Chapter 2: Machine learning basics

After completing this chapter, you will understand the basics of machine learning and the types of applications that it is useful for. You will be introduced to tensors, TensorFlow, and Keras.

Table of contents

Document conventions

Convention	Usage	Example
`Courier New`	Displays code and text commands	`CY_ISR_PROTO(MyISR);` `make build`
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

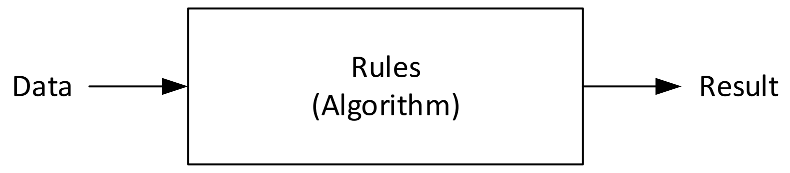
2.1 Glossary

Before diving into machine learning, here is a glossary of terms. Even though many of these may not make sense yet, you can refer back to this table when a new term is introduced.

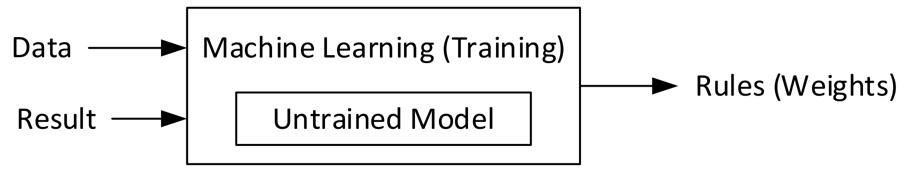
Term	Definition
Back-propagation	An algorithm used during training to test for errors working back from output nodes to input nodes.
Batch or Mini-Batch	The number of samples to send through the model during training before updating the model's weights.
Binary classification	A task in which outputs can only be one of two categories.
Classes	The set of possible labels for a classification machine learning model.
Epoch	One training pass of the entire training data set. If the data is split into batches, an epoch is complete once all batches have been put through the model.
Features	Independent variables that act like inputs into the system.
Gradient descent	An optimization algorithm used during training to minimize loss.
Generalization	The model's ability to provide good results on previously unseen data.
Inference	The process of sending data into a machine learning model to calculate an output. The "Inference Engine" is the system component (in our case, a software library) that applies rules to the model to produce a prediction.
Label	This is attached to a sample to indicate what the target is for the given sample.
Layer	A component of a machine learning model that performs some transformation on the data. Deep learning models have three or more layers, but often many more.
Learning Rate	A measure of how much the weights are changed between iterations to attempt to find the minimum loss point for the model.
Loss	The difference between the expected output from a model and the actual output from the model. This is the quantity that should be minimized during training.
Multi-class classification	A task in which outputs are categories from a list of possible outputs (>2).
Multi-label classification	A task in which a single sample can have more than one label from the list of possible outputs.
Neural Network (NN)	A specific type of machine learning model that uses interconnected nodes in a layered structure that resembles neurons in the human brain.
Prediction, Output or Result	The output from a machine learning model.
Rules or Weights	The parameters that are used to influence how a model determines its predictions. These are adjusted during training.
Sample, Input or Data	An input to a machine learning model.
Scalar regression	A task in which the output is a continuous value rather than a discrete item from a list.
Target	The expected or correct output from a machine learning model.
Training	The process of adjusting model weights to optimize how well the model predicts the correct output for any given input.
Vector regression	A task in which the output is a set of continuous values.

2.2 What is machine learning?

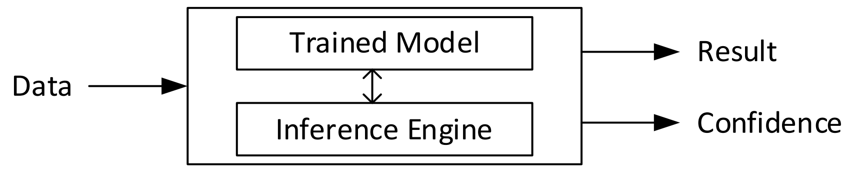
With traditional programming, you write a set of rules (i.e. an algorithm) that takes data as an input and then comes up with a result or an answer as an output.



However, with machine learning, instead of writing an algorithm you write a model. You also provide a set of input data samples and the expected result for each sample (called a label). The ML system uses the input data and expected results to come up with a set of rules (called weights) that will be used by the model to provide a result for any given input. The process of coming up with the rules is called "training the model."



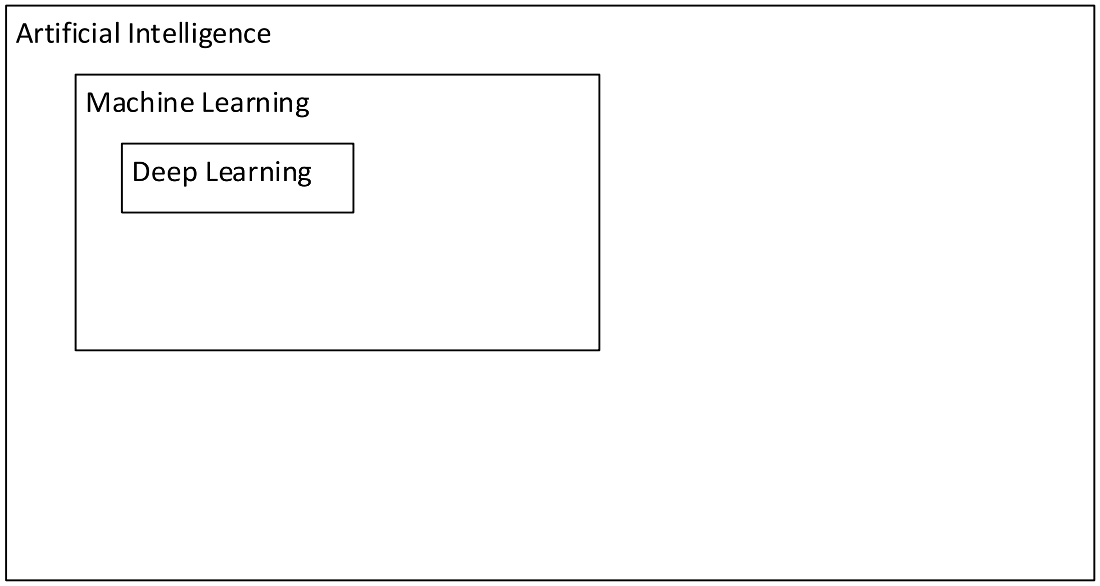
Once the model has been trained, it can be deployed to classify inputs into categories or predict results for numerical values with any input data (within reason). The model can also provide a confidence in its result, showing how likely it is that the result is correct. The inference engine shown below is the processing component of the system, doing calculations based on the model that you provide.



The end result is ultimately the same: you provide data to the MCU and it provides the answer for that data. The difference is in how you get there. Will you manually write an algorithm? Or will you provide lots of data and matching answers to a machine learning system that does the work for you?

2.2.1 Artificial intelligence, machine learning, and deep learning

Before getting deeper into ML, it's worth taking a step back to understand how artificial intelligence (AI), machine learning, and deep learning fit together. Essentially, each one is a subset of the one before it.



Artificial intelligence is the attempt to take intellectual tasks that would typically be performed by a human and automate them using a computer. There has been lots of work done on different AI approaches over many years.

One such AI approach is machine learning. As mentioned previously, ML is a system that uses a model trained to recognize patterns between input data and expected results. Once the model is trained it can be used on new data to predict the expected result. ML has been very useful in solving certain types of problems and is at the forefront of AI research.

Deep learning is a subset of machine learning in which the model is comprised of multiple layers. Each layer performs a transformation on the data that makes it increasingly useful in obtaining the desired result.

As an example of a transformation, electrical engineering problems are often greatly simplified by using a Laplace transform to convert the time domain to the frequency domain. For a capacitor, the relationship between voltage and current is:

Time domain:	$i = C\frac{dv}{dt}$	(differential equation)
Frequency domain:	$I = j\omega CV$	(linear equation)

The Laplace transform turns a differential equation into an algebraic equation. It represents the same thing, but now the data is in a format that makes it easier to arrive at the result.

A deep learning model for a complex problem may contain hundreds or even thousands of layers to gradually simplify the process of arriving at the correct result for a given input.

2.3 Applications of machine learning

Machine learning is great at solving some of the problems that would be extremely difficult or even impossible to navigate with traditional programing techniques. However, both methods still have their place. Specifically, machine learning is more adept at handling perception problems where variations in the inputs will occur and large input data sets can be produced for learning. For example, voice recognition, handwriting recognition and face recognition are excellent use cases for machine learning.

On the other hand, problems where specific inputs lead to specific outputs are generally better handled with traditional programming. For example, creating a machine learning model to sort lists of numbers is theoretically possible but it would require large models and large input data sets for something that could be done easily with just a few lines of computer code. Even worse, a machine learning model for sorting would occasionally produce the wrong result for new input number sets.

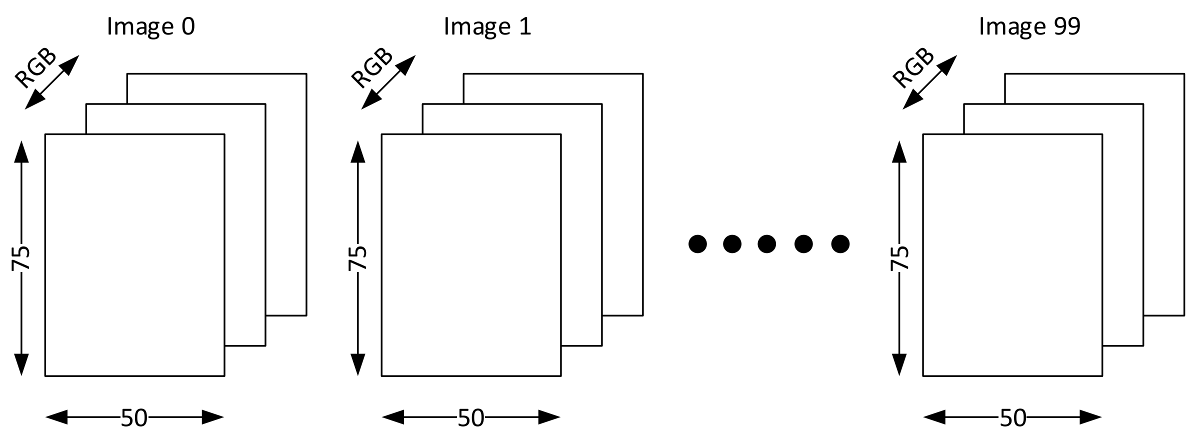
Overall, it's important to first understand the problem well enough to determine if machine learning is really the best approach before investing in a machine learning model to solve a problem.

2.4 Tensors

Machine learning systems store data in the form of multi-dimensional arrays called "Tensors". A tensor is just a name for a container for data - usually numerical data. The number of axes of a tensor is called its "rank". For example, a scalar value is a rank-0 tensor, a vector is a rank-1 tensor, and a 2D matrix is a rank-2 tensor. Each rank in the tensor can contain scalar values or vectors. Vectors will have a number of entries called "dimensions." Don't confuse that with the dimensions (or axes) of the tensor itself.

In addition to the rank, tensors have a "shape" and a "data type." The shape describes how many dimensions the tensor has along each axis while the data type describes the type of data that the tensor contains – uint8, float16, float32, etc.

Consider a machine learning model used for image classification. In this case, the tensors will be rank-4 (samples, height, width, color channels). As an example, assume we have 100 color images with a height of 75 pixels and width of 50 pixels. The color is represented for each pixel as 3 RGB values. The shape of the tensor will be (100, 75, 50, 3). Pictorially, it resembles this image:



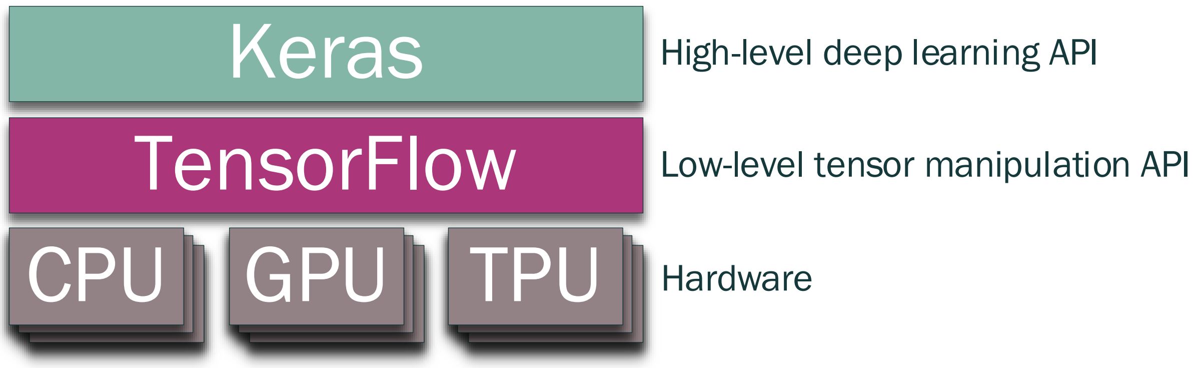
In TensorFlow, which we will discuss next, the color channel is moved before the height and width in the tensor (this is called channel-first representation), so it would instead be organized as (samples, color channels, height, width) and the shape would be (100, 3, 75, 50). This is done to improve the calculation efficiency.

2.5 TensorFlow and Keras

TensorFlow is an open-source machine learning platform that was developed primarily by Google. It is Python-based so if you are familiar with Python, the syntax of TensorFlow will look familiar. It is a low-level API with functions that can facilitate manipulating tensors (e.g. reshape, dot product). In addition, the TensorFlow API can automatically calculate the gradient of any differentiable function. Keep this in mind for when we discuss model training later.

TensorFlow can be run on a CPU, GPU, or TPU (tensor processing unit) which is specifically optimized for tensor operations. Operations can be distributed across many machines in parallel. This makes it possible to train extremely large models with extremely large data sets.

Keras is a set of higher-level Python deep learning APIs that can sit on top of TensorFlow. It provides a way to define and train a deep learning model at a higher level than writing individual TensorFlow code. It is analogous to a HAL for TensorFlow. Keras supports other underlying tensor APIs, but TensorFlow is by far the most commonly used.


Keras
High-level deep learning API
TensorFlow
Low-level tensor manipulation API
CPU
GPU
TPU
Hardware

More information on these APIs can be found at tensorflow.org, and keras.io.

2.5.1 TensorFlow Lite (TF Lite)

While TensorFlow is intended for computer such as PCs or even servers, TensorFlow Lite is optimized for deploying ML models on mobile and edge devices containing smaller but still relatively high-powered MCUs such as Cortex-A class devices. It supports a subset of the operations that are supported in the full TensorFlow API.

2.5.2 TensorFlow Lite for Microcontrollers (TFLM)

TensorFlow Lite for Microcontrollers contains even smaller subset of operations than TF Lite. It is intended for deploying ML models on edge devices containing MCUs such as Cortex-M class devices. For Infineon MCUs such as a PSoC™ 6, we will be using TFLM models. Some TensorFlow or TF Lite models can be converted to TFLM, but only if they don't use operations that TFLM does not support in its API.

2.6 Creating and Training Models

Keras and TensorFlow can be used to create and train custom models. This section covers the basics of those activities. However, depending on your application and the complexity of your system, you may be able to find existing models or you may decide to work with an Infineon partner to create and train models for you. Several partner systems will be covered in later chapters. Either way, it is still useful to understand a little bit about what goes on "beneath the surface."

2.6.1 Model types

Machine learning models can be organized into four categories:

- binary classification
- multi-class classification
- scalar regression
- vector regression

Each is described below.

Binary classification models choose between two categories. A good example are the robot tests that I'm sure you are familiar with from the internet such as "select all images that contain a stop sign." It is simple for a human to do this task but much more difficult for a computer; unless of course it has been trained to do that specific task with a machine learning model.

Multi-class classification models group items into more than two categories. For example, a pet image classification model might sort pictures into categories for images that contain dogs, cats, birds, fish, etc. This is also an example of a multi-label classification since one image may contain more than one type of pet. Such as a cat and a dog in the same picture.

Scalar regression models output a continuous scalar value instead of a discrete classification. For example, a machine learning model that attempts to predict future stock prices would output a predicted value.

Vector regression models output a set of continuous values rather than just a single value.

2.6.2 Model architecture

As previously mentioned, models in deep learning consist of multiple layers. Every layer takes one or more inputs from the layer above it, transforms the data in some way, and then produces one or more outputs to the layer below. There are many different types of layers used in ML such as:

Layer Type	**Function**
Input	Receives the initial input to the model.
Normalization	Scales the input data so that during training, adjustments to weights operate equivalently on different types of values.
Convolutional	Applies filters to the input data to isolate desired features to be detected in the data.
Activation	Applies mathematical functions to the input data.
Pooling	Decreases the sensitivity to certain features to allow a more generalized model.
Dropout	Nullifies certain random input values to help prevent overfitting the model to specific data points.
Output	Produces the output result from the model.

Depending on the problem being solved, the model may have zero, one, or multiple instances of each type of layer. The idea is that the input data is slowly transformed into a state in which the result can be easily determined. The model will always have at least an input layer and an output layer. The layers between that are often classified together as "hidden" layers.

Each layer has parameters called "weights." These are used to control how that layer operates on the input data. In fact, as you will see in a minute, the process of training a model consists of determining the set of weights for each layer that will produce the best results.

2.6.3 Model decisions

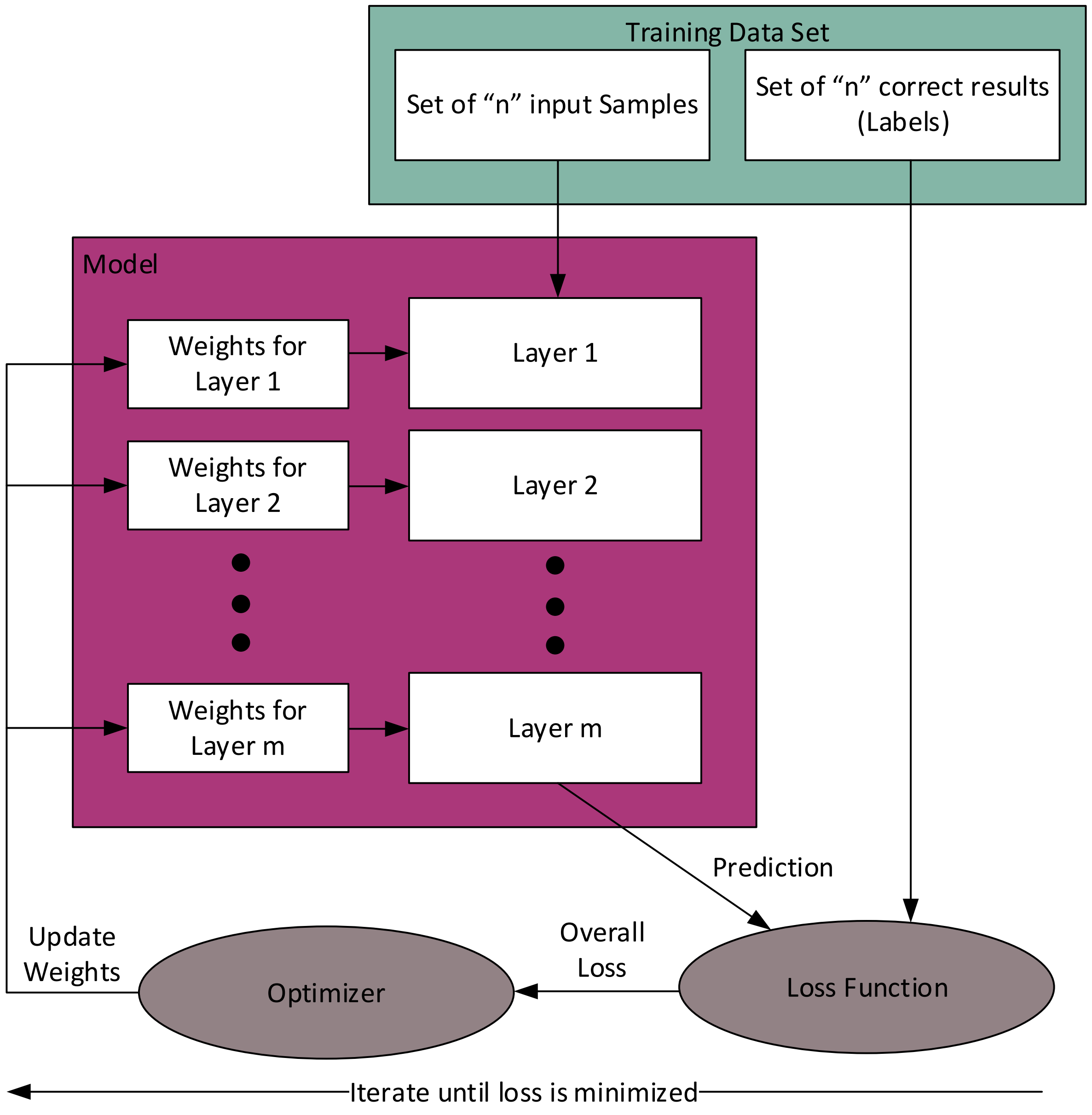
When creating a new ML model, you must ask several questions that will help determine how the model performs. These questions include:

- How many layers and what types of layers will be used?
- What units will be used for each layer and how are they normalized?
- What loss function will be used to determine how well the model is performing?

Before creating a brand-new model, it is always best to do some research first to see if a model already exists. Even if an exact model for your problem doesn't exist, it is likely that similar problems have been solved, so a good starting point for your model may be available. For hundreds of existing examples, take a look at the Keras website, which includes examples for computer vision, language processing, audio, timeseries, etc.

2.6.4 Training

Once you have a model, it needs to be trained so that it can successfully predict answers for given input data. As mentioned before, training consists of determining the correct weights for each layer. In order to accomplish this, you must provide the model with lots of example inputs (samples) that have been labeled with the correct results (labels). These examples are put through the model and the predictions it makes are compared with the labels. The difference between the set of predictions and the set of correct results is called the "loss." The weights are systematically adjusted by an optimizer until the overall loss is minimized.



2.6.4.1 Data sets

When training a machine learning model, you need a set of samples that have each been "labeled" with the correct answers. For example, if you have a model to identify pets in images, a given sample showing a dog and a cat would have two labels: "dog" and "cat." Labeled samples are important because they allow you to compare the model's output for each sample to its label(s), using the loss function. The loss function may be a simple percentage of incorrect predictions or it may be a more complex function involving multiple items and values. It depends on what is required as an output or outputs from the model.

You may have noticed in the previous figure that the data used is called the "Training data set." There are actually three types of data sets required to do proper training, validation and testing of a machine learning model. As you may expect from the names, they are:

- Training data set
- Validation data set
- Test data set

Typically, you will take your complete set of labeled samples and separate them into these three sets. The rough guideline is to use 60% of the data for the training set, 20% for the validation set, and 20% for the test set. The training data is used during training loops as illustrated above.

The validation data is also used during the training process. It is used after each set of training loops (called an "epoch" as you will see in a minute) to see how well the model performs on data that it has not seen before. This information is critical because if you keep training a model, it will continue to get better at predicting results for the training data, but will eventually get worse at predicting results for new data, an issue known as "overfitting." You will see how that occurs shortly. It is important that the training data and validation data are completely separate data sets – if you were to validate the model on the training data or a subset of it, you would end up with a model that works great on the training data but doesn't work as well out in the real world on data that it has not seen before.

Finally, once you have a trained and validated model that you think is ready for release, a final check is done to see how the model will perform in the real world by using the test data. Again, you don't want to re-use the training or validation data – the idea is to find out how the model works on data that it hasn't already seen.

All three sets of data should be as representative of the real world as possible. Therefore, you will typically need to randomize any ordered sets of samples before breaking them into three data sets. This eliminates any bias in the different data sets.

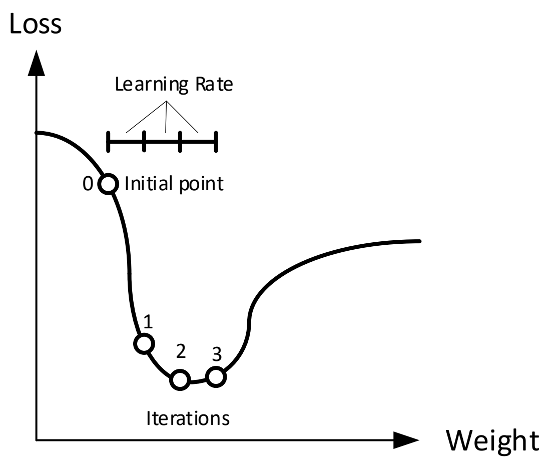
One exception to randomizing is if your model is trying to predict the future. In that case, you want to keep the samples time ordered. You will want to train and validate using older data and then use newer samples for the test data to see how well the model can predict the future from past data.

A final concept to cover with respect to the data sets is the use of mini-batches during training. This is useful for very large data sets. Instead of doing one pass of the iteration with the entire training data set, a smaller sub-set of the data (a mini-batch) is used for each iteration.

As usual, the weights are updated based on the loss calculation as well as based on the results of the mini-batch. On the next iteration, a different mini-batch is used and the weights are updated again. It turns out that using this approach can give similar results in much less time. Each iteration requires only a few samples to be passed through the model instead of the entire training data set. Once all mini-batches from the training data set have gone through an iteration, an "epoch" has been completed. Then the model is tested on the validation data set.

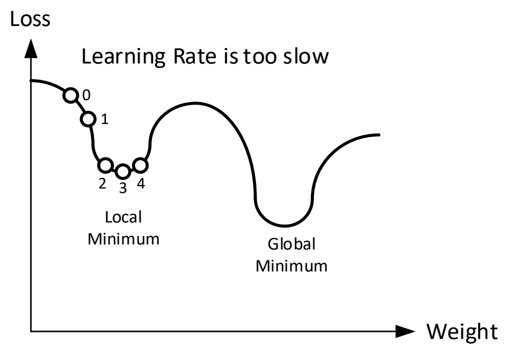
2.6.4.2 Training iterations

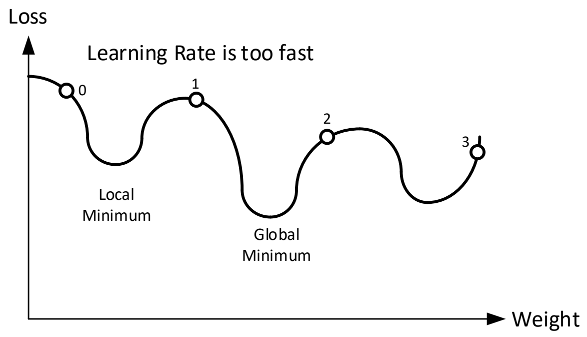
Different optimizers can be used during training, depending on the model. One commonly used algorithm is gradient descent with back-propagation. The details of those methods are beyond the scope of this course, but the result is that each training iteration adjusts the weights for all levels at once, according to the gradient of the loss function. In other words, the optimizer finds the "slope" of the multi-dimensional surface comprised of all weights from all levels and adjusts them towards the direction of decreased loss. The size of the adjustment is called the "learning rate." If we simplify our system to a single weight, the process will resemble this:



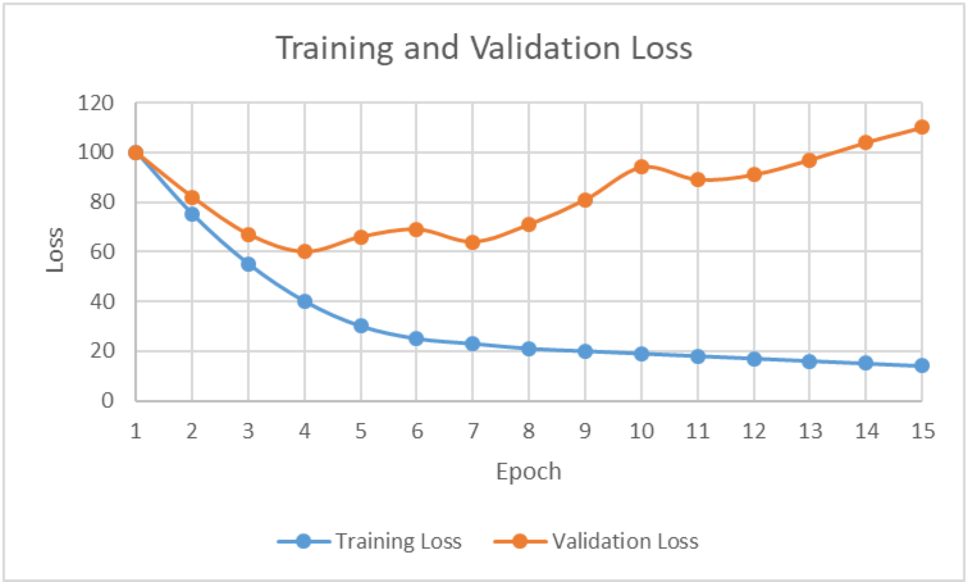
As you can see in the diagram, the optimal point is found after the second iteration, which the optimizer realizes after the third iteration.

If the learning rate is too slow the model could take a very long time to converge. Even worse, it could get stuck at a local minimum of the loss function. If the learning rate is too fast, the model may overshoot the minimum and not converge at all. For those reasons, the learning rate is often variable with adjustments based on how much the loss changes from one iteration to the next; this process is called momentum.





As we briefly mentioned earlier, if too many iterations are run while training the model, overfitting may occur. This can be easily assessed by looking at the model's loss on the training data and the validation data after each epoch.



You can clearly see why the validation data is necessary from the graph above. Without this data, we would have no idea when to stop training and we would not know how well the model would behave on new data. In this case, the lowest validation loss occurs after the fourth epoch even though the training loss continues to slowly improve with each additional epoch.

The reason overfitting occurs is that on each epoch the model weights are adjusted to focus more and more on exact details of the training data set. That is, the model will start to use obscure patterns in the training data that has nothing to do with the actual classification task.

If you are using Keras to train your models, it can automatically compute the loss calculations after each epoch on the validation data and find the epoch that gives the best results against the validation data. That is, it will run epochs until it finds the location where overfitting occurs and will then back off to the weights that were used to achieve the minimum loss point.

Once you achieve the best result possible with your starting model, you need to decide if it is good enough. If not, you may need to collect more data or even iterate on the model itself. Based on the losses that you see, you may need a model with more layers, fewer layers, different types of layers, different ordering of layers, different layer properties, etc.

2.6.4.3 Model and learning improvement techniques

The best way to improve a machine learning model is to collect more training data. Essentially, deep-learning is curve-fitting using gradients. If you don't have enough data to get a smooth curve fit, the model won't perform well.

Therefore, spending more effort during data collection is almost always better than spending more effort on the model. In some cases, you can evaluate the data you have and try to identify weak spots or missing data that is leading to poor performance. Samples can be added to the data set specifically to address holes in the data set.

Beyond collecting more data, there are a few other techniques that can help you to build a better model. A few of them are:

Augmentation – Add additional data to the dataset by using transformations of existing samples to create new samples. This is especially useful when using small training datasets. For example, if you are using a motion sensor to model repetitive gestures (e.g. a hand moving in circles), you can take each existing sample, time shift it by some amount, and store it as a new sample. This creates new samples without having to collect more data. Since the gestures are repeating, time shifting gives a sample in which the circle starts and ends at a different point than the original sample.

Note: *In the case of image data, augmentation can be done by changing brightness, contrast, or saturation, by mirroring, rotating, translating, resizing, cropping, adding noise, etc. to add many more samples each with minor variations from the original image.*

Feature engineering – Use knowledge of the problem and the data set to apply hardcoded transformations before sending the data to the model. For example, converting from cartesian coordinates to polar coordinates may help to simplify a model in some cases.

Weight regularization – Force the learning algorithm to keep the weights small. Large weight values mean that small changes in input can lead to large changes in the output. Using weight regularization impedes how well the model will fit to the training data. This may seem counter-intuitive, but it can reduce overfitting and improve generalization of the model. That is, the model may do worse against the training data, but better against data that it has never seen before.

Dropout regularization – During training, randomly exclude some number of layers from the model on each iteration. This makes the training process less likely to cause the model to be over-sensitive to a pattern that may be more prevalent in the training data than in general real-world data. Training iterations that exclude the layer or layers that are especially sensitive to the pattern will help to prevent the overall training from overfitting to those specific patterns.

Anomaly detection and filtering – Machine learning models typically perform poorly on data that is unlike anything it has seen before. Anomaly detection and filtering is a way to determine if a given input sample is unlike the existing classes (i.e. an anomaly). Rather than attempt to classify the sample and likely get it wrong, you can filter it out. This can reduce the possibility of a false classification and result in better model performance. You can even collect anomalies, label them and feed them back into training to improve the model over time (i.e. continuous training).

2.7 References

Since Artificial Intelligence, Machine Learning, and Deep Learning are hot topics, there is a lot of information available. Spend some time searching the web to see what interests you. A few good references to start with include:

- Machine Learning Crash Course

  Online class developed by Google
- Deep Learning with Python 2nd Edition by François Chollet

  E-book with lots of examples that cover deep learning using Python, TensorFlow and Keras
- Machine Learning Mastery

  Website of guides and tutorials that cover and define many ML cocepts
- TensorFlow
- Keras

Published by

Infineon Technologies AG

81726 Munich, Germany