DSC 102
Systems for Scalable Analytics

Arun Kumar

Topic 6: Deep Learning Systems
Outline

❖ Rise of Deep Learning Methods
❖ Deep Learning Systems: Specification
❖ Deep Learning Systems: Execution
❖ Future of Deep Learning Systems
Unstructured Data Applications

❖ A lot of emerging applications need to deal with unstructured data: text, images, audio, video, time series, etc.
❖ **Examples:** Machine translation, radiology, automatic speech recognition, video surveillance, exercise activity analysis, etc.

❖ Such data have low level formatting: strings, pixels, temporal shapes, etc.
❖ Not intuitive what the *features* for prediction should be
Past Feature Engineering: Vision

- Decades of work on in machine vision on *hand-crafted* featurization based on crude heuristics
- **Examples:**
  - Histogram of Oriented Gradient (HOG)
  - Fisher Vectors
  - Scale-invariant Feature Transform (SIFT)

![Diagram of feature extraction and classification](image)

*Fig. 3. Histogram of oriented gradient extraction from face.*

Histogram of Oriented Gradient (HOG)
Unfortunately, such ad hoc hand-crafted featurization schemes had major disadvantages:

- *Loss of information* when “summarizing” the data
- *Purely syntactic* and lack “semantics” of real objects

Similar issues occur with text data and hand-crafted text featurization schemes such as Bag-of-Words, parsing-based approaches, etc.

**Q:** *Is there a way to mitigate above issues with hand-crafted feature extraction from such low-level data?*
Learned Feature Engineering

❖ **Basic Idea:** Instead of hand-defining summarizing features, exploit some *data type-specific invariants* and construct *weighted feature extractors*

❖ **Examples:**
- Images have *spatial dependency* property; not all pairs of pixels are equal—nearby ones “mean something”
- Text tokens have a mix of local and global dependency properties within sentence—not all words can go in all locations
- Deep learning models “bake in” such data type-specific invariants to enable *end-to-end learning*, i.e., learn weights using ML training from (close-to-)raw input to output and avoid non-learned feature extraction as much as feasible
Neural Architecture as Feature Extractors

- Different invariants baked into different deep learning models
- **Examples:** CNNs

Convolutional Neural Networks (CNNs) use *convolutions* to exploit invariants and learn hierarchy of relevant features from images
Neural Architecture as Feature Extractors

- Different invariants baked into different deep learning models
- Examples: LSTMs

---

Cell sensitive to position in line:

*The male importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action: the one Kutuzov and the general mass of the army demanded—namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all—carried on by vis inertiae—pressed forward into boats and into the ice-covered water and did not, surrender.*

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be amused by the same desire.

Cell that robustly activates inside if statements:

```c
int sig = next_signal(pending, mask);
if (sig) {
    if (current->notify) {
        clear_thread_flag(TIF_SX_PENDING);
        return 0;
    }
    collect_signal(sig, pending, info);
    return sig;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```c
char *str = ((bufp || (len >= 0)) || (len > *remain)) ? return ERR_PTR(-EINVAL); + defines the longest valid length
```

Long Short Term Memory Networks (LSTMs) use memory cells to exploit invariants in textual/sequence data processing.
Neural Architecture as Feature Extractors

- It is also possible to mix and match learned feature extractors in deep learning!
- **Example:** CNN-LSTMs for time series

CNNs extract temporally relevant features locally, while LSTMs learn more global behavior; whole neural architecture (CNN-LSTM) is trained *end-to-end*
Neural Architecture as Feature Extractors

❖ It is also possible to mix and match learned feature extractors in deep learning!
❖ Example: CNN-LSTMs for video

CNNs extract visually relevant features at each time step, while LSTMs learn over those features across time; whole neural architecture (CNN-LSTM) is trained end-to-end.
Versatility of Deep Learning

- **Versatility** is a superpower of deep learning:
  - Any data type/structure as input and/or output
  - Dependencies possible within input/output elements

### Examples of Versatility:

- **Click Prediction**
- **Image Captioning**
- **Sentiment Prediction**
- **Machine Translation**
- **Video Surveillance**
Pros and Cons of Deep Learning

❖ All that versatility and representation power has costs:
  ❖ “Neural architecture engineering” is the new feature engineering; painful for data scientists to select it! 😊
  ❖ Need large labeled datasets to avoid overfitting
  ❖ High computational cost of end-to-end learning and training of deep learning models on large data

❖ But pros outweigh cons in most cases with unstruct. data:
  ❖ Substantially higher prediction accuracy over hand-crafted feature extraction approaches
  ❖ Versatility enables unified analysis of multimodal data
  ❖ More compact artifacts for model and code (e.g., 10 lines in PyTorch API vs 100s of lines of raw Python/Java)
  ❖ Model predictable resource footprint for model serving
Outline

❖ Rise of Deep Learning Methods
❖ Deep Learning Systems: Specification
❖ Deep Learning Systems: Execution
❖ Future of Deep Learning Systems
Deep Learning Systems

❖ **Main Goals:**
  ❖ Make it *easier to specify* complex neural architectures in a higher-level API (CNNs, LSTMs, Transformers, etc.)
  ❖ Make it *easier to train* deep nets with SGD-based methods
  ❖ Also these goals to a lower extent:
    ❖ Scale out training easily to multi-node clusters
    ❖ Standardize model specification and exchange
    ❖ Make it easier to deploy trained models to production
  ❖ Highly successful: enabled 1000s of companies and papers!
Deep Learning Systems APIs

- TensorFlow (TF) is now widely used in both industry and academic research; PyTorch is second most popular.

Most data scientists prefer the Python API. Higher-level APIs are more succinct but more restrictive in terms feature transformations.

Under the covers, TF compiles deep net specification to C++-based “kernels” to run on various processors.
Neural Computational Graphs

❖ Abstract representation of neural architecture and specification of training procedure

❖ Basically a dataflow graph where the nodes represent operations in DL system’s API and edges represent tensors

Q: What is the analogue of this produced by an RDBMS when you write an SQL query?
Model Exchange Formats

- **Basic Goal:** Portability of model specification across systems
- These are domain-specific file formats that prescribe how to \textit{(de)}serialize the neural architecture and training options
  - Dataflow graph typically human-readable, e.g., JSON
  - Weight matrices typically stored in binary format
Keras is an even higher-level API that sits on top of APIs of TF, PyTorch, etc.; popular in practice.

- TensorFlow recently adopted Keras as a first-class API.
- More restrictive specifications of neural architectures; trades off flexibility/customization to get lower usability barrier.
- Perhaps more suited for data scientists than lower-level TF or PyTorch APIs (more suited for DL researchers/engineers).
- AutoKeras is an AutoML tool that sits on top of Keras to automate neural architecture selection.
Outline

❖ Rise of Deep Learning Methods
❖ Deep Learning Systems: Specification
❖ Deep Learning Systems: Execution
❖ Future of Deep Learning Systems
Recall that DL training uses SGD-based methods.

Regular SGD has a simple update rule:

$$W^{(t+1)} \leftarrow W^{(t)} - \eta \nabla \tilde{L}(W^{(t)})$$

Often, we can converge faster with cleverer update rules, e.g., adapt the learning rate over time automatically, exploit descent differences across iterates ("momentum"), etc.

Popular variants of SGD: Adam, RMSProp, AdaGrad.

But same data access pattern at scale as regular SGD.

TF, PyTorch, etc. offer many such variants of SGD.
AutoDiff for Backpropagation

- Recall that unlike GLMs, neural networks are compositions of functions (each layer is a function)
- Gradient not one vector but multiple layers of computations

\[
\nabla \tilde{L}(W) = \sum_{i \in B} \nabla l(y_i, f(W, x_i))
\]

- Backpropagation procedure uses calculus chain rule to propagate gradients through layers
- **AutoDiff**: DL systems handle this symbolically and automatically!
AutoDiff and simpler APIs for neural architectures have led to an “Cambrian explosion” of architectures in deep learning!

**Software 2.0:** Buzzword to describe deep learning

**Differentiable programming:** New technical term in PL field to characterize how people work with tools like TF & PyTorch

- Programmer/developer has to write software by composing layers that can be *automatically differentiated* by AutoDiff and is *amenable to SGD-based training*

- Different from and contrasted with “imperative” PLs (C++, Java, Python), “declarative” PLs (SQL, Datalog), and “functional” PLs (Lisp, Haskell)
Industrial-strength tools like TF and MXNet (Amazon’s equivalent of TF) offer some software to make it easier to deploy trained deep nets; custom APIs.
Outline

❖ Rise of Deep Learning Methods
❖ Deep Learning Systems: Specification
❖ Deep Learning Systems: Execution
❖ Future of Deep Learning Systems
Future of Deep Learning Systems

- From the systems standpoint, 4 main active lines of work:
  - **Specification:** Google and Facebook are actively looking into developing a new PL for differentiable programming!
  - **Execution:** Better scalability to multi-node execution on clusters and clouds; better scalability to very large models
  - **Hardware:** Custom processors for training and inference keep popping up! Reduce energy use and monetary costs
  - **Environments:** Google/TF in particular is expanding footprint of DL training/inference software to interesting new environments, e.g., smartphones and IoT (TF Lite and “federated learning”) and Web browsers (TF.JS)