# VII.4) Hyperparameters

A neural net has many parameters that we optimize during the training process. However, there are other parameters, called **hyperparameters**, that can greatly affect the training of our neural net and its accuracy.

---

## The learning rate

Recall that the optimization algorithms we use to train the neural net are based on the iteration:

$$x_{k+1} = x_k + s_k d_k, \quad k = 0, 1, 2, \ldots,$$

where $d_k$ is the **search direction** and $s_k$ is the **step length**.

Many optimization algorithms choose $s_k$ using a line search procedure. However, this is too expensive for neural nets, so often a fixed step length, called the learning rate, is used.

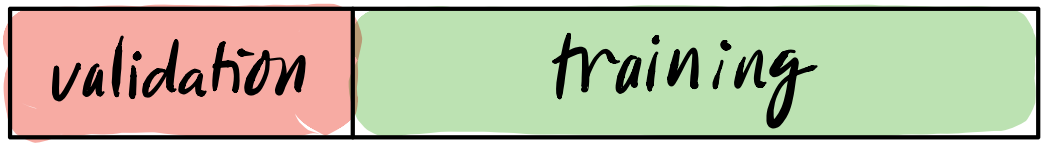
The quality of the neural net parameters found by the optimization algorithm can be very sensitive to the learning rate.

A learning rate that is too large may overshoot the optimal solution, but a learning rate that is too small may require a huge number of steps to get close.

# Cross-validation

A good way to find a good learning rate is to train your neural net with different learning rates and then see how well your neural net performs on some unseen validation data.

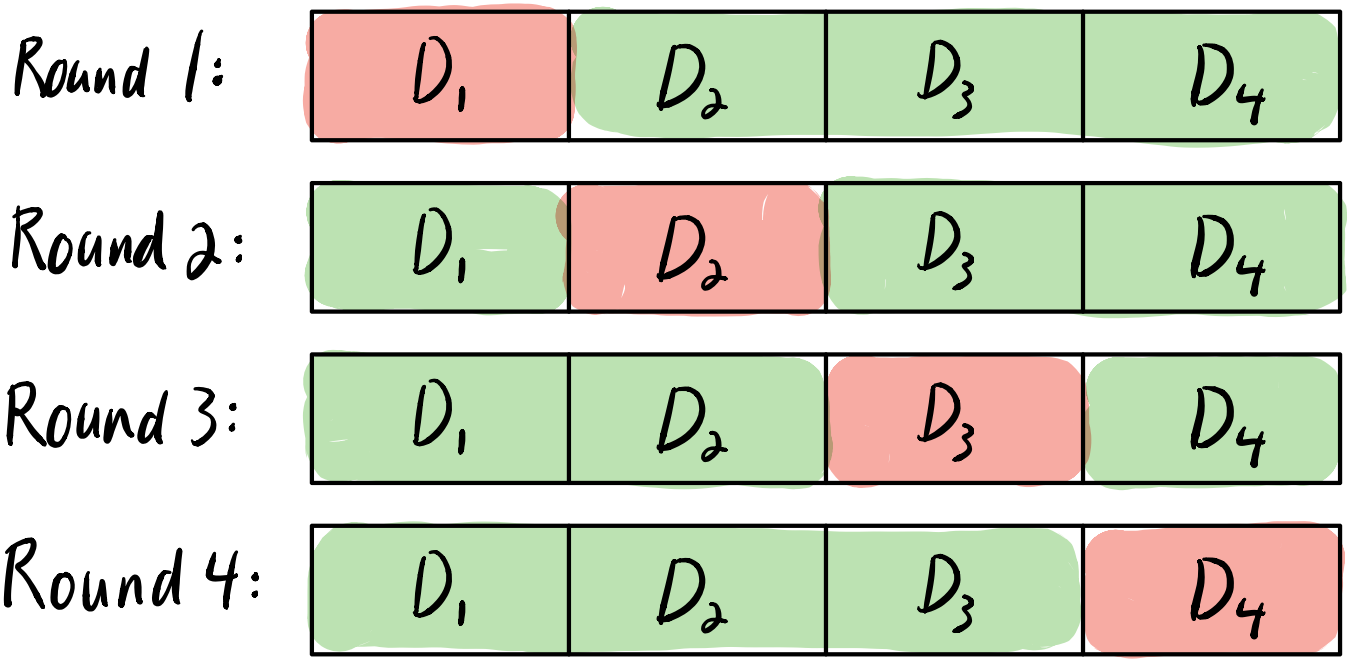
This validation data is often selected from your training data. Putting the validation data aside, we train our neural net using the remaining training data. After training is done, we test our neural net on the validation data.



This is cross-validation.

Since we want to make sure we are not biased, it is better to make many such partitions and to average the results.

K-fold cross-validation partitions the data into K equal subsets; each subset is used as the validation data for one of the K rounds of cross-validation.



Each data point is used once for validation and K-1 times for training. The validation scores are then averaged.

After using cross-validition to find good values for the hyperparameters, we train the neural net using all the training data and test it using the test data.

---

## Batch Normalization

Neural nets that use the sigmoid activation function may find the the mean of the output of each layer is shifted and the standard deviation is changed.

To avoid this, we can normalize the output. Let $v_1, \dots, v_B$ be the output of a layer of the neural net for a minibatch of size $B$. Then we let

$$\mu = (v_1 + \cdots + v_B)/B,$$

$$\sigma^2 = \left(\|v_1 - \mu\|^2 + \cdots + \|v_B - \mu\|^2\right)/B,$$

$$V_i = (v_i - \mu)/\sqrt{\sigma^2 + \varepsilon}, \quad i = 1, \dots, B,$$

$$y_i = \gamma V_i + \beta, \quad i = 1, \dots, B,$$

where $\gamma$ and $\beta$ are learnable parameters. Then the mean and standard deviation of $y_1, \dots, y_B$ are $\beta$ and $\gamma$.

# Dropout

A common difficulty with training neural nets is getting high accuracy on the training data but low accuracy on the test data. This is known as overfitting the training data.

One way to address overfitting is to train many different neural nets and then average their output. However, this would take a very long time to train those neural nets. An efficient alternative to this is dropout.

Dropout has been shown to be a simple way to avoid overfitting.

Certain neurons in the network are randomly kept with probability $p$ at each gradient step. Thus, with probability $1-p$, those neurons are temporarily removed, their associated weights are taken to be zero and are not updated. The neurons that remain have their parameters updated as usual.

After training all dropout neurons are present, but have their weights scaled by $p$. In a way we are averaging many different neural nets.