Rethinking Deep Learning: Architectures and Algorithms

George A. Constantinides

Circuits and Systems Group, Imperial College London

April 5, 2019
Outline

- The supervised learning problem
- Good functions for inference
- Typed graphs and hardware accelerators
- Binarised NNs are functionally complete
- Generalisation: Lipschitz topologies, coding, functions
- Bridging Boolean networks and DNNs: LUTNet
- Some open questions
Rethinking Deep Learning
Problem Definition

Consider a class of functions \( f(w; x) \).

- \( w \) is the parameter (choose once, at training time)
- \( x \) is the activation (changes for each inference)

**Definition (Supervised Training [Sch16])**

\[
\argmin_{w \in \mathcal{W}} \mathbb{E}_{(x, y) \in (\mathcal{X}, \mathcal{Y})} \{ \ell(f(w; x), y) \}
\]

- distribution of \((\mathcal{X}, \mathcal{Y})\) only accessible through sampling
- \( \ell \) of practical interest may be difficult to optimise

So:

\[
w^* = \argmin_{w \in \mathcal{W}} \sum_{i=1}^{n} \ell'(f(w; x_i), y_i)
\]
What class of functions $f(w; x)$ to use?

A Recipe for Good Inference Functions

- Sufficiently general / expressive
- Cheap to compute
- Generalise well: once $w^*$ is selected, $f(w^*; x)$ should also tend to perform well over the data $x$ encountered ‘in the wild’
- Easy to learn: the process of selecting $w^*$ such that $f(w^*; x)$ performs well over the training data $(x_i, y_i)$ should be efficient

Strang [Str18] argues for continuity as key to generalisation.
Typed Graphs: Neural Networks or Circuits?

Acknowledgements to Erwei Wang
Binarised Neural Networks (BNNs)

How to make the hardware simpler? BNNs take an extreme approach to quantization [CHS+16].

A classical node function is $\mathbb{R}^{n+1} \times \mathbb{R}^n \rightarrow \mathbb{R}$ given by $(w, c; x) \mapsto \sigma(w^T x + c)$ where $\sigma(\cdot)$ is a sigmoid.

**BNN Node Function**

Let $f : \{-1, +1\}^n \times \mathbb{Z} \times \{-1, +1\}^n \rightarrow \{-1, +1\}$ be given by

\[
(w, c; x) \mapsto \begin{cases} +1 & \text{if } w^T x \geq c, \\ -1 & \text{otherwise.} \end{cases}
\]

General perception: quantisation to 8-bit or 4-bit is OK, below that is problematic, except in special cases.
Theorem

**Binarised Neural Networks are functionally complete.**

Proof.

Let \( \phi : \mathbb{B} \to \{-1, +1\} \) be defined by \( \bot \mapsto -1, \top \mapsto +1 \). The following hold:

\[
\begin{align*}
  x \land y & \iff \phi^{-1} \circ \text{BNN} \left((+1, +1), +2; (\phi(x), \phi(y))\right) \\
  x \lor y & \iff \phi^{-1} \circ \text{BNN} \left((+1, +1), 0; (\phi(x), \phi(y))\right) \\
  \neg x & \iff \phi^{-1} \circ \text{BNN} \left((-1), +1; \phi(x)\right)
\end{align*}
\]

Completeness then follows from completeness of \( \{\land, \lor, \neg, \bot, \top\} \).
Corollary

*Accuracy-optimal network topology depends on finite-precision datatype.*

Related Observations in the Deep Learning Literature

- **Venkatesh et al.:** “The data shows that low-precision networks provide better accuracy as the network depth increases.” [VNM17].
- **Su et al.:** “we found that in MNIST classification, 1-bit parameter networks require more operations and connections to compensate the accuracy loss” [Su18].
Rethinking Topologies for Discrete Inference

**Definition**

Suppose $f : X \to Y$, where $X$ is equipped with a metric $d$ and $Y$ is equipped with a metric $e$. The function $f$ is $k$-Lipschitz if for all $a, b \in X$, $e(f(a), f(b)) \leq kd(a, b)$.

Which networks give rise to $k$-Lipschitz functions for small $k$?
Consider as function $\text{ADD} : \mathbb{B}^{2n+1} \rightarrow \mathbb{B}^{n+1}$.

Using $\varphi : \mathbb{B} \rightarrow \{0, 1\}$ given by $\bot \mapsto 0$, $\top \mapsto 1$, consider an encoding of the inputs:

$$w_k(x) = \sum_{i=0}^{k-1} \varphi(x_i)2^i.$$ 

$$d(\cdot) = |w_n(a) - w_n(a')| + |w_n(b) - w_n(b')| + |\varphi(c) - \varphi(c')|$$

$$e(\cdot) = |w_{n+1}(c, s_{n-1}, \ldots, s_0) - w_{n+1}(c', s'_{n-1}, \ldots, s_0)|$$
ADD : \( (\mathbb{B}^{2n+1}, d) \rightarrow (\mathbb{B}^{n+1}, e) \) is 1-Lipschitz.

Now consider the function NEWFUNC we would get by replacing the FA blocks.

NEWFUNC : \( (\mathbb{B}^{2n+1}, d) \rightarrow (\mathbb{B}^{n+1}, e) \) is not \( k \)-Lipschitz for any \( k < 2^n \).
ADD : $(\mathbb{B}^{2n+1}, d) \rightarrow (\mathbb{B}^{n+1}, e)$ is 1-Lipschitz.

Now consider the function NEWFUNC we would get by replacing the FA blocks.

NEWFUNC : $(\mathbb{B}^{2n+1}, d) \rightarrow (\mathbb{B}^{n+1}, e)$ is not $k$-Lipschitz for any $k < 2^n$. 
Practical Progress
LUTNet: Deep Learning for FPGAs

Intel Stratix IV (courtesy of Intel)
LUTnet: The Idea

A first look at bridging DNNs and Boolean networks.

BNNs: A reminder

\[ y = \sigma \left( \sum_{n=1}^{N} w_n x_n \right) \]

LUTs are used for scalar products – Boolean XNOR.

The LUTNet Approach

- Replace scalar products \((w; x) \mapsto w^T x\) by \(\mathbb{B}^{2^K} \times \{-1, +1\}^K \rightarrow \{-1, +1\}\)
  - Strict generalisation: embrace nonlinearity and extra inputs.
- Retrain with SGD: learn new Boolean functions of nodes.
- Prune network – area halved [WDCC19].
Conclusions

- Designing discrete classifiers (all practical classifiers) by quantising continuous classifiers can be suboptimal.
- Extreme quantisation can recover any finite classifier.
- ... so topology and datatypes are intimately connected.
- Generalisation and continuity are closely linked.
- Topology, metrics / coding and leaf functionality together impact continuity of function.
- LUTNet: It is possible to learn arbitrary Boolean node functions for a predefined coding and with limited topological changes, resulting in considerable area savings.
Conjectures and Open Questions

**Conjecture**
Future efficient neural network topologies will be driven by both the topology of the data and by the nature of the finite representation of the activations.

**Open Question**
What input and output codings are commensurate with the properties of good inference functions, and how do they depend on the probability measure?

**Open Question**
Given metrics, a Lipschitz constant $k$ and a network topology, how to characterise functions implemented only from functional decouplings?
References


