Chapter 5 - The mechanics of learning

(Dated: April 8, 2021)

5.1 A TIMLESS LESSON IN MODELING

'How to do science' by Johannes Kepler

1. collect data
2. attempt to visualize
3. locate simplest model
4. split data into training and validation
5. begin with pretrained parameters, fit over time
6. test model on validation data
7. be shocked by results

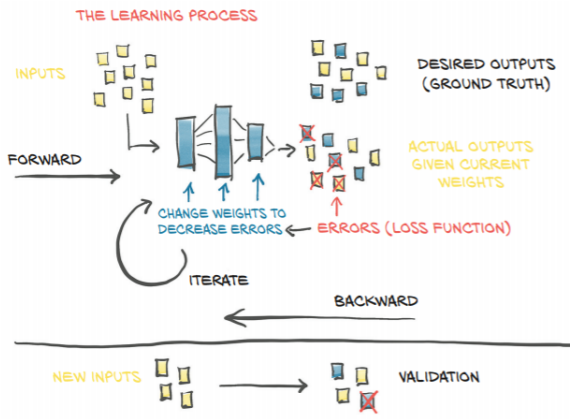

Figure 1. The learning process (this will make sense by the chapter's end)

5.2 LEARNING IS JUST PARAMETER ESTIMATION

By feeding a model data while knowing the desired outputs, we can begin to optimize the model by comparing the resultant outputs to the ground truths and adjusting the weights and biases of the model in a direction which minimizes the difference.

5.3 LESS LOSS IS WHAT WE WANT

The function which uses the difference between model output and ground truth in an attempt to positively modify the weights and biases is called the 'loss function' (or cost function). After looking at set of temperature data and attempting to model it linearly, we are able to calculate the loss using a convex *mean square* loss function. Great, but now that we have calculated the loss, what do we do with it to improve the model?

5.4 DOWN ALONG THE GRADIENT

To optimize the loss function with respect to the parameters, we use *gradient descent*. Essentially, gradient descent uses the derivative of the loss function with respect to each parameter to incrementally update each parameter minimizing loss and thus improving the model. At this point, we can describe one *epoch* (iteration through the training dataset) of learning as such:

```
predictions = model(inputs, params)
loss = loss_function(predictions, ground_truths)
params -= learning_rate * params.grad
```

with the learning rate being a small value (say 10^{-3}) to prevent overshooting the minima, and the `grad` attribute of `params` containing derivatives of loss with respect to each parameter.

5.5 PYTORCH'S AUTOGRAD: BACKPROPAGATING ALL THE THINGS

Computing gradients numerically can quickly escape the realm of manual human possibility when considering complex models and loss functions with thousands and even millions of parameters. This is where PyTorch and it's *autograd* component come to the rescue.

5.5.3 Training, validation and overfitting

Given a sufficiently uniform, small, or otherwise compromised dataset, any model we attempt to create will be prone to *overfitting*. Overfitting occurs when rather than generalizing outputs, the model memorizes the input data over many epochs and is unable to perform adequately on new data.

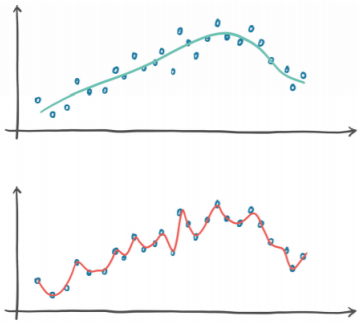

Figure 2. Example of overfit data

The solution to this is dividing data into a training and validation set. After remaining hidden from the model as it learns from the training set, the model can use the validation set to ensure generalizability.