Carnegie Mellon University
Language Technologies Institute

Neural Architecture Search

Students are too slow, let's use computers to create networks instead

Yonatan Bisk & Emma Strubell

Overflow

Beyond preserved accuracy: Loyalty & Robustness

- **Label loyalty:** How similar are the predicted labels compared to the full model?

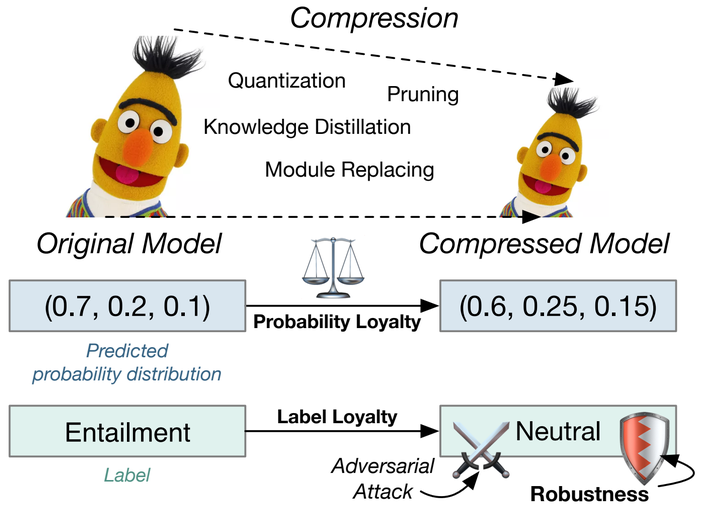
$$L_l = \text{Accuracy}(pred_t, pred_s)$$

- **Probability loyalty:** How similar are the compressed model's output probabilities compared to the full model?

$$D_{\text{KL}}(P\|Q) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right) \qquad L_p(P\|Q) = 1 - \sqrt{D_{\text{JS}}(P\|Q)}$$

$$D_{\text{JS}}(P\|Q) = \frac{1}{2}D_{\text{KL}}(P\|M) + \frac{1}{2}D_{\text{KL}}(Q\|M)$$

- **Robustness:** How robust is the compressed model against an adversarial attack, compared to the full model?



Method	Speed	MNLI	L-L	P-L	AA	# Q
Teacher	1.0×	84.5 / 83.3	100	100	8.1	89.6
Head Prune	1.2×	80.9 / 80.6	87.8	85.5	9.1	90.5
+Finetune	1.2×	83.2 / 81.9	89.1	85.5	7.2	83.2
+KD	1.2×	**84.2 / 83.0**	**93.3**	**93.0**	8.3	90.5
+KD+PTQ	2.2×	80.8 / 80.4	89.6	86.3	**38.4**	**90.9**
Q8-QAT	1.8×	83.4 / 82.4	89.7	88.2	6.8	82.7
Q8-PTQ	1.8×	80.7 / 80.4	89.6	80.8	**40.2**	**91.6**
+Finetune	1.8×	82.9 / 81.9	89.7	84.8	7.1	84.5
+KD	1.8×	**84.1 / 83.5**	**94.0**	**93.9**	7.5	86.1
BERT-PKD	2.0×	81.3 / 81.1	88.9	89.0	6.4	81.9
Theseus	2.0×	81.8 / 80.7	88.1	82.5	8.3	89.7
+KD	2.0×	**82.6 / 81.7**	**91.2**	**91.4**	8.0	88.7
+KD+PTQ	3.6×	80.2 / 79.9	89.5	80.3	**36.5**	**91.3**

Xu, Zhou, Ge, Xu, McAuley, Wei. Beyond Preserved Accuracy: Evaluating Loyalty and Robustness of BERT Compression. EMNLP 2021.

Adaptive computation

- Original idea: Deploy smaller, simpler models for easier examples, larger, more complex models for difficult examples.

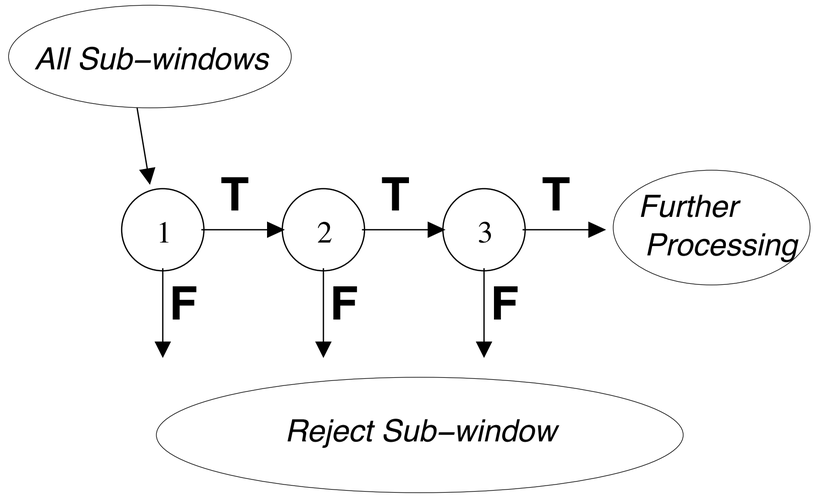
ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION 2001

CVPR 2001

Rapid Object Detection using a Boosted Cascade of Simple Features

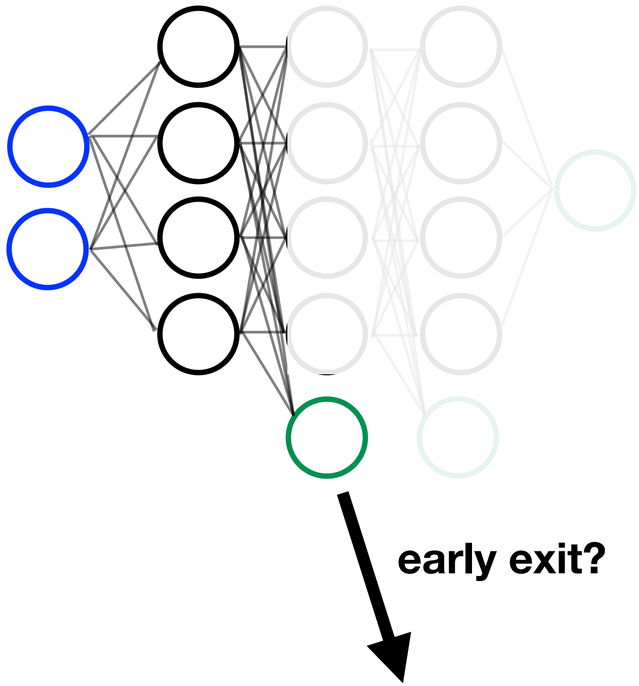
Paul Viola
viola@merl.com
Mitsubishi Electric Research Labs
201 Broadway, 8th FL
Cambridge, MA 02139

Michael Jones
mjones@crl.dec.com
Compaq CRL
One Cambridge Center
Cambridge, MA 02142



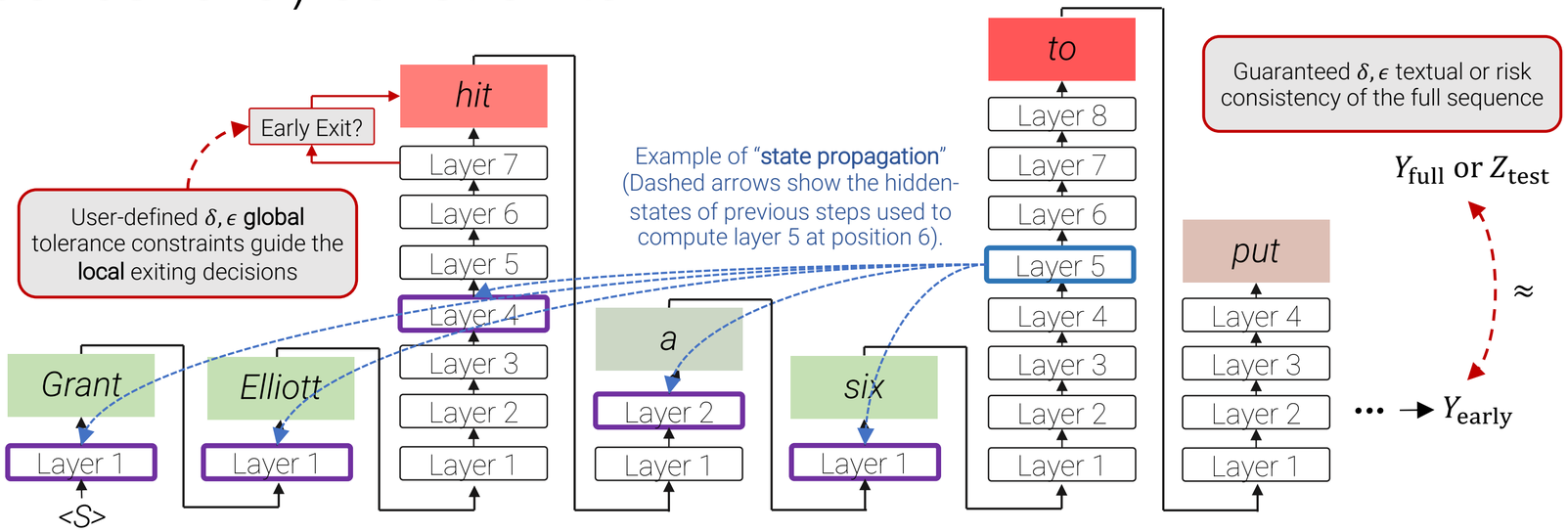
Adaptive computation

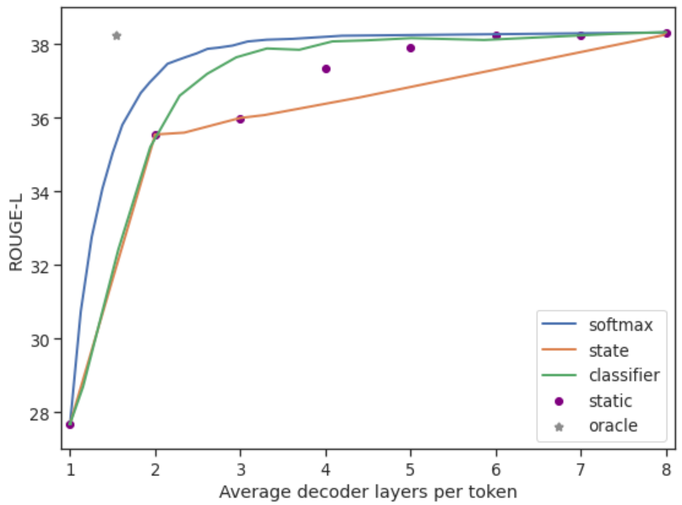
- Orig. idea: Deploy smaller, simpler models for easier examples, larger, more complex models for difficult examples.
- **Key idea:** Use more or fewer parameters depending on example difficulty.
- Ideally, parameters are shared (~self-distillation).
- Modeling decisions / challenges:
  - How to (efficiently) compute **confidence** / early stopping criterion?
  - **Batching** to maximize compute throughput on accelerators?



CALM: Confident Adaptive Language Modeling

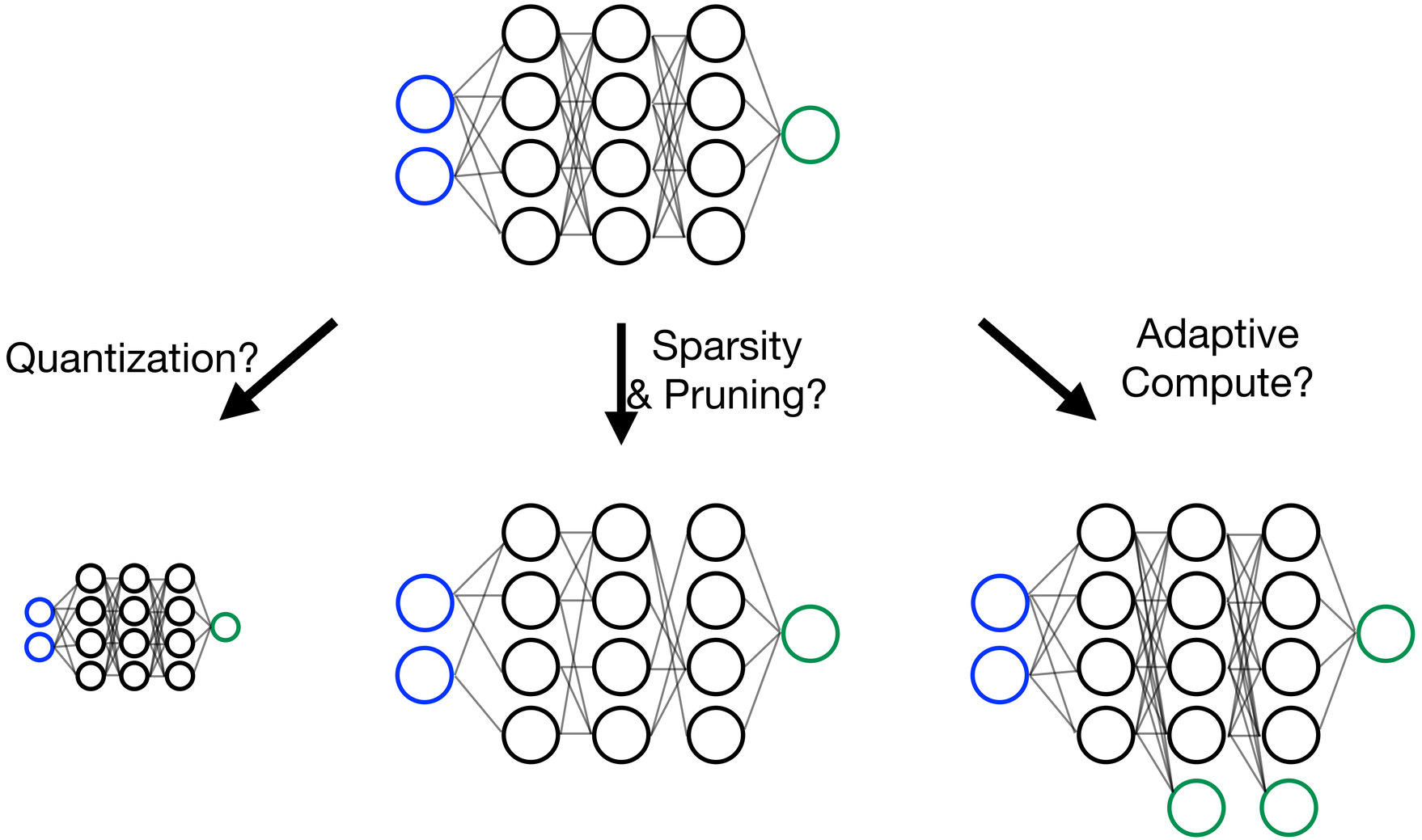
- **Key idea:** Calibrate local, per-token early exit decisions using global (sequence-level) constraints.



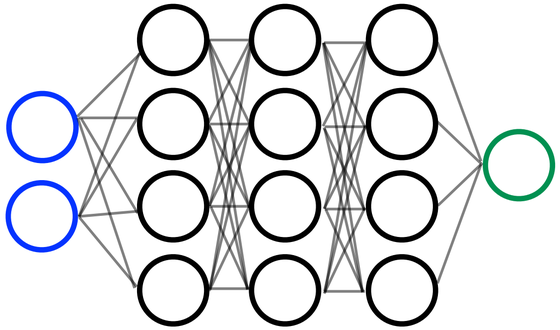


T. Schuster, A. Fisch, J. Gupta, M. Dehghani, D. Bahri, V. Q. Tran, Y. Tay, D. Metzler. Confident Adaptive Language Modeling. NeurIPS 2022.

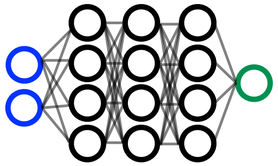
Model compression: Overview



Model compression: How to choose?

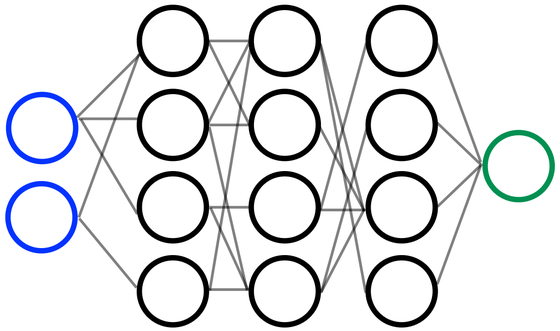


Quantization



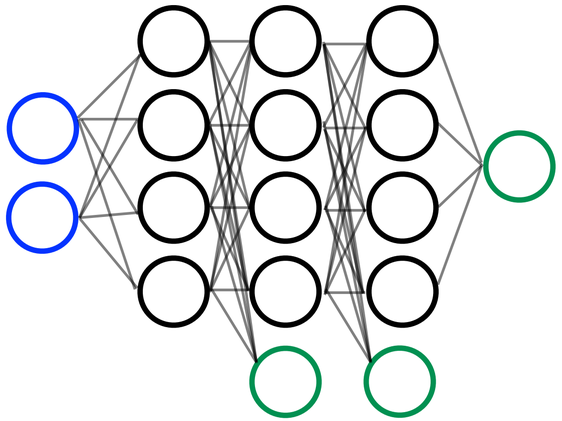
Pro: Fewer Bits/Memory
Con: Requires hardware support for efficiency, otherwise you're pretending which is probably slower.

Sparsity & Pruning



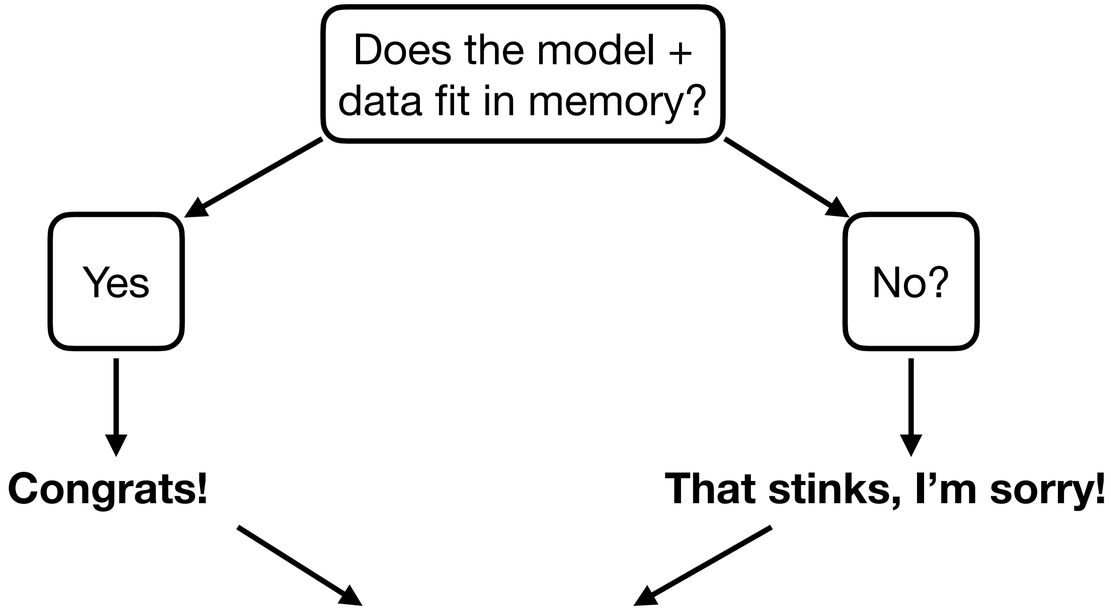
Pro: Substantially fewer parameters and operations
Con: Requires hardware support for Sparse Ops

Adaptive Compute



Pro: Early exit, process only if it changes the outcome
Con: Requires additional control flow logic. Better suited to CPU than GPU

Model compression: Is hardware the answer?



Now go worry about:
- Power consumption
- Wall clock
- Vector ops
- Competition with OS
- Model accuracy
- Other platforms …

CPUs:
- Designed for control flow
- Vector ops (very limited)
- Small cache, reliant on system RAM

GPUs:
- No control flow
- Large matrix multiplies
- Mixed precision support
- Large RAM
- Recent! devices have sparse ops

TPUs:
- No control flow
- Large matrix multiplies
- Large RAM
- Native mixed & low precision

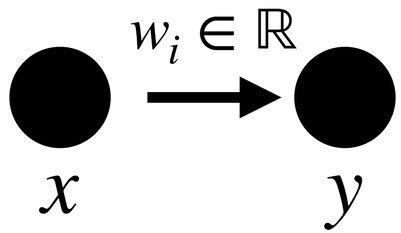
FPGAs:
- Whatever you want boss

Back to NAS

Evolution and Fitness

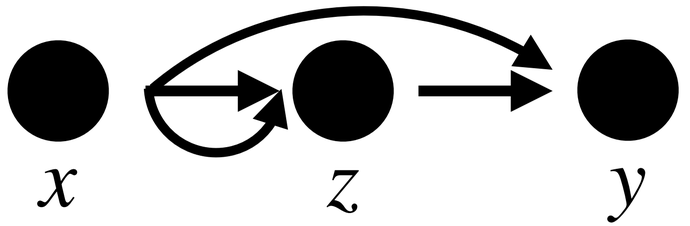
Search and Loss

- Given n functions:
  - Evaluate them
  - Choose the best m
  - “Change” them to create n
  - Repeat



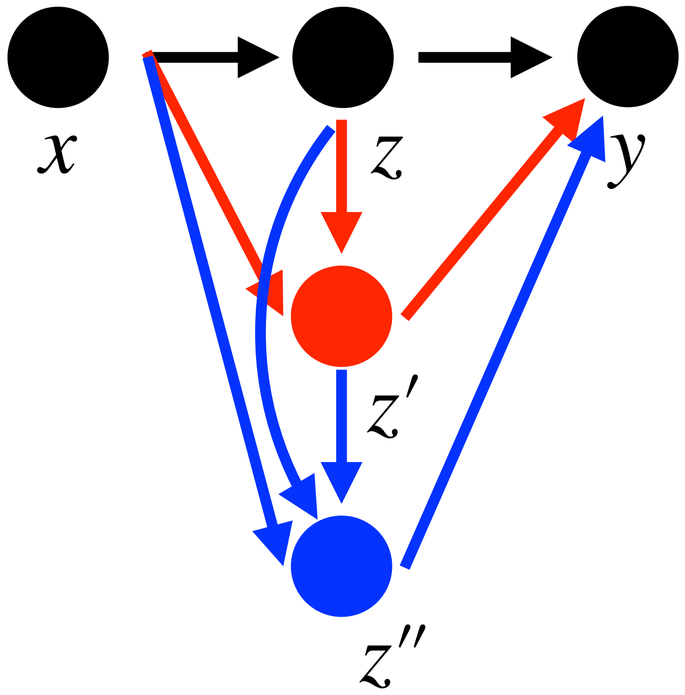
“Changes”:

- $w_i \pm \delta$
- Add edge
- Add node



Cascade Correlation

Fahlman and Lebiere, N(eur)IPS 1989

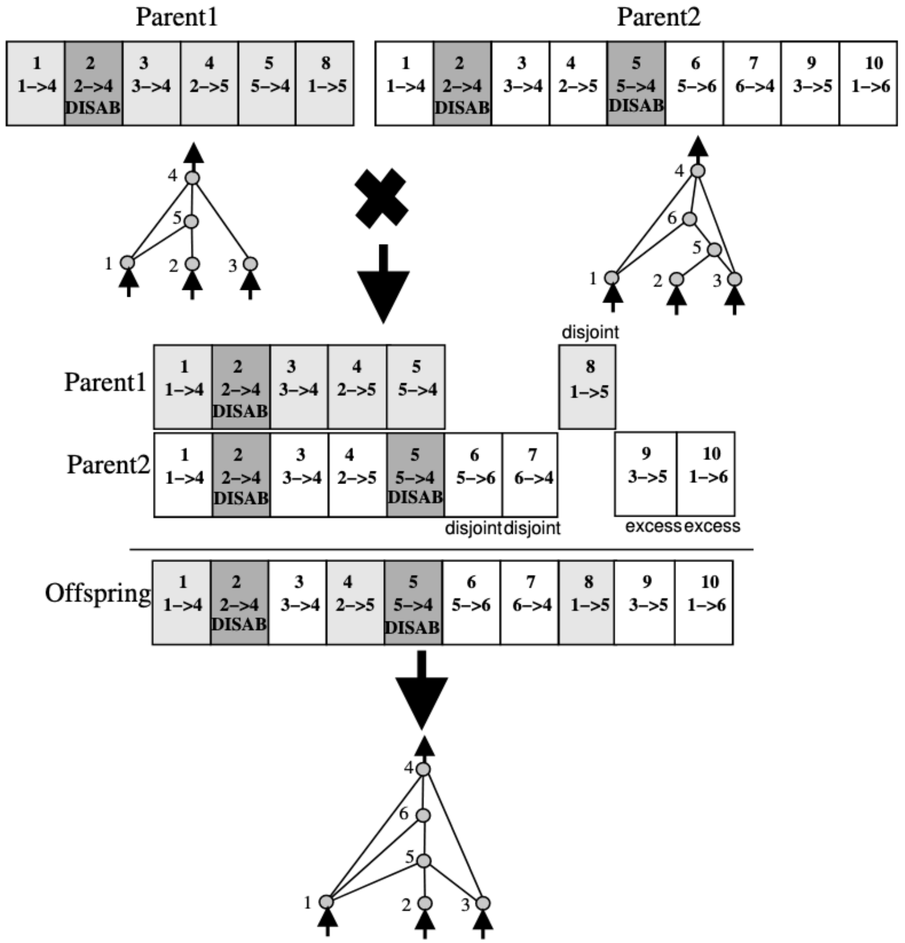


What is the atomic “unit”?

A new layer in the hierarchy

Can Networks Merge?

Efficient Reinforcement Learning through Evolving Neural Network Topologies
Stanley and Miikkulainen — GECCO 2002



Yes! We can encode “genotypes” and “phenotypes"

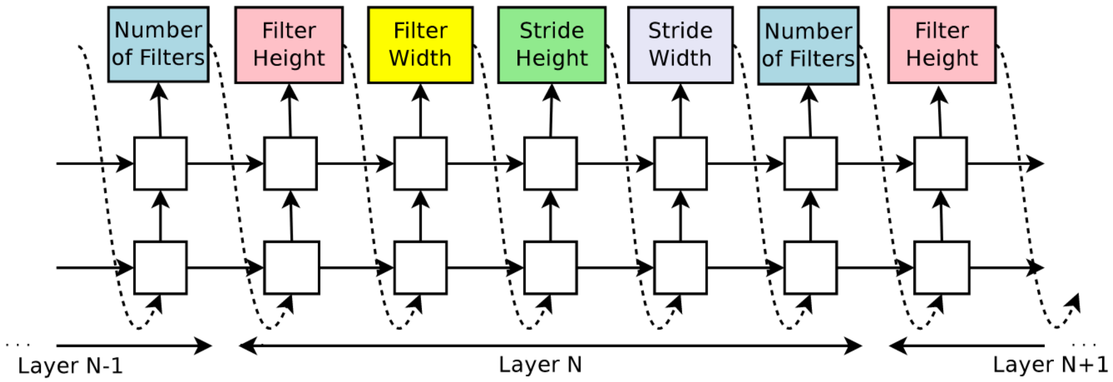
Problems

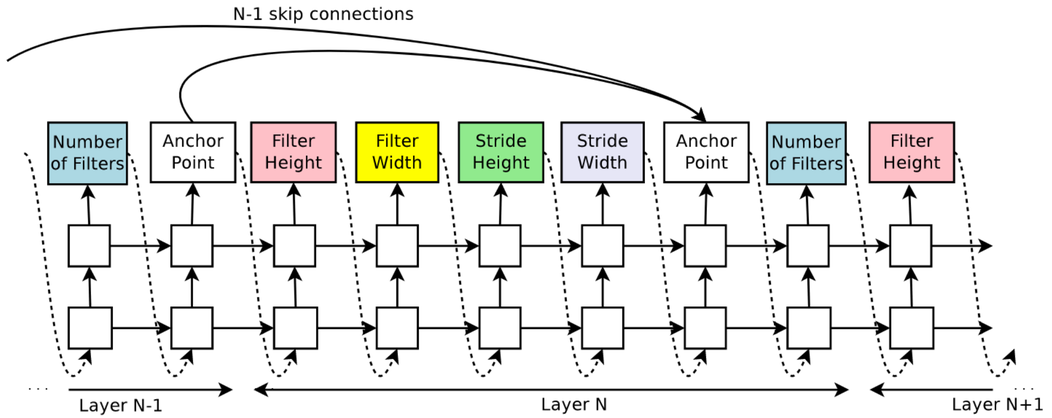
- Should we really be changing weights randomly?
- Should we really be adding/subtracting individual nodes?
- How should you initialize?
- Is merging networks worth it?
- Is this expensive?

Need to define the space

- What defines a convolution?
- What defines the conv stack?

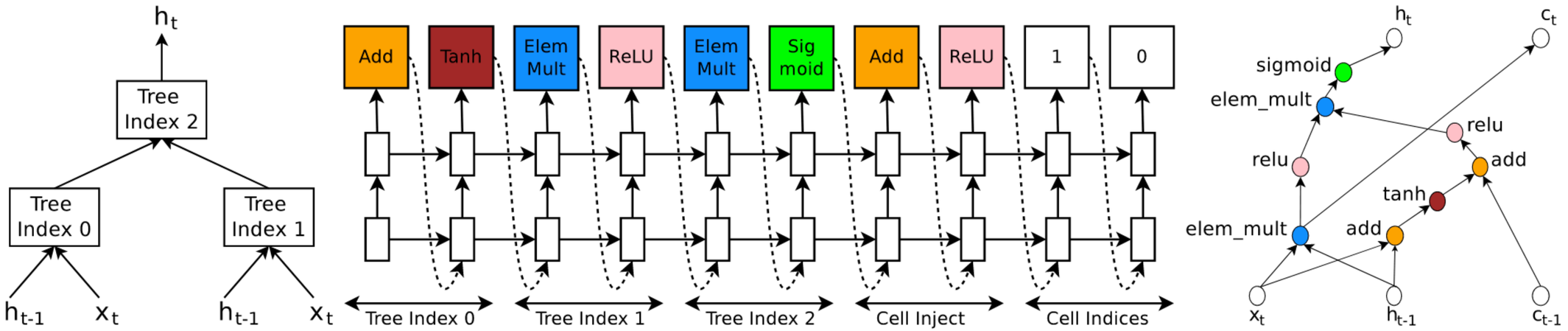
This model can be trained with reinforcement learning based on the performance of the resulting model





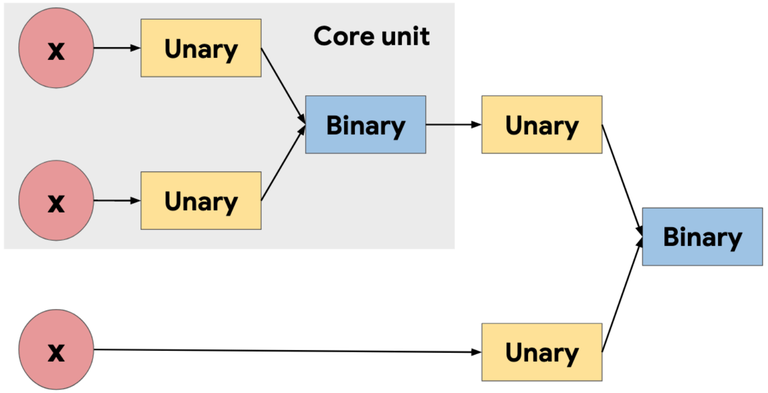
Neural Architecture Search with Reinforcement Learning
https://arxiv.org/abs/1611.01578

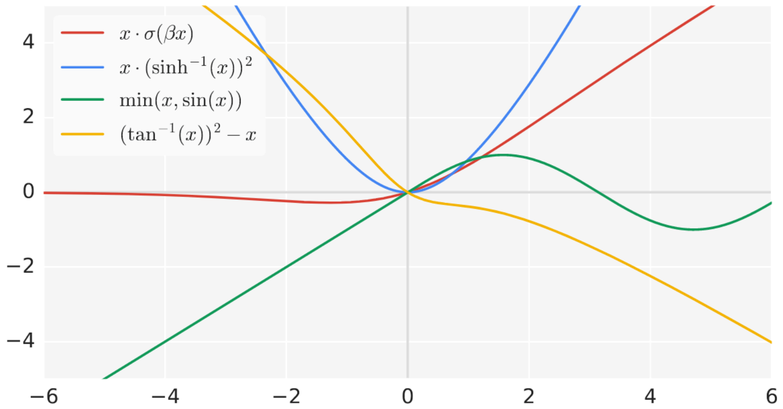
What about recurrence?

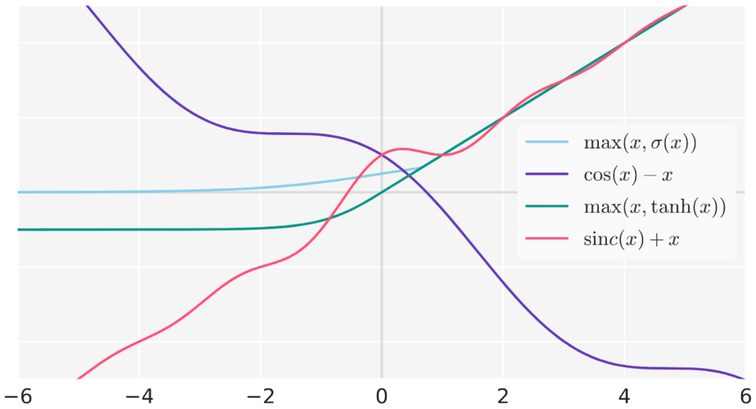


Working with indices instead of layers

Even activation functions







- **Unary functions**: x, $-x$, $|x|$, x^2, x^3, $\sqrt{x}$, βx, $x + \beta$, $\log(|x| + \epsilon)$, $\exp(x)$ $\sin(x)$, $\cos(x)$, $\sinh(x)$, $\cosh(x)$, $\tanh(x)$, $\sinh^{-1}(x)$, $\tan^{-1}(x)$, $\mathrm{sinc}(x)$, $\max(x, 0)$, $\min(x, 0)$, $\sigma(x)$, $\log(1 + \exp(x))$, $\exp(-x^2)$, $\mathrm{erf}(x)$, β

- **Binary functions**: $x_1 + x_2$, $x_1 \cdot x_2$, $x_1 - x_2$, $\frac{x_1}{x_2 + \epsilon}$, $\max(x_1, x_2)$, $\min(x_1, x_2)$, $\sigma(x_1) \cdot x_2$, $\exp(-\beta(x_1 - x_2)^2)$, $\exp(-\beta|x_1 - x_2|)$, $\beta x_1 + (1 - \beta)x_2$

Searching for Activation Functions
https://arxiv.org/abs/1710.05941

Top-level comments

1. You define the structure and the space
2. How may samples do you need for accurate estimates?
3. How expensive is each sample?
4. If RL, “goodness” measure can be anything you want
5. When are you overfitting?

Method	GPUs	Times (days)	Params (million)	Error (%)
DenseNet-BC (Huang et al., 2016)	–	–	25.6	3.46
DenseNet + Shake-Shake (Gastaldi, 2016)	–	–	26.2	2.86
DenseNet + CutOut (DeVries & Taylor, 2017)	–	–	26.2	**2.56**
Budgeted Super Nets (Veniat & Denoyer, 2017)	–	–	–	9.21
ConvFabrics (Saxena & Verbeek, 2016)	–	–	21.2	7.43
Macro NAS + Q-Learning (Baker et al., 2017a)	10	8-10	11.2	6.92
Net Transformation (Cai et al., 2018)	5	2	19.7	5.70
FractalNet (Larsson et al., 2017)	–	–	38.6	4.60
SMASH (Brock et al., 2018)	1	1.5	16.0	4.03
NAS (Zoph & Le, 2017)	800	21-28	7.1	4.47
NAS + more filters (Zoph & Le, 2017)	800	21-28	37.4	**3.65**
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	**3.87**
Hierarchical NAS (Liu et al., 2018)	200	1.5	61.3	3.63
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3	–	3.60
Progressive NAS (Liu et al., 2017)	100	1.5	3.2	3.63
NASNet-A (Zoph et al., 2018)	450	3-4	3.3	3.41
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4	3.3	**2.65**
ENAS + micro search space	1	0.45	4.6	3.54
ENAS + micro search space + CutOut	1	0.45	4.6	**2.89**

Efficient Neural Architecture Search via Parameters Sharing
http://proceedings.mlr.press/v80/pham18a.html

EfficientNet

Assume we fix our structure

$$Y_i = \mathcal{F}_i(X_i)$$

$$\mathcal{N} = \mathcal{F}_k \odot \ldots \odot \mathcal{F}_2 \odot \mathcal{F}_1(X_i)$$

$$\mathcal{N} = \odot_{i=1\ldots s} \mathcal{F}_i^{L_i}(X_{<H_i, W_i, C_i>})$$

Repeating layer L times for stage i

Constrained Optimization

$$\max_{d,w,r} \quad Accuracy(\mathcal{N}(d,w,r))$$

$$s.t. \quad \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\left(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\right)$$

$$\text{Memory}(\mathcal{N}) \leq \text{target\_memory}$$

$$\text{FLOPS}(\mathcal{N}) \leq \text{target\_flops}$$

Scaling Intuition

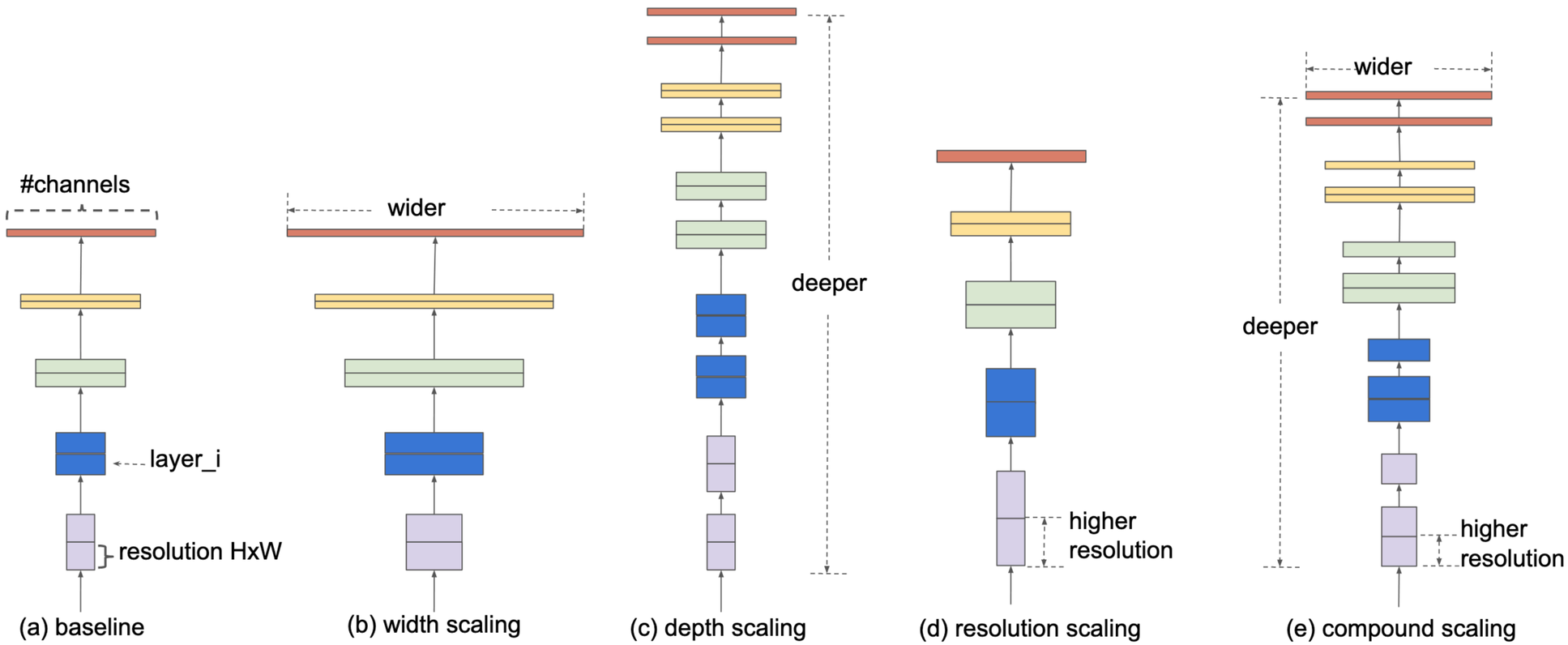


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Simplify Further

Single compound coefficient

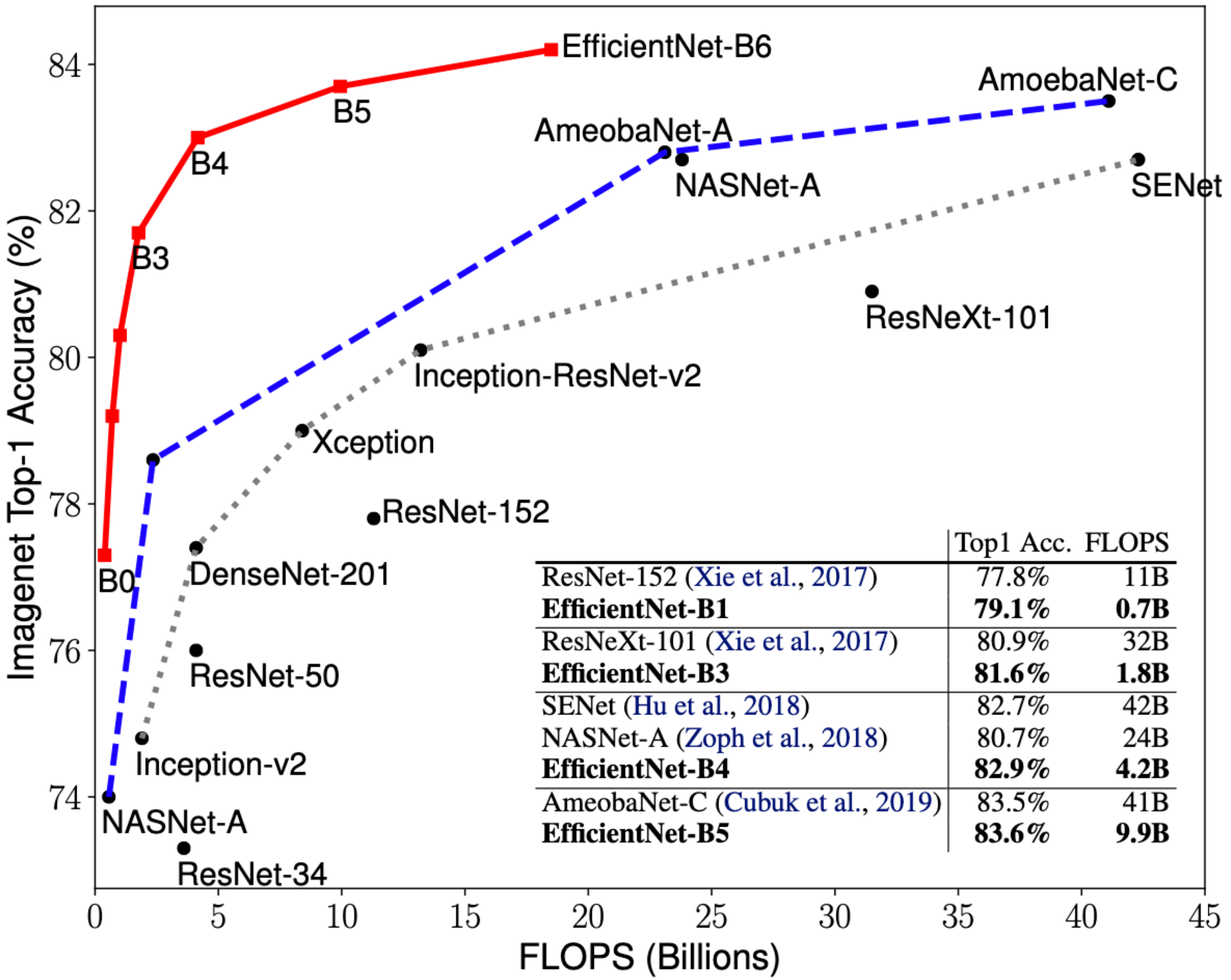
$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

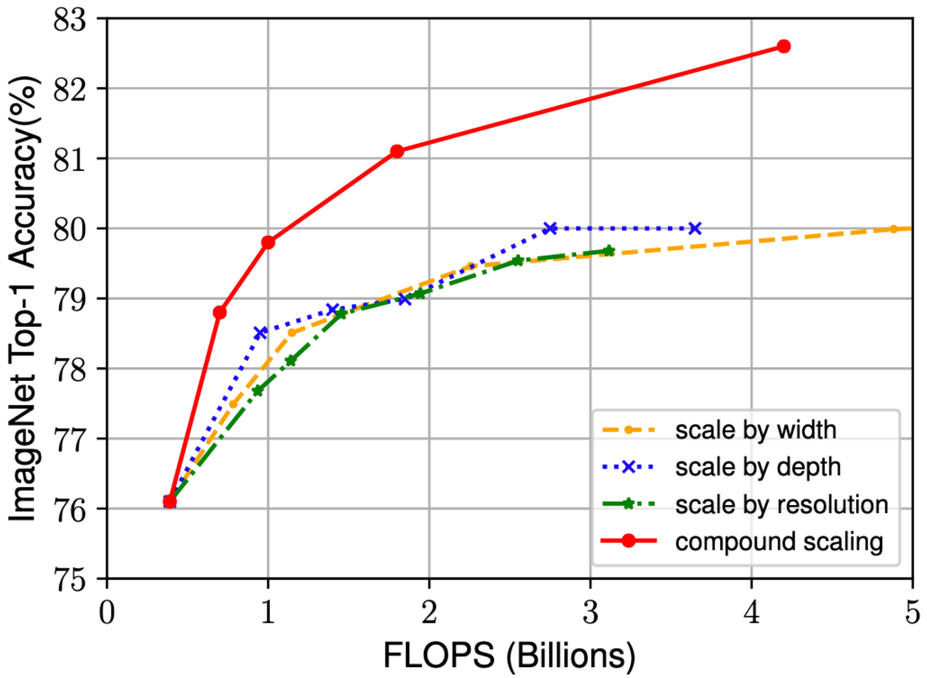
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

Why all the dimensions?



Model	FLOPS	Top-1 Acc.
Baseline model (EfficientNet-B0)	0.4B	77.3%
Scale model by depth (d=4)	1.8B	79.0%
Scale model by width (w=2)	1.8B	78.9%
Scale model by resolution (r=2)	1.9B	79.1%
Compound Scale (d=1.4, w=1.2, r=1.3)	**1.8B**	**81.1%**

Is it worth the cost, and how to measure?

Case Study: Evolved Transformer and Meena

- Evolved Transformer (So et al. 2018):
  - +0.5 BLEU on en-de translation.
  - 1.6X fewer FLOPS and 1.1X–1.3X less time than original Transformer.
- Amortized cost: Did anyone ever use it again, justifying the up-front compute?

But, what about everyone else?

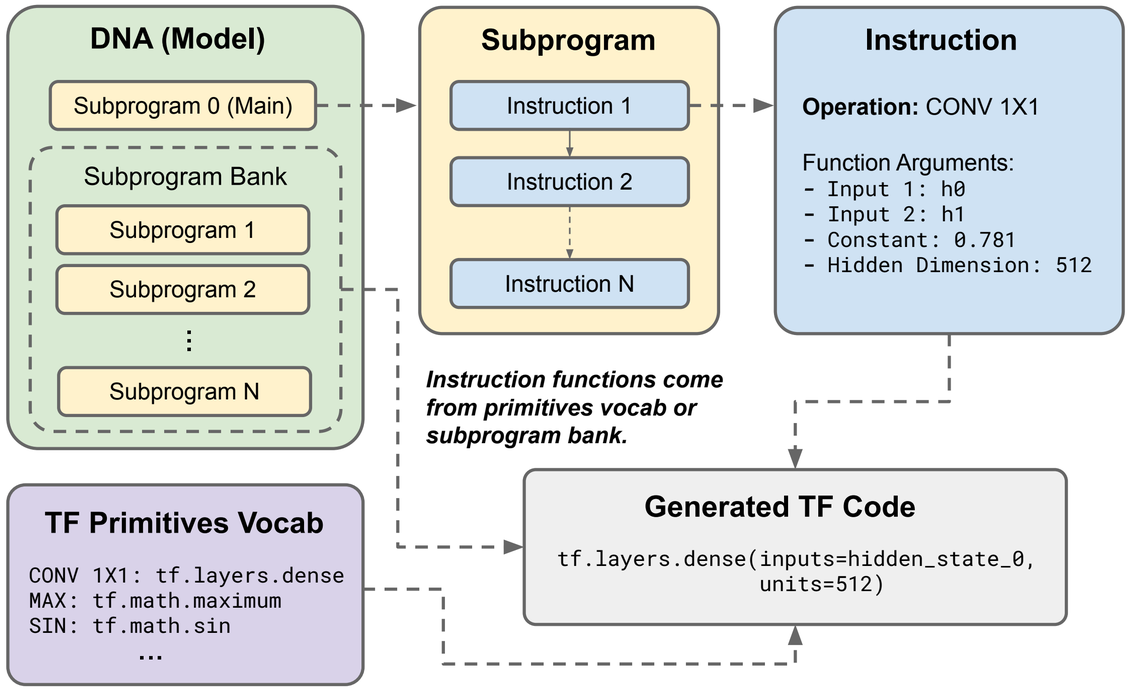
- How to benchmark NAS?

Primer:
Searching for Efficient Transformers for Language Modeling

- Search space: TensorFlow programs.
  - Initialize search space with copies of Transformer.
- Limited compute budget (24 TPUv2 hours) optimizes for efficiency.
  - This means: changes may increase step (inference) time but improve sample (training) efficiency.
- Result: Reduce T5 training cost by 4x.

Primer: Searching for Efficient Transformers for Language Modeling

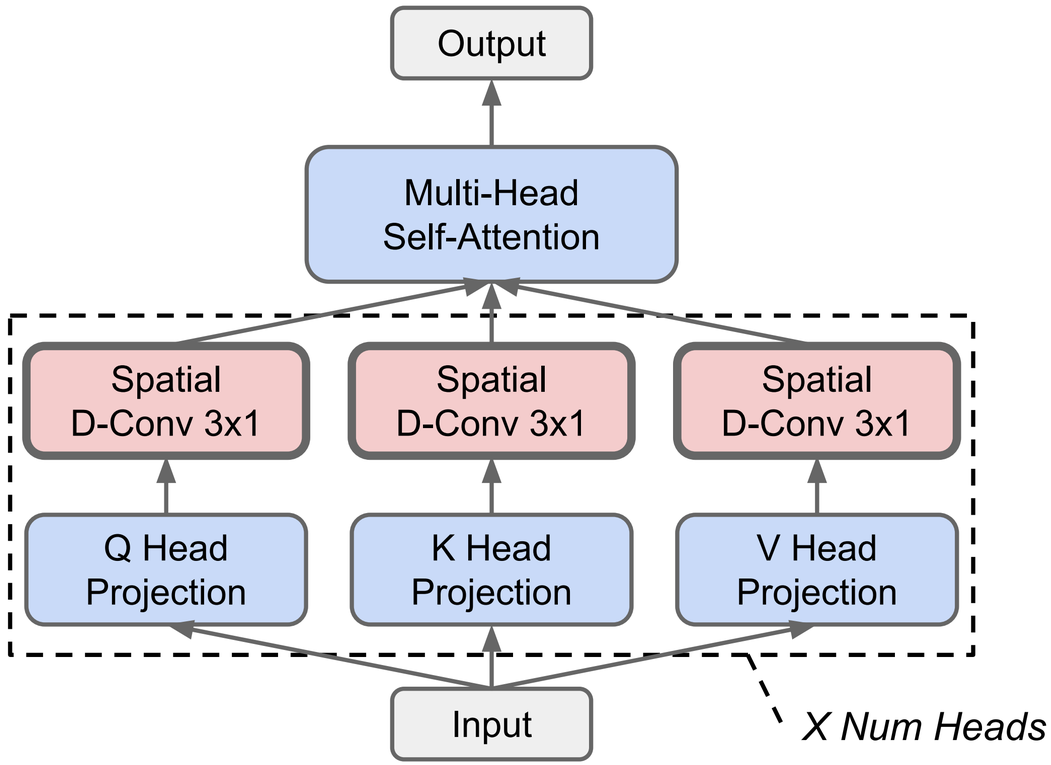
David R. So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, Quoc V. Le
Google Research, Brain Team
{davidso, wojciechm, hanxiaol, zihangd, noam, qvl}@google.com



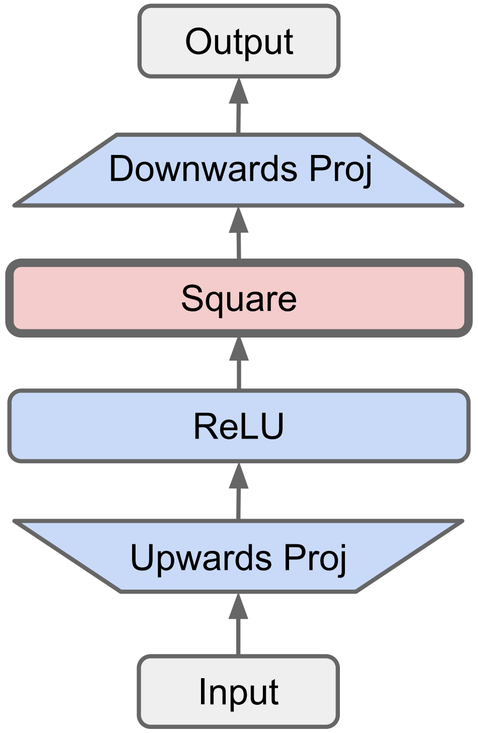
Primer

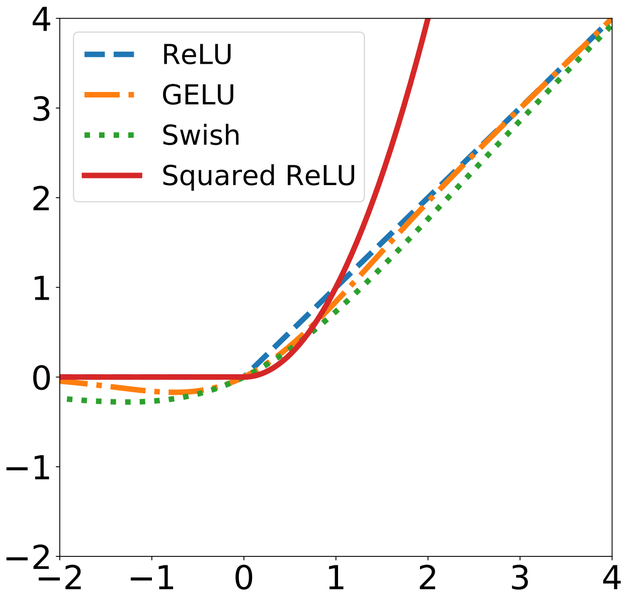
- So, what modifications did it learn?

Multi-DConv-Head Attention (MDHA)



Squared ReLU in Feed Forward Block





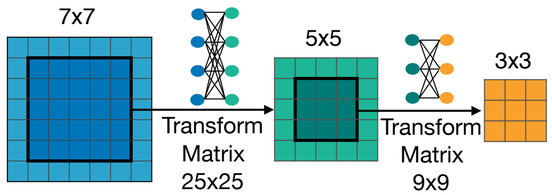
Primer

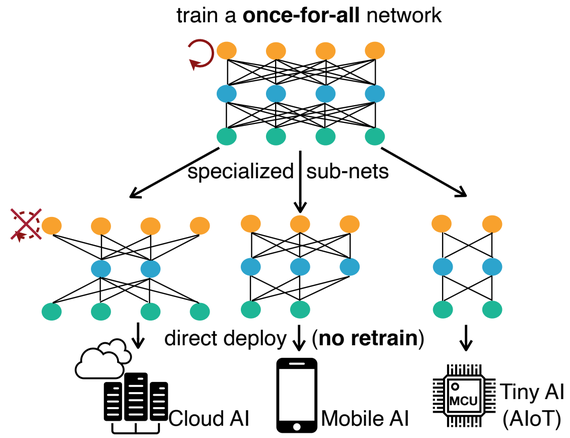
- Amortized cost?

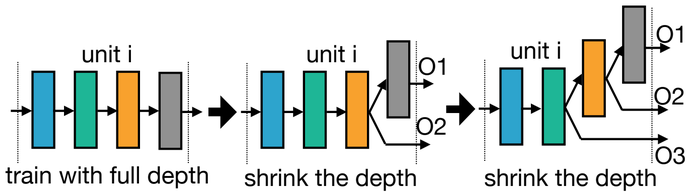
Primer's Return on Investment: The compute savings in this large-scale experiment demonstrate the return on investment for the Primer search. The search for Primer itself cost ∼2.14E+21 FLOPs. Training Transformer for this experiment cost ∼2.96E+22 FLOPs, which means the compute saved by Primer to reach the same performance is ∼1.98E+22 FLOPs. Thus, for this single training, the return on investment for the architecture search is roughly 9.24X.

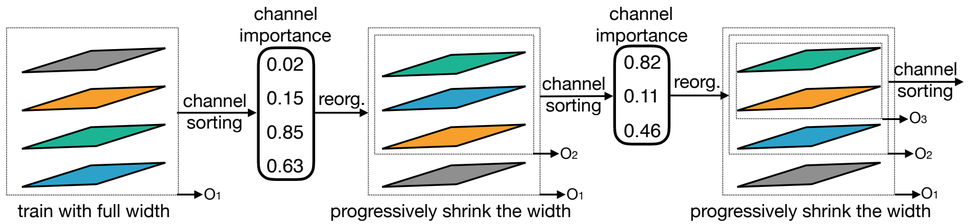
Once-for-All: Train One Network and Specialize it for Efficient Deployment

- **Key idea:** Train once to decouple model training and NAS.
  - Train model to allow for nested models of different sizes by **progressive shrinking**.
  - **Elastic kernel size:** 3x3 conv within 5x5 conv within 7x7… + shared kernel transformation weights.
  - **Elastic depth:** Share weights of first *D* layers.
  - **Elastic width:** Sort channels by importance using L_1 norm of weights, take top *k* channels.
- **Model specialization:** Train hardware-specific accuracy/latency predictors (Liu et al. 2018).







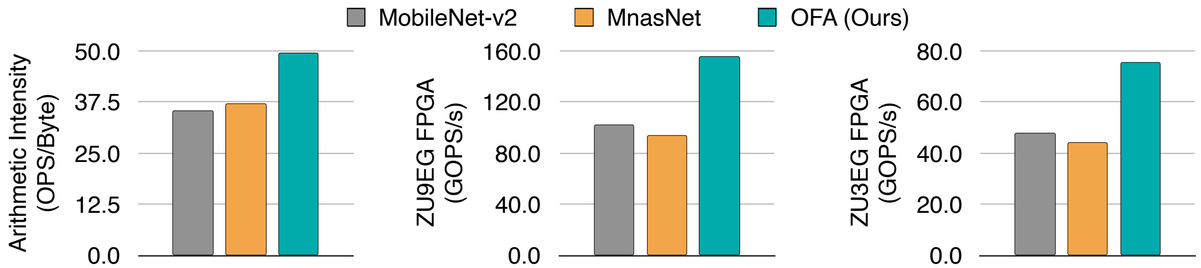


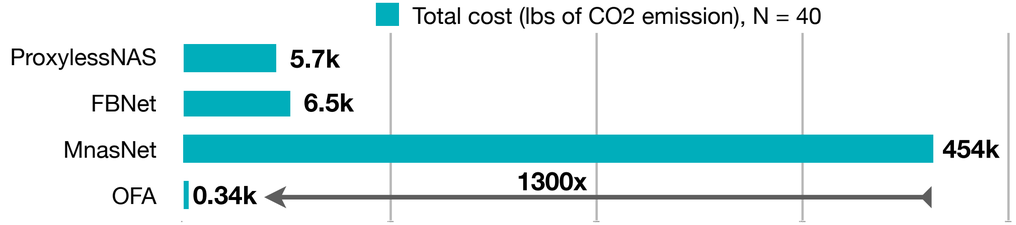
H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han. Once-for-All: Train One Network and Specialize it for Efficient Deployment. ICLR 2020.

Once-for-All: Train One Network and Specialize it for Efficient Deployment

- Evaluation on Pixel 1 phone and Xilinx FPGAs: Better performance w/ less energy.

Model	ImageNet Top1 (%)	MACs	Mobile latency	Search cost (GPU hours)	Training cost (GPU hours)	Total cost ($N = 40$)		
						GPU hours	CO_2e (lbs)	AWS cost
MobileNetV2 [31]	72.0	300M	66ms	0	$150N$	6k	1.7k	$18.4k
MobileNetV2 #1200	73.5	300M	66ms	0	$1200N$	48k	13.6k	$146.9k
NASNet-A [44]	74.0	564M	-	$48{,}000N$	-	1,920k	544.5k	$5875.2k
DARTS [25]	73.1	595M	-	$96N$	$250N$	14k	4.0k	$42.8k
MnasNet [33]	74.0	317M	70ms	$40{,}000N$	-	1,600k	453.8k	$4896.0k
FBNet-C [36]	74.9	375M	-	$216N$	$360N$	23k	6.5k	$70.4k
ProxylessNAS [4]	74.6	320M	71ms	$200N$	$300N$	20k	5.7k	$61.2k
SinglePathNAS [8]	74.7	328M	-	$288 + 24N$	$384N$	17k	4.8k	$52.0k
AutoSlim [38]	74.2	305M	63ms	180	$300N$	12k	3.4k	$36.7k
MobileNetV3-Large [15]	75.2	219M	58ms	-	$180N$	7.2k	1.8k	$22.2k
OFA w/o PS	72.4	235M	59ms	40	1200	1.2k	0.34k	$3.7k
OFA w/ PS	**76.0**	230M	58ms	40	1200	1.2k	0.34k	$3.7k
OFA w/ PS #25	**76.4**	230M	58ms	40	$1200 + 25N$	2.2k	0.62k	$6.7k
OFA w/ PS #75	**76.9**	230M	58ms	40	$1200 + 75N$	4.2k	1.2k	$13.0k
$\text{OFA}_{\text{Large}}$ w/ PS #75	**80.0**	595M	-	40	$1200 + 75N$	4.2k	1.2k	$13.0k

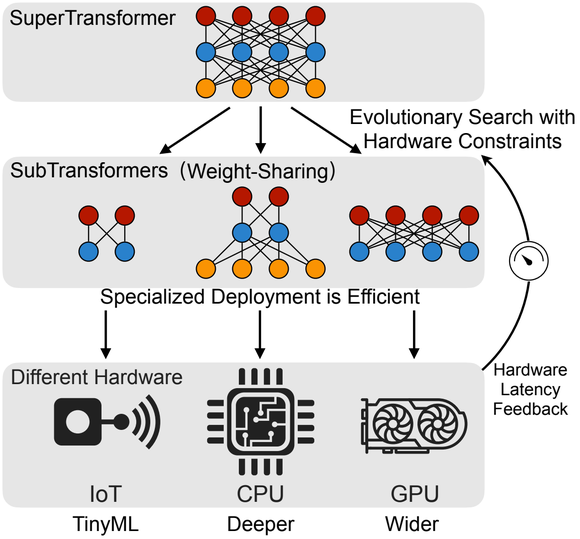


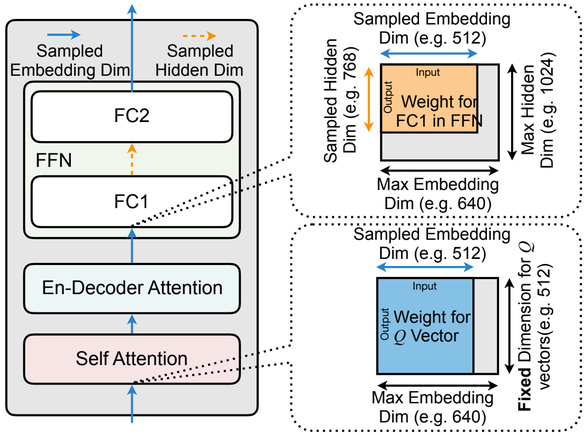


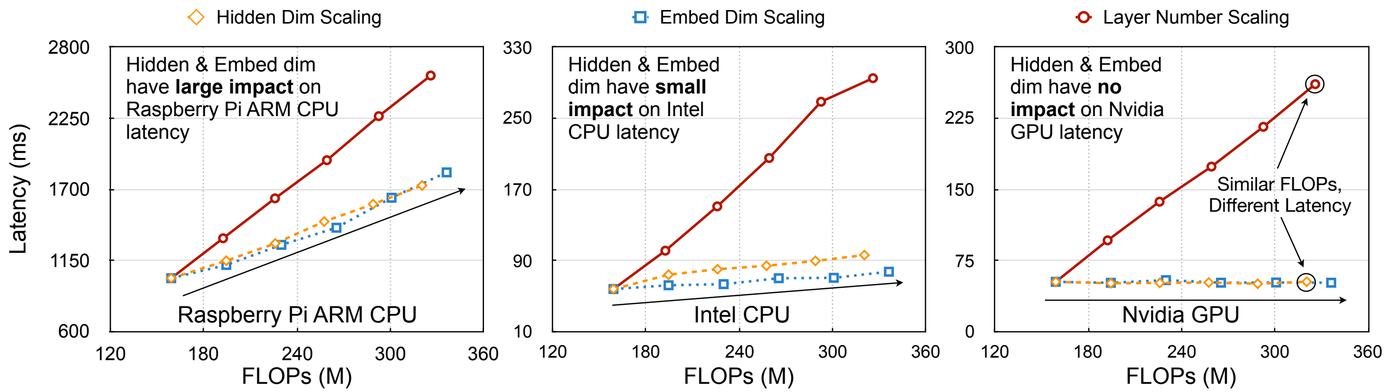
H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han. Once-for-All: Train One Network and Specialize it for Efficient Deployment. ICLR 2020.

HAT: Hardware Aware Transformers

- **Key idea:** Once-for-all, adapted to Transformer architecture.
  - **Heterogeneous Transformer layers:** Different number of self-attention heads, feed-forward widths.
  - **Arbitrary encoder-decoder attention:** Each decoder layer may attend to multiple encoder layers.







H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, S. Han. HAT: Hardware-Aware Transformers for Efficient Natural Language Processing. ACL 2020.