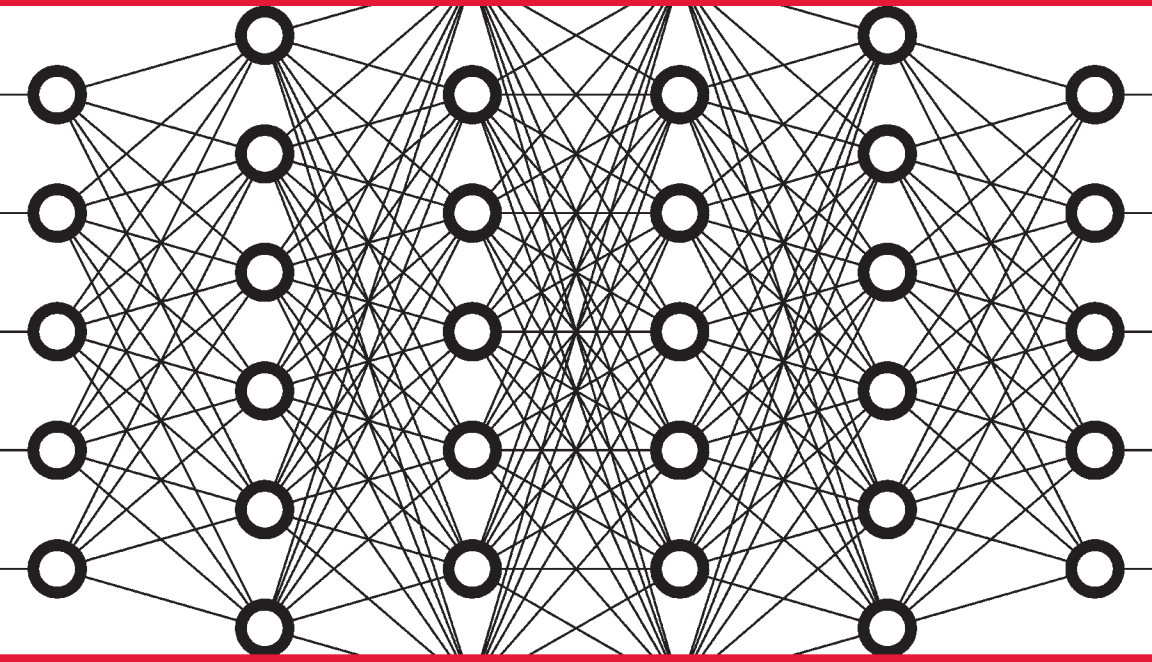
O'REILLY®

Compliments of IBM

Applying Deep Learning in Business

How to Start Developing Solutions for Your Organization



Federico Castanedo

Applying Deep Learning in Business

How to Start Developing Solutions for Your Organization

Federico Castanedo

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Applying Deep Learning in Business
by Federico Castanedo

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Nicole Tache and Michele Cronin
Production Editor: Nicholas Adams
Copyeditor: Octal Publishing, LLC

Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

September 2018: First Edition

Revision History for the First Edition
2018-09-19: First Release

978-1-492-03918-1

[LSI]

Table of Contents

Applying Deep Learning in Business

Introduction

Machine learning can be defined as a method to achieve artificial intelligence (AI) through algorithms that can find patterns in a set of data. *Deep learning* is a subset of machine learning and can be generally defined as a *technique for implementing machine learning algorithms, inspired by the structure and function of the brain called artificial neural networks*. In this context, deep learning is the coolest artificial intelligence technique nowadays, and it deserves all the attention it is getting. In recent years, deep learning achieved superhuman results in areas like computer vision, autonomous cars, chess, and the game of Go. Based on these results, business executives are increasingly requesting enterprise developers to apply deep learning technologies to improve their key business objectives. Just a few software companies have the necessary expertise to develop AI tools that simplify the arduous process of developing deep learning applications. With the open sourced release of TensorFlow by Google in November 2015, the development of deep learning models in the enterprise has exploded. Google is also providing a set of APIs in its cloud based on deep learning models that are already trained for image, video, and speech analysis. On the other hand, Amazon supports a wide range of deep learning frameworks within its cloud offering, including TensorFlow. Finally, IBM also released a set of cloud-based tools to help train and deploy deep learning models in the enterprise.

We're now seeing deep learning used to assist humans with automation of different tasks and to help them to make better data-driven decisions. In general, benefits of applying deep learning in business can fall into these categories:

- Reduce costs by reducing the time required to complete a given task.
- Create data-driven decisions, to increase revenue.

However, a couple of things are not always clear:

- How can deep learning provide incremental benefits, and how does this differ from traditional techniques?
- How can an enterprise software developer exploit deep learning techniques for their unique business objectives?

This report aims to clarify both of these points and introduce a set of IBM tools for deep learning that increase the productivity of data scientists and software developers as well as the quality and maintainability of deep learning models.

In this report, we provide an introduction to deep learning, demonstrate how you can use deep learning networks in the enterprise today, and enumerate the pros and cons of existing alternatives to developing and deploying deep learning solutions in-house. We describe an example of using *Deep Learning as a Service* within IBM Watson to overcome the common barriers of deep learning deployment: skills, standardization, and complexity. We also cover real-world use cases of deep learning in business and finally, speculate on the future of deep learning.

To begin, let's take a quick look at how deep learning has evolved.

A Brief History of Deep Learning

Current deep learning methods are based on the earlier work of neural network models. Neural networks have their roots in the late 1940s with the work of Hebbian Learning from Hebb, the McCulloch and Pitts finite automata, and the development of the Perceptron by Rosenblatt in 1958. Although those earlier works established the roots for deep learning, initial models were difficult to train due to the inexistence of efficient learning algorithms and insufficient

data. The first step to simulate a neural network had been made by Nathaniel Rochester from IBM Research laboratories in the 1950s. The development of *backpropagation algorithm* by Paul Werbos in 1975 accelerated the computation time required for training multilayer networks. And later on, the use of neural networks was restricted to specific problems using small networks trained with small available data. Thanks to the recent advances in learning algorithms, computing power, the availability of powerful Graphics Processing Units (GPUs), and the huge amount of available data—mainly due to the exponential decrease in the cost of data storage—deep learning researchers have begun training deeper and bigger networks with awesome results. Today, we see networks with hundreds of hidden layers that just a few years ago had three to five layers.

A good example of how neural networks have grown is within a Google paper published in 2012, from Quoc V. Le and others, in which they describe how a deep learning network was developed to recognize cats from watching unlabeled images taken from YouTube videos. The network required the computing power of one thousand machines with 16 cores for three days to compute the optimal weights of one billion neural connections.

Advances in computing power, an exponential increase in data, and huge decrease in the price of data storage are not the only reasons we've seen a recent explosion of interest in deep learning. Several improvements over traditional neural network algorithms have also been proposed.

Deep learning is currently still a very active field of research; people are continuously searching for the best topology of networks, optimization models, hyperparameters, and more. You can find more information about the history of deep learning in Andrew L. Beam's blog post, "Deep Learning 101 - Part 1: History and Background".

What Can You Do with Deep Learning in the Enterprise?

Because deep learning is a subset of machine learning, let's first classify the different types of machine learning models based on their goal:

Regression
: These models aim to answer the questions of "How much?" or, "How many?" For instance, "*What will the stock price be?*" "*How many people will buy some product?*" and so on.

Classification
: Here, the goal is to differentiate among different categories, so we need to answer the following question: "*Is it this or that?*" Examples can be to classify whether an email is spam, a person as sick or healthy, or the topic of a specific news article.

Anomaly detection
: The goal with these models is to detect unusual or abnormal behaviors, like credit card fraud.

Clustering
: In these models, the goal is to find some order or patterns in the data. Use cases include: "*Which users like similar movies?*" or "*Let's break these products into five different groups.*"

Reinforcement
: The goal here is to predict the next action based on the feedback of the environment or the user. This type of model will answer questions like, "*Should we raise or lower the temperature?*"

Deep learning methods have been applied to all of these categories and thus can help make predictions that inform and guide tactical, operational, and strategic actions for businesses. These methods have achieved impressive results in different areas of computer science, such as the following:

- Near-human-level speech recognition and image classification.
- Near-human-level handwriting transcription and autonomous driving.
- Improved text-to-speech conversion and machine translation.
- Emergence of digital assistants like Google Now, Siri, Watson Assistant, or Amazon Alexa.

Deep learning shines when dealing with audio, images, or videos as input data or in those cases for which the amount of possible options are exponential, like in the game of Go. That doesn't mean we can't use deep learning to solve traditional machine learning problems or modern business problems. A key draw to using deep

learning models is its ability to improve its performance almost linearly as we increase the amount of available training data, in contrast to traditional machine learning models that were not able to improve the performance when we use more data, as has been pointed out by Andrew Ng (see Figure 1-1). The best way to obtain better model performance nowadays is to use more training data and bigger networks; this requires efficient algorithms and state-of-the-art hardware.

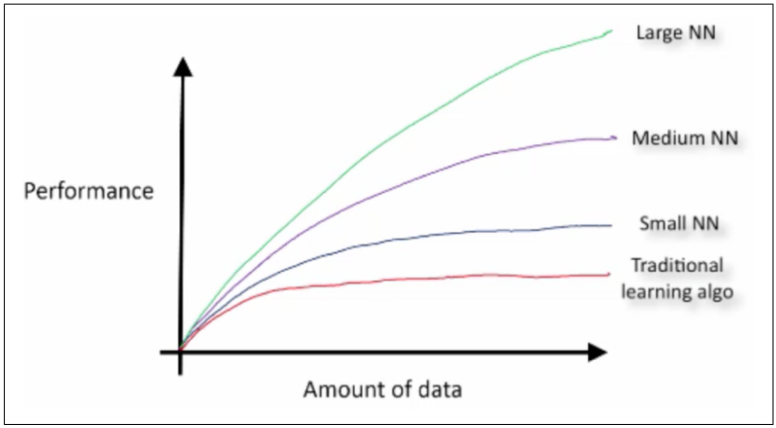


Figure 1-1. Performance of Machine Learning models as a function of the amount of training data. (Source: Andrew Ng, Deeplearning.ai course on Coursera; used with permission)

A common and recurring question is "*When should I use deep learning?*" In general, you should consider deep learning only if you have enough training data; otherwise, traditional machine learning algorithms will provide similar results. Let's try to give some high-level pointers to clarify this answer:

- If you are interested in doing pattern recognition in videos, images, or sounds and you also have a lot of labeled examples (more than thousands, such as the ImageNet challenge), deep learning through convolutional neural networks will be useful.
- If you are interested in finding patterns and similarities in a large number of documents that are stored as text, deep learning can be an option. After those documents are cleaned (without abbreviations and special characters) it will make sense to check out Word2Vec or Recurrent Neural Networks (RNN) techniques.

- If you are dealing with an anomaly detection problem and you have clean and structured data containing those anomalies, it is worthwhile to check out autoencoder networks.
- If your project involves understanding sequences of events or actions such as weather forecasting or stock market prediction, it makes sense to check out RNNs. If those actions involve getting some feedback from the environment, it is interesting to explore deep reinforcement learning techniques.

Deep learning models can learn complex relationships hierarchically because each layer in the neural network extracts new features from the output of the previous layer. Because these models have the ability to "learn" new features that are composed of complex nonlinear relations from previous layers, deep learning models are also known as *feature learning models*. Those nonlinear relations are possible thanks to the activation function of each neuron in the deep network. An activation function ensures that the representation in the input space is mapped to a different space in the output. Essentially each neuron sums all the inputs (x) and their corresponding weights (w) and then apply the activation function $f(x)$ to obtain nonlinearities (see Figure 1-2). For instance, if we use thousands or millions of neurons, input data is transformed by complex nonlinear relations introduced by the activation functions of those neurons. That makes it possible to generate any given output from the specific input, which is the power of neural networks. In fact, neural networks are also known as Universal Approximators because they can approximate any given continuous function with an arbitrary error.[1]

1 For a visual proof that neural networks can approximate any given function, we suggest that you take a look to Michael Neilson's tutorial.

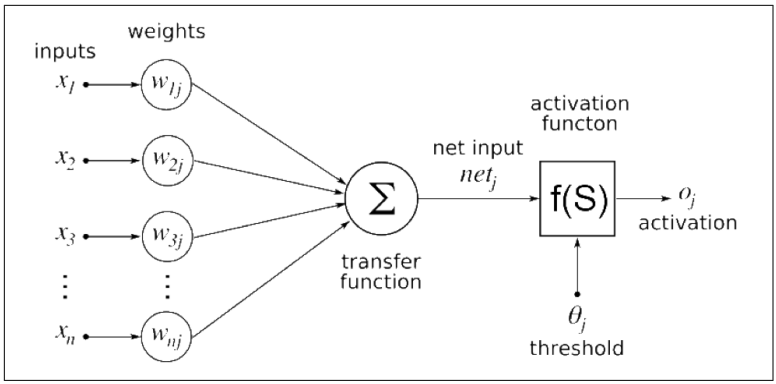


Figure 1-2. Schematic representation of an activation function. (Source: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.png)

In the next two sections, we discuss two methods to apply deep learning in the enterprise: building and training your own deep learning models, and deploying prebuilt models. The first option requires you to handle all aspects of building deep learning models, whereas the second option uses frameworks that simplify and automate most of the common steps in the deep learning process.

Option 1: Build and Train Your Own Deep Learning Models

In traditional software development, the way we write a computer program is by coding explicit rules to follow each step. For instance, if we are writing a program to classify whether incoming emails are spam, we can encode the following rules:

> If email contains Viagra, then mark as spam
>
> If email contains Drugs, then mark as spam

The combination of these rules can be really huge. On the other hand, if we are developing a machine learning model, we write a computer program to learn the rules from the input data, with the following pattern:

> Try to classify some emails

Optimize self to reduce errors

> Repeat

In a nutshell, machine learning is generating a function $f(x)$ that maps a known input x to an output y (see Figure 1-3), but without the requirement of giving a detailed explanation of rules, because these rules will be learned from data.

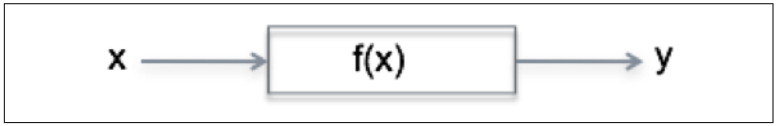


Figure 1-3. An abstraction of machine learning process. (Source: Federico Castanedo)

In the example about detecting spam, we are doing supervised learning because we must have positive and negative examples of what a spam message contains. In particular, we are training a binary classification algorithm based on historical data, and the output of this training will be a model that can be applied to incoming message data. As you can imagine, with traditional machine learning it is very important to capture the relevant features or predictors that allow the model to differentiate among the different classes (spam or not in this example). So, a field named *feature engineering* has emerged as one the most important steps in machine learning model development. Fortunately, one of the strengths of deep learning is the ability to perform feature learning by combining the outputs of different layers hierarchically (see Figure 1-4).

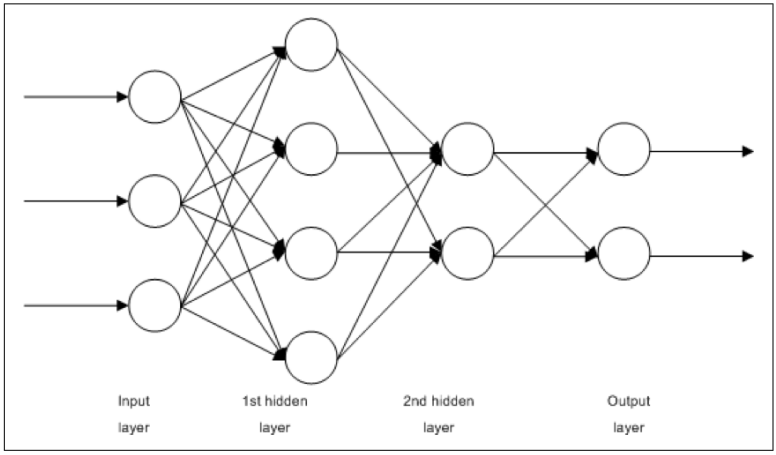


Figure 1-4. An example of a feed-forward fully connected neural network with two hidden layers. (Source: https://commons.wikimedia.org/wiki/File:Multilayer_Neural_Network.png)

As you can see in Figure 1-5, training machine learning algorithms is an iterative process, and in the specific case of deep learning, it involves the adjustment of the neuron weights to the optimal values in order to reduce the training error. With this iterative process and defined set of rules, deep learning is creating algorithms that learn complex functions from data and allows it to make predictions.

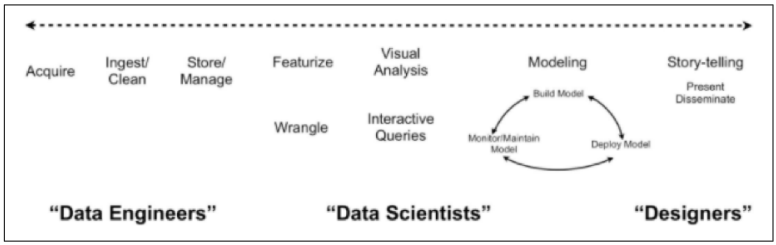


Figure 1-5. Common end-to-end steps that are carried out when analyzing data. The modeling part is an iterative process.

Because the main goal of deep learning is to generalize unseen or future data, it becomes mandatory to avoid overfitting the data as much as possible. A good practice in machine learning is to use three different datasets: training, validation, and test. The model parameters are learned using the training set, the validation set is used to determine the stopping point, and, finally, the test set is used to assess the performance of the model. *Overfitting* occurs when we have a deep learning model that learns both patterns and the noise from the training set, instead of the underlying statistical structure of the dataset. The result is a model that performs extremely well in training data and poorly on future data because the algorithm memorizes all possible combinations. On the other hand, the effect of *underfitting* the training data is also something to be avoided during the machine learning model development process because when training data is underfit, we are not able to learn its underlying patterns. In Figure 1-6 we illustrate the effect of underfitting, overfitting, and a model that is just right.

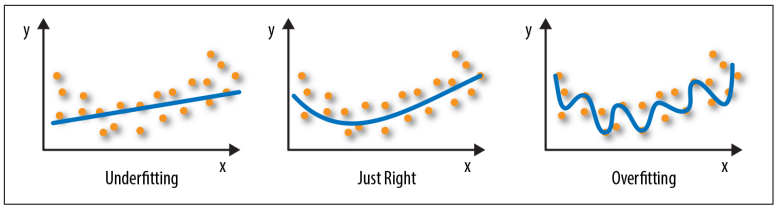


Figure 1-6. An example of a model that underfits, overfits, or learns the patterns in the data.

A common practice in developing a deep learning model is to overfit the training data. When that is done, you can begin applying tricks to deal with overfitting such as *Dropout*, *L1/L2 regularization*, or *early stopping*. Dropout is a technique that randomly "turns off" some neurons of the network. Surprisingly, this generates a more flexible network and performs a better generalization. L1 and L2 regularization are methods that add a penalty term to the function used to guide the training process, also known as *loss function*. Finally, early stopping consists of monitoring the error evolution of the training dataset on a validation dataset that is used as a proxy for the generalization error. When the error in the validation set becomes higher than a previous number of iterations, the training would stop. This method requires the developer to define the number of iterations carried out before the learning phase stops. The main idea is to check the ability of the model to generalize or predict on a dataset different from the one that is being used for training (see Figure 1-7).

In the training phase, since we usually have a limited amount of data, it becomes extremely important to train the networks to ensure the best possible generalization over future data. The best way to measure this generalization error (and also avoid overfitting) is to use cross-validation in the training phase. Cross-validation involves partitioning the training set into complementary subsets and perform the training and validation steps in different subsets. The size of each subset is specified by a user-defined parameter, so if we are talking about a 4-Fold Cross-Validation we refer to four different combinations using 25% of available data for validation in each round.

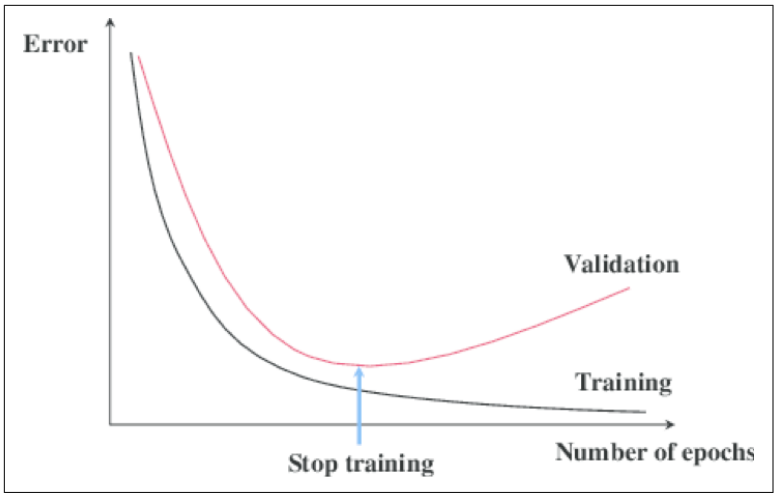


Figure 1-7. Early stopping. The validation error begins increasing (point marked as "Stop training") as the learning phase finishes.

To quantify the benefit of using more training data in deep learning models, we can measure the evolution of training and test errors as we increase the amount of training data. The test set simply explains how well the model is performing but won't be used in the training phase. If the test error decreases as more data is added to the training set, it suggests that using a bigger training set and more data usually improves model performance (see Figure 1-8).

Figure 1-8. High variance or Overfit training curve. (Source: Andrew Ng; used with permission)

On the other hand, if we add more data but the test and train error converge into similar undesired values, adding more data usually does not help and you need to try more complex network models or use more input features, as demonstrated in Figure 1-9.

Figure 1-9. High bias or Underfit training curve. (Source: Andrew Ng; used with permission)

To avoid the overfitting effect, Hinton et al. proposed in 2012 the Dropout method that provides a better generalization when training deep networks.[2]

Neural network parameters consist of the weights and biases of the neuron connections, which might be initialized to a specific value. Networks are very sensitive to this initialization method because a network with better weights in their connections will converge earlier. Several advances have been made to the initialization method as well, like the Xavier initialization method or the PReLU initialization.

An important aspect to consider when developing deep learning applications is the network topology and the value of the configuration parameters. Network topology is related to the number of hidden layers and the activation functions. Selecting an adequate number of layers and neurons is one of the most time-consuming tasks in developing deep learning applications. A common practice

2 For more details, we refer you to this article published in 2014 in the *Journal of Machine Learning Research*.

is to start by training smaller networks and increase the number of layers and neurons until you reach an upper bound.

As you can see, building and training your own deep learning model from scratch is not an easy task. Another option is to deploy prebuilt deep learning models and use Software as a Service (SaaS) hosted in the cloud.

Option 2: Deploying Prebuilt Deep Learning Models

Successfully applying deep learning methods to your enterprise use case requires more than just a good knowledge of which algorithms exist and how they work. There are a lot of examples in the research literature about the application of deep learning to solve a specific problem. However, successfully deploying a deep learning system inside an enterprise requires you to carefully consider more steps. Of critical importance is the integration of the deep learning system with current subsystems, and how the outcomes of the model will be applied.

A good deep learning developer also needs to know the available options for prebuilt models in order to choose the best one to deploy for their particular use case. One of the main questions is what kind of architecture should be considered—on-premises, or SaaS/hosted in the cloud. The main goal of SaaS architectures is to provide enough flexibility for the developer, both in terms of the available hardware (CPUs and GPUs) and in the software frameworks and system administration. By contrast, on-premises architecture can be useful if there are privacy restrictions and you expect an extensive use of the system that requires specific hardware. However, current cloud solutions provide enough flexibility and computation power to learn complex models, so this option is the one that provides the lowest barrier to entry in terms of hardware investment.

Nevertheless, if you decide to use a SaaS architecture, there are some challenges that you need to consider. First of all, it is necessary to have direct connectivity with the input data, both for the training phase (if you need to retrain your models) and for the predictions you will make using the model. So, it is common to have a *data lake* connected to the SaaS solution. This input data also usually needs

some transformation, given that input data must be normalized before using the model.

Aside from the technical requirements of deploying a deep learning model, finding available skilled resources are another challenge. There is a huge demand right now for people who have the knowledge of deep learning systems. One way to cope with this scarcity of talent is to use frameworks that simplify and automate most of the common steps in the deep learning process. Although the basic concepts must be understood by the deep learning developers in order to determine the right algorithm, this new level of automation removes many time-consuming steps.

For instance, IBM has recently launched Deep Learning as a Service (DLaaS) within IBM Watson Studio. With this service, it is possible to design deep learning models in the cloud by using on-demand GPU clusters and coding with popular frameworks like TensorFlow, Caffe, PyTorch, and Keras. You also can train models by using, a graphical interface known as Neural Network Modeler. The tool allows users to design neural network models using a drag-and-drop interface. You can export these network models to TensorFlow, Keras, PyTorch, and Caffe, as well as in JSON format to share within blogs or for posting to GitHub.

One of the novel features is the Experiment Assistant that supports the end-to-end workflow. It simplifies the execution of hundreds of experiments with different hyperparameters and automatically distributes the code in GPU-enabled containers. You can access results from the Experiment Assistant via the Python Client using notebooks, the command-line interface (CLI), or the GUI. Hyperparameter Optimization also brings automation and efficiency to the process of selecting the best hyperparameters.

Let's take a look at DLaaS in action.

Developing Deep Learning Models with IBM DLaaS

It is difficult to find an area of life where the representation of ideas through multiple layers of abstraction is not useful; this is one of the reasons deep learning receives so much attention.

A well-known business problem is *customer churn*, which means loss of a customer. Being able to predict customer churn in advance gives enterprises an opportunity to retain them by taking actions to reduce the likelihood that the customer will leave. A good example of this in the telecommunication sector has been presented by Zaratiegui et al. in 2016 at the IJCAI conference. In this work, the use of GPUs and convolutional neural networks provide a significant advantage over traditional machine learning methods. Although this example demonstrates how deep learning models simplify the feature engineering process, there still remain the challenges of creating and tuning different neural networks to find the best model for your use case.

To overcome these challenges, services like IBM's DLaaS with IBM Watson Studio allow you to train numerous models to identify the right combination of data and hyperparameters, and therefore perform experiments faster. This iterative cycle is accelerated by simplifying the process of training models in parallel using an on-demand cluster.

Let's have a look to the modeling process using the DLaaS CLI.

The first step is to upload your data into the IBM Cloud Object Storage service instance. The output of the model and log files are also written to the same cloud object storage. For this step, you need to set up Cloud Object Storage.[3]

Then, it is also necessary to define a training run. Each training run consists of the following parts: the model definition in one of the supported frameworks, the configuration on how to run the training including the number of GPUs and the location of the IBM Cloud Object Storage that contains your data set (see Figure 1-10).

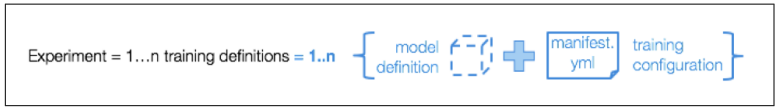


Figure 1-10. Illustration of the components of a deep learning experiment. (Figure courtesy of IBM)

Each model needs to follow some coding standards for the input, the output model, the logs, and, more important, for the hyperpara-

3 See this article for an explanation of Cloud Object Storage.

meters you would like to tune using DLaaS.[4] The code that generates the deep learning model in whatever framework you choose needs to be packaged using the standard ZIP format.

To launch the models, you need to create a manifest file, essentially a YAML file with some description fields such as the name, the framework version, the execution command, the hardware resources you want to use, and the input data.

It is important to note that all the folder references should use relative paths of the form *${DATA_DIR}/path*. You can generate an example of a training runs manifest by running the following command:

```
$>bx ml generate-manifest training-definitions
```

When the model is ready and you want to begin training, you can use the following command:

```
$>bx ml train <path-to-model-definition-zip> \
    <path-to-training-manifest-yaml>
```

This gives you a unique model ID with which you can continuously monitor the logs:

```
$>bx ml monitor training-runs model-ID
```

Internally, IBM Watson Studio allocates a Kubernetes container to each training run, along with the requested resources and the specific deep learning framework. Training runs are executed in parallel depending on the available GPU resources, as shown in Figure 1-11.

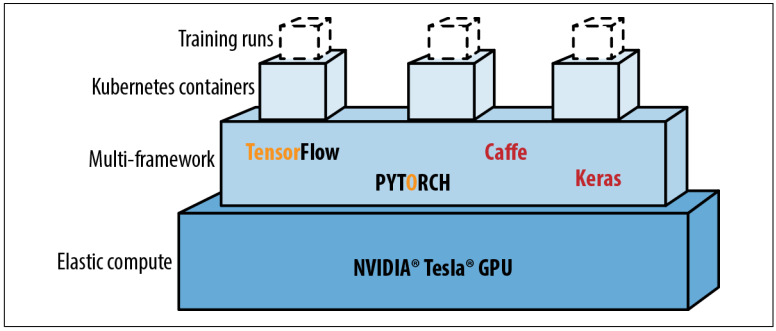


Figure 1-11. An example of four training runs that are allocated to four Kubernetes containers. (Figure courtesy of IBM)

4 See this documentation for more details on the coding style.

The main advantage of this flexible schema is that resources are allocated on demand, so you are charged only for the time you actually use it.

Finally, after you've finished your experiments and have selected an optimal model, it is possible to deploy this model as a REST endpoint and generate the predictions for incoming data

An Example of DLaaS in Action

Let's have a closer look at this process by developing an example. The aim will be to predict the final price of a home using 79 explanatory variables. You can download the data and have a look to the description from kaggle.

First, we create a specific bucket for training data:

```
$>aws \
--endpoint-url=http://s3-api.us-geo.objectstorage.softlayer.net \
    s3 mb s3://housetrain
```

Next, we upload the data to the Cloud Object Storage:

```
$>aws \
--endpoint-url=http://s3-api.us-geo.objectstorage.softlayer.net \
    s3 cp train.csv s3://housetrain
```

Now, we need to prepare the code. The following example reads the input, imputes missing values, and creates a TensorFlow model using numerical features. The model is then evaluated using a generated test set from the training data.

```
import pandas as pd
input_data_folder = os.environ["DATA_DIR"]
def main(argv):
    i = 1
    while i <= 2:
        arg = str(argv[i])
        if arg == "--learningRate":
            learning_rate = float(argv[i+1])
        elif arg =="--trainingIters":
            training_iters = int(argv[i+1])
        i += 2

    df_train = pd.read_csv(os.path.join(input_data_folder,
        "train.csv"))
    df_train = df_train.select_dtypes(exclude=['object'])
    df_train.drop('Id',axis = 1, inplace = True)
    df_train.fillna(0,inplace=True)
```

```
# Rescale the data with the function MinMaxScaler from sklearn
col_train = list(df_train.columns)
col_train_bis = list(df_train.columns)

col_train_bis.remove('SalePrice')

mat_train = np.matrix(df_train)
mat_new = np.matrix(df_train.drop('SalePrice',axis = 1))
mat_y = np.array(df_train.SalePrice).reshape((1460,1))

prepro_y = MinMaxScaler()
prepro_y.fit(mat_y)

prepro = MinMaxScaler()
prepro.fit(mat_train)

train = pd.DataFrame(prepro.transform(mat_train),
    columns = col_train)

# To use tensorFlow we will transform the features
# List of features
COLUMNS = col_train
FEATURES = col_train_bis
LABEL = "SalePrice"

# Columns for tensorflow
feature_cols = [tf.contrib.layers.real_valued_column(k)
    for k in col_train_bis]

# Training set and Prediction set with the features to predict
training_set = train[COLUMNS]
prediction_set = train.SalePrice

# Train and Test
x_train, x_test, y_train, y_test = train_test_split(
    training_set[FEATURES],
    prediction_set,xtest_size=0.33, random_state=42)
y_train = pd.DataFrame(y_train, columns = [LABEL])
training_set = pd.DataFrame(x_train, columns = FEATURES).merge
    (y_train, left_index = True, right_index = True)

training_sub = training_set[col_train]
# Same thing but for the test set
y_test = pd.DataFrame(y_test, columns = [LABEL])
testing_set = pd.DataFrame(x_test, columns = FEATURES).merge
    (y_test, left_index = True, right_index = True)

tf.logging.set_verbosity(tf.logging.ERROR)
regressor = tf.contrib.learn.DNNRegressor(feature_columns=
    feature_cols, activation_fn = tf.nn.relu,
```

```
        hidden_units=[50, 25, 10, 10, 6],
            optimizer = tf.train.GradientDescentOptimizer
         (learning_rate=learning_rate))

    # Reset the index of training
    training_set.reset_index(drop = True, inplace =True)

    regressor.fit(input_fn=lambda: input_fn(training_set,
        FEATURES, LABEL), steps=training_iters)

    # Evaluation on the test set created by train_test_split
    ev = regressor.evaluate(input_fn=lambda: input_fn(testing_set,
        FEATURES, LABEL), steps=1)

    loss_score1 = ev["loss"]
    print("Final Loss on the testing set: {0:f}".format(
        loss_score1))

def input_fn(data_set, FEATURES, LABEL, pred = False):
    if pred == False:
        feature_cols = {k: tf.constant(data_set[k].values)
                for k in FEATURES}
        labels = tf.constant(data_set[LABEL].values)
        return feature_cols, labels
    if pred == True:
        feature_cols = {k: tf.constant(data_set[k].values)
            for k in FEATURES}
        return feature_cols

if __name__ == "__main__":
    main(sys.argv)
```

We also need to define a manifest file. This YAML file has all of the metadata necessary to launch and execute the models:

```
model_definition:
  name: tf-houseprice
  author:
    name: DL Developer
    email: dl@example.com
  description: House price model
  framework:
    name: tensorflow
    version: "1.5"
  execution:
    command: python3 house_price_network.py --learningRate 0.001
        --trainingIters 20000
    compute_configuration:
      name: k80
training_data_reference:
  name: training_data_reference_name
```

```
  connection:
    endpoint_url:
      https://s3-api.us-geo.objectstorage.service.networklayer.com
    access_key_id: XX
    secret_access_key: XX
  type: s3
  source:
    bucket: housetrain
training_results_reference:
  name: training_results_reference_name
  connection:
    endpoint_url:
      https://s3-api.us-geo.objectstorage.service.networklayer.com
    access_key_id: XX
    secret_access_key: XX
  type: s3
  target:
    bucket: housetrain
```

Now, to deploy the model and start the execution in the cloud, you need to package the model code into a ZIP file and send it together with the YAML file using the `bx` command, like the following:

```
$> bx  ml  train  tf-model.zip  tf-train.yaml
```

The main idea in this example is that you can easily train deep learning models and change the parameters by remotely publishing this code into DLaaS within Watson Studio. In our example, we can change the parameters in the YAML file to train different executions. The best way to perform this task is to use the Hyperparameter Optimization.[5]

The Future of Deep Learning

AI and machine learning are transforming the way we develop software and do business. According to a published survey by MIT *Sloan Review*, 76% of enterprises with at least $500 million in revenue claim to having plans to use machine learning to increase sales, and at least 40% of them already use it in sales and marketing. So, there is still a great opportunity to deploy these techniques more broadly in business.

Deep learning, as we've discussed, is still a very active area of research and we are going to see many advances in the near future.

5 An explanation of the Hyperparameter Optimization is located here.

One drawback of current models for enterprise applications is that deep learning algorithms require large amounts of labeled data to achieve good accuracy, though there are efforts underway to reduce the amount of required labeled data. Transfer learning, for example, is the idea of using models that have learned from a lot of labeled data in scenarios that have only a small amount of training data. It turns out that this technique enables new models to learn quicker.[6]

Tools like Neural Network Modeler within Watson Studio lowers the barrier of entry for developing deep learning models because it provides a drag-and-drop interface to configure and design the network. Several advances have also simplified and optimized the process of selecting hyperparameters when building deep learning models. Essentially training deep networks can be seen as a complex optimization problem for which we need to explore a huge set of hyperparameters (parameters of the model that can't be learned from training data). Tools aimed to simplify this process, like Hyperparameter Optimization with Watson Studio, have been very well received by the deep learning community. Researchers at Google DeepMind also have several efforts in the works to develop novel models that perform hyperparameter optimization, such as Population-Based Training.

Capsule Networks are a recent improvement over Convolutional Neural Networks (CNN), recently proposed by Sabour, Frosst, and Hinton in a paper titled "Dynamic Routing Between Capsules". The idea is to improve CNN and, in particular, the max-pooling step, by having information about the location of the patterns. For example, they will provide the same probability of the presence of an object if you change the viewpoint. Internally, they concatenate several convolutional layers. For more information, we refer you to the following post.

There are also a lot of people collaborating to ensure AI—and deep learning—is used in a way that will benefit humanity. We refer you to OpenAI, the Allen Institute for Artificial Intelligence, and the NASA Frontier Development Lab to learn more about using AI for social good.

6 You can find a good overview of transfer learning in this blog post.

Finally, it seems that the amount of code we need to develop to generate deep learning models will lessen in the coming years. As we have seen, products like IBM DLaaS within IBM Watson Studio or DeepCognition are aiming to help developers solve problems and be more productive in their daily work.

About the Author

Federico Castanedo is the Lead Data Scientist at Vodafone Group in Spain, where he applies AI techniques using massive amounts of data. Previously, he was Chief Data Scientist and cofounder at WiseAthena.com, a company that provides business value through AI. For more than a decade, he has been involved in projects related to data analysis in academia and industry. He has published several scientific papers about data fusion techniques, visual sensor networks, and machine learning. He holds a PhD in artificial intelligence from the University Carlos III of Madrid (2010) and has also been a visiting researcher at Stanford University.