

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800

Open access books available

122,000

International authors and editors

135M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Deep Learning: Exemplar Studies in Natural Language Processing and Computer Vision

Selma Tekir and Yalin Bastanlar

Abstract

Deep learning has become the most popular approach in machine learning in recent years. The reason lies in considerably high accuracies obtained by deep learning methods in many tasks especially with textual and visual data. In fact, natural language processing (NLP) and computer vision are the two research areas that deep learning has demonstrated its impact at utmost level. This chapter will firstly summarize the historical evolution of deep neural networks and their fundamental working principles. After briefly introducing the natural language processing and computer vision research areas, it will explain how exactly deep learning is used to solve the problems in these two areas. Several examples regarding the common tasks of these research areas and some discussion are also provided.

Keywords: deep learning, machine learning, natural language processing, computer vision, transfer learning

1. Introduction

Early approaches of artificial intelligence (AI) have sought solutions through formal representation of knowledge and applying logical inference rules. Later on, with having more data available, machine learning approaches prevailed which have the capability of learning from data. Many successful examples today, such as language translation, are results of this data-driven approach. When compared to other machine learning approaches, deep learning (deep artificial neural networks) has two advantages. It benefits well from vast amount of data—more and more of what we do is recorded every day, and it does not require defining the features to be learned beforehand. As a consequence, in the last decade, we have seen numerous success stories achieved with deep learning approaches especially with textual and visual data.

In this chapter, first a relatively short history of neural networks will be provided, and their main principles will be explained. Then, the chapter will proceed to two parallel paths. The first path treats text data and explains the use of deep learning in the area of natural language processing (NLP). Neural network methods first transformed the core task of language modeling. Neural language models have been introduced, and they superseded n-gram language models. Thus, initially the task of language modeling will be covered. The primary focus of this part will be

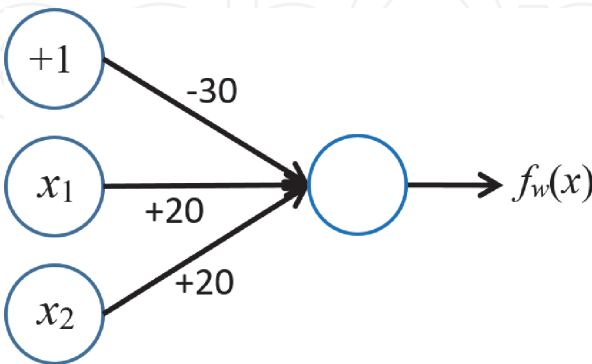
representation learning, where the main impact of deep learning approaches has been observed. Good dense representations are learned for words, senses, sentences, paragraphs, and documents. These embeddings are proved useful in capturing both syntactic and semantic features. Recent works are able to compute contextual embeddings, which can provide different representations for the same word in different contextual units. Consequently, state-of-the-art embedding methods along with their applications in different NLP tasks will be stated as the use of these pre-trained embeddings in various downstream NLP tasks introduced a substantial performance improvement.

The second path concentrates on visual data. It will introduce the use of deep learning for computer vision research area. In this aim, it will first cover the principles of convolutional neural networks (CNNs)—the fundamental structure while working on images and videos. On a typical CNN architecture, it will explain the main components such as convolutional, pooling, and classification layers. Then, it will go over one of the main tasks of computer vision, namely, image classification. Using several examples of image classification, it will explain several concepts related to training CNNs (regularization, dropout and data augmentation). Lastly, it will provide a discussion on visualizing and understanding the features learned by a CNN. Based on this discussion, it will go through the principles of how and when transfer learning should be applied with a concrete example of real-world four-class classification problem.

2. Historical evolution of neural networks and their fundamental working principles

2.1 Historical evolution of neural networks

Deep neural networks currently provide the best solutions to many problems in computer vision and natural language processing. Although we have been hearing the success news in recent years, artificial neural networks are not a new research area. In 1943, McCulloch and Pitts [1] built a neuron model that sums binary inputs, and outputs 1 if the sum exceeds a certain threshold value, and otherwise outputs 0. They demonstrated that such a neuron can model the basic OR/AND/NOT



$$f_w(x) = w_0 + w_1x_1 + w_2x_2$$

$$f_w(x) = -30+20x_1+20x_2$$

Figure 1. A neuron that mimics the behavior of logical AND operator. It multiplies each input (x_1 and x_2) and the bias unit (+1) with a weight and thresholds the sum of these to output 1 if the sum is big enough (similar to our neurons that either fire or not).

functions (**Figure 1**). Such structures are called neurons due to the biological inspiration: inputs (x_i) represent activations from nearby neurons, weights (w_i) represent the synapse strength to nearby neurons, and activation function (f_w) is the cell body, and if the function output is strong enough, it will be sensed by the synapses of nearby neurons.

In 1957, Rosenblatt introduced perceptrons [2]. The idea was not different from the neuron of McCulloch and Pitts, but Rosenblatt came up with a way to make such artificial neurons learn. Given a training set of input-output pairs, weights are increased/decreased depending on the comparison between the perceptron's output and the correct output. Rosenblatt also implemented the idea of the perceptron in custom hardware and showed it could learn to classify simple shapes correctly with 20×20 pixel-like inputs (**Figure 2**).

Marvin Minsky who was the founder of MIT AI Lab and Seymour Papert together wrote a book related to the analysis on the limitations of perceptrons [4]. In this book, as an approach of AI, perceptrons were thought to have a dead end. A single layer of neurons was not enough to solve complicated problems, and Rosenblatt's learning algorithm did not work for multiple layers. This conclusion caused a declining period for the funding and publications on AI, which is usually referred to as "AI winter."

Paul Werbos proposed that backpropagation can be used in neural networks [5]. He showed how to train multilayer perceptrons in his PhD thesis (1974), but due to the AI winter, it required a decade for researchers to work in this area. In 1986, this approach became popular with "Learning representations by back-propagating errors" by Rumelhart et al. [6]. First time in 1989, it was applied to a computer vision task which is handwritten digit classification [7]. It has demonstrated excellent performance on this task. However, after a short while, researchers started to face problems with the backpropagation algorithm. Deep (multilayer) neural networks trained with backpropagation did not work very well and particularly did not work as well as networks with fewer layers. It turned out that the magnitudes of

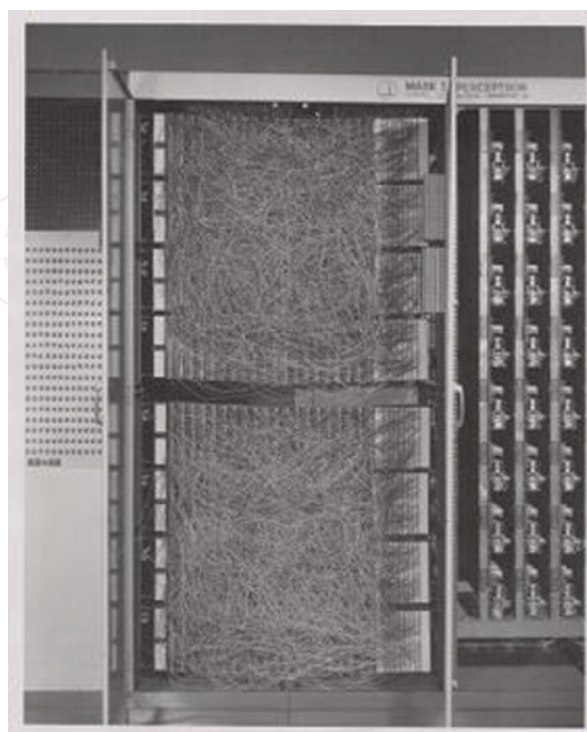


Figure 2.
 Mark I Perceptron at the Cornell Aeronautical Laboratory, hardware implementation of the first perceptron
 (source: Cornell University Library [3]).

backpropagated errors shrink very rapidly and this prevents earlier layers to learn, which is today called as “the vanishing gradient problem.” Again it took more than a decade for computers to handle more complex tasks. Some people prefer to name this period as the second AI winter.

Later, it was discovered that the initialization of weights has a critical importance for training, and with a better choice of nonlinear activation function, we can avoid the vanishing gradient problem. In the meantime, our computers got faster (especially thanks to GPUs), and huge amount of data became available for many tasks. G. Hinton and two of his graduate students demonstrated the effectiveness of deep networks at a challenging AI task: speech recognition. They managed to improve on a decade-old performance record on a standard speech recognition dataset. In 2012, a CNN (again G. Hinton and students) won against other machine learning approaches at the Large Scale Visual Recognition Challenge (ILSVRC) image classification task for the first time.

2.2 Working principles of a deep neural network

Technically any neural network with two or more hidden layers is “deep.” However, in papers of recent years, deep networks correspond to the ones with many more layers. We show a simple network in **Figure 3**, where the first layer is the input layer, the last layer is the output layer, and the ones in between are the hidden layers.

In **Figure 3**, $a_j^{(i)}$ denotes the value after activation function is applied to the inputs in j th neuron of i th layer. If the predicted output of the network, which is $a_1^{(4)}$ in this example, is close to the actual output, then the “loss” is low. Previously mentioned backpropagation algorithm uses derivatives to carry the loss to the previous layers. $\frac{\partial L}{\partial a_1^{(4)}}$ represents the derivative of loss with respect to $a_1^{(4)}$, whereas $\frac{\partial L}{\partial a_1^{(2)}}$ represents the derivative of loss with respect to a second layer neuron $a_1^{(2)}$. The derivative of loss with respect to $a_1^{(2)}$ means how much of the final error (loss) is neuron $a_1^{(2)}$ responsible for.

Activation function is the element that gives a neural network its nonlinear representation capacity. Therefore, we always choose a nonlinear function. If activation function was chosen to be a linear function, each layer would perform a linear mapping of the input to the output. Thus, no matter how many layers were there, since linear functions are closed under composition, this would be equivalent to having a single (linear) layer.

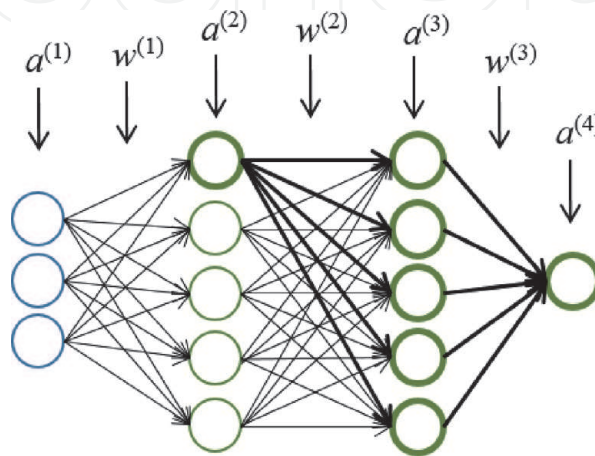


Figure 3.

A simple neural network with two hidden layers. Entities plotted with thicker lines are the ones included in Eq. (1), which will be used to explain the vanishing gradient problem.

The choice of activation function is critically important. In early days of multi-layer networks, people used to employ *sigmoid* or *tanh*, which cause the problem named as vanishing gradient. Let's explain the vanishing gradient problem with the network shown in **Figure 3**.

$$\frac{\partial L}{\partial a_1^{(2)}} = w^{(2)} \cdot \sigma'(z^{(3)}) \cdot w^{(3)} \cdot \sigma'(z_1^{(4)}) \cdot \frac{\partial L}{\partial a_1^{(4)}} \tag{1}$$

Eq. (1) shows how the error in the final layer is backpropagated to a neuron in the first hidden layer, where $w^{(i)}$ denotes the weights in layer i and $z_j^{(i)}$ denotes the weighted input to the j th neuron in layer i . Here, let's assume *sigmoid* is used as the activation function. Then, $a_j^{(i)}$ denotes the value after the activation function is applied to $z_j^{(i)}$, i.e., $a_j^{(i)} = \sigma(z_j^{(i)})$. Finally, let σ' denote the derivative of *sigmoid* function. Entities in Eq. (1) are plotted with thicker lines in **Figure 3**.

Figure 4 shows the derivative of *sigmoid*, where we observe that the highest point derivative is equal to 25% of its original value. And most of the time, derivative is much less. Thus, at each layer $w^{(j)} \cdot \sigma'(z^{(j+1)}) \leq 0.25$ in Eq. (1). As a result, products decrease exponentially. $\frac{\partial L}{\partial a_1^{(2)}}$ becomes 16 (or more) times smaller than $\frac{\partial L}{\partial a_1^{(4)}}$. Thus, gradients become very small (vanish), and updates on weights get smaller, and they begin to “learn” very slowly. Detailed explanation of the vanishing gradient problem can be found in [8].

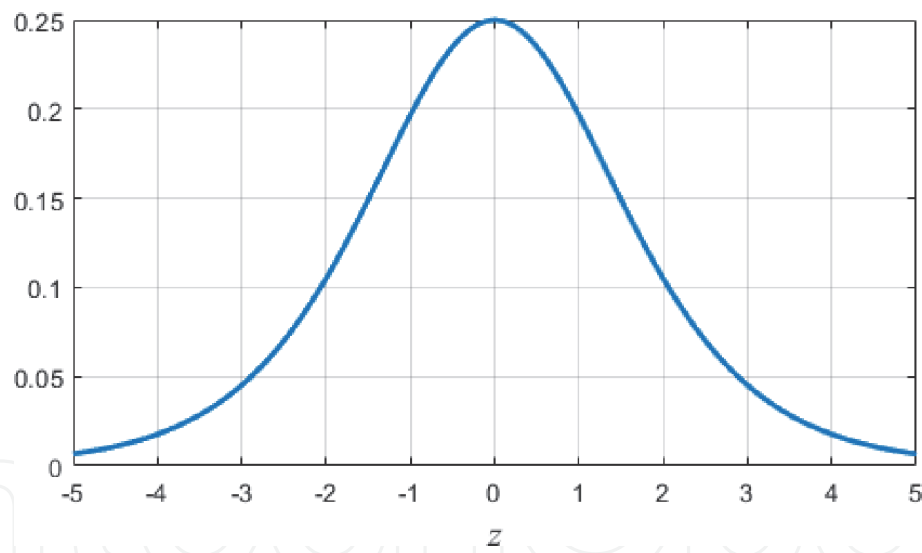


Figure 4.
Derivative of the sigmoid function.

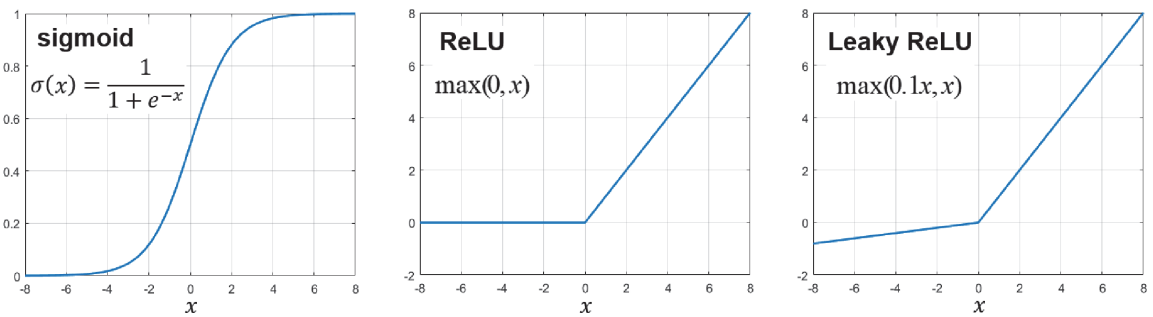


Figure 5.
Plots for some activation functions. Sigmoid is on the left, rectified linear unit is in the middle, and leaky rectified linear unit is on the right.

Today, choices of activation function are different. A rectified linear unit (ReLU), which outputs zero for negative inputs and identical value for positive inputs, is enough to eliminate the vanishing gradient problem. To gain some other advantages, leaky ReLU and parametric ReLU (negative side is multiplied by a coefficient) are among the popular choices (**Figure 5**).

3. Natural language processing

Deep learning transformed the field of natural language processing (NLP). This transformation can be described by better representation learning through newly proposed neural language models and novel neural network architectures that are fine-tuned with respect to an NLP task.

Deep learning paved the way for neural language models, and these models introduced a substantial performance improvement over n-gram language models. More importantly, neural language models are able to learn good representations in their hidden layers. These representations are shown to capture both semantic and syntactic regularities that are useful for various downstream tasks.

3.1 Representation learning

Representation learning through neural networks is based on the distributional hypothesis: “words with similar distributions have similar meanings” [9] where distribution means the neighborhood of a word, which is specified as a fixed-size surrounding window. Thus, the neighborhoods of words are fed into the neural network to learn representations implicitly.

Learned representations in hidden layers are termed as distributed representations [10]. Distributed representations are local in the sense that the set of activations to represent a concept is due to a subset of dimensions. For instance, cat and dog are hairy and animate. The set of activations to represent “being hairy” belongs to a specific subset of dimensions. In a similar way, a different subset of dimensions is responsible for the feature of “being animate.” In the embeddings of both cat and dog, the local pattern of activations for “being hairy” and “being animate” is observed. In other words, the pattern of activations is local, and the conceptualization is global (e.g., cat and dog).

The idea of distributed representation was realized by [11] and other studies relied on it. Bengio et al. [11] proposed a neural language model that is based on a feed-forward neural network with a single hidden layer and optional direct connections between input and output layers.

The first breakthrough in representation learning was word2vec [12]. The authors removed the nonlinearity in the hidden layer in the proposed model architecture of [11]. This model update brought about a substantial improvement in computational complexity allowing the training using billions of words. Word2vec has two variants: continuous bag-of-words (CBOW) and Skip-gram.

In CBOW, a middle word is predicted given its context, the set of neighboring left and right words. When the input sentence “creativity is intelligence having fun” is processed, the system predicts the middle word “intelligence” given the left and right contexts (**Figure 6**). Every input word is in one-hot encoding where there is a vocabulary size (V) vector of all zeros except the one in that word’s index. In the single hidden layer, instead of applying a nonlinear transformation, the average of the neighboring left and right vectors (w_c) is computed to represent the context. As the order of words is not taken into consideration by averaging, it is named as a bag-of-words model. Then the middle word’s (w_t) probability given the context

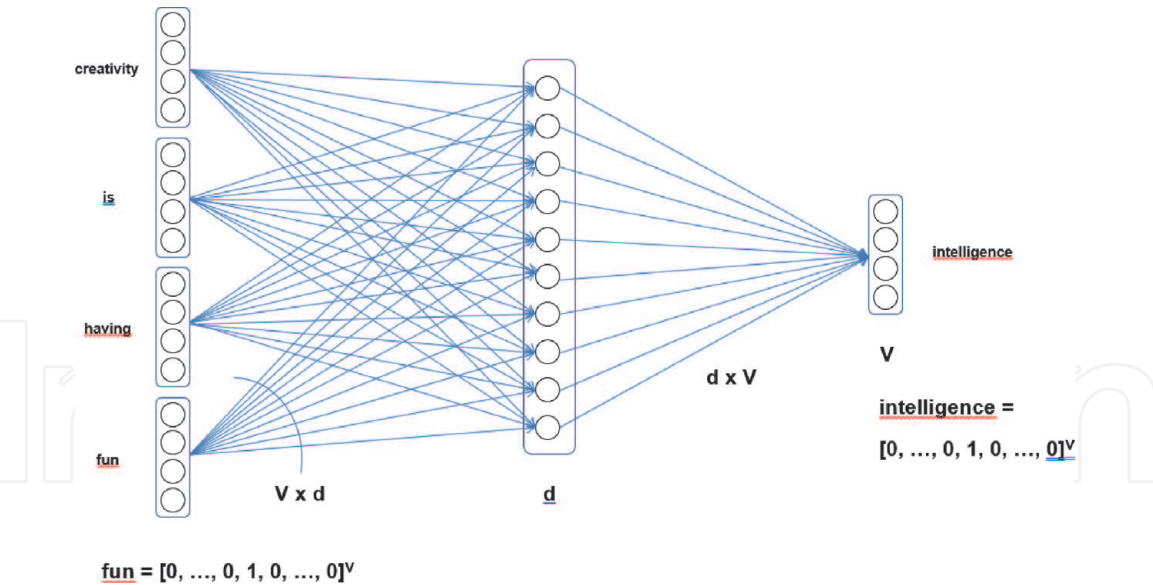


Figure 6.
CBOW architecture.

$(p(w_t|w_c))$ is calculated through softmax on context-middle word dot product vector (Eq. (2)). Finally, the output loss is calculated based on the cross-entropy loss between the system predicted output and the ground-truth middle word.

$$p(w_t|w_c) = \frac{\exp(w_c \cdot w_t)}{\sum_{j \in V} \exp(w_j \cdot w_t)} \quad (2)$$

In Skip-gram, the system predicts the most probable context words for a given input word. In terms of a language model, while CBOW predicts an individual word's probability, Skip-gram outputs the probabilities of a set of words, defined by a given context size. Due to high dimensionality in the output layer (all vocabulary words have to be considered), Skip-gram has higher computational complexity than CBOW (Figure 7). To deal with this issue, rather than traversing all vocabulary in the output layer, Skip-gram with negative sampling (SGNS) [13] formulates the problem as a binary classification where one class represents the current context's occurrence probability, whereas the other is all vocabulary terms' occurrence in the present context. In the latter probability calculation, a sampling approach is incorporated. As vocabulary terms are not distributed uniformly in contexts, sampling is performed from a distribution where the order of the frequency of vocabulary words in corpora is taken into consideration. SGNS incorporates this sampling idea by replacing the Skip-gram's objective function. The new objective function (Eq. (3)) depends on maximizing $P(D = 1|w, c)$, where w, c is the word-context pair. This probability denotes the probability of (w, c) coming from the corpus data. Additionally, $P(D = 0|u_i, c)$ should be maximized if (u_i, c) pair is not included in the corpus data. In this condition, (u_i, c) pair is sampled, as the name suggests negative sampled k times.

$$\sum_{w,c} \left(\log \sigma(\vec{w} \cdot \vec{c}) \right) + \sum_{i=1}^k \left(\log \sigma(-\vec{u}_i \cdot \vec{c}) \right) \quad (3)$$

Both word2vec variants produced word embeddings that can capture multiple degrees of similarity including both syntactic and semantic regularities.

A regular extension to word2vec model was doc2vec [14], where the main goal is to create a representation for different document levels, e.g., sentence and

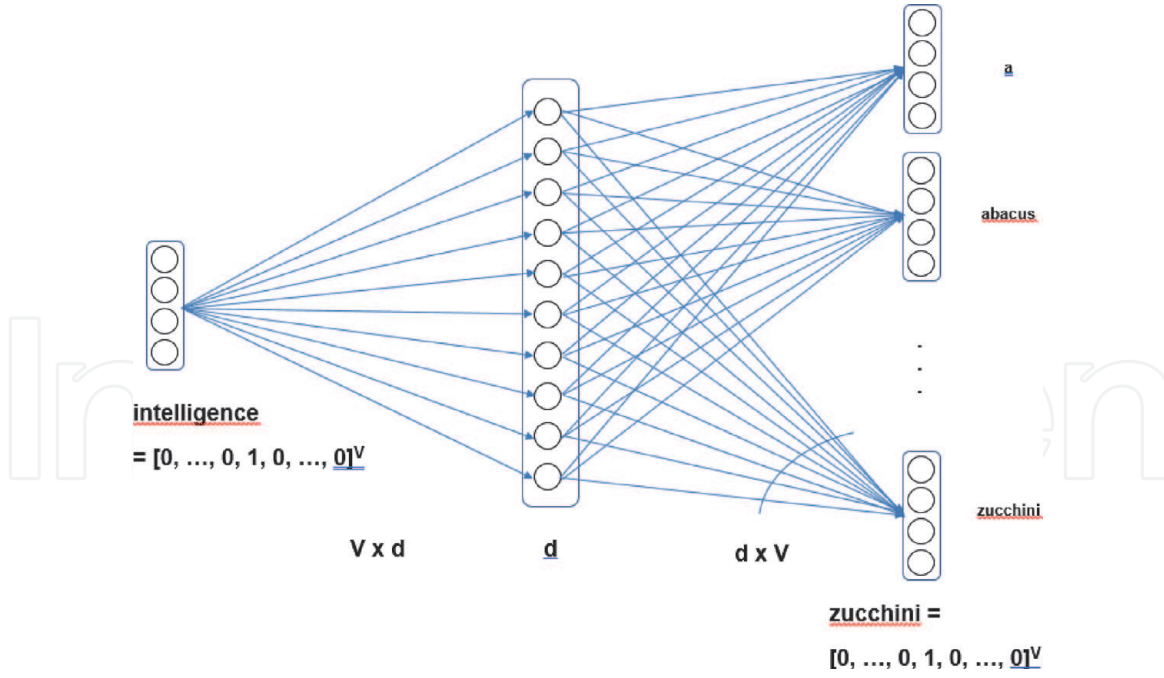


Figure 7.
Skip-gram architecture.

paragraph. Their architecture is quite similar to the word2vec except for the extension with a document vector. They generate a vector for each document and word. The system takes the document vector and its words' vectors as an input. Thus, the document vectors are adjusted with regard to all the words in this document. At the end, the system provides both document and word vectors. They propose two architectures that are known as distributed memory model of paragraph vectors (DM) and distributed bag-of-words model of paragraph vectors (DBOW).

DM: In this architecture, inputs are the words in a context except for the last word and document, and the output is the last word of the context. The word vectors and document vector are concatenated while they are fed into the system.

DBOW: The input of the architecture is a document vector. The model predicts the words randomly sampled from the document.

An important extension to word2vec and its variants is fastText [15], where they considered to use characters together with words to learn better representations for words. In fastText language model, the score between a context word and the middle word is computed based on all character n -grams of the word as well as the word itself. Here n -grams are contiguous sequences of n letters like unigram for a single letter, bigram for two consecutive letters, trigram for three letters in succession, etc. In Eq. (4), v_c represents a context vector, z_g is a vector associated with each n -gram, and G_w is the set of all character n -grams of the word w together with itself.

$$s(w, c) = \sum_{g \in G_w} (z_g^T v_c) \quad (4)$$

The idea of using the smallest syntactic units in the representation of words introduced an improvement in morphologically rich languages and is capable to compute a representation for out-of-vocabulary words.

The recent development in representation learning is the introduction of contextual representations. Early word embeddings have some problems. Although they can learn syntactic and semantic regularities, they are not so good in capturing a mixture of them. For example, they can capture the syntactic pattern *look-looks-looked*.

In a similar way, the words *hard*, *difficult*, and *tough* are embedded into closer points in the space. To address both syntactic and semantic features, Kim et al. [16] used a mixture of character- and word-level features. In their model, at the lowest level of hierarchy, character-level features are processed by a CNN; after transferring these features over a highway network, high-level features are learned by the use of a long short-term memory (LSTM). Thus, the resulting embeddings showed good syntactic and semantic patterns. For instance, the closest words to the word *richard* are returned as *eduard*, *gerard*, *edward*, and *carl*, where all of them are person names and have syntactic similarity to the query word. Due to character-aware processing, their models are able to produce good representations for out-of-vocabulary words.

The idea of capturing syntactic features at a low level of hierarchy and the semantic ones at higher levels was realized ultimately by the Embeddings from Language Models (ELMo) [17]. ELMo proposes a deep bidirectional language model to learn complex features. Once these features are learned, the pre-trained model is used as an external knowledge source to the fine-tuned model that is trained using task-specific data. Thus, in addition to static embeddings from the pre-trained model, contextual embeddings can be taken from the fine-tuned one.

Another drawback of previous word embeddings is they unite all the senses of a word into one representation. Thus, different contextual meanings cannot be addressed. The brand new ELMo and Bidirectional Encoder Representations from Transformers (BERT) [18] models resolve this issue by providing different representations for every occurrence of a word. BERT uses bidirectional Transformer language model integrated with a masked language model to provide a fine-tuned language model that is able to provide different representations with respect to different contexts.

3.2 NLP with neural network solutions

In NLP, different neural network solutions have been used in various downstream tasks.

Language data are temporal in nature so recurrent neural networks (RNNs) seem as a good fit to the task in general. RNNs have been used to learn long-range dependencies. However, because of the dependency to the previous time steps in computations, they have efficiency problems. Furthermore, when the length of sequences gets longer, an information loss occurs due to the vanishing gradient problem.

Long short-term memory architectures are proposed to tackle the problem of information loss in the case of long sequences. Gated recurrent units (GRUs) are another alternative to LSTMs. They use a gate mechanism to learn how much of the past information to preserve at the next time step and how much to erase.

Convolutional neural networks have been used to capture short-ranging dependencies like learning word representation over characters and sentence representation over its n-grams. Compared to RNNs, they are quite efficient due to independent processing of features. Moreover, through the use of different convolution filter sizes (overlapping localities) and then concatenation, their learning regions can be extended.

Machine translation is a core NLP task that has witnessed innovative neural network solutions that gained wide application afterwards. Neural machine translation aims to translate sequences from a source language into a target language using neural network architectures. Theoretically, it is a conditional language model where the next word is dependent on the previous set of words in the target sequence and the source sentence at the same time. In traditional language

modeling, the next word's probability is computed based solely on the previous set of words. Thus, in conditional language modeling, conditional means conditioned on the source sequence's representation. In machine translation, source sequence's processing is termed as encoder part of the model, whereas the next word prediction task in the target language is called decoder. In probabilistic terms, machine translation aims to maximize the probability of the target sequence y given the source sequence x as follows.

$$\arg \max_y P(y|x) \quad (5)$$

This conditional probability calculation can be conducted by the product of component conditional probabilities at each time step where there is an assumption that the probabilities at each time step are independent from each other (Eq. (6)).

$$\begin{aligned} P(y|x) &= P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x), \dots, P(y_t|y_1, \dots, y_{t-1},x) \\ &= \prod_{i=1}^t P(y_i|y_1, \dots, y_{i-1},x) \end{aligned} \quad (6)$$

The first breakthrough neural machine translation model was an LSTM-based encoder-decoder solution [19]. In this model, source sentence is represented by the last hidden layer of encoder LSTM. In the decoder part, the next word prediction is based on both the encoder's source representation and the previous set of words in the target sequence. The model introduced a significant performance boost at the time of its release.

In neural machine translation, the problem of maximizing the probability of a target sequence given the source sequence can be broken down into two components by applying Bayes rule on Eq. (5): the probability of a source sequence given the target and the target sequence's probability (Eq. (7)).

$$\arg \max_y P(x|y)P(y) \quad (7)$$

In this alternative formulation, $P(x|y)$ is termed as translation model and $P(y)$ is a language model. Translation model aims to learn correspondences between source and target pairs using parallel training corpus. This learning objective is related to the task of learning word-level correspondences between sentence pairs. This alignment task is vital in that a correct translation requires to generate the counterpart word(s) for the local set of words in the source sentence. For instance, the French word group "tremblement de terre" must be translated into English as the word "earthquake," and these correspondences must be learned in the process.

Bandanau et al. [20] propose an attention mechanism to directly connect to each word in the encoder part in predicting the next word in each decoder step. This mechanism provides a solution to alignment in that every word in translation is predicted by considering all words in the source sentence, and the predicted word's correspondences are learned by the weights in the attention layer (**Figure 8**).

Attention is a weighted sum of values with respect to a query. The learned weights serve as the degree of query's interaction with the values at hand. In the case of translation, values are encoder hidden states, and query is decoder hidden state at the current time step. Thus, weights are expected to show each translation step's grounding on the encoder hidden states.

Eq. (8) gives the formulae for an attention mechanism. Here h_i represents each hidden state in the encoder (VALUES in **Figure 8**), w_i is the query vector coming

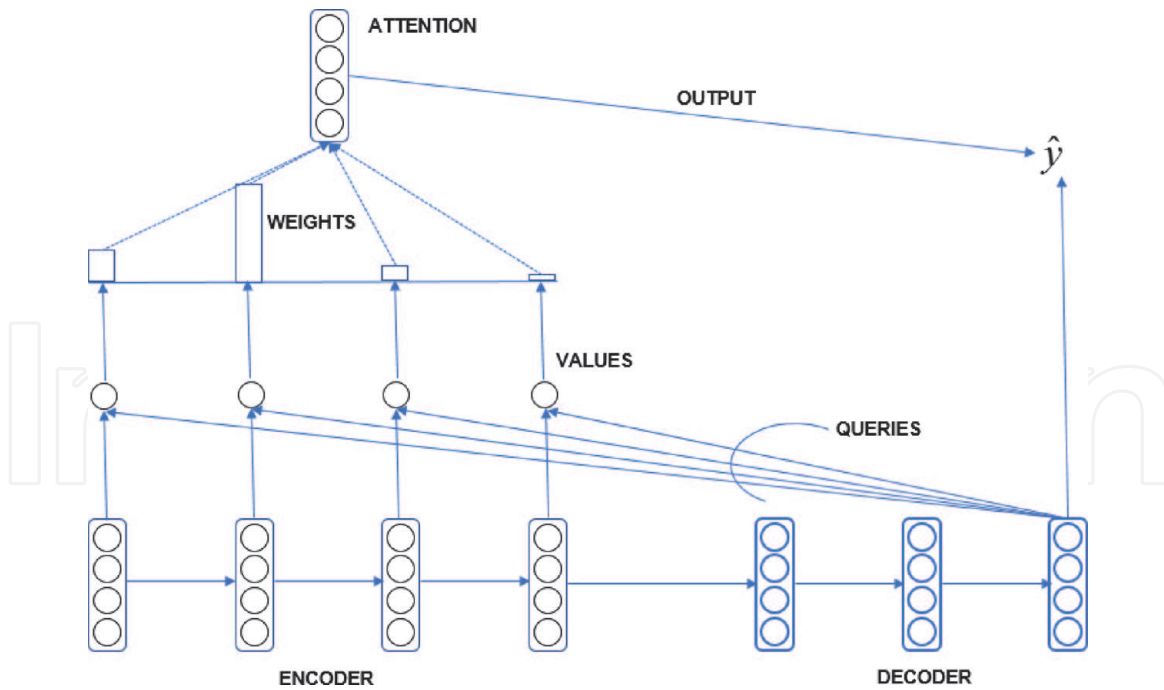


Figure 8.
 Sequence-to-sequence attention.

from the current hidden state of the decoder (each QUERY in **Figure 8**), α_i (WEIGHTS in **Figure 8**) are attention weights, and K (OUTPUT in **Figure 8**) is the attention output that is combined with the last hidden state of the decoder to make the next word prediction in translation.

$$\begin{aligned}\alpha_i &= \frac{\exp(h_i \cdot w_i)}{\sum_j \exp(h_j \cdot w_j)} \\ o_i &= \alpha_i h_i \\ K &= \sum_i o_i\end{aligned}\tag{8}$$

The success of attention in addressing alignment in machine translation gave rise to the idea of a sole attention-based architecture called Transformer [21]. The Transformer architecture produced even better results in neural machine translation. More importantly, it has become state-of-the-art solution in language modeling and started to be used as a pre-trained language model. The use of it as a pre-trained language model and the transfer of this model's knowledge to other models introduced performance boost in a wide variety of NLP tasks.

The contribution of attention is not limited to the performance boost introduced but is also related to supporting explainability in deep learning. The visualization of attention provides a clue to the implicit features learned for the task at hand.

4. Computer vision and CNNs

To observe the performance of the developed methods on computer vision problems, several competitions are arranged all around the world. One of them is Large Scale Visual Recognition Challenge [22]. This event contains several tasks which are image classification, object detection, and object localization. In image classification task, the aim is to predict the class of images in the test set given a set of discrete labels, such as dog, cat, truck, plane, etc. This is not a trivial task since

different images of the same class have quite different instances and varying view-points, illumination, deformation, occlusion, etc.

All competitors in ILSVRC train their model on ImageNet [22] dataset. ImageNet 2012 dataset contains 1.2 million images and 1000 classes. Classification performances of proposed methods were compared according to two different evaluation criteria which are top 1 and top 5 score. In top 5 criterion, for each image top 5 guesses of the algorithm are considered. If actual image category is one of these five labels, then the image is counted as correctly classified. Total number of incorrect answers in this sense is called top 5 error.

An outstanding performance was observed by a CNN (convolutional neural network) in 2012. AlexNet [23] got the first place in classification task achieving 16.4% error rate. There was a huge difference between the first (16.4%) and second place (26.1%). In ILSVRC 2014, GoogleNet [24] took the first place achieving 6.67% error rate. Positive effect of network depth was observed. One year later, ResNet took the first place achieving 3.6% error rate [25] with a CNN of 152 layers. In the following years, even lower error rates were achieved with several modifications. Please note that the human performance on the image classification task was reported to be 5.1% error [22].

4.1 Architecture of a typical CNN

CNNs are the fundamental structures while working on images and videos. A typical CNN is actually composed of several layers interleaved with each other.

4.1.1 Convolutional layer

Convolutional layer is the core building block of a CNN. It contains plenty of learnable filters (or kernels). Each filter is convolved across width and height of input images. At the end of training process, filters of network are able to identify specific types of appearances (or patterns). A mathematical example is given to illustrate how convolutional layers work (**Figure 9**). In this example, a 5×5 RGB image is given to the network. Since images are represented as 3D arrays of numbers, input consists of three matrices. It is convolved with a filter of size $3 \times 3 \times 3$ (height, weight, and depth). In this example, convolution is applied by moving the filter one pixel at a time, i.e., stride size = 1. First convolution operation can be seen at **Figure 9a**. After moving the kernel one pixel to the right, second convolution operation can be seen at **Figure 9b**. Element-wise multiplication \odot is applied in each convolution phase. Thus the operation in **Figure 9a** is shown below Eq. (9).

$$\begin{aligned}
 & \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 2 \\ 2 & 1 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 1 \\ 1 & 0 & 0 \\ 2 & 2 & 0 \end{bmatrix} \odot \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & -1 & -1 \end{bmatrix} \\
 & + \begin{bmatrix} 2 & 2 & 0 \\ 0 & 1 & 2 \\ 0 & 1 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & -1 & -1 \end{bmatrix} + 1 = 8
 \end{aligned} \tag{9}$$

Convolution depicted in **Figure 9** is performed with one filter which results in one matrix (called activation map) in the convolution layer. Using n filters for the input in **Figure 9** produces a convolution layer of depth n (**Figure 10**).

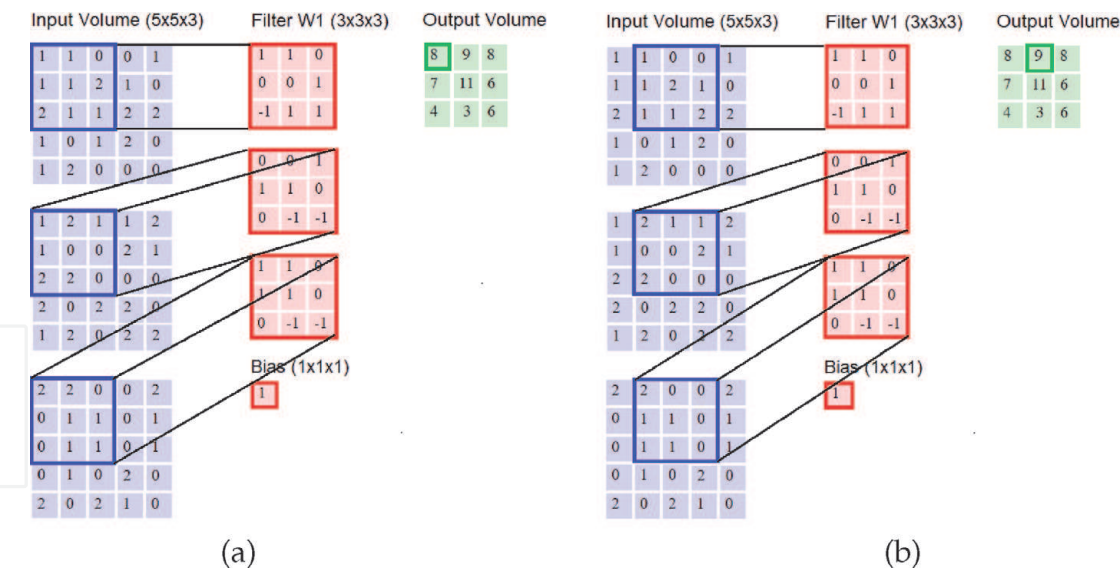


Figure 9. Convolution process. (a) First convolution operation applied with filter W1. Computation gives us the top-left member of an activation map in the next layer. (b) Second convolution operation, again applied with filter W1.

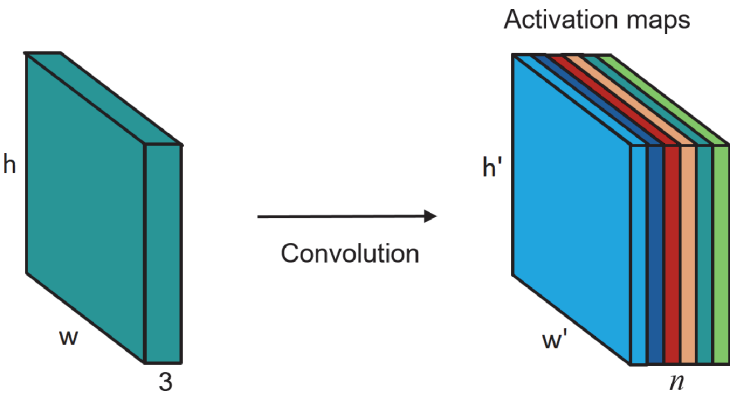


Figure 10. Formation of a convolution layer by applying n number of learnable filters on the previous layer. Each activation map is formed by convolving a different filter on the whole input. In this example input to the convolution is the RGB image itself (depth = 3). For every further layer, input is its previous layer. After convolution, width and height of the next layer may or may not decrease.

4.1.2 Pooling layer

Pooling layer is commonly used between convolutional layers to reduce the number of parameters in the upcoming layers. It makes the representations smaller and the algorithm much faster. With max pooling, filter takes the largest number in the region covered by the matrix on which it is applied. Example input, on which 2×2 max pooling is applied, is shown in **Figure 11**. If the input size is $w \times h \times n$, then the output size is $(w/2) \times (h/2) \times n$. Techniques such as min pooling and average pooling can also be used.

Standard CNNs generally have several convolution layers, followed by pooling layers and at the end a few fully connected layers (**Figure 12**). CNNs are similar to standard neural networks, but instead of connecting weights to all units of the previous layer, a convolution operation is applied on the units (voxels) of the previous layer. It enables us scale weights in an efficient way since a filter has a fixed number of weights and it is independent of the number of the voxels in the previous layer.

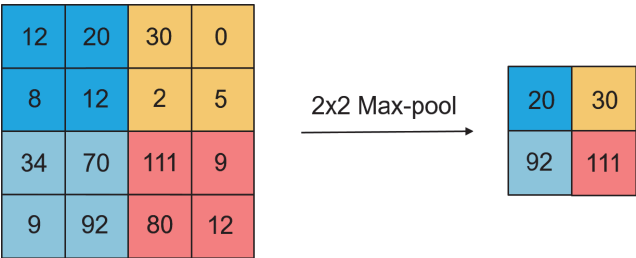


Figure 11.
Max pooling.

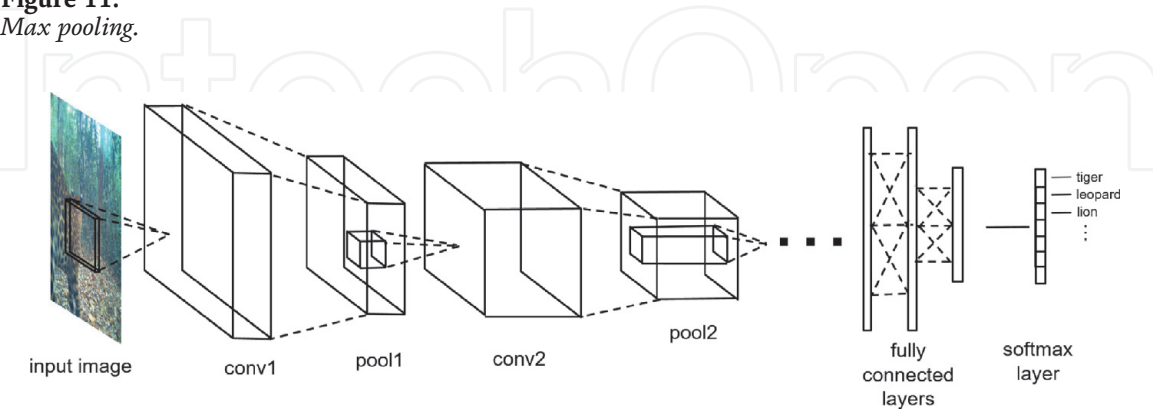


Figure 12.
A typical CNN for image classification task.

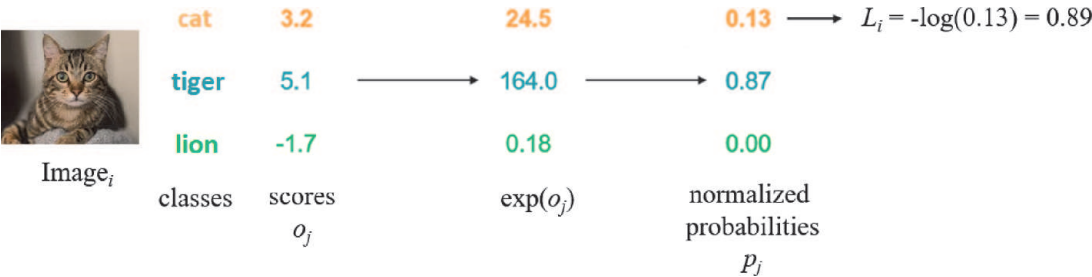


Figure 13.
An example of softmax classification loss calculation. Computed loss, L_i , is only for the i th sample in the dataset.

4.1.3 Classification layer

What we have in the last fully connected layer of a classification network is the output scores for each class. It may seem trivial to select the class with the highest score to make a decision; however we need to define a loss to be able to train the network. Loss is defined according to the scores obtained for the classes. A common practice is to use softmax function, which first converts the class scores into normalized probabilities (Eq. (10)):

$$p_j = \frac{e^{o_j}}{\sum_k e^{o_k}} \tag{10}$$

where k is the number of classes, o_j are the output neurons (scores), and p_j are the normalized probabilities. Softmax loss is equal to the log of the normalized probability of the correct class. An example calculation of softmax loss with three classes is given in **Figure 13**.

4.2 Generalization capability of CNNs

The ability of a model to make correct predictions for new samples after trained on the training set is defined as generalization. Thus, we would like to train a CNN

with a high generalization capacity. Its high accuracy should not be only for training samples. In general, we should increase the size and variety of the training data, and we should avoid training an excessively complex model (simply called overfitting). Since it is not always easy to obtain more training data and to pick the best complexity for our model, let's discuss a few popular techniques to increase the generalization capacity.

4.2.1 Regularization loss

This is a term, $R(W)$, added to the data loss with a coefficient (λ) called regularization strength (Eq. (10)). Regularization loss can be a sum of L1 or L2 norm of weights. The interpretation of $R(W)$ is that we want smaller weights to be able to achieve smoother models for better generalization. It means that no input dimension can have a very large influence on the scores all by itself.

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda \cdot R(W) \quad (11)$$

4.2.2 Dropout

Another way to prevent overfitting is a technique called dropout, which corresponds to removing some units in the network [26]. The neurons which are “dropped out” in this way do not contribute to the forward pass (computation of loss for a given input) and do not participate in backpropagation (**Figure 14**). In each forward pass, a random set of neurons are dropped (with a hyperparameter of dropping probability, usually 0.5).

4.2.3 Data augmentation

The more training samples for a model, the more successful the model will be. However, it is rarely possible to obtain large-size datasets either because it is hard to collect more samples or it is expensive to annotate large number of samples. Therefore, to increase the size of existing raw data, producing synthetic data is sometimes preferred. For visual data, data size can be increased by rotating the picture at different angles, random translations, rotations, crops, flips, or altering brightness and contrast [27].

4.3 Transfer learning

Short after people realized that CNNs are very powerful nonlinear models for computer vision problems, they started to seek an insight of why these models

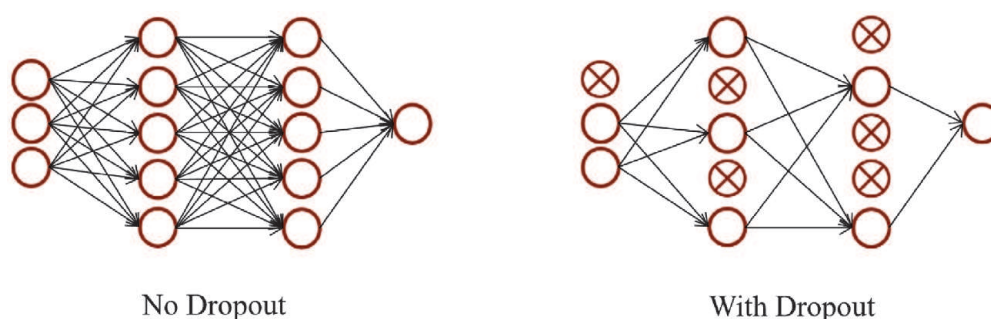


Figure 14.
 Applying dropout in a neural net.

perform so well. To this aim, researchers proposed visualization techniques that provide an understanding of what features are learned in different layers of a CNN [28]. It turns out that first convolutional layers are responsible for learning low-level features (edges, lines, etc.), whereas as we go further in the convolutional layers, specific shapes and even distinctive patterns can be learned (**Figure 15**).

In early days of observing the great performance of CNNs, it was believed that one needs a very large dataset in order to use CNNs. Later, it was discovered that, since the pre-trained models already learned to distinguish some patterns, they provide great benefits for new problems and new datasets from varying domains. Transfer learning is the name of training a new model with transferring weights from a related model that had already been trained.

If the dataset in our new task is small but similar to the one that was used in pre-trained model, then it would work to change the classification layer (according to our classes) and train this last layer. However, if our dataset is also big enough, we can include a few more layers (starting from the fully connected layers at the end) to our retraining scheme, which is also called fine-tuning. For instance, if a face recognition model trained with a large database is available and you would like to use that model with the faces in your company, that would constitute an ideal case of transferring the weights from the pre-trained model and fine-tune one or two layers with your local database. On the other hand, if the dataset in our new task is not similar to the one used in pre-trained model, then we would need a larger dataset and need to retrain a larger number of layers. An example of this case is learning to classify CT (computer tomography) images using a CNN pre-trained on ImageNet dataset. In this situation, the complex patterns (cf. **Figure 15c** and **d**) that were learned within the pre-trained model are not much useful for your new task. If both the new dataset is small and images are much different from those of a trained model, then users should not expect any benefit from transferring weights. In such cases users should find a way to enlarge the dataset and train a CNN from scratch using the newly collected training data. The cases that a practitioner may encounter from the transfer learning point of view are summarized in **Table 1**.

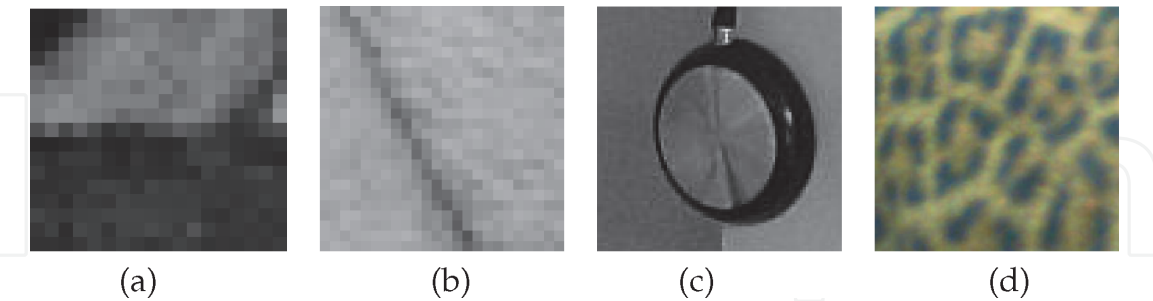


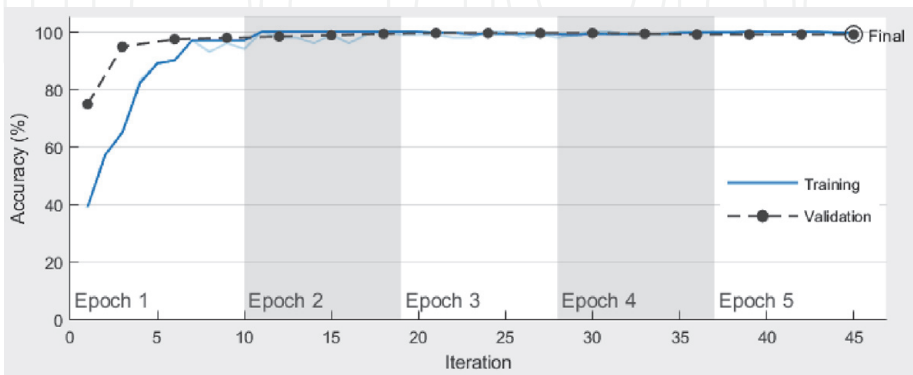
Figure 15. Image patches corresponding to the highest activations in a random subset of feature maps. First layer's high activations occur at patches of distinct low-level features such as edges (a) and lines (b); further layers' neurons learn to fire at more complex structures such as geometric shapes (c) or patterns on an animal (d). Since activations in the first layer correspond to small areas on images, resolution of patches in (a) and (b) is low.

	Very similar dataset	Very different dataset
Very little data	Replace the classification layer	Not recommended
A lot of data	Fine-tune a few layers	Fine-tune a larger number of layers

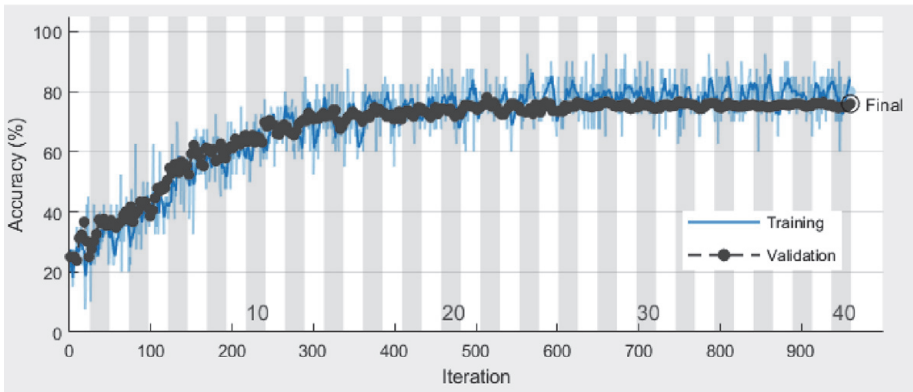
Table 1. Strategies of transfer learning according to the size of the new dataset and its similarity to the one used in pre-trained model.



Figure 16.
Example images for each class used in the experiment of transfer learning for animal classification.



(a)



(b)

Figure 17.
Training and validation set accuracies obtained (a) with transfer learning and (b) without transfer learning.

To emphasize the importance of transfer learning, let us present a small experiment where the same model is trained with and without transfer learning. Our task is the classification of animals (four classes) from their images. Classes are zebra, leopard, elephant, and bear where each class has 350 images collected from the Internet (**Figure 16**). Transfer learning is performed using an AlexNet [23] pre-trained on ImageNet dataset. We have replaced the classification layer with a four-neuron layer (one for each class) which was originally 1000 (number of classes in ImageNet). In training conducted with transfer learning, we reached a 98.81% accuracy on the validation set after five epochs (means after seeing the dataset five times during training). Readers can observe that accuracy is quite satisfactory even after one epoch (**Figure 17a**). On the other hand, in training without transfer learning, we could reach only 76.90% accuracy even after 40 epochs (**Figure 17b**). Trying different hyperparameters (regularization strength, learning rate, etc.) could have a chance to increase accuracy a little bit more, but this does not alleviate the importance of applying transfer learning.

5. Conclusions

Deep learning has become the dominant machine learning approach due to the availability of vast amounts of data and improved computational resources. The main transformation was observed in text and image analysis.

In NLP, change can be described in two major lines. The first line is learning better representations through ever-improving neural language models. Currently, self-attention-based Transformer language model is state-of-the-art, and learned representations are capable to capture a mix of syntactic and semantic features and are context-dependent. The second line is related to neural network solutions in different NLP tasks. Although LSTMs proved useful in capturing long-term dependencies in the nature of temporal data, the recent trend has been to transfer the pre-trained language models' knowledge into fine-tuned task-specific models. Self-attention neural network mechanism has become the dominant scheme in pre-trained language models. This transfer learning solution outperformed existing approaches in a significant way.

In the field of computer vision, CNNs are the best performing solutions. There are very deep CNN architectures that are fine-tuned, thanks to huge amounts of training data. The use of pre-trained models in different vision tasks is a common methodology as well.

One common disadvantage of deep learning solutions is the lack of insights due to learning implicitly. Thus, attention mechanism together with visualization seems promising in both NLP and vision tasks. The fields are in the quest of more explainable solutions.

One final remark is on the rise of multimodal solutions. Till now question answering has been an intersection point. Future work are expected to be devoted to multimodal solutions.

Author details


Selma Tekir*[†] and Yalin Bastanlar[†]

Computer Engineering Department, Izmir Institute of Technology, Izmir, Turkey

*Address all correspondence to: selmatekir@iyte.edu.tr

[†] These authors are contributed equally.

IntechOpen

© 2020 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

- [1] McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*. 1943;5: 115-133
- [2] Rosenblatt F. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory; 1957
- [3] *Images from the Rare Book and Manuscript Collections*. Cornell University Library. Available from: <https://digital.library.cornell.edu/catalog/ss:550351>
- [4] Minsky M, Papert S. *Perceptrons. An Introduction to Computational Geometry*. Cambridge, MA: MIT Press; 1969
- [5] Werbos P. *Beyond regression: New tools for prediction and analysis in the behavioral sciences [PhD thesis]*. Cambridge, MA: Harvard University; 1974
- [6] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. *Nature*. 1986;323:533-536
- [7] LeCun Y, Jackel LD, Boser B, Denker JS, Graf HP, Guyon I, et al. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*. 1989; 27(11):41-46
- [8] Nielsen M. *Neural Network and Deep Learning*. Available from: <http://neuralnetworksanddeeplearning.com/chap5.html> [Accessed: 30 December 2019]
- [9] Harris Z. Distributional structure. *Word*. 1954;10(23):146-162
- [10] Hinton GE, McClelland JL, Rumelhart DE. Distributed representations. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1. Cambridge, MA: MIT Press; 1986. pp. 77-109
- [11] Bengio Y, Ducharme R, Vincent P, Janvin C. A neural probabilistic language model. *Journal of Machine Learning Research*. 2003;3:1137-1155
- [12] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. In: *Workshop Proceedings of ICLR*. 2013
- [13] Mikolov T, Sutskever I, Chen K, Corrado G, Dean J. Distributed representations of words and phrases and their compositionality. *CoRR*. abs/1310.4546. Available from: <http://arxiv.org/abs/1310.4546>
- [14] Le Q, Mikolov T. Distributed representations of sentences and documents. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML)*. 2014
- [15] Bojanowski P, Grave E, Joulin A, Mikolov T. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics (TACL)*. 2016;5:135-146
- [16] Kim Y, Jernite Y, Sontag D, Rush AM. Character-aware neural language models. In: *Proceedings of Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*. 2016
- [17] Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, et al. Deep contextualized word representations. In: *Proceedings of NAACL*. 2018
- [18] Devlin J, Chang M, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for

- language understanding. In: Proceedings of NAACL. 2019
- [19] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Proceedings of Advances in Neural Information Processing Systems (NIPS). 2014
- [20] Bandanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. In: Proceedings of 3rd International Conference on Learning Representations (ICLR). 2015
- [21] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Proceedings of Advances in Neural Information Processing Systems (NIPS). 2017
- [22] Russakovsky O et al. ImageNet large scale visual recognition challenge. International Journal of Computer Vision (IJCV). 2015;115(3):211-252
- [23] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Proceedings of NIPS. 2012
- [24] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In: Proceedings of CVPR. 2015
- [25] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of CVPR. 2016
- [26] Srivastava N et al. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine learning Research. 2014;15(1):1929-1958
- [27] Goodfellow I, Bengio Y, Courville A. Deep Learning. Cambridge, MA: MIT Press; 2016
- [28] Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. In: Proceedings of ECCV. 2014