（2016 年 7 月 13 日講演）

機械学習が変えるプログラミングパラダイム

株式会社 Preferred Networks
最高戦略責任者　丸山宏委員

今日は、「機械学習が変えるプログラミングパラダイム」という話をさせてもらう。前回安宅委員が人工知能というのは期待感が高過ぎるという話をされたが、私はもう一歩さらにステップバックして機械学習というものを考えたい。

その背景としては、私自身は古い世代のコンピュータサイエンティストだと思っており、最初に高校で OKITAC3400 というのでプログラミングをした時に面白いなと思い、当時は情報系の大学がほとんどなく、探してみたら東工大に情報科学科があったので、そこで情報科学を学んだ。私の指導教官は木村先生だったが、実際に指導してくれたのは米澤先生で、米澤先生の下で自然言語理解システムをやった。これは Prolog という言語で書いた。それから IBM に入って自然言語処理、機械翻訳、XML、セキュリティなど、ありとあらゆるコンピュータサイエンスをやったが、これは典型的なコンピュータサイエンスの領域である。これを 26 年間やり、2009 年に会社を辞めた。少しブランクがあったが、その後、統計数理研究所から声が掛かり、今は Preferred Networks という会社にいる。ここは全く文化が違っていて、例えば久世委員などが学んできた典型的なコンピュータサイエンスとは違う文化の世界である。

問題は、典型的なバックグラウンドを持っている人から見て、どのように見えるか、どういうインプリケーションがあるのか、私はこの 5 年間ずっと頭の中で、これはどういう意味があるのだろうと思い続けてきた。

人工知能はコンピュータサイエンスに影響を与えている。最初の人工知能ブームは私が生まれた頃で、サイバネティクスや、LISP などが流行っていて、その当時のコンピュータは計算するマシンで、記号を処理するなどということは誰も考えていなかった。当時は人工知能をやりたかったら記号処理をやらないといけないということで LISP というものが生まれてくる。それから、プランニング主体。例えば「あそこに行きたい」と言ったときに、その目標をベースにステップバックして、何を考えるかということをやるアルゴリズムが Means-End Analysis と呼ばれていた。その他に自然言語処理なども行われていた。これはよく考えてみると、LISP でやっているのは動的メモリ管理である。ガベージコレクションは LISP が最初であり、それ以外にはなかった。今私たちが当たり前に思っているヒープやガベージコレクションというものも人工知能から出てきている。それから、探索アルゴリズムである。深さ優先探索や幅優先探索も人工知能から出てくる。コンパイラを作るのに形式言語理論が必要であるが、それも実はチョムスキーの文法理論である自然言語

処理から出てきている。これらが第 1 次人工知能ブームから現れてきたコンピュータサイエンスに与えたインパクトである。

第 2 次人工知能ブームは、ちょうど私が大学を卒業するころで、第 5 世代や知識表現などと言われている。例えば、車というのはタイヤが 4 つあってハンドルがあるというのは知識表現である。エキスパートシステム、OPS5 など、ルールベースでルールを変えていく。それから常識を知らないといけないからと、オントロジーを使う。常識の知識体系を作ろうという Cyc というプロジェクトが起きる。世の中の常識を百科事典としてコンピュータに分かるように書こうという巨大なプロジェクトである。このフレームは今の言葉で言うとオブジェクト指向である。つまり、このオブジェクトには車とタイヤがあってとか、メンバーをずっと書いているわけである。それから、エキスパートシステムやフレームもモデリング言語に影響を与えている。オントロジーの世界というのは、セマティックウェブのようなものに影響を与えるとか、そのような考えである。

今は第 3 次の人工知能ブームで、キーのテクノロジーは何かというと、統計的機械学習であり、深層学習であるが、では、これが今後のコンピュータサイエンスにどういうインパクトを与えていくのか、私の頭の中で、どうなのだろうかを一生懸命今考えているところである。だから、この委員会は、人工知能が社会や産業をひっくり返すと言うが、今日の私の話は少し狭い領域で、統計的機械学習、深層学習がコンピュータサイエンスにどのように影響を与えていくかを皆さんと議論していきたいと思う。

（統計的）機械学習再訪

そもそもプログラムとは何か、このようなところから始める。チューリングマシーンでアブストラクションするなど、いろいろな考え方がある。一つのアブストラクション、プログラムとは関数である、つまり入力のデータを送って出力を出すようなものだと考えることができる。普通のプログラミングはどういうプログラミングかというと、例えば、摂氏を入れると華氏が出てくるようなプログラム、そういうシステムが欲しいと皆さん思ったとする。普通のやり方は、こういうプログラムシステムが欲しいと、出力と入力の関係はこういう関係になっているようなシステムを作ってほしいと仕様で書く。それを、これはどのように計算するのか。ある程度世の中こうなっているということを形式的に書き下すのがモデルだが、そのモデルというものを書く。それをさらに段階的に詳細化していって、例えばプログラムに落とす。Java のプログラムであるが、詳細まで全部詰めると、実際に動くシステムになる。モデルはそのままでは動かないので、動くようにする。これが一つのシステムの作り方である。今私たち IT 産業の人が、この 60 年、70 年の歴史の中でずっとやってきたことである。

ところが、システムの作り方はこれだけではない。もう一つの作り方は、むしろ入出力の例示を与えることでシステムを作るという考え方ができるのではないか、これが機械学習的な物の考え方である。華氏と摂氏の温度計を同時に見て、何度もサンプル点を取ると、ばらつきが表れる。そこで、ある意味仕様を与えて、こういう入出力の関係のあるシステ

ムを作れ、華氏、摂氏という入出力を出せと言って回帰分析をするとが引ける。実は機械学習の最もシンプルなフォームが回帰分析のようなものだと思ってほしい。『PRML(Pattern Recognition and Machine Learning)』という分厚い、この世界の標準的な教科書があるが、そこに最初に出てくる機械学習のデータが、実はこの回帰分析の例である。回帰分析は何かというと、データを与えると、こういうモデルが出てくるというものが機械学習の本質になる。

機械学習から出てきた結果で、我々がよく知っているプログラミングシステムの作り方と違うのは、単純に回帰分析するとこのような線が出るが、これは入力データをうまく表していないかもしれない。入力データにはばらつきがあり、ばらつきが大きいものと小さいものに関しては、このままの形では表せないので、ばらつきもあるような形でモデル化し、回帰分析することもできる。数式に誤差項があるが、例えば正規分布で誤差が出るようなことをやると、入力に対して常に同じ答えを返す確定的な関数ではなく、確率的に答えを返すようなシステムになる。我々古いコンピュータサイエンティストから見ると、システムは同じ入力を与えたら同じ振る舞いをするべきだろう、このようなものは嫌だと思うかもしれないが、実は機械学習というのは本質的にこういう概念を含んでいるのではないかと思う。

このような話はいわゆる統計モデリングという世界である。機械学習の最初の部分は統計モデリングだと思えばよいわけであるが、統計モデリングとはどういうことか。点々がサンプルだとする。それに対して線形で回帰すると、左のような感じになる。真ん中は sin カーブから作ったのだが、3 次式で回帰すると、そこそこきれいかなと思う。右は次数を 40 次にして回帰したものであるが、よく見ると、本来スムーズになっているべきものがぐちゃぐちゃ曲がっていて、たまたま入力のサンプルによくヒットしているが、これはいわゆる過学習というものになる。統計モデリングとはモデル選択とパラメーター推定であるが、モデル選択を 1 次式か、3 次式か、40 次式にするか、そういうことをまず決める。そしてパラメーター、つまり係数がどれになるかを決める。ただし、そのときにあまりパラメーター数の多いモデルにすると過学習になる。今までの統計的モデリングあるいは昔の統計的機械学習は、モデル選択とパラメーター設定をやらないといけないのだが、モデル選択を間違えると、過学習してひどいことになるということがよく知られていた。その世界では、統計数理研究所の赤池先生が、赤池情報量基準や、いろいろな選択基準を作っていたが、この問題はずっと解けていなかった。しかし、深層学習というものが出てきて、どうも世の中の様子が変わったぞというような話である。

深層学習は 1 入力 1 出力でやっていたが、多次元にするのは全然問題がなく、多重回帰などをやるわけであるが、多次元といっても、統計的にやっている人たちの多次元は大した多次元ではなく、深層学習の多次元はものすごい多次元である。例えば 100 ピクセル×100 ピクセルのイメージを入れれば、それだけで 1 万個の入力がある。もしそれが RGB だったら、3 万個の入力があるような関数を考える。そのイメージの中に猫が入っているとか、入っていないといったものがこの出力になるなど、そのような入出力の関数である。この

ような関数を計算したいのだが入出力はとにかく非常に次元数が多い。しかも、ニューラルネットの一つ一つの線に重みが付くが、この重みの一つ一つがパラメーターになるので、パラメーターの数が桁違いに多い。1 万次元あったとすると、最初の convolution 層がどのくらいになるか、1 万の 8 倍ぐらいである。そのくらいの線になるが、この上にさらにフリーコネクテッド層があり、この層とこの層のすべてのノードがつながるから、そこも非常に大きな数になる。だから、桁違いに多いパラメーターになるが、多いからこそ実は任意の多次元の非線形関数を近似する能力があるわけである。ニューラルネットというのは、入力から出力、イメージを 1 つ与えて、そこに猫が写っているかどうかというのは関数である。それは恐らく超多次元の非線形の関数であるが、そういうものを近似する能力があるわけである。パラメーター数が多いから。だが、どうやらあまり過学習はしないらしい。それがなぜかはあまりよく分かっていないが、それによって実はモデル選択をあまり真面目に考えなくてもよいと、とにかく流してみると入出力をうまく学習してくれるようである。これは、この世界を知っている人には衝撃的なことである。松尾先生はこのこと、フィーチャーエンジニアリングをしなくてもよいことがそのインパクトだと言われているが、言っていることは同じことである。つまり、フィーチャーエンジニアリングというのはどのパラメーターを選ぶかということをやるという意味であるが、これはパラメーターを選ばなくてよい、モデルを選ばなくてよいということを言っているので同じことである。

そうはいっても、ネットワークの構造だけは選んでやったほうが学習が速く進むことが知られている。そもそもディープニューラルネットがはやったのはこれである。ImageNet というところで、イメージの画像を認識するタスクのコンペティションをやった。バンジョーの写真をいろいろ見せられ、バンジョーが写っているものを出力する関数を設計してほしいというものである。エラー率が 2010 年 28.2%、2011 年が 25.8%で、30%近い。ここに AlexNet というディープニューラルネットがでてきて、さらにエラー率が 10 ポイント以上も改善されたということで衝撃が走った。それ以降はずっとニューラルネットの独壇場になっている。また Human というものでやると 5%ぐらいのエラー率らしいが、人よりも精度の高い認識ができることが知られている。

このネットワークはどういうことかというと、入力はイメージの各ピクセルが全部入力のノードになっている。それを convolution 層と言って、9 ピクセル、あるいはもう少し幅を取ってもよいが、それを convolution して作った層があり、さらに Pooling 層があり、それから先により大域的なフィーチャーを集めていくというようなことをする。最後にアウトプットが出て、そのアウトプットが間違えると、誤差信号がバックプロパケーションされて少しずつ重みが修正されていくようなものになる。とにかくこういうものができて、これはすごいなと、イメージの世界ではこれで行けるなという話になった。

こういうネットワークを作ると、入力は非常に多次元であるが、固定サイズである。ところが、例えば時系列のデータのようにバーッと流れているものは固定サイズではないので、そういうのは困るねという話があった。そこにリカレント・ニューラルネットが出てきた。これは、1 個入力が来るたびに、ネットワーク全体をコピーするようなことをやるの

だが、そうすると、入力がパッパッパッパッと入ってきて、ここで End と言うと、それに対して答えを出す。これは何に使えるかというと、例えば自然言語の入出力に使える。

このようなことをやることにどのくらい意味があるのかよく分からないが、次のような論文がある。これを読ませる。「Sam walks into the kitchen. Sam picks up an apple. Sam walks into the bedroom. Sam drops the apple.」で、 apple はどこにあるかと。こういう入出力のペアをたくさん入れていって学習をさせると、bedroom にあることが見つかる。このようなものも学習できるようになっている。だから、画像のようなものは convolution ネット、それから自然言語のように長さが変わるものについてはリカレントネット、そういうものを使えば長さが変わるものもできる。これとこれを組み合わせれば画像を見て、自然言語でこの画像は何をしているということを示すようなネットワークも作られていく。

それから、恐らくもっと衝撃的なものは、Variational Autoencoder というものだと思う。今までは教師信号を入れて学習させるという話をしていたが、訓練データを教師なしでたくさん入れて学習させる。ただし、入力に対して、先のほうのノード数は少ないような層を作る。そうすると何が起きるかというと、例えば入力が猫らしい、飛行機らしいなど、そういうようなものがだんだんあらわれるようになってくる。なぜかというと、それは入力の統計的な性質をできるだけ小さいノード数で表現しようとするからである。それを変分自己符号化、Variational Autoencoder と呼ぶが、これを逆向きに使うと何ができるかというと、入力のパターン、入力の統計的な性質と同じ性質のアウトプットを出すことができる。しかも、ある任意の非線形の関数であるので、例えば非常に簡単な単位正規分布にするということをする。それを今度は後で 1 と出すことをすると、次のようなことができたりする。

当社の松元と言う若い社員が作ったものだが、いろいろな絵をたくさん読ませると何が起きるかというと、何々スタイル様のものというのがだんだん現れてくるような符号化がされる。この符合を 1 と生成させることによって、入力にこの猫、スタイルはこの絵とすると、右のようなアウトプットが出る。だから、同じ入力に対して、このスタイルで入れるとこの猫、このスタイルで出すとこの猫と、このようなことができる。それはなぜできるかというと、Autoencoder である。こういうところにスタイルに対する符号化がされていて、入力をたくさん入れると、私もよく分からないが、ゴッホ風や何々風というものが出ると、そのようなことができる。私らコンピュータサイエンティストの人は、このようなものは今まで見たこともないので、これが何に使えるのか、まだあまりよく分かっていないところにあると思う。

Alpha Go は、前回安宅委員もご説明になったが、これ一つはイメージだと思う。19×19 で、白か黒か何もないかという入力をインプットして、アウトプットは、まず 19×19 で、例えばこの盤面を見たらここに人間のプレーヤーだったら打つ確率が幾らかを学習させる。30 万局の人間のプレーヤーの棋譜を入れて、全部で数千万のオーダーの盤面があるわけであるが、ある盤面に対して、次にここに打ったということを教師データとして覚えさせる。そうすると、この盤面になったら人はどこに打ちそうかというようなニューラルネットワ

ークができる。それはある意味、人の直感を非常に表している。全体をパッと見て、ここでここだと思うという、ポリシーネットワークができる。だが、それだけではないところが Google Alpha Go の素晴らしいところで、それをベースに今度は自分同士で戦わせるという探索をする。それは最初に言った人工知能の means-End analysis である。探索と、こういう直感を合わせると、Alpha Go というのができる。だから、これは実は昔のコンピュータサイエンスと今の機械学習を非常にうまく合わせた成果だと言えると思うし、私たちに与えてくれる示唆が非常に大きいのではないかというような気がする。深層機械学習というのはこのようなものだということがお分かりになったと思う。

機械学習は帰納的にやるが、結果は結構確率的に出てくる。本来はモデル選択をしてパラメーター推定しなくてはいけなかったのであるが、深層学習が出てきてから、あまりモデル選択のことを気にしなくてもよくなった。だいぶ帰納的なインダクティブなシステムというのが我々にとって使いやすくなった。と、さあ、それでどういう結論が出るだろうかということで、次に、2 点目、後半に移る。

帰納的プログラミングとその帰結

そもそも私たちがこの 60 年､70 年ずっとたたき込まれてやってきたシステム設計のやり方というのは、いわゆるウォーターフォール開発というやつである。みずほ銀行がそれでさんざん苦労されているようであるが、何をやるかというと、要件定義があり、一生懸命やる。要件定義をモデリングして、それを段階的に詳細化、組み込み開発や V 字型開発と言ったりするが、段階的に詳細化していく。

なぜこういうことをするかというと、作ってみてから要件が間違っていたというと非常に大変だからである。要求工学など、そのような一分野があるが、とにかくこれをきちんとやろうと。それで、一回詳細化するたびにレビューして、そこは間違っていないねと言って次へ進むというのが、今までの私らがたたき込まれたシステム開発のやり方である。例えば自動運転をやりたいと言ったら、要するにセンサー入力を入れてアクチュエータ出力を出す関数を作るわけであるが、その関数の作り方として、こういう要件があり、こういうモデルだから、それを段階的に詳細化していって、モジュール化していって、画像認識をやるモジュール､空間認識をするモジュール､物理モデリングをするモジュールとか､そうやってだんだん詳細化していくというのが今までの物の作り方である。

自動運転を例にとって考えよう。なぜかというと、次のビデオを見せたいが、これは nVIDIA の GTC というカンファレンスで紹介されたものである｡彼らは車に載せる nVIDIA の GPU を売りたいのであるが、最初のうちはこのような感じである（笑)。人が 3,000 マイルをドライビングした入力出力を全部取っている。カメラが 2 つあるが、そのカメラの入力から、ハンドル、ブレーキ、アクセル操作を入出力の教師、訓練例だと思って学習する。そうすると、このようなマルチレーンや、中央線のないところ、舗装路でないところなど、こういうところを運転できるようになったという話である。

このシステムの作り方は入力出力のペアをトレーニングデータとして入れる。この場合

だと 3,000 マイル分の、カメラの入力に対して人が何をやったのかが教師データである。そのデータを入れてとにかく機械学習を回す。nVIDIA のチップで回したと思うが、それをやると、あとは自動運転システムが出てくる。それは何かというと、センサー入力に対してアクチュエータが出力を出す。その中にはモジュール化が一切ない。入力に対して出力、その間は一つのディープニューラルネットワークがやっている。このようなもので本当に車が動くのかと思われるかもしれないが、そこそこうまくいくと思う。さらに、この先に何があるかというと、要件定義というトレーニングデータセットである。

今年の 1 月に、私たち Preferred Networks が CES でデモをしたので、少しだけご覧いただく。銀色の車は、実はレゴマインドストームにプリウスの側をかぶせただけのものであるが、それが信号のない交差点をぶつからずに通り過ぎていくというデモである。赤い車は手で操縦していてぶつかったりするわけであるが、それでもほかの車はそれをうまくよけて通っているのがわかると思う。このポイントは何かというと、先ほどの機械学習でプログラミングされており、いわゆる V 字型開発や、要件定義など、そういうものは一切ない。

これをどのように作ったか。nVIDIA のものとは少し違う。実は最初にシミュレーターの上で学習をさせる。そしてレーストラックのようなものがあり、そこで車を動かす。ビデオの中で丸が付いている車が 32 方向にビームを出していて、そのビームが壁からはね返ってくるのを入力としている。センサー入力である。現在の速度、ハンドルの角度、少し前の自分の位置、そういうものを入力している。それでハンドルを右に、左に切る、アクセル、ブレーキを踏むなど、出力は 2 値の 5bit のアウトプットである。それで、入力に対して出力を 7 層のニューラルネットで深層学習する。つまり、273 次元に対して 5bit のアウトプットを出すような関数を学習しているのが、これのやり方である。そこまでは先ほどと同じであるが、問題は、その学習の訓練データをどのように作るかというと、訓練データを作る代わりに実際にシミュレーター上で動かしてみて、うまくいったら報酬を与える、ぶつかったらペナルティーと、そういうことをやる。最初はランダムに動くので、もがいているだけである。だが、そのうち前に進むと報酬がもらえると分かると、そういう強化信号が入ってくる。最終的にはこのようにスムーズに動くようになる。

よく見ていただくと分かるが、コーナーを曲がるときに車がアウトに膨らんでいる。実はそのほうが早く走れることがだんだん分かってきて、自分で見つけているわけである。だから、車にどのように走れとか、右にカーブしていったら右にハンドルを切れとか、そういうことは一切言っていない。そもそもこの 273 次元のビームが、これは右のビームだとか、左何度のビームだとか、そのようなことは一切教えていない。単にあの入力に対して何か起きたら報酬がもらえた、もらえなかったなど、そのようなことである。前回の安宅委員のプレゼンの中でブロック崩しがあったが、あれとやっていることは全く同じである。

先ほどの交差点のケースではカリキュラム学習をやる。最初はとにかく易しい環境から、何もないところでとにかくぐるぐる回ることを覚えることを延々とやる。いきなり交差点

をやると、時間が掛かって学習ができない。ある程度ぐるぐる回れるようになったら、今度は衝突したら駄目だということを覚え込ませる。突然その評価関数が変わるので非常に混乱するが、だんだん適応してきて、少なくともぶつからずに周回できるようになる。だいぶ学習できたら、今度はコースを難しくする。これで交差点を通るようにする。さらに、車を増やしていく。そのたびごとに環境が変わるとやはり皆とまどっているが、だんだん学習していって、最終的には 40 台の車がスムーズに動くことが見えると思う。これが強化学習である。深層学習は、トレーニングデータセットを非常にたくさん用意しないといけないが、実はシミュレーターで結果をシミュレートできるような世界では、トレーニングデータをつくる代わりに、シミュレーターの上でやって強化学習をすることができる。

さて、これは我々にとってどういう意味があるか。強化学習とは何かというと、別に訓練データを用意したわけではなく、やってみて駄目だったら教師信号を返すことをやる。そういう意味では、最初からシステムが動いているが、最初はぼろぼろである。それを試行錯誤でやってみるというようなことに対応していると思う。そうすると、仕様はどこに出てくるか。フィードバックする教師信号のところに仕様の要件定義の概念があるわけである。これは何か。我々オールドタイマーのコンピュータサイエンティストの言葉で言えば、これは要求定義を事後にやっていることに当たるのではないかというのが、もしかしたら非常にインパクトのあることなのではないかという気がする。

コンピュータサイエンスを長いことやっている人は、『オレゴン大学の実験』のカートゥーンを見たことがあると思う。お客様がこのようなものが欲しいと言ったとする。すると、プロジェクトリーダーはお客様は多分このようなものが欲しいのだろうと思う。アナリストはこうやってデザインし、プログラマーはこのようなコードを書き、営業はお客様にこのようなものができると言うし、プロジェクトのドキュメンテーションでは何もないが、実装されたのはこのようなもので、でも顧客に請求された金額はこうだと。だが、本当はお客様が欲しかったのはこれである、という話である。

こういうことが、実は今まで 60 年の IT 産業の歴史の中で、死屍累々である。要求定義は本当に難しい。最初にお客様がこのようなものが欲しいと言われたものが、そのままで行けた試しがない、そのような世界である。だが、私たちは、一度作ってしまったら元に戻るのは非常にコストが掛かるから、アップフロントの要求定義をきちんとやろうということを延々とたたき込まれてやってきた。だが、少し待てよと、強化学習とは何か、強化学習はやってみて駄目だったら、駄目と言って、それを修正するときはただで修正してくれるわけである。だとすれば、やってみればよいではないかと。事後要求定義に基づくシステムの作り方がもしかしてあるのではないかという気がしている。

最近の IT の世界は、アジャイルや DevOps という言葉がはやっている。アジャイルや DevOps とは何かというと、お客様の要求は変わるのだから少しずつ作ろうという考え方である。そういうものの先に、実はやってみて駄目だったら、これは違うと言うと、システムが自動的に修正してくれるようなものがあるのではないかという気がする。それが実は深層学習が我々に与えるインパクトの 1 つではないか。

機械学習でシステム開発の常識がどう変わるかと言うと、演繹的から帰納的、事前要求定義から事後要求定義、確定的から確率的になる。また、分割統治をするのではなく、全体を見るような計算をするとか、アナログ計算をする可能性があるなど、いろいろなことがある。

もう一つ、今までのコンピュータサイエンスでできたものはどこに価値があるか。私らから見ると、一番大きな価値は、ソースコードは知財である。だが、今の深層学習の作り方だと、ソースコードはない。では、ソースコードに代わるものは何かというと、どうやら学習済みのモデルがソースコードに対応するもののような気がする。私が今日皆さんに考えていただきたいのは、ここである。

学習済みモデルとは何だったのかともう一回考えてみると、少なくとも深層学習の世界では入力があって出力がありネットワークがある。ネットワークのコンボルーションネット、リカレントネットなど、トポロジーを決めないといけない。トポロジーの中に、さらに各リンクの重みがあり、これが何千万というオーダーのフローティングポイントで数値が並んでいるのであるが、これが学習済みモデルの正体である。どのくらいリンクの数があるかというと、物によっては 160billion parameters である。160billion というと想像できない感じがするが、とにかく多い。これが多分知財だろうと。確かに価値があるものである。なぜなら、それだけ訓練データを集めてきて時間を掛けている。先ほどの Alpha Go の計算をするのに、Google Cloud を使うと 30 億円掛かったはずだという計算もある。だから、とにかくノウハウもあるし、コストもたくさん掛かっている。それが知財のはずである。

それだけ金を掛けたのだから再利用しようという話になると思う。再利用というのは、どのようにするかというと、一番簡単なのはできた学習済みモデルをそのままコピーする。それをいったん外部ファイルに書き出して、それをコピーして次のマシンに入れて、そこで実行するというのが分かりやすい。これが普通の典型的な知財の使い方である。それから、これは機械学習であるので、そのモデルに、少し違う領域の訓練データをさらにプラスアルファしている。簡単なモデルを作り、もう少し訓練データを足して、よりファインチューンされたモデルを作るのがファインチューニングという考え方である。転移学習という言葉で呼ばれたりするが、新たな訓練データを加えて、新たなタスク、ただし、多分似たタスクを利用するというようなことである。また、今まではホワイトボックスの利用だが、ブラックボックスの再利用と言って Google の API を公開している。API に対して入力をランダムに入れると出力が出る。全部集めると、これは私にとっての訓練データである。その訓練データを使って新たなモデルを訓練したら、中身は全然違うものになる。だが、恐らく入出力の対応関係はほとんど同じである。だから、これはリバースエンジニアリングせずに、外から振る舞いだけを見てシステムを事実上コピーすることができるようになると思う。今までの私たちのコンピュータサイエンスの世界では、こういう概念はなかったような気がする。それから、アンサンブルという考え方がある。結果は確率的だという話をしたが、ブラックボックスのモデルがたくさんある中で、同じタスクのものがあ

ったら、それをかき集めてきて多数決をするというのがアンサンブルという考え方であるが、その結果として、同じタスクであるが、より精度の高いモデルを得ることができるようになると思う。

実はこの辺りの再利用は、あまりよく分かっていない。こういうものに対して知財をどのように守ったらよいのか、実はあまりよく分かっていない。全然議論されていない。恐らくモデルができると、その派生モデル、そのさらに派生モデルと、非常にたくさんのモデルができていって、その間のエコシステムがいずれできるようになる。だから、そのデータのオーナーや、モデルのオーナーや、そのモデルの二次モデルを作る人や、それを最後に使う人や、そのモデルを作るときには当然新たなクレンジングしたデータを入れて計算をしないといけないし、それでテストしないといけない。そういうノウハウがたくさん入る。さらに、こういうモデル間の依存関係、このモデルはあのモデルから来たと、だから、これを使ったときにはあちらに金を払ってとか、そのようなことをやる市場マーケットプレイスのプレーヤーが出てくるのではないかと思うわけである。

実はこれを戦略としてやろうとしているのが、私の見るところでは Microsoft や、Google である。Google は Tensor Flow を出しているが、Tensor Flow は実は Google のデータセンターの中で動かすと一番気持ちよく動くわけである。だから、皆さんデータを Google のデータセンターに上げて、そこでモデルを作る。モデルがそこにあるから、派生モデルもその上で作れば、気持ちよく使えて、その中にどんどんたまってくる。そうすると、いずれこういうモデル間の依存関係、来歴、課金などの概念が出てきたときに、皆さん、モデルが全部当社のデータセンターの中にあるから、それの課金は全部任せてほしいということが Google はできるはずである。Microsoft はクリアに、それは当社の戦略だと言っている。そのようなことを彼らに決められてしまうと嫌だなと思って、実は辻井先生と一緒に機械学習の利用促進勉強会を作って、先月、人工知能大会でワークショップをやったが、こういう学習済みモデルの知財をどのように扱っていくかについて、今議論を始めているところである。できればこの手のルール作りを、US からこれがディファクトだとやってくる前に、我々の中で議論を深めていきたいなと思っている。

今日は、深層学習・機械学習がどのようにインパクトを与えるかについて 2 点話した。事後要求定義の話と、知財の話である。きっとそれ以外にもいろいろあると思う。こういう機械学習によるシステム開発は私らにどういうインプリケーションがあるのかをぜひこれから考えていきたいので、それを皆さんとご議論させていただければと思う。