BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 66, No. 6, 2018
DOI: 10.24425/bpas.2018.125928

Probabilistic adaptive computation time

M. FIGURNOV[1]*, A. SOBOLEV[2], and D. VETROV[1]

[1]National Research University Higher School of Economics, Moscow, Russia
[2]Luka Inc., Moscow, Russia

Abstract. We present a probabilistic model with discrete latent variables that control the computation time in deep learning models such as ResNets and LSTMs. A prior on the latent variables expresses the preference for faster computation. The amount of computation for an input is determined via amortized maximum a posteriori (MAP) inference. MAP inference is performed using a novel stochastic variational optimization method. The recently proposed adaptive computation time mechanism can be seen as an ad-hoc relaxation of this model. We demonstrate training using the general-purpose concrete relaxation of discrete variables. Evaluation on ResNet shows that our method matches the speed-accuracy trade-off of adaptive computation time, while allowing for evaluation with a simple deterministic procedure that has a lower memory footprint.

Key words: deep learning, probabilistic models, adaptive computation time.

1. Introduction

In the past years, deep learning models have become significantly deeper and more computationally expensive. As evident from the ImageNet competition results [1–4], increasing the depth of the models indeed leads to improved results. However, such expensive models are suitable not for many applications, including deployment on low-power devices and real-time data processing. Thus, acceleration of deep learning models becomes an important area of research. The acceleration methods can be broadly divided into static and dynamic methods. The static methods, such as factorization of weight matrices [5] and convolutional kernels [6], as well as sparsification [7], reduce the computation equally for all input objects. Some of these methods are now an integral part of the modern deep networks. For example, Residual Networks [4, 8] use factorized convolutional kernels. In this paper, we focus on the dynamic methods that vary the amount of computation depending on the input object [9–11] (or even across spatial regions of objects [12]). This allows to allocate less computation for easier objects and therefore improve the computational efficiency of the deep architectures. The dynamic methods are naturally connected to discrete latent variable models [13]: the (discrete) amount of computation can be considered as a latent variable. The standard approach to training discrete latent variable models is REINFORCE [14]. However, it suffers from a large variance of gradients, making training of complex models problematic. Thus, the methods trained with REINFORCE usually have few latent variables, such as the number of glimpses over an image [15], or the number of objects in a scene [10].

Adaptive computation time (ACT) [16] is a recently proposed mechanism that adjusts the computational depth of deep models: the harder the object is, the more iterations it is processed for. This mechanism does not rely on REINFORCE and thus has low variance of gradients, is end-to-end trainable, problem-agnostic and does not require an explicit supervision for the number of computational iterations. It has been applied to recurrent networks for the problems of text modelling [16], reasoning [17] and early sequence classification [18]. Spatially adaptive computation time (SACT) [12] applies the ACT mechanism to the spatial positions of residual networks [8], a popular convolutional neural network model. This results in computational savings and interpretable computation time maps that highlight the regions of the image that the network considers relevant to the task at hand.

In this paper, we introduce probabilistic adaptive computation time (PACT), a probabilistic model with discrete latent variables that specify the number of iterations to execute. We define a prior on the latent variables that encodes the desired trade-off between speed and accuracy. Then, we perform amortized maximum a posteriori (MAP) inference to find the proper amount of computation for a given object. The ACT mechanism can be seen as an ad-hoc relaxation of the PACT model with a specific prior distribution. A significant downside of the ACT relaxation is that it provides a discontinuous objective. Since reparameterization trick is only valid for continuous objectives, ACT cannot be incorporated into stochastic models trained with reparameterization, such as variational autoencoder [19].

We extend variational optimization [20, 21], a method for MAP inference, to handle intractable expectations using REINFORCE or reparameterization trick. For discrete latent variables, we propose to apply the concrete relaxation [22, 23] and then perform the reparameterization. We call the obtained method stochastic variational optimization and apply it to the PACT model. Evaluation on ResNets shows that training using

*e-mail: michael@figurnov.ru

Manuscript submitted 2018-05-15, revised 2018-06-11, initially accepted for publication 2018-06-12, published in December 2018.

the relaxation outperforms the REINFORCE based method and matches the performance of the heuristic ACT. We show that the relaxation allows to train the model with up to 1344 discrete latent variables. Additionally, the models trained with the proposed relaxation can be evaluated with a simple deterministic approach that reduces the memory consumption, compared to ACT. Evaluation of the ACT models in the same manner decreases the performance.

2. Background

Notation. Let $\mathbb{E}_{q(z)} f(z)$ be the expectation of a function $f(z)$ over a probability distribution $q(z)$, $\sigma(z) = \frac{1}{1+\exp(-z)}$ the sigmoid function, $\sigma^{-1}(z) = \log \frac{1}{1-z}$ the logit function, $[\text{cond}]$ the step-function that is equal to 1 if cond is true and 0 otherwise. Also, let $z_{<k}$ be a shorthand notation for $z_1, \ldots, z_{k-1}$.

2.1. Variational optimization [20, 21] is a method for maximization of a function $f(z)$ of an argument z. This argument can be either continuous or discrete. To apply variational optimization, we choose an auxiliary parametric probability distribution over the arguments values $q_\phi(z)$. The following lower bound on the optimal value holds for any distribution $q_\phi(z)$:

$$L(\phi) = \mathop{\mathbb{E}}_{q_\phi(z)} f(z) \leq \mathop{\mathbb{E}}_{q_\phi(z)} \max_z f(z) = \max_z f(z). \tag{1}$$

Suppose that the parametric family of distributions $q_\phi(z)$ can model arbitrary delta-functions. Then the bound is tight and the optimum is achieved when $q_\phi(z) = \delta(z - z^*)$, where $f(z^*) = \max_z f(z)$.

Let us assume that the density $q_\phi(z)$ is a smooth function of ϕ. Then, $L(\phi)$ is a smooth function. Variational optimization further assumes that the expectation in $L(\phi)$ is tractable and maximizes $L(\phi)$ with a gradient-based method. However, it is not applicable when the expectation is intractable. We address this limitation in sec. 3.

2.2. Variational Optimization for Probabilistic Models. Consider a discriminative probabilistic model with latent variables $p(y, z|x) = p(y|x, z)p(z)$, where x is the object, y is the target label and z is the latent variable. The prior $p(z)$ encodes our preference for the values of z. The maximum a posteriori (MAP) inference problem is to find z^* that maximizes the density of the posterior distribution $p(z|x, y) = \frac{p(y, z|x)}{p(y|x)}$. During training time, we know both x and y, while during testing time we only have x and would like to find the distribution y. Therefore, we search for z^* in a parametric form that only depends on x, so that we can use it during the test time. This can be achieved by performing variational optimization with an auxiliary distribution $q_\phi(z|x)$:

$$L_{\text{MAP}}(\phi) = \mathop{\mathbb{E}}_{q_\phi(z|x)} \big(\log p(y|x, z) + \log p(z)\big). \tag{2}$$

For training, we plug in the ground-truth label y and optimize $L_{\text{MAP}}(\phi)$. During testing, we sample $z \sim q_\phi(z|x)$ and obtain the distribution over the labels $p(y|x, z)$.

Let us analyze a special case of this approach that has been extensively used in attention models literature [15, 24–27]. Consider a probabilistic model $p_\phi(y, z|x) = p(y|x, z)p_\phi(z|x)$ with a learnable prior. We can use the prior $p_\phi(z|x)$ as the approximate posterior in variational inference. The corresponding evidence lower bound is

$$L_{\text{ML}}(\phi) = \mathop{\mathbb{E}}_{p_\phi(z|x)} \log p(y|x, z) \leq \log p_\phi(y|x). \tag{3}$$

Renaming $p_\phi(z|x)$ into $q_\phi(z|x)$, we recognize the objective (2), where the prior distribution is uniform, $p(z) \propto 1$ (for a continuous latent variable on unbounded domain, this prior is improper). Applying the inequality (1), we have $L_{\text{ML}}(\phi) \leq \max_z \log p(y|x, z)$. Therefore, optimization of $L_{\text{ML}}(\phi)$ corresponds to maximum likelihood inference of the latent variables. On the other hand, the bound (2) allows to incorporate an explicit prior distribution over the latent variables and perform MAP inference. This is a crucial requirement for the models such as the one proposed in the paper that have a non-uniform prior distribution.

The objective (2) can also be seen as evidence lower bound on the marginal likelihood without the entropy term. Indeed, adding the entropy of $q_\phi(z|x)$ to the eqn. (2) yields

$$\mathop{\mathbb{E}}_{q_\phi(z|x)} \log \frac{p(y|x, z)p(z)}{q_\phi(z|x)} \leq \log p(y|x). \tag{4}$$

Unlike MAP inference, variational inference provides a distribution over the latent variable. In our case, this is undesirable since we are interested in the single "best" value for the latent variables at the test time. To obtain a single value of the variables for evaluation, we could choose a maximum of the approximate posterior. However, this would introduce a gap between the train- and test-time behavior of the model.

2.3. Concrete Distribution and Reparametrization. Suppose that we would like to stochastically optimize parameters ϕ of an intractable expectation $\mathbb{E}_{q_\phi(z)} f(z)$, where $f(z)$ is smooth. The *reparametrization trick* [18, 28] allows for this, provided that the distribution $q_\phi(z)$ can be reparametrized, *i.e.*, we can sample $z \sim q_\phi(z)$ as follows:

$$\varepsilon \sim q(\varepsilon), \; z = g(\varepsilon, \phi), \tag{5}$$

where $g(\varepsilon, \phi)$ is smooth w.r.t. ε and ϕ. Then, by applying the chain rule we have:

$$\nabla_\phi \mathop{\mathbb{E}}_{q_\phi(z)} f(z) = \mathop{\mathbb{E}}_{q(\varepsilon)} f'\big(g(\varepsilon, \phi)\big) \nabla_\phi g(\varepsilon, \phi). \tag{6}$$

This expectation can be approximated using Monte-Carlo sampling. The reparameterization trick is most commonly used for Normal distribution. If $z \sim \text{Normal}(\mu, \sigma^2)$, then $q(\varepsilon) = \text{Normal}(0, 1)$ and $g(\varepsilon, \phi) = \mu + \varepsilon\sigma$.

Unfortunately, the reparameterization trick cannot be directly applied to discrete random variables, since the corresponding function $g(\varepsilon, \phi)$ is a non-smooth step function. However, it is possible to *relax* a discrete random variable so that the relaxation becomes reparameterizable.

The Concrete distribution [22, 23] is a continuous reparameterizable relaxation of a discrete random variable. For the purposes of this paper, we only consider relaxation of Bernoulli (binary) discrete random variables. Consider a random variable $v \sim \text{Bernoulli}(\gamma)$, where $p(v=1)=\gamma \in (0, 1)$. We introduce a temperature parameter $\lambda > 0$. The relaxed random variable $\hat{v} \sim \text{RelaxedBernoulli}(\gamma; \lambda)$ is defined via the following sampling procedure:

$$\varepsilon \sim \text{Uniform}(0, 1),\ l = \sigma^{-1}(\gamma) + \sigma^{-1}(\varepsilon),\ \hat{v} = \sigma\left(\frac{l}{\lambda}\right) \quad (7)$$

The RelaxedBernoulli distribution has several useful properties [22]. First, the probability to be greater than 0.5 is equal for Bernoulli and RelaxedBernoulli random variables. However, the mean value of RelaxedBernoulli is, in general, *not* equal to γ. For $\lambda \to 0$, the distribution of $\hat{v}$ approaches $\text{Bernoulli}(\gamma)$. Next, for $\lambda \leq 1$ the density $p(\hat{v})$ does not have modes in the interior of the $(0, 1)$ range. As a result, the samples are typically close to either zero or one, which makes the relaxation work well for our purposes. Importantly for us, when $\gamma \to 0$ or $\gamma \to 1$, the distribution of RelaxedBernoulli approaches a delta-function at 0 or 1, respectively. This means that for extreme values of probability, the gap between the relaxed and non-relaxed distributions vanishes, regardless of the temperature λ.

3. Stochastic Variational Optimization

Consider the variational optimization objective $L(\phi) = \mathbb{E}_{q_\phi(z)} f(z)$, where z is a latent variable. Stochastic variational optimization estimates the gradient $\nabla_\phi L(\phi)$ stochastically, even when the expectation is intractable. First, we consider the case of a reparameterizable distribution, and then cover the case of discrete distributions.

If the distribution $q_\phi(z)$ is reparameterizable, *e.g.*, is a Normal distribution, we can perform reparameterization trick and calculate the stochastic gradients directly. We then apply stochastic gradient optimization methods, resulting in stochastic variational optimization of the objective.

Now, we switch to the case where z is discrete. One popular method for this type of problems is the REINFORCE [14] training rule

$$\nabla_\phi L(\phi) = \mathop{\mathbb{E}}_{q_\phi(z)} \big(f(z) - c\big) \nabla_\phi \log q_\phi(z), \quad (8)$$

where c is a scalar baseline. The expectation can be approximated by Monte-Carlo sampling. Although this procedure provides unbiased gradients, the estimate often has an impractically high variance.

We propose to apply the Concrete relaxation to the distribution $q_\phi(z)$ and then use the reparameterization trick. This results in lower-variance gradients at the cost of a bias. Assume that $z \in \{0, 1\}^d$. Let's decompose the proposal distribution using the chain rule, $q_\phi(z) = \prod_{i=1}^d q_\phi(z_i|z_{<i})$ (this sidesteps enumeration of all the 2^d configurations of z during sampling). We make two assumptions: (1) $f(z)$ is defined and smooth for $z \in \{0, 1\}^d$; (2) each factor $q_\phi(z_i|z_{<i})$, $i > 1$ is defined and smooth for $z_{<i} \in [0, 1]^{i-1}$. Then, we can apply the Concrete relaxation with temperature $\lambda > 0$ to each factor (the hat denotes relaxation):

$$q_{\phi,\lambda}(\hat{z}) = \prod_{i=1}^d q_{\phi,\lambda}(\hat{z}_i|\hat{z}_{<i}). \quad (9)$$

The relaxed objective has the form

$$\hat{L}_\lambda(\phi) = \mathop{\mathbb{E}}_{q_{\phi,\lambda}(\hat{z})} f(\hat{z}). \quad (10)$$

This objective can now be stochastically optimized using the reparameterization trick.

If all the probabilities in the relaxed distribution approach extreme values (0 or 1), the relaxed distribution approaches the non-relaxed one, for any temperature λ. In this case, the value of the relaxed objective $\hat{L}_\lambda(\phi)$ approaches the value of the original objective $L(\phi)$.

4. Probabilistic Adaptive Computation Time

First, we introduce the adaptive computation block. It is a computation module that chooses the number of iterations depending on the input. Depending on the specific type of the latent variables, we obtain a discrete, thresholded or relaxed block. Importantly, the blocks are compatible in the sense that one can train a model with one type of block and then switch to another during evaluation. Then, we present a probabilistic model that incorporates the number of iterations as a latent variable into a discriminative model. The prior on the latent variable favors using less iterations. Finally, we perform MAP inference over the number of iterations via stochastic variational optimization.

Discrete adaptive computation block (Algorithm 1) performs $z \in \{1, \ldots, L\}$ iterations of computation, where z is a discrete latent variable. Let us assume that the l-th iteration outputs

Algorithm 1 Discrete adaptive computation block.

Input: maximum number of iterations L
Output: output of the block
Output: number of executed iterations z

for $l = 1 \ldots L$ **do**
  Compute u^l
  if $l < L$ **then** $h = H^l(u^l)$
  else $h = 1$
  end if
  $\xi \sim \text{Bernoulli}(h)$
  if $\xi =$ **true then**
    output $= u^l$
    $z = l$
    return output, z
  end if
end for

a value u^l (we use superscripts to index the iterations in a block), and that all $u^1, \ldots, u^L$ have the same dimensions. The output of the block is u^z, the output of the z-th iteration. To perform optimization over the discrete latent variable z, we introduce a distribution $q_\phi(z)$ with parameters ϕ. Denote $z^l = [z = l]$ the \textit{halting unit} of the block: when it is equal to one, the computation is halted. The two desiderata for $q_\phi(z)$ are: (1) the probability of halting at the l-th step should depend on u^l; (2) it should be possible to sample z^l after only executing the first l iterations.

To satisfy the first property, we introduce a *halting probability* for every iteration:

$$h^l = H^l_\phi\left(u^l\right),\ l = 1, \ldots, (L-1),\ h^L = 1. \tag{11}$$

For the second property, we define the following sampling procedure for the distribution $q_\phi(z)$:

$$\xi^l \sim \text{Bernoulli}\left(h^l\right),\ l = 1, \ldots, L-1,\ \xi^L = 1, \tag{12}$$

$$z^l = \xi^l \prod_{i=1}^{l-1} \left(1 - \xi^i\right),\ l = 1, \ldots, L. \tag{13}$$

The vector $\left(z^1, \ldots, z^L\right)$ is a one-hot representation of the discrete L-ary latent variable z. We reparameterize z via $(L-1)$ Bernoulli latent variables $\left(\xi^1, \ldots, \xi^{L-1}\right)$. The distribution of z can be obtained by taking an expectation over the independent random variables ξ^l:

$$q_\phi\left(z^l = 1\right) = q_\phi\left(z = l\right) = h^l \prod_{i=1}^{l-1} \left(1 - h^i\right). \tag{14}$$

Thresholded adaptive computation block (Algorithm 2) is a deterministic version of the (stochastic) discrete adaptive computation block. Since we perform MAP inference over the latent variables, we expect the halting probabilities h^l to be sufficiently close to either zero or one. Therefore, during evaluation we can replace sampling (12) with thresholding of the halting probabilities:

$$\xi^l = \left[h^l > 0.5\right]. \tag{15}$$

Algorithm 2 Thresholded adaptive computation block.

Input: maximum number of iterations L
Output: output of the block
Output: number of executed iterations z

for $l = 1 \ldots L$ **do**
  Compute u^l
  if $l < L$ **then** $h = H^l(u^l)$
  else $h = 1$
  end if
  if $h > 0.5$ **then**
    output $= u^l$
    $z = l$
    return output, z
  end if
end for

The advantage of this block is an extremely simple implementation: we stop as soon as the halting probability exceeds 0.5.

Relaxed adaptive computation block (Algorithm 3) is obtained from the discrete adaptive computation block by replacing the Bernoulli random variables with RelaxedBernoulli. We denote the relaxed variables with a hat and define the temperature of the relaxation $\lambda > 0$. Sampling the vector $\hat{z} = \left(\hat{z}^1, \ldots, \hat{z}^L\right)$ from $q_{\phi,\lambda}(\hat{z})$ proceeds as follows:

$$\hat{\xi}^l \sim \text{RelaxedBernoulli}\left(h^l, \lambda\right),\ l = 1 \ldots L-1, \tag{16}$$

$$\hat{\xi}^L = 1,\ \hat{z}^l = \hat{\xi}^l \prod_{i=1}^{l-1} \left(1 - \hat{\xi}^i\right),\ l = 1 \ldots L. \tag{17}$$

Algorithm 3 Relaxed adaptive computation block.

Input: maximum number of iterations L
Input: temperature of relaxation λ
Output: output of the block
Output: expected number of iterations N

$S_{\hat{\xi}} = 1$ ▷ Remaining stick length for $\hat{\xi}$
$S_h = 1$ ▷ Remaining stick length for h
$N = 0$
$\widehat{\text{output}} = 0$
for $l = 1 \ldots L$ **do**
  Compute u^l
  if $l < L$ **then** $h = H^l(u^l)$
  else $h = 1$
  end if
  $\hat{\xi} \sim \text{RelaxedBernoulli}(h; \lambda)$
  $\hat{z} = S_{\hat{\xi}} \cdot \hat{\xi}$
  $\widehat{\text{output}} = \widehat{\text{output}} + \hat{z} \cdot u^l$
  $N = N + l \cdot S_h \cdot h$
  $S_{\hat{\xi}} = S_{\hat{\xi}}(1 - \hat{\xi})$
  $S_h = S_h(1 - h)$
end for
return output, N

The vector $\hat{z}$ is no longer one-hot. However, since it is produced by a stick-breaking procedure, it forms a discrete probability distribution over the iterations that we call the *halting distribution*. Finally, we define the output of the relaxed adaptive computation block as an expectation of the iteration outputs w.r.t. the halting distribution $\hat{z}$:

$$\widehat{\text{output}} = \sum_{l=1}^{L} \hat{z}^l u^l. \tag{18}$$

The whole procedure is illustrated on Fig. 1.

Probabilistic model. Consider a discriminative model with a likelihood $p_\theta(y|x)$ of the target label y given an object x (for simplicity of notation, we consider just one object), parameterized by θ. This model can be a deep network for classification or regression problem. In many cases we prefer that the model make the prediction as quickly as possible. Assume that we have incorporated K adaptive computation blocks into the

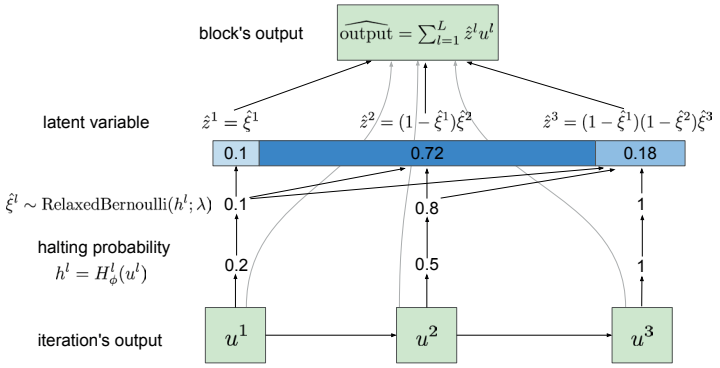

Fig. 1. Relaxed adaptive computation block

likelihood with the corresponding latent variables (number of computation iterations) $\mathbf{z} = (z_1, \ldots, z_K)$. Also, denote the maximum number of iterations in the k-th block as L_k.

We now discuss the prior distribution $p(\mathbf{z})$ that encodes the preference for less iterations. For simplicity, we assume that it factorizes over the blocks, $p(\mathbf{z}) = \prod_{k=1}^{K} p(z_k)$. The prior for each block $p(z_k)$ is a discrete distribution over L_k iterations. To make our model directly comparable to ACT, we choose a prior distribution that provides the same log-linear penalty as the ACT model (up to a normalization constant), a truncated Geometric distribution. We parameterize the Geometric distribution via a log-scale number of iterations penalty $\tau_k > 0$ (the canonical Geometric distribution's probability for success α_k can be recovered as $\alpha_k = 1 - \exp(-\tau_k)$). The prior distribution for a single block is

$$\begin{aligned}&\text{TruncatedGeometric}(z_k|\tau_k, L_k) = \\ &= \frac{\exp(\tau_k) - 1}{1 - \exp(-\tau_k L_k)} \exp(-\tau_k z_k),\ z_k \in \{1, \ldots, L_k\}.\end{aligned} \tag{19}$$

Using the described prior, we obtain the following probabilistic model:

$$\begin{aligned}&p_\theta(y, \mathbf{z}|x)p(\mathbf{z}),\ p_\theta(y|x, \mathbf{z})p(\mathbf{z}), \\ &p(\mathbf{z}) = \prod_{k=1}^{K} \text{TruncatedGeometric}(z_k|\tau_k, L_k) = \\ &= \left(\prod_{k=1}^{K} \frac{\exp(\tau_k) - 1}{1 - \exp(-\tau_k L_k)}\right) \exp\left(-\sum_{l=1}^{L} \tau_k z_k\right).\end{aligned} \tag{20}$$

We perform MAP inference of the latent variable z via variational optimization with an auxiliary distribution

$$q_\phi(\mathbf{z}|x) = \prod_{k=1}^{K} q_\phi(z_k|z_{<k}, x), \tag{21}$$

where $q_\phi(z_k|z_{<k}, x)$ is defined via eqn. (13). The dependence on the input and the previous latent variables is via the inputs of the block. We refer to this probabilistic model as *discrete*. The objective for maximization w.r.t. θ and ϕ is

$$\begin{aligned}L(\theta, \phi) &= \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}|x)} \log p_\theta(y, \mathbf{z}|x) = \\ &= \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}|x)} \left(\log p_\theta(y|\mathbf{z}, x) + \sum_{k=1}^{K} \log p(z_k)\right).\end{aligned} \tag{22}$$

To reduce the variance of the stochastic estimate of the objective, we analytically compute the expectation of the log-prior:

$$\begin{aligned}&\mathop{\mathbb{E}}_{q_\phi(\mathbf{z}|x)} \log p(z_k) = \\ &= -\tau_k \mathop{\mathbb{E}}_{q_\phi(z_{<k}|x)} \underbrace{\sum_{l=1}^{L} l\, h_k^l \prod_{i=1}^{l-1} \left(1 - h_k^i\right)}_{N_k} + \text{const}.\end{aligned} \tag{23}$$

Here N_k is the expected number of iterations in the k-th block. Ignoring the additive constant, we have

$$L(\theta, \phi) = \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}|x)} \left(\log p_\theta(y|\mathbf{z}, x) - \sum_{k=1}^{K} \tau_k z_k\right). \tag{24}$$

The objective in eqn. (24) is intractable for deep models consisting of several stacked adaptive computation blocks, as the complexity of direct evaluation of the expectation grows exponentially in the number of blocks. One heuristic is to replace the random variables z_k with their expectations and optimize the probabilities directly. However, this simple approach fails for deep networks as they learn to trick the objective by increasing the halting probability for the first iterations and decreasing it for the latter iterations, while significantly boosting the magnitude of the outputs for the latter iterations [16]. The prior term value then reflects that few iterations are used, while the outputs of the blocks are dominated by the last iterations.

Instead, we stochastically optimize the objective (24). In sec. 3 we proposed two approaches to do this, one using REINFORCE and another using relaxation.

In the first approach, we directly apply REINFORCE to the objective (24), obtaining the following gradients w.r.t. ϕ:

$$\begin{aligned}\nabla_\phi L(\theta, \phi) = \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}|x)} \Bigg(&\left(\log p_\theta(y|\mathbf{z}, x) - c\right) \times \\ &\times \nabla_\phi \log q_\phi(\mathbf{z}|x) - \sum_{k=1}^{K} \tau_k \nabla_\phi N_k\Bigg),\end{aligned} \tag{25}$$

where c is a scalar baseline. The value $q_\phi(z|x)$ is defined by eqn. (14). Note that we have neglected the dependency of N_k on $z_{<k}$ to reduce the variance of the gradients.

For the second approach, we replace every adaptive computation block with a relaxed counterpart, and the corresponding distribution $q_\phi(z)$ with the relaxed distribution $q_{\phi,\lambda}(\hat{z})$. This *relaxed* model has an objective that can be optimized via the reparameterization trick:

$$\hat{L}_\lambda(\theta, \phi) = \mathop{\mathbb{E}}_{q_{\phi,\lambda}(\hat{\mathbf{z}}|x)} \left(\log p_\theta(y|\hat{\mathbf{z}}, x) - \sum_{k=1}^{K} \tau_k N_k\right). \tag{26}$$

4.1. Application: Probabilistic Spatially Adaptive Computation Time for Residual Networks. Residual network (ResNet) [4, 8] is a deep convolutional neural network architecture that has been successfully applied to many computer vision problems [29, 30]. We describe ResNet-32 and ResNet-110 models for CIFAR image classification dataset [31]. They contain three stacked *blocks*, each consisting of several *residual units* (5 for ResNet-32 and 18 for ResNet-110). The computational iteration of a ResNet is a residual unit of the form $F_k^l\left(u_k^{l-1}\right) = u_k^{l-1} + f_k^l\left(u_k^{l-1}\right)$, where f_k^l is a sub-network consisting of two convolutional layers. u_k^0 is the output of the previous block of residual units. The outputs of the residual units in each block have the same size. The first units in the second and third blocks are applied with stride 2 to perform spatial downsampling, while also increasing the number of output channels by a factor of two. Thus, the spatial dimensions of the first block are 32×32 (same as the size of CIFAR-10 images), the second block 16×16 and the third block 8×8. In this way, the amount of computation for every residual unit is roughly constant. The outputs of the last block are passed through a global average pooling and linear layers to obtain the class probabilities logits.

SACT [12] applies the ACT mechanism to every spatial position of every residual network block. Likewise, we apply an adaptive computation block to every spatial position of every residual network block. We call the obtained model *PSACT*, probabilistic spatially adaptive computation time. The corresponding latent variable is $z_{k,ij}$ where k is the number of residual network block and ij is the spatial position. The halting probability map is computed as $H_k^l(u) = \sigma\left(\tilde{W}_k^l * u + W_k^l \mathrm{pool}(u) + b_k^l\right)$, where $*$ is 3×3 convolution and pool is global average pooling. The computation time penalty for a block is chosen to be τ/HW, where τ is a global computation time penalty and H and W are the height and width of the ResNet block.

In order to impute the non-computed intermediate values, we redefine the residual unit as

$$F_k^l\left(u_k^{l-1}\right) = u_k^{l-1} + f_k^l\left(u_k^{l-1}\right) \cdot a\left(\xi_k^{<l}\right), \tag{27}$$

where $a\left(\xi_k^{<l}\right)$ is an *active positions mask*. For the discrete model, we choose $a\left(\xi_k^{<l}\right) = \prod_{t=1}^{l-1}\left(1 - \xi_k^l\right)$, with the operation performed element-wise. Thus, if the position is no longer evaluated (hence, $z_k < l$), the value is zero and we simply carry the features from the previous iteration. Otherwise, the value is one. For the relaxed model, we use $\hat{a}\left(\hat{\xi}_k^{<l}\right) = r \cdot [r > \delta]$, $r = \prod_{t=1}^{l-1}\left(1 - \xi_k^t\right)$, where $\delta > 0$ is a scalar hyperparameter. By clipping the values of r, we obtain strict zeros and can skip computing the corresponding values during the training time. We have verified that setting δ to zero gives similar results, although without a possibility of computation savings during training.

4.2. Application: Probabilistic Adaptive Computation Time for Recurrent Neural Networks. We can also apply the proposed model to dynamically vary the amount of computation in Recurrent Neural Networks, such as Long Short-Term Memory networks (LSTMs) [32]. Let us denote the input sequence $\mathbf{x} = (x_1, \ldots, x_T)$, where T is the number of timesteps. An adaptive computation block is associated with each timestep. Therefore, each timestep is processed for an adaptive number of iterations. We can use the same computation time penalty τ for all iterations. The computation iteration consists of applying the RNN's transition function to obtain the new state of the RNN: $u_k^l = F_\theta\left(x_k, [l = 1], u_k^{l-1}\right)$. Here u_k^0 is the output state from the previous block/timestep. The binary input feature $[l = 1]$ allows the network to detect the beginning of a new timestep. The halting probability is computed as $h_k^l = H_\phi\left(u_k^l\right) = \sigma\left(W u_k^l + b\right)$. The output state of a block is used as an input state for the next block and as features for predicting the emission values for the timestep.

5. Related work

Adaptive Computation Time (ACT) mechanism [16] can be seen as a heuristic deterministic relaxation of our PACT model. Specifically, ACT transforms the halting probabilities $\left(h^1, \ldots, h^L\right)$ into the halting distribution $\left(\hat{z}^1, \ldots, \hat{z}^L\right)$ as follows:

$$N = \min\left\{n \in \{1 \ldots L\} : \sum_{l=1}^{n} h^l \geq 0.99\right\}, \tag{28}$$

$$R = 1 - \sum_{l=1}^{N-1} h^l, \quad \hat{z}^l = \begin{cases} h^l & \text{if } l < N, \\ R & \text{if } l = N, \\ 0 & \text{if } l > N. \end{cases} \tag{29}$$

Since the halting distribution is not one-hot, additional memory is required to maintain the output $\sum_{l=1}^{L} \hat{z}^l u^l$ during evaluation. For completeness, we include the pseudocode in Algorithm 4. In

Algorithm 4 Adaptive computation block with Adaptive Computation Time relaxation.

Input: maximum number of iterations L
Input: $0 < \varepsilon < 1$ ▷ Recommended value: 0.01
Output: output of the block
Output: ponder cost ρ ▷ Upper bound on the number of executed iterations

$c = 0$ ▷ Cumulative halting probability
$R = 1$ ▷ Remainder
output $= 0$
$\rho = 0$
for $l = 1 \ldots L$ **do**
  Compute u^l
  if $l < L$ **then** $h = H^l(u^l)$
  else $h = 1$
  end if
  $c = c + h$
  $\rho = \rho + 1$
  if $c < 1 - \varepsilon$ **then**
    output $=$ output $+ h \cdot u^l$
    $R = R - h$
  else
    output $=$ output $+ R \cdot u^l$
    $\rho = \rho + R$
    break
  end if
end for
return output, ρ

discrete and thresholded PACT models, the halting distribution is one-hot and this memory can be saved.

The stopping time N has zero gradients almost everywhere. In order to optimize the stopping time, a differentiable upper bound, ponder cost, $\rho = N + R$ is introduced. Ponder cost is linear almost everywhere, but is a discontinuous function of the halting probabilities, with discontinuities arising in the configurations where N changes the value, see Fig. 2. For instance, this means that ACT cannot be used with reparameterization trick that is only valid for continuous objectives. The objective of ACT, for several adaptive computation blocks, is $\log p(y|\hat{\mathbf{z}}, x) - \sum_{k=1}^{K} \tau_k \rho_k$.

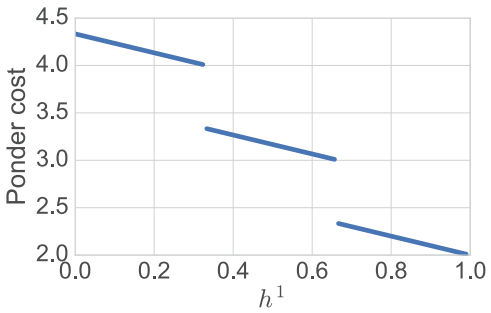

Fig. 2. Ponder cost ρ is a discontinuous function of the halting probability h^1. Here $h^2 = h^3 = h^4 = 1/3$

Let us summarize why the proposed PACT model is more principled than ACT. First, the discrete PACT model straightforwardly defines the halting time as the iteration where the halting unit is fired. On the other hand, ACT that uses an ad-hoc definition (28). Second, PACT allows to directly minimize the expected halting time, while ACT minimizes the discontinuous ponder cost.

Several recent papers propose models that are related to ACT. [33] updates a dynamically chosen subset of the hidden state of a recurrent network. [34] develops a model that "jumps" over regions of text, therefore allowing to skip the less informative pieces. [35] considers a recurrent model that can adaptively skip state updates. For training of the corresponding discrete latent variables, these models either use REINFORCE, or heuristic training methods such as straight-through [36]. [37] develops a hybrid of a recurrent and residual network: a single residual unit which is applied a dynamic number of iterations determined by ACT, thus obtaining a compact model with adaptive computation cost.

Concurrent works [38–40] propose to adaptively drop residual units in ResNet models. Each residual unit is equipped with a binary latent variable indicating whether to compute this particular unit. In this way, they drop any combination of residual units, while ACT can only drop the last units in each block. This can be thought of as an alternative probabilistic model to ACT that has greater flexibility. However, this model is ResNet-specific and, importantly, leads to a much more challenging discrete optimization problem: the solution space for each block now contains 2^L configurations instead of just L. To solve this problem, [38] employs Actor-Critic training method from the reinforcement learning literature and combines it with curriculum learning. However, the large variance of the gradients makes the training an order of magnitude slower than for the original ResNet. The other two works use biased gradient estimators with unclear theoretical properties: [39] employs the "hard" Gumbel-Softmax estimator [23] (which rounds the samples), while [40] pretrains using the straight-through estimator [36] and then fine-tunes using REINFORCE. Thus, these methods either suffer from large variance, or use unprincipled biased gradient estimators. In this paper, we propose a probabilistic view of ACT and SACT mechanisms and a principled training method for it. The resulting method is generally applicable to sequential models, including ResNets and RNNs. Furthermore, while the abovementioned models only consider dropping the whole residual units, we demonstrate training of the spatially adaptive ResNet. We hypothesize that the proposed ideas can be combined with the concurrently developed models.

Our work follows a trend in machine learning of interpreting methods as approximate Bayesian procedures. For example, in the field of topic modelling, Latent Dirichlet Allocation [41] is a probabilistic counterpart of Latent Semantic Indexing [42]. Recently, Dropout [43] has been interpreted as variational inference in a probabilistic model [44, 45]. This spurred the development of more innovative ways of using Dropout, *e.g.*, in RNNs [46] and for sparsifying neural networks [47]. We hope that our paper will similarly open the way for various extensions of adaptive computation time.

6. Experiments

We focus on the PSACT model for ResNets, since it allows to adjust the number of latent variables by grouping the spatial positions. First, we demonstrate that the relaxed model's parameters are compatible with the discrete and thresholded models. Then, we compare training of the relaxed model to training of the discrete model with with REINFORCE, for varying number of latent variables. Finally, we demonstrate that the relaxed PSACT model achieves close results to ACT. We also verify that the parameters obtained by the relaxed model can be used in a thresholded model with extremely simple test-time behavior, and that it is not the case for SACT.

We consider pre-activation ResNets [8] with 32 and 110 convolutional layers. We use CIFAR-10 image classification dataset [31]. Unless otherwise noted, PSACT is trained using the relaxed model and evaluated using the discrete model. As a proxy to the potential time savings, we compute the number of floating point operations (FLOPs) required to evaluate the positions with non-zero values in the active positions mask, as done in [12].

In the first experiment, we train a relaxed PSACT model. The obtained parameters are continuously evaluated on the test set in three models: relaxed (Concrete relaxation of the Bernoulli variables), discrete (discrete latent variables), and thresholded

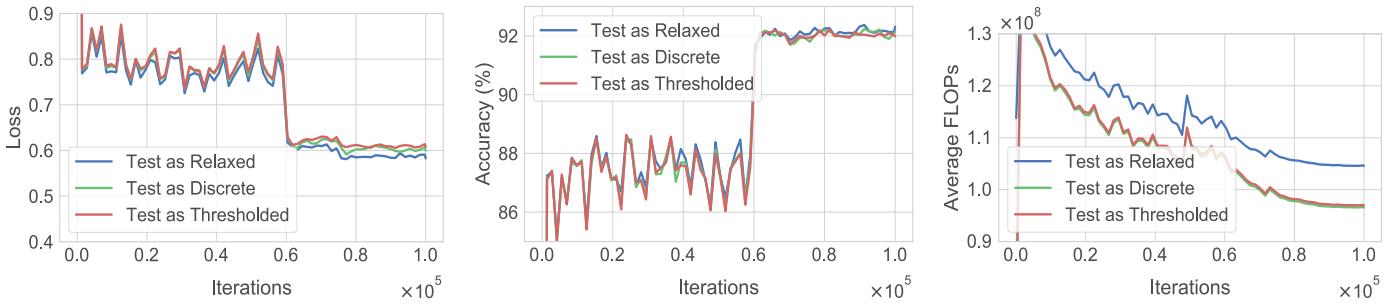

Fig. 3. The parameters from a relaxed PSACT model (ResNet-32, $\tau = 0.01$) for different training iterations are evaluated on the test set in Relaxed, Discrete and Thresholded models. The gap between the models is small throughout the training

(deterministic latent variables). The results on Fig. 3 show that the loss function and accuracy stay close for the three models. However, since the computation in relaxed model is stopped when $\prod_{i=1}^{l}(1 - \xi_k^i) < \delta$, and $\hat{\xi}_k^i$ might take non-extreme values, the relaxed model requires more computation.

Next, we compare training of the relaxed model to training of the discrete model using REINFORCE. We use an exponential moving average reward baseline with a decay factor of 0.99. We do not employ an input-dependent baseline to simplify the model, since the paper [13] finds small improvement from using it. Additionally, for REINFORCE, we use Adam optimizer [48] with initial learning rate of 10^{-3} (the decay schedule is kept the same), since SGD with momentum used in other experiments results in unstable training.

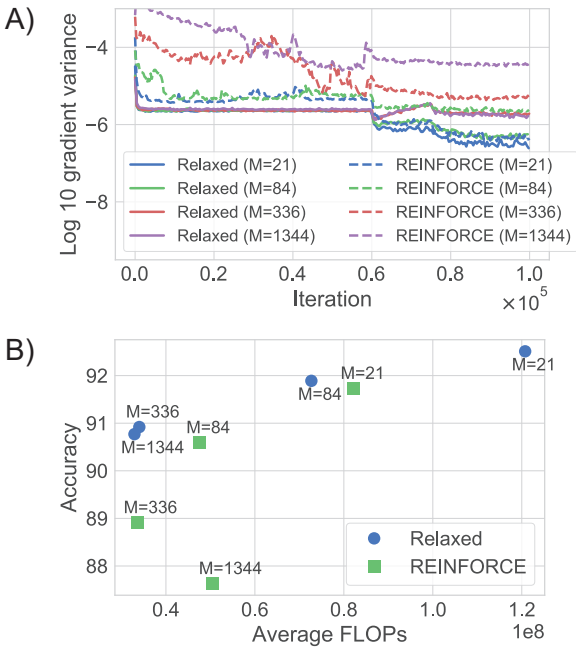

Fig. 4. Training of relaxed PSACT model (ResNet-32, $\tau = 0.1$) and training of discrete PACT model using REINFORCE, for varying number of the latent variables M. REINFORCE exhibits much higher variance of gradients and fails to reach a competitive accuracy for $M > 84$. A) $\log_{10}$ of the parameters gradient variance as a function of the training iteration. B) test FLOPs and accuracy at convergence (evaluation is performed in the discrete mode)

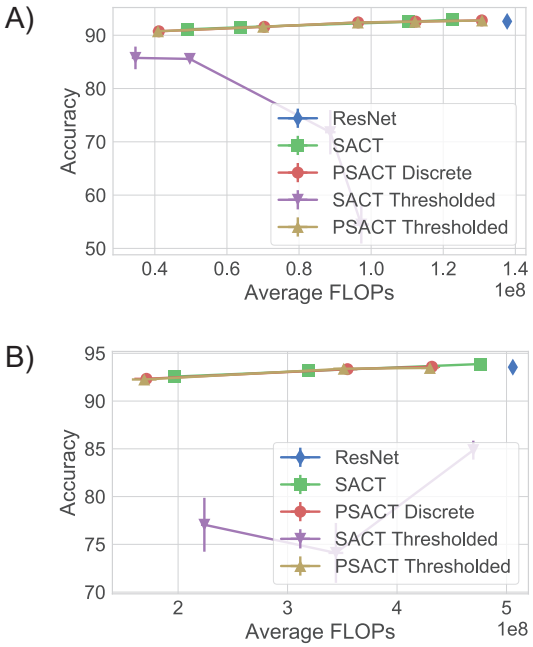

Fig. 5. Comparison of PSACT (proposed method) and SACT [12] for various values of the computation time penalty τ. PSACT is trained using the relaxed model. The results are averaged over five runs, with error bars denoting one standard deviation. A) ResNet-32, B) ResNet-110

PSACT model for ResNet-32 has $M = 1344$ 5-ary categorical latent variables: one variable per $(32 \cdot 32 + 16 \cdot 16 + 8 \cdot 8)$ spatial positions. To study the effect of the number of the latent variables on the training, we group the latent variables spatially. Namely, in every ResNet block, we group the spatial positions into non-overlapping $n \times n$ patches, $n \in \{2, 4, 8\}$. Within each patch, we average the logits of the halting probabilities and sample a single latent variable per patch. The results presented on Fig. 4 show that REINFORCE has a much higher gradient variance. For $M = 1344$ latent variables, the difference is about two orders of magnitude. REINFORCE achieves comparable results for $M = 21$ and $M = 84$ latent variables, but the accuracy quickly deteriorates when the number of latent units is increased.

Finally, we compare SACT and PSACT models for ResNet-32 and ResNet-110 on Fig. 5. The PSACT model is trained using

Fig. 6. Left to right: CIFAR-10 test image, ponder cost map for SACT, expected number of residual units per spatial position for PSACT. SACT and PSACT are applied to ResNet-110 with $\tau = 0.005$. Both methods allocate the computation to the object of interest

the relaxation and then evaluated in the discrete and thresholded regimes. PSACT and SACT perform similarly. We find that PSACT requires using somewhat lower computation time penalty τ to achieve the same number of FLOPs, perhaps because the expected number of iterations penalty in PSACT is easier to optimize than the surrogate ponder cost of SACT. Relaxed PSACT successfully trains on ResNet-110, where we have $M = 1344$ 18-ary discrete latent variables. PSACT can be evaluated in deterministic Thresholded mode with very close results, indicating that the latent variables probabilities have saturated. This is not the case for SACT: evaluation in Thresholded mode reduces the accuracy by at least 5%. We also present the comparison of the learned computation time maps on Fig. 6.

7. Conclusion

We have presented probabilistic adaptive computation time, a latent variable model for varying the amount of computation in deep models. The proposed stochastic variational optimization allows to perform approximate MAP inference in this model. Experimentally, we find that training using concrete relaxation of discrete latent variables outperforms REINFORCE-based training. The model achieves similar results to the heuristic method adaptive computation time, while enjoying a principled formulation. It can also be used in tresholded mode with a very simple test-time behavior and lower memory footprint. In future, we plan to explore different training techniques and modifications of the proposed latent variable model. Additionally, we expect that the proposed techniques could be useful for replacing REINFORCE in hard attention models.

Acknowledgements. Theoretical results in Chapters 3–4 were obtained with support from the Russian Science Foundation grant 17–71–20072. Experimental results in Chapter 5 were obtained with support from the Ministry of Education and Science of the Russian Federation (grant 14.756.31.0001).

References

[1] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," *NIPS*, 2012.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.
[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CVPR*, 2015.
[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CVPR*, 2016.
[5] T.N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," *ICASSP*, 2013.
[6] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *BMVC*, 2014.
[7] K. Neklyudov, D. Molchanov, A. Ashukha, and D.P. Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," *NIPS*, 2017.
[8] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *ECCV*, 2016.
[9] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional computation in neural networks for faster models," *ICLR Workshop*, 2016.
[10] A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, K. Kavukcuoglu, and G. E. Hinton, "Attend, infer, repeat: Fast scene understanding with generative models," *NIPS*, 2016.
[11] M. McGill and P. Perona, "Deciding how to decide: Dynamic routing in artificial neural networks," *ICML*, 2017.

[12] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," *CVPR*, 2017.
[13] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," *ICML*, 2014.
[14] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, 1992.
[15] Z. Li, Y. Yang, X. Liu, S. Wen, and W. Xu, "Dynamic computational time for visual attention," *ICCV*, 2017.
[16] A. Graves, "Adaptive computation time for recurrent neural networks," *arXiv*, 2016.
[17] M. Neumann, P. Stenetorp, and S. Riedel, "Learning to reason with adaptive computation," *NIPS Workshop on Interpretable Machine Learning in Complex Systems*, 2016.
[18] M. Ryabinin and E. Lobacheva, "Adaptive prediction time for sequence classification," *arXiv*, 2018.
[19] D.P. Kingma and M.Welling, "Auto-encoding variational bayes," *ICLR*, 2014.
[20] J. Staines and D. Barber, "Variational optimization," *arXiv*, 2012.
[21] J. Staines and D. Barber, "Optimization by variational bounding," *ESANN*, 2013.
[22] C.J. Maddison, A. Mnih, and Y.W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *ICLR*, 2017.
[23] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *ICLR*, 2017.
[24] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *NIPS*, 2015.
[25] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," *ICLR*, 2015.
[26] J. Ba, R.R. Salakhutdinov, R. B. Grosse, and B.J. Frey, "Learning wake-sleep recurrent attention models," *NIPS*, 2015.
[27] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R.S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *ICML*, 2015.
[28] M. Titsias and M. Lázaro-Gredilla, "Doubly stochastic variational bayes for non-conjugate inference," *ICML*, 2014.
[29] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *NIPS*, 2016.
[30] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A.L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *TPAMI*, 2017.
[31] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Computer Science Department, University of Toronto, Tech. Rep*., 2009.
[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, 1997.
[33] Y. Jernite, E. Grave, A. Joulin, and T. Mikolov, "Variable computation in recurrent neural networks," *ICLR*, 2017.
[34] A.W. Yu, H. Lee, and Q.V. Le, "Learning to skim text," *ACL*, 2017.
[35] V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang, "Skip rnn: Learning to skip state updates in recurrent neural networks," *ICLR*, 2018.
[36] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv*, 2013.
[37] S. Leroux, P. Molchanov, P. Simoens, B. Dhoedt, T. Breuel, and J. Kautz, "Iamnn: Iterative and adaptive mobile neural network for efficient image classification," *ICLR Workshop*, 2018.
[38] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "BlockDrop: Dynamic inference paths in residual networks," *CVPR*, 2018.
[39] A. Veit and S. Belongie, "Convolutional networks with adaptive computation graphs," *arXiv*, 2017.
[40] X. Wang, F. Yu, Z.-Y. Dou, and J.E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," *arXiv*, 2017.
[41] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *JMLR*, 2003.
[42] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, 1990.
[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.
[44] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *NIPS*, 2015.
[45] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *ICML*, 2016.
[46] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," *NIPS*, 2016.
[47] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," *ICML*, 2017.
[48] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.