f(4) = 4

As we mentioned earlier, the activation value (z) of the final unit (D0) is that of the whole model. Therefore, our model predicted an output of one for the set of inputs {0, 0}. Calculating the loss/cost of the current iteration would follow:

Loss = actual\_y - predicted\_y

    =    0     -     4

    =    -4

The actual\_y value comes from the training set, while the predicted\_y value is what our model yielded. So the cost at this iteration is equal to -4.

## When Do You Use Backpropagation in Neural Networks?

According to our example, we now have a model that does not give [accurate predictions](https://builtin.com/machine-learning/bias-machine-learning). It gave us the value four instead of one and that is attributed to the fact that its weights have not been tuned yet. They’re all equal to one. We also have the loss, which is equal to -4. Backpropagation is all about feeding this loss backward in such a way that we can fine-tune the weights based on this. The optimization function, gradient descent in our example, will help us find the weights that will hopefully yield a smaller loss in the next iteration. So, let’s get to it.

If [feeding forward](https://builtin.com/data-science/feedforward-neural-network-intro) happened using the following functions: f(a) = a

Optimization function in gradient descent equationOptimization function equation in a gradient descent. | Image: Anas Al-Masri

Then feeding backward will happen through the partial derivatives of those functions. There is no need to go through the equation to arrive at these derivatives. All we need to know is that the above functions will follow:

f'(a) = 1

J'(w) = Z . delta

Z is just the z value we obtained from the activation function calculations in the feed-forward step, while delta is the loss of the unit in the layer.

I know it’s a lot of information to absorb in one sitting, but I suggest you take your time to really understand what is going on at each step before going further.

A video tutorial on the basics of backpropagation. | Video: 3Blue1Brown

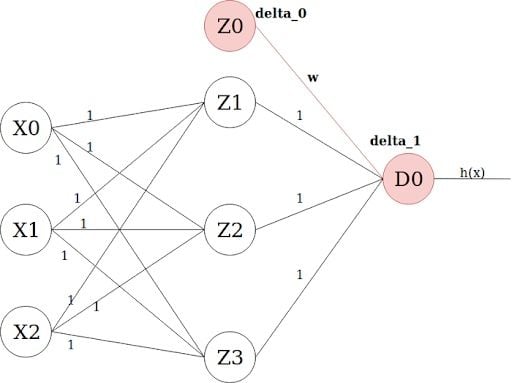
## How to Calculate Deltas in Backpropagation Neural Networks

Now we need to find the loss at every unit/node in the neural net. Why is that? Well, think about it this way: Every loss the deep learning model arrives at is actually the mess that was caused by all the nodes accumulated into one number. Therefore, we need to find out which node is responsible for the most loss in every layer, so that we can penalize it by giving it a smaller weight value, and thus lessening the total loss of the model.

Calculating the delta for every unit can be problematic. However, thanks to computer scientist and founder of DeepLearning, [Andrew Ng](https://www.andrewng.org/), we now have a shortcut formula for the whole thing:

delta\_0 = w . delta\_1 . f'(z)

Where values delta\_0, w and f’(z) are those of the same unit’s, while delta\_1 is the loss of the unit on the other side of the weighted link. For example:

A neural network model going through backpropagation. | Image: Anas Al-Masri

In order to get the loss of a node (e.g. Z0), we multiply the value of its corresponding f’(z) by the loss of the node it is connected to in the next layer (delta\_1), by the weight of the link connecting both nodes.

This is how backpropagation works. We do the delta calculation step at every unit, backpropagating the loss into the neural net, and find out what loss every node/unit is responsible for.

Let’s calculate those deltas.

delta\_D0 = total\_loss = -4

delta\_Z0 = W . delta\_D0 . f'(Z0) = 1 . (-4) . 1 = -4

delta\_Z1 = W . delta\_D0 . f'(Z1) = 1 . (-4) . 1 = -4

delta\_Z2 = W . delta\_D0 . f'(Z2) = 1 . (-4) . 1 = -4

delta\_Z3 = W . delta\_D0 . f'(Z3) = 1 . (-4) . 1 = -4

There are a few things to note here:

* The loss of the final unit (i.e. D0) is equal to the loss of the whole model. This is because it is the output unit, and its loss is the accumulated loss of all the units together.
* The function f’(z)will always give the value one, no matter what the input (i.e. z) is equal to. This is because the partial derivative, as we said earlier, follows: f’(a) = 1
* The input nodes/units (X0, X1 and X2) don’t have delta values, as there is nothing those nodes control in the neural net. They are only there as a link between the data set and the neural net. This is why the whole layer is usually not included in the layer count.

MORE ON AI[How to Get Started With Regression Trees](https://builtin.com/data-science/regression-tree)

## Updating the Weights in Backpropagation for a Neural Network

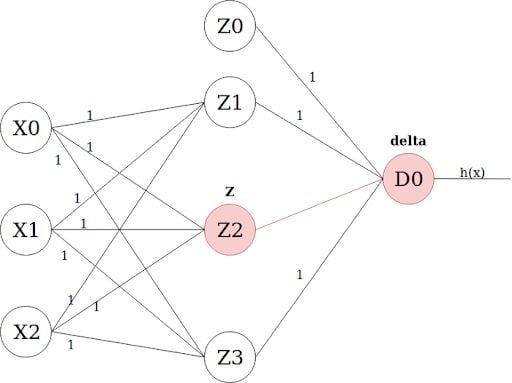
All that’s left is to update all the weights we have in the neural net. This follows the batch gradient descent formula:

W := W - alpha . J'(W)

Where W is the weight at hand, alpha is the learning rate (i.e. 0.1 in our example) and J’(W) is the partial derivative of the cost function J(W) with respect to W. Again, there’s no need for us to get into the math. Therefore, let’s use Mr. Andrew Ng’s partial derivative of the function:

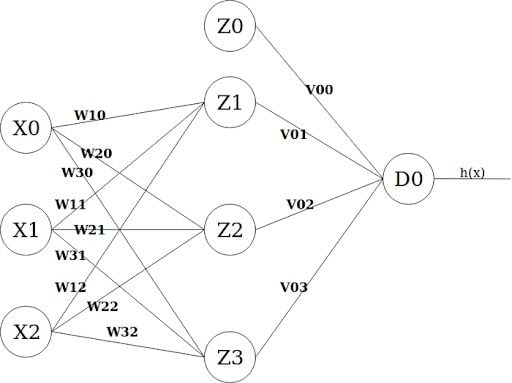
J'(W) = Z . delta

Where Z is the Z value obtained through forward propagation, and delta is the loss at the unit on the other end of the weighted link:

Weighted links added to the neural network model. | Image: Anas Al-Masri

Now we use the batch gradient descent weight update on all the weights, utilizing our partial derivative values that we obtain at every step. It is worth emphasizing that the Z values of the input nodes (X0, X1, and X2) are equal to one, zero, zero, respectively. The one is the value of the bias unit, while the zeroes are actually the feature input values coming from the [data set](https://builtin.com/data-science/what-is-a-data-set). There is no particular order to updating the weights. You can update them in any order you want, as long as you don’t make the mistake of updating any weight twice in the same iteration.

In order to calculate the new weights, let’s give the links in our neural nets names:

Neural network model with updated link names to calculate the new weights. | Image: Anas Al-Masri

New weight calculations will happen as follows:

W10 := W10 - alpha . Z\_X0 . delta\_Z1

= 1 - 0.1 . 1 . (-4) = 1.4

W20 := W20 - alpha . Z\_X0 . delta\_Z2

= 1 - 0.1 . 1 . (-4) = 1.4

. . . . .

. . . . .

. . . . .

W30 := 1.4

W11 := 1.4

W21 := 1.4

W31 := 1.4

W12 := 1.4

W22 := 1.4

W32 := 1.4

V00 := V00 - alpha . Z\_Z0 . delta\_D0

= 1 - 0.1 . 1 . (-4) = 1.4

V01 := 1.4

V02 := 1.4

V03 := 1.4

The model is not trained properly yet, as we only back-propagated through one sample from the training set. Doing everything all over again for all the samples will yield a model with better accuracy as we go, with the aim of getting closer to the minimum loss/cost at every step.

It might not make sense that all the weights have the same value again. However, training the model on different samples over and over again will result in nodes having different weights based on their contributions to the total loss.

The theory behind machine learning can be really difficult to grasp if it isn’t tackled the right way. One example of this would be backpropagation, whose effectiveness is visible in most real-world [deep learning applications](https://builtin.com/artificial-intelligence/deep-learning-applications), but it’s never examined. Backpropagation is just a way of propagating the total loss back into the [neural network](https://builtin.com/data-science/recurrent-neural-networks-and-lstm) to know how much of the loss every node is responsible for, and subsequently updating the weights in a way that minimizes the loss by giving the nodes with higher error rates lower weights, and vice versa.

## Best Practices for Optimizing Backpropagation

Backpropagation in a neural network is designed to be a seamless process, but there are still some best practices you can follow to make sure a backpropagation algorithm is operating at peak performance.

### SELECT A TRAINING METHOD

The pace of the training process depends on the method you choose. Going with a stochastic gradient descent speeds up the training, but the actual fine-tuning of the backpropagation algorithm can be tedious. On the other hand, batch gradient descent is easier to perform, but the overall learning process takes longer. For these reasons, the stochastic approach is preferred, but it’s important to pick a training method that best fits your circumstances.

### PROVIDE PLENTY OF DATA

Feeding a backpropagation algorithm lots of data is key to reducing the amount and types of errors it produces during each iteration. The size of your data set can vary, depending on the learning rate of your algorithm. In general, though, it’s better to include larger data sets since models can gain broader experiences and lessen their mistakes in the future.

### CLEAN ALL DATA

Backpropagation training is much smoother when the training data is of the highest quality, so [clean your data](https://builtin.com/data-science/data-preparation-cleaning) before feeding it to your algorithm. This means [normalizing the input values](https://builtin.com/data-science/database-normalization), which involves checking that the mean of the data is zero and the data set has a standard deviation of one. A backpropagation algorithm can then more easily analyze the data, leading to faster and more accurate results.

### CONSIDER THE IMPACTS OF LEARNING RATE

Deciding on the learning rate for training a backpropagation model depends on the size of the data set, the type of problem and other factors. That said, a higher learning rate can lead to faster results, but not the optimal performance. A lower learning rate produces slower results, but can lead to a better outcome in the end. You’ll want to consider which learning rate best applies to your situation, so you don’t under- or overshoot your desired outcome.

### TEST THE MODEL WITH DIFFERENT EXAMPLES

To get a sense of how well a backpropagation model performs, it helps to test the algorithm with data not used during the training period. Compiling diverse data also exposes the model to different situations and tests how well it can adapt to a range of scenarios. You can then make more informed adjustments to enhance the algorithm’s learning process.

# **Activation Functions in Neural Networks**

A paradigm for information processing that draws inspiration from the brain is called an artificial neural network (ANN). ANNs learn via imitation just like people do. Through a learning process, an ANN is tailored for a particular purpose, including such pattern classification or data classification. The synapses interconnections that exist between both the neurons change because of learning.

What input layer to employ with in hidden layer and at the input level of the network is one of the decisions you get to make while creating a neural network. This article discusses a few of the alternatives.

The nerve impulse in neurology serves as a model for activation functions within computer science. A chain reaction permits a neuron to "fire" and send a signal to nearby neurons if the induced voltage between its interior and exterior exceeds a threshold value known as the action potential. The next series of activations, known as a "spike train," enables motor neurons to transfer commands from of the brain to the limbs and sensory neurons too transmit sensation from the digits to the brain.

## Neural Network Components

Layers are the vertically stacked parts that make up a neural network. The image's dotted lines each signify a layer. A NN has three different types of layers.

ADVERTISEMENT

### Input Layer

The input layer is first. The data will be accepted by this layer and forwarded to the remainder of the network. This layer allows feature input. It feeds the network with data from the outside world; no calculation is done here; instead, nodes simply transmit the information (features) to the hidden units.

### Hidden Layer

Since they are a component of the abstraction that any neural network provides, the nodes in this layer are not visible to the outside world. Any features entered through to the input layer are processed by the hidden layer in any way, with the results being sent to the output layer. The concealed layer is the name given to the second kind of layer. For a neural network, either there are one or many hidden layers. The number inside the example above is 1. In reality, hidden layers are what give neural networks their exceptional performance and intricacy. They carry out several tasks concurrently, including data transformation and automatic feature generation.

### Output Layer

This layer raises the knowledge that the network has acquired to the outside world. The output layer is the final kind of layer The output layer contains the answer to the issue. We receive output from the output layer after passing raw photos to the input layer.

Data science makes extensive use of the rectified unit (ReLU) functional or the category of sigmoid processes, which also includes the logistic regression model, logistic hyperbolic tangent, and arctangent function.

## Activation Function

### Definition

In artificial neural networks, an activation function is one that outputs a smaller value for tiny inputs and a higher value if its inputs are greater than a threshold. An activation function "fires" if the inputs are big enough; otherwise, nothing happens. An activation function, then, is a gate that verifies how an incoming value is higher than a threshold value.

Because they introduce non-linearities in neural networks and enable the neural networks can learn powerful operations, activation functions are helpful. A feedforward neural network might be refactored into a straightforward linear function or matrix transformation on to its input if indeed the activation functions were taken out.

By generating a weighted total and then including bias with it, the activation function determines whether a neuron should be turned on. The activation function seeks to boost a neuron's output's nonlinearity.

**Explanation**: As we are aware, neurons in neural networks operate in accordance with weight, bias, and their corresponding activation functions. Based on the mistake, the values of the neurons inside a neural network would be modified. This process is known as back-propagation. Back-propagation is made possible by activation functions since they provide the gradients and error required to change the biases and weights.

## Need of Non-linear Activation Functions

An interconnected regression model without an activation function is all that a neural network is. Input is transformed nonlinearly by the activation function, allowing the system to learn and perform more challenging tasks.

It is merely a thing procedure that is used to obtain a node's output. It also goes by the name Transfer Function.

The mixture of two linear functions yields a linear function, so no matter how several hidden layers we add to a neural network, they all will behave in the same way. The neuron cannot learn if all it has is a linear model. It will be able to learn based on the difference with respect to error with a non-linear activation function.

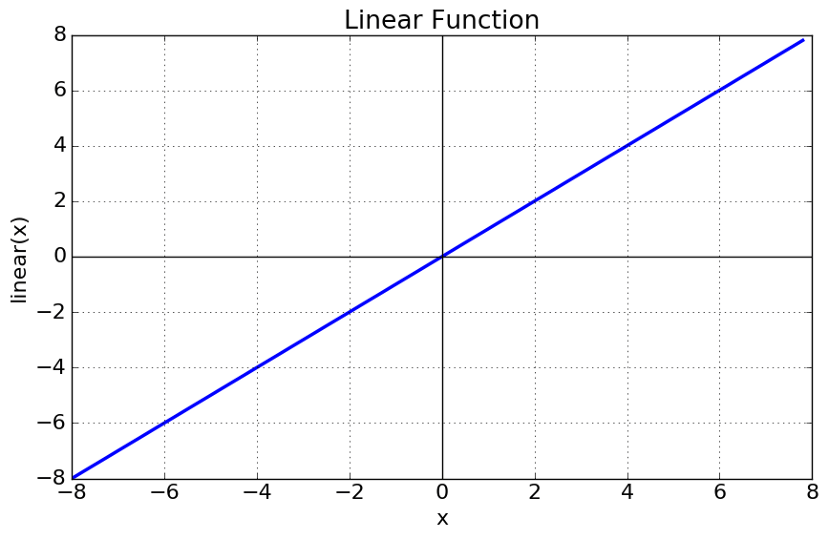
The mixture of two linear functions yields a linear function in itself, so no matter how several hidden layers we add to a neural network, they all will behave in the same way. The neuron cannot learn if all it has is a linear model.

The two main categories of activation functions are:

* Linear Activation Function
* Non-linear Activation Functions

### Linear Activation Function

As can be observed, the functional is linear or linear. Therefore, no region will be employed to restrict the functions' output.



The normal data input to neural networks is unaffected by the complexity or other factors.

### Non-linear Activation Function

The normal data input to neural networks is unaffected by the complexity or other factors.

## Activation Function

* **Linear Function**

Equation: A linear function's equation, which is y = x, is similar to the eqn of a single direction.

The ultimate activation function of the last layer is nothing more than a linear function of input from the first layer, regardless of how many levels we have if they are all linear in nature. -inf to +inf is the range.

Uses: The output layer is the only location where the activation function's function is applied.

If we separate a linear function to add non-linearity, the outcome will no longer depend on the input "x," the function will become fixed, and our algorithm won't exhibit any novel behaviour.

A good example of a regression problem is determining the cost of a house. We can use linear activation at the output layer since the price of a house may have any huge or little value. The neural network's hidden layers must perform some sort of non-linear function even in this circumstance.

* **Sigmoid Function**

It is a functional that is graphed in a "S" shape.

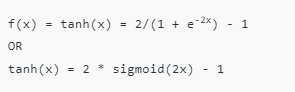
A is equal to 1/(1 + e-x).

Non-linear in nature. Observe that while Y values are fairly steep, X values range from -2 to 2. To put it another way, small changes in x also would cause significant shifts in the value of Y. spans from 0 to 1.

Uses: Sigmoid function is typically employed in the output nodes of a classi?cation, where the result may only be either 0 or 1. Since the value for the sigmoid function only ranges from 0 to 1, the result can be easily anticipated to be 1 if the value is more than 0.5 and 0 if it is not.

* **Tanh Function**

The activation that consistently outperforms sigmoid function is known as tangent hyperbolic function. It's actually a sigmoid function that has been mathematically adjusted. Both are comparable to and derivable from one another.



Range of values: -1 to +1. non-linear nature

Uses: - Since its values typically range from -1 to 1, the mean again for hidden layer of a neural network will be 0 or very near to it. This helps to centre the data by getting the mean close to 0. This greatly facilitates learning for the following layer.

**Equation:**

max A(x) (0, x). If x is positive, it outputs x; if not, it outputs 0.

Value Interval: [0, inf]

Nature: non-linear, which allows us to simply backpropagate the mistakes and have the ReLU function activate many layers of neurons.

Uses: Because ReLu includes simpler mathematical processes than tanh and sigmoid, it requires less computer time to run. The system is sparse and efficient for computation since only a limited number of neurons are activated at any given time.

Simply said, RELU picks up information considerably more quickly than sigmoid and Tanh functions.

* **ReLU (Rectified Linear Unit) Activation Function**

Currently, the ReLU is the activation function that is employed the most globally. Since practically all convolutional neural networks and deep learning systems employ it.

The derivative and the function are both monotonic.

However, the problem is that all negative values instantly become zero, which reduces the model's capacity to effectively fit or learn from the data. This means that any negative input to a ReLU activation function immediately becomes zero in the graph, which has an impact on the final graph by improperly mapping the negative values.

ADVERTISEMENT

* **Softmax Function**

Although it is a subclass of the sigmoid function, the softmax function comes in handy when dealing with multiclass classification issues.

Used frequently when managing several classes. In the output nodes of image classification issues, the softmax was typically present. The softmax function would split by the sum of the outputs and squeeze all outputs for each category between 0 and 1.

The output unit of the classifier, where we are actually attempting to obtain the probabilities to determine the class of each input, is where the softmax function is best applied.

The usual rule of thumb is to utilise RELU, which is a usual perceptron in hidden layers and is employed in the majority of cases these days, if we really are unsure of what encoder to apply.

A very logical choice for the output layer is the sigmoid function if your input is for binary classification. If our output involves multiple classes, Softmax can be quite helpful in predicting the odds for each class.

**Deep neural network**

A Deep Neural Network (DNN) is a machine learning technique that allows a computer, by training it, to do tasks that would be very difficult to do using conventional programming techniques. Neural network algorithms were inspired by the human brain and its functions: like our human mind, it is designed to work not only by following a preset list of rules, but by predicting solutions and drawing conclusions based on previous iterations and experiences.

### What is neural network architecture?

A neural network is composed of multiple layers of nodes that receive input from other layers and produce an output until a final result is reached. Neural networks can have any number of hidden layers: the more layers of nodes are in the network, the higher the complexity. Here are different neural network architectures:

1. **Traditional neural networks**, usually composed of 2 or 3 hidden layers;
2. **Deep learning network**, that can have up to 150 hidden layers;

### What is the difference between neural networks and deep neural networks?

A deep neural network is a much more complicated system than a “simple” neural system. A neural network is comparable to a chess game, and behaves according to algorithms: different tactics will be suggested according to the opponent’s moves and actions. This neural network will be limited to what the programmer’s input:

* How do the chess pieces move;
* The size of the chessboard;
* Different strategies for different situations;

#### A neural network that goes beyond the input data and can learn from previous experiences becomes a deep neural network

On the same computer you could, for example, train a neural network, then play with it against other people and let it learn as it played. From there, if a neural network can learn from different players, it may become difficult, or literally impossible, to defeat a deep neural network, even for chess masters.

Deep neural networks can recognize voice commands, identify voices, recognize sounds and graphics and do much more than a neural network. Deep learning networks utilize "Big Data'' along with algorithms in order to solve a problem, and these deep neural networks can solve problems with limited or no human input.

##### How to visualize the work of a deep neural network?

A deep neural network’s process is best understood by looking at an example. Imagine you had hundreds of thousands of images, some of which had dogs in them, and you decided you wanted to write a computer program to recognize dogs in pictures.

You have two choices. You can write a program to explicitly identify dogs, or you can write a program that “learns” how to identify dogs.

You unwisely decide to try to do the former.

### Using conventional programming techniques is long and difficult, and results aren't always accurate

In order to identify dog pictures, you create a software program using “if” and “then” statements where the likelihood that you are looking at a dog is programmed to increase every time you identify a doglike attribute such as fur, floppy ears and a tail. However, this type of neural system is difficult on many levels:

For example if a clump of pixels resembles a tail increase the likelihood that you are looking at a dog. Your deep neural network needs to identify groups of pixels that correspond to the doglike attributes. Even if you manage to do that, there are issues for your algorithm:

* Many photographed objects share some of the dog-like attributes, especially photographs of similar animals. You will need to add rules;
* Sometimes the attributes are there but obscured. Your algorithms will not pick them up;
* Sometimes attributes are only of importance when other attributes are present. More decision rules will be needed;

Your classification fails. You realize you cannot manually identify the complete set of attributes let alone devise all the rules needed to deal with all these special cases.

You wisely give up and decide to try the latter approach. To use a neural network or, even better, a deep neural network.

## A deep learning model can save you coding time and offer better results

The neural network is so named because there is a similarity between this programming approach and the way the brain works.

Just like the brain, the neural net algorithms use a network of neurons or nodes. And like the brain, these neurons are discrete functions (or little machines if you like) that take in inputs and generate outputs. These nodes are arranged in layers whereby the outputs of neurons in one layer become the inputs to neurons in the next layer until the neurons on the outer layer of the network generate the final result.

There are therefore layers of neurons with each individual neuron receiving very limited inputs and generating very limited outputs just like in the brain. The first layer (or input layer) of neurons takes in the inputs and the last layer of neurons (or output layer) in the network outputs the result.

### Is it accurate to call this type of algorithm a “neural network”?

The human brain is far more complex and powerful than a neural network of course. Naming the algorithm a “deep neural network” was a branding coup but it may create unrealistic expectations about what is achievable with these techniques. That said there are people trying to re-engineer the brain, using a very complex neural network, in the hope that by doing this they will be able to replicate general, human-like intelligence in bot development. So how does a neural net and machine learning techniques help us with our dog recognition problem?

Well, instead of manually defining dog-like attributes, a deep neural network algorithm can identify the important attributes and deal with all the special cases without programming.

## How does a deep neural network function?

It does this as follows:

Each neuron on the input layer receives a bit of information from the image as an input and then randomly weights (between zero and one) whether that information suggests a dog or not. A low weight (less than 0.5) means it's less likely that the information is associated with a dog and a high weight means it's more likely the information is associated with a dog. This multilayer neural network approach is called deep learning. Neural networks and deep learning are very powerful techniques for achieving computer comprehension.

#### Deep neural networks are composed of multiple layers of nodes behaving like neurons in our brain

So to continue on deep neural networks, the weights of these neurons are then fed as inputs into the other layers of neurons which also randomly assign weights and pass them on as inputs to yet more neurons in the network. This continues until the output layer of neurons gives a binary verdict. If the average of the weights passed to them is greater than 0.5, it's a dog otherwise it's not. These connections between and activation of neurons across a multiple layers of nodes are what give deep neural network applications their power.

### How does a deep neural network know if it produces the right answer?

Relevant questions at this point are: has the deep neural network guessed correctly or not and what happens if it has or has not guessed correctly? And how does a neural network know if it's guessed correctly or not?

One way it would know is if you undertook the extremely time consuming classification task of labelling all the photographs “dog” or “not dog” depending as to whether there is a dog in the photo or not. The neural net will simply look at the label to see if it correctly identified the dog or not.

And of course we are not interested in whether it got the “Dog or not” question right on a single dog picture. We are interested in whether it got the question right for every photo, or at least to find out what percentage of the time it was accurate in assessing if there was a dog or not in the picture.

## How do deep neural networks use training to learn?

#### A neural network learns from each iteration through data to improve its accuracy

For a given set of weights across all neurons in the network, the neural net will make guesses for all photos and then determine how accurate it was. What percentage of the time did it get the right result i.e. say the dog was in the photo when it was in the photo, and how many times did it get the wrong result, say the dog was in the photo when it wasn't or say the dog wasn't in the photo when it was. This indication as to how accurate the AI algorithm is essential feedback for the neural network model.

Once it has run through all the photos once, it can randomly (or otherwise) adjust some of the weights and then do the whole exercise of guessing what is in the photo again. If the result from the second run is better, it will keep instead of reverting back to the previous set of weights. If the result from the second run was worse it may revert to the previous set of weights and then try different modifications to those weights.

This process will carry on in this way until the neural net becomes good at identifying dogs in photos (hopefully).

When the algorithm can accurately identify dogs it is said to have converged. It has been successfully “trained” to identify dogs.

## What are the different types of neural networks?

### Convolutional neural networks

Convolutional neural networks (CNN) are a type of artificial intelligence designed to process, or learn from, large data sets. Convolutional neural network is a newly coined term specifically describing this type of network, or AI technology in general.

CNNs are powerful image recognition AI tools that use deep learning to perform not only generative tasks, but also descriptive tasks. Examples of generative tasks include auto-cropping, caption writing, videography, mimeograph, and image overlays. A convolutional neural network contains what are called convolutional layers. Each neuron in these layers only processes information from a small part of a visual field. Each neuron’s inputs are lined up in a checksum-like fashion to generate a feature map.

### Artificial neural networks

An artificial neural network (ANN) is a network of many perceptrons at different depths or layers that can be understood as a Logistic Regression. ANN is often called Feed Forward Neural Network because inputs are processed only in the forward direction: a layer receives input and sends an output in a linear fashion.

Artificial neural networks are also known as universal function approximators. Pure neural network algorithms such as ANNs and mapping functions that can be implemented as deep learning allow computers to learn any function. One of the reasons why universal approximation is important is the activation function. Activation functions introduce nonlinear properties into the network while learning any complex relationship between input and output. It helps different kinds of the network learn from each other.

## How does a deep neural network improve over time?

One way to imagine what the algorithm is doing is to imagine each neuron as a kind of certainty test. Instead of coding all those if-then statements to identify dogs, each neuron is calibrated to add or take away from the final judgment that the object in the photo is a dog. It's as though the judgment (such as dog or not) is split into a large number of connected judgements that contribute in aggregate to a final judgment.

Of course the primary objective is to achieve convergence if that is possible. It's also an important objective to do this in a reasonable amount of time, preferably in a short time.

#### Deep learning algorithms have a learning process that makes them difficult to understand for humans

What is interesting is that the logic that allows neural networks to identify the dogs in the picture is not human understandable. Deep learning models have hidden logic, essentially a black box of hidden layers of nodes creating its own deep network. That said, there have been some attempts to try to visually represent the logic behind neural networks for image recognition tasks. For other cases, it is not possible to see what the algorithm is doing behind the scenes and the deep learning methods remain hidden.

##### Neural networks and machine learning are popular now but many of these algorithms were known around 50 years ago.

## Why are deep neural networks increasingly popular in various industries?

One of the primary reasons that neural nets are much more popular now than when they were first invented, is that processing power is faster and cheaper than it was. Computing power has made all the difference in achieving fast convergence. The other reason is that data is now ubiquitous which increases the value of algorithms that can make use of the data like chatbots for business

### Advanced neural networks require high processing power and a lot of data

Deep learning neural networks are data and processor-hungry techniques that can achieve results that would be impossible for programmers using programming techniques to achieve. They are ideally suited to certain problems where ubiquitous data is available and it is easy to categorize or rank preferable outcomes.

Without having hundreds of thousands or preferably millions of photos of dogs it would be impossible to train the algorithm. These techniques only work when a lot of data is available. This is fairly obvious as all the special cases are unlikely to be represented in a set of just 1000 photos.

### Neural networks can be faced with structured data or unstructured data

One problem in the above example is that much manual work is involved in labelling all the photos. It is easier for the algorithms to use data that is labelled in a structured way. Neural network machine learning that uses structured data is called supervised learning.

This brings us to a central question: is it possible to avoid all of that tagging work? That would be good because not only could you avoid a lot of manual work but also most of the data available on the internet is unstructured i.e. is not carefully labelled or structured.

## Can neural networks work with unstructured data?

Artificial neural networks and machine learning that works with unstructured data are called unsupervised learning. Of course, this is the holy grail of machine learning and is more analogous to how humans learn. However, even unsupervised learning by machines requires much more data to “learn” than humans do and machines cannot easily extrapolate to examples that are outside what they have been trained on.

#### Many deep learning models try to reproduce the human brain processes

Some people believe that these types of algorithms can be developed, perhaps by re-engineering the brain, to the point that the algorithms start to approach a human-level “understanding”. They believe that it will be possible to use sophisticated scanning technology of the brain to give us insights into how the neural networks of the brain actually work. By copying these designs and patterns we can replicate human-level intelligence.

While the techniques are no doubt ingenious and very useful, especially where a large data set is available, it is hard to imagine that such simple algorithms could be the basis of a highly creative human-like intelligence.

‍

Top of Form

Bottom of Form