Lecture 11

Nonlinear Function Approximation

As powerful as linear function approximation is, it still has some fundamental limitations. Consider the case where you're trying to model the torque due to the aerodynamic forces on a weather-vane. The dynamics are given as

$$\ddot{\theta} = \frac{1}{2I}\rho v^2 C_n(\alpha) S l_w \tag{11.1}$$

where $\alpha = \arctan \frac{vz}{vx}$ and $v = \sqrt{v_x^2 + v_z^2}$, but $v_z = \omega l_w$.

If the center of pressure l_w is not known, then we are no longer affine in the parameters. Similarly, if we choose to use basis functions, there are some important design decisions we must make regarding the non-affine parameters. For instance, for the radial basis functions, we must choose the centers and the spread:

$$\phi_i(x) = \exp \frac{||x - \mu_i||^2}{2\sigma^2} \tag{11.2}$$

Both μ_i and σ are non-affine parameters.

11.1. Nonlinear Least-Squares

Consider $y = f(x, \theta)$ where $y = \mathbb{R}^{m\times 1}$ and $x = \mathbb{R}^{n\times 1}$

$$J = \frac{1}{2}\sum_{i=1}^{N} ||y_i - f(x,\theta)||^2 = \sum_{i=1}^{N} (y_i - f(x,\theta))^T (y_i - f(x,\theta)) \tag{11.3}$$

$$\hat{\theta} = \arg\min_{\theta} J(\theta) \tag{11.4}$$

A very simple approach would be to apply gradient descent

$$\hat{\theta}^{i+1} = \hat{\theta}^i - \alpha \frac{\partial J}{\partial \theta} \quad \alpha > 0 \tag{11.5}$$

$$\frac{\partial}{\partial \theta} J = \frac{\partial}{\partial \theta} \frac{1}{2} \sum_{i=1}^{N} (y_i - f(x,\theta))^T (y_i - f(x,\theta)) \tag{11.6}$$

$$= \frac{1}{2} \sum_{i=1}^{N} \frac{\partial}{\partial \theta} (y_i - f(x,\theta))^T (y_i - f(x,\theta)) \tag{11.7}$$

$$= \sum_{i=1}^{N} -(y_i - f(x_i,\theta))^T \frac{\partial f(x_i,\theta)}{\partial \theta} \tag{11.8}$$

$$\tag{11.9}$$

11.2. Multi-Layer Neural Networks

Multi-layer neural networks have risen to the fore in terms of methods for function approximation. As with many of the approaches we've explored, computing the gradient is often key for training these function approximators. There are actually "deep" connections to the "method of adjoints" we used to derive the "backpropagation through time" trajectory optimization method. The standard deep learning objective is

$$\min_{\theta} \frac{1}{n} \sum_{k=1}^{n} g(\phi(x_k), y_k). \tag{11.10}$$

For a multi-layer neural network, we have

$$\phi(x, \theta) = f_\ell \circ f_{\ell-1} \circ f_{\ell-2} \ldots \circ f_1 x \tag{11.11}$$

where each f_i has a vector of parameters, θ_{i-1}, which may be optimized.

Let's write down our optimization problem:

$$\min_{\theta} \frac{1}{n} \sum_{k=1}^{n} g(z_k^{(l)}, y_k) \tag{11.12}$$

$$s.t. \quad z_k^{(\ell)} = f_\ell(z_k^{\ell-1}, \theta_\ell) \tag{11.13}$$

$$z_k^{(\ell)-1} = f_{\ell-1}(z_k^{\ell-2}, \theta_{\ell-1}) \tag{11.14}$$

$$\vdots \tag{11.15}$$

$$z_k^{(1)} = f_1(x_k, \theta_1) \tag{11.16}$$

Now we can form the Lagrangian

$$\mathcal{L} = g(z^{(\ell)}, y) - \sum_{i=1}^{\ell} p_i^T (z^{(i)} - f_i(z^{(i-1)}, \theta_i)) \tag{11.17}$$

Take the derivatives and set to zero!

$$\nabla_{z^{(i)}} \mathcal{L} = -p_i + \nabla_{z^{(i)}} f_{i+1}(z^{(i)}, \theta_{i+1})^T p_{i+1} \tag{11.18}$$

$$\nabla_{z^{(\ell)}} \mathcal{L} = -p_\ell + \nabla_{z^{(\ell)}} g(z^{(\ell)}, y) \tag{11.19}$$

$$\nabla_{\theta_i} \mathcal{L} = \nabla_{\theta_i} f_i(z^{(i-1)}, \theta_i)^T p_i \tag{11.20}$$

$$\nabla_{p_i} \mathcal{L} = z^{(i)} - f_i(z^{(i-1)}, \theta_i) \tag{11.21}$$

If we set $\nabla_{p_i} \mathcal{L} = 0$, this enforces the constraints $z^{(i)} - f_i(z^{(i-1)}, \theta_i)$, which can be satisfied via a forward pass. Using the backward pass, we can then compute p_i where

$$p_\ell = \nabla_{z^{(\ell)}} g(z^{(\ell)}, y) \tag{11.22}$$

and

$$p_i = \nabla_{z^{(i)}} f_{i+1}(z^{(i)}, \theta_{i+1})^T p_{i+1}. \tag{11.23}$$

The gradients can then be computed as

$$\nabla_{\theta_i} \mathcal{L} = \nabla_{\theta_i} f_i(z^{(i-1)}, \theta_i)^T p_i. \tag{11.24}$$

This update can be computed in batch, for all the data points. However, sometimes, this is not computationally tractable for very large data-sets. Sometimes, only a single pair of x, y values are used. This results in Stochastic Gradient Descent, since this is a stochastic approximation of the gradient for the whole data set. Regularization can also be used as part of the back-propagation algorithm. Simply add a cost on the weights.

11.3. Nonlinear Observers with an Augmented State

Another way to handle estimating parameters that appear nonlinearly is to use a state estimator designed for nonlinear systems. Consider the system

$$\dot{x} = f(x, u, \theta) \tag{11.25}$$

which can be rewritten as

$$\dot{x} = f(x, u, \theta) \tag{11.26}$$
$$\dot{\theta} = 0 \tag{11.27}$$

or

$$\bar{x} = f(\bar{x}, u) \tag{11.28}$$

where $\bar{x} = \begin{bmatrix} x \\ \theta \end{bmatrix}$.
In discrete time, we have

$$x_{k+1} = x_k + f(x_k, u_k, \theta_k)dt \tag{11.29}$$
$$\theta_{k+1} = \theta_k \tag{11.30}$$

which we can write

$$\bar{x}_{k+1} = f_d(\bar{x}_k, u_k). \tag{11.31}$$

If $y = g(\bar{x}, u)$, it would be straightforward to write the equations for the Extended Kalman Filter:

$$P_{k+1} = F_k \hat{P}_k F_k^T + Q_k \quad \tilde{y}_k = z_k - h(\hat{x}_k) \tag{11.32}$$

$$S_k = H_k P_{k+1} H_k^T + R_k \quad K_k = P_k H_k^T S_k^{-1} \tag{11.33}$$

$$\hat{x}_{k+1} = \hat{x}_k + K_k \tilde{y}_k \quad \hat{P}_{k+1} = (I - K_k H_k) P_{k+1} \tag{11.34}$$

where

$$F_k = \frac{\partial f_d}{\partial x}(x_k, u_k) \quad H_k = \frac{\partial g}{\partial x}(x_k, u_k) \tag{11.35}$$

Note that augmenting the state space can introduce additional nonlinearities into your process model. Since it is assumed that the parameters can not be directly measured, the observability of your system could also impact parameter convergence. Other alternatives to the EKF might be the Unscented Kalman Filter (UKF). This algorithm uses Sigma Points instead of linearization of the dynamics to approximate the propagation of uncertainty through nonlinear dynamics.

11.4. Moving Horizon Estimation

What is the connection between trajectory optimization and parameterization? Consider the modified optimization problem:

$$\min_{x,\theta} \sum_{i=0}^{N} (x - x_{m,i})^T (x - x_{m,i}) \tag{11.36}$$
$$s.t. \quad x_{k+1} = x_k + f(x_k, u_k, \theta_k)dt \tag{11.37}$$
$$\theta_{k+1} = \theta_k \tag{11.38}$$

We've already seen a way for solving a problem like this–via direct transcription. Instead of solving for x_k, u_k we now solve for x_k, θ_k. Similarly, we could apply the adjoint methods as before to satisfy the dynamics constraints via a forward pass. Nonlinear Moving Horizon Estimation (NMHE) is a an approach that solves the state-estimation problem looking backwards.

Bibliography

[1] Dimitri Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

