Advanced Machine Learning – Deep Learning – Week 1

This week, whilst not covering neural networks explicitly, introduce some of the techniques that are used by neural networks, the first being linear regression.

Much of machine learning is about fitting functions to data. To start with we could use an *affine* function of a vector **x** that takes a weighted sum of each input.

In the above each column is a feature and each row is a sample, therefore N is the number of classes and D is the number of features.

We can extend this by using *basis* functions. Each row of the design matrix becomes an arbitrary function of the original input. If the function is non-linear, our function f(x) will be non-linear but we it will still be a linear map of a known vector phi.

Note that we don’t need to include the bias term in the mathematics because one of the basis functions can always be set to a constant. Common examples of these are radial basis functions and the logistic-sigmoid function.

We then need to measure an error of a model - the most popular being the mean squared error.

We need to minimise this w.r.t our parameters. Because we normally use non-linear models and most of them will not use closed-form solutions we need to approach them numerically. Normally this is a case of using gradient-based methods for optimisation problems. Even if it is differentiable, inverting a matrix for high-dimensional data is so extremely computationally expensive as to be a limiting factor.

The first task is to find the gradient of the cost function with respect to the weights – i.e. the vector of partial derivatives. At any given position in the weight-space this vector of derivatives will point in the direction in which a small movement will increase the cost the most.

A naïve way to use the gradient is the steepest-descent method: For each example the weights are pulled in the direction of inputs that had large prediction errors, to reduce the misfit in those directions.

For logistic regression (classes that are binary) we force our function to lie in the desired [0,1] range by using a logistic sigmoid function:

Softmax regression is a generalisation of logistic regression to cases with more than two labels. If there are K possible classes we model the labels with a length-K one-hot encoding. The output of the softmax function can be used to represent a categorical distribution – a probability distribution over K different possible outcomes.

Gradient descent can be used to optimise any differentiable loss function. One starts with an optimisation problem in which we want to find a local minimum. We look for a direction in which the function decreases and take a step in this direction, stopping at some stopping condition. Compared to analytical methods, gradient descent requires less memory and is easier to implement.

A cost function can have a regularisation term added to discourage extreme weights. Common regularisation terms include:

1. L2 regularisation – ridge regression

This adds a squared magnitude as a penalty term.

1. L1 regression – lasso regression

Uses a non-squared coefficient which leads to sparse solution as it drives parameters to zero. This removes some features altogether thus acting as feature selection.

1. Dimensionality reduction techniques such as PCA.
2. Data augmentation, e.g. images can be distorted.
3. Drop out in nets - During the training process, specific neurons may become influenced solely by particular neurons in the previous layer, leading to overfitting. Dropout prevents this by randomly cutting the connections between neurons in successive layers during training. All it does is set a fraction of the inputs to 0.
4. Early stopping
5. Collect more data

Gradient descent is infeasible for large data sets as 1000 gradient steps would require the calculation of billions to gradients. Instead *stochastic gradient descent* can be used. The difference is that at each step we select a random example from training set and calculate the gradient only for that example and take the step in that direction. This leads to noisy approximations. From iteration to iteration the loss will frequently increase. However, these can be used in an online setting.

* *Mini-batch gradient descent:* at every iteration we use m number of examples to estimate the full gradient. Much less noise than stochastic gradient descent.
* Elliptic form of gradient leads to oscillation in convergence. Momentum method – we maintain vetor h at every iteration. It is a weighted sum of all gradients from all previous iterations. It cancels some coordinates that lead to oscillation.
* AdaGrad – we maintain an additional value for each parameter G. Sum of squares of gradients at previous iterations. Suits sparse data. Does not need to cleverly choose learning rate. Can stop working when G becomes large. Chooses its own learning rate for each example – e.g. in text frequent words will have smaller steps than infrequent words.
* RMSprop – similar to AdaGrad but learning rate adapts to latest gradient steps.

We use the exponential to force a number to be positive and the logarithm to make a positive number unconstrained. We use the sigmoid to make numbers lie in [0,1] and the logit (its inverse) to make these values unconstrained.

An aside: we use the log probability to score our models rather than evaluating and differentiating the probability itself for the following reasons:

* The log probability of a training set is a sum rather than a product, making the maths more convenient. A sum can be approximated with a subset so that stochastic gradient descent can be performed.
* Log probabilities avoid numerical underflow – e.g. 0.5 to the power 1000 underflows to 0, even with IEEE floating points.
* The log probability is more appropriate for numerical operators – e.g. a Gaussian’s negative log pdf gives a quadratic cost function, which is convex. The pdf itself is not convex and has values in a narrow numerical range near zero.

They also briefly talk about model evaluation techniques. Normally we use a training set (for fitting the main parameters of a model) and validation set (different variants of a model, e.g. with different regularisation parameters are evaluated on the valisation set) and a test set (to report how likely the selected model is to perform on future data)

K-fold cross-validation can be appropriate for small datasets. The data is split into K parts and the model is trained K times, each with a different block used as a validation set, and the remaining K-1 blocks used for training. The model with the lowest validation error when averaged over the K folds is selected. However, within each fold you need to make model choices or hyperparameter choices, such as the regularisation constant. We can’t fit such choices to the training data but fitting them to the test data would be cheating. This leaves splitting the already small amount of training data into a validation set which leaves the validation scores being noisy. Often K-fold cross validation is a costly mess.

Data Science Journal Club: Snorkel: Rapid Training Data Creation with Weak Supervision