Reinforcement Meta Learning

A thesis submitted by

Matthew Stachyra

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Tufts University

December 2023

Advisor: Jivko Sinapov

The abstract is a very brief summary of the dissertation's contents. It should be half a page long at most. Somebody unfamiliar with your project should have a good idea of what it's about having read the abstract alone and will know whether it will be of interest to them.

This is not needed, but common.

Chapter 1

Background

1.1 Artificial Neural Networks

Artificial neural networks (ANNs) are non-linear computational models that approximate a target function $f : \mathbb{R}^n \to \mathbb{R}^m$, where n and m are integers [1]. Given a set $X = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\} \subset \mathbb{R}^n \times \mathbb{R}^m$ of input-output pairs of size N the model is trained to approximate f such that $f(\mathbf{x}_i) = \mathbf{y}_i \forall i \in \{1, 2, \ldots, N\}$. By the Universal Approximation theorem, a neural network's approximation of a continuous function is theoretically guaranteed to be as precise as it needs to be given the network has at least one hidden layer with some finite number of nodes [2].

1.1.1 Starting from linear regression

Consider the problem of predicting the value of one or more continuous target variables $\mathbf{t} \subset \mathbb{R}^m$ provided a D-dimensional vector $\mathbf{x}$ of input variables, or what is called regression. Given a training set consisting of N observations $\{\mathbf{x}_n\}_{n=1}^N$ and values $\{\mathbf{t}_n\}_{n=1}^N$, the objective is to predict the output value for any input vector $\mathbf{x}$ as close as possible to the provided target variable $\mathbf{t}$.

One approach is linear regression, or a linear combination over the input variables

$$y(\mathbf{x}_n, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D \tag{1.1}$$

where $\mathbf{x}_n$ has D dimensions, $\mathbf{x}_n = (x_1, \ldots, x_D)^T$ and $w \in \mathbb{R}^{D+1}$ represents the parameters of the function, $w = (w_0, \ldots, w_D)$ and D is extended to $D+1$

for the bias weight w_0.

As is, this regression function is limited to being a linear function over the input vector $\mathbf{x}_n$. Non-linear basis functions ϕ on the input variables make the function $y(\mathbf{x}_n, \mathbf{w})$ non-linear for an input $\mathbf{x}_n$:

$$y(\mathbf{x}_n, \mathbf{w}) = w_0 + \sum_{i=1}^{D} w_i \phi_i (x_i) \tag{1.2}$$

This equation can be simplified further if we define a useful basis function for the bias $\phi_0(\mathbf{x}) = 1$ such that

$$y(\mathbf{x}_n, \mathbf{w}) = \sum_{i=0}^{D} w_i \phi_i (x_i) \tag{1.3}$$

Despite producing non linear outputs over the input $\mathbf{x}$ this is *linear regression* because it is linear with respect to $\mathbf{w}$.

1.1.2 Constructing neural networks

Basic ANNs can be seen as an extension to linear regression where the basis functions become parameterized. The basis functions continue to be non-linear functions over the linear combination of the input, but now the output of the basis function is dependent on the learned coefficients $\{ w_j \}$. In this construction, basis functions are known as *activation* functions h in the context of neural networks.

We start by rewriting equation 1.2 as a linear combinations over the input variables to produce a or the *activation*.

$$a = \sum_{i=1}^{D} w_i x_i + w_0 \phi (x_i) \tag{1.4}$$

The value a is transformed using a non-linear activation function h. This transformation produces z and is referred to as a *hidden unit*.

$$z = h (a) \tag{1.5}$$

The coefficients $\{ w_j \}$ parameterizing this non-linear transformation are referred to as a *layer*.

An ANN has a minimum of two layers - an input layer and output layer. However, ANNs are not limited to 2 layers. ANNs can have l many layers where $l \in [2, +\infty)$. Networks with > 2 layers are referred to as *deep neural networks*. For the purposes of the background, we will continue with the simple 2-layer case to establish preliminaries.

The input layer operates on an input $(x_1, \ldots, x_D)$ to produce *activations* $a_j = (a_1, \ldots, a_M)$, where M denotes the number of parameters $\{ w_j \}$ in the input layer. The parameters for the input layer are represented with a superscript (1) and the parameters for the output layer will be represented with a superscript (2).

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{1.6}$$

The activations are passed through a non-linear activation h

$$z_j = h\left(a_j\right) \tag{1.7}$$

The output layer then transforms the hidden units z_j to produce output unit activations a_k where $k \in (1, \ldots, K)$ and K is the number of outputs expected for this problem (i.e., appropriate to the target variable $\mathbf{t}_i$ for $\mathbf{x}_i$).

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{1.8}$$

The activation a_k is transformed by a different non-linear activation function that is appropriate for K. Here this activation function is represented as σ. A common choice of activation function h for non-output layers is the rectified linear unit $h(a) = \min(0, a)$. A common choice of activation function for σ is the sigmoid function $\sigma(a) = \frac{1}{1+e^{-a}}$ for classification problems and the identity $y_k = a_k$ for simple regression problems. We now present the equation for a *feed-forward* pass through a 2-layer ANN.

$$y_k(\mathbf{x}_n, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} + w_{j0}^{(1)} \phi\left(x_i\right)\right) + w_{k0}^{(2)}\right) \tag{1.9}$$

A neural network then is a non-linear function over an input $\mathbf{x_n}$ to an output y_k that seeks to approximate $\mathbf{t_n}$ and is controlled by a set of adaptable parameters $\mathbf{w}$.

1.1.3 Training a neural network

The goal of learning for a neural network is to optimize the parameters of the network such that the loss function $E(X, \mathbf{w})$ takes the lowest value. Continuing with the previous example for regression, we look at the sum-of-squares error function

$$E(X, \mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \| y\left(\mathbf{x}_n, \mathbf{w}\right) - \mathbf{t}_n \|^2 \tag{1.10}$$

There is typically not an analytical solution and iterative procedures are used to minimize the loss function E. The steps taken are

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta \mathbf{w}^{(\tau)} \tag{1.11}$$

where τ is the iteration step. An approach for the weight update step with $\Delta \mathbf{w}^{(\tau)}$ is to use the gradient of $E(X, \mathbf{w})$ with respect to the parameters $\mathbf{w}$. The weights are updated in the direction of steepest error function decrease or in the $-\nabla E(X, \mathbf{w})$ direction.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \alpha \nabla E\left(X, \mathbf{w}^{(\tau)}\right) \tag{1.12}$$

where $\alpha > 0$ is the learning rate controlling the size of update step taken. This iterative procedure is called *gradient descent optimization* [3].

1.1.4 Error function derivatives

The gradient $\nabla E(X, \mathbf{w})$ is calculated with respect to every $w \in \mathbf{w}$, for all $\mathbf{x}_n \in X$.

We start with one input pattern $\mathbf{x_n}$ and rewrite the error function as

$$\begin{aligned} E_n &= \frac{1}{2} \left(y(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n} \right)^2 \\ &= \frac{1}{2} \sum_{j=1} (w_{kj} z_j - t_{nk})^2 \end{aligned} \tag{1.13}$$

The calculation starts with the gradient of E_n with respect to each w_{kj} in the output layer (2) then continues backwards to layer (1) for w_{ji}. This method can extend to l-layer networks where $l = (1, \ldots, L)$ and $L \subseteq \mathbb{R}$.

Observe that

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}} \tag{1.14}$$

We start by calculating the partial derivative of E_n with respect to the activation a_k. Recall that $a_k = \sum_k w_{kj} z_j$. By the chain rule:

$$\begin{aligned} \frac{\partial E_n}{\partial a_k} &= (h(a_k) - t_{nk}) h'(a_k) \\ &= h'(a_k)(\hat{y}_n - t_{nk}) \end{aligned} \tag{1.15}$$

We introduce a new notation to call this partial derivative an *error*

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} \tag{1.16}$$

Next we calculate the partial derivate of a_k with respect to w_{kj}

$$\begin{aligned} \frac{\partial a_k}{\partial w_{kj}} &= \frac{\partial}{\partial w_{kj}} \left(\sum_k w_{kj} z_k \right) \\ &= z_k \end{aligned} \tag{1.17}$$

With which we can write

$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_k \tag{1.18}$$

The procedure will continue in the same way for the remainder of the layers and their units, where we calculate the errors δ for the units in the layer and multiply error of that unit by its activation z. For layer (1) (input layer) we need to calculate

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{1.19}$$

starting with δ_j or $\frac{\partial E_n}{\partial a_j}$. Observe that

$$\frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{1.20}$$

where k is the number of outputs (here, k=1 for the continued regression example).

We calculated $\frac{\partial E_n}{\partial a_k}$ above. Continue with

$$\begin{aligned} \frac{\partial a_k}{\partial a_j} &= \frac{\partial}{\partial a_j}\left(\sum_k w_{kj} h(a_j)\right) \\ &= h'(a_j) w_{kj} \end{aligned} \tag{1.21}$$

We can finish calculating the error δ_j for equation 1.18

$$\begin{aligned} \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} &= h'(a_k)(\hat{y}_n - t_{nk}) h'(a_j) w_{kj} \\ &= \frac{\partial E_n}{\partial a_j} \\ &= \delta_j \end{aligned} \tag{1.22}$$

Thus we obtain the *backpropagation* formula

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \tag{1.23}$$

where the error for a unit j is the result of backpropagating the errors in the units later in the network.

Calculating the gradient is backpropgating the errors. As we have seen, this procedure begins with a forward propagation of the input vectors x_n to calculate the activations of all units. Then, it involves calculating the errors δ_k in the output layer. Using δ_k we can calculate δ_j for the hidden units in previous layers. With all errors δ, the gradient is calculated by multiplying the error by the activations a transformed by their non-linear function h where $h(a) = z$.

1.1.5 Recurrent Neural Networks

ANNs can be constructed as *directed graphs*, formally defined as $G = (V, E)$ where V is the set of vertices $\{ v_1, \ldots, v_n \}$ and E is the set of edges $\{ (u, v) \mid u, v \in V \}$.

We show neural networks are directed becuause the edges are a set of ordered pairs. In comparison, an undirected graph would have edges $\{ \{u, v\} \mid u, v \in V \}$. In terms appropriate to neural networks, V corresponds to our hidden units $\{ z \}$ and output units and E corresponds to the parameters $\{ w \}$.

The 2-layer network we constructed above was a *directed acyclic graph*. G is acyclic if $\forall v \in V$, there does not exist a cycle containing v. This means that for $\forall (u, v) \in E$, $u \neq v$.

ANNs can contain cycles however. A type of neural network that contains cycles is a *recurrent neural network* (RNN). An RNN is recurrent in that its output is part of forward propagation in the next timestep.

1.1.6 Vanishing Gradient Problem

1.2 Reinforcement Learning

1.2.1 Markov Decision Processes

1.2.2 Policy Gradient Methods

1.2.3 PPO

1.3 Meta Learning

Chapter 2

Title chapter 2

2.1 Title section 2.1

2.1.1 If needed

2.1.2 If needed

2.2 Title section 2.2

2.2.1 If needed

2.2.2 If needed

2.3 Title section 2.3

2.3.1 If needed

2.3.2 If needed

Appendix A

Title of the Appendix

Bibliography

[1] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.