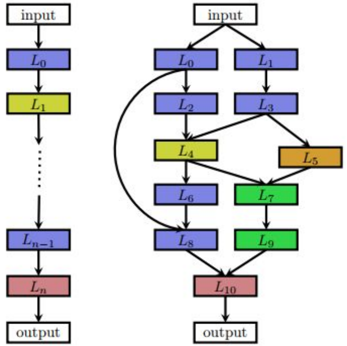
Neural Architecture Search

Yu Cao

NAS search space

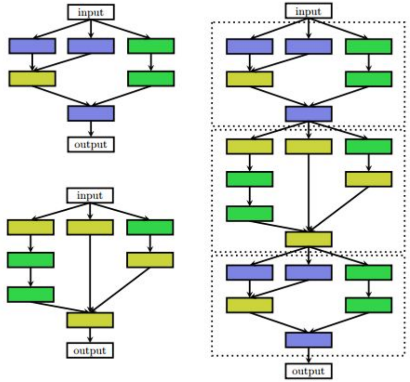
1. **Architecture space**

   Every layer (even an activation) in a model is involved



2. **Cell space**

   Multiple layers compose a single cell and cells are involved as search space (smaller size)



Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search: A Survey." Journal of Machine Learning Research 20 (2019): 1-21.

NAS Search Strategy

Traditionally, the search procedure is not differentiable

1. **Random Search**: random select a series of models and test their performance
2. **Evolutionary method**: shrink the search space step by step via filtering low-performance models using fewer training steps.
3. **Bayesian Optimization**: Assuming the performance of a model follows the Gaussian distribution, maximize the acquisition function based on the performance observation of models.
4. **Reinforcement Learning**: regard a the generation of a model as an action of the agent and the reward is the performance of current generation.
5. **Gradient-based method**: transfer the procedure as a differentiable operation using soft weights to combine different candidate ops for a node. (**Most popular approach now**)

NAS Performance Estimation Strategy (Speed up)

1. **Lower Fidelity Estimates**: training using fewer epochs, subset of the data, downscaled models, etc.
2. **Learning Curve Extrapolation**: training stops when the performance can be extrapolated after few epochs.
3. **Weight Inheritance**: model can be trained from a parent model.
4. **One-shot model**: only the one-shot model is trained while its weight is shared across different architectures.

DARTS: Differentiable Architecture Search

Hanxiao Liu (CMU), Karen Simonyan (DeepMind), Yiming Yang (CMU)

ICLR 2019

Contribution

1. It transforms the NAS problem into differentiable one using soft weighting on the possible operations of nodes in a complex topologies, which can be used on both convolutional and recurrent networks.

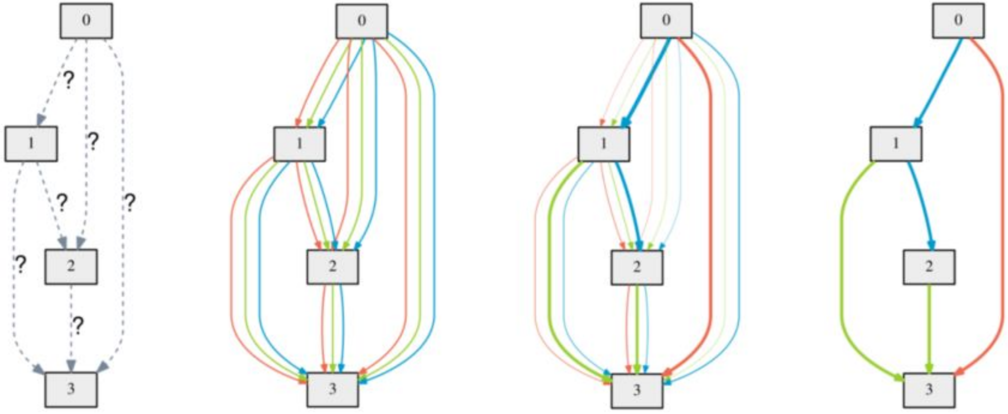
2. Such method can also achieve efficiency improvement, as it uses gradient-based optimization to find the best architecture among all possible ones jointly instead of one by one.

Search Space

It is a cell-level search, in which each cell is a **directed acyclic graph (DAG)**, in which each node x^i is a representation and edge (i, j) is the operation $o^{i,j}$ on x^i.

The final representation of node j is the combination of results from all input edges

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$$



(a) (b) (c) (d)

Optimization Procedure

Given a set of operation $\mathcal{O}$, the output of an operation is weighted using softmax on a weight vector $\alpha^{(i,j)}$ in dimension $|\mathcal{O}|$.

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

Thus the goal is jointly learn the architecture α and layer weight w within all mixed operations, given the training loss $\mathcal{L}_{train}$ and validation loss $\mathcal{L}_{val}$

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

Gradient Approximation

Directly optimize the objective is too resource-consuming with complexity $O(|\alpha||w|)$

An approximation is

$$\begin{aligned}&\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha)\\ \approx&\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)\end{aligned}$$

Applying chain rule yields $\nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$

Where $w' = w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha)$ is a one-step forward model.

The second item is approximated using finite difference approximation

$$\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon} \qquad w^\pm = w \pm \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$$

DARTS Algorithm

The final optimization on α turns to be following, with complexity $O(|\alpha| + |w|)$

$$\nabla_\alpha L_{val}(w', \alpha) - \xi \frac{\nabla_\alpha L_{train}(w + \epsilon \nabla_{w'} L_{val}(w', \alpha), \alpha) - \nabla_\alpha L_{train}(w - \epsilon \nabla_{w'} L_{val}(w', \alpha), \alpha)}{2\epsilon}$$

The algorithm will optimize α and w iteratively, in which the optimization of α is described as above

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)

while *not converged* **do**

1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$
   ($\xi = 0$ if using first-order approximation)
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned α.

max value in α for each node indicates the selected operation

Experiments

DARTS is tested on CIFAR-10 (conv net) and PTB (RNN)

CIFAR-10

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	#ops	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	–	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	2000	13	RL
NASNet-A + cutout (Zoph et al., 2018)†	2.83	3.1	2000	13	RL
BlockQNN (Zhong et al., 2018)	3.54	39.8	96	8	RL
AmoebaNet-A (Real et al., 2018)	3.34 ± 0.06	3.2	3150	19	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	19	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	19	evolution
Hierarchical evolution (Liu et al., 2018b)	3.75 ± 0.12	15.7	300	6	evolution
PNAS (Liu et al., 2018a)	3.41 ± 0.09	3.2	225	8	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	6	RL
ENAS + cutout (Pham et al., 2018b)*	2.91	4.2	4	6	RL
Random search baseline‡ + cutout	3.29 ± 0.15	3.2	4	7	random
DARTS (first order) + cutout	3.00 ± 0.14	3.3	1.5	7	gradient-based
DARTS (second order) + cutout	2.76 ± 0.09	3.3	4	7	gradient-based

Experiments

DARTS is tested on CIFAR-10 (conv net) and PTB (RNN)

PTB

Architecture	Perplexity valid	Perplexity test	Params (M)	Search Cost (GPU days)	#ops	Search Method
Variational RHN (Zilly et al., 2016)	67.9	65.4	23	–	–	manual
LSTM (Merity et al., 2018)	60.7	58.8	24	–	–	manual
LSTM + skip connections (Melis et al., 2018)	60.9	58.3	24	–	–	manual
LSTM + 15 softmax experts (Yang et al., 2018)	58.1	56.0	22	–	–	manual
NAS (Zoph & Le, 2017)	–	64.0	25	1e4 CPU days	4	RL
ENAS (Pham et al., 2018b)*	68.3	63.1	24	0.5	4	RL
ENAS (Pham et al., 2018b)†	60.8	58.6	24	0.5	4	RL
Random search baseline‡	61.8	59.4	23	2	4	random
DARTS (first order)	60.2	57.6	23	0.5	4	gradient-based
DARTS (second order)	58.1	55.7	23	1	4	gradient-based

Conclusion

DARTS significantly reduces the resource consumption of NAS while provides comparable performance compared to RL or Evolution approaches, which makes it followed by many related works (690 citations in past 2 years).

Gradient-based NAS has also become the main trend, 80% of last related papers use gradient-based optimization.

NAS in NLP

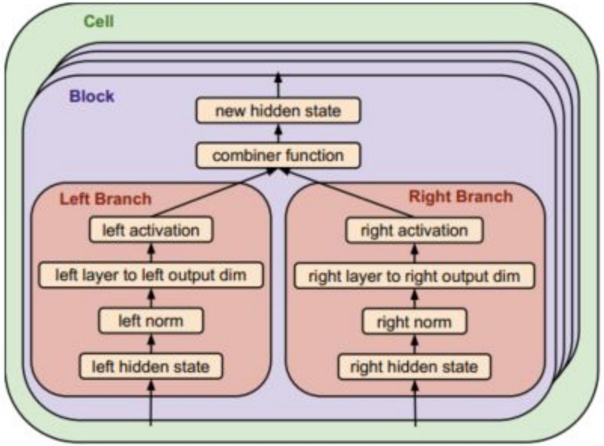
1. The Evolved Transformer (ICML 2019, Google, Quoc V.Le)
2. Continual and Multi-Task Architecture Search (ACL 2019, UNC Chapel Hill)
3. Improved Differentiable Architecture Search for Language Modeling and Named Entity Recognition (EMNLP 2019 (short), NEU China)
4. Learning Architectures from an Extended Search Space for Language Modeling (ACL 2020, NEU China)
5. Improving Transformer Models by Reordering their Sublayers (ACL 2020, UW and Allen AI)

1. The Evolved Transformer

So, David, Quoc Le, and Chen Liang. "The Evolved Transformer." *International Conference on Machine Learning*. 2019.

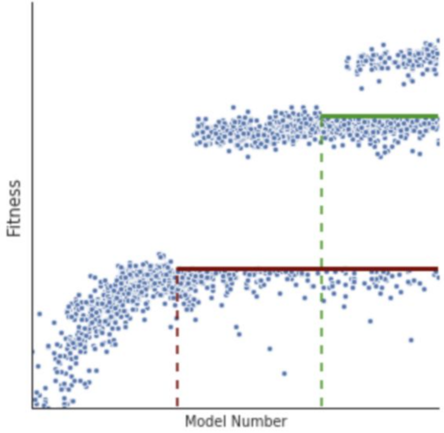
This paper utilizes **evolution algorithm** to search **a better architecture of Transformer** for MT task.

The search space is 14 blocks (6 for encoder and 8 for decoder), each block contains left and right branch (input, normalization, layer, output dimension and activation in each of them) and combination function



Evolution algorithm

1. Random sampling architecture as the initial child models, build a set of small training step number set <*s, s1, s2,...*>
2. Train each model with a small step *s* and evaluate their fitness (performance)
3. Set the hurdle as the mean fitness of all models. Models with lower fitness than hurdle will be fitered.
4. Rest models will be trained for further step *si* and repeat 2,3,4 until all step numbers in a set are used or no model left.



y-axis is the fitness while the x-axis is the order of the generating of candidate models. Solid lines are hurdles.

Experiment and results

It uses WMT datasets, initial model number m=5000, and step numbers set are <60k, 60k, 120k>, ~50,000 TPU hour (~ 1,000,000 GPU hour) to find 20 best architecture and find the best one with full training.

TASK	SIZE	TRAN PARAMS	ET PARAMS	TRAN PERP	ET PERP	TRAN BLEU	ET BLEU
WMT'14 EN-DE	BASE	61.1M	64.1M	4.24 ± 0.03	$\mathbf{4.03} \pm 0.02$	28.2 ± 0.2	$\mathbf{28.4} \pm 0.2$
WMT'14 EN-DE	BIG	210.4M	221.7M	3.87 ± 0.02	$\mathbf{3.77} \pm 0.02$	29.1 ± 0.1	$\mathbf{29.3} \pm 0.1$
WMT'14 EN-DE	DEEP	224.0M	218.1M	3.86 ± 0.02	$\mathbf{3.69} \pm 0.01$	29.2 ± 0.1	$\mathbf{29.5} \pm 0.1$
WMT'14 EN-FR	BASE	60.8	63.8M	3.61 ± 0.01	$\mathbf{3.42} \pm 0.01$	40.0 ± 0.1	$\mathbf{40.6} \pm 0.1$
WMT'14 EN-FR	BIG	209.8M	221.2M	3.26 ± 0.01	$\mathbf{3.13} \pm 0.01$	41.2 ± 0.1	$\mathbf{41.3} \pm 0.1$
WMT'14 EN-CS	BASE	59.8M	62.7M	4.98 ± 0.04	$\mathbf{4.42} \pm 0.01$	27.0 ± 0.1	$\mathbf{27.6} \pm 0.2$
WMT'14 EN-CS	BIG	207.6M	218.9M	4.43 ± 0.01	$\mathbf{4.38} \pm 0.03$	28.1 ± 0.1	$\mathbf{28.2} \pm 0.1$
LM1B	BIG	141.1M	151.8M	30.44 ± 0.04	$\mathbf{28.60} \pm 0.03$	-	-

Model	Embedding Size	Parameters	Perplexity	BLEU	Δ BLEU
Transformer	128	7.0M	8.62 ± 0.03	21.3 ± 0.1	-
ET	128	7.2M	$\mathbf{7.62} \pm 0.02$	$\mathbf{22.0} \pm 0.1$	**+ 0.7**
Transformer	432	45.8M	4.65 ± 0.01	27.3 ± 0.1	-
ET	432	47.9M	$\mathbf{4.36} \pm 0.01$	$\mathbf{27.7} \pm 0.1$	+ 0.4
Transformer	512	61.1M	4.46 ± 0.01	27.7 ± 0.1	-
ET	512	64.1M	$\mathbf{4.22} \pm 0.01$	$\mathbf{28.2} \pm 0.1$	+ 0.5
Transformer	768	124.8M	4.18 ± 0.01	28.5 ± 0.1	-
ET	768	131.2M	$\mathbf{4.00} \pm 0.01$	$\mathbf{28.9} \pm 0.1$	+ 0.4
Transformer	1024	210.4M	4.05 ± 0.01	28.8 ± 0.2	-
ET	1024	221.7M	$\mathbf{3.94} \pm 0.01$	$\mathbf{29.0} \pm 0.1$	+ 0.2

Conclusion

ET only provides 0.2 BLEU value promotion compared to large transformer, and a bit more obvious improvement on small transformer with BLEU value 0.7, which are minor for MT task.

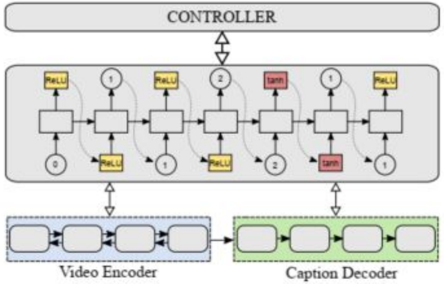
The experiment cannot be reproduced due to huge computation resource requirements. Thus using traditional algorithm including evolution algorithm as well as lage models in NAS is not an ideal direction.

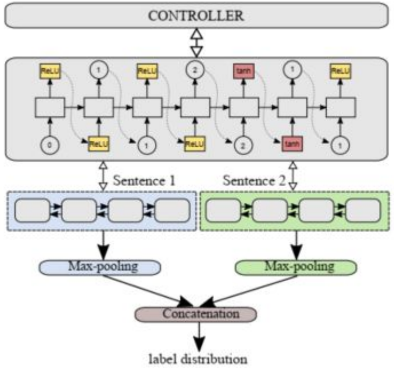
2. Continual and Multi-task Search

Pasunuru, Ramakanth, and Mohit Bansal. "Continual and Multi-Task Architecture Search." Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019.

This paper utilizes **ENAS** (Efficient Neural Architecture Search) with some modifications in sequential or combined multi-task tasks to **enhance the generalization** as well as performance of obtained model architectures.

ENAS: Using a RNN as a controller to determine the network structure. Two steps:
1) controller sampler a architecture and optimize its parameters.
2) controller sampler a architecture and use its validation performance as the loss to optimize the parameters of itself.

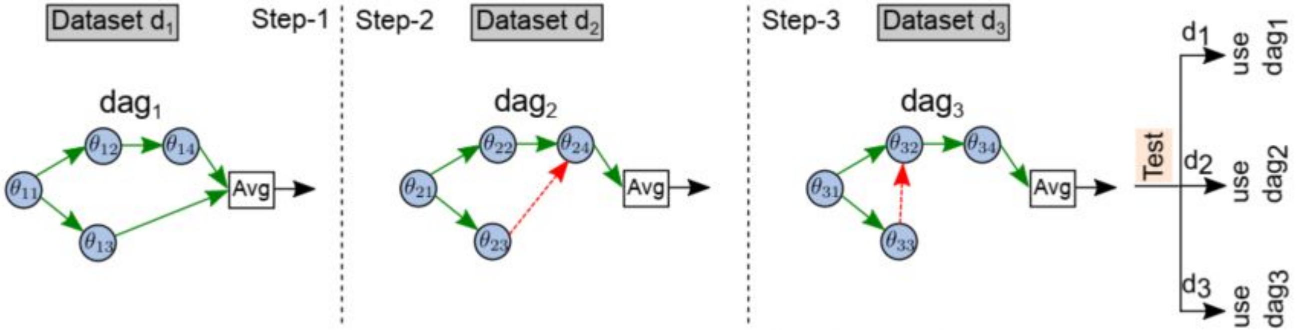




Continual architecture search (CAS)

Given several datasets sequentially

1) The model on the first dataset d1 trained using ENAS and obtaining parameters $\theta_{1,k}$ with sparse constraint $\sum_{i=1}^{m} |(||\theta_{1,k}[i,:]||_2)|_1$ and corresponding architecture dag1
2) In next dataset d2 run ENAS but with parameters initialized from $\theta_{1,k}$, obtaining architecture dag2 and parameters $\theta_{2,k}$ with extra loss item $L_p(\theta_{2,k}) = ||\theta_{1,k}^T \cdot \psi_{2,k}||_2^2$, where is curr$\psi_{2,k}$ parameter change compared to $\theta_{1,k}$
3) Continue 2) for following datasets, using the final parameters but corresponding architecture in evaluating

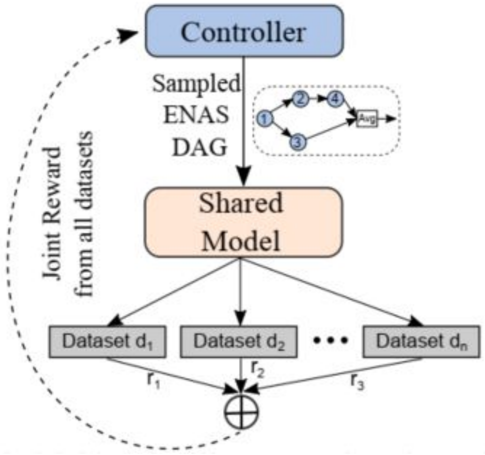


Multi-task architecture search (MAS)

Given several datasets at the same time.

All datasets will use the shared model, but the loss for training the controller will become the joint loss for current model on all datasets $r_c = \frac{1}{n}\sum_{i=1}^{n} r_i$

Such obtained architecture can obain a higher generalization on all datasets.



Experiments

Both CAS and MAS are tested on text classification tasks (QNLI, RTE, WNLI) and video captioning tasks (MSR-VTT, MSVD, DiDeMo)

The generalization performance indeed shows promotion due to more data is involved in the training.

Performance on text classification by CAS compared to baselines

Models	QNLI	RTE	WNLI
PREVIOUS WORK			
BiLSTM+ELMo (2018)	69.4	50.1	65.1
BiLSTM+ELMo+Attn (2018)	61.1	50.3	65.1
BASELINES			
Baseline (with ELMo)	73.2	52.3	65.1
ENAS (Architecture Search)	74.5	52.9	65.1
CAS RESULTS			
CAS Step-1 (QNLI training)	73.8	N/A	N/A
CAS Step-2 (RTE training)	73.6	54.1	N/A
CAS Step-3 (WNLI training)	73.3	54.0	64.4

Performance on RTE by raw LSTM, ENAS on each dataset and MAS

Cell Structure	Performance on RTE
LSTM cell	52.3
QNLI cell	52.4
WNLI cell	52.2
RTE cell	52.9
Multi-Task cell	53.9

Performance on DiDeMo by raw LSTM, ENAS on each dataset and MAS

Cell Structure	Performance on DiDeMo			
	M	C	B	R
LSTM cell	12.7	26.7	7.6	30.6
MSR-VTT cell	12.9	25.7	7.4	30.3
MSVD cell	12.1	25.2	7.9	30.6
DiDeMO cell	13.1	27.1	7.9	30.9
Multi-Task cell	13.4	27.5	8.1	30.8

Performance on video captioning by CAS compared to baselines

Models	MSR-VTT					MSVD				
	C	B	R	M	AVG	C	B	R	M	AVG
Baseline (Pasunuru and Bansal, 2017b)	48.2	40.8	60.7	28.1	44.5	85.8	52.5	71.2	35.0	61.1
ENAS	48.9	41.3	61.2	28.1	44.9	87.2	52.9	71.7	35.2	61.8
CAS Step-1 (MSR-VTT training)	48.9	41.1	60.5	27.5	44.5	N/A	N/A	N/A	N/A	N/A
CAS Step-2 (MSVD training)	48.4	40.1	59.9	27.1	43.9	88.1	52.4	71.3	35.1	61.7

3. Using DARTS in LM and NER

Jiang, Yufan, et al. "Improved differentiable architecture search for language modeling and named entity recognition." Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2019.

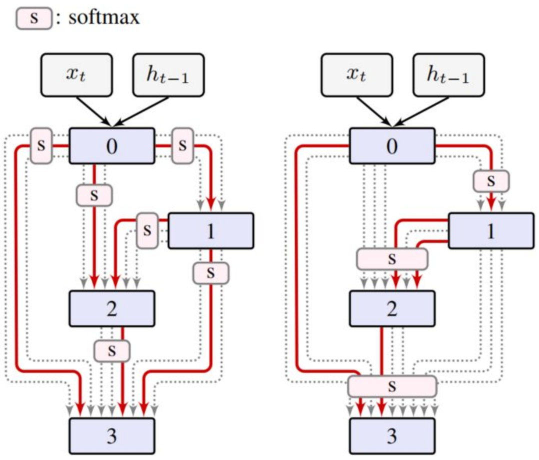
This paper tries to improve the performance of searched architecture by **modifying the raw DARTS** method.

Raw DARTS: softmax weight is calculated independently on the edge node(cell) to node(cell).

$$\alpha_k^{i,j} = \frac{\exp(w_k^{i,j})}{\sum_{k'} \exp(w_{k'}^{i,j})}$$

Modified DARTS: the weight of all edges imported to a node will be calculated together

$$\alpha_k^{i,j} = \frac{\exp(w_k^{i,j})}{\sum_{j<i} \sum_{k'} \exp(w_{k'}^{i,j})}$$



(a) DARTS cell (b) Our cell

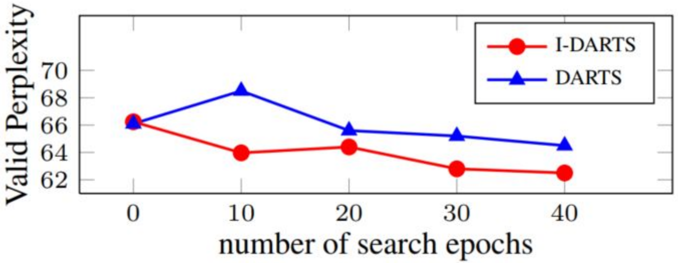
Experiment

Such approach is substantially a additional pruning compared to raw one.

It is tested on PTB LM task and CoNLL-2003 NER task, showing a very slight promotion on performance and search cost compared to DARTS.

CoNLL-2003 NER

Model	F1
best published	
BiLSTM-CRF (Lample et al., 2016)	90.94
BiLSTM-CRF+ELMo (Peters et al., 2018)	92.22
BERT Base (Devlin et al., 2018)	92.40
BERT Large (Devlin et al., 2018)	92.80
BiLSTM-CRF+PCE (Akbik et al., 2019)	93.18
Random RNNs w/o pre-trained LM	90.64
DARTS w/o pre-trained LM	91.05
I-DARTS ($n = 2$) w/o pre-trained LM	90.96
I-DARTS ($n = 1$) w/o pre-trained LM	91.23
Random RNNs	92.89
DARTS	93.13
I-DARTS ($n = 2$)	93.14
I-DARTS ($n = 1$)	93.47



PTB

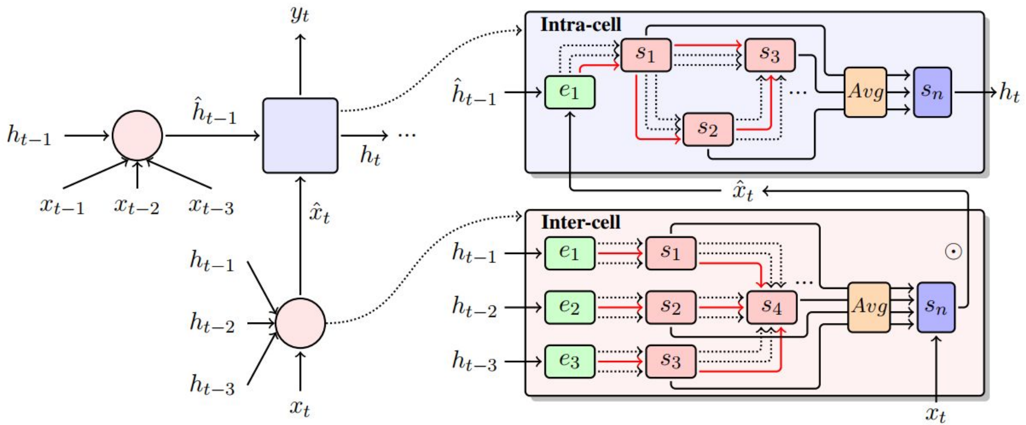
Architecture	Perplexity		Search Cost (GPU days)
	val	test	
V-RHN	67.9	65.4	-
LSTM	60.7	58.8	-
LSTM + SC	60.9	58.3	-
LSTM + SE	58.1	56.0	-
ENAS	60.8	58.6	0.50
DARTS	58.3	56.1	0.25
Random RNNs	63.7	61.2	-
I-DARTS ($n = 1$)	58.0	56.0	0.17
I-DARTS ($n = 2$)	-	-	-

4. Further extend DARTS on LM and NER

Li, Yinqiao, et al. "Learning Architectures from an Extended Search Space for Language Modeling." Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020.

This paper is an extend of previous paper, who uses both intra-cell level DARTS (the same as original DARTS) and **inter-cell level DARTS** (new parts, substantially adding attention to RNN) on LM and NER tasks.

For a RNN, inter-cell learns the way how current cell connecting with previous cells and the input vectors. While intra-cell learns the intra architecture of a cell.



Combine inter-cell and intra-cell search

It splits RNN into 3 functions, $f(\cdot)$ that generates $\hat{h}_{t-1}$, $g(\cdot)$ that generates $\hat{x}_t$ and $\pi(\cdot)$ takes the output from former two functions and generate h_t . Since each function has two input vectors, DAGs can be created on them separately and the final output is the element-wise product of the last node from them $F(\alpha;\beta) = s^\alpha \odot s^\beta$

The original claimed two input sources for each function are

Function	α	β
$\pi(\cdot)$	$\{\hat{h}_{t-1}, \hat{x}_t\}$	1
$f(\cdot)$	$h_{[0,t-1]}$	$x_{[1,t-1]}$
$g(\cdot)$	$x_{[1,t]}$	$h_{[0,t-1]}$

But in the real implementation, **they are simplified**(???) as

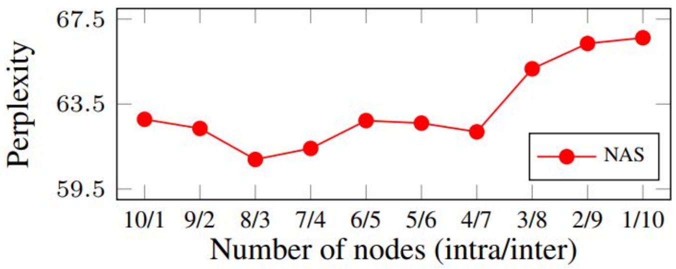
$$\begin{aligned} f(h_{[0,t-1]}; x_{[1,t-1]}) &= f'(h_{t-1}; x_{[t-m,t-1]}) \\ g(x_{[1,t]}; h_{[0,t-1]}) &= g'(x_t; h_{[t-m,t-1]}) \end{aligned}$$

Only one DAG is remained in each function so the function F is totally unused?. In fact it is just a refined version of DARTS with two intermediate state $\hat{h}_{t-1}$ an $\hat{x}_t$,more previous state rather than last step are considered, just like RNN with attention.

Experiments

It is also tested on LM tasks (PTB and WikiText-103) and NER tasks (CoNLL-2003, WNUT-2017, CoNLL-2000, same architecture transferred from WikiText-103). There is no doubt that this method is obviously better than DARTS (it is substantially RNN with attention compared to raw RNN).

Dataset	Method	Search Space		Params	Perplexity		Search Cost
		intra-cell	inter-cell		valid	test	(GPU days)
PTB	AWD-LSTM (Merity et al., 2018c)	-	-	24M	61.2	58.8	-
	Transformer-XL (Dai et al., 2019)	-	-	24M	56.7	54.5	-
	Mogrifier LSTM (Melis et al., 2019)	-	-	23M	**51.4**	**50.1**	-
	ENAS (Pham et al., 2018)	✓	-	24M	60.8	58.6	0.50
	RS (Li and Talwalkar, 2019)	✓	-	23M	57.8	55.5	2
	DARTS†	✓	-	23M	55.2	53.0	0.25
	ESS	-	✓	23M	54.1	52.3	0.5
	ESS	✓	✓	23M	**47.9**	**45.6**	0.5
WT-103	QRNN (Merity et al., 2018a)	-	-	151M	32.0	33.0	-
	Hebbian + Cache (Rae et al., 2018)	-	-	-	29.9	29.7	-
	Transformer-XL (Dai et al., 2019)	-	-	151M	**23.1**	**24.0**	-
	DARTS†	✓	-	151M	31.4	31.6	1
	ESS	✓	✓	156M	**28.8**	**29.2**	1.5



5. Sandwich architecture for Transformer

Press, Ofir, Noah A. Smith, and Omer Levy. "Improving Transformer Models by Reordering their Sublayers." Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020.

This paper split a transformer layer into self-attention sublayer s and feedforward sublayer f and a raw transformer can be represented by

sfsfsfsfsfsfsfsfsfsfsfsfsfsf

They reorder these sublayers randomly to form a new transformer, trying to promote its performance. And based on their analysis, they proposes a **sandwich transformer** in which multiple redundant s are stacked in the lower layers while multiple redundant f are stacked in the higher layers.

sssssssfsfsfsfsfsfsfsfffffff

This paper can be regarded as a **simplified cell-level NAS** in which there is only one input edge and one output edge in the DAG for each cell.

Random search

s and f can be defined as the corresponding layer and residual connection after it.

$$\mathbf{X}_1 = \text{self-attention}(\mathbf{X}_0) + \mathbf{X}_0$$

$$\mathbf{X}_2 = \text{feedforward}(\mathbf{X}_1) + \mathbf{X}_1$$

Taking a 16-layer 16-head transformer with d=1024 as the baseline, each s contains 4d^2 parameters and each f contains 8d^2 parameters(omitting bias).

First 20 unbalance transformers with 16 s and 16 f randomly ordered to test their PPL on WikiText-103, with raw transformer bold. It can be found most random architectures are worse than the raw one.

Model	PPL
fsfsfffsffsfsssffsfssfssssffsffs	20.74
sfssffsffffssssfsfffsfsffsfssssf	20.64
fsffssffssssffsssssffsfssfsfffff	20.33
fsffffffsssfssffsfssffsfsssffsss	20.27
fssffffffsfsssfffssssfffssssffss	19.98
sssfssfsffffssfsfsfsssffsfsfffsf	19.92
fffsfsssfsffsfsffsffsssssffssffs	19.69
fffsffssffsssfssfsssfffffsfsssfs	19.54
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**19.13**
fsffssfssfffssssfffsssffffsfssfs	19.08
sfsffssssffssffffsssffsssfsffsff	18.90
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.83**
sssssssffsffsfsfsffffsfffsfssffs	18.83
sffsfsffsfsssffssfssssssfffffffs	18.77
sssfssffsfssfsffsfffssffsfsffssf	18.68
fffsssssfffsfssssffsfsfsfssffsff	18.64
sfffsssfsfssfsssssfssfffffsfffsf	18.61
ssffssfssssffffffssffsssfsffssff	18.60
fsfsssssfsfsfffffsfffsffssffssss	18.55
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.54**
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.49**
fsfsssssfsfffssfsffsfsfsfsffffss	18.38
sfssffsfsfsffsssssfffsssfffsffsf	18.28
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.25**
sfsfssfsssffsfsfsfsffffssffsfssf	18.19

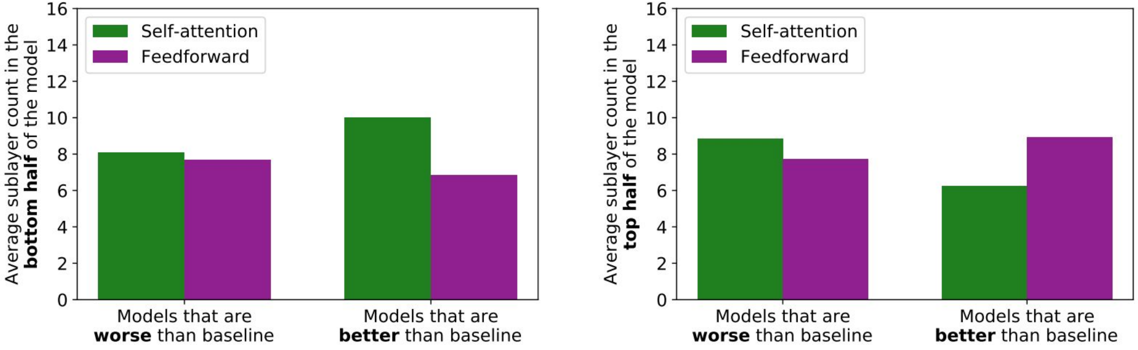
Random search

Then it randomly samples 20 architectures with the equivalent parameter numbers as the raw transformer but different layer numbers (may contains different numbers of s and f , the model depth ranges from 24 layer (all f) to 48 layers (all s)), also found general worse performance than the raw one.

Model	PPL
sfffssfsfsfssffffsfsffsffffff	22.80
sffssfsssssssssssssfsfsssfsffsssfsssfs	21.02
ssssssffsffffssfffffsssfsfsssssssss	20.98
fffffffffsffssffsffssssfsfsssf	20.75
fssfsssffffffssfsssfsfffssssfsfss	20.43
sffsffffffsfsfssfsssfsfsfssfssfs	20.28
sffssffsfffsfsfssssffffffssssff	20.02
fsffsfssffffsfsfffsfffssfffsss	19.93
sffsffssffsfsffsssfsssssfsssfffsss	19.85
ssfffffffssfffssfssffsfsfsffsf	19.82
sfsfsfffsfffssfsfffsffssfsfsfss	19.77
sfsffsssffsffsssfssfffffssssfsssf	19.55
sffsfssfffsffsfssssfsfsffffsfsss	19.49
sffffsffssssfsssfssfffsssfssssfsfs	19.47
fsssffssssssfsfsfsffsffffssfsfssss	19.25
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**19.13**
fssssssfsfsfsfffsfsssfssffssssfsff	18.86
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.83**
ssfsfsssfsssssffsfsfsssfssfsfsssssssf	18.62
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.54**
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.49**
sssfsffsfssfsssffsffffffssfsfff	18.34
sssfsfsffsssfsfffffsfsffffsssff	18.31
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.25**
ssssssfsssffffsfsfffffffffffsf	18.12

Sandwich Transformer

However, they find that better models usually have more s at first and more f at the last of the model according to their analysis



Therefore they propose sandwich transformer, with first k layers as s, last k layers as f and n-k sf in the middle, which can be represented as $s^k (sf)^{n-k} f^k$

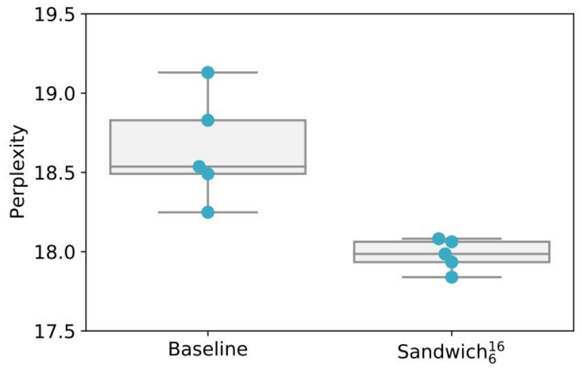
n=16, k ranges from 0 to 15, the best k is determined by the best performance on a specific acrossing all enumerations.

Experiments

The sandwich model is tested on LM tasks (WikiText-103, Toronto Book Corpus, performance is measured by PPL), and character-level LM (text8 and enwik8) which shows slight promotion compared to transformer baselines.

WikiText-103

Model	Test
Baseline (Baevski and Auli, 2019)	18.70
Transformer XL (Dai et al., 2019)	18.30
kNN-LM (Khandelwal et al., 2019)	15.79
Baseline (5 Runs)	18.63 ± 0.26
Sandwich_6^{16}	17.96



Character-level LM

Model	text8 (BPC)	enwik8 (BPC)
Transformer-XL (Dai et al., 2019)	1.08	0.99
Adaptive Span (Sukhbaatar et al., 2019)	1.07	0.98
Compressive (Rae et al., 2020)	—	0.97
Baseline (Adaptive Span; 5 Runs)	1.0802 ± 0.0103	0.9752 ± 0.0008
Sandwich_3^{24}	1.076	—
Sandwich_5^{24}	—	0.968

Book Corpus

Model	PPL
Baseline (5 runs)	11.89 ± 0.35
kNN-LM (Khandelwal et al., 2019)	10.89
Sandwich_7^{16}	10.83

Experiments

It is also tested on MT tasks (WMT2014 En-de) using encoder-decoder architecture with additional cross-attention sublayer c involved in decoder.

$$\mathbf{Y}_2 = \text{cross-attention}(\mathbf{Y}_1, \mathbf{X}) + \mathbf{Y}_1$$

The concatenation of self-attention and cross-attention sc is used to replace s in the original sandwich. Using 6 layers in both encoder and decoder, with k varies from 0 to 5 with sandwich applied to encoder or decoder, it only achieves very slight promotion in BLEU under specific configurations.

Sandwich Coefficient	**Encoder Sandwich**	**Decoder Sandwich**
0 (Baseline)	28.74 ± 0.15	
1	28.71	28.64
2	28.71	28.56
3	28.81	28.67
4	28.48	28.66
5	28.45	28.76

Conclusion

- NAS in traditional NLP tasks usually cannot obtain as significant promotion as CV, as most popular models (e.g. transformers, pre-trained models) are already carefully designed.
- Gradient-based optimization for NAS is becoming a main direction instead of Evolution Algorithm or Reinforcement Learning due to the much less computing resource requirement (considering the 1M GPU hours in Evolved Transformer)
- Trying to simplifying or redesign the NAS problem for current NLP models is a possible direction (e.g. sandwich transformer), but I don’t think fine-grained NAS on NLP models is a good idea.
- Maybe NAS can be applied to some special setting of NLP tasks, e.g. few-shot learning, UDA, generalization across different datasets.

Thank you!