Data Preprocessing

- Mean subtraction is the most common form of preprocessing. It involves subtracting the mean across every individual feature in the data, and has the geometric interpretation of centering the cloud of data around the origin along every dimension.
- Normalization: refers to normalizing the data dimensions so that they are of approximately the same scale. There are two common ways of achieving this normalization. One is to divide each dimension by its standard deviation, once it has been zero-centered. Another form of this preprocessing normalizes each dimension so that the min and max along the dimension is -1 and 1 respectively. It only makes sense to apply this preprocessing if you have a reason to believe that different input features have different scales (or units), but they should be of approximately equal importance to the learning algorithm.
- PCA and Whitening: In this process, the data is first centered as described above. Then, we can compute the covariance matrix that tells us about the correlation structure in the data.
- Common pitfall. An important point to make about the preprocessing is that any preprocessing statistics (e.g. the data mean) must only be computed on the training data, and then applied to the validation / test data. E.g. computing the mean and subtracting it from every image across the entire dataset and then splitting the data into train/val/test splits would be a mistake. Instead, the mean must be computed only over the training data and then subtracted equally from all splits (train/val/test).

Weight Initialization

Pitfall: all zero initialization: This turns out to be a mistake, because if every neuron in the network computes the same output, then they will also all compute the same gradients during backpropagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same.

Small random numbers: Therefore, we still want the weights to be very close to zero, but as we have argued above, not identically zero. As a solution, it is common to initialize the weights of the neurons to small numbers and refer to doing so as symmetry breaking.

Calibrating the variances with 1/sqrt(n): the recommended heuristic is to initialize each neuron's weight vector as: w = np.random.randn(n) / sqrt(n), where n is the number of its inputs. This ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

Sparse initialization. Another way to address the uncalibrated variances problem is to set all weight matrices to zero, but to break symmetry every neuron is randomly connected (with weights sampled from a small gaussian as above) to a fixed number of neurons below it. A typical number of neurons to connect to may be as small as 10.

Initializing the biases. It is possible and common to initialize the biases to be zero, since the asymmetry breaking is provided by the small random numbers in the weights. For ReLU

non-linearities, some people like to use small constant value such as 0.01 for all biases because this ensures that all ReLU units fire in the beginning and therefore obtain and propagate some gradient.

Batch Normalization: forces the activations throughout a network to take on a unit gaussian distribution at the beginning of the training.

Regularization

- L2 regularization: penalizing the squared magnitude of all parameters directly in the objective. That is, for every weight w in the network, we add the term 12λw2 to the objective, where λ is the regularization strength. It is common to see the factor of 12 in front because then the gradient of this term with respect to the parameter w is simply λw instead of 2λw.
- L1 regularization: for each weight w we add the term λ|w| to the objective. It is possible to combine the L1 regularization with the L2 regularization: λ1|w|+λ2w2 (this is called Elastic net regularization). The L1 regularization has the intriguing property that it leads the weight vectors to become sparse during optimization (i.e. very close to exactly zero).
- Max norm constraints: enforces an absolute upper bound on the magnitude of the weight vector for every neuron and use projected gradient descent to enforce the constraint.
- Dropout: is implemented by only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise.
- Bias regularization. Although it is not common to do so, iin practical applications (and with proper data preprocessing) regularizing the bias rarely leads to significantly worse performance. This is likely because there are very few bias terms compared to all the weights, so the classifier can “afford to” use the biases if it needs them to obtain a better data loss.
- Per-layer regularization. It is not very common to regularize different layers to different amounts (except perhaps the output layer)