University of Amsterdam

MSc Artificial Intelligence
Master Thesis

Probabilistic approach to Adaptive Computation Time in (Deep) Neural Networks

by

Jörg Sander
(10881530)

supervised by Patrick Putzky & Matthias Reisser

5th July, 2017

AMLAB
Amsterdam
Machine Learning Lab

1 Introduction

Anticipating how much mental effort is needed to solve a problem is a notoriously difficult task for us humans although experience and intuition can give us some guidance especially if the task has strong resemblance with previous encountered challenges.

Machine learning algorithms generally *suffer* from the same lack of foresight and mostly miss the ability to dynamically adjust the number of computational steps to the complexity of the task at hand. More specifically the architecture of feedforward neural networks consists of an a priori fixed number of layer-to-layer transformation (network *depths*) and in recurrent neural networks (RNN) additionally the fixed sequence length determines the number of computational steps.

Clearly there is a strong motivation to endow a machine learning model with the ability to be parsimonious in its use of computation, ideally limiting itself to the minimum number of steps necessary to solve a specific problem. This view is also motivated by the assumption that for the most general case in which the model tries to minimize some objective function f, we suppose the evaluation of f for some input x and the computation of the gradients of f w.r.t. the model parameters θ to be expensive operations.

In this work we describe a probabilistic, gradient-based approach in order for an iterative machine learning algorithm (1) to use less computational steps to achieve its objective and (2) to dynamically determine the time step when to stop computation.

The rest of this document is structured as follows. After giving an overview of the related work in section 2 we explicitly state the objectives of this research in section 3. The details of our approach are outlined in section 4 and section 5 specifies the conducted experiments for which we report the results in section 6.

2 Related work

Our work is related to the recent surge of interest in using neural networks to learn optimization procedures, applying a range of innovative meta-learning techniques ([8], [1], [2]). In particular the work we present, took the *LSTM*[1] *optimizer* described in [1] as a baseline model. The approach taken in this study focuses on learning how to utilize gradient observations over time, of a function f to be optimized, in order to achieve fast learning of the underlying model. The authors show that such a model outperforms w.r.t. convergence speed and final minimum, hand-designed optimizers like ADAM, RMSprop or SGD on tasks like simple quadratic function optimization or training of a neural network for image classification.

Another recent study by [2] uses a meta-learning approach for training RNNs to perform black-box global optimization. Inspired by the Bayesian optimization framework the authors replace the expectation in the loss function of [1] with the *expected posterior improvement* which encourages an exploratory behavior into the meta learning model. The work is interesting for us because the model it focuses on the issues of computation speed, horizon length (i.e. number of iterative optimization steps during training) and exploration-exploitation trade-offs. Their research reveals that the RNN optimizer is faster and tends to achieve better performance within the horizon for which it is trained. It however underperforms against Bayesian optimization for much longer horizons as it has not learned to explore for longer horizons.

Another lead of this project comes from the recent work of [6], [10], [7] and [9] in which the authors pursue different approaches to induce a machine learning model with the ability to dynamically adjust the computational time needed to solve a specific task. We are particularly interested in the research of Alex Graves [6] that outlines an approach of *Adaptive Computational Time* (ACT) applied to Recurrent Neural Networks that learn how many computational steps to take between receiving an input and emitting an output. The work is important because it makes a significant contribution towards gradient-based approaches for learning the number of computational steps in a neural computation graph. The RNN learns a so called *halting distribution* by adding parallel (to the recurrent states) layers of sigmoidal halting units, which are computed at each iteration. The cumulative probability of the halting units determines the moment when computation stops.

A very similar approach has been proposed in [7]. The work extends the basic Elman RNN unit [3] with the ability to decide at each time step how much computation it requires to perform, based on the current hidden state and input.

The work of [5] extends ACT to spatially adaptive computational time (SACT) for Residual Networks. Their model incorporates attention into Residual Networks by learning a deterministic policy that stops computation in a spatial position as soon as the features become *good enough.*

A stochastic approach to ACT has been applied to scene understanding with recurrent networks [4]. This study develops a probabilistic inference framework that learns to choose the appropriate number of inference steps. The model attends to scene elements and processes them one at a time, deciding autonomously to how many objects in the scene it attends to.

[1] Long Short Term Memory

A related line of work uses approaches from reinforcement learning to increase the computational efficiency of (deep) neural networks. In [10] the underlying idea is that machine learning models are often used at test-time subject to constraints and trade-offs not present at training-time. The authors propose a model that learns to change behavior at test time with reinforcement learning by adaptively constructing computational graphs from sub-modules on a per-input basis. The recent work of [9] is particularly close in spirit to the work we have presented here. The authors propose a dynamic computational time model to adaptively adjust computational time during inference for a recurrent visual attention (RAM) model. Using reinforcement learning the model learns the optimal attention and *stopping* policy which is modelled by separate parameters of the recurrent neural network.

3 Objectives of research (will be skipped in final version)

The work described here combines the ideas of [1] and [6]. We take the LSTM optimizer from [1] as a baseline model and develop the following extensions: (1) a loss function that encourages the model to use less iterative steps than the baseline LSTM optimizer; (2) an approximation of a discrete posterior distribution over time steps t that specifies the probability that computation stops at step t. The posterior distribution will be exploited to dynamically determine the optimization step when computation should stop.

4 Approach

4.1 LSTM optimizer

We will implement the baseline model described in [1] as a recurrent neural network (RNN) m parameterized by ϕ. The update steps $\mathbf{g}_t$ of the optimizer will be the output of the RNN. Given a distribution over functions f the expected loss of the model for a discrete number of computational steps T will be equal to

$$\mathbb{E}[\mathbb{L}_{meta}(\phi)] = \mathbb{E}_{p(f)}\left[\sum_{t=1}^{T} f(\boldsymbol{\theta}_t)\right] \text{ where } \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{g}_t \ , \tag{1}$$

where f is the function to be optimized (hereafter referred to as *optimizee*) which is parameterized by $\boldsymbol{\theta}$. The input of the RNN are the gradients of f denoted by $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)$ (we will use the shorthand notation $\nabla_t f(\boldsymbol{\theta}_t)$ hereafter). The model can be formalized as follows

$$\begin{bmatrix} \mathbf{g}_t \\ \mathbf{h}_{t+1} \end{bmatrix} = m(\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t), \mathbf{h}_t, \phi) \ , \tag{2}$$

where $\mathbf{h}$ denotes the hidden state calculated by the RNN which is passed as input to the next time step $t+1$ and can be interpreted as a compressed state of the current and previous time steps (note that $\mathbf{h}_0$ will be initialized with $\mathbf{0}$).

4.2 ACT optimizer

The so called ACT optimizer is an extension of the LSTM optimizer and additionally learns to approximate a discrete posterior distribution over time steps $q(\mathbf{t}|.)$ that specifies the probability that computation stops at step t. The posterior distribution will be used to dynamically determine how many computational steps to take before the algorithm submits an output (the exact way still has to be developed).

Our approach is inspired by the work of [11] in which the authors develop an algorithm to translate the problem of solving a Markov Decision Problem (MDP) into a problem of likelihood maximization.

We are defining a probabilistic model in which $\mathbf{x}$ denotes the observed, continuous random variable (e.g. the gradients of the function f we want to optimize) and $\mathbf{t}$ (time steps) a discrete T-dimensional binary latent variable which has a 1-of-T representation. The joint distribution $p_\phi(\mathbf{x}, \mathbf{t})$ is parameterized by a set of parameters ϕ (e.g. modelled by an RNN). Our goal is to maximize the marginal likelihood function $p_\phi(\mathbf{x})$ which we can decompose into

$$\begin{aligned} p_\phi(\mathbf{x}) &= \sum_{T=1}^{\infty} p(T)\ p_\phi(\mathbf{x}|T) \\ &= \sum_{T=1}^{\infty} p(T) \sum_{t=1}^{T} p_\phi(\mathbf{x}, t|T) \end{aligned} \tag{3}$$

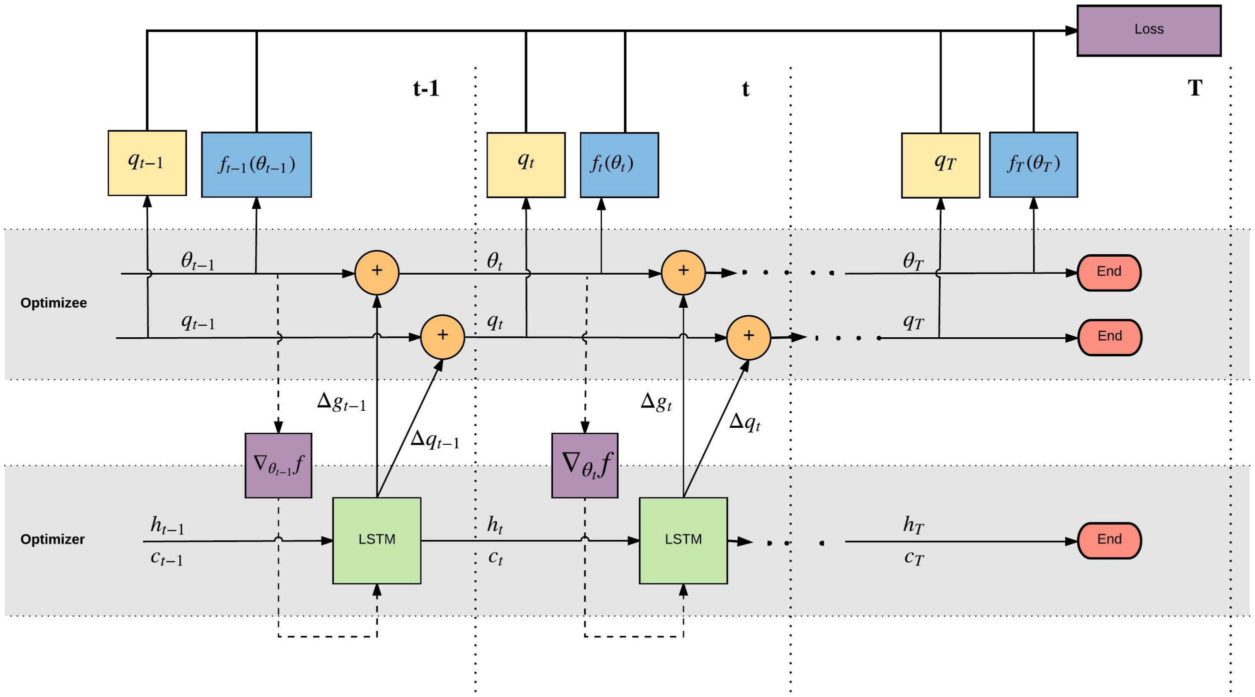

Figure 1: Computational graph of the ACT optimizer

by making use of the sum and product rules of probability. We can then *lower bound* the marginal log-likelihood $\log p_\phi(\mathbf{x})$ by applying Jensen's inequality (making use of the concavity of the log function)

$$\log p_\phi(\mathbf{x}) = \log \sum_{T=1}^{\infty} p(T)\ p_\phi(\mathbf{x}|T) \geq \sum_{T=1}^{\infty} p(T)\ \log p_\phi(\mathbf{x}|T)\ . \tag{4}$$

This result implies that we need to maximize the marginal log-likelihood $\log p_\phi(\mathbf{x}|T)$ but we instead maximize the complete-data log-likelihood function $\log p_\phi(\mathbf{x}, \mathbf{t}|T)$ because this is significantly easier. We introduce the distribution $q_\phi(\mathbf{t}|\mathbf{x})$ which is an approximation of the true prior distribution $p(\mathbf{t})$ which we assume belongs to the family of geometric distributions. Making use of the following decomposition

$$\log p_\phi(\mathbf{x}|T) = \mathcal{L}(q, \phi) + D_{KL}\big(q_\phi(\mathbf{t}|\mathbf{x}, T)\ ||\ p(\mathbf{t}|T)\big)\ , \tag{5}$$

where $\mathcal{L}$ denotes the *estimated lower bound* and D_{KL} the Kullback-Leibler divergence. In the following we will omit the parameterization notation by ϕ to prevent cluttering. Instead of minimizing the KL-divergence we can maximize the lower bound which decomposes into

$$\begin{aligned} \log p_\phi(\mathbf{x}|T) \geq \mathcal{L}(q, \phi) &= \mathbb{E}_{q(\mathbf{t}|\mathbf{x},T)}\Big[\log p(\mathbf{x}, \mathbf{t}|T) - \log q(\mathbf{t}|\mathbf{x}, T)\Big] \\ &= \mathbb{E}_{q(\mathbf{t}|\mathbf{x},T)}\Big[\log p(\mathbf{x}|\mathbf{t}, T)\Big] - D_{KL}\big(q(\mathbf{t}|\mathbf{x}, T)\ ||\ p(\mathbf{t}|T)\big) \\ &= \sum_{k=1}^{T} q(t_k|\mathbf{x}, T) \log p(\mathbf{x}|t_k, T) - D_{KL}\Big(q(t_k|\mathbf{x}, T)\ ||\ p(t_k|T)\Big)\ . \end{aligned} \tag{6}$$

Please note that $q_\phi(t_k|\mathbf{x})$ can be decomposed into

$$q(t_k|\mathbf{x}) = \sum_{T=t_k}^{\infty} q(t_k|\mathbf{x}, T)p(T)\ , \tag{7}$$

where again T denotes some finite time horizon (the details of the distributions $q(\mathbf{t}|\mathbf{x}, T)$ and $p(T)$ are given in section 4.3).

Inspired by [11] we assume that the joint distribution $p_\phi(\mathbf{x}, \mathbf{t}|T)$ can be factored into a mixture of independent distributions

$$p_\phi(\mathbf{x}, \mathbf{t}|T) = \prod_{k=1}^{T} \Big[p_\phi(\mathbf{x}|t_k, T)\ p(t_k|T) \Big]^{t_k} . \tag{8}$$

Finally using 4 we can *lower bound* the marginal log-likelihood $\log p_\phi(\mathbf{x})$ by using the result for $\log p_\phi(\mathbf{x}|T)$ from equation 6

$$\begin{aligned} \log p_\phi(\mathbf{x}) &\geq \sum_{T=1}^{\infty} p(T) \sum_{k=1}^{T} q(t_k|\mathbf{x}, T) \log p(\mathbf{x}|t_k, T) - D_{KL}\Big(q(t_k|\mathbf{x}, T) \mid\mid p(t_k|T) \Big) \\ &\geq \mathbb{E}_{p(T)} \Big[\mathbb{E}_{q(\mathbf{t}|\mathbf{x},T)} \Big[\log p(\mathbf{x}|\mathbf{t}, T) \Big] - D_{KL}\big(q(\mathbf{t}|\mathbf{x}, T) \mid\mid p(\mathbf{t}|T) \big) \Big] . \end{aligned} \tag{9}$$

Using this result we can formulate the lower bound on the log-likelihood of the ACT optimizer as follows:

$$\mathcal{L}_{act}(q, \phi) = \mathbb{E}_{p(T)} \Big[\sum_{k=1}^{T} q(t_k|\mathbf{x}, T)\ \log p(\mathbf{x}|t_k, T) - D_{KL}\big(q(t_k|\mathbf{x}, T) \mid\mid p(t_k|T) \big) \Big] . \tag{10}$$

In comparison to the LSTM optimizer which outputs the update steps g_t our model will additionally emit a differential Δq_t logit that will be used in the update rule of the q_t logit

$$q_{t+1} = q_t + \Delta q_t \quad \text{where} \ \ q_0 = 0 . \tag{11}$$

During training the final probabilities $q(\mathbf{t}|\mathbf{x}, T)$ over the horizon T will be calculated by means of the Softmax function and the q_t logits computed at each time step t. At inference time the probabilities need to be computed at each time step t up to horizon $T = t$.

$$q(t|\mathbf{x}, T) = \frac{\exp\left(q_t\right)}{\sum_{t'=1}^{T} \exp(q_{t'})} . \tag{12}$$

The ACT optimizer will be implemented as an RNN and can be formalized as follows

$$\begin{bmatrix} \mathbf{g}_t \\ \Delta q_t \\ \mathbf{h}_t \end{bmatrix} = m'(\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t), \mathbf{h}_{t-1}, \phi, \mathbf{w}_q) \ \text{ where } \ q_{t+1} = q_t + \Delta q_t \text{ and } \Delta q_t = \mathbf{w}_q^T \mathbf{h}_t \tag{13}$$

where $\Theta = \{\phi, \mathbf{w}_q\}$ is the set of learnable parameters.

4.3 Training the ACT optimizer

The important extension of our model in comparison to the LSTM optimizer is the approximation $q(\mathbf{t}|\mathbf{x})$ of the true prior probability distribution over time steps $p(\mathbf{t})$ that specifies the probability that computation stops at step t. Please note that this distribution is task-dependent. In order to be able to learn this distribution the decomposition in equation 7 of the approximated distribution $q(\mathbf{t}|\mathbf{x})$ is essential for training the model. We make the assumption that the true distribution $p(\mathbf{t})$ belongs to the family of geometric distributions. The approximated distributions during training $q(\mathbf{t}|\mathbf{x})$ mainly differs from the true distribution in the fact that the former only has finite support (whereas the true distribution has infinite support).

During training we need to be able to sample the horizon T from $p(T)$ in equation 7 for the optimizee (e.g. in case the optimizee is a regression function f we sample T for each function the model optimizes). In addition in order to compute $\mathcal{L}_{act}$ as specified in equation 10 we are required to compute the prior probabilities of the true distribution $p(\mathbf{t}|T)$.

Hence we need to find a decomposition of $p(\mathbf{t})$ into

$$p(\mathbf{t}) = \sum_{T=t}^{\infty} p(\mathbf{t}|T)\ p(T) \tag{14}$$

such that

$$p(t) = \mu(1-\mu)^{t-1} = \mu\nu^{t-1} \qquad \text{where} \ \ \mu, \nu \in (0, 1] \tag{15}$$

where μ is the *success* parameter (i.e. computation stops) and t denotes the trial number i.e. halting step.

A possible solution for $p(T)$ and $p(t|T)$ is given by

$$\begin{aligned} p(T) &= \widetilde{\nu}^{T-1}(1-\widetilde{\nu}^{T})(1-\widetilde{\nu}^{2}) \\ p(t|T) &= \frac{\widetilde{\mu}\,\widetilde{\nu}^{t-1}}{1-\widetilde{\nu}^{T}} \end{aligned} \tag{16}$$

with $\widetilde{\nu} = \sqrt{\nu}$ and $\widetilde{\mu} = 1 - \sqrt{\mu}$ (a proof is given in appendix B).

5 Experiments

As previously mentioned, we will use the LSTM optimizer described in [1] as our baseline model in order to evaluate the performance (or *behavior*) of the newly developed ACT optimizer. We intend to conduct the following experiments: (1) training the optimizer models on synthetic, ten dimensional regression functions; (2) train the optimizers to learn optimize a small neural network on the MNIST dataset; (3) evaluate the performance of the trainable optimizers when optimizing a convolutional neural network for image classification on the CIFAR-10 dataset. Due to the *dynamic nature* of the model we will be using *PyTorch*[2] to implement both models.

5.1 Quadratic functions (replication of first L2L experiment)

In this experiment we replicated the *quadratic functions* experiment from [1] in order to validate our implementation of the LSTM baseline optimizer. The optimizer minimizes functions of the form

$$f(\boldsymbol{\theta}) = ||\mathbf{W}\boldsymbol{\theta} - \mathbf{y}||_2^2 \tag{17}$$

where $\mathbf{W}$ is a 10x 10 matrix and $\mathbf{y}$ is a 10-dimensional vector whose values are sampled from a uniform distribution. The initial values of the 10-dimensional parameter vector $\boldsymbol{\theta}$ are sampled from an IID Gaussian distribution with a standard deviation of 0.01.

The LSTM optimizer consists of a two-layer LSTM with 20 hidden units in each layer. The final output value $\mathbf{g}_t$ per time step is generated by means of a linear transformation that omits the output bias. In order to prevent the LSTM network of having a huge hidden state and a vast number of parameters the optimizer operates *coordinatewise* on the parameters of the optimizee. Therefore the optimizer parameters are shared across different optimizee parameters although each of the latter has its own hidden state.

The model is trained for 120 epochs where each epoch contained 10,000 newly sampled functions using a mini-batch size of 125 and each batch of functions is minimized for a fixed horizon T=100 time steps. The training procedure used truncated Back-Propagation Through Time (BPTT) in which the LSTM was unrolled for 20 time steps using a learning rate of 5e-6 (which was determined by random search) and ADAM for the minimization procedure. The performance of the model was evaluated every 10 epochs on a fixed set of 15,000 functions using again an optimization horizon of 100 steps and the model parameters were saved after each validation run. We choose the best model based on the final validation loss (sum of time step losses) and tested its performance on a freshly sampled set of 20,000 functions which were optimized for 200 time steps.

5.1.1 Results quadratic functions

Figure 2 shows the learning curve of the LSTM optimizer on the test set of 20,000 functions. The curve shows the average loss over test functions for each of the 200 optimization steps (please note that the y-axis has a logarithmic scale). The result is comparable to figure 4 (left on page 5) from the L2L paper. Both figures indicate that the learning curves reach a value of 0.1 around time step 30 and converge between 0.1 and 0.01.

5.2 Linear regression with 10 parameters

In this experiment the LSTM and ACT model will learn to optimize linear regression functions with two parameters (denoted $\boldsymbol{\theta}$) and one input variable x

$$f(\mathbf{x}^{(n)}, \boldsymbol{\theta}^*) = \sum_{j=1}^{M} \theta_j^* x_j^{(n)} + \epsilon\ , \tag{18}$$

where $M = 10$, $\boldsymbol{\theta}, \mathbf{x} \in \mathbb{R}^{10}$ and $\boldsymbol{\theta}^*$ denotes the true parameters to be determined by the optimizer models. For each regression function to be optimized we sample the *true* parameters $\boldsymbol{\theta}^*$ from a Gaussian distribution

[2] http://pytorch.org/

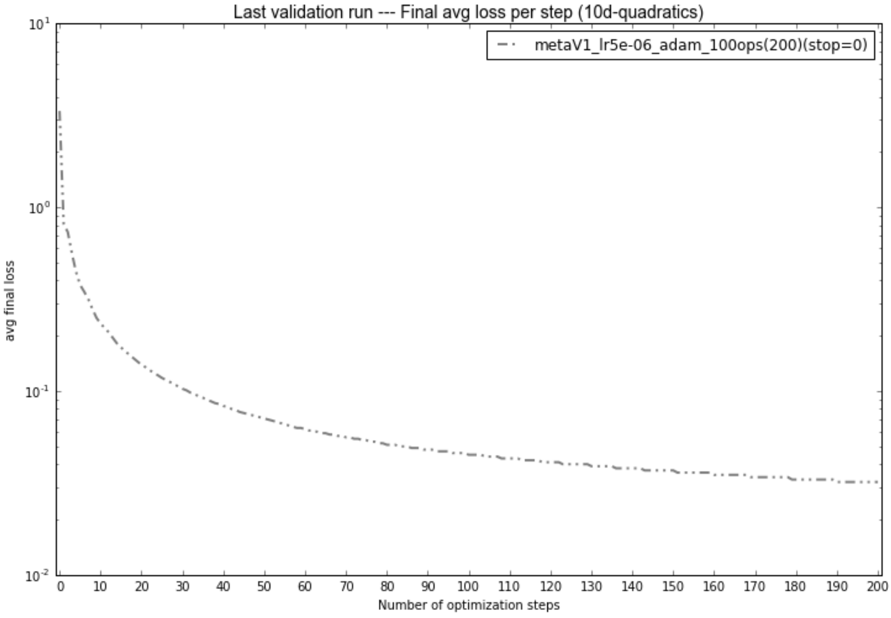

Figure 2: Performance of LSTM optimizer on 10-dimensional quadratic functions

$\mathcal{N}(0,\alpha^{-1})$. In order to determine the log-likelihood of the parameters we sample N observations $\{x_n\}$, where $n = 1,\dots,N$ for each regression function f and add some fixed noise ϵ to each function value where $\epsilon \sim \mathcal{N}(0,\beta^{-1})$.

The LSTM optimizer will minimize the expected negative log-likelihood for a discrete number of computational steps T given a distribution over functions $p(f)$

$$\begin{aligned}\mathbb{E}[L_{meta}(\phi)] &= \mathbb{E}_{p(f)}\Big[\sum_{t=1}^{T} -\log p\big(f(x,\boldsymbol{\theta}_t)|x,\boldsymbol{\theta}_t,\beta^{-1}\big)\Big]\\ &= \sum_{t=1}^{T}\ \frac{1}{L}\sum_{l=1}^{L} -\log p\big(f^{(l)}(x,\boldsymbol{\theta}_t)|x,\boldsymbol{\theta}_t,\beta^{-1}\big)\end{aligned} \tag{19}$$

,

where the negative log-likelihood is equal to

$$NLL = -\log p\big(f(x,\boldsymbol{\theta}_t)|x,\boldsymbol{\theta}_t,\beta^{-1}\big) = \frac{\beta}{2}\sum_{n=1}^{N}\Big\{f(x^{(n)},\boldsymbol{\theta}_t) - f(x^{(n)},\boldsymbol{\theta}^*)\Big\}^2 - \frac{N}{2}\log\beta + \frac{N}{2}\log(2\pi)\ . \tag{20}$$

where L denotes the size of the *mini-batch* of functions f in order to form the Monte Carlo (MC) estimates (we used $L = 125$).

As previously mentioned the ACT optimizer will optimize the expected lower bound on the log-likelihood $\log p(\mathbf{x})$. Equation 10 can be easily adjusted by replacing the log-likelihood term $\log p(\mathbf{x}|t,T)$ with the log-likelihood specified above, which results in

$$\begin{aligned}\mathcal{L}_{act}(q,\phi) &= \mathbb{E}_{p(T)}\Big[\mathbb{E}_{p(f)}\Big[\sum_{k=1}^{T} q(t_k|\mathbf{x},T)\ \ \Big(-\frac{\beta}{2}\sum_{n=1}^{N}\Big\{f(x^{(n)},\boldsymbol{\theta}_t) - f(x^{(n)},\boldsymbol{\theta}^*)\Big\}^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)\Big)\\ &\quad - D_{KL}\big(q(t_k|\mathbf{x},T)\ ||\ p(t_k|T)\big)\Big]\Big]\\ &= \sum_{T=1}^{\infty} p(T)\Big(\mathbb{E}_{p(f)}\Big[\sum_{k=1}^{T} q(t_k|\mathbf{x},T)\ \ \Big(-\frac{\beta}{2}\sum_{n=1}^{N}\Big\{f(x^{(n)},\boldsymbol{\theta}_t) - f(x^{(n)},\boldsymbol{\theta}^*)\Big\}^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)\Big)\\ &\quad - D_{KL}\big(q(t_k|\mathbf{x},T)\ ||\ p(t_k|T)\big)\Big]\Big)\ .\end{aligned} \tag{21}$$

With MC estimates of expectation w.r.t. $p(f)$

$$
\begin{aligned}
\mathcal{L}_{act}(q,\phi) = &\sum_{T=1}^{\infty} p(T)\Big(\mathbb{E}_{p(f)}\Big[\sum_{k=1}^{T} q(t_k|\mathbf{x},T)\ \ \Big(-\frac{\beta}{2}\sum_{n=1}^{N}\Big\{f(x^{(n)},\boldsymbol{\theta}_t)-f(x^{(n)},\boldsymbol{\theta}^*)\Big\}^2+\frac{N}{2}\log\beta-\frac{N}{2}\log(2\pi)\Big)\\
&-D_{KL}\big(q(t_k|\mathbf{x},T)\ ||\ p(t_k|T)\big)\Big]\Big)\\
\approx &\sum_{T=1}^{\infty} p(T)\Big[\frac{1}{L}\sum_{l=1}^{L}\sum_{k=1}^{T} q(t_k|f^{(l)}(\mathbf{x},\boldsymbol{\theta}_t),T)\ \ \Big(-\frac{\beta}{2}\sum_{n=1}^{N}\Big\{f^{(l)}(x^{(n)},\boldsymbol{\theta}_t)-f^{(l)}(x^{(n)},\boldsymbol{\theta}^*)\Big\}^2+\frac{N}{2}\log\beta-\frac{N}{2}\log(2\pi)\Big)\\
&-D_{KL}\big(q(t_k|f^{(l)}(\mathbf{x},\boldsymbol{\theta}_t),T)\ ||\ p(t_k|T)\big)\Big]
\end{aligned}
\tag{22}
$$

where L denotes again the size of the mini-batch of functions f.

6 Results

7 Conclusion and discussion

References

[1] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and de Freitas, N. Learning to learn by gradient descent by gradient descent. *arXiv:1606.04474 [cs]* (June 2016). arXiv: 1606.04474.

[2] Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., and de Freitas, N. Learning to Learn for Global Optimization of Black Box Functions. *arXiv:1611.03824 [cs, stat]* (Nov. 2016). arXiv: 1611.03824.

[3] Elman, J. L. Finding structure in time. *Cognitive science 14*, 2 (1990), 179–211.

[4] Eslami, S. M. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Kavukcuoglu, K., and Hinton, G. E. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. *arXiv:1603.08575 [cs]* (Mar. 2016). arXiv: 1603.08575.

[5] Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. Spatially adaptive computation time for residual networks. *arXiv preprint arXiv:1612.02297* (2016).

[6] Graves, A. Adaptive Computation Time for Recurrent Neural Networks. *arXiv:1603.08983 [cs]* (Mar. 2016). arXiv: 1603.08983.

[7] Jernite, Y., Grave, E., Joulin, A., and Mikolov, T. Variable Computation in Recurrent Neural Networks. *arXiv:1611.06188 [cs, stat]* (Nov. 2016). arXiv: 1611.06188.

[8] Li, K., and Malik, J. Learning to Optimize. *arXiv:1606.01885 [cs, math, stat]* (June 2016). arXiv: 1606.01885.

[9] Li, Z., Yang, Y., Liu, X., Wen, S., and Xu, W. Dynamic Computational Time for Visual Attention. *arXiv:1703.10332 [cs]* (Mar. 2017). arXiv: 1703.10332.

[10] Odena, A., Lawson, D., and Olah, C. Changing model behavior at test-time using reinforcement learning. *arXiv preprint arXiv:1702.07780* (2017).

[11] Toussaint, M., and Storkey, A. Probabilistic Inference for Solving Discrete and Continuous State Markov Decision Processes. In *Proceedings of the 23rd International Conference on Machine Learning* (New York, NY, USA, 2006), ICML '06, ACM, pp. 945–952.

Appendices

A Derivation of loss function

Our goal is to maximize the marginal likelihood function $p_\phi(\mathbf{x})$ but we instead maximize the complete-data likelihood function $p_\phi(\mathbf{x}, \mathbf{t})$ because this is significantly easier. We introduce the distribution $q_\phi(\mathbf{t}|\mathbf{x})$ which is an approximation of the true prior distribution $p(\mathbf{t}|\mathbf{x})$ which we assume belongs to the family of geometric distributions. Making use of the following decomposition

$$\log p_\phi(\mathbf{x}) = \mathcal{L}(q,\phi) + D_{KL}\big(q_\phi(\mathbf{t}|\mathbf{x}) \mid\mid p(\mathbf{t}|\mathbf{x})\big)\ , \tag{23}$$

where $\mathcal{L}$ denotes the *estimated lower bound* and D_{KL} the Kullback-Leibler divergence. In the following we will omit the parameterization notation by ϕ to prevent cluttering. Instead of minimizing the KL-divergence we can maximize the lower bound which decomposes into

$$\begin{aligned}\log p(\mathbf{x}) \geq \mathcal{L}(q,\phi) &= \mathbb{E}_{q(\mathbf{t}|\mathbf{x})}\Big[\log p(\mathbf{x},\mathbf{t}) - \log q(\mathbf{t}|\mathbf{x})\Big] = \mathbb{E}_{q(\mathbf{t}|\mathbf{x})}\Big[\log p(\mathbf{x}|\mathbf{t})\Big] - D_{KL}\big(q(\mathbf{t}|\mathbf{x}) \mid\mid p(\mathbf{t})\big) \\ &\geq \sum_{t=1}^{\infty} q(t|\mathbf{x})\Big(\log p(\mathbf{x}|t) - \log\frac{q(t|\mathbf{x})}{p(t)}\Big)\ .\end{aligned} \tag{24}$$

We can always decompose $q(t|\mathbf{x})$ as

$$q(t|\mathbf{x}) = \sum_{T=t}^{\infty} q(t|\mathbf{x},T)p(T)\ , \tag{25}$$

where again T denotes some finite time horizon (the details of the distributions $q(t|\mathbf{x},T)$ and $p(T)$ are given in section 4.3). Replacing equation 25 in 24 results in

$$\begin{aligned}\log p(\mathbf{x}) \geq &\sum_{t=1}^{\infty}\sum_{T=1}^{\infty} q(t|\mathbf{x},T)p(T)\ \log p(\mathbf{x}|t,T) \\ &-\sum_{t=1}^{\infty}\sum_{T=1}^{\infty} q(t|\mathbf{x},T)p(T)\log\sum_{T=1}^{\infty}\frac{q(t|\mathbf{x},T)p(T)}{p(t|T)p(T)}.\end{aligned} \tag{26}$$

Pulling the summation over T to the front results in

$$\begin{aligned}\log p(\mathbf{x}) &\geq \sum_{T=1}^{\infty} p(T)\sum_{t=1}^{T} q(t|\mathbf{x},T)\Big(\log p(\mathbf{x}|t,T) - \log\frac{q(t|\mathbf{x},T)}{p(t|T)}\Big) \\ &\geq \sum_{T=1}^{\infty} p(T)\ \Big[\mathbb{E}_{q(\mathbf{t}|\mathbf{x},T)}\big[\log p(\mathbf{x}|\mathbf{t},T)\big] - D_{KL}\big(q(\mathbf{t}|\mathbf{x},T) \mid\mid p(\mathbf{t}|T)\big)\Big] \\ &\geq \mathbb{E}_{p(T)}\Big[\mathbb{E}_{q(\mathbf{t}|\mathbf{x},T)}\big[\log p(\mathbf{x}|\mathbf{t},T)\big] - D_{KL}\big(q(\mathbf{t}|\mathbf{x},T) \mid\mid p(\mathbf{t}|T)\big)\Big].\end{aligned} \tag{27}$$

B Decomposition of geometric distribution

$$\begin{aligned}p(t) &= \mu\ \nu^{t-1} \\ p(t) &= \sum_{T\geq t} p(t|T)p(T) = \frac{\widetilde{\mu}\ \widetilde{\nu}^{t-1}}{1-\widetilde{\nu}^{T}}\ \widetilde{\nu}^{T-1}(1-\widetilde{\nu}^{T})(1-\widetilde{\nu}^{2}) \\ &= \sum_{T\geq t}\widetilde{\mu}\ \widetilde{\nu}^{t-1}\widetilde{\nu}^{T-1}\ (1-\widetilde{\nu}^{2}) \\ &= \widetilde{\mu}\ \widetilde{\nu}^{t-1}\ (1-\widetilde{\nu}^{2})\sum_{T\geq t}\widetilde{\nu}^{T-1} \\ &= \widetilde{\mu}\ \widetilde{\nu}^{t-1}\ (1-\widetilde{\nu}^{2})\Big(\sum_{T=1}^{\infty}\widetilde{\nu}^{T-1} - \sum_{T=1}^{t-1}\widetilde{\nu}^{T-1}\Big) \\ &= \widetilde{\mu}\ \widetilde{\nu}^{t-1}\ (1-\widetilde{\nu}^{2})\Big(\frac{1}{1-\widetilde{\nu}} - \frac{1-\widetilde{\nu}^{t-1}}{1-\widetilde{\nu}}\Big) \\ &= \widetilde{\nu}^{2(t-1)}\ (1-\widetilde{\nu}^{2}) \\ &= \nu^{t-1}\ (1-\nu) = \mu\ \nu^{t-1} = p(t)\ \text{ because with } \widetilde{\nu}^{2} = \nu\end{aligned} \tag{28}$$

C Efficient sampling of horizons T from $p(T)$ during training ACT model

```
Sampling

Goal: Efficient sampling of horizon $T$ from $p(T)$ during training

Input: Shape parameter $\nu$, the number of mini-batches per epoch $B$ , number of samples $N$ from $p(T|\nu)$
Output: the set $\mathcal{F}$ which consists of $B$ subsets of horizons $T$

p(T) ← $\widetilde{\nu}^{T-1}(1-\widetilde{\nu}^{T})(1-\widetilde{\nu}^{2})$
// the set $\mathcal{F}$ will contain sets of horizons $T$ that will be evaluated for a mini-batch
$\mathcal{F} \leftarrow \emptyset$
j ← 0
while $|\mathcal{F}| < B$ do
    // Note that $\mathcal{T}$ is a multiset i.e. it can contain multiple elements with the same value
    $\mathcal{T} \leftarrow$ sample $N$ horizons T from $p(T|\nu)$
    // where $\mathcal{T} = \{T_n\}_{n=1}^{N}$ using inverse transform sampling method
    // sort the set in descending order
    $\mathcal{T} \leftarrow \mathcal{T}$.descending()
    while $\mathcal{T} \neq \emptyset \wedge |\mathcal{F}| < B$ do
        j ← j + 1
        // pick the maximum horizon T
        $T_{max} \leftarrow \mathcal{T}.\max()$
        $\mathcal{T}_j \leftarrow \{T_{max}\}$
        // get all elements of $\mathcal{T}$ that are smaller than $T_{max}$. Note, the set $\mathcal{T}$ probably contains multiple samples of the same value. We only pick each value once.
        $\mathcal{T}_j \leftarrow \{\forall\, T' \in \mathcal{T} \wedge T' < T_{max}\} \cup \mathcal{T}_j$
        // remove elements of $T_j$ from set $\mathcal{T}$
        $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{T}_j$
        // add the set $\mathcal{T}_|$ as a subset to $\mathcal{F}$
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{\mathcal{T}_|\}$
return $\mathcal{F}$
```

Algorithm 1: Sample T from $p(T)$ during training ACT model

D Computation of $q(t|\mathbf{x})$

Please note that t denotes the trial number i.e. halting step ($t \in \mathbb{N}^+$)[3].

Again, the following decomposition of $q(t|\mathbf{x})$ holds

$$q_\phi(t|\mathbf{x}) = \sum_{T=t}^{\infty} q_\phi(t|\mathbf{x}, T)\; p(T)\ . \tag{29}$$

During training instead of using an infinite horizon we choose a certain fixed horizon H for T. The discrete probabilities $q(t|\mathbf{x})$ are then computed as follows:

$$\begin{aligned} q_\phi(t=1|\mathbf{x}) &\geq \sum_{T=1}^{H} q_\phi(t=1|\mathbf{x}, T)\; p(T) \\ q_\phi(t=1|\mathbf{x}) &\geq q_\phi(t=1|\mathbf{x}, T=1)\; p(T=1) \\ &\quad + q_\phi(t=1|\mathbf{x}, T=2)\; p(T=2) \\ &\quad \vdots \\ &\quad + q_\phi(t=1|\mathbf{x}, T=H)\; p(T=H) \end{aligned} \tag{30}$$

Note, the value of $q_\phi(t=1|\mathbf{x}, T=1)$ is always equal to 1 because we use the Softmax function to calculate probabilities from RNN output values q_t which are generated up to horizon $T=t$.

[3] where $\mathbb{N}^+$ denotes the set of natural numbers without 0

$$
\begin{aligned}
q_\phi(t=2|\mathbf{x}) &\geq \sum_{T=2}^{H} q_\phi(t=2|\mathbf{x},T)\; p(T) \\
q_\phi(t=2|\mathbf{x}) &\geq q_\phi(t=2|\mathbf{x},T=2)\; p(T=2) \\
&\vdots \\
&\quad + q_\phi(t=2|\mathbf{x},T=H)\; p(T=H)
\end{aligned}
\tag{31}
$$