# LYU2002

Study Neural Architecture Search

[Neural Architecture Search on BERT for Network Compression]

ESTR4999 2020/21 Term 2 Oral Presentation

Yau Chung Yiu, Oscar

1155109029

30 min present, 15 min Q&A

# Background

- Natural Language Processing uses deep neural networks that are
  - Computationally expensive, slow
    - BERT 12-layer: 22.2 Billion FLOPS, for classification of sequence at most 128 tokens long
    - Processing 144 sentences / second on GPU
    - Processing 4 sentences / second on CPU
    - BERT 12-layer: 110 Million parameters -> requires 419 MB to load the model into memory for inference -> limits parallelization
- We achieve network compression by Neural Architecture Search

# Related Study – Lottery Ticket Hypothesis

- Consider training a larger neural network as buying a lot of lottery ticket
  - Larger network initialized for training -> Higher chance of winning
  - After training only the winning ticket (subnetwork) is responsible for prediction
  - The weight initialization is crucial for a subnetwork to be a winning ticket
    - Reinitializing the winning ticket will lead to a poor model

[1] J. Frankle, M. Carbin: *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks* (2018)

# Problem Setting – Pruning NN: BERT

- Search for optimal sub-architecture from original network.

- Define
  - $\alpha$ – architecture parameters
  - $r$ – sub-architecture FLOPS / original architecture FLOPS
  - $E$ – training error on data samples
  - $w$ – neural network parameters

- Objective:
  - Finding sub-architecture $\alpha$ of small $r$ that achieve small training error $E$
  - $\min_{\alpha,w} E$ such that $\mathrm{Ratio}(\alpha) = r$
  - We approach solving this problem by $\min_{w} \{\min_{\alpha,w} E\}$

# Problem Structure

- Objective:
  - Finding sub−architecture $\alpha$ of small $r$ that achieve small training error $E$
  - $\min_{\alpha,w} E$ such that $\mathrm{Ratio}(\alpha) = r$
  - We approach solving this problem by $\min_{w} \{\min_{\alpha,w} E\}$

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\text{s.t.} \quad w^*(\alpha) = \mathrm{argmin}_w \ \mathcal{L}_{train}(w, \alpha)$$

*Fig 1. Bilevel architecture search formulation. [1]*

- $\alpha$ can be understood as mask variable $\in \{0,1\}^n$
  - $n$ – number of pruning groups (e.g. pruning an attention-head)
  - $n$ depends on the search space
  - e.g. for TinyBERT-4L, $n = 3072\times4 + 12\times4 = 12336$
- $2^n$ is large, it cannot be solved by exhaustively searching
- This optimization problem is non-convex, because
  - combinatorial nature
  - bilevel optimization

[2] H. Liu, K. Simonyan, Y. Yang: *Darts: Differentiable architecture search* (2019)

# Methodology – Existing Baseline Method

- Magnitude Pruning
  - Remove weights connections of small magnitude
  - Intuitively small magnitude weights have small impact to the output

| Dataset | MNLI | QQP | STS-B | WNLI | QNLI | MRPC | RTE | SST-2 | CoLA | SQuAD | MLM |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sparsity | 70% | 90% | 50% | 90% | 70% | 50% | 60% | 60% | 50% | 40% | 70% |
| Full $\text{BERT}_{\text{BASE}}$ | $82.4 \pm 0.5$ | $90.2 \pm 0.5$ | $88.4 \pm 0.3$ | $54.9 \pm 1.2$ | $89.1 \pm 1.0$ | $85.2 \pm 0.1$ | $66.2 \pm 3.6$ | $92.1 \pm 0.1$ | $54.5 \pm 0.4$ | $88.1 \pm 0.6$ | $63.5 \pm 0.1$ |
| $f(x, m_{\text{IMP}} \odot \theta_0)$ | $82.6 \pm 0.2$ | $90.0 \pm 0.2$ | $88.2 \pm 0.2$ | $54.9 \pm 1.2$ | $88.9 \pm 0.4$ | $84.9 \pm 0.4$ | $66.0 \pm 2.4$ | $91.9 \pm 0.5$ | $53.8 \pm 0.9$ | $87.7 \pm 0.5$ | $63.2 \pm 0.3$ |
| $f(x, m_{\text{RP}} \odot \theta_0)$ | 67.5 | 76.3 | 21.0 | 53.5 | 61.9 | 69.6 | 56.0 | 83.1 | 9.6 | 31.8 | 32.3 |
| $f(x, m_{\text{IMP}} \odot \theta_0')$ | 61.0 | 77.0 | 9.2 | 53.5 | 60.5 | 68.4 | 54.5 | 80.2 | 0.0 | 18.6 | 14.4 |
| $f(x, m_{\text{IMP}} \odot \theta_0'')$ | 70.1 | 79.2 | 19.6 | 53.3 | 62.0 | 69.6 | 52.7 | 82.6 | 4.0 | 24.2 | 42.3 |

*Table from [3], results of iterative magnitude pruning*

[3] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, M. Carbin*: The Lottery Ticket Hypothesis for Pre-trained BERT Networks* (2020)

# Methodology – Existing Baseline Method

- Structured Pruning
  - Masking the variables according to the structure of the original network
  - Measure importance of weight by sensitivity to mask variables
  - Performance aware pruning – pruning continues as long as validation performance is over 90% of the full model
  - Algorithm is unstable [5], all tickets are winning!
    - Compared between “Good” and “Bad” subnetworks.
    - Suggests that all weights in BERT contains useful information, enough for prediction

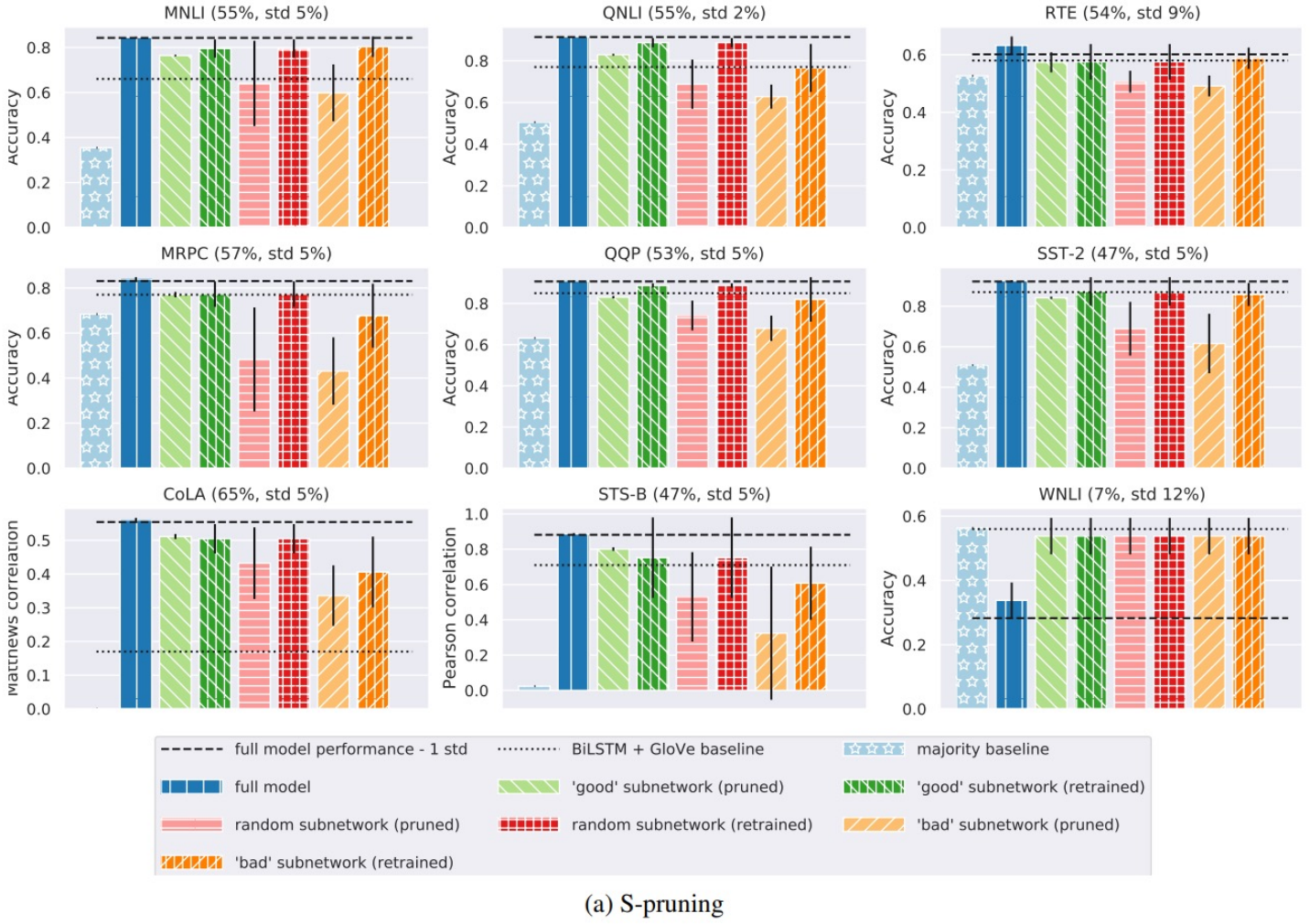
$$I_h^{(h,l)} = E_{x \sim X} \left| \frac{\partial \mathcal{L}(x)}{\partial \xi^{(h,l)}} \right|$$

*Equation from [4]. Output sensitivity w.r.t. the mask variables.*

[4] P. Michel, O. Levy, G. Neubig: *Are Sixteen Heads Really Better than One?* (2019)
[5] S. Prasanna, A. Rogers, A. Rumshisky: *When BERT Plays the Lottery, All Tickets Are Winning* (2020)

# Methodology – Existing Baseline Method

- Structured Pruning
  - Bad subnetworks are also winning




(a) S-pruning

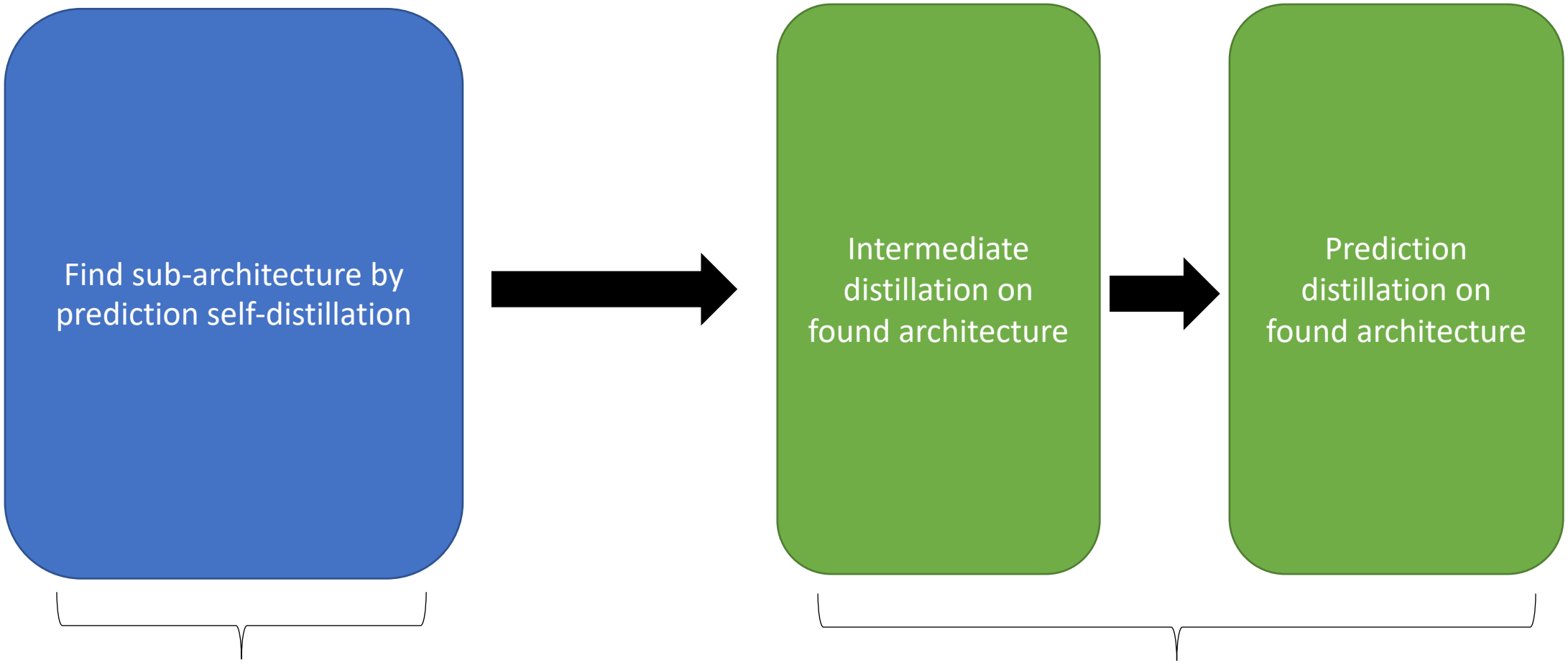
*Fig. from [5], performance on GLUE tasks by structured pruning.*

[5] S. Prasanna, A. Rogers, A. Rumshisky: *When BERT Plays the Lottery, All Tickets Are Winning* (2020)

# Methodology – Introduction

- Our NAS approach:
  - Search for the optimal architecture by learning
  - Relaxation on mask variable by $sigmoid(\alpha), \alpha \in \mathbb{R}^n$
  - Gradient descent to the architecture parameters $\alpha$
- Combining
  - Structured Pruning
    - Prune Multi-heads by blocks
    - Prune Feed-forward intermediate by dimensions
  - Distillation
    - Self-supervised distillation to recover original performance

# Methodology – Procedure



# Methodology – Search Method

- Loss functions compose of
  1. Prediction self-distillation loss $(\mathcal{L}_\alpha, \mathcal{L}_w)$
  2. FLOPS loss $(\mathcal{L}_\alpha)$

$$\mathcal{L}_{arch} = \underbrace{\mathrm{MSE}(o_{student}, o_{teacher})}_{\textit{prediction distillation loss}} + \underbrace{\lambda_{cost}\mathcal{L}_{cost}}_{\textit{computation cost loss}} \quad (3)$$

*Positive FLOPS loss if the current architecture size > required size*

$$\mathcal{L}_{cost} = \begin{cases} \log\left(\mathbb{E}_{cost}(\mathbb{A})\right) & \text{when } F_{cost}(\mathbb{A}) > (1+t) \times R \\ 0 & \text{when } (1-t) \times R < F_{cost}(\mathbb{A}) < (1+t) \times R \\ -\log\left(\mathbb{E}_{cost}(\mathbb{A})\right) & \text{when } F_{cost}(\mathbb{A}) < (1+t) \times R \end{cases} \quad (4)$$

*Negative FLOPS loss if the current architecture size < required size*

*Fig 2. Formulation of loss functions, from report Section 5.2.*

# Methodology – Search Method Tricks

- Architecture target ratio control (FLOPS loss)
  - Target ratio $R$ moves linearly throughout the searching procedure
  - e.g. $R$ moves from 1 to 0.3
  - Sharp drop of architecture size damages the learning outcome

- Update control
  - Only top 10% of the large magnitude gradients are backpropagated to the $\alpha$ for update.

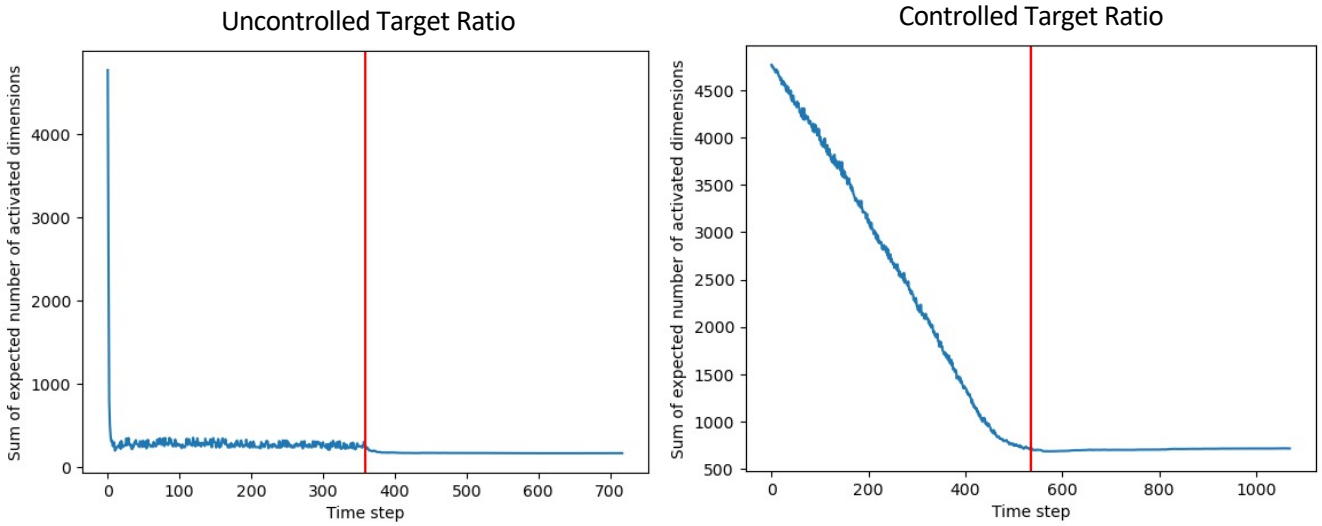



*Fig 3. Comparison between controlled arch. target ratio and uncontrolled learning of alpha, from report Section 7.4.*

=> Simulate iterative pruning

# Methodology – Search Space

- Multi-head attention
  - Different heads learn different kind of similarity
  - When trained on down-stream task many heads are redundant [4]

- Feed-forward intermediate dimensions
  - Major contribution to FLOPS (61%)
  - Intermediate dimension expanded to learn high dimensional features

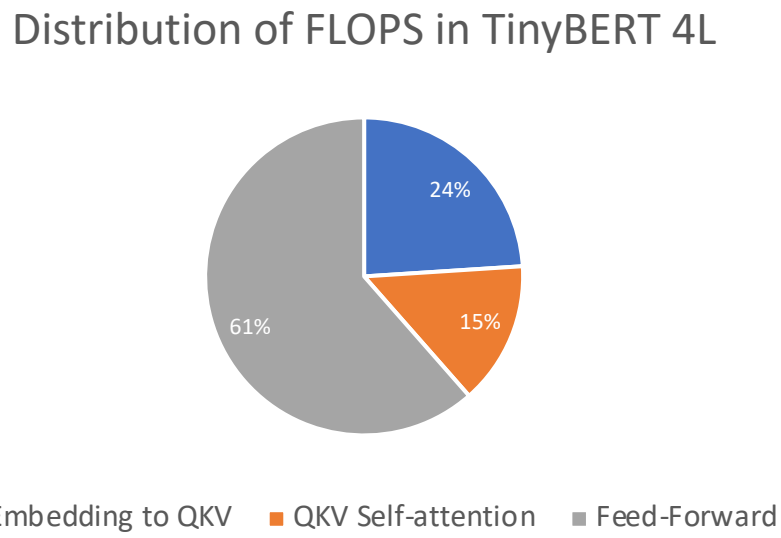
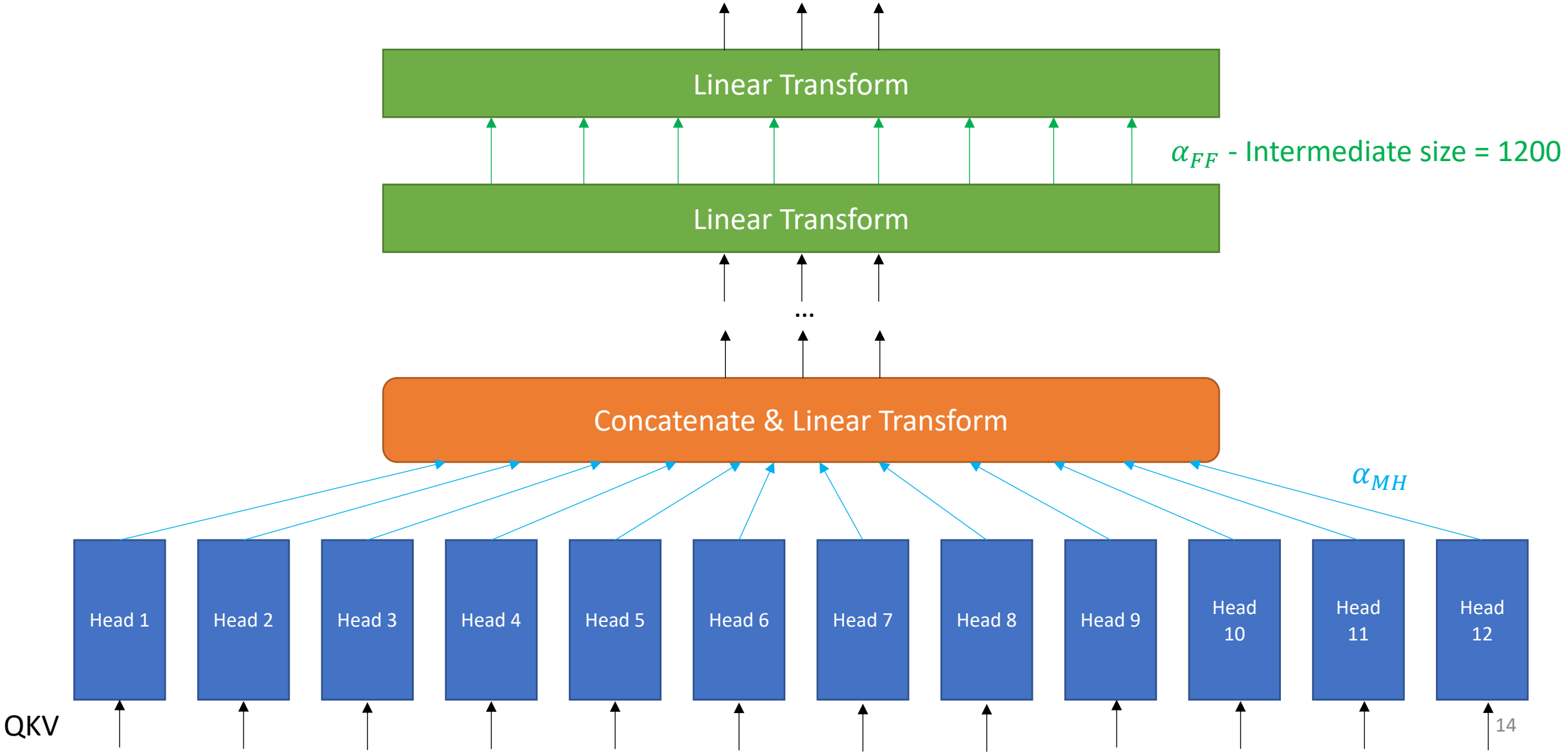



*Fig 4. Distribution of FLOPS in TinyBERT 4L, from report Section 7.2.*

[4] P. Michel, O. Levy, G. Neubig: Are Sixteen Heads Really Better than One? (2019)

# Methodology – Search Space Illustration



# Main Result

*All tables from report Section 6.3.2*

*Metrics reported*
*Mcc: CoLA*
*F1: QQP, MRPC*
*Pearson-Spearman correlations: STS-B*
*Accuracy: MNLI, QNLI, SST-2, RTE*

| Models | FLOPS (B) | Speedup | MNLI (-m/-mm) | QQP | QNLI | SST-2 | CoLA | STS-B (Pear/Spea) | MRPC | RTE |
|---|---|---|---|---|---|---|---|---|---|---|
| TinyBERT-4L | 1.239 | 1.0x | 82.5/81.8 | 71.3 | 87.7 | 92.6 | 44.1 | -/80.4 | 86.4 | 66.6 |
| 30% TinyBERT-4L | [0.375, 0.403] | [3.0x, 3.3x] | 80.8/80.4 | 70.9 | 84.4 | 91.8 | 40.7 | 79.9/78.6 | 85.4 | 61.4 |
| Percentage Change in Accuracy | / | / | -2.06%/-1.71% | -0.56% | -3.76% | -0.86% | -7.71% | -2.24% | -1.16% | -7.81% |

*Table 1. GLUE Test Result on TinyBERT-4L pruning*

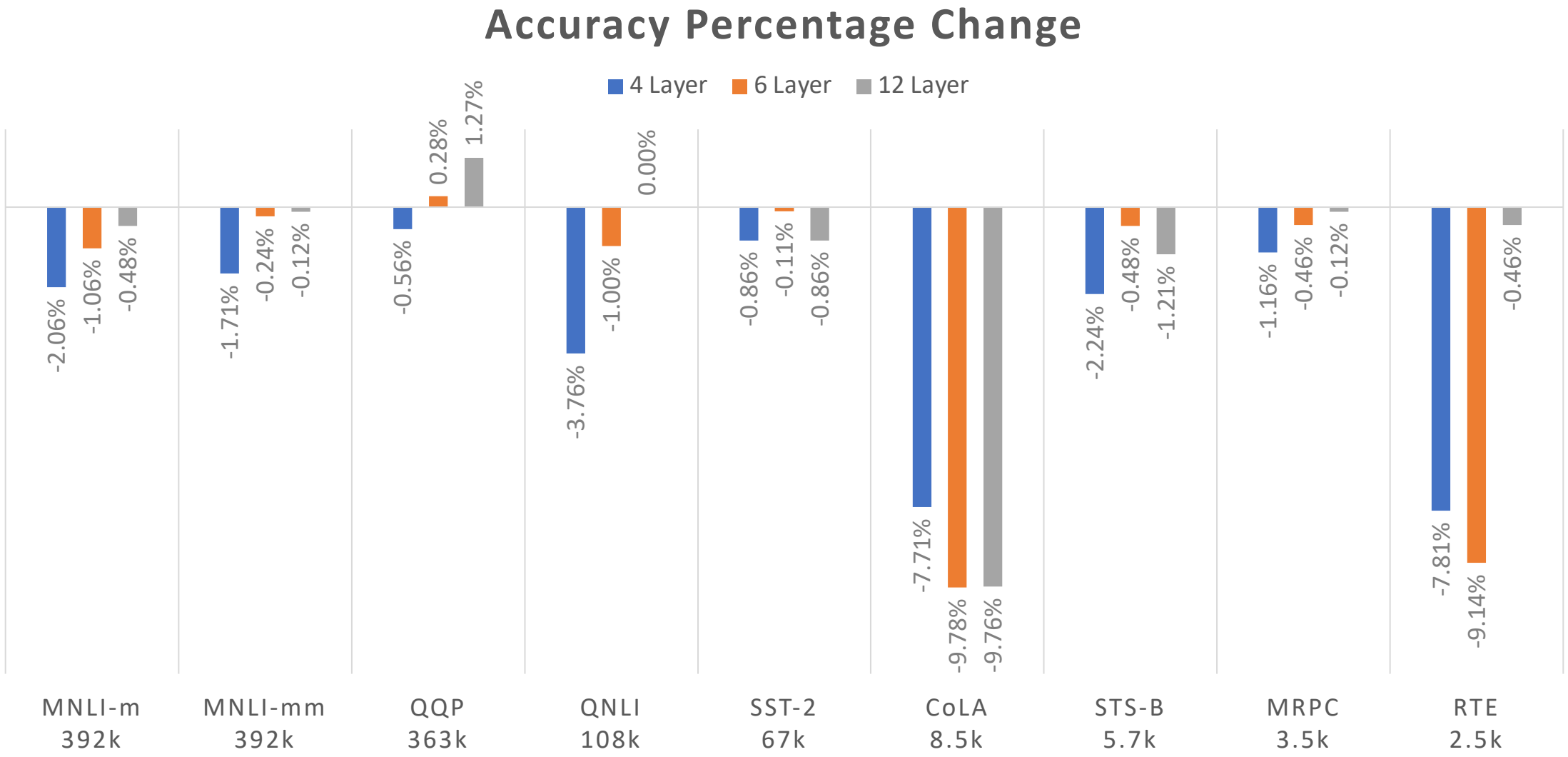
| Models | FLOPS (B) | Speedup | MNLI (-m/-mm) | QQP | QNLI | SST-2 | CoLA | STS-B (Pear/Spea) | MRPC | RTE |
|---|---|---|---|---|---|---|---|---|---|---|
| TinyBERT-6L | 11.100 | 1.0x | 84.6/83.2 | 71.6 | 90.4 | 93.1 | 51.1 | -/83.7 | 87.3 | 70.0 |
| 30% TinyBERT-6L | [3.348, 3.590] | [3.1x, 3.3x] | 83.7/83.0 | 71.8 | 89.5 | 93.0 | 46.1 | 84.2/83.3 | 86.9 | 63.6 |
| Percentage Change in Accuracy | / | / | -1.06%/-0.24% | +0.28% | -1.00% | -0.11% | -9.78% | -0.48% | -0.46% | -9.14% |

*Table 2. GLUE Test Result on TinyBERT-6L pruning*

| Models | FLOPS (B) | Speedup | MNLI (-m/-mm) | QQP | QNLI | SST-2 | CoLA | STS-B (Pear/Spea) | MRPC | RTE |
|---|---|---|---|---|---|---|---|---|---|---|
| Reproduced Bertbase-12L | 22.199 | 1.0x | 84.0/83.2 | 70.7 | 91.1 | 92.7 | 55.3 | 84.2/82.5 | 86.6 | 65.5 |
| 30% Bertbase-12L | [6.697, 9.471] | [2.3x, 3.3x] | 83.6/83.1 | 71.6 | 91.1 | 91.9 | 49.9 | 82.9/81.5 | 86.5 | 65.2 |
| Percentage Change in Accuracy | / | / | -0.48%/-0.12% | 1.27% | 0.00% | -0.86% | -9.76% | -1.54%/-1.21% | -0.12% | -0.46% |

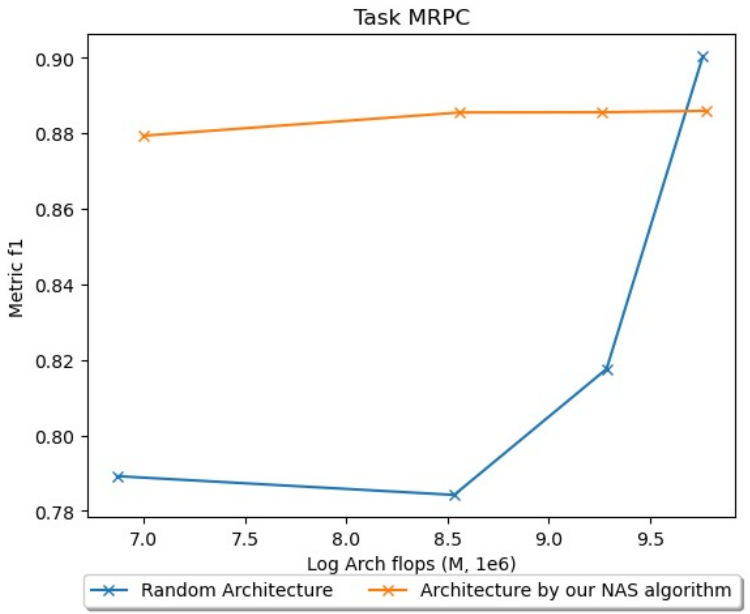
*Table 3. GLUE Test Result on Bertbase-12L pruning*

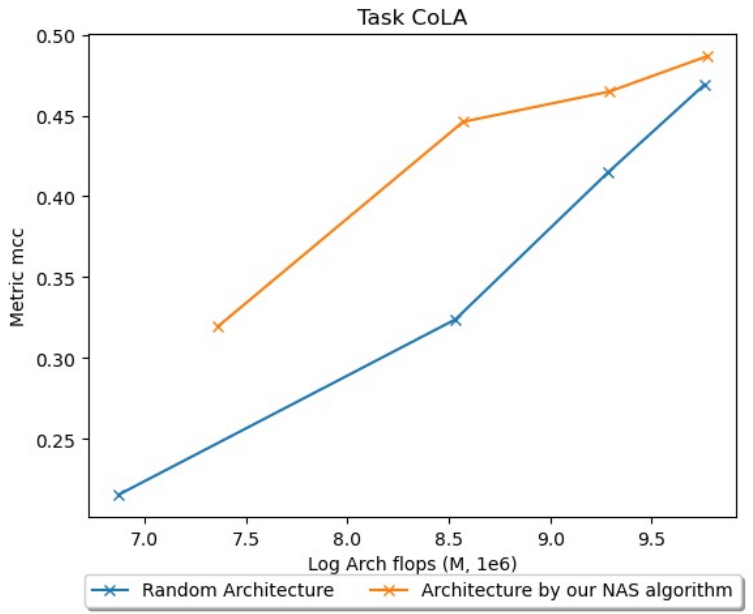
# Main Result


Accuracy Percentage Change
4 Layer
6 Layer
12 Layer
-2.06%
-1.06%
-0.48%
-1.71%
-0.24%
-0.12%
-0.56%
0.28%
1.27%
-3.76%
-1.00%
0.00%
-0.86%
-0.11%
-0.86%
-7.71%
-9.78%
-9.76%
-2.24%
-0.48%
-1.21%
-1.16%
-0.46%
-0.12%
-7.81%
-9.14%
-0.46%
MNLI-m 392k
MNLI-mm 392k
QQP 363k
QNLI 108k
SST-2 67k
CoLA 8.5k
STS-B 5.7k
MRPC 3.5k
RTE 2.5k

# Main Result – Comparison with Simple Baseline: Random Architecture

- We have shown that our algorithm can find architecture that performs better than a random architecture of similar FLOPS.



# Main Result – Comparison with Existing Baseline: Google BERT

- We have shown significant prediction accuracy improvement compared to existing BERT model with similar FLOPS.

| Models | FLOPS (B) | MNLI (-m/-mm) | QQP | QNLI | SST-2 | CoLA | STS-B (Pear/Spea) | MRPC | RTE |
|---|---|---|---|---|---|---|---|---|---|
| BERT-Mini [25] | 0.873 | 74.8/74.3 | 66.4 | 84.1 | 85.9 | 0.0 | 75.4/73.3 | 81.1 | 57.9 |
| 30% TinyBERT-4L | [0.375, 0.403] | 80.8/80.4 | 70.9 | 84.4 | 91.8 | 40.7 | 79.9/78.6 | 85.4 | 61.4 |
| Percentage Change in Accuracy | / | +8.02%/+8.21% | +6.78% | +0.36% | +6.87% | +inf | +5.97% | +5.30% | +6.04% |

*Table 4. Comparison between BERT-Mini by Google and pruned TinyBERT-4L, from report Section 6.3.2.*

# Main Result – Comparison with Existing Baseline: Magnitude Pruning

| Dataset | MNLI | QQP | STS-B | WNLI | QNLI | MRPC | RTE | SST-2 | CoLA | SQuAD | MLM |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sparsity | 70% | 90% | 50% | 90% | 70% | 50% | 60% | 60% | 50% | 40% | 70% |
| Full $\text{BERT}_{\text{BASE}}$ | $82.4 \pm 0.5$ | $90.2 \pm 0.5$ | $88.4 \pm 0.3$ | $54.9 \pm 1.2$ | $89.1 \pm 1.0$ | $85.2 \pm 0.1$ | $66.2 \pm 3.6$ | $92.1 \pm 0.1$ | $54.5 \pm 0.4$ | $88.1 \pm 0.6$ | $63.5 \pm 0.1$ |
| $f(x, m_{\text{IMP}} \odot \theta_0)$ | $82.6 \pm 0.2$ | $90.0 \pm 0.2$ | $88.2 \pm 0.2$ | $54.9 \pm 1.2$ | $88.9 \pm 0.4$ | $84.9 \pm 0.4$ | $66.0 \pm 2.4$ | $91.9 \pm 0.5$ | $53.8 \pm 0.9$ | $87.7 \pm 0.5$ | $63.2 \pm 0.3$ |
| $f(x, m_{\text{RP}} \odot \theta_0)$ | 67.5 | 76.3 | 21.0 | 53.5 | 61.9 | 69.6 | 56.0 | 83.1 | 9.6 | 31.8 | 32.3 |
| $f(x, m_{\text{IMP}} \odot \theta_0')$ | 61.0 | 77.0 | 9.2 | 53.5 | 60.5 | 68.4 | 54.5 | 80.2 | 0.0 | 18.6 | 14.4 |
| $f(x, m_{\text{IMP}} \odot \theta_0'')$ | 70.1 | 79.2 | 19.6 | 53.3 | 62.0 | 69.6 | 52.7 | 82.6 | 4.0 | 24.2 | 42.3 |

*Table from [3], results of iterative magnitude pruning on dev set*

| Dataset | MNLI (-m) | QQP | QNLI | SST-2 | CoLA | STS-B (Pearson) | MRPC | RTE |
|---|---|---|---|---|---|---|---|---|
| Ratio | 28.3% | 28.3% | 39.9% | 28.6% | 28.9% | 31.5% | 29.2% | 28.2% |
| 30% Bertbase-12L | 84.2 | 90.3 | 91.7 | 92.3 | 58.3 | 88.8 | 85.3 | 69.7 |
| Ratio | 30% | 10% | 30% | 40% | 50% | 50% | 50% | 40% |
| IMP [3] | 82.6 | 90.0 | 88.9 | 91.9 | 53.8 | 88.2 | 84.9 | 66.0 |
| Difference | +1.6 | +0.3 | +2.8 | +0.4 | +4.5 | +0.6 | +0.4 | +3.7 |

*Table: comparison between IMP [3] and our results, on dev set.*

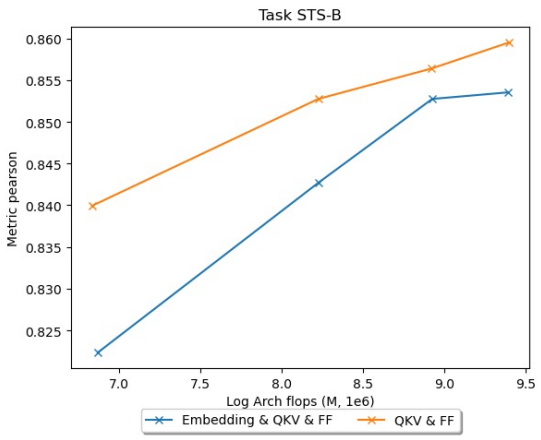
Our method is stronger because we do distillation and data augmentation during fine-tuning.

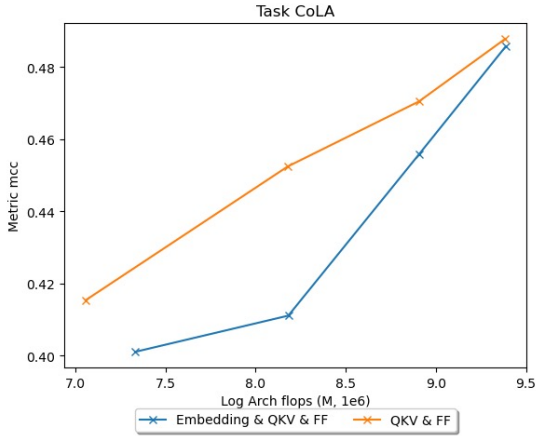


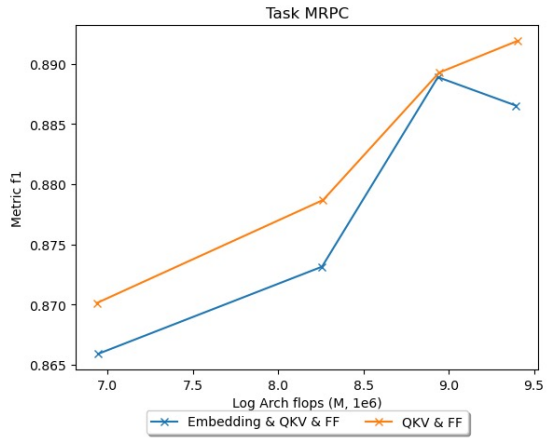
[3] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, M. Carbin: *The Lottery Ticket Hypothesis for Pre-trained BERT Networks* (2020)

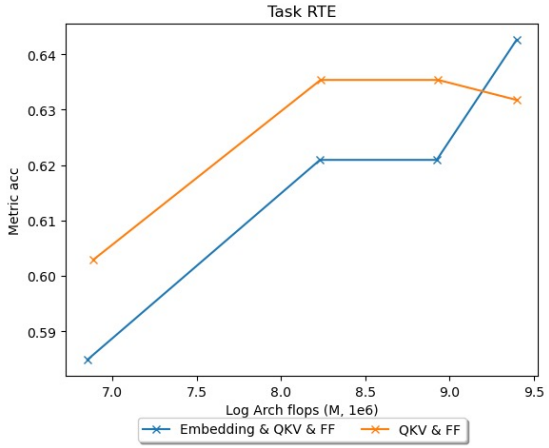
# Discussion – Input Embedding Pruning

- In semester 1 we proposed pruning input embedding.
- By experiment results we see that pruning input embedding damages network prediction accuracy by a lot.
- Because input embedding carries either
  - Sentence token information (layer 1)
  - Previous layer output (layer 2 – last layer)



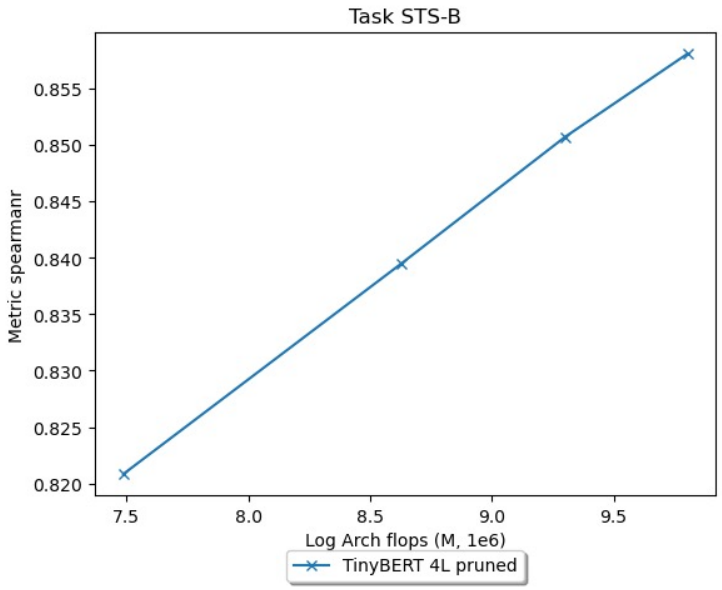
# Discussion – QKV pruning VS Multi-head pruning

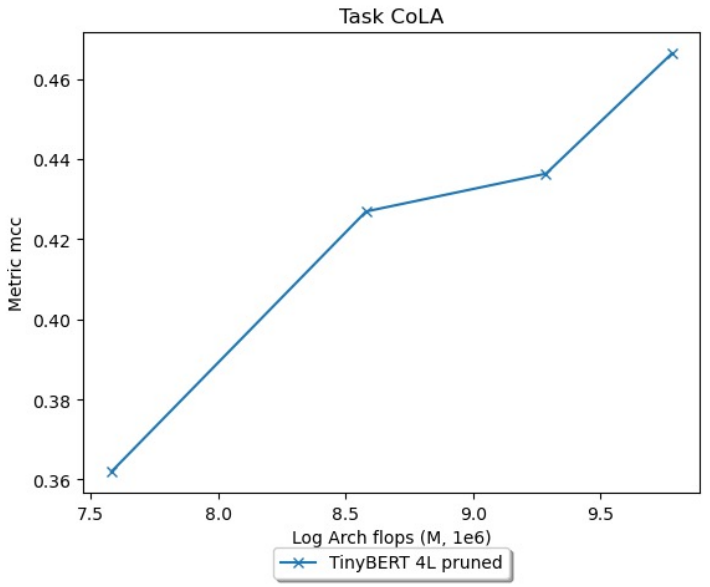
- Multi-head pruning is a restricted version of QKV pruning
  - An attention head = A group of QKV dimensions
  - Structured Pruning

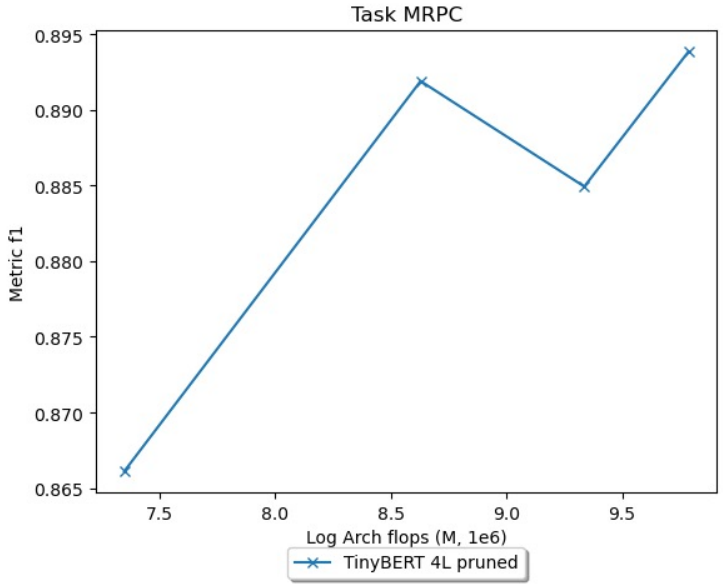
- Structured Pruning outputs an architecture that is computationally faster

# Discussion – Architecture Size Ratio

- By experiment we observed that below the ratio 0.3, the prediction performance damage is significant.

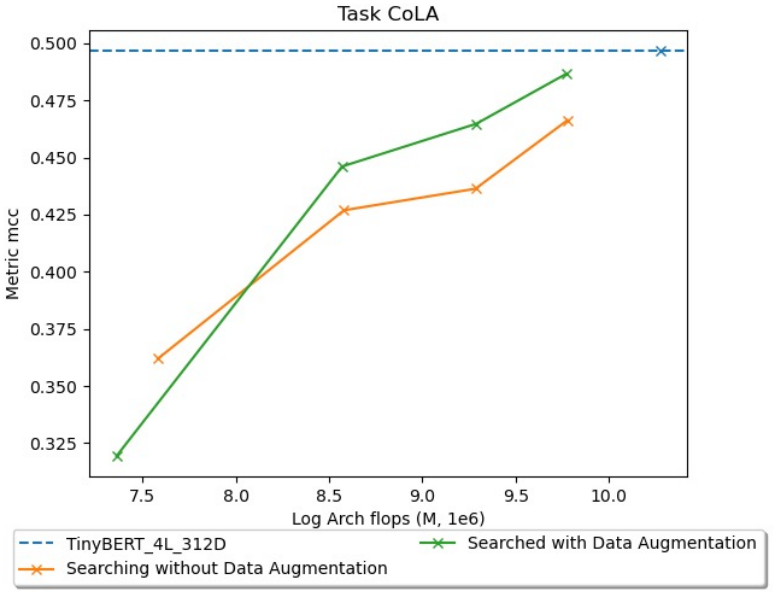


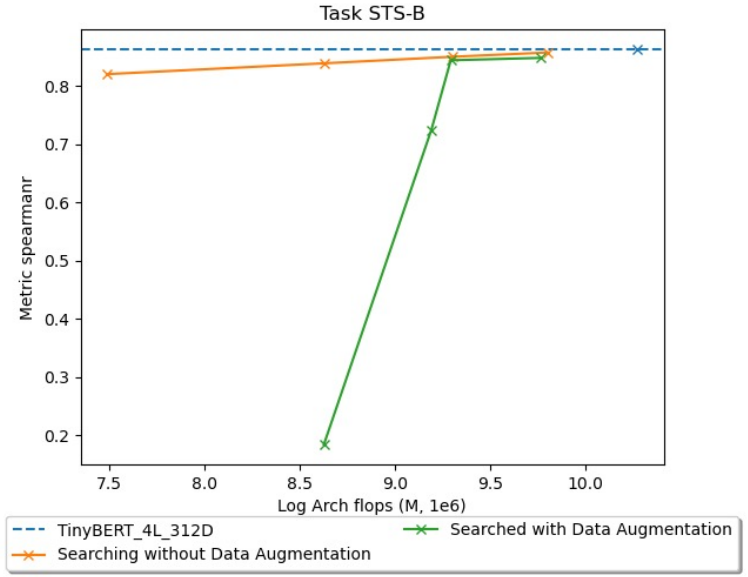



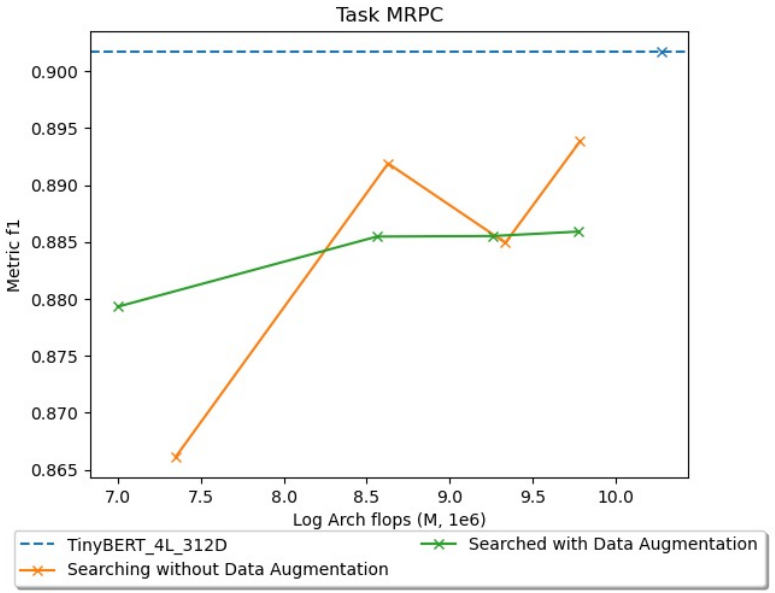



# Discussion – Effect of Data Augmentation

- For fine-tuning data augmentation always improves prediction accuracy (more generalized model)
- For searching for architecture, data augmentation is
  - harmful for data sufficient task, and
  - useful for data insufficient task (CoLA, RTE)



# Discussion – Pros and Cons of our algorithm

- Our searching algorithm can search for an architecture of any size (architecture size ratio $R$ is a parameter)
- Our method considers all the possible search regions of BERT and assuming equal importance over all regions
- Our method is applicable to other variants of BERT
  - TinyBERT 4L / 6L
  - Bert-base 12L
- The algorithm computationally expensive
  - 10 epochs for architecture search
  - 20 epochs for two-stage distillation fine-tuning
  - 12 hours – 2 days on CSE GPU
- Instability

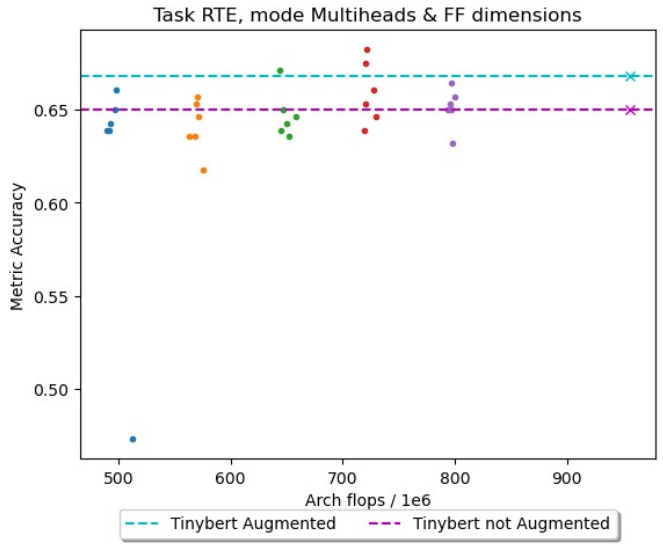



*Fig. Random seeding*

# Conclusion

- Larger neural networks have higher redundancy (tolerant to pruning)
  - Performance drop is larger on small neural network, when pruning on the same ratio
  - For production deep neural networks are highly accelerable
- The quality of model depends on the quality of learning data
  - Difficult tasks: RTE, CoLA (difficult and insufficient data)

# Reflection

- We should compare among baselines fairly:
  - Compare at the similar searched architecture size
  - When comparing we need to care about whether
    - Is the searched architecture better? (fix fine-tuning method to compare)
    - Is the fine-tuning method better? (fix architecture to compare -> more difficult to handle)

# Future Direction

- Fixing network parameter during search
  - Less interference with the architecture parameter $\alpha$
- Experiment on knowledge transfer (distillation from stronger BERT)
- Apply performance aware searching
  - Searching continues when the pruned model recovered its prediction performance
  - Searching terminates when the pruned model cannot recover
- Analysis the attention pattern
  - Between the pruned network and the original network

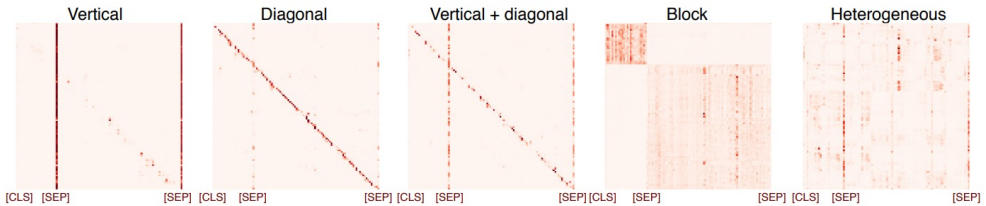



Fig. from O. Kovaleva, A. Romanov, A. Rogers, A. Rumshisky: *Revealing the Dark Secrets of BERT* (2019)