## Exercise 1

## Introduction data set

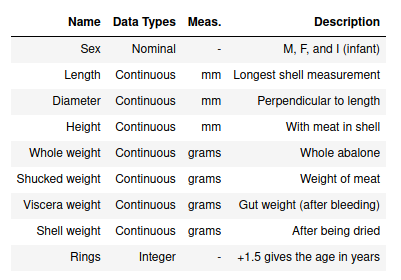
The dataset: predict Abalone age

Abalone is a mollusc with a peculiar ear-shaped shell lined of mother of pearl. Its age can be estimated counting the number of rings in their shell with a microscope, but it is a time consuming process, we will use Machine Learning to predict the age using physical measurements.

Picture of the shell of an Abalone:



The dataset is retrieved from http://archive.ics.uci.edu/ml/datasets/Abalone. Given is the attribute name, attribute type, the measurement unit and a brief description. The number of rings is the value to predict.



## Classification model

* **K-nearest neighbors algorithm**

The k-nearest neighbors classifier (kNN) is a non-parametric supervised machine learning algorithm. It’s distance-based: it classifies objects based on their proximate neighbors’ classes. kNN is most often used for classification, but can be applied to regression problems as well.

The parameter k in kNN refers to the number of labeled points (neighbors) considered for classification. The value of k indicates the number of these points used to determine the result. Our task is to calculate the distance and identify which categories are closest to our unknown entity.

How it work: The main concept behind k-nearest neighbors is as follows. Given a point whose class we do not know, we can try to understand which points in our feature space are closest to it. These points are the k-nearest neighbors. Since similar things occupy similar places in feature space, it’s very likely that the point belongs to the same class as its neighbors. Based on that, it’s possible to classify a new point as belonging to one class or another.

A simple implementation of KNN classification is a majority voting mechanism.

* Given: training examples D={xi , yi }, and a new example x where
* xi : attribute-value representation of the example ith • yi : corresponding label or class of example ith
* Algorithm:
* Compute distance D(x,xj ) for every xj of the training data D
* Select k closest instances xi1, ..., xik with their labels are yi1,..yik
* y = majority (yi1,..yik) is the predicted label of x

Summary

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It’s easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows. KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression). In the case of classification and regression, we saw that choosing the right K for our data is done by trying several Ks and picking the one that works best.

* **Decision Trees**

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.



**How does the Decision Tree algorithm Work?**

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

* **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
* **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
* **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
* **Step-4:** Generate the decision tree node, which contains the best attribute.
* **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node

Sumary

## Advantages of CART

* Simple to understand, interpret, visualize.
* Decision trees implicitly perform variable screening or feature selection.
* Can handle both numerical and categorical data. Can also handle multi-output problems.
* Decision trees require relatively little effort from users for data preparation.
* Nonlinear relationships between parameters do not affect tree performance.

## **Disadvantages of CART**

* Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting.
* Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called  variance, which needs to be lowered by methods like bagging and boosting.
* Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
* Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.
* **Support Vector Machine**

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



## Performance Measures

**Accuracy** - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model. For our model, we have got 0.803 which means our model is approx. 80% accurate.

Accuracy = TP+TN/TP+FP+FN+TN

**Precision** - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Precision = TP/TP+FP

**Recall**(Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? We have got recall of 0.631 which is good for this model as it’s above 0.5.

Recall = TP/TP+FN

**F1 score** - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall. In our case, F1 score is 0.701.

F1 Score = 2\*(Recall \* Precision) / (Recall + Precision)

## Exercise 2

* 1. **What is Correlation**

Variables within a dataset can be related for lots of reasons.

For example:

* One variable could cause or depend on the values of another variable.
* One variable could be lightly associated with another variable.
* Two variables could depend on a third unknown variable.

It can be useful in data analysis and modeling to better understand the relationships between variables. The statistical relationship between two variables is referred to as their correlation.

A correlation could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable’s value increases, the other variables’ values decrease. Correlation can also be neutral or zero, meaning that the variables are unrelated.

* **Positive Correlation**: both variables change in the same direction.
* **Neutral Correlation**: No relationship in the change of the variables.
* **Negative Correlation**: variables change in opposite directions.

The performance of some algorithms can deteriorate if two or more variables are tightly related, called multicollinearity. An example is linear regression, where one of the offending correlated variables should be removed in order to improve the skill of the model.

We may also be interested in the correlation between input variables with the output variable in order provide insight into which variables may or may not be relevant as input for developing a model.

The structure of the relationship may be known, e.g. it may be linear, or we may have no idea whether a relationship exists between two variables or what structure it may take. Depending what is known about the relationship and the distribution of the variables, different correlation scores can be calculated.

The Pearson correlation coefficient (*r*) is the most widely used correlation coefficient and is known by many names:

* Pearson’s *r*
* Bivariate correlation
* Pearson product-moment correlation coefficient (PPMCC)
* The correlation coefficient

The Pearson correlation coefficient is a [descriptive statistic](https://www.scribbr.com/statistics/descriptive-statistics/), meaning that it summarizes the characteristics of a dataset. Specifically, it describes the strength and direction of the linear relationship between two quantitative variables.

Although interpretations of the relationship strength (also known as [effect size](https://www.scribbr.com/statistics/effect-size/)) vary between disciplines, the table below gives general rules of thumb:



The Pearson correlation coefficient is also an inferential static, meaning that it can be used to test statistical hypotheses. Specifically, we can test whether there is a significant relationship between two variables.

## Linear regresstion

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :



While training the model we are given :

x: input training data (univariate – one input variable(parameter))

y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.

θ1: intercept

θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

## Mean Absolute Error (MAE)

## Mean Absolute Error calculates the average difference between the calculated values and actual values. It is also known as scale-dependent accuracy as it calculates error in observations taken on the same scale. It is used as evaluation metrics for regression models in machine learning. It calculates errors between actual values and values predicted by the model. It is used to predict the accuracy of the machine learning model.

## Formula:

## Mean Absolute Error = (1/n) \* ∑|yi – xi|

## Where,

## Σ: Greek symbol for summation

## yi: Actual value for the ith observation

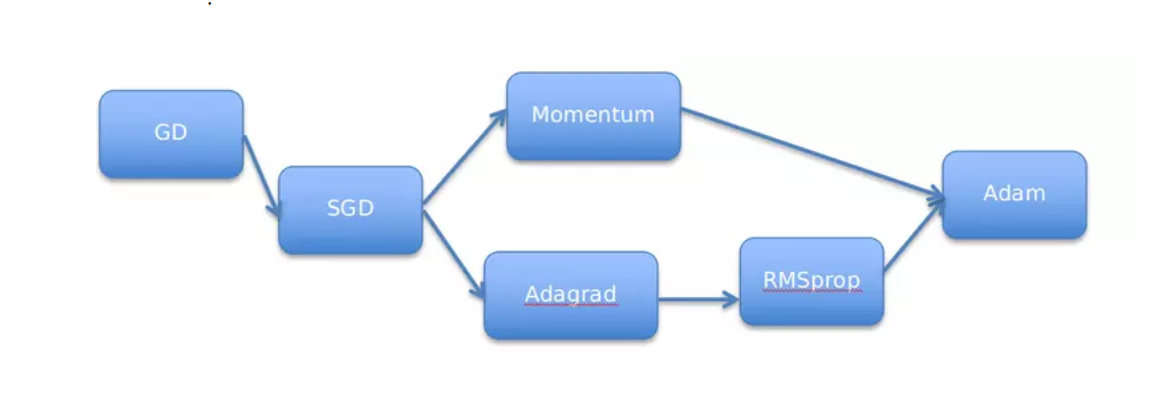
## xi: Calculated value for the ith observation

## n: Total number of observations

## III. Exercise 3:

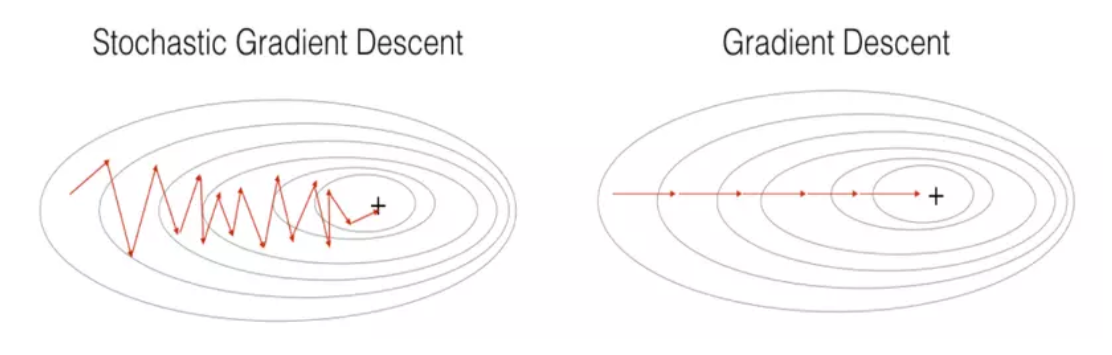
- The Stochastic Gradient Descent and Adam algorithms are both collectively known as the optimization algorithm.

- Here are pictures of some commonly used Optimizer algorithms:

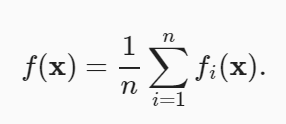
****

**3.1 Stochastic Gradient Descent Algorithm (SGD):**

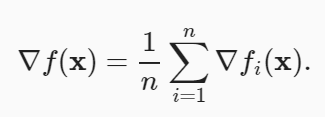
+ Description: Stochastic is a variation of Gradient Descent. Instead of after each epoch we will update the weight (Weight) once, in each epoch with N data points we will update the weight N times. Looking at it on the one hand, SGD will reduce the speed of 1 epoch. However, looking at it the other way, SGD will converge very quickly after only a few epochs. The SGD formula is similar to GD but works on each data point**.**



**-** In deep learning, the objective function is usually the average of the loss functions for each sample in the training set. Assume that the training set has n samples, fi(x) is the loss function of the i sample, and the parameter vector is x. We have the objective function:

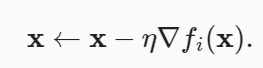


- The gradient of the objective function at x is calculated as follows:

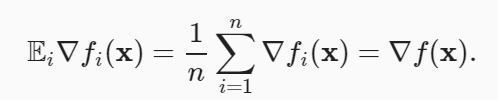


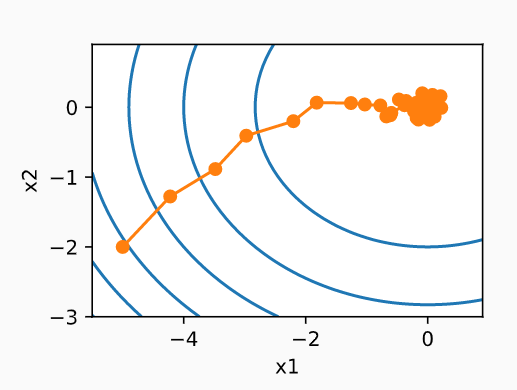
- If gradient descent is used, the computation cost for each independent loop is O(n), , increases linearly with n, Therefore, with a large training set, the cost of gradient descent for each iteration will be very high.

- Stochastic gradient descent (SGD) reduces the computational cost of each iteration. In each loop, we randomly take a data sample with index I ∈ {1,…,n} evenly distributed, and updated only x by gradient ∇fi(x) :



- Here, η is the learning rate. We can see that the computation cost per loop is reduced from O(n) of lowering the gradient to a constant O(1). Remember that gradients are random ∇fi(x) is an unbiased estimate of the gradient ∇f(x).

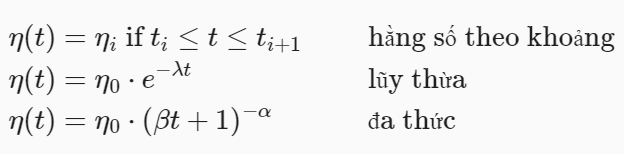




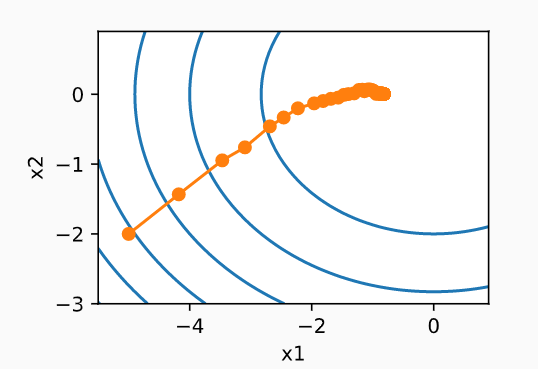
- As can be seen, the trajectories of the variables in the SGD fluctuate more strongly than the lower gradient in the previous section. This is due to the random nature of the gradient. That is, even when approaching the minimum value, we still encounter the uncertainty caused by the random gradient η∇fi(x). Even after 50 steps the quality is still not very good. Worse, it still won't improve with more steps (we encourage you to read the test with a larger number of steps to confirm this for yourself). I only have one choice left-change the learning rate η. However, if we choose too small a value, we will not make any significant progress in the first steps. On the other hand, if we choose too large a value, we will not get a good solution, as seen above. The only way to resolve these two conflicting goals is to dynamically reduce the learning rate during the optimization process.

This is also the reason for adding the learning rate lr function to the step function SGD. In the above example, the learning rate scheduling function is not enabled because we set the function lrwith() a constant, ie .lr = (lambda: 1)

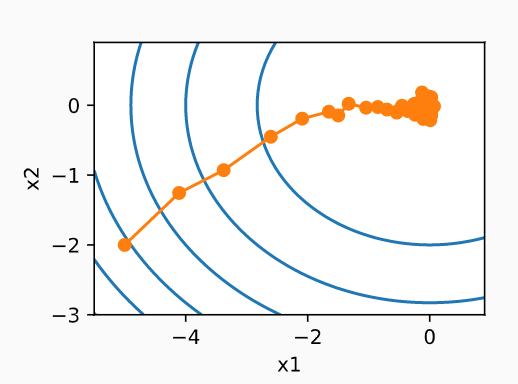
- Replacing η with a time-dependent learning rate η(t) will make controlling the convergence of the optimal algorithm more complicated. Specifically, we need to find a reasonable degree of reduction. If the reduction is too fast, the optimization will stop too soon. If the reduction is too slow, we will waste a lot of time on optimization. There are a few basic strategies used to adjust η over time (more advanced strategies will be discussed in the following chapter):



- In the first case, we reduce the learning rate whenever the optimization process stalls. This is a popular strategy for training deep networks. In addition, we can reduce the learning rate faster by exponential attenuation. Unfortunately, this approach leads to stopping the optimization too early before the algorithm converges. Another popular choice is polynomial reduction withα=0.5. In the case of convex optimization, there are proofs that this value gives good results. Let's see how it works in practice.



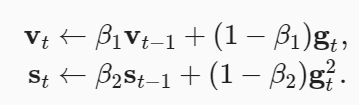
- As expected, the variance values ​​of the parameters decreased significantly. However, the exponential attenuation does not converge to the optimal solution X= (0,0). Even after 1000 iterations, the solution found is still very far from the optimal solution. In fact, this algorithm does not converge. On the other hand, if we use polynomial attenuation where the decaying learning rate is inversely proportional to the square root of time, the algorithm converges well.



**3.2 Adam Optimization Algorithm:**

Adam is the optimal algorithm that combines all the techniques like GD, SGD, momentum method, adagrad and RMSprop into one efficient learning algorithm. As expected, this is one of the powerful and efficient optimization algorithms commonly used in deep learning. However, it also has a few weaknesses.

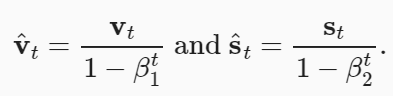
One of Adam's key components are exponential weighted moving averages (also known as leaky averages) for estimating both the momentum and the quadratic moment of the gradient. Specifically, it uses state variables

****

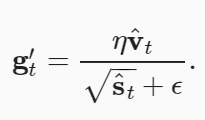
Here β1 and β2 are non-negative weight parameters. Popular choices for them areβ1=0.9and β2=0.999. This means that the variance estimate moves much slower than the momentum term. Note that if we initialize v0=s0=0, the algorithm will have a significant initial bias on smaller values. This problem can be solved using:



To normalize the terms. Similarly, the state variables are normalized as follows:



With the appropriate estimates, we can now write down the updated equations. First, we adjust the gradient value, similar to RMSProp to get



Unlike RMSProp, the update equation uses momentum  instead of gradients. Furthermore, there is a slight difference here: the transformation is performed using  instead of   In practice, the first one works slightly better, resulting in this difference compared to RMSProp. Usually, we choose ϵ= balance between numerical stability and reliability.

Now we will combine all of the above to calculate the update step. You may feel a little bit down because it's actually quite simple

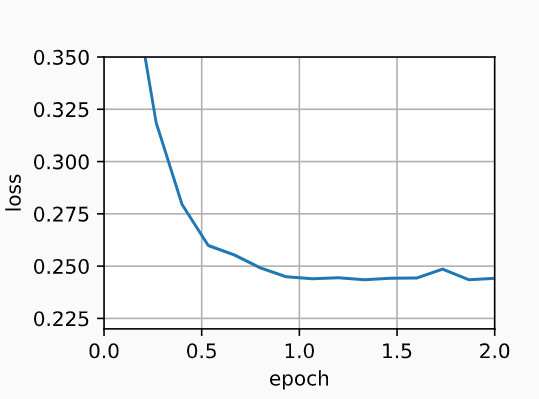


When looking at Adam's design, the inspiration behind the algorithm is clear. Momentum and range are clearly represented in the state variables. Their rather odd definition requires that we reduce the bias of the terms (which can be done by slightly tweaking the initialization and update conditions). Second, the combination of both terms is quite simple, based on RMSProp. Finally, the explicit learning rateηηallows us to control the update step length to solve convergence problems.

Programming Adam from scratch isn't too difficult. For convenience, we store the time step counter variable t in the dictionary hyperparams. Other than that, everything else is pretty straightforward.

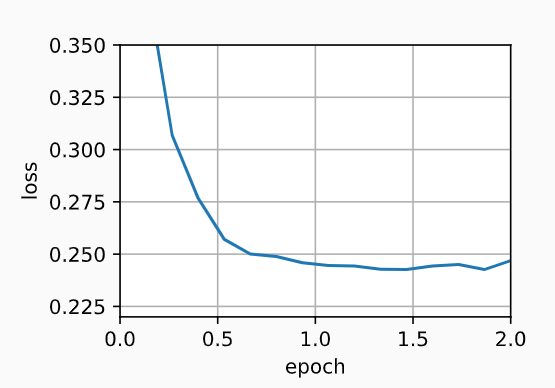
We are ready to use Adam to train the model. We use learning rate η=0.01.





A more concise way of programming is to call it directly adamprovided in trainerGluon's optimization library. So we only need to pass configuration parameters to program in Gluon.



****

## Summary

* Adam combines the techniques of multiple optimization algorithms into a fairly powerful update rule.
* Based on RMSProp, Adam also uses an exponential moving average for the minibatch random gradient.
* Adam uses bias correction to correct for slow start when estimating quadratic momentum and torque.
* For gradients with significant variance, we may encounter problems related to convergence. These problems can be overcome by using larger sized minibatches or by switching to an improved estimator for stst. Yogi is one such solution.