A Survey on Dynamic Neural Networks for Natural Language Processing

Canwen Xu, Julian McAuley
University of California, San Diego
{cxu,jmcauley}@ucsd.edu

Abstract

Effectively scaling large Transformer models is a main driver of recent advances in natural language processing. Dynamic neural networks, as an emerging research direction, are capable of scaling up neural networks with sub-linear increases in computation and time by dynamically adjusting their computational path based on the input. Dynamic neural networks could be a promising solution to the growing parameter numbers of pretrained language models, allowing both model pretraining with trillions of parameters and faster inference on mobile devices. In this survey, we summarize the progress of three types of dynamic neural networks in NLP: skimming, mixture of experts, and early exit. We also highlight current challenges in dynamic neural networks and directions for future research.

1 Introduction

Scaling up model capacity is an obvious yet effective approach for better performance in natural language processing (NLP) tasks (Brown et al., 2020; Kaplan et al., 2020; Ghorbani et al., 2021; Zhou et al., 2020b). However, the resulting increase in computational complexity and memory consumption becomes a bottleneck for scaling, making these models hard to train and use. On the other hand, it is not necessary to allocate the same amount of computation to all instances. For example, categorizing "I love you" as a positive sentence does not require a model containing dozens of Transformer layers. To resolve the aforementioned problems, *dynamic neural networks* have been a significant thrust of recent research in NLP. Dynamic networks can adjust their computational path based on the input for better efficiency, making it possible to train models with trillions of parameters and accelerate models in a low-resource setting.

In this survey, we review the latest state of research on three types of dynamic neural networks

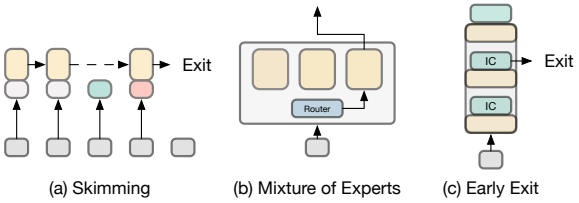

Figure 1: The three types of dynamic neural networks summarized in this paper. They dynamically adjust computation *timewise*, *widthwise* and *depthwise*, respectively.

that have been adopted in NLP: *skimming*, *mixtures of experts* (MoE), and *early exit*, as illustrated in Figure 1. These three types of techniques share a common idea of dynamically adjusting computation with respect to input, to save computation through bypassing unnecessary modules in a large neural network. However, they implement the goal via different approaches. **Skimming** was well-researched in the era of recurrent neural networks (RNN). Skimming models save computation *timewise* by dynamically allocating computation to different time steps, based on the input tokens. Since RNN models process the input sequence recurrently, it allows skimming models to achieve a substantial acceleration, especially when the sequence is long (Li et al., 2019). Different from RNN, recent works on Transformers skip tokens between layers instead of time steps.

For Transformer models (Vaswani et al., 2017; Devlin et al., 2019; Lan et al., 2020; Brown et al., 2020), the input tokens are fed into the model in parallel, while models have dozens of Transformer layers. This motivates the development of MoE and early exit. **MoE** horizontally extends a feedforward neural network (FFNN) with multiple sub-networks. During inference, only one or a few of these sub-networks will be activated for computation, thus can save *widthwise* computation. **Early exit**, on the other hand, terminates inference

Findings of the Association for Computational Linguistics: EACL 2023, pages 2370–2381

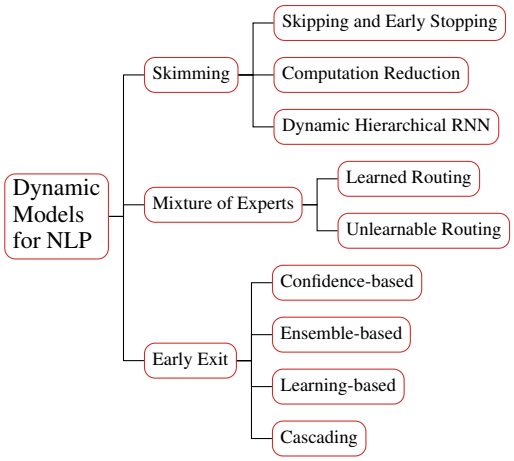

Figure 2: Taxonomy of dynamic neural networks for NLP.

at an early layer, without exhausting full computational capacity, thus saves *depthwise* computation. Early exit techniques often insert a series of lightweight classifiers which help decide when to exit, based on an exit strategy.

Note that this stream of works is distinct from static model acceleration, which is often referred to as *model compression*, including knowledge distillation, weight sharing, pruning and quantization (Sanh et al., 2019; Xu et al., 2020; Lan et al., 2020; Zafrir et al., 2019; Xu et al., 2021) (etc., see another survey (Xu and McAuley, 2022)). The major difference is that the computational path in a statically compressed model does not condition on the input and is invariable for all examples in inference. These two streams of research are in fact orthogonal and recent works Schwartz et al. (2020), Liu et al. (2020) and Zhu (2021) have shown that static and dynamic approaches can be combined for even faster inference and better performance.

To summarize, our contribution is two-fold: (1) We review the latest studies on the topic of dynamic neural networks for NLP by providing a comprehensive comparison and organize them with a new taxonomy, as shown in Figure 1. (2) We analyze current challenges in dynamic neural networks and point out directions for future research.

2 Skimming

Skimming techniques, as summarized in Table 1, skip some time steps or allocate different computation on them. Intuitively, skimming matches how human beings efficiently read text and extract information from it (Li et al., 2019). By emphasizing the important information within a sequence and ignoring parts with little importance, skimming helps the model achieve faster inference speed and better capture long-term dependencies. The three categories of skimming are *skipping and early stopping*, *computation reduction*, and *dynamic hierarchical RNN*, corresponding with three motivations: to skip unimportant input, to allocate less computation to unimportant input, and to increase computation to important input only.

Skipping and Early Stopping Skipping and early stopping aim to improve efficiency for a long sequence by skipping some tokens or stopping reading early. LSTM-Jump (Yu et al., 2017) is a skipping mechanism to ignore irrelevant information for natural language understanding (NLU). At each step, the current states are used to compute a "jumping softmax", which decides how many steps to jump forward and whether to stop reading. LSTM-Jump employs policy gradient to train the model to make non-differentiable discrete jumping decisions. The reward is a binary function which rewards a correct prediction and penalizes an incorrect prediction of the label. Compared to a standard LSTM, LSTM-Jump achieves better accuracy with up to 6× speed-up. Skip RNN (Campos et al., 2018) introduces a binary gate to learn whether to skip a state update. If the gate decides to skip a time step, the hidden states will be directly copied without any update.

To stop reading early as needed, ReasoNet (Shen et al., 2017) introduces a terminal state which decides whether to terminate early for machine reading comprehension on each time step at the token level. Jumper (Liu et al., 2018) first splits a paragraph to several sub-sentences and encodes them into sentence embeddings. They then apply early stopping at a sentence when the policy network decides to stop reading. Li et al. (2019) use eye-tracking devices and confirm that skipping and early stopping are common when humans read text. They propose Reading Inspired Model to mimic the behaviors of humans, which allows the model to decide whether to skip a single time step or stop reading early. Yu et al. (2018) add a re-reading operation to LSTM-Jump (Yu et al., 2017) which allows the model to stay on the current time step, allocating more computation to important information.

The aforementioned techniques can only go forward, which makes it impossible to regret if hav-

Method	Decision based on	Operation options
LSTM-Jump (Yu et al., 2017)	hidden states	skip multiple steps; stop reading
Skip RNN (Campos et al., 2018)	states of the update gate; hidden states	skip a single step
ReasoNet (Shen et al., 2017)	hidden states	stop reading
Jumper (Liu et al., 2018)	input sentence; hidden states	stop reading
RIM (Li et al., 2019)	input sentence; hidden states	skip a single step; stop reading
Yu et al. (2018)	hidden states	skip multiple steps; stop reading; re-read
LSTM-Shuttle (Fu and Ma, 2018)	hidden states	skip multiple steps; jump back multiples steps
Struc. Jump-LSTM (Hansen et al., 2019)	hidden states	stop reading; jump to next (,;) or (.!?)
PoWER (Goyal et al., 2020)	attention	drop tokens
TR-BERT (Ye et al., 2021)	hidden states	forward tokens
LAT (Kim and Cho, 2021)	attention	forward tokens
LTP (Kim et al., 2022)	attention	drop tokens
Transkimmer (Guan et al., 2022)	hidden states	forward tokens
VCRNN (Jernite et al., 2017)	input token; hidden states	partial update with zero-masked weights
Skim-RNN (Seo et al., 2018)	input token; hidden states	partial update with a small RNN
HM-RNN (Chung et al., 2017)	states of the gates	skip a single step; "flush"
FHRNN (Ke et al., 2018)	query; hidden states	update the upper RNN layer

Table 1: A summary of skimming techniques.

ing jumped over important information. LSTM-Shuttle (Fu and Ma, 2018) proposes a bidirectional shuttling mechanism, which can jump multiple time steps both forward and backward, allowing the model to ignore unimportant information and recover lost information if needed.

Structural information that naturally exists in sentences can also play a role in skimming. Structural Jump-LSTM (Hansen et al., 2019) can jump to the next word, next sub-sentence separator (a comma or colon), next sentence end symbols (a period, exclamation mark or question mark), or to the end of the text (i.e., stop reading).

In the era of Transformers, there have been works attempting to reduce computation by either skip tokens at higher layers or forward tokens to higher layers. The PoWER-BERT model (Goyal et al., 2020) reduces the number of tokens processed by each Transformer layer based on their attention scores. The number of tokens to be dropped, referred to as the "schedule," is optimized by combining the sparsity of a soft mask layer with the original loss function. This results in an improved balance between accuracy and processing time. TR-BERT (Ye et al., 2021) uses a dynamic approach to determine which tokens to skip, using reinforcement learning to train the model with a reward system that prioritizes classifier confidence while also penalizing the number of tokens retained. In contrast to PoWER-BERT, TR-BERT passes the skipped tokens to the final layer rather than discarding them. The Length-Adaptive Transformer (LAT, Kim and Cho, 2021) utilizes LengthDrop to randomly skip tokens during pretraining, aiming to close the gap between pretraining and fine-tuning. The schedule for LAT is found through an evolutionary search algorithm. LTP (Kim et al., 2022) trains a threshold for each Transformer layer, instead of following a predetermined schedule. It simply drops tokens with attention scores lower than the learned threshold. Transkimmer (Guan et al., 2022) incorporates a skim predictor module, consisting of a small MLP and Gumbel-Softmax reparameterization, before each layer. This module outputs a mask to determine whether a token should be dropped, and a skim loss is used to optimize the ratio of skipped tokens to total tokens, promoting sparsity.

Computation Reduction Different from skipping, computation reduction applies a reduced computational workload for some time steps instead of skipping it completely. VCRNN (Jernite et al., 2017) explores a scheduler to decide which proportion of computation to use for each time step. Upon making the decision, only the corresponding proportion of the weight matrix will be used to update the hidden states while the rest part of the weight matrix will be masked out with zero.

Instead of using part of weights to update the hidden states, Skim-RNN (Seo et al., 2018) has a big RNN and a separate small RNN. At each time step, the model decides whether to read or skim based on hidden states from the last time step and the input token. If the model decides to skim, the small RNN will update only a fraction of the hid-

den states. Otherwise, a regular full update will be conducted by the big RNN.

Dynamic Hierarchical RNN Different from the aforementioned two categories of skimming, dynamic hierarchical RNN can *increase* computation by calling the upper layer RNN when needed. HM-RNN (Chung et al., 2017) automatically discovers the hierarchical multi-scale structure in the data for a hierarchical RNN architecture. In addition to the update and copy operations as in Skip RNN (Campos et al., 2018), they add a flush operation which ejects the summarized representation of the current time step to the upper layer and reinitializes the states for the next time step.

In question answering, only a small portion of tokens are relevant and can be used to answer the question while the rest can be safely skimmed. Based on this observation, Focused Hierarchical RNN (Ke et al., 2018) aims to only pick up information that is relevant to the query for question answering. It applies a binary gate to control whether to update the upper layer of the RNN, based on the current hidden states of the lower-level RNN and the question embedding.

3 Mixture of Experts

Increasing the number of parameters in a model often leads to increased computation and memory consumption. To take the advantages of parameter scaling without a proportional increase in computation, mixture of experts (MoE) (Jacobs et al., 1991) is introduced to large language models, as summarized in Table 2. In these models, a layer typically contains multiple sub-networks (i.e., "experts"). During inference, only part of these experts will be activated on a per-example basis.

The key element of MoE methods is the routing mechanism. The routing mechanism has to be lightweight, not to significantly slower the speed of the model. We categorize MoE methods into two groups: *learned routing* and *unlearnable routing*. Learned routing often requires some load balancing mechanisms to ensure that all experts are trained with enough examples thus are useful during inference. Unlearnable routing usually slightly underperforms learned routing but does not require complicated load balancing.

MoE Layers with Learned Routing A straightforward idea to implement MoE is to learn a router to allocate inputs to experts. Sparsely-Gated MoE layer (Shazeer et al., 2017) contains up to thousands of feed-forward sub-networks with a trainable gating network which determines a sparse combination of these experts to use for each example. There are two major challenges to address: **(1) Sparsity.** The gating network predicts a softmax weight for the experts based on the input. The gating network is trained by simple back-propagation, together with other parts of the model. Then, only the top-k experts in the layer will be activated based on the softmax prediction of the gating network. They insert one MoE layer between stacked LSTM layers and achieve improvement on language modeling and machine translation tasks. **(2) Load balancing.** Shazeer et al. (2017) observe a self-reinforcing phenomenon that the gating network tends to converge to a state where it always produces large weights for the same few experts. They resolve the problem by defining the importance of an expert relative to a batch of training examples to be batch-wise sum of the *gate values* for that expert. Then, they introduce an additional loss, the square of the coefficient of variation of the set of importance values, to encourage a more balanced update during training. Besides encouraging a balanced update, the authors also introduce a loss function with a smooth estimator that estimate the number of examples assigned to each expert for a batch of inputs, to encourage experts to receive roughly equal *numbers of training examples*.

GShard (Lepikhin et al., 2021) enables scaling up multilingual neural machine translation Transformer beyond 600 billion parameters. It adapts Sparsely-Gated MoE (Shazeer et al., 2017) to Transformer (Vaswani et al., 2017) by replacing every other feed forward layer with an MoE layer, which routes to top-2 experts. When scaling to multiple devices, the MoE layer is sharded across devices, i.e., each device has different allocated experts, while all other layers are replicated. To achieve workload balance, GShard employs a threshold, namely expert capacity, to limit the maximum number of tokens processed by one single expert. They also introduce a local group dispatching mechanism, which partitions all tokens in a training batch evenly into groups to be processed independently in parallel, to balance the overall workload. Following (Shazeer et al., 2017), they use an additional loss to enforce even

Method	Base Model	Sparsity	Load Balance
Sparsely (Shazeer et al., 2017)	LSTM	top-k	auxiliary loss
GShard (Lepikhin et al., 2021)	Transformer (NMT)	top-2	expert capacity; local group dispatching; auxiliary loss; random routing
Switch (Fedus et al., 2021)	Transformer (T5)	top-1	expert capacity; auxiliary loss
BASE (Lewis et al., 2021)	Transformer (GPT)	top-1	linear assignment
M6-T (Yang et al., 2021)	Transformer (M6)	k top-1	expert capacity
DTS (Nie et al., 2021)	Transformer (GPT)	dynamic	sparsity scheduler
Hash (Roller et al., 2021)	Transformer	hash	deterministic hash
THOR (Zuo et al., 2022)	Transformer (NMT)	random	random selection

Table 2: A summary of Mixture of Experts (MoE) methods.

allocation for experts. Additionally, they propose a random routing mechanism, which only routes to the second-best expert with probability proportional to its weight, to simplify sparse training.

Switch Transformer (Fedus et al., 2021) aims to simplify the Sparsely-Gated MoE (Shazeer et al., 2017) for efficiency and performance. They propose a Switch Layer which only routes to one expert at a time, to reduce gating computation, batch size and communication costs. Switch Transformer inherits expert capacity and an auxiliary load balancing loss from GShard (Lepikhin et al., 2021). Combined with low-precision training, compared to T5-Base and T5-Large (Raffel et al., 2020), Switch Transformer obtains up to 7× increases in pretraining speed with the same computational resources. They further scale Switch Transformer to more than 1.5 trillion parameters and achieve 4× speed-up over T5-XXL.

The Balanced Assignment of Sparse Experts (BASE) layer (Lewis et al., 2021) formulates token-to-expert allocation as a linear assignment problem and solves it with the auction algorithm (Bertsekas, 1992). This allows an optimal assignment in which each expert receives an equal number of tokens, improving efficiency and getting rid of the expert capacity and auxiliary loss in previous works. The experiments show that BASE layers are more efficient for training compared to Sparsely-Gated MoE layers (Shazeer et al., 2017) and Switch Layers (Fedus et al., 2021), and can successfully learn a good balanced routing without any auxiliary balancing loss.

M6 (Lin et al., 2021) is a multi-modal multitask Transformer, trained in the same way as Switch Transformer (Fedus et al., 2021), scaling up to 100B parameters. Following this, M6-T (Yang et al., 2021) splits experts into k prototypes (i.e., groups of experts). In each forward pass, each token is sent to the k prototypes, within which the top-1 routing is done locally. The experiments demonstrate this "k top-1" strategy outperforms the top-1 routing in Switch Transformer (Fedus et al., 2021) while being more computation-efficient than "top-k" routing. They also claim that the load balancing loss may be ineffective for improving the performance of an MoE model, although it can indeed help balance the workload. They subsequently train a 1 trillion parameter model with the finding.

Dense-to-Sparse gate (Nie et al., 2021) begins as a dense gate that routes tokens to all experts then gradually learns to become sparser and route tokens to fewer experts, demonstrating higher training efficiency in experiments. Their experiments confirm the finding in Yang et al. (2021) that an auxiliary load balancing loss does not improve the model performance.

MoE Layer with Unlearnable Routing Although learning-based routing has shown effectiveness only with the help of complicated load balancing mechanisms, recent studies have attempted to get rid of those. Hash Layer (Roller et al., 2021) simplifies routing by using a parameter-free hashing function to route tokens to specific experts. This design eliminates the need for a load balancing loss and sophisticated assignment algorithms. They also study the performance of different hashing techniques, hash sizes and input features, and conclude that balanced and random hashes focused on the most local features work best. The experiments show that a Hash Layer achieves comparable performance with a Switch Layer (Fedus et al., 2021) and BASE Layer (Lewis et al., 2021).

THOR (Zuo et al., 2022) is a special form of MoE layer, which completely discards the conditional routing mechanism and instead optimizes the consistency between a randomly selected pair of experts. During inference, one expert will be randomly selected to be activated.

Applications and Analysis GLaM (Du et al., 2021) trains a family of GPT-style language models with up to 1.2 trillion parameters using GShard (Lepikhin et al., 2021). CPM-2 (Zhang et al., 2022b) trains a large Chinese language model with 198 billion parameters with BASE layers (Lewis et al., 2021).

Artetxe et al. (2021) conduct a detailed empirical study of how autoregressive MoE language models scale compared to dense models. They find MoEs to be substantially more efficient with the exception of fine-tuning. MoE models can match the performance of dense models with 25% of computation in a low-resource setting. Although the advantage fades at scale, their largest MoE model with 1.1 trillion parameters can consistently outperform its dense counterpart with the same amount of computation. Clark et al. (2022) examine the scaling law of BASE Layer (Lewis et al., 2021), Hash Layer (Roller et al., 2021) and earlier Reinforcement Learning-based routing algorithms providing suggestions for best-practices in training MoE models.

Zhang et al. (2021) propose MoEfication to split feedforward neural networks (FFNN) in a trained large model to experts. They find that a T5-Large (Raffel et al., 2020) model with 700 million parameters only activates 5% neurons for 80% inputs on a downstream task, indicating high redundancy within large pretrained language models. To transform a pretrained language model to an MoE model, they first construct a co-activation graph for each FFNN and then divide the graph into subgraphs with strong internal connections with graph partitioning algorithm. Each subgraph forms an expert. They train a router with oracle best routing for training data. Then, they further fine-tune the resulted model for better performance.

4 Early Exit

Early exit techniques aim to terminate model inference in early layers, to save computation and sometimes improve performance by resolving the overthinking problem (Kaya et al., 2019), i.e., possible performance degradation at a later layer. It can be useful especially in the era of pretrained language models (PLM), since increasing the size of PLMs can often lead to better performance, although a smaller model can already predict most examples (i.e., "easy examples") correctly.

The main idea of early exit is to exit inference at an earlier layer, rather than the last layer. Early exit often involves a series of internal classifiers inserted into a large network, providing signals for early exiting. The core of early exit methods is the exit criterion. Based on their exit strategies, we categorize the early exit methods into three classes: confidence-based, ensemble-based and learning-based, as listed in Table 3.

Despite better performance, speed and adversarial robustness (Zhou et al., 2020a), an additional benefit is that the speed-accuracy trade-off can be adjusted as needed by tuning the exit threshold (θ in Table 3), without the need of retraining the model. A main drawback is that early exit is often applied on a per-instance basis, meaning that to maximize the speed-up ratio, a small batch size (often 1) has to be used.

Confidence-based Early Exit Early works for early exit in computer vision (Park et al., 2015; Teerapittayanon et al., 2016; Kaya et al., 2019) often fall into this category. They define a metric as the proxy for confidence of a model prediction. The model exits early when the confidence hits a predefined threshold. DeeBERT (Xin et al., 2020b) applies BranchyNet (Teerapittayanon et al., 2016) to BERT inference. The training for DeeBERT is two-stage: they first train BERT on downstream tasks following standard fine-tuning. Then, they freeze the parameters of the Transformer and insert a linear classifier (i.e., internal classifier) after each Transformer layer. They train the classifiers by minimizing the sum of their cross-entropy loss. For inference, the model exits early when an internal classifier outputs a prediction probability distribution that has an entropy lower than a predefined threshold. RightTool (Schwartz et al., 2020) jointly fine-tunes BERT with internal classifiers. They use the temperature-calibrated maximum class probability as confidence. FastBERT (Liu et al., 2020) first trains the BERT backbone and the final classifier. Then, they distill the final classifier layer to the internal classifiers (Hinton et al., 2015). For inference, the model exits when the entropy of a prediction is below the threshold. RomeBERT (Geng et al., 2021) provides a simple fix for learning internal classifiers efficiently. Besides self-distillation as in FastBERT, they propose gradient regularization (GR) to facilitate distillation. SkipBERT (Wang et al., 2022) caches pre-computed representation of text chunks to re-

Method	Internal classifier training	Exit criterion
DeeBERT (Xin et al., 2020b)	two-stage; sum of CE loss	entropy $< \theta$
RightTool (Schwartz et al., 2020)	joint; sum of CE loss	calibrated max class probability $> \theta$
FastBERT (Liu et al., 2020)	two-stage; self-distillation	entropy $< \theta$
RomeBERT (Geng et al., 2021)	joint; self-distillation + GR	entropy $< \theta$
SkipBERT (2022)	joint; weighted sum of CE + KD	max class probability $> \theta$
PABEE (Zhou et al., 2020a)	joint; weighted sum of CE loss	patience (#consistent prediction $> \theta$)
Voting (Sun et al., 2021)	joint; sum of CE + diversity loss	accumulated votes $> \theta$
LeeBERT (Zhu, 2021)	joint; auto-weighted sum of CE + KD loss	patience (#consistent prediction $> \theta$)
Past-Future (Liao et al., 2021)	joint; weighted sum of CE + imitation learning	entropy $< \theta$
PCEE-BERT (2022a)	joint; weighted sum of CE	patience (#consistent IC confidence $> \theta$)
BERxiT (Xin et al., 2021)	alternate; sum of CE loss	estimated confidence $> \theta$
CAT (Schuster et al., 2021)	joint; avg. of CE loss	estimated conformity $> \theta$
CascadeBERT (Li et al., 2021a)	standard model FT with confidence calibration	calibrated max class probability $> \theta$

Table 3: A summary of early exit methods. θ is a predefined threshold for exiting. This table is extended from a table in Xu and McAuley (2022).

place lower BERT layers and uses confidence-based early exit for higher layers to achieve maximum acceleration.

Ensemble-based Early Exit One drawback in confidence-based early exit is wasted computation. That is to say, if the confidence of an internal classifier does not satisfy the exit criterion, it will be disregarded. Ensemble-based early exit recycles these predictions and considers output from multiple internal classifiers to make better predictions. Based on the similarity between overfitting and overthinking, PABEE (Zhou et al., 2020a) borrows early stopping from model training. They first jointly train the internal classifiers with BERT by a weighted sum of cross-entropy losses that assigns larger weights for upper classifiers. For inference, the model exits when k consecutive internal classifiers make the same prediction. Other than improvement on performance and efficiency, they find that PABEE can improve adversarial robustness, which they attribute to the ensemble effect. Sun et al. (2021) further introduce a diversity loss that encourages internal classifiers to have a diverse predicted probability distribution. They propose a voting mechanism to ensemble the internal classifiers by exiting early when a class has accumulated more votes than the threshold. Interestingly, LeeBERT (Zhu, 2021) adopts the opposite strategy: they promote consistency across internal classifiers by distilling them to each other. However, they introduce a learnable weight for the cross-entropy loss of each classifier and the distillation loss between each pair. They optimize these weights by a cross-level optimization algorithm. They adopt PABEE's patience-based strategy for exiting. Liao et al. (2021) train linear transformation layers called "imitation learners", to approximate the hidden states of future layers based on current hidden states. For inference, the prediction after each layer is calculated by mixing the past predictions and the future predictions of the imitation learners. Entropy is used as the exit criterion. PCEE-BERT (Zhang et al., 2022a) borrows from both ensemble-based exit and confidence-based methods. The inference is terminated when multiple layers are confident.

Learning-based Early Exit Another stream of research is to *learn* a criterion for early exiting. BERxiT (Xin et al., 2021) alternates between joint fine-tuning and two-stage fine-tuning by freezing parameters of Transformer and the final classifier for even-numbered iterations and unfreezing them for odd-numbered iterations. They also train a linear layer called a *learning-to-exit* (LTE) module to predict whether the current internal classifier makes the correct prediction. It takes the hidden states as input and outputs a confidence score, which is used to decide whether to exit. CAT (Schuster et al., 2021) introduces a "meta consistency classifier" to predict whether the output of an internal classifier conforms to the final classifier and exits when the consistency classifier predicts a certain level of conformity.

Cascading Cascading can be seen as a special form of early exit, performed at the model level. Li et al. (2021a) find that shallow features and internal classifiers in the first few layers of BERT utilized by early exit methods like DeeBERT (Xin

et al., 2020b) are not sufficient and reliable, underperforming a fine-tuned BERT with the same number of layers. Therefore, they propose to use a suite of complete models with different numbers of layers for cascading. CascadeBERT executes models one by one, from the smallest to the largest. It stops when a model outputs a confidence score (calibrated maximum class probability) that reaches the threshold.

Applications Although early exit is originally developed for classification, there have been works extending it to more tasks and settings. Li et al. (2021b) propose Token-Level Early-Exit that targets early exiting for sequence labeling. They use the maximum class probability as confidence on a per-token basis. Once the confidence hits the threshold, the hidden states of the corresponding tokens will be frozen and directly copied to upper layers. These exited tokens will not attend to other tokens at upper layers but can still be attended by other tokens. The model completely exits when every token exits. A similar idea is also presented in Elbayad et al. (2020) and Liu et al. (2021b) where hidden states of some positions can be frozen and directly copied to upper layers, although the former is focused on generation and the latter is for classification. Xin et al. (2020a) apply DeeBERT (Xin et al., 2020b) to document ranking and set different thresholds to the negative and positive classes for early exiting, to accommodate the imbalanced class distribution in document ranking. ELUE (Liu et al., 2021a) is a benchmark which evaluates the Pareto Front of early exit models on the FLOPs-performance plane. They provide a BERT-like baseline with jointly pretrained internal classifiers, to mitigate the gap between pretraining and fine-tuning.

5 Challenges and Future Directions

Evaluation Evaluating dynamic neural networks can be difficult since we cannot pre-define a few break points to compare different methods at the exact same amount of computation or time. ELUE score (Liu et al., 2021a) may be a promising solution to this problem by considering both computation and performance, depicting the Pareto Front. Besides, different works have different calculation for speed-up ratio. For example, some works use the ratio of layers involved in computation to estimate speed-up ratio (Zhou et al., 2020a; Sun et al., 2021; Liao et al., 2021). This can be misleading since internal classifiers introduce extra computational costs, especially when more complicated mechanism introduced, e.g., future-layer imitation (Liao et al., 2021). Also, the reported speed of MoE models, greatly differs on different hardware and distribution settings, making it hard to compare across papers.

Data Parallelism One drawback of dynamic neural networks is their inefficiency on data parallelism. To be specific, MoE methods introduce extra communication costs for dynamic routing and could be a bottleneck for efficiency. Skimming and early exit methods often employ an "online inference" setting where the batch size is fixed to 1, to achieve maximum acceleration. However, for batched inference, the efficiency of these methods will drastically degrade, since the already-exited instances will have to wait all instances to exit, which causes a low parallelism and low utilization of GPU.

Optimized Runtime Since dynamic neural networks are an emerging type of neural networks, most hardware and libraries are not well-optimized for these models. For example, sparse matrix multiplication in MoE needs specialized hardware and software support to achieve its theoretical efficiency. Also, current dynamic neural networks are often implemented in eager execution, which prevents them from low-level optimization of graph execution. There have been works exploring optimized runtime for MoE (Shazeer et al., 2018; Jia et al., 2020; He et al., 2021; Rajbhandari et al., 2022) and early exit (Paul et al., 2019) while more to be done in the future.

Theoretical Analysis and Support While the dynamic neural networks have demonstrated empirical improvement over static counterparts, dynamic networks are not solidly backed by theoretical analysis. For example, the theoretical analysis in PABEE (Zhou et al., 2020a) is based on an assumption that internal classifiers are independent to each other, which is unrealistic. More research should be done from the perspective of optimization and effect of data distribution on dynamic neural networks.

Explainability The decision-making process of the dynamic neural networks can be important to explain the model prediction and even understand

more fundamental research questions in machine learning, including scaling law and generalization. Can we use skimming to explain sequence classification? Is it consistent with attention-based explanation (Xu et al., 2015)? What does each expert in MoE learn and what makes them different? Why does a lower internal classifier make different prediction from an upper classifier despite equally trained with the same objective? These questions warrant further exploration, from both data and model perspectives.

Limitations

A limitation of this survey is that we do not draw a direct quantitative comparison for the methods surveyed in this paper since different methods have their own accuracy-speed curves, with their own unique limitations (e.g., many early exit methods can only handle a batch size of 1). Also, we do not discuss some works in depth and in detail due to space limit.

References

Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, et al. 2021. Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*.

Dimitri P. Bertsekas. 1992. Auction algorithms for network flow problems: A tutorial introduction. *Comput. Optim. Appl.*, 1(1):7–66.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Víctor Campos, Brendan Jou, Xavier Giró-i-Nieto, Jordi Torres, and Shih-Fu Chang. 2018. Skip RNN: learning to skip state updates in recurrent neural networks. In *ICLR*. OpenReview.net.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. In *ICLR*. OpenReview.net.

Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. 2022. Unified scaling laws for routed language models. *arXiv preprint arXiv:2202.01169*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186. Association for Computational Linguistics.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. 2021. Glam: Efficient scaling of language models with mixture-of-experts. *arXiv preprint arXiv:2112.06905*.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *ICLR*. OpenReview.net.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*.

Tsu-Jui Fu and Wei-Yun Ma. 2018. Speed reading: Learning to read forbackward via shuttle. In *EMNLP*, pages 4439–4448. Association for Computational Linguistics.

Shijie Geng, Peng Gao, Zuohui Fu, and Yongfeng Zhang. 2021. Romebert: Robust training of multi-exit bert. *arXiv preprint arXiv:2101.09755*.

Behrooz Ghorbani, Orhan Firat, Markus Freitag, Ankur Bapna, Maxim Krikun, Xavier Garcia, Ciprian Chelba, and Colin Cherry. 2021. Scaling laws for neural machine translation. *arXiv preprint arXiv:2109.07740*.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, et al. 2020. Power-bert: Accelerating BERT inference via progressive word-vector elimination. In *ICML*.

Yue Guan, Zhengyi Li, Jingwen Leng, et al. 2022. Transkimmer: Transformer learns to layer-wise skim. In *ACL*.

Christian Hansen, Casper Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. 2019. Neural speed reading with structural-jump-lstm. In *ICLR*. OpenReview.net.

Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. 2021. Fastmoe: A fast mixture-of-expert training system. *arXiv preprint arXiv:2103.13262*.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Yacine Jernite, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2017. Variable computation in recurrent neural networks. In *ICLR*. OpenReview.net.

Xianyan Jia, Le Jiang, Ang Wang, Jie Zhang, Xinyuan Li, Wencong Xiao, Yong Li, Zhen Zheng, Xiaoyong Liu, Wei Lin, et al. 2020. Whale: Scaling deep learning model training to the trillions. *arXiv preprint arXiv:2011.09208*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 3301–3310. PMLR.

Nan Rosemary Ke, Konrad Zolna, Alessandro Sordoni, Zhouhan Lin, Adam Trischler, Yoshua Bengio, Joelle Pineau, Laurent Charlin, and Christopher J. Pal. 2018. Focused hierarchical rnns for conditional sequence processing. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 2559–2568. PMLR.

Gyuwan Kim and Kyunghyun Cho. 2021. Length-adaptive transformer: Train once with length drop, use anytime with search. In *ACL-IJCNLP*.

Sehoon Kim, Sheng Shen, David Thorsley, et al. 2022. Learned token pruning for transformers. In *KDD*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*. OpenReview.net.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. Gshard: Scaling giant models with conditional computation and automatic sharding. In *ICLR*. OpenReview.net.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. BASE layers: Simplifying training of large, sparse models. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 6265–6274. PMLR.

Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. 2021a. Cascadebert: Accelerating inference of pre-trained language models via calibrated complete models cascade. In *EMNLP (Findings)*, pages 475–486. Association for Computational Linguistics.

Xiangsheng Li, Jiaxin Mao, Chao Wang, Yiqun Liu, Min Zhang, and Shaoping Ma. 2019. Teach machine how to read: Reading behavior inspired relevance estimation. In *SIGIR*, pages 795–804. ACM.

Xiaonan Li, Yunfan Shao, Tianxiang Sun, Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2021b. Accelerating BERT inference for sequence labeling via early-exit. In *ACL-IJCNLP*, pages 189–199. Association for Computational Linguistics.

Kaiyuan Liao, Yi Zhang, Xuancheng Ren, Qi Su, Xu Sun, and Bin He. 2021. A global past-future early exit method for accelerating inference of pre-trained language models. In *NAACL-HLT*, pages 2013–2023. Association for Computational Linguistics.

Junyang Lin, Rui Men, An Yang, Chang Zhou, Ming Ding, Yichang Zhang, Peng Wang, Ang Wang, Le Jiang, Xianyan Jia, et al. 2021. M6: A chinese multimodal pretrainer. *arXiv preprint arXiv:2103.00823*.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling BERT with adaptive inference time. In *ACL*, pages 6035–6044. Association for Computational Linguistics.

Xianggen Liu, Lili Mou, Haotian Cui, Zhengdong Lu, and Sen Song. 2018. Jumper: Learning when to make classification decisions in reading. *arXiv preprint arXiv:1807.02314*.

Xiangyang Liu, Tianxiang Sun, Junliang He, Lingling Wu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. 2021a. Towards efficient nlp: A standard evaluation and a strong baseline. *arXiv preprint arXiv:2110.07038*.

Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. 2021b. Faster depth-adaptive transformers. In *AAAI*, pages 13424–13432. AAAI Press.

Xiaonan Nie, Shijie Cao, Xupeng Miao, Lingxiao Ma, Jilong Xue, Youshan Miao, Zichao Yang, Zhi Yang, and Bin Cui. 2021. Dense-to-sparse gate for mixture-of-experts. *arXiv preprint arXiv:2112.14397*.

Eunhyeok Park, Dongyoung Kim, Soobeom Kim, Yong-Deok Kim, Gunhee Kim, Sungroh Yoon, and Sungjoo Yoo. 2015. Big/little deep neural network for ultra low power inference. In *CODES+ISSS*, pages 124–132. IEEE.

Debdeep Paul, Jawar Singh, and Jimson Mathew. 2019. Hardware-software co-design approach for deep learning inference. In *ICSCC*, pages 1–5. IEEE.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. *arXiv preprint arXiv:2201.05596*.

Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. 2021. Hash layers for large sparse models. In *NeurIPS*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Tal Schuster, Adam Fisch, Tommi Jaakkola, and Regina Barzilay. 2021. Consistent accelerated inference via confident adaptive transformers. *arXiv preprint arXiv:2104.08803*.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *ACL*, pages 6640–6651. Association for Computational Linguistics.

Min Joon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Neural speed reading via skim-rnn. In *ICLR*. OpenReview.net.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake A. Hechtman. 2018. Mesh-tensorflow: Deep learning for supercomputers. In *NeurIPS*, pages 10435–10444.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*. OpenReview.net.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. Reasonet: Learning to stop reading in machine comprehension. In *KDD*, pages 1047–1055. ACM.

Tianxiang Sun, Yunhua Zhou, Xiangyang Liu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. 2021. Early exiting with ensemble internal classifiers. *arXiv preprint arXiv:2105.13792*.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *ICPR*, pages 2464–2469. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, pages 5998–6008.

Jue Wang, Ke Chen, Gang Chen, et al. 2022. Skipbert: Efficient inference with shallow layer skipping. In *ACL*.

Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. 2020a. Early exiting bert for efficient document ranking. In *SustaiNLP*, pages 83–88.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020b. Deebert: Dynamic early exiting for accelerating BERT inference. In *ACL*, pages 2246–2251. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. Berxit: Early exiting for BERT with better fine-tuning and extension to regression. In *EACL*, pages 91–104. Association for Computational Linguistics.

Canwen Xu and Julian McAuley. 2022. A survey on model compression and acceleration for pretrained language models. *arXiv preprint arXiv:2202.07105*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. Bert-of-theseus: Compressing BERT by progressive module replacing. In *EMNLP*, pages 7859–7869. Association for Computational Linguistics.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Ke Xu, Julian J. McAuley, and Furu Wei. 2021. Beyond preserved accuracy: Evaluating loyalty and robustness of BERT compression. In *EMNLP*, pages 10653–10659. Association for Computational Linguistics.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2048–2057. JMLR.org.

An Yang, Junyang Lin, Rui Men, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Jiamang Wang, Yong Li, et al. 2021. M6-t: Exploring sparse expert models and beyond. *arXiv preprint arXiv:2105.15082*.

Deming Ye, Yankai Lin, Yufei Huang, and Maosong Sun. 2021. TR-BERT: dynamic token reduction for accelerating BERT inference. In *NAACL-HLT*.

Adams Wei Yu, Hongrae Lee, and Quoc V. Le. 2017. Learning to skim text. In *ACL*, pages 1880–1890. Association for Computational Linguistics.

Keyi Yu, Yang Liu, Alexander G. Schwing, and Jian Peng. 2018. Fast and accurate text classification: Skimming, rereading and early stopping. In *ICLR (Workshop)*. OpenReview.net.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*.

Zhen Zhang, Wei Zhu, Jinfan Zhang, et al. 2022a. PCEE-BERT: accelerating BERT inference via patient and confident early exiting. In *NAACL-HLT (Findings)*.

Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, Yanzheng Cai, et al. 2022b. Cpm-2: Large-scale cost-effective pre-trained language models. *AI Open*.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2021. Moefication: Conditional computation of transformer models for efficient inference. *arXiv preprint arXiv:2110.01786*.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian J. McAuley, Ke Xu, and Furu Wei. 2020a. BERT loses patience: Fast and robust inference with early exit. In *NeurIPS*.

Xuhui Zhou, Yue Zhang, Leyang Cui, and Dandan Huang. 2020b. Evaluating commonsense in pre-trained language models. In *AAAI*, pages 9733–9740. AAAI Press.

Wei Zhu. 2021. Leebert: Learned early exit for BERT with cross-level optimization. In *ACL-IJCNLP*, pages 2968–2980. Association for Computational Linguistics.

Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Jianfeng Gao, and Tuo Zhao. 2022. Taming sparsely activated transformer with stochastic experts. In *ICLR*.