FastInference – Applying Large Models on Small Devices

SFB876-Summerschool2020

Sebastian Buschjäger
September 3, 2020

TU Dortmund University - Artificial Intelligence Group - Collaborative Research Center 876
Why inference matters

So far Autopilot for self-driving cars are in reach
Tesla Use Deep Learning for image recognition and steering
Why inference matters

So far Autopilot for self-driving cars are in reach
Tesla Use Deep Learning for image recognition and steering

But Deep Learning is very energy and resource hungry
Tesla Use custom inference chip which requires 57W
Why inference matters

So far Autopilot for self-driving cars are in reach
Tesla Use Deep Learning for image recognition and steering

But Deep Learning is very energy and resource hungry
Tesla Use custom inference chip which requires 57 W

What does this mean for the environment?

- All cars in Germany combined travelled 630 Billion Kilometers in 2018
- The average speed of personal cars in Germany is ?
Why inference matters

So far Autopilot for self-driving cars are in reach

Tesla Use Deep Learning for image recognition and steering

But Deep Learning is very energy and resource hungry

Tesla Use custom inference chip which requires 57\,W

What does this mean for the environment?

- All cars in Germany combined travelled 630 Billion Kilometers in 2018
- The average speed of personal cars in Germany is 45\,km/h
Why inference matters

So far Autopilot for self-driving cars are in reach

**Tesla** Use Deep Learning for image recognition and steering

But Deep Learning is very energy and resource hungry

**Tesla** Use custom inference chip which requires $57\,\text{W}$

What does this mean for the environment?

- All cars in Germany combined travelled 630 Billion Kilometers in 2018
- The average speed of personal cars in Germany is 45km/h

Thus Using the autopilot in all trips requires 0.79 TWh

For reference The largest run-of-the-river hydroelectricity plant in Germany produces 0.79 TWh
So far Search engines are the main entry point into the internet
Bing.com Second largest search engine behind google
  Ranking algorithm is 90% ML powered, specifically Gradient Boosted Trees
So far Search engines are the main entry point into the internet
Bing.com Second largest search engine behind google
  Ranking algorithm is 90% ML powered, specifically Gradient Boosted Trees

Trees are simple, right? Assume we require 1 ms rank a single query

What does this mean for the environment?

- 12 billion search queries per month world-wide, 4,480,287 queries per second world-wide
- 4481 Intel i7-7700K CPU required each using approx. 91W
So far Search engines are the main entry point into the internet
Bing.com Second largest search engine behind google
   Ranking algorithm is 90% ML powered, specifically Gradient Boosted Trees

Trees are simple, right? Assume we require 1 ms rank a single query

What does this mean for the environment?

- 12 billion search queries per month world-wide, 4.480.287 queries per second world-wide
- 4481 Intel i7-7700K CPU required each using approx. 91W

Thus Running these CPUs for a year requires 3.57 TWh

For reference The largest black coal plant in Germany is produces 3.43 TWh
Recap ML basics and Goal

**Usually** Formalize problem mathematically $\rightarrow$ formulate algorithm $\rightarrow$ implement solution

**Note** Some problems are hard to be exactly formalized
Recap ML basics and Goal

**Usually** Formalize problem mathematically $\rightarrow$ formulate algorithm $\rightarrow$ implement solution

**Note** Some problems are hard to be exactly formalized

**Example** Identify cats vs dogs on the given pictures

What is a mathematical representation of a cat?
Idea Formalize given problem by labelled examples ("the data")
**Idea** Formalize given problem by labelled examples ("the data")

- Targets, e.g. $\mathcal{Y} = \{-1, +1\}$ (binary classification), $\mathcal{Y} = \mathbb{R}$ (regression)
Recap ML basics and Goal

**Idea** Formalize given problem by labelled examples ("the data")

- Targets, e.g. \( \mathcal{Y} = \{-1, +1\} \) (binary classification), \( \mathcal{Y} = \mathbb{R} \) (regression)
- Data \( \mathcal{D} = \{(\tilde{x}_1, y_1), \ldots, (\tilde{x}_N, y_N) \mid (\tilde{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\} \)

Machine Learning

Choose model which fits data best

\[ f^* = \arg\min_{f \in F} \frac{1}{N} \sum_{(x, y) \in \mathcal{D}} \ell(f(x), y) \]

Our goal in this talk

Run model \( f^* \) on resource-constraint devices
Recap ML basics and Goal

**Idea** Formulate given problem by labelled examples ("the data")

- **Targets**, e.g. \( \mathcal{Y} = \{-1, +1\} \) (binary classification), \( \mathcal{Y} = \mathbb{R} \) (regression)
- **Data** \( \mathcal{D} = \{ (\tilde{x}_1, y_1), \ldots, (\tilde{x}_N, y_N) \mid (\tilde{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} \} \)
- **Model** \( f \in \mathcal{F} = \{ f : \mathcal{X} \rightarrow \mathcal{Y} \}, \text{ e.g. } f(x) = \langle x, w \rangle \)
Recap ML basics and Goal

**Idea** Formalize given problem by labelled examples (“the data”)

- **Targets, e.g.** $\mathcal{Y} = \{-1, +1\}$ (binary classification), $\mathcal{Y} = \mathbb{R}$ (regression)
- **Data** $\mathcal{D} = \{ (\bar{x}_1, y_1), \ldots, (\bar{x}_N, y_N) \mid (\bar{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} \}$
- **Model** $f \in \mathcal{F} = \{ f : \mathcal{X} \to \mathcal{Y} \}$, e.g. $f(x) = \langle x, w \rangle$
- **Loss** $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
Recap ML basics and Goal

**Idea** Formalize given problem by labelled examples ("the data")
- Targets, e.g. $\mathcal{Y} = \{-1, +1\}$ (binary classification), $\mathcal{Y} = \mathbb{R}$ (regression)
- Data $\mathcal{D} = \{(\tilde{x}_1, y_1), \ldots, (\tilde{x}_N, y_N) \mid (\tilde{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$
- Model $f \in \mathcal{F} = \{f : \mathcal{X} \rightarrow \mathcal{Y}\}$, e.g. $f(x) = \langle x, w \rangle$
- Loss $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$

**Machine Learning** Choose model which fits data best

$$f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{(x, y) \in \mathcal{D}} \ell(f(x), y)$$
Recap ML basics and Goal

**Idea** Formalize given problem by labelled examples (“the data”)
- Targets, e.g. $\mathcal{Y} = \{-1, +1\}$ (binary classification), $\mathcal{Y} = \mathbb{R}$ (regression)
- Data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \mid (\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$
- Model $f \in \mathcal{F} = \{f : \mathcal{X} \rightarrow \mathcal{Y}\}$, e.g. $f(x) = \langle x, w \rangle$
- Loss $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$

**Machine Learning** Choose model which fits data best

$$
    f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \ell(f(x), y)
$$

**Our goal in this talk** Run model $f^*$ on resource-constraint devices
Recap What models are we talking about?

- Perform matrix-vector multiplication
- Apply non-linear activation after each layer
- Usually trained with stochastic gradient descent
Recap What models are we talking about?

Decision Tree (Ensembles)

- Start at root node and compare $x_i \leq t$
- Traverse tree until leaf is found
- Usually used in ensembles $f(x) = \sum_{i=1}^{T} w_i h_i(x)$
Recap Memory Hierarchy and Instruction Set Architecture

CPU 1
  ↓
Cache 1
  ↓
Shared Cache
  ↓
Main memory

CPU 2
  ↓
Cache 2
  ↓
Shared Cache
  ↓
Main memory
Recap Memory Hierarchy and Instruction Set Architecture
Recap Memory Hierarchy and Instruction Set Architecture

FastInference – Applying Large Models on Small Devices by Sebastian Buschjäger@SFB876-Summerschool2020
## Recap Instruction Set Architecture

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy [pJ]</th>
<th>Relative Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bit integer ADD</td>
<td>0.03</td>
<td>0.3</td>
</tr>
<tr>
<td>32 bit integer ADD</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>8 bit integer MULT</td>
<td>0.2</td>
<td>2</td>
</tr>
<tr>
<td>16 bit float ADD</td>
<td>0.4</td>
<td>4</td>
</tr>
<tr>
<td>32 bit float ADD</td>
<td>0.9</td>
<td>9</td>
</tr>
<tr>
<td>16 bit float MULT</td>
<td>1.1</td>
<td>11</td>
</tr>
<tr>
<td>32 bit integer MULT</td>
<td>3.1</td>
<td>31</td>
</tr>
<tr>
<td>32 bit float MULT</td>
<td>3.7</td>
<td>37</td>
</tr>
<tr>
<td>Cache access</td>
<td>5-50</td>
<td>50-500</td>
</tr>
<tr>
<td>DRAM access</td>
<td>320-640</td>
<td>3200-6400</td>
</tr>
</tbody>
</table>
Recap What devices are we talking about?

<table>
<thead>
<tr>
<th>CPU</th>
<th>Clock</th>
<th>RAM</th>
<th>Word size</th>
<th>Floating point</th>
<th>SIMD</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP 430</td>
<td>1 core 16 Mhz</td>
<td>0.5 KB</td>
<td>16 bit inst.</td>
<td>no</td>
<td>no</td>
<td>5.7 mW</td>
</tr>
<tr>
<td>MSP 430</td>
<td>1 core 20 Mhz</td>
<td>10 KB</td>
<td>16 bit inst. 32 bit mult.</td>
<td>no</td>
<td>no</td>
<td>15 mW</td>
</tr>
<tr>
<td>ARM1176JZF-S</td>
<td>1 core 1 Ghz</td>
<td>512 MB</td>
<td>32 bit</td>
<td>yes</td>
<td>limited</td>
<td>0.7 W</td>
</tr>
<tr>
<td>Cortex A53</td>
<td>4 cores 1.5 Ghz</td>
<td>512 MB</td>
<td>64 bit</td>
<td>yes</td>
<td>yes</td>
<td>1.5 W</td>
</tr>
<tr>
<td>i7-7700K</td>
<td>4 cores 4.5 Ghz</td>
<td>8-64 GB</td>
<td>64 bit</td>
<td>yes</td>
<td>yes</td>
<td>91 W</td>
</tr>
</tbody>
</table>
How to approach inference

**Recall** Total Energy = Energy per Time · Time

\[ W = P \cdot t \]
**Recall** Total Energy = Energy per Time \cdot Time

\[ W = P \cdot t \]
How to approach inference

Recall Total Energy = Energy per Time \cdot Time

\[ W = P \cdot t \]

Required Energy

Power consumption of hardware

Inference time

For Inference

Total Energy = Hardware used \cdot Model executed
1) Train different model
   Use models which are more resource-friendly
1) **Train different model**
Use models which are more resource-friendly

2) **Change model after training**
Train complex model first, then change it to be more resource-friendly
How to approach inference (2)

1) **Train different model**
   Use models which are more resource-friendly

2) **Change model after training**
   Train complex model first, then change it to be more resource-friendly

3) **Use hardware differently**
   Use special hardware features during inference
1) **Train different model**
Use models which are more resource-friendly

2) **Change model after training**
Train complex model first, then change it to be more resource-friendly

3) **Use hardware differently**
Use special hardware features during inference

4) **Use different hardware**
Design new entirely chips
How to approach inference (2)

1) Train different model
Use models which are more resource-friendly

2) Change model after training
Train complex model first, then change it to be more resource-friendly

3) Use hardware differently
Use special hardware features during inference

4) Use different hardware
Design new entirely chips
Idea 1: Train different model

\[ f^* = \arg \min_{f \in \mathcal{F}} \frac{1}{N} \sum_{(x, y) \in \mathcal{D}} \ell(f(x), y) + \lambda R(f) \]
Idea 1: Train different model

Use resource-friendly model class $\mathcal{F}$ directly

$\Rightarrow$ Guaranteed resource consumption, but maybe weak loss

Guide selection via regularization

$\Rightarrow$ Direct trade-off between loss and model complexity via $\lambda$

$$f^* = \arg \min_{f \in \mathcal{F}} \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \ell(f(x), y) + \lambda R(f)$$
Example 1: Binarized Neural Networks

Usually $\mathcal{F} = \{\text{NN of some architecture with weights from } \mathbb{R}\}$
Example 1: Binarized Neural Networks

**Usually** \( \mathcal{F} = \{ \text{NN of some architecture with weights from } \mathbb{R} \} \)

**Courbariaux et al. 2015, Hubara et al. 2016, Rastegari et al. 2016**

Binarized Neural Networks: \( \mathcal{F} = \{ \text{NN of some architecture with weights from } \{-1, 1\} \} \)
Example 1: Binarized Neural Networks

Usually $\mathcal{F} = \{\text{NN of some architecture with weights from } \mathbb{R}\}$

Binarized Neural Networks: $\mathcal{F} = \{\text{NN of some architecture with weights from } \{-1, 1\}\}$

Then Train on GPU, but use small-devices for inference

- BNNs use weights from $\{-1, 1\}$ instead of $\mathbb{R}$ which can be stored in bool instead of float
- Inference of BNNs only requires bool and integer operations
- Requires $\approx 32$ times less memory
Example 1: Binarized Neural Networks (2)

**Typical Structure** VGG-Like architectures

```
Input → Conv → BN → HTanH → MP → FC
```

Assume Inputs are already boolean or integer

<table>
<thead>
<tr>
<th>Operation</th>
<th>Parameter</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>$\cdot s + b$</td>
<td>$s \in \mathbb{R}$</td>
</tr>
<tr>
<td>HTanH</td>
<td>2</td>
<td>${x &gt; 0} - 1$, ${-1, 1}$</td>
</tr>
<tr>
<td>MP</td>
<td>max</td>
<td>Input type preserved</td>
</tr>
<tr>
<td>Conv</td>
<td>$\sum_{i} \sum_{j} w_{i,j} \cdot x_{i+k, j+l}$</td>
<td>$w_{i,j} \in {-1, 1}$</td>
</tr>
<tr>
<td>FC</td>
<td>$\sum_{i} w_{i} \cdot x_{i}$</td>
<td>$w_{i,j} \in {-1, 1}$</td>
</tr>
</tbody>
</table>

FastInference – Applying Large Models on Small Devices by Sebastian Buschjäger©SFB876-Summerschool2020
Example 1: Binarized Neural Networks (2)

**Typical Structure** VGG-Like architectures

Assume Inputs are already bool or integer

<table>
<thead>
<tr>
<th>Operation</th>
<th>Parameter</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>$x \cdot s + b$</td>
<td>$s, b \in \mathbb{R}$</td>
</tr>
</tbody>
</table>
**Example 1: Binarized Neural Networks (2)**

**Typical Structure** VGG-Like architectures

```
Input Conv BN HTanH MP FC
repeat k times
```

**Assume** Inputs are already bool or integer

<table>
<thead>
<tr>
<th>Operation</th>
<th>Parameter</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>$x \cdot s + b$</td>
<td>$s, b \in \mathbb{R}$</td>
</tr>
<tr>
<td>HTanH</td>
<td>$2 \cdot 1{x &gt; 0} - 1$</td>
<td>-</td>
</tr>
</tbody>
</table>
Example 1: Binarized Neural Networks (2)

Typical Structure VGG-Like architectures

Assume Inputs are already bool or integer

<table>
<thead>
<tr>
<th>Operation</th>
<th>Parameter</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>$x \cdot s + b$</td>
<td>$s, b \in \mathbb{R}$</td>
</tr>
<tr>
<td>HTanh</td>
<td>$2 \cdot 1{x &gt; 0} - 1$</td>
<td>-</td>
</tr>
<tr>
<td>MP</td>
<td>$\max(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1})$</td>
<td>-</td>
</tr>
</tbody>
</table>
**Example 1: Binarized Neural Networks (2)**

**Typical Structure** VGG-Like architectures

Assume Inputs are already bool or integer

<table>
<thead>
<tr>
<th>Operation</th>
<th>Parameter</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>$x \cdot s + b$</td>
<td>$s, b \in \mathbb{R}$</td>
</tr>
<tr>
<td>HTanH</td>
<td>$2 \cdot \mathbb{1}{x &gt; 0} - 1$</td>
<td>-</td>
</tr>
<tr>
<td>MP</td>
<td>$\max(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1})$</td>
<td>-</td>
</tr>
<tr>
<td>Conv</td>
<td>$\sum_i \sum_j w_{i,j} \cdot x_{i+k,j+l}$</td>
<td>$w_{i,j} \in {-1, 1}$</td>
</tr>
<tr>
<td>FC</td>
<td>$\sum_i w_i \cdot x_i$</td>
<td>$w_{i,j} \in {-1, 1}$</td>
</tr>
</tbody>
</table>
Example 1: Binarized Neural Networks (2)

**Typical Structure** VGG-Like architectures

Assume Inputs are already bool or integer

<table>
<thead>
<tr>
<th>Operation</th>
<th>Parameter</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>$x \cdot s + b$</td>
<td>$s, b \in \mathbb{R}$</td>
</tr>
<tr>
<td>HTanH</td>
<td>$2 \cdot \mathbb{1}{x &gt; 0} - 1$</td>
<td>-</td>
</tr>
<tr>
<td>MP</td>
<td>$\max(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1})$</td>
<td>-</td>
</tr>
<tr>
<td>Conv</td>
<td>$\sum_i \sum_j w_{i,j} \cdot x_{i+k,j+l}$</td>
<td>$w_{i,j} \in {-1, 1}$</td>
</tr>
<tr>
<td>FC</td>
<td>$\sum_i w_i \cdot x_i$</td>
<td>$w_{i,j} \in {-1, 1}$</td>
</tr>
</tbody>
</table>
**Note** BN layer introduces float types, but is required for effective training.
Note BN layer introduces float types, but is required for effective training

Idea Combine BN output directly with Hard-Tanh activation

\[
\text{HTanh}(\text{BN}(x)) = 2 \cdot \mathbb{1}\{\text{BN}(x) > 0\} - 1 = 2 \cdot \mathbb{1}\left\{x > -\frac{b}{s}\right\} - 1
\]

This way Pre-compute \(-\frac{b}{s}\) and eval \(\mathbb{1}\left\{x > -\frac{b}{s}\right\}\) without floating point unit
**Overall approach** Map weights/inputs to bitstring ‘−1 → 0’ and ‘+1 → 1’

- $f_i w_i$ is ‘+1’ if same sign, else ‘0’. This is an XOR operation
- $\sum_i f_i w_i$ counts occurrences of same sign. This is the popcount operation.

**Thus** 2 clocks needed to process 32/64/128/256 bits (= weights)

$$\sum_i f_i w_i = \text{POPCNT}(f \text{ XOR } W)$$
Example 1: Binarized Neural Networks (5)

Usual training loop

- **Forward** Compute $\hat{y} = f_W(x)$
- **Backward** Update parameters $W = W - \alpha_t \cdot \frac{\partial \ell(\hat{y}, y)}{\partial W}$

Observation $\frac{\partial \ell(\hat{y}, y)}{\partial W}$ does not exist if $w \in \{-1, 1\}$
Example 1: Binarized Neural Networks (5)

**Usual training loop**

- **Forward** Compute \( \hat{y} = f_W(x) \)
- **Backward** Update parameters \( W = W - \alpha_t \cdot \frac{\partial \ell(\hat{y}, y)}{\partial W} \)

Observation \( \frac{\partial \ell(\hat{y}, y)}{\partial W} \) does not exist if \( w \in \{-1, 1\} \)

**Thus** Use deterministic binarization + full precision SGD

- **Forward** binarize weights with \( w_i^b = 2 \cdot \mathbb{1}\{w_i > 0\} - 1 \) and Compute \( \hat{y} = f_{W^b}(x) \)
- **Backward** Keeps original gradients using \( w_i \) instead of \( w_i^b \)

**Note** Training is usually slower, due to binarization
### Example 1: Binarized Neural Networks (6)

**Buschjäger, Pfahler et al. 2020**

Classification of telescope data into gamma and proton particles

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy, no quality data</th>
<th>Accuracy, quality data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>epochs:100</td>
<td>epochs:10</td>
</tr>
<tr>
<td>CNN(small)</td>
<td>0.90825</td>
<td>0.88867</td>
</tr>
<tr>
<td>BNN(small)</td>
<td>0.90861</td>
<td>0.88644</td>
</tr>
<tr>
<td>CNN(large)</td>
<td><strong>0.91094</strong></td>
<td><strong>0.90251</strong></td>
</tr>
<tr>
<td>BNN(large)</td>
<td>0.90011</td>
<td>0.89925</td>
</tr>
</tbody>
</table>
Example 1: Binarized Neural Networks (6)

**Buschjäger, Pfahler et al. 2020**
Classification of telescope data into gamma and proton particles

<table>
<thead>
<tr>
<th>System</th>
<th>Type</th>
<th>Runtime [ms/event]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>float</td>
<td>binary</td>
<td></td>
</tr>
<tr>
<td>ONNX Runtime</td>
<td>large</td>
<td>21.083 ± 0.078</td>
<td>26.642 ± 0.100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>small</td>
<td>0.957 ± 0.020</td>
<td>1.861 ± 0.037</td>
<td></td>
</tr>
<tr>
<td>Our Approach (CPU)</td>
<td>large</td>
<td>78.583 ± 1.704</td>
<td>11.250 ± 0.077</td>
<td></td>
</tr>
<tr>
<td></td>
<td>small</td>
<td>2.757 ± 0.026</td>
<td>1.574 ± 0.014</td>
<td></td>
</tr>
<tr>
<td>Our Approach (FPGA)</td>
<td>large</td>
<td>-</td>
<td>72.657 ± 0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>small</td>
<td>-</td>
<td>0.662 ± 0.000</td>
<td></td>
</tr>
</tbody>
</table>
Idea 1: Train different model (Recap)

\[
f^* = \arg \min_{f \in F} \frac{1}{N} \sum_{(x,y) \in D} \ell(f(x), y) + \lambda R(f)
\]

Use resource-friendly model class \( F \) directly

\( \Rightarrow \) Guaranteed resource consumption, but maybe weak loss

Guide selection via regularization

\( \Rightarrow \) Direct trade-off between loss and model complexity via \( \lambda \)
Idea 1: Train different model (Recap)

\[ f^* = \arg \min_{f \in F} \frac{1}{N} \sum_{(x,y) \in D} \ell(f(x), y) + \lambda R(f) \]

Use resource-friendly model class \( \mathcal{F} \) directly

\( \Rightarrow \) Guaranteed resource consumption, but maybe weak loss

Guide selection via regularization

\( \Rightarrow \) Direct trade-off between loss and model complexity via \( \lambda \)

To combat overfitting literature often enforces resource-friendly models

- **Decision Trees** Limit height / number of nodes
- **Ensembles** Limit number of base learners / Use ‘weak’ base learners
- **Kernel methods** Stronger regularization, sampling
- **Linear models** Stronger \( \ell_0 \) regularization
- **Deep Learning** Smaller architectures, enforce different data types
Idea 2: Change model after training

**Idea** Compute resource-friendly model $h \in \mathcal{H}_f \subseteq \mathcal{F}$ from *already trained* model $f$ with minimal loss $\ell$.
Idea 2: Change model after training

**Idea** Compute resource-friendly model $h \in \mathcal{H}_f \subseteq \mathcal{F}$ from *already trained* model $f$ with minimal loss $\ell$

**Formally** $\mathcal{H}_f = \{ r(f) | r : \mathcal{F} \to \mathcal{H} \}$ where $r : \mathcal{F} \to \mathcal{H}$ is a (non-deterministic) ‘mapping function’
Idea 2: Change model after training

**Idea** Compute resource-friendly model $h \in \mathcal{H}_f \subseteq \mathcal{F}$ from *already trained* model $f$ with minimal loss $\ell$

**Formally** $\mathcal{H}_f = \{r(f) | r : \mathcal{F} \to \mathcal{H}\}$ where $r : \mathcal{F} \to \mathcal{H}$ is a (non-deterministic) ‘mapping function’

**Step 1** Train model as usually

$$f^* = \arg \min_{f \in \mathcal{F}} \frac{1}{N} \sum_{(x, y) \in \mathcal{D}} \ell(f(x), y) + \lambda R(f)$$

**Step 2** Choose simpler model from $\mathcal{H}$

$$h^* = \arg \min_{h \in \mathcal{H}_f} \frac{1}{N} \sum_{(x, y) \in \mathcal{D}} \ell(h(x), y) + \lambda C(h)$$
Example 2 Use smaller data types (1)

Gupta et al. 2015, Han et. al 2016, Gysel et. al 2016 . . .

Reduction $32 \rightarrow 16$ bit nearly no performance difference in training
Reduction $32 \rightarrow 8$ bit nearly no performance difference in inference
Reduction $32 \rightarrow 2$ bit also possible, but requires special training
Example 2 Use smaller data types (1)

Gupta et al. 2015, Han et. al 2016, Gysel et. al 2016 . . .

Reduction 32 → 16 bit nearly no performance difference in training
Reduction 32 → 8 bit nearly no performance difference in inference
Reduction 32 → 2 bit also possible, but requires special training

Thus Compression factor 2 − 4 for free using fixed point
But Predictions are changed after model has been chosen!
Example 2 Use smaller data types (2)

**Fixed point**

\[
X_{fx} = \underbrace{X_{(1)}X_{(0)}}_{N_f} \cdot \underbrace{X_{(-1)}X_{(-2)}X_{(-3)}X_{(-4)}}_{N_r}
\]

**Implementation** As scaled integer of size \(N_t\), e.g. char for 8 bit
Example 2 Use smaller data types (2)

Fixed point

\[ X_{fx} = \underbrace{X(1)X(0)}_{N_l} \cdot \underbrace{X(-1)X(-2)X(-3)X(-4)}_{N_r} \]

Implementation As scaled integer of size \( N_t \), e.g. char for 8 bit

Fixed \( \rightarrow \) float: 
\[ X_{fl} = \sum_{i=0}^{N_l} X(i)2^i + \sum_{i=-1}^{-N_r} X(-i)2^{-i} \]

Float \( \rightarrow \) fixed: ?
**Example 2 Use smaller data types (2)**

**Fixed point**

\[
X_{fx} = \left\{ \begin{array}{c}
X_{(1)} \cdot X_{(0)} \\
X_{(-1)} \cdot X_{(-2)} \cdot X_{(-3)} \cdot X_{(-4)}
\end{array} \right\}_N
\]

**Implementation** As scaled integer of size \(N_t\), e.g. char for 8 bit

Fixed \(\rightarrow\) float: \(X_{fl} = \sum_{i=0}^{N_l} X(i)2^i + \sum_{i=-1}^{-N_r} X(-i)2^{-i}\)

Float \(\rightarrow\) fixed: \(X_{fx} = \lfloor X_{fl} \cdot 2^{N_r} \rfloor\)
Example 2 Use smaller data types (2)

Fixed point

\[ X_{fx} = \frac{X_{(1)}X_{(0)}}{N_l} \cdot \frac{X_{(-1)}X_{(-2)}X_{(-3)}X_{(-4)}}{N_r} \]

Implementation As scaled integer of size \( N_t \), e.g. char for 8 bit

Fixed \( \rightarrow \) float: \( X_{fl} = \sum_{i=0}^{N_l} X_{(i)}2^i + \sum_{i=-1}^{-N_r} X_{(-i)}2^{-i} \)

Float \( \rightarrow \) fixed: \( X_{fx} = \lfloor X_{fl} \cdot 2^{N_r} \rfloor \)

use shift operations!
\( 2^i = (1 << i) \)
\( 2^{-i} = (1 >> i) \)
Example 2 Use smaller data types (3)

**Addition/subtraction** No changes required

\[
X''_{fx} = X_{fx} + X'_{fx} = \lfloor X_{fl} \cdot 2^{N_r} \rfloor + \lfloor X'_{fl} \cdot 2^{N_r} \rfloor = \lfloor (X_{fl} + X'_{fl}) \cdot 2^{N_r} \rfloor
\]
**Addition/substraction** No changes required

\[ X_{fx}'' = X_{fx} + X_{fx}' = \lfloor X_{fl} \cdot 2^{N_r} \rfloor + \lfloor X'_{fl} \cdot 2^{N_r} \rfloor = \lfloor (X_{fl} + X'_{fl}) \cdot 2^{N_r} \rfloor \]

**Multiplication/Division** Correct scaling

\[ X_{fx} \cdot X'_{fx} = \lfloor X_{fl} \cdot 2^{N_r} \rfloor \cdot \lfloor X'_{fl} \cdot 2^{N_r} \rfloor = \lfloor (X_{fl} \cdot X'_{fl}) \cdot (2^{N_r} \cdot 2^{N_r}) \rfloor \]

\[ \Rightarrow X_{fx}'' = X_{fx} \cdot X_{fx}' \cdot 2^{-N_r} \]
**Han et al 2015** Weights are somewhat Gaussian distributed

![Graph showing Gaussian distribution with fixed and dynamic quantization points.](image)
Han et al 2015 Weights are somewhat Gaussian distributed

**In red** Fixed point quantization

**In green** Dynamic quantization
Example 2 Use smaller data types (5)

Han et al 2015 / Han et al 2016 Cluster weights after training

- Train connections
- Prune connections
- Retrain connections
- Retrain Code book

Cluster weights, e.g. k-means
Extract centroids
Assign indexing scheme
Update shared weights jointly
Example 2 Use smaller data types (5)

Han et al 2015 / Han et al 2016 Cluster weights after training

- Train connections
- Prune connections
- Retrain connections
- Retrain Code book

Cluster weights, e.g. k-means
Extract centroids
Assign indexing scheme
Update shared weights jointly

Results
~ 20x compression in overall size
Han et al 2016: Clustering changes weight distribution

Example 2: Use smaller data types (6)

Idea: Use Huffman encoding to further reduce size

Results: Additional $\sim 10\times$ compression
Han et al 2016 Clustering changes weight distribution

Idea
Use huffman encoding to further reduce size

Results Additional $\sim 10x$ compression
Han et al 2016 Combining all three approaches

35 – 49x compression ratio
3 – 4x speed-up
3 – 7x less energy
No loss in accuracy

But Slow, due to post-processing
Idea 2: Change model after training (Recap)

**Idea** Compute resource-friendly model $r(f) \in \mathcal{H} \subseteq \mathcal{F}$ from *already trained* model $f$ with minimal loss $\ell$

**Formally** $\mathcal{H}_f = \{r(f)| r: \mathcal{F} \rightarrow \mathcal{H}\}$ where $r: \mathcal{F} \rightarrow \mathcal{H}$ is a (non-deterministic) ‘mapping function’

- **Step 1** Train model as usually
- **Step 2** Choose simpler model from $\mathcal{H}$
- **Step 3 (Optional)** Repeat 1) with restricted class $\mathcal{H}$
Idea 2: Change model after training (Recap)

Idea Compute resource-friendly model \( r(f) \in \mathcal{H} \subseteq \mathcal{F} \) from already trained model \( f \) with minimal loss \( \ell \)

Formally \( \mathcal{H}_f = \{ r(f) | r: \mathcal{F} \to \mathcal{H} \} \) where \( r: \mathcal{F} \to \mathcal{H} \) is a (non-deterministic) ‘mapping function’

- **Step 1** Train model as usually
- **Step 2** Choose simpler model from \( \mathcal{H} \)
- **Step 3 (Optional)** Repeat 1) with restricted class \( \mathcal{H} \)

Again literature often has starting points established

- **Decision Trees** Pessimistic Error Pruning, Minimum Error Pruning, …
- **Ensembles** Ensemble-pruning Via Semi-definite Programming, Pareto Ensemble Pruning, …
- **Linear and Kernel models** Remove unused features / support vectors, Weight Clustering, …
- **Deep Learning** Prune weights, prune connections, fixed convolutions, …
So far We talked about models, but what about implementations?
Kriegel et al. 2016 Are we comparing algorithms or implementations?
So far We talked about models, but what about implementations?

Kriegel et al. 2016 Are we comparing algorithms or implementations?

Implementation must respect any specifics of the model
Storing binary weights in float instead of bool defeats the purpose
Idea 3: Use hardware differently

So far We talked about models, but what about implementations?
Kriegel et al. 2016 Are we comparing algorithms or implementations?

Implementation must respect any specifics of the model
Storing binary weights in float instead of bool defeats the purpose

Implementation must respect any specifics of the hardware
SIMD instructions often work on char or larger datatypes not bool
Idea 3: Use hardware differently

Fact There are at-least two ways to implement decision trees

```c
Node t[] = {/* ... */ };
bool predict(short const * x){
unsigned int i = 0;
while(!t[i].isLeaf) {
if (x[t[i].f] <= t[i].s) {
i = t[i].l;
} else {
i = t[i].r;
}
}
return t[i].pred;
}
```

```c
bool predict(short const * x){
if(x[0] <= 8191){
if(x[1] <= 2048){
return true;
} else {
return false;
}
} else {
if(x[2] <= 512){
return true;
} else {
return false;
}
}
```
Fact There are at-least two ways to implement decision trees

**Native-Tree** Store nodes in array and iterate via loop

```c
Node t[] = {/* ... */};
bool predict(short const * x){
    unsigned int i = 0;
    while(!t[i].isLeaf) {
        if (x[t[i].f] <= t[i].s) {
            i = t[i].l;
        } else {
            i = t[i].r;
        }
    }
    return t[i].pred;
}
```

**If-Else-Tree** Unroll tree into if-else structure

```c
bool predict(short const * x){
    if(x[0] <= 8191){
        if(x[1] <= 2048){
            return true;
        } else {
            return false;
        }
    } else {
        if(x[2] <= 512){
            return true;
        } else {
            return false;
        }
    }
}
```
**Idea 3: Use hardware differently**

**Fact** There are at least two ways to implement decision trees

**Native-Tree** Store nodes in array and iterate via loop

```c
Node t[] = {/* ... */};
bool predict(short const * x){
    unsigned int i = 0;
    while(!t[i].isLeaf) {
        if (x[t[i].f] <= t[i].s) {
            i = t[i].l;
        } else {
            i = t[i].r;
        }
    }
    return t[i].pred;
}
```

**If-Else-Tree** Unroll tree into if-else structure

```c
bool predict(short const * x){
    if(x[0] <= 8191){
        if(x[1] <= 2048){
            return true;
        } else {
            return false;
        }
    } else {
        if(x[2] <= 512){
            return true;
        } else {
            return false;
        }
    }
}
```
What is the Caching behavior of these two implementations?
What is the Caching behavior of these two implementations?

Native

+ Simple to implement and small "hot code".
+ Requires data and instruction cache.
- Has indirect memory accesses

If-Else

+ No indirect memory access
+ Only instruction cache is required.
- No small "hot code".
Example 3: Use hardware differently

Lucchese et al. 2015, Dato et al. 2016
Implement trees with bitoperations

Recall Each node in DT \((f, t)\) checks \(x_f \leq t\)

- Represent leaves of tree as bitvector of active / inactive nodes
- Set all bits of left (right) sub-trees to 0 depending on \(x_f \leq t\)
- Invalidating sub-trees AND-ing bitmasks to bitvector
- Once all false nodes are evaluated left-most 1-bit indicates prediction node
Example 3: Use hardware differently

Lucchese et al. 2015, Dato et al. 2016
Implement trees with bitoperations

Recall Each node in DT \((f, t)\) checks \(x_f \leq t\)
- Represent leaves of tree as bitvector of active / inactive nodes
- Set all bits of left (right) sub-trees to 0 depending on \(x_f \leq t\)
- Invalidating sub-trees AND-ing bitmasks to bitvector
- Once all false nodes are evaluated left-most 1-bit indicates prediction node

Example

\((0, 1)\) are true, rest false : \(1111111 \land 0111111 \land 1110011 \land 1110111 \land 1111101 = 011001 \Rightarrow (8) \text{ is leaf}\)
Example 3: Use hardware differently

Lucchese et al. 2015, Dato et al. 2016
Implement trees with bitoperations

Recall Each node in DT \((f, t)\) checks \(x_f \leq t\)

- Represent leaves of tree as bitvector of active / inactive nodes
- Set all bits of left (right) sub-trees to 0 depending on \(x_f \leq t\)
- Invalidating sub-trees AND-ing bitmasks to bitvector
- Once all false nodes are evaluated left-most 1-bit indicates prediction node

Example

\((0, 1)\) are true, rest false : \(1111111 \land 0111111 \land 1110011 \land 1110111 \land 1111101 = 011001 \Rightarrow (8)\) is leaf

\((2, 5)\) are true, rest false : ?
Example 3: Use hardware differently

Lucchese et al. 2015, Dato et al. 2016
Implement trees with bitoperations

Recall Each node in DT \((f, t)\) checks \(x_f \leq t\)

- Represent leaves of tree as bitvector of active / inactive nodes
- Set all bits of left (right) sub-trees to 0 depending on \(x_f \leq t\)
- Invalidating sub-trees AND-ing bitmasks to bitvector
- Once all false nodes are evaluated left-most 1-bit indicates prediction node

Example

\((0, 1)\) are true, rest false : \(1111111 \land 0111111 \land 1110011 \land 1110111 \land 1111101 = 011001 \Rightarrow (8)\) is leaf

\((2, 5)\) are true, rest false : \(1111111 \land 0001111 \land 0011111 \land 0111111 \land 1111101 = 0001101 \Rightarrow (9)\) is leaf
Example 3: Use hardware differently (2)

But Potentially all nodes are evaluated
Example 3: Use hardware differently (2)

**But** Potentially all nodes are evaluated

**QuickScorer**
Take all nodes from the entire forest and sort them according to feature/split

- Loop over all features and check split $x_f > t$. Once a comparison is true all future comparisons for that feature are
- Clever implementations utilizing single arrays as much as possible
Example 3: Use hardware differently (2)

**But** Potentially *all* nodes are evaluated

**QuickScorer**
Take all nodes from the entire forest and sort them according to feature/split

- Loop over all features and check split \( x_f > t \). Once a comparison is true *all* future comparisons for that feature are
- Clever implementations utilizing single arrays as much as possible

**Good**

- Skip as many comparisons as possible
- Easy on branch-prediction and caching
Example 3: Use hardware differently (2)

**But** Potentially all nodes are evaluated

**QuickScorer**
Take all nodes from the entire forest and sort them according to feature/split

- Loop over all features and check split $x_f > t$. Once a comparison is true all future comparisons for that feature are
- Clever implementations utilizing single arrays as much as possible

**Good**

- Skip as many comparisons as possible
- Easy on branch-prediction and caching

**Bad**

- Worst case performs all comparisons
- Efficient implementation limits tree depth ($d = 6$ in $2^6 = 64$ bit CPUs)
Example 3: Use hardware differently (3)

Lucchese et al. 2015, Dato et al. 2016 QuickScorer
Represent leaves as bit-vectors and ‘reuse’ as many comparisons as possible
Speed-Up 2 − 6.5 over baseline
**Example 3: Use hardware differently (3)**

**Lucchese et al. 2015, Dato et al. 2016** QuickScorer  
Represent leaves as bit-vectors and 'reuse' as many comparisons as possible  
Speed-Up 2 – 6.5 over baseline

**Lucchese et al. 2016** V-QuickScorer  
Vectorized version which classifies multiple examples at a time  
Speed-Up 1.5 – 3.2 over QuickScorer
Example 3: Use hardware differently (3)

**Lucchese et al. 2015, Dato et al. 2016** QuickScorer
Represent leaves as bit-vectors and ‘reuse’ as many comparisons as possible
Speed-Up 2 – 6.5 over baseline

**Lucchese et al. 2016** V-QuickScorer
Vectorized version which classifies multiple examples at a time
Speed-Up 1.5 – 3.2 over QuickScorer

**Ye et al. 2018** RapidScorer
Encode bitmask to reduce memory while keeping vectorization
Speed-Up 1.3 – 3.5 over V-QuickScorer
Example 3: Use hardware differently (4)

Buschjäger et al. 2016; Buschjäger, Chen et al. 2018
Keep most important paths in cache

Branch-probability $p_{i \rightarrow j}$

Path-probability $p(\pi) = p_{\pi_0 \rightarrow \pi_1} \cdot \ldots \cdot p_{\pi_{L-1} \rightarrow \pi_L}$

Expected path length $\mathbb{E}[L] = \sum_\pi p(\pi) \cdot |\pi|$
Example 3: Use hardware differently (4)

Buschjäger et al. 2016; Buschjäger, Chen et al. 2018
Keep most important paths in cache

Branch-probability \( p_{i \rightarrow j} \)

Path-probability \( p(\pi) = p_{\pi_0 \rightarrow \pi_1} \cdot \cdots \cdot p_{\pi_{L-1} \rightarrow \pi_L} \)

Expected path length \( E[L] = \sum_\pi p(\pi) \cdot |\pi| \)

Example

\[
p((0, 1, 3)) = 0.3 \cdot 0.4 \cdot 0.25 = 0.03
\]

\[
p((0, 2, 6)) = 0.7 \cdot 0.8 \cdot 0.85 = 0.476
\]
Example 3: Use hardware differently (5)

**Capacity misses** Cache memory is not enough to store all code

**But** Computation kernel of tree might fit into cache

**Solution**

TreeFraming

Compute computation kernel for budget \( \beta \)

\[
K = \arg \max \left\{ p(T) \mid T \subseteq T \text{ s.t. } \sum_{i \in T} s(i) \leq \beta \right\}
\]

- Start with the root node
- Greedily add nodes until budget is exceeded

**Note**

- Estimate \( s(\cdot) \) based on assembly analysis
- Choose \( \beta \) based on the properties of specific CPU model
Example 3: Use hardware differently (5)

**Capacity misses** Cache memory is not enough to store all code

**But** Computation kernel of tree might fit into cache

**Solution** TreeFraming

Compute computation kernel for budget $\beta$

$$
\mathcal{K} = \arg \max \left\{ p(T) \mid T \subseteq \mathcal{T} \text{ s.t. } \sum_{i \in T} s(i) \leq \beta \right\}
$$

- Start with the root node
- Greedily add nodes until budget is exceeded

**Note**
- Estimate $s(\cdot)$ based on assembly analysis
- Choose $\beta$ based on the properties of specific CPU model
Example 3: Use hardware differently (5)

**Capacity misses** Cache memory is not enough to store all code

**But** Computation kernel of tree might fit into cache

**Solution** TreeFraming
Compute computation kernel for budget $\beta$

$$K = \arg \max \{ p(T) \mid T \subseteq T \text{s.t. } \sum_{i \in T} s(i) \leq \beta \}$$

- Start with the root node
- Greedily add nodes until budget is exceeded

**Note**
- Estimate $s(\cdot)$ based on assembly analysis
- Choose $\beta$ based on the properties of specific CPU model
Note Experiments are performed on an Raspberry Pi 3 (ArmV7)

Results

- Optimizations improve performance
- No clear winner for larger trees

Interpretation

- Smaller I-Cache (32 KiB) only fits small trees
- Smaller D-Cache (512 KiB) only fits small trees
- Requires more instructions than CISC
Note Experiments are performed on an Raspberry Pi 3 (ArmV7)

Results
• Optimizations improve performance
• No clear winner for larger trees

Interpretation
• Smaller I-Cache (32 KiB) only fits small trees
• Smaller D-Cache (512 KiB) only fits small trees
• Requires more instructions than CISC

Take-away?
Example 3: Use hardware differently (6)

Note Experiments are performed on an Raspberry Pi 3 (ArmV7)

Results

- Optimizations improve performance
- No clear winner for larger trees

Interpretation

- Smaller I-Cache (32 KiB) only fits small trees
- Smaller D-Cache (512 KiB) only fits small trees
- Requires more instructions than CISC

Take-away Use if-else version for small trees. For larger ones there is no clear recommendation
Implementation has a unique place for fast inference

1) It must respect any specifics of the model
2) It must respect any specifics of the hardware
Implementation has a unique place for fast inference

1) It must respect any specifics of the model
2) It must respect any specifics of the hardware

Again literature / tools offer some starting points

- **Decision Trees** If-Else Trees, TreeFraming, . . .
- **Ensembles** QuickScorer, RapidScorer, . . .
- **Linear and Kernel models** SIMD / GPU, HW-friendly kernel, Fixed-point SVM, . . .
- **Deep Learning** SIMD / GPU, Fast Fourier Transforms, Winograd convolution, . . .
## Pros and cons of each approach

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train new model</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Alter pre-trained model</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Optimized implementation</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Pros</td>
<td>Cons</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Train new model</td>
<td>Optimal model for resources</td>
<td></td>
</tr>
<tr>
<td>Alter pre-trained model</td>
<td>Separation of concerns</td>
<td></td>
</tr>
<tr>
<td>Optimized implementation</td>
<td>No effect on model’s accuracy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Training can be difficult</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accuracy might suffer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardware- and model-specific</td>
<td></td>
</tr>
</tbody>
</table>
Tooling Support

**Deep Learning**
ONNX Runtime, Apache TVM, Glow, TF-XLA, TF-Lite, NGraph, OpenVino, TensorRT, FINN, ...

**(Tree) Ensembles**
QuickScorer, Treelite, ...

**The Rest (!?)**
CoreML, Iree, ...
Tooling Support

Deep Learning
ONNX Runtime, Apache TVM, Glow, TF-XLA, TF-Lite, NGraph, OpenVino, TensorRT, FINN, ...

(Tree) Ensembles
QuickScorer, Treelite, ...

The Rest (!?)
CoreML, Iree, ...

But These are mostly runtime engines and/or require a runtime engine
Thus Difficult to use bare-metal
Why not generate optimized code for every hardware / model?
Why not generate optimized code for every hardware / model?

- Read File
- Optimize Model
- Choose Target Architecture
- Combine Templates

Linear Model
Ensemble
Decision Tree
Neural Network

Weight clustering
Tree Framing
X86
ARM
FPGA

templates/NN/binary/gmm.j2
templates/NN/binary/conv.j2
templates/NN/float/gmm.j2
templates/NN/float/conv.j2
...
Template-based model implementation (2)

Nice

+ Model is statically unrolled
+ No runtime environment required
+ Easy to support new targets / models
Template-based model implementation (2)

Nice

+ Model is statically unrolled
+ No runtime environment required
+ Easy to support new targets / models

But

- 2-step process (code generation + compilation)
- Some optimizations cannot be performed in templates
Template-based model implementation (2)

**Nice**

+ Model is statically unrolled
+ No runtime environment required
+ Easy to support new targets / models

**But**

- 2-step process (code generation + compilation)
- Some optimizations cannot be performed in templates

**Status**

Scikit-Learn: Linear models, Decision Trees, Ensembles. Some optimizations available.

PyTorch / ONNX: Convolutional Neural Networks, BNNs. ResNets are currently developed
Model application matters
Autopilot / bing.com require entire energy plant!
Model application matters
Autopilot / bing.com require entire energy plant!

How to approach fast inference

- **Idea 1** Train a different model ⇒ Example: BNNs
- **Idea 2** Change model after training ⇒ Example: Weight clustering in NN
- **Idea 3** Use hardware differently ⇒ Examples: QuickScorer and TreeFraming
- **Idea 4** Use different hardware
Model application matters
Autopilot / bing.com require entire energy plant!

How to approach fast inference

- **Idea 1** Train a different model ⇒ Example: BNNs
- **Idea 2** Change model after training ⇒ Example: Weight clustering in NN
- **Idea 3** Use hardware differently ⇒ Examples: QuickScorer and TreeFraming
- **Idea 4** Use different hardware

Many specialized tools exist. But the usually require runtime environment.

Upcoming FastInference as a code generator