**Optimisation Algorithms**

Optimisation Algoritms are used to update weights and biases i.e. the internal parameters of a model to reduce the error. They can be divided into two categories:

**Constant Learning Rate Algorithms:**Most widely used Optimisation Algorithm, the Stochastic Gradient Descent falls under this category.

https://cdn-images-1.medium.com/max/800/1*LiInWola0bhFRTKzF2L2WA.gif

Here **η** is called as learning rate which is a hyperparameter that has to be tuned. Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully *slow convergence i.e* will result in **small** baby steps towards finding optimal parameter values which minimize loss and finding that valley which directly affects the overall training time which gets too large. While a learning rate that is too **large** can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.

A similar hyperparameter is **momentum**, which determines the velocity with which learning rate has to be increased as we approach the minima.

**Adaptive Learning Algorithms:**The challenge of using gradient descent is that their **hyper parameters have to be defined in advance** and they depend heavily on the type of model and problem. Another problem is that the **same learning rate is applied to all parameter updates**. If we have sparse data, we may want to update the parameters in different extent instead.

Adaptive gradient descent algorithms such as Adagrad, Adadelta, RMSprop, Adam, provide an alternative to classical SGD. They have per-paramter learning rate methods, which provide heuristic approach without requiring expensive work in tuning hyperparameters for the learning rate schedule manually.

**Working with Optimisation Functions:**

We used three first order optimisation functions and studied their effect.

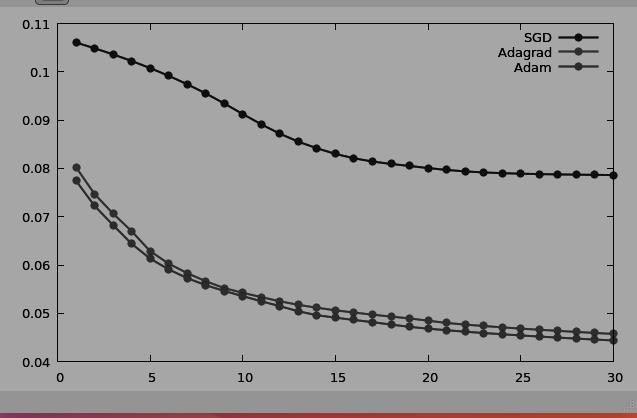
1. Stochastic Gradient Decent
2. Adagrad
3. Adam

**S Gradient Descent** calculates gradient for the whole dataset and updates values in direction opposite to the gradients until we find a local minima. **Stochastic Gradient Descent performs a parameter update for each training exampl**e unlike normal Gradient Descent which performs only one update. **Thus it is much faster. Gradient Decent algorithms can further be improved by tuning important parametes like momentum, learning rate** etc.

**Adagrad** is more **preferable for a sparse data** set as **it makes big updates for infrequent parameters and small updates for frequent parameters**. It uses **a different learning Rate for every parameter θ at a time step based on the past gradients which were computed for that parameter.** Thus we do not need to manually tune the learning rate.

**Adam** stands for Adaptive Moment Estimation. **It also calculates different learning rate.** Adam works well in practice, is faster, and outperforms other techniques.

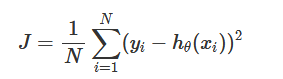
***Stochastic Gradient Decent was much faster than the other algorithms but the results produced were far from optimum. Both, Adagrad and Adam produced better results that SGD, but they were computationally extensive. Adam was slightly faster than Adagrad. Thus, while using a particular optimiza***



**The Mean-Squared Loss:**

For a model prediction such as hθ(xi)=θ0+θ1x

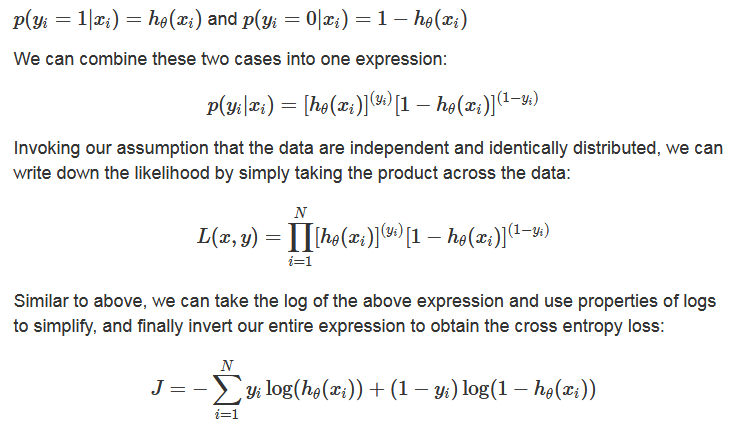
(a simple linear regression in 2 dimensions) where the inputs are a feature vector xi, the mean-squared error is given by summing across all N training examples, and for each example, calculating the squared difference from the true label yi and the prediction hθ(xi)

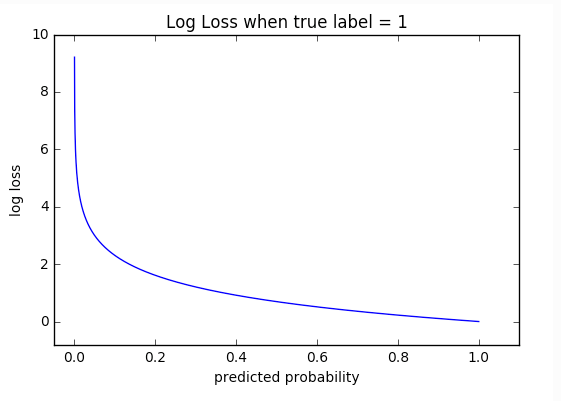


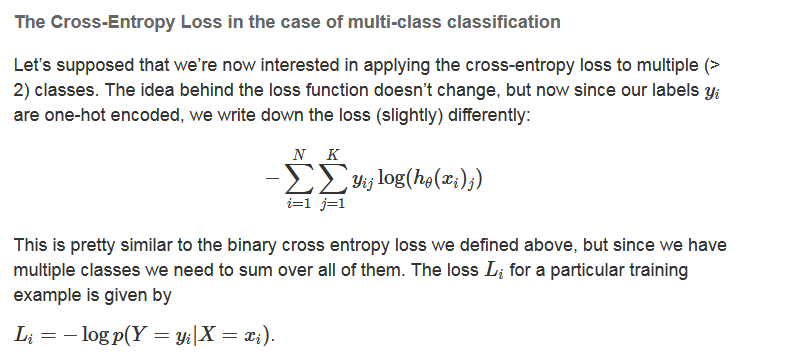
It turns out we can derive the mean-squared loss by considering a typical linear regression problem.

**The Cross-Entropy Loss:**

In the context of classification, our model’s prediction hθ(xi) will be given by σ(Wxi+b) which produces a value between 0 and 1 that can be interpreted as a probability of example xi belonging to the positive class. If this probability were less than 0.5

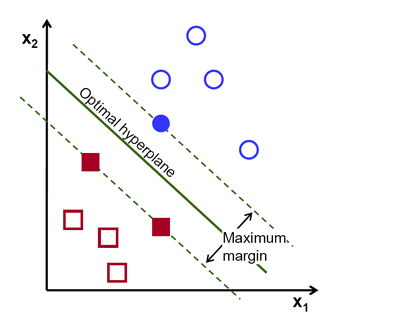
we’d classify it as a negative example, otherwise we’d classify it as a positive example. This means that we can write down the probabilily of observing a negative or positive instance:



****

**Loss Function Hinge (binary)**

The (L2-regularized) hinge loss leads to the canonical support vector machine model with the max-margin property: the margin is the smallest distance from the line (or more generally, hyperplane) that separates our points into classes and defines our classification:



For binary classification problems, the output is a single value ˆy and the intended output y is in {+1, −1}.

The classification rule is sign(ˆy), and a classification is considered correct if y · y >ˆ 0, meaning that y and ˆy share the same sign.

The hinge loss, also known as margin loss:



The loss is 0 when y and ˆy share the same sign and |yˆ| ≥ 1. Otherwise, the loss is linear.

The binary hinge loss attempts to achieve a correct classification, with a margin of at least 1.

Our labels yi are either -1 or 1, so the loss is only zero when the signs match and |(hθ(xi))|≥1. For example, if our score for a particular training example was 0.2 but the label was −1, we’d incur a penalty of 1.2, if our score was −0.7 (meaning that this instance was predicted to have label −1) we’d still incur a penalty of 0.3, but if we predicted −1.1 then we would incur no penalty.

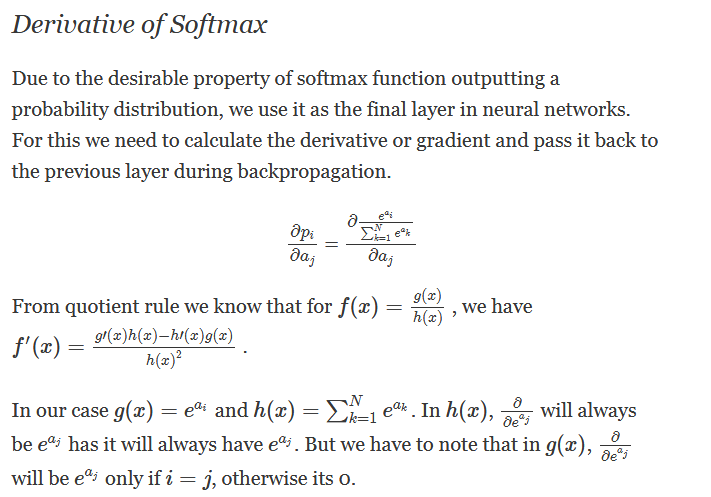
**The main difference between the hinge loss and the cross entropy loss** is that the former arises from trying to maximize the margin between our decision boundary and data points - thus attempting to ensure that each point is correctly and confidently classified\*, while the latter comes from a maximum likelihood estimate of our model’s parameters. The softmax function, whose scores are used by the cross entropy loss, allows us to interpret our model’s scores as relative probabilities against each other. For example, the cross-entropy loss would invoke a much higher loss than the hinge loss if our (un-normalized) scores were [10,8,8] versus [10,−10,−10], where the first class is correct. In fact, the (multi-class) hinge loss would recognize that the correct class score already exceeds the other scores by more than the margin, so it will invoke zero loss on both scores. Once the margins are satisfied, the SVM will no longer optimize the weights in an attempt to “do better” than it is already.

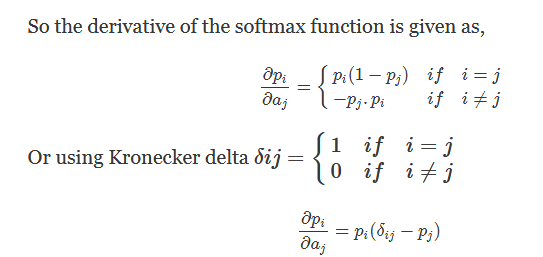
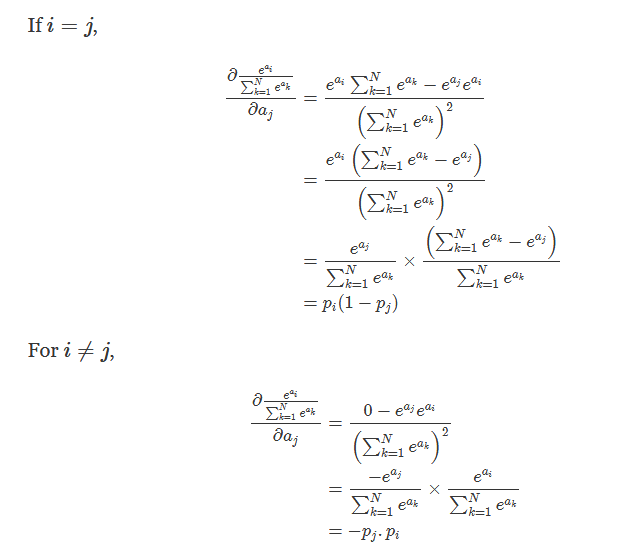
### The Softmax Function

Softmax function takes an N-dimensional vector of real numbers and transforms it into a vector of real number in range (0,1) which add upto 1. *pi*=*eai*∑*Nk*=1*eak*

As the name suggests, softmax function is a “soft” version of max function. Instead of selecting one maximum value, it breaks the whole (1) with maximal element getting the largest portion of the distribution, but other smaller elements getting some of it as well.

This property of softmax function that it outputs a probability distribution makes it suitable for probabilistic interpretation in classification tasks.





**Cross Entropy Loss**

Cross entropy indicates the distance between what the model believes the output distribution should be, and what the original distribution really is. It is defined as, *H*(*y*,*p*)=−∑*iyilog*(*pi*)

Cross entropy measure is a widely used alternative of squared error. It is used when node activations can be understood as representing the probability that each hypothesis might be true, i.e. when the output is a probability distribution. Thus it is used as a loss function in neural networks which have softmax activations in the output layer.

