Master thesis

Machine Learning and Deep Learning Notes

RWTH Aachen University

Lehrstuhl für Bildverarbeitung

Contents

[Mandatory checklists 1](#_Toc104410355)

[Learning Curve 2](#_Toc104410356)

[Train Learning Curve 2](#_Toc104410357)

[Validation Learning Curve 2](#_Toc104410358)

[Optimization Learning Curves 2](#_Toc104410359)

[Performance Learning Curves 2](#_Toc104410360)

[1. Underfitting 2](#_Toc104410361)

[2. Overfitting 2](#_Toc104410362)

[3. Generalization Gap 2](#_Toc104410363)

[4. Good fit 3](#_Toc104410364)

[Unrepresentative Dataset 3](#_Toc104410365)

[Unrepresentative Training Dataset 3](#_Toc104410366)

[Unrepresentative Validation Dataset. 3](#_Toc104410367)

[Quotes 4](#_Toc104410368)

[Effect of training set size [2] 6](#_Toc104410369)

[No Free Lunch theorem [3] 6](#_Toc104410370)

[Regularization [4] 6](#_Toc104410371)

[Hyperparameter [6] 6](#_Toc104410372)

[Validation set [7] 7](#_Toc104410373)

[Probability 8](#_Toc104410374)

[Cross Entropy [8] 11](#_Toc104410375)

[Properties of Maximum Likelihood [10] 11](#_Toc104410376)

[Bayesian Statistics [11] 11](#_Toc104410377)

[Choosing a loss function [13] 11](#_Toc104410378)

[Denisity estimation [14] 11](#_Toc104410379)

[Cross Entropy [15] 12](#_Toc104410380)

[L1 and L2 losses [18] 13](#_Toc104410381)

[Which loss function leads to fast convergence [18]? 13](#_Toc104410382)

[Loss functions: 14](#_Toc104410383)

[Losses [22] 14](#_Toc104410384)

[Mean Squared Loss 14](#_Toc104410385)

[Mean Squared Logarithmic Loss 14](#_Toc104410386)

[Binary Cross-Entropy Loss 14](#_Toc104410387)

[Hinge Loss 14](#_Toc104410388)

[Squared Hinge Loss 15](#_Toc104410389)

[Multi-Class Cross-Entropy Loss 15](#_Toc104410390)

[Sparse Multi-Class Cross-Entropy Loss 15](#_Toc104410391)

[Kullback Leibler Divergence Loss 15](#_Toc104410392)

[Batch Normalization in CNN [19] 16](#_Toc104410393)

[Why does Batch Normalization Work? [19] 17](#_Toc104410394)

[Advantages of Batch Normalization: 17](#_Toc104410395)

[Machine Learning Practices and Advices 18](#_Toc104410396)

[Data Leakage 18](#_Toc104410397)

[Gradient Descent: Optimizer[26] 18](#_Toc104410398)

[Sampling methods for Gradien Descent[26] 18](#_Toc104410399)

[Convolution Neural Networks [26] 20](#_Toc104410400)

[Convolution Neural Networks Models [26] 20](#_Toc104410401)

[Abstraction libraries for Tensor flow [26 – Chapter 7] 21](#_Toc104410402)

[Types of machine learning [27]. 21](#_Toc104410403)

[Batch on online learning: 21](#_Toc104410404)

[Batch learning: 21](#_Toc104410405)

[Online learning: 21](#_Toc104410406)

[Instance based learning and Model based learning: 21](#_Toc104410407)

[Instance based learning: 21](#_Toc104410408)

[Model based learning: 22](#_Toc104410409)

[Main challenges of machine learning [27]. 22](#_Toc104410410)

[Irrelevant features: 22](#_Toc104410411)

[Performance measure [27] 22](#_Toc104410412)

[Accuracy 22](#_Toc104410413)

[Confusion matrix 22](#_Toc104410414)

[Precision and Recall 22](#_Toc104410415)

[- Precision Recall tradeoff 22](#_Toc104410416)

[- Precision Recall curve 22](#_Toc104410417)

[Dimensionality Reduction [27] 23](#_Toc104410418)

[Curse of dimensionality 23](#_Toc104410419)

[Solution to the curse 23](#_Toc104410420)

[Manifold Learning 24](#_Toc104410421)

[- Manifold 24](#_Toc104410422)

[- Manifold Assumption [27, page 210] 24](#_Toc104410423)

[PCA – Principal Component Analysis 24](#_Toc104410424)

[- Preserving the Variance. 24](#_Toc104410425)

[- Principal Components. 24](#_Toc104410426)

[- Explained Variance Ratio. 25](#_Toc104410427)

[- Choosing right number of dimensions. 25](#_Toc104410428)

[- Incremental PCA 25](#_Toc104410429)

[Other Dimensionality Reduction Techniques [27, page 223] 25](#_Toc104410430)

# 

# Mandatory checklists

1. In neural networks, it's often great to work in the ranges of [0, 1] or [-1, 1] or around there.
2. Always plot your data with color bars, which helps you catch issues before training.
3. Some of the most popular models like to have inputs that are sized as multiples of 2^N for N being the number of layers.
   1. Here, we force our images to be size 32 (2x 2^4).

<https://machinelearningmastery.com/gradient-descent-for-machine-learning/>

1. Optimization is a big part of machine learning.

2. Gradient descent is a simple optimization procedure that you can use with many machine learning algorithms.

3. Batch gradient descent refers to calculating the derivative from all training data before calculating an update.

4. Stochastic gradient descent refers to calculating the derivative from each training data instance and calculating the update immediately.

1. One cycle through the entire training dataset is called a training epoch.

2. Commonly, batch gradient descent is implemented in such a way that it requires the entire training dataset in memory and available to the algorithm.

3. Mini-batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

<https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/>

1. Refer to the papers mentioned at the end of this post.

2. Papers have been downloaded.

1. The number of epochs: = is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.
   1. You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset.
   2. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified “batch size” number of samples.
   3. The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized.
      1. You may see examples of the number of epochs in the literature and in tutorials set to 10, 100, 500, 1000, and larger.
   4. Learning curve is a plot of epochs vs error.

# Learning Curve

<https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>

1. Learning curves are plots that show changes in learning performance over time in terms of experience.
2. Learning curves of model performance on the train and validation datasets can be used to diagnose an underfit, overfit, or well-fit model.
3. Learning curves of model performance can be used to diagnose whether the train or validation datasets are not relatively representative of the problem domain.

## Train Learning Curve

Learning curve calculated from the training dataset that gives an idea of how well the model is learning.

## Validation Learning Curve

Learning curve calculated from a hold-out validation dataset that gives an idea of how well the model is generalizing.

## Optimization Learning Curves

Learning curves calculated on the metric by which the parameters of the model are being optimized, e.g., loss.

## Performance Learning Curves

Learning curves calculated on the metric by which the model will be evaluated and selected, e.g., accuracy.

### Underfitting

1. The training loss remains flat regardless of training.

2. The training loss continues to decrease until the end of training i.e., the training was stopped prematurely.

### Overfitting

1. The plot of training loss continues to decrease with experience.

2. The plot of validation loss decreases to a point and begins increasing again.

.\*. The inflection point in validation loss may be the point at which training could be halted as experience after that point shows the dynamics of overfitting.

### Generalization Gap

1. The loss of the model will almost always be lower on the training dataset than the validation dataset.

2. This means that we should expect some gap between the train and validation loss learning curves. This gap is referred to as the “generalization gap.”

### Good fit

1. The plot of training loss decreases to a point of stability.
2. The plot of validation loss decreases to a point of stability and has a small gap with the training loss.

.\*. Continued training of a good fit will likely lead to an overfit.

## Unrepresentative Dataset

### Unrepresentative Training Dataset

* 1. An unrepresentative training dataset means that the training dataset does not provide sufficient information to learn the problem, relative to the validation dataset used to evaluate it.
  2. This may occur if the training dataset has too few examples as compared to the validation dataset.

This situation can be identified by:

* A learning curve for training loss that shows improvement, and
* Similarly, a learning curve for validation loss that shows improvement, but a large gap remains between both curves.

### Unrepresentative Validation Dataset.

* 1. An unrepresentative validation dataset means that the validation dataset does not provide sufficient information to evaluate the ability of the model to generalize.
  2. This may occur if the validation dataset has too few examples as compared to the training dataset.

This situation can be identified by:

* A learning curve for training loss that looks like a good fit (or other fits), and
* A learning curve for validation loss that shows noisy movements around the training loss.

It may also be identified by:

* A validation loss that is lower than the training loss.
* In this case, it indicates that the validation dataset may be easier for the model to predict than the training dataset.

# Quotes

A deep learning neural network learns to map a set of inputs to a set of outputs from training data.

* We cannot calculate the perfect weights for a neural network; there are too many unknowns. Instead, the problem of learning is cast as a search or optimization problem and an algorithm is used to navigate the space of possible sets of weights the model may use to make good or good enough predictions.

The “gradient” in gradient descent refers to gradient of error function.

* The model with a given set of weights is used to make predictions and the error for those predictions is calculated.

Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions; we therefore present the two central approaches to statistics: frequentist estimators and Bayesian inference.

* Page 98, Deep Learning, Ian Goodfellow.

Most deep learning algorithms are based on an optimization algorithm called stochastic gradient descent.

* Page 98, Deep Learning, Ian Goodfellow.

Usually, this performance measure P is specific to the task T being carried out by the system.

* Page 103, Deep Learning, Ian Goodfellow.

A dataset can be described in many ways. In all cases, a dataset is a collection of examples, which are in turn collections of features.

* Page 106, Deep Learning, Ian Goodfellow.

Section 9.7 and chapter 10 describe how to handle different types of such heterogeneous data.

* Page 106, Deep Learning, Ian Goodfellow.

What separates machine learning from optimization is that we want the generalization error, also called the test error, to be low as well. The generalization error is defined as the expected value of the error on a new input.

* Here the expectation is taken across different possible inputs, drawn from the distribution of inputs we expect the system to encounter in practice.
* Page 110, Deep Learning, Ian Goodfellow.

The train and test data are generated by a probability distribution over datasets called the data generating process. We typically make a set of assumptions known collectively as the i.i.d. assumptions. These assumptions are that the examples in each dataset are independent from each other, and that the train set, and test set are identically distributed, drawn from the same probability distribution as each other.

* Page 111, Deep Learning, Ian Goodfellow.

One immediate connection we can observe between the training and test error is that the expected training error of a randomly selected model is equal to the expected test error of that model.

* Page 111, Deep Learning, Ian Goodfellow.

y' = w1.x + b:

* mapping from parameters to prediction is still a linear function.
* mapping from features to prediction is now an affine function.
  + - Page 109, Deep Learning, Ian Goodfellow.

y' = w1.x + w2.x^2 + b:

* prediction is still a linear function of parameters.
* mapping from features to prediction is now a quadratic function.
  + - Page 112, Deep Learning, Ian Goodfellow.

We can control whether a model is more likely to overfit or underfit by altering its capacity.

* Page 111, Deep Learning, Ian Goodfellow.

The error incurred by an oracle making predictions from the true distribution p (x, y) is called the Bayes error.

* Page 116, Deep Learning, Ian Goodfellow.

# Effect of training set size [2]

1. For a model with fixed capacity (i.e., quadratic model),
   1. The training error increases with increase in the training set size. This is because larger datasets are hard to fit. The training error must rise to the Bayes error, at the least.
   2. Simultaneously, the test error decreases.
   3. The quadratic model does not have enough capacity to solve the task, so its test error asymptotes to a high value.
2. For a model with optimal capacity,
   1. The test error asymptotes to the Bayes error.
      1. Bayes error: The error incurred by an oracle making predictions from the true distribution p (x, y) is called the Bayes error [1].
   2. The training error can fall below the Bayes error, due to the ability of the training algorithm to memorize specific instances of the training set.
3. As the training set increases, the optimal capacity of the model should increase.

# No Free Lunch theorem [3]

* In other words, in some sense, no machine learning algorithm is universally any better than any other. The most sophisticated algorithm we can conceive of has the same average performance (over all possible tasks) as merely predicting that every point belongs to the same class.
* Fortunately, these results hold only when we average over all possible data generating distributions. If we make assumptions about the kinds of probability distributions we encounter in real-world applications, then we can design learning algorithms that perform well on these distributions.
  + No Free Lunch theorem implies that we must design machine learning algorithms to perform well on a specific task. We do so by building set of preferences into the learning algorithm.

# Regularization [4]

* *Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error [5].*
* In our weight decay example, we expressed our preference for linear functions defined with smaller weights explicitly, via an extra term in the criterion we minimize.
* There are many other ways of expressing preferences for different solutions, both implicitly and explicitly. Together these different approaches are known as regularization.

# Hyperparameter [6]

* A setting is chosen as hyperparameter for two reasons: -
  + Learning algorithm does not learn because it is difficult to optimize.
  + More frequently, the setting must be a hyperparameter because it is not appropriate to learn the setting on training set e.g., if the degree of the polynomial is to be learnt on the training set, then of course the maximum possible value will be chosen to fit all the training data. Thus, increasing the capacity and overfitting the model to the training data.

# Validation set [7]

* Held-out test set, composed of examples coming from the same distribution as the training set, can be used to estimate the generalization error of a learner, after the learning process is completed.
* However, it is important that the test examples are not used in any way to make choices about the model, including its hyperparameters.
* Validation set, a subset of training dataset, allows for updating the hyperparameters i.e., validation set is primarily used to ‘train’ the hyperparameters.

# Probability

Text

Description automatically generated

Text

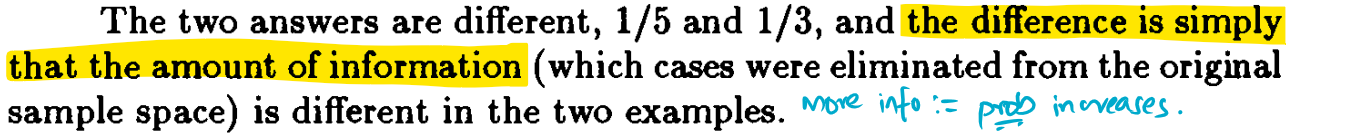
Description automatically generated

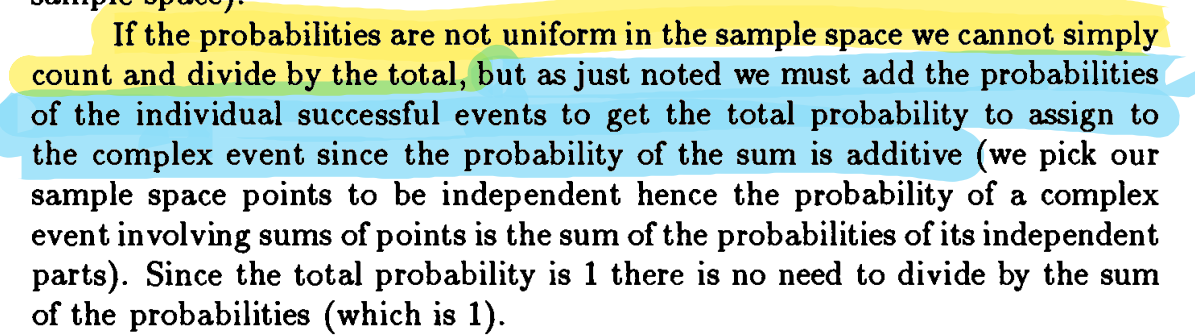
Text

Description automatically generated

Text

Description automatically generated





Text, letter

Description automatically generated

Graphical user interface, text, application

Description automatically generated

A picture containing timeline

Description automatically generated

# Cross Entropy [8]

* Many authors use the term ‘cross-entropy’ to identify specifically the negative log-likelihood of a Bernoulli or SoftMax distribution, but this is a misnomer. Any loss consisting of a negative log-likelihood is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by model.
* In software, we often phrase both (Maximum likelihood and KL Divergence [9]) as minimizing a cost function. Maximum likelihood thus becomes minimization of the negative log-likelihood (NLL), or equivalently, minimization of the cross entropy. The perspective of maximum likelihood as minimum KL divergence becomes helpful in this case because the KL divergence has a known minimum value of zero [9]. The negative log-likelihood can become negative when x is real-valued.

## Properties of Maximum Likelihood [10]

* Maximum Likelihood estimator has the property of consistency i.e., as the number of training examples approaches infinity, the maximum likelihood estimate of a parameter converges to the true value of the parameter.
* Cramér-Rao lower bound shows that no consistent estimator has a lower mean squared error than the maximum likelihood estimator, as ‘m’ increases and for larger value of ‘m’.

## Bayesian Statistics [11]

* Typically, the priori begins as a uniform distribution or as a Gaussian distribution with high entropy, and the observation of the data causes the posterior to lose entropy and concentrate around few highly likely values of the parameters, unlike the point based estimate that gives only one value for a parameter.
* However, critics of Bayesian approach identify the priori as a source of subjective human judgement that impacts the predictions.
* Bayesian methods typically generalize much better when the available training data is limited, but typically suffer from high computational cost when the number of training examples is large.

Training a neural network solves an optimization problem [12].

## Choosing a loss function [13]

* This is a challenging problem. Because the function must capture the properties of the problem and be motivated by concerns that are important to project and the stakeholders.
* *It is important, therefore, that the function faithfully represent our design goals. If we choose a poor error function and obtain unsatisfactory results, the fault is ours for badly specifying the goal of the search.*

*— Page 155, Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, 1999.*

## Denisity estimation [14]

* There are three approaches:
  + Parametric based:
    - A specific functional form of the density model is assumed.
    - This function contains number of parameters which are then optimized by fitting the model to the dataset.
    - **Drawback**: The chosen functional form might not be capable of providing a good representation of the true density.
  + Non-parametric based:
    - No assumption on any functional forms.
    - The functional form of the density is entirely determined by the data.
    - **Drawback**: The number of parameters in the model grows with the size of the data set, and the model becomes unwieldly – difficult to carry.
  + Semi-parametric based:
    - Best of both the worlds.
    - E.g., mixture of Gaussian distributions.

# Cross Entropy [15]

* The error between two probability distributions is measured using cross-entropy.
* Cross entropy can be thought to calculate the total entropy between the two distributions [16].
* Cross entropy is the average number of bits needed to encode data coming from source with distribution p when we use model q [17].
* Graphical user interface, text, application

  Description automatically generated
* KL divergence = - H(p) + H(p, q) [17].
  + H(p) = H(p, p) := is the exact number of bits needed to encode data coming from true distribution.
  + H(p, q) := is the average number of bits needed to encode data coming from distribution p modeld using distribution q. This has to be greater than H(p).
  + Therefore, H(p, q) – H(p) gives the additional number of bits needed when ‘q’ is used as a model instead of ‘p’.
  + KL divergence is often called as *relative entropy.*
  + KL divergence is asymmetric.
  + Cross entropy is asymmetric i.e., H(p, q) != H(q, p)

**Cross entropy** is a measure of similarity between two distributions. Since the classification models used in deep learning typically output probabilities for each class, we  
can compare the true class (distribution *p*) with the probabilities of each class given  
by the model (distribution *q*). The more similar the two distributions, the smaller our  
cross entropy will be [26]. The minimum value of cross entropy can be entropy itself i.e., H(p,p).

L1 and L2 regularization penalties do not work well in deep neural networks – use dropouts [18].

* Question: How do I compute loss at the output layer and propagate it back to the layers? – See backpropagation.

## L1 and L2 losses [18]

1. Typically considered as purely regressive losses, which should not be used in classification.
   1. L1 is often used as auxilliary loss in deep nets to ensure sparsness of representations.
   2. L2 is sometimes applied to weights – to prevent them from growing to infinity.
   3. L1 and L2 losses might lead to slower training.
2. However, despite their regression roots – they still have reasonable probabilistic interpretation for classification.
3. L1 loss applied to probability estimates leads to minimization of expected misclassification probability. Whereas, ‘log loss’ leads to maximization of fully correct labelling.
4. L1 and L2 losses applied to probabilistic estimates coming from sigmoid (or softmax) have non-monotonic partial derivaties wrt output of the final layer. Finally, they vanish in both infinities, which slows down the learning of heavily misclassified examples.

## Which loss function leads to fast convergence [18]?

Faster and better convergence can be achieved when the architecture together with the loss function produces a piecewise linear partial derivatives (but not constant) wrt final layer activations.

* A well known deep learning hypothesis is that the deep models learn well when dealing with piecewise linear functions.
  + An interesting phenomenon in classification based on neural networks is – even in deep linear model or rectifier network the top layer is often non-linear, as it uses softmax or sigmoid activation to produce probability estimates. Once this is introduced, even the partial derivatives stop being piecewise linear.

# Loss functions:

1. ‘hinge square’ and ‘hinge cube’ losses are consistently the fastest in training.
   1. With enough hidden layers, even the L2 loss is fast in training.
2. Margin based loss systems seem to outperform all other losses in terms of generalization capabilities.
3. L1 and L2 regularization penalties do not work well in deep networks.
4. Cauchy-Schwarz Divergence as the optimisation criterion seems to be a consistently better choice than log loss, but is very rarely used in the deep learning community.
5. Log loss is still the unquestionable favorite.
6. Intuitions:
   1. Linear models rarely transfer to highly non-linear deep networks.
      1. Observations and conclusions drawn from the linear model (no hidden layers) do not seem to transfer to deep nets.
   2. Depending on the application, losses other than ‘log loss’ are preferable.
      1. For accuracy focussed research, squared hinge loss seems to be a better choice – it converges faster and better performance.
   3. For highly noised dataset, expectation loss seems to be the better choice.

# Losses [22]

## Mean Squared Loss

* 1. Default loss for regression problems.
  2. Recommend that output layer has one node for target variable and the linear activation function.

## Mean Squared Logarithmic Loss

* 1. Has the effect of relaxing the punishing effect of large differences in large predicted values.

Binary classification problems is framed to predict a value of 0 or 1 for the first or second class and is often implemented as predicting the probability of the example belonging to class 1.

## Binary Cross-Entropy Loss

* 1. Default loss for binary classification problems.
  2. Perfect cross-entropy is 0.
  3. Keras’ binary cross-entropy function requires that the output layer is configured to have single node and a ‘sigmoid’ activation function to predict the probability of class 1.

## Hinge Loss

* 1. Alternative to binary cross-entropy loss – developed for use with Support Vector Machines.
  2. Target values are in the set {-1, 1}.
  3. Hinge loss function encourages the examples to have correct sign – assigning more error when there is a difference in the sign between the actual and predicted values.
  4. Final output layer is configured to have a single node and the hyperbolic tangent activation function to output values in the range of [-1, 1].

## Squared Hinge Loss

* 1. Has the effect of smoothening the surface of the error function and making it numerically easier to work with.
  2. Small changes to weights are causing large changes in loss.

Multi-Class classification – examples are assigned with one or more than two classes.

## Multi-Class Cross-Entropy Loss

* 1. Cross-entropy loss is the default loss for multi-class classification problems.
  2. Final output layer is configured to have n-nodes and ‘softmax’ activation function to predict the probability for each class.

## Sparse Multi-Class Cross-Entropy Loss

* 1. Target variable need not be one hot encoded prior to training.
  2. In certain applications, one hot encoded means tremondous memory would be needed.

## Kullback Leibler Divergence Loss

* 1. Is more commonly used when using models that learn to approximate a more complex function than simply multi-class classification.
  2. Nevertheless, it can be used for multi-class classification – in that case it is functionally equivalent to multi-class cross-entropy.

# Batch Normalization in CNN [19]

Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data. It is done along mini-batches instead of the full data set. It serves to speed up training and use higher learning rates, making learning easier.

In the following image, we can see a regular feed-forward Neural Network: x\_i are the inputs, z the output of the neurons, a the output of the activation functions, and y the output of the network:

Diagram

Description automatically generated

Batch Norm – in the image represented with a red line – is applied to the neurons’ output just before applying the activation function. Usually, a neuron without Batch Norm would be computed as follows:



being g() the linear transformation of the neuron, w the weights of the neuron, b the bias of the neurons, and f() the activation function. The model learns the parameters w and b. Adding Batch Norm, it looks as:

A picture containing diagram

Description automatically generated

being zN the output of Batch Norm, mz the mean of the neurons’ output, sz the standard deviation of the output of the neurons, and \gamma and \beta learning parameters of Batch Norm. Note that the bias of the neurons (b) is removed. This is because as we subtract the mean m\_z, any constant over the values of z – such as b – can be ignored as it will be subtracted by itself.

Normalization is the process of transforming the data to have zero mean and unit standard deviation [20].

In the above step, we have input for our hidden layers ‘z’ [20].

* We calculate the mean of this hidden activation :=

(sum.of.inputs.to.each.neuron/no.of.neurons.in.the.hidden.layer)

* Similarly, we calculate the standard deviation of the hidden activations.
* Normalization := (value - mean) / s.d

**The parameters \beta and \gamma shift the mean and standard deviation, respectively.** Thus, the outputs of Batch Norm over a layer results in a distribution with a mean \beta and a standard deviation of \gamma. These values are learned over epochs and the other learning parameters, such as the weights of the neurons, aiming to decrease the loss of the model.

## Why does Batch Normalization Work? [19]

1. Normalizing the inputs to take on a similar range of values can speed up learning – Batch normalization is doing this for all the layers’ inputs in the network.
2. According to the original paper [21], Batch Normalization reduces the internal covariate shift of the network.
   1. Covariate shift: is change in the distribution.
   2. Internal covariate shift: is change in the input distribution of the internal layers of a neural network.
      1. For neurons in an internal layer, the inputs received from the previous layer are constantly changing. This is due to the multiple computations done before it and the weights over the training data.
3. Applying Batch Normalization ensures that all the layers have a distribution with same mean (beta) and same standard deviation (gamma), and these will always be the same.
4. Thus, the amount of change in distribution is reduced and the internal covariate shift is avoided.
5. Lastly, Batch Normalization has the effect of regularization.
   1. Since Batch Normalization is done over mini batches, each time the model sees the data distribution it sees noise.
   2. This can act as a regualizer and avoid overfitting to the data.
   3. However, the noise is small and is not enough to act as a proper regulaizer thus it is always accompanied with dropouts.

## Advantages of Batch Normalization:

1. Higher learning rates [21].
2. Can be less careful with initializations [21].

# Machine Learning Practices and Advices

1. In fact, using the train/test method of estimating the skill of the procedure on unseen data often has a high variance (unless we have a heck of a lot of data to split). This means that when it is repeated, it gives different results, often very different results [23].
2. Cross-validation systematically creates and evaluates multiple models on multiple subsets of the dataset.
   * 1. We can calculate the mean of these measures to get an idea of how well the procedure performs on average.
     2. We can calculate the standard deviation of these measures to get an idea of how much the skill of the procedure is expected to vary in practice.
   1. This is also helpful for providing a more nuanced comparison of one procedure to another when you are trying to choose which algorithm and data preparation procedures to use.

## Data Leakage

1. Data leakage is a big problem in machine learning when developing predictive models [24].
   1. Data leakage is when information from outside the training dataset is used to create the model [24].
2. A leading text in the field called data leakage as one of the top ten machine learning mistakes [25].
3. For example, if you normalize or standardize your entire dataset, then estimate the performance of your model using cross validation, you have committed the sin of data leakage [24].
   1. The data rescaling process that you performed had knowledge of the full distribution of data in the training dataset when calculating the scaling factors (like min and max or mean and standard deviation). This knowledge was stamped into the rescaled values and exploited by all algorithms in your cross validation test harness.
   2. A non-leaky evaluation of machine learning algorithms in this situation would calculate the parameters for rescaling data within each fold of the cross validation and use those parameters to prepare the data on the held out test fold on each cycle.

# Gradient Descent: Optimizer[26]

1. The gradient descent algorithms work well on highly complicated network architectures and therefore are suitable for a wide variety of problems.
2. More specifically, recent advances make it possible to compute these gradients by utilizing massively parallel systems, so the approach scales well with dimensionality.
3. While convergence to the global minimum is guaranteed for convex functions, for nonconvex problems (which are essentially all problems in the world of deep learning) they can get stuck in local minima. In practice, this is often good enough, as is evidenced by the huge success of the field of deep learning.

## Sampling methods for Gradien Descent[26]

The gradient of the objective is computed with respect to the model parameters and evaluated using a given set of input samples. How many of the samples should we take for this calculation? Intuitively, it makes sense to calculate the gradient for the entire set of samples in order to benefit from the maximum amount of available information. This method, however, has some shortcomings. For example,

1. It can be very slow and,
2. It is intractable when the dataset requires more memory than is available.

A more popular technique is the **stochastic gradient descent (SGD)**, where instead of feeding the entire dataset to the algorithm for the computation of each step, a subset of the data is sampled sequentially. The number of samples ranges from one sample at a time to a few hundred, but the most common sizes are between around 50 to around 500 (usually referred to as mini-batches).

1. Using smaller batches usually works faster, and
2. The smaller the size of the batch, the faster are the calculations.

However, there is a trade-off in that

1. Small samples lead to lower hardware utilization and,
2. Tend to have high variance, causing large fluctuations to the objective function.

Nevertheless, it turns out that some fluctuations are beneficial since they enable the set of parameters to jump to new and potentially better local minima.

Using a relatively smaller batch size is therefore effective in that regard, and is currently overall the preferred approach.

Text, letter

Description automatically generated

* If the weights are multidimensional then partial derivative has to be taken on each dimension and update has to be done on each of them.
* If ‘m’ is the entire dataset, then it is Batch Gradient Descent.
* If ‘m’ is a small number, then it is Stochastic Gradient Descent with mini batches.
* If ‘m’ is 1, then it is ………………………………………………………………………………………………

# Convolution Neural Networks [26]

1. The Nobel Prize–winning neurophysiologists Hubel and Wiesel discovered as early as the 1960s that the first stages of visual processing in the brain consist of application of the same local filter (e.g., edge detectors) to all parts of the visual field.
2. The current understanding in the neuroscientific community is that as visual processing proceeds, information is integrated from increasingly wider parts of the input, and this is done hierarchically.
3. Convolution neural networs follows the same pattern. Each convolution layer looks at increasingly larger part of the image as we go deeper into the network.
   * This is because of the increasing receptive fields.

## Convolution Neural Networks Models [26]

* 1. In CNN models, the convolution layers are stacked up hierarchically, and feature maps are simply the output of each of these layers.
  2. Feature maps can also be called as *processed images* – the result of applying the filters on the input image.
  3. Convolution layers or fully connected layers are regarded as linear layers. It is common practice to apply nonlinear activation functions to them.
     1. Because, consecutive linear operations can be replaced by a single operation, and thus the depth of the network doesn’t contribute to the expressiveness of the model unless we introduce nonlinearity between the linear layers.
  4. It is common to follow convolution layers with pooling of outputs. Pooling means reducing the size of the data with some local aggregation function, within each feature map. It is done for two reasons:
     1. Technical reason: reduces the size of the data to be processed downstream, especially important when the convolution layers are followed by fully connected layers.
     2. Theoretical reason: Allows the model to overcome spatial variability – for a model trying to find eye, the position does not matter if it is at the center of the image or slightly moved by two pixels.
  5. Dropout: regularization effect, by forcing the network to distribute the learned representation across all the neurons and not specific to few neurons.
     1. This is used only during training and not during testing.
  6. Weight initializations that generally produces good results are with random normal distribution with standard deviation 0.1 that is truncated at the tails.
     1. A bad initialization can make the training process “get stuck,” or fail completely due to numerical issues.
     2. Using random initialization instead of constant initializations will help break the symmetry between learned features – allowing the model to learn a diverse and rich representations.
  7. Major source of power in CNN models is the utilization of the spatial meanings when considering images.

# Abstraction libraries for Tensor flow [26 – Chapter 7]

More than a few great TensorFlow open-source extensions were available at the time of writing the book. Popular ones among them are:

* tf.contrib.learn
* TF-Slim
* TFLearn
* Keras
* While **contrib.learn** is more transparent and low-level, the other three extensions are a bit cleaner and more abstract, and each has its own specialties and little advantages that might come in handy depending on the needs of the user.
* TFLearn and Keras are full of functionality and have many of the elements needed for various types of state-of-the-art modeling. Unlike all the other libraries, which were created to communicate solely with TensorFlow, Keras supports both **TensorFlow** and **Theano** (a popular library for deep learning).
* TF-Slim was created mainly for designing complex convolutional nets with ease and has a wide variety of pretrained models available, relieving us from the expensive process of having to train them ourselves.

# Types of machine learning [27].

1. Supervised and unsupervised learning.
2. Batch and online learning.
3. Instance based versus model based learning.

## Batch on online learning:

### Batch learning:

* + In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data.
  + This will generally take a lot of time and computing resources, so it is typically done offline.
  + First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning.

### Online learning:

* + In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini batches.
  + Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives.
  + Online learning is of great use for systems that receives data as continuous flow e.g., stock prices and not to adapt to change rapidly or autonomously.

## Instance based learning and Model based learning:

Is another way of categorizing machine learning algorithms based on *how they generalize.*

### Instance based learning:

* + System learns the examples by heart, then generalizes the new cases using a similarity measure. E.g., k-nearest neighbors.

### Model based learning:

* + Build a model from the examples, and then use the model to make predictions. E.g., linear regression model or nonlinear regression model.

# Main challenges of machine learning [27].

1. Insufficient quantity of training data.
2. Nonrepresentative training data.
3. Poor-quality data.
4. Irrelevant features.
5. Overfitting of the training data.
6. Underfitting of the training data.

## Irrelevant features:

* As the saying goes: garbage in, garbage out. Your system will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones.
* A critical part of the success of a Machine Learning project is coming up with a good set of features to train on. This process, called feature engineering, involves:
  + Feature selection: selecting most useful features among the existing ones.
  + Feature extraction: combining existing features to produce more useful one.
  + Creating new features by gathering new data i.e., not just the population, but also the number of immigrants. Thus, we have two features now.

# Performance measure [27]

## Accuracy

Accuracy is not the preferred performance measure for classifiers, especially when the datasets are skewed e.g., when only 10% of the images are 5s, so if you always guess the image is *not a* 5, you will be right about 90% of the times.

## Confusion matrix

## Precision and Recall

### Precision Recall tradeoff

### Precision Recall curve

Text, letter

Description automatically generated

# Dimensionality Reduction [27]

Many machine learning problems involves thousands or millions of features for each training instance. This makes training extremely slow and [can also make it much harder to find a good solution](#Curse_of_Dimensionality). This problem is often referred to as *curse of dimensionality*.

1. The border pixels in MNSIT dataset are all white, so cropping them out from the training set would not result in losing information.
2. The two neighboring pixels are often highly correlated, so merging them into a single pixel will not result in losing information.

## Curse of dimensionality

However, reducing dimensionality does lose some information (just like compressing an image to JPEG can degrade its quality). So even though it speeds up the training, it may also make your system perform slightly worse.

Therefore, advice is to first train the system with original data before considering dimensionality reduction. In some cases, reducing dimensionality filters out some noise and unnecessary details and thus result in higher performance. But in general, it won’t; only speeds up the training.

***Higher dimensions make it harder to find a good solution:***

*Here is a more troublesome difference: if you pick two points randomly in a unit square, the distance between these two points will be, on average, roughly 0.52. If you pick two random points in a unit 3D cube, the average distance will be roughly 0.66.*

*But what about two points picked randomly in a 1,000,000-dimensional hypercube? Well, the average distance, believe it or not, will be about 408.25 (roughly 1, 000, 000/6)! This is quite counterintuitive: how can two points be so far apart when they both lie within the same unit hypercube?*

*This fact implies that high dimensional datasets are at risk of being very sparse: most training instances are likely to be far away from each other. Of course, this also means that a new instance will likely be far away from any training instance, making predictions much less reliable than in lower dimensions, since they will be based on much larger extrapolations. In short, the more dimensions the training set has, the greater the risk of overfitting it.*

## Solution to the curse

Increase the size of the training set to reach a sufficient density of training instances (to counter the sparsity of data in higher dimensional space). Unfortunately, in practice, the number of training instances needed to reach a given density grows exponentially with number of dimensions.

* With just 100 features (i.e., 100 dimensions), the number of training instances needed is more than the atoms in the observable universe for each training instances to be within 0.1 of each other on average, assuming they are spread uniformly across all dimensions [27, page no 207].

## Manifold Learning

Many dimensionality reduction algorithms work by modeling the *manifold* on which the training instances lie; this is called Manifold Learning. It relies on *manifold assumptions,* also called the *manifold hypothesis*, which holds that most real-world high-dimensional datasets lie close to a much lower-dimensional manifold. This assumption is very often empirically observed.

### Manifold

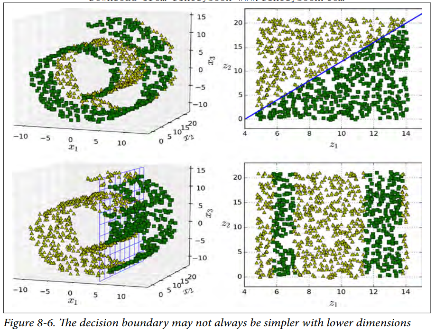
A picture containing food

Description automatically generated2D manifold is a 2D shape that can be bent and twisted in a higher dimensional space. E.g., Swiss roll i.e., it locally resembles a 2D plane, but it is rolled in the third dimension.

More generally, a *d*-dimensional manifold is part of an *n­-*dimensional space (where *d < n*)that locally resembles a *d*-dimensional hyperplane.

### Manifold Assumption [27, page 210]

The most real-world high-dimensional datasets lie close to a much lower-dimensional manifold. This assumption is very often empirically observed.

This assumption is also accompanied by an implicit assumption: that the task at hand will be simpler if expressed in the lower-dimensional space of the manifold. However, the Figure 8.6, shows that it is not the case always.

In short, if you reduce the dimensionality of your training set before training a model, it will speed up the training, but it may not always lead to a better solution; it all depends on the dataset.

## PCA – Principal Component Analysis

Is by far the most popular dimensionality reduction algorithm. First it identifies the hyperplane that lies closest to the data, and then projects the data onto it.

### Preserving the Variance.

Text

Description automatically generatedThe axis of the hyperplane onto which the datapoints are projected are chosen such that the projected datapoints preserves the maximum amount of variance, as it will likely lose less information than the other projections. These axes are eigen axes chose in the increasing value of their eigen values associated with them.

### Principal Components.

The unit vectors that define the hyperplane are called *principal components*. The unit vector that defines the ith axis is called the ith *principal component.* These *principal components* are obtained from Singular Value Decomposition (SVD) that can decompose the training set matrix **X** into the dot product of three matrices **U.D.VT**, where **VT** contains all the principal components that we are looking for.

### Explained Variance Ratio.

It indicates the proportion of the dataset’s variance that lies along the axis of each principal component. E.g., the first principal component may amount to 84% of the variance, and the second principal component may amount to 14% of the variance. Therefore, with just two axes, we have amounted to 98% of the dataset’s variance, which is good.

### Choosing right number of dimensions.

Chart

Description automatically generatedInstead of randomly choosing the number of dimensions to reduce to, it is generally preferable to choose the number of dimensions that add up to a sufficiently large portion of the variance (e.g., 95%).

Plot the explained variance against the number of dimensions. There will usually be an elbow in the curve, where the explained variance stops growing fast. This point, you can think that this dimension is the intrinsic dimensionality of the dataset.

### Incremental PCA

PCA requires all the datapoints (the entire training set) to fit in the memory for the SVD algorithm to run. This might not be possible and fortunately, we have *Incremental PCA (IPCA).*

## Other Dimensionality Reduction Techniques [27, page 223]

1. Multidimensional Scaling (MDS)
2. Isomap
3. t-Distributed Stochastic Neighbor Embedding (t-SNE)
4. Linear Discriminant Analysis (LDA)
5. Local Linear Embedding (LLE)

# TensorFlow Practice Tips

## Loading datasets:

* 1. Use tf.data.Dataset.cache to easily cache calculations across epochs. This is very efficient, especially when the data fits in memory (Google Colab, 2022).
  2. If the data does not fit the memory, then use a cache file (Google Colab, 2022).
  3. TFRecord File:
     1. TFRecord files are a simple format to store a sequence of binary blobs. By packing multiple examples into the same file, TensorFlow is able to read multiple examples at once, which is especially important for performance when using a remote storage service such as GCS (Google Colab, 2022).
  4. Serialized Tensor (Google Colab, 2022):

## tf.data.Dataset.from\_tensor\_slices()

Graphical user interface, text, application, email

Description automatically generated

* 1. features and labels arrays are stored in as tf.constant() operations, this is bad because it wastes memory by creating multiple copies (TensorFlow, 2022).

## Output shape

* 1. The output\_shapes argument is not required but is highly recommended as many TensorFlow operations do not support tensors with an unknown rank. If the length of a particular axis is unknown or variable, set it as None in the output\_shapes.

# References

*Google Colab*. (2022, June 08). Retrieved from https://colab.research.google.com/github/tfindiamooc/tfindiamooc.github.io/blob/master/colabs/images.ipynb#scrollTo=lV1NOn2zE2lR

*TensorFlow*. (2022, June 10). Retrieved from https://www.tensorflow.org/guide/data