Machine Learning Systems Design

Modeling Pipeline

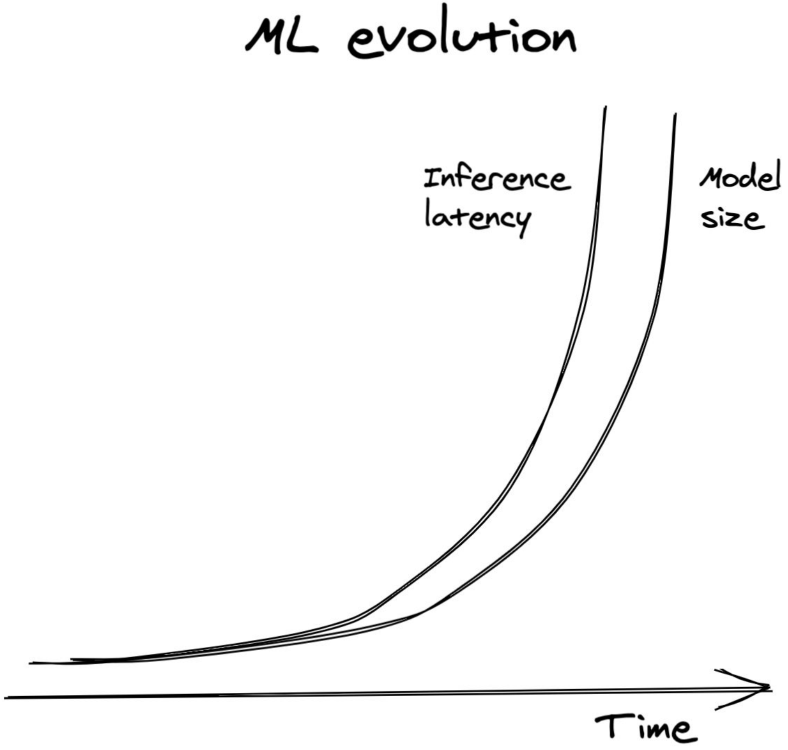
Lecture 14: Model Resource Management

CE 40959 Spring 2023
Ali Zarezade
SharifMLSD.github.io

Agenda

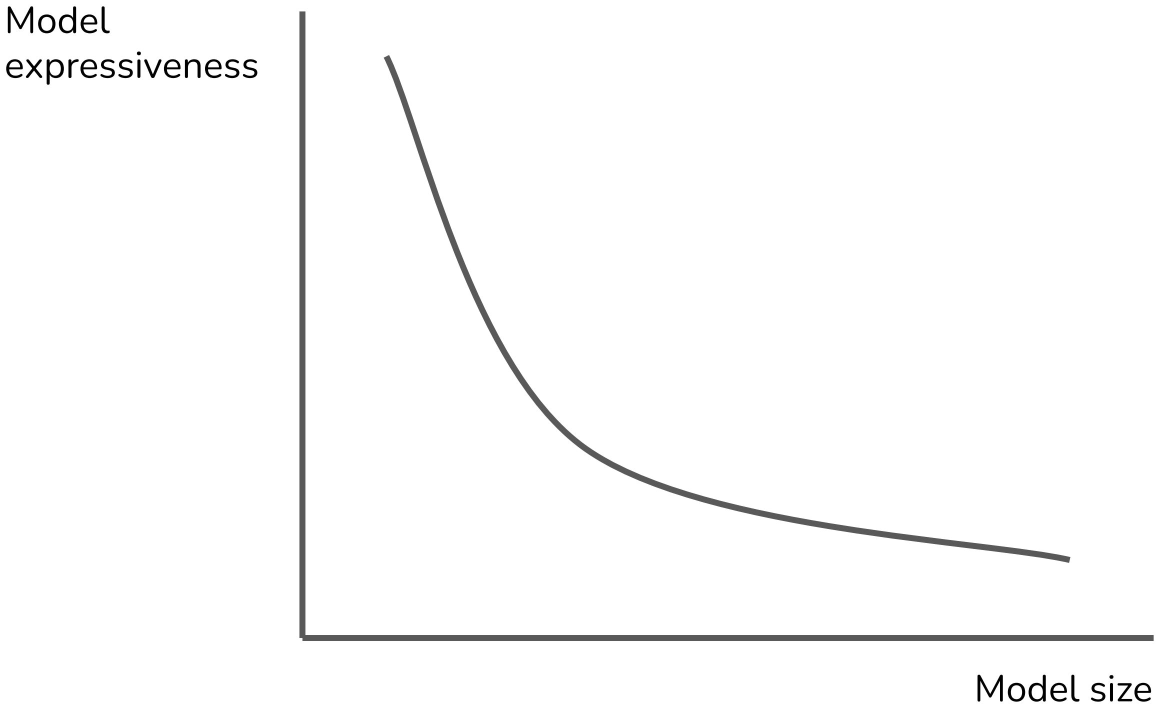
1. Model Compression
2. Cloud vs Edge Computing
3. Optimizing Model for Edge

1. Model Compression



Bigger, better, slower

No free lunch!



Model compression

1. Quantization
2. Knowledge distillation
3. Pruning
4. Low-ranked factorization

Model compression: active research/development

The Top 121 Model Compression Open Source Projects on Github

Categories > Machine Learning > **Model Compression**

Nni 10910 ⭐
An open source AutoML toolkit for automate machine learning lifecycle, including feature engineering, neural architecture search, model compression and hyper-parameter tuning.

Pocketflow 2676 ⭐
An Automatic Model Compression (AutoMC) framework for developing smaller and faster AI applications.

Neuronblocks 1404 ⭐
NLP DNN Toolkit - Building Your NLP DNN Models Like Playing Lego

Ghostnet 1823 ⭐
CV backbones including GhostNet, TinyNet and TNT, developed by Huawei Noah's Ark Lab.

Channel Pruning 1021 ⭐
Channel Pruning for Accelerating Very Deep Neural Networks (ICCV'17)

Model Optimization 1189 ⭐
A toolkit to optimize ML models for deployment for Keras and TensorFlow, including quantization and pruning.

Knowledge Distillation Pytorch 1291 ⭐
A PyTorch implementation for exploring deep and shallow knowledge distillation (KD) experiments with flexibility

Awesome Pruning 1361 ⭐
A curated list of neural network pruning resources.

Awesome Knowledge Distillation 1612 ⭐
Awesome Knowledge-Distillation. 分类整理的知识蒸馏paper(2014-2021)。

https://opensourcelibs.com/libs/model-compression | https://awesomeopensource.com/projects/model-compression

Model compression: quantization

- Reduces the size of a model by using fewer bits to represent parameter values
    - E.g. half-precision (16-bit) or integer (8-bit) instead of full-precision (32-bit)

100M params

32-bit	16-bit
400MB	200MB

Model compression: quantization

- Reduces the size of a model by using fewer bits to represent parameter values
  - E.g. half-precision (16-bit) or integer (8-bit) instead of full-precision (32-bit)
  - 1-bit representation: BinaryConnect, Xnor-Net

Exclusive: Apple acquires Xnor.ai, edge AI spin-out from Paul Allen's AI2, for price in $200M range

BY ALAN BOYLE, TAYLOR SOPER & TODD BISHOP on January 15, 2020 at 10:44 am

Model compression: quantization

Pros	

1. Reduce memory footprint
2. Increase computation speed
    a. Bigger batch size
    b. Computation on 16 bits is faster than on 32 bits	

BFloat16: The secret to high performance on Cloud TPUs

Model compression: quantization

<table>
<tr><th>Pros</th><th>Cons</th></tr>
<tr><td>1. Reduce memory footprint
2. Increase computation speed
    a. Bigger batch size
    b. Computation on 16 bits is faster than on 32 bits</td><td>1. Smaller range of values
2. Values rounded to 0

Need efficient rounding/scaling techniques</td></tr>
</table>

Model compression: quantization

- Post-training quantization

```
torch.quantization.convert(model, inplace=True)
```

- Quantization-aware training

Model compression: knowledge distillation

- Train a small model (“student”) to mimic the results of a larger model (“teacher”)
  - Teacher & student can be trained at the same time.
  - E.g. DistillBERT, reduces size of BERT by 40%, and increases inference speed by 60%, while retaining 97% language understanding.

DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter (Sanh et al., 2019)

Model compression: knowledge distillation

- Train a small model (“student”) to mimic the results of a larger model (“teacher”)
- Pros:
    - Fast to train student network if teacher is pre-trained.
    - Teacher and student can be completely different architectures.
- Cons:
    - If teacher is not pre-trained, may need more data & time to first train teacher.
    - Sensitive to applications and model architectures.

Model compression: pruning

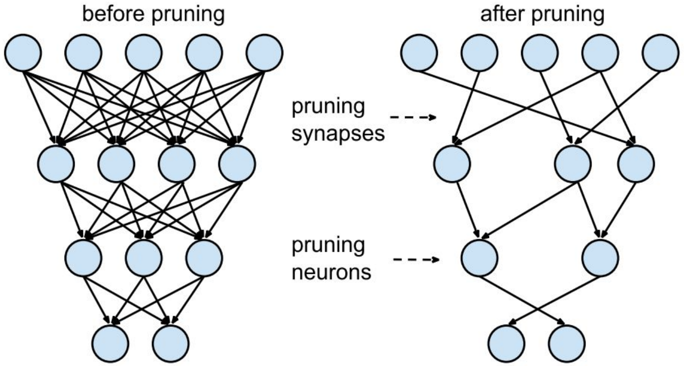
- Originally used for decision trees to remove uncritical sections
- Neural networks: reducing over-parameterization

Model compression: pruning methods

1. Remove nodes
    a. Changing architectures & reducing number of params

Model compression: pruning methods

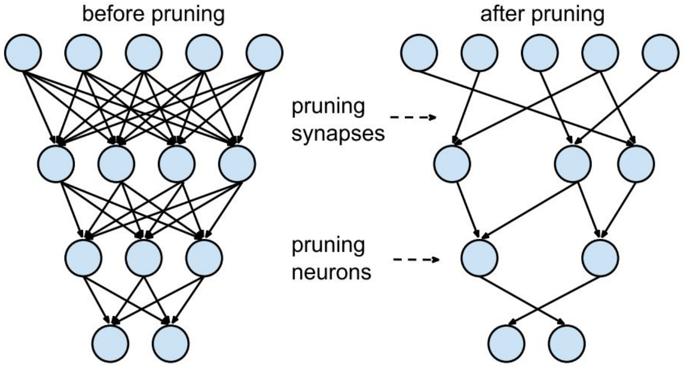
1. Remove nodes
2. Find least useful params & set to 0
    a. Number of params remains the same
    b. Reducing number of non-zero params



Learning both Weights and Connections for Efficient Neural Networks (Han et al., 2015)

Model compression: pruning methods

1. Remove nodes
2. Find least useful params & set to 0 ???
    a. Number of params remains the same
    b. Reducing number of non-zero params



Learning both Weights and Connections for Efficient Neural Networks (Han et al., 2015)

Model compression: pruning methods

1. Remove nodes
2. Find least useful params & set to 0
    a. Number of params remains the same
    b. Reducing number of non-zero params

Makes models more sparse

- lower memory footprint
- increased inference speed

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks (Frankle et al., ICLR 2019)

Model compression: pruning methods

1. Remove nodes
2. Find least useful params & set to 0
   a. Number of params remains the same
   b. Reducing number of non-zero params

Can be used for architecture search

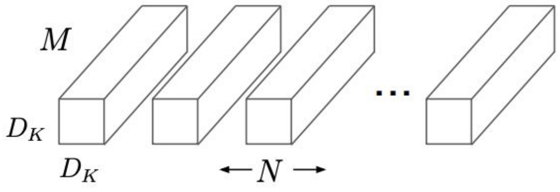
Model compression: factorization

The key idea behind low-rank factorization is to replace high-dimensional tensors with lower-dimensional tensors.

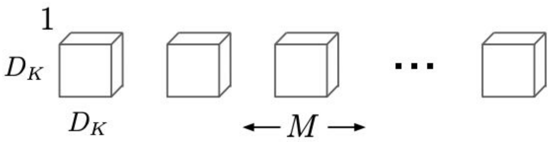
One type of low-rank factorization is compact convolutional filters, where the over-parameterized convolution filters are replaced with compact blocks.

Model compression: factorization

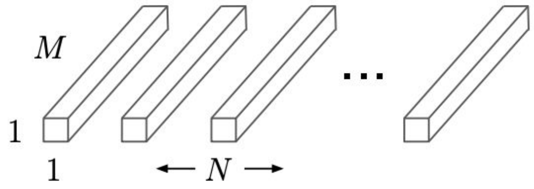
- 3 x 3 matrix can be written as a product of 3 x 1 and 1 x 1
  - 6 params instead of 9
  - SqueezeNets achieves AlexNet-level accuracy on ImageNet with 50 times fewer parameters.
- Replace convolution filters (many parameters) with compact blocks
  - E.g. MobileNets:
  - decomposes the standard convolution of size *Dk* × *Dk* × *M* into a depthwise convolution (*Dk* × *Dk* × 1) and a pointwise convolution (1 × 1 × *M*)
    - (a) are replaced by depthwise convolution
    - (b) and pointwise convolution
    - (c) to build a depthwise separable filter



(a) Standard Convolution Filters



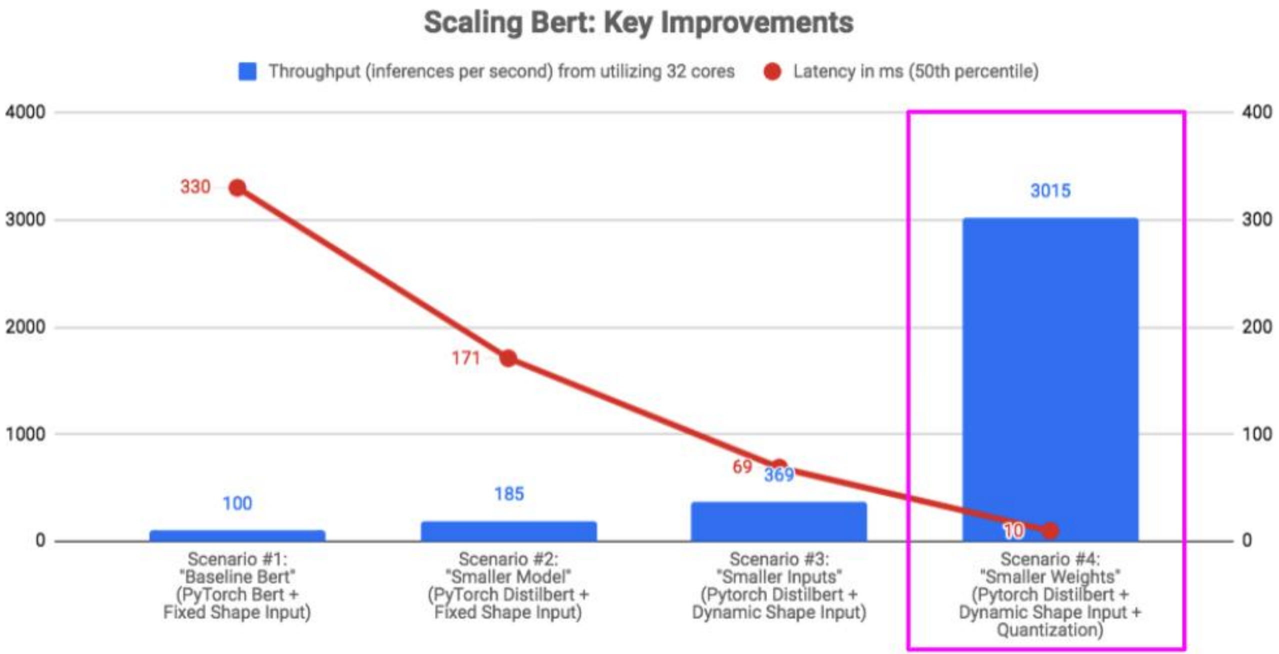
(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

[MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications] (Howard et al., 2017)

Make models smaller: case study



How We Scaled Bert To Serve 1+ Billion Daily Requests on CPUs (Roblox, 2020)

2. Cloud vs Edge Computing

	Cloud computing	Edge computing
Computations	Done on cloud (servers)	Done on edge devices (browsers, phones, tablets, laptops, smart watches, activity watchers, cars, etc.)
Examples	• Most queries to Alexa, Siri, Google Assistant • Google Translate for rare language pairs (e.g. English - Yiddish)	• Wake words for Alexa, Siri, Google Assistant • Google Translate for popular language pairs (e.g. English - Spanish) • Predictive text • Unlocking with fingerprints, faces

Benefits of edge computing

- Can work without (Internet) connections or with unreliable connections
  - Many companies have strict no-Internet policy
  - **Caveat**: devices are capable of doing computations but apps need external information
    - e.g. ETA needs external real-time traffic information to work well

Benefits of edge computing

- Can work without (Internet) connections or with unreliable connections
    - Many companies have strict no-Internet policy
    - **Caveat**: devices are capable of doing computations but apps need external information
        - e.g. ETA needs external real-time traffic information to work well
- Don't have to worry about network latency
    - Network latency might be a bigger problem than inference latency
    - Many use cases are impossible with network latency
        - e.g. predictive texting

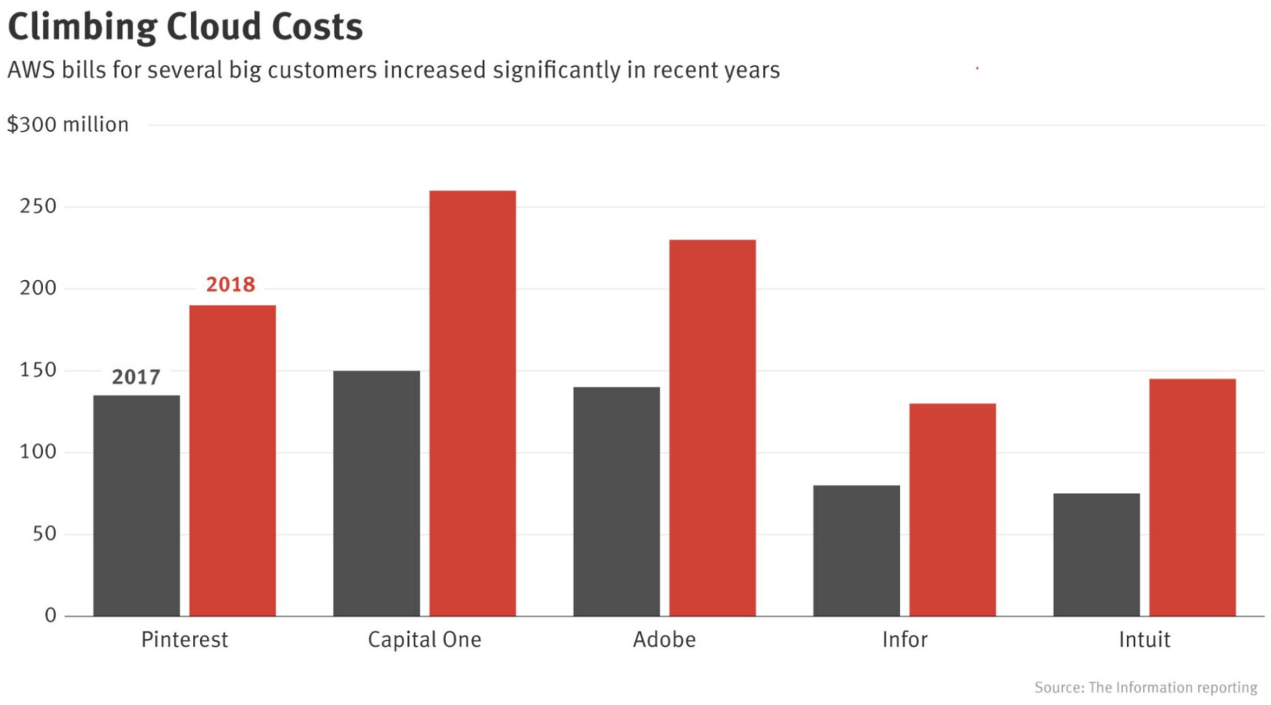
Benefits of edge computing

- Can work without (Internet) connections or with unreliable connections
  - Many companies have strict no-Internet policy
  - **Caveat**: devices are capable of doing computations but apps need external information
    - e.g. ETA needs external real-time traffic information to work well
- Don't have to worry about network latency
  - Network latency might be a bigger problem than inference latency
  - Many use cases are impossible with network latency
    - e.g. predictive texting
- Fewer concerns about privacy
  - Don't have to send user data over networks (which can be intercepted)
  - Cloud database breaches can affect many people
  - Easier to comply with regulations (e.g. GDPR)
  - **Caveat**: edge computing might make it easier to steal user data by just taking the device

Benefits of edge computing

- Can work without (Internet) connections or with unreliable connections
  - Many companies have strict no-Internet policy
  - **Caveat**: devices are capable of doing computations but apps need external information
    - e.g. ETA needs external real-time traffic information to work well
- Don't have to worry about network latency
  - Network latency might be a bigger problem than inference latency
  - Many use cases are impossible with network latency
    - e.g. predictive texting
- Fewer concerns about privacy
  - Don't have to send user data over networks (which can be intercepted)
  - Cloud database breaches can affect many people
  - Easier to comply with regulations (e.g. GDPR)
  - **Caveat**: edge computing might make it easier to steal user data by just taking the device
- Cheaper
  - The more computations we can push to the edge, the less we have to pay for servers

A cloud mistake can bankrupt your startup!



Hybrid

- Common predictions are precomputed and stored on device
- Local data centers: e.g. each warehouse has its own server rack
- Predictions are generated on cloud and cached on device

Challenges of ML on the edge

- Device not powerful enough to run models
  - Energy constraint
  - Computational power constraint
  - Memory constraint

Challenges of ML on the edge

1. Hardware: Make hardware more powerful
2. Model compression: Make models smaller
3. Model optimization: Make models faster

Make hardware more powerful: big companies

Musk Boasts Tesla Has 'Best Chip in the World'

The CEO's newest big prediction: that Tesla will have self-driving cars on the road next year.

Bloomberg
APR 23, 2019

⚠️ unreliable narrator ⚠️

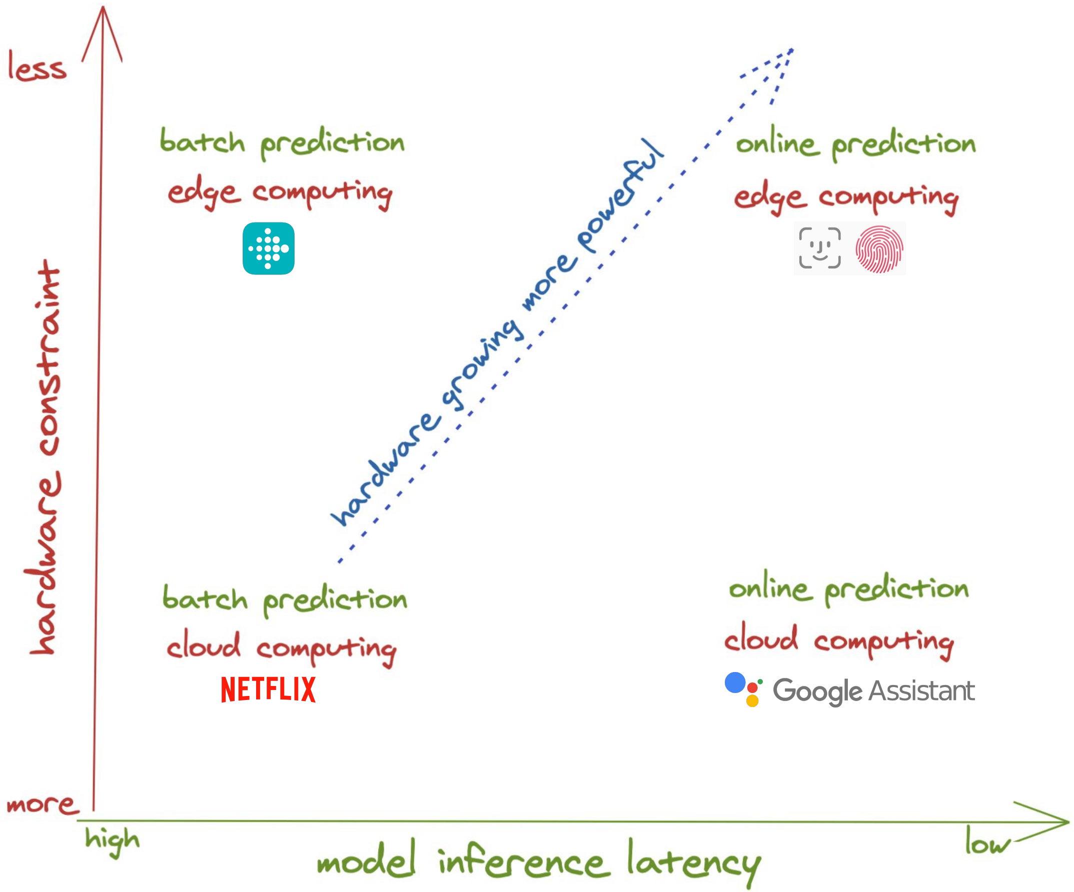
Apr 14, 2020 - Technology

Scoop: Google readies its own chip for future Pixels, Chromebooks

Make hardware more powerful: startups

Hardware startup	Raised ($M)	Year founded	Location
SambaNova	1100	2017	Bay Area
Graphcore	682	2016	UK
Groq	362	2016	Bay Area
Nuvia	293	2019	Bay Area
Wave Computing	203	2008	Bay Area
Cambricon	200	2016	China
Cerebras	112	2016	Bay Area
Hailo	88	2017	Israel
Habana Labs	75	2016	Israel
Kneron	73	2015	San Diego
Prophesee	65	2014	France
Syntiant	65	2017	LA
Groq	62	2016	Bay Area
EdgeQ	53	2018	Bay Area
LeapMind	50	2012	Japan

Future of ML: online and on-device



3. Optimizing Models for Edge

"With PyTorch and TensorFlow, you've seen the frameworks sort of converge. The reason quantization comes up, and a bunch of other lower-level efficiencies come up, is because the next war is compilers for the frameworks — XLA, TVM, PyTorch has Glow, a lot of innovation is waiting to happen," he said. "For the next few years, you're going to see ... how to quantize smarter, how to fuse better, how to use GPUs more efficiently, [and] how to automatically compile for new hardware."

Soumith Chintala, creator of PyTorch (VentureBeat, 2020)

How to run model on different hardware backends?

→ classifier → "tabby cat"

PyTorch TensorFlow

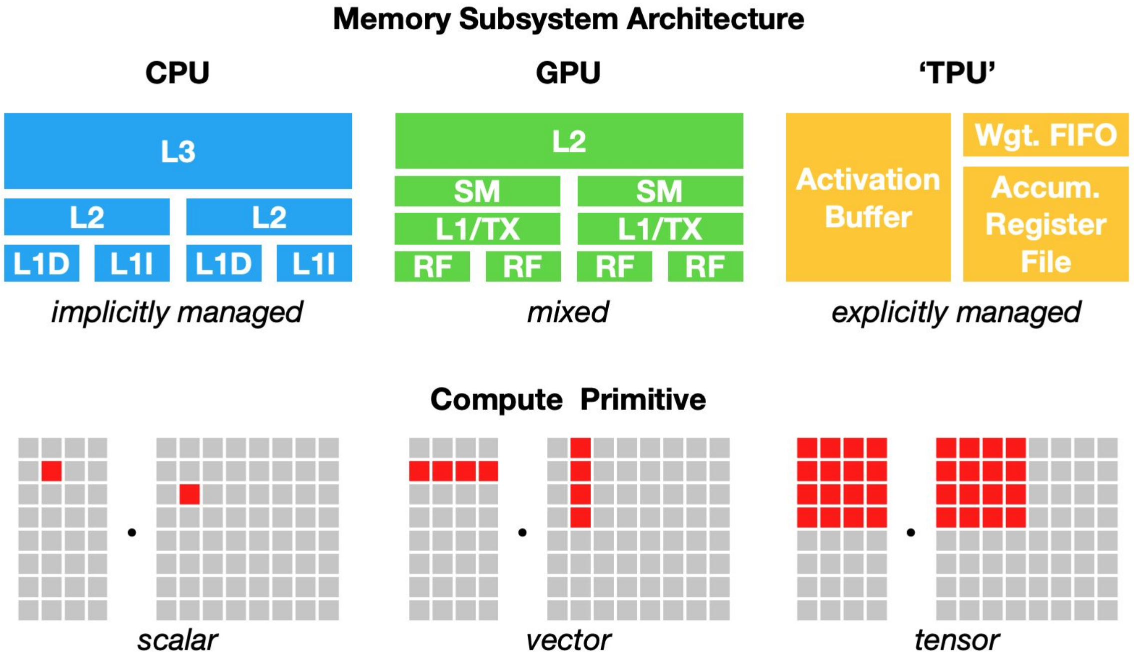
scikit learn LightGBM

mxnet JAX

Thanks Chris Hoge for the cute slide!

Backends: memory layout + compute primitives

Deep learning models	high-dim instructions
Hardware backend	low-dim compute primitives



TVM: An Automated End-to-End Optimizing Compiler for Deep Learning (Chen et al., 2018)

1. Compatibility

Growing

Backend

Frontend

PyTorch	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
scikit learn	?	?	?	?	?	?	?	?	?	?
LightGBM	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
JAX	?	?	?	?	?	?	?	?	?	?

Growing

2. Performance across frameworks

```
df = pd.read_csv("train.csv")
filtered = df.dropna()
features = np.mean(filtered)
model.fit(features)
```

No end-to-end optimization across frameworks

2. Performance

```
df = pd.read_csv("train.csv")
filtered = df.dropna()
features = np.mean(filtered)
model.fit(features)
```

Typical data science workloads using NumPy, Pandas and TensorFlow run 23× slower one thread compared to hand-optimized code
(Palkar et al., ‘18)

Evaluating End-to-End Optimization for Data Analytics Applications in Weld (Palkar et al., VLDB 2018). Slide inspired by Palkar’s talk.

Frontend & backend

Backend

Frontend

PyTorch

TensorFlow

scikit learn

LightGBM

mxnet

- Framework developers:
  - Offer support across a narrow range of server-class hardware
- Hardware vendors:
  - Offer their own SDK / kernel libraries for a narrow range of frameworks (CUDA, OpenVino toolkit, etc.)

→ Hardware lock-in

Optimizing compilers: lowering & optimizing

Generating

Compatibility	**Lowering** hardware-native code for your models
Performance	**Optimizing** your models to run on that hardware

Compatibility

Growing

Backend

Frontend

Growing

PyTorch	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
scikit learn	?	?	?	?	?	?	?	?	?	?
LightGBM	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
JAX	?	?	?	?	?	?	?	?	?	?

Compatibility: bridging frontend & backend

PyTorch

TensorFlow

scikit learn

LightGBM

mxnet

Common language

Intermediate Representations

Machine code

Compatibility: different IR levels

PyTorch

TensorFlow

scikit learn

LightGBM

mxnet

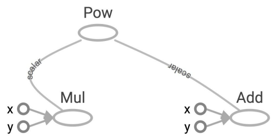
Intermediate Representations

High-level IRs → Tuned IRs → Low-level IRs → Machine code

Computation graphs
Hardware agnostic
E.g.: XLA HLO, TensorFlowLite, TensorRT

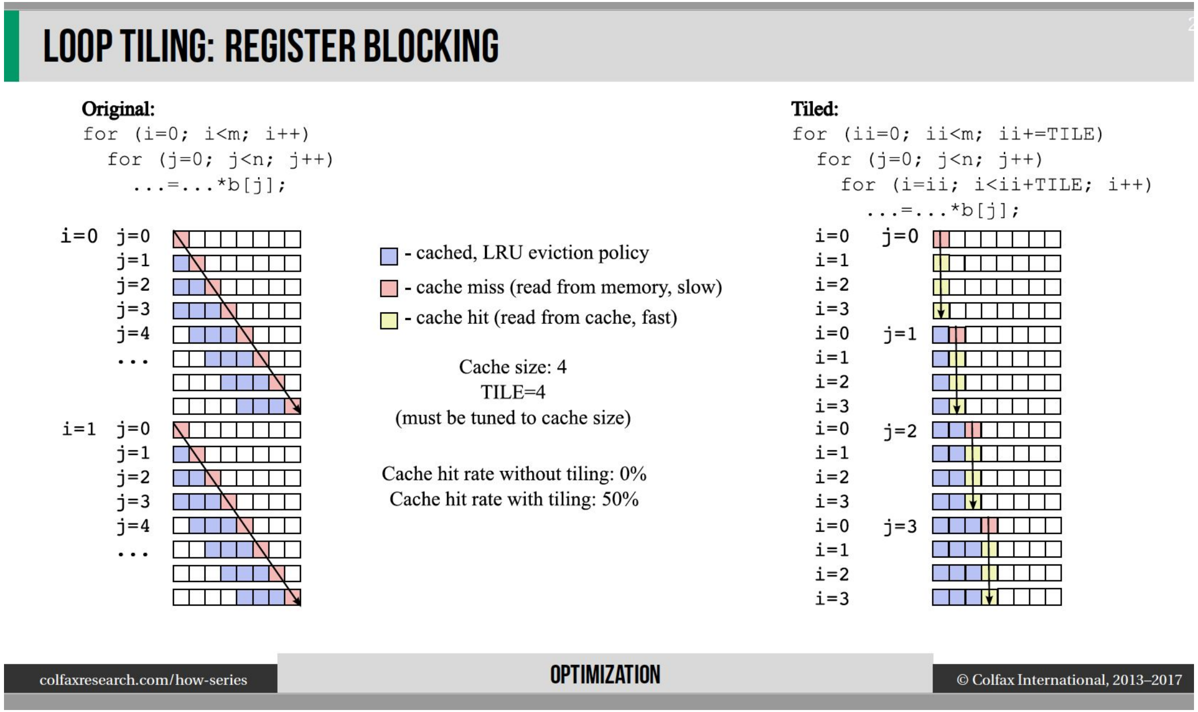
- Hand-tuned
- ML-based

Language agnostic
E.g.: LLVM, NVCC

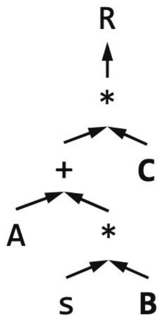


Performance: how to optimize your models

- Standard optimizations
  - vectorization
  - loop tiling
  - explicit parallelism
  - cache
  - etc.



Operator fusion




```
for( i in 1:n )
  tmp1[i,1] = s * B[i,1];
for( i in 1:n )
  tmp2[i,1] = A[i,1] + tmp1[i,1];
for( i in 1:n )
  R[i,1] = tmp2[i,1] * C[i,1];
```

```
for( i in 1:n )
  R[i,1] = (A[i,i] + s*B[i,1]) * C[i,1];
```


https://mboehm7.github.io/teaching/ss19_amls/04_AdvancedCompilation.pdf

Operator fusion

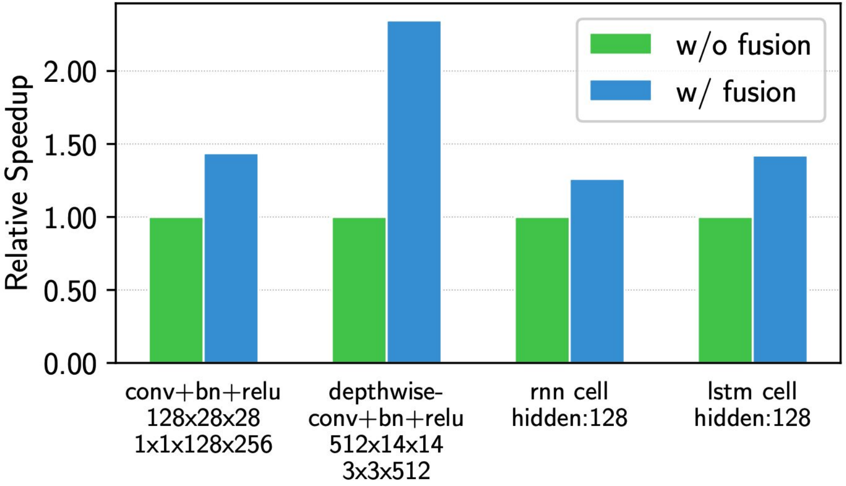
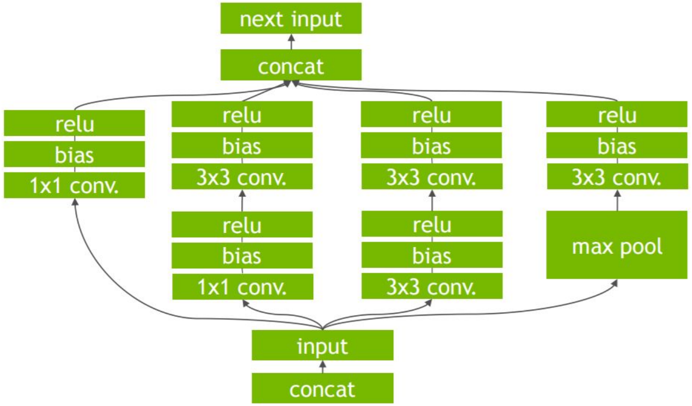


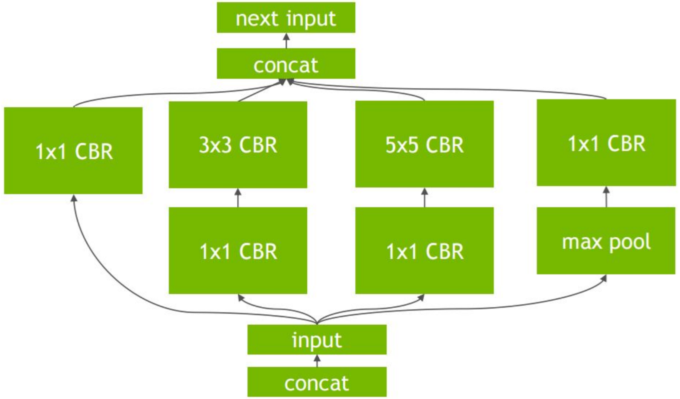
Figure 4: Performance comparison between fused and non-fused operations. TVM generates both operations. Tested on NVIDIA Titan X.

TVM: An Automated End-to-End Optimizing Compiler for Deep Learning (Chen et al., 2018)

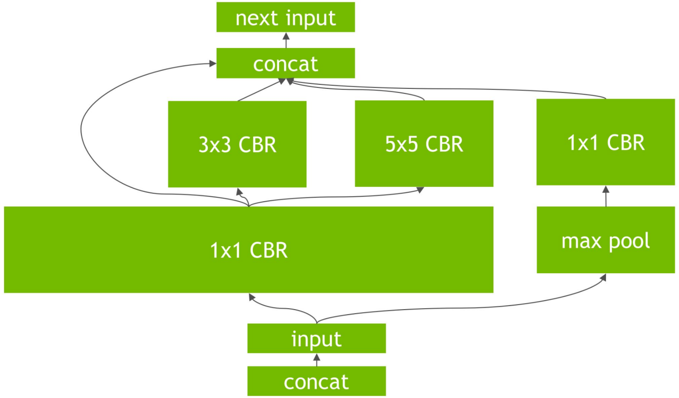
Graph optimization



Original graph



Vertical fusion



Horizontal fusion

https://developer.nvidia.com/tensorrt

Why is it hard?

- Hardware-dependent
  - Different processing/memory/cache/latency hiding
  - Different compute primitives
  - Different instruction sets (RISC-V, ARM, x86, etc.)
- Operator-dependent
- New models being developed all the time
- Many possible paths to execute a graph

Why is it hard?

- Hardware-dependent
  - Different processing/memory/cache/latency hiding
  - Different compute primitives
  - Different instruction sets
- Operator-dependent
- New models being developed all the time
- Many possible paths to execute a graph

Hand-tuned:

- Heuristics-based (non-optimal)
- Non-adaptive
  - **Custom hardware? New framework? New model?**

Idea: automate the optimization process

- What if we explore all possible paths to find the optimal path?
    - Run each path end-to-end to find out how long it takes to execute the path
    - Too slow because of too many possible paths

Idea: automate the optimization process

- What if we explore all possible paths to find the optimal path?
    - Run each path end-to-end to find out how long it takes to execute the path
    - Too slow because of too many possible paths

Combinatorial search
problem

AutoScheduler

- What if we explore all possible paths to find the optimal path?
    - ~~Run each path end-to-end to find out how long it takes to execute the path~~
    - ~~Too slow because of too many possible paths~~
    - Use ML to solve it: narrow down the search space to find approximately the optimal one

AutoScheduler

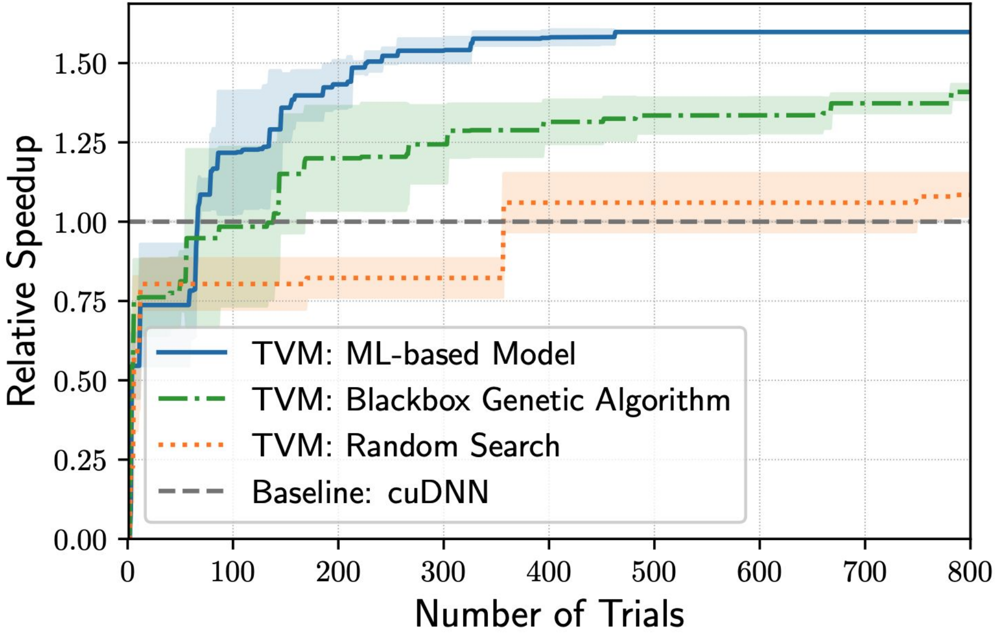
1. Break the graph into subgraphs
2. Predict how big each subgraph is
3. Allow time for each subgraph
4. Stitch them together

AutoScheduler

- cuDNN autotune:
  - for PyTorch on GPU
  - operator-level (only selecting convolutional operator)
- TVM’s autoscheduler:
  - multiple frameworks / multiple hardware
    - automatically adapt to hardware type
  - subgraph-level

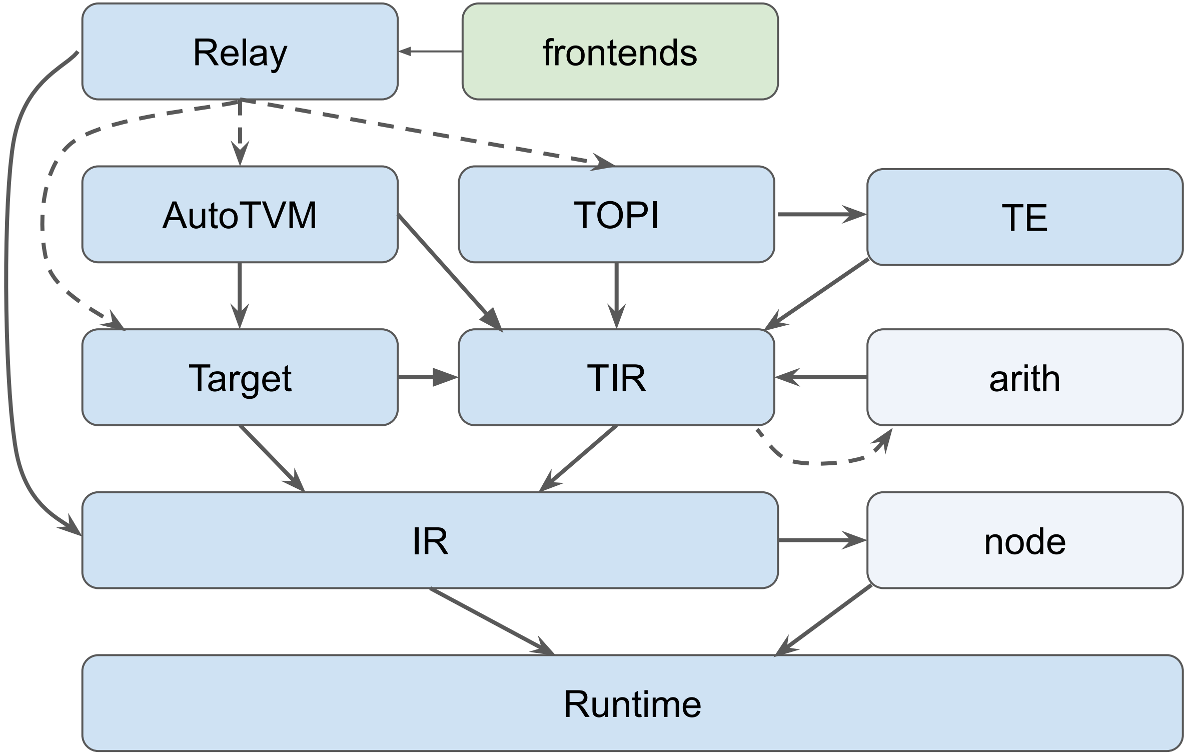
AutoTVM: GPUs

Conv2d operator in ResNet-18 on TITAN X



[TVM: An Automated End-to-End Optimizing Compiler for Deep Learning]() (Chen et al., 2018)

TVM: compiler stack



https://tvm.apache.org/

TVM: Apache OSS

- Compile time:
  - might be slow (lots of paths to explore/evaluate)
  - hours, even days
- Compile once, no need to update even when weights are updated
  - especially useful when you have multiple copies of models on multiple machines

Install TVM the Easy Way - tlcpack.ai

About TLCPack

TLCPack – Tensor learning compiler binary package. It is a community maintained binary builds of deep learning compilers. TLCPack does not contain any additional source code release. It takes source code from Apache TVM and build the binaries by turning on different build configurations. Please note that additional licensing conditions may apply(e.g. CUDA EULA for the cuda enabled package) when you use the binary builds.

TLCPack is not part of Apache and is run by thirdparty community volunteers. Please refer to the official Apache TVM website for Apache source releases.

Licenses for TVM and its dependencies can be found in the github repository.

Build	0.8.dev107+g7b11b9217	Nightly	
Your OS	Linux	Mac	Windows
Package	Conda	Pip	Source
CUDA	10.0	None	
Run this Command:	`conda install tlcpack-nightly -c tlcpack`		

Compatibility: browsers

PyTorch

TensorFlow

scikit learn

LightGBM

mxnet

Browser

ML in browsers

- Compile models to JavaScript
    - TensorFlow.js, Synaptic, and brain.js

ML in browsers

- Compile models to JavaScript
- Compile models to WASM (WebAssembly)
  - Open standard that allows running executable programs in browsers
  - Supported by 93%

https://caniuse.com/wasm

Machine Learning Systems Design

Modeling Pipeline

Next Lecture: High-Performance Modeling

CE 40959 Spring 2023
Ali Zarezade
SharifMLSD.github.io