Learning Anytime Predictions in Neural Networks via Adaptive Loss Balancing

Hanzhang Hu
School of Computer Science
Carnegie Mellon University
hanzhang@cs.cmu.edu

Debadeepta Dey
Microsoft Research
dedey@microsoft.com

Martial Hebert
School of Computer Science
Carnegie Mellon University
hebert@cs.cmu.edu

J. Andrew Bagnell
School of Computer Science
Carnegie Mellon University
dbagnell@cs.cmu.edu

Abstract

This work considers the trade-off between accuracy and test-time computational cost of deep neural networks (DNNs) via *anytime* predictions from auxiliary predictions. Specifically, we optimize auxiliary losses jointly in an *adaptive* weighted sum, where the weights are inversely proportional to average of each loss. Intuitively, this balances the losses to have the same scale. We demonstrate theoretical considerations that motivate this approach from multiple viewpoints, including connecting it to optimizing the geometric mean of the expectation of each loss, an objective that ignores the scale of losses. Experimentally, the adaptive weights induce more competitive anytime predictions on multiple recognition data-sets and models than non-adaptive approaches including weighing all losses equally. In particular, anytime neural networks (ANNs) can achieve the same accuracy faster using adaptive weights on a small network than using static constant weights on a large one. For problems with high performance saturation, we also show a sequence of exponentially deepening ANNs can achieve near-optimal anytime results at any budget, at the cost of a const fraction of extra computation.

1 Introduction

Recent years have seen advancement in visual recognition tasks by increasingly accurate convolutional neural networks, from AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) and VGG (Simonyan and Zisserman, 2015), to ResNet (He et al., 2016), ResNeXt (Xie et al., 2017), and DenseNet (Huang et al., 2017). As models become more accurate and computationally expensive, it becomes more difficult for applications to choose between slow predictors with high accuracy and fast predictors with low accuracy. Some applications also desire multiple trade-offs between computation and accuracy, because they have computational budgets that may vary at test time. E.g., web servers for facial recognition or spam filtering may have higher load during the afternoon than at midnight. Autonomous vehicles need faster object detection when moving rapidly than when it is stationary. Furthermore, real-time and latency sensitive applications may desire fast predictions on easy samples and slow but accurate predictions on difficult ones.

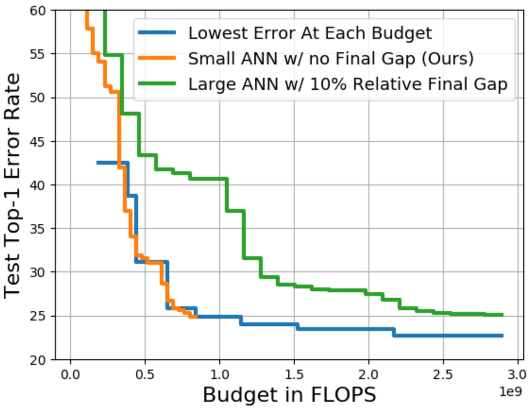


Figure 1: The common ANN training strategy increases final errors from the optimal (green vs. blue), which decreases exponentially slowly. By learning to focus more on the final auxiliary losses, the proposed adaptive loss weights make a small ANN (orange) to outperform a large one (green) that has non-adaptive weights.

An **anytime predictor** (Horvitz, 1987; Boddy and Dean, 1989; Zilberstein, 1996; Grubb and Bagnell, 2012; Huang et al., 2018) can automatically trade off between computation and accuracy. For each test sample, an anytime predictor produces a fast and crude initial prediction and continues to refine it as budget allows, so that at any test-time budget, the anytime predictor has a valid result for the sample, and the more budget is spent, the better the prediction. Anytime predictors are different from cascaded predictors (Viola and Jones, 2001; Xu et al., 2014; Cai, Saberian, and Vasconcelos, 2015; Bolukbasi et al., 2017; Guan et al., 2017) for **budgeted prediction**, which aim to minimize **average test-time computational cost** without sacrificing average accuracy: a different task (with relation to anytime prediction). Cascades achieve this by early exiting on easy samples to save computation for difficult ones, but cascades cannot incrementally improve individual samples after an exit. Furthermore, early exit policy of cascades can be combined with existing anytime predictors (Bolukbasi et al., 2017; Guan et al., 2017). Hence, we consider cascades to be orthogonal to anytime predictions.

This work studies how to convert well-known DNN ar-

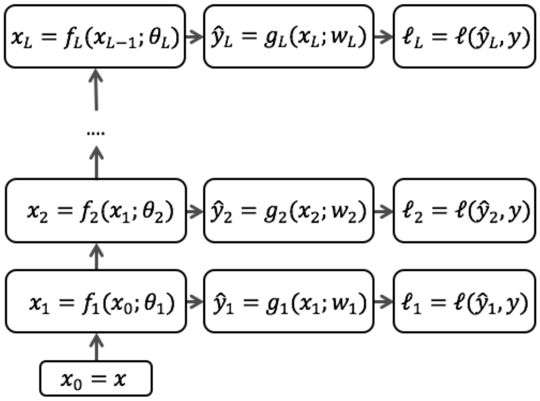

$x_L = f_L(x_{L-1}; \theta_L)$
$\hat{y}_L = g_L(x_L; w_L)$
$\ell_L = \ell(\hat{y}_L, y)$
....
$x_2 = f_2(x_1; \theta_2)$
$\hat{y}_2 = g_2(x_2; w_2)$
$\ell_2 = \ell(\hat{y}_2, y)$
$x_1 = f_1(x_0; \theta_1)$
$\hat{y}_1 = g_1(x_1; w_1)$
$\ell_1 = \ell(\hat{y}_1, y)$
$x_0 = x$

Figure 2: Anytime neural networks contain auxiliary predictions and losses, $\hat{y}_i$ and ℓ_i, for intermediate feature unit f_i.

chitectures to produce competitive anytime predictions. We form anytime neural networks (ANNs) by appending auxiliary predictions and losses to DNNs, as we will detail in Sec. 3 and Fig. 2. Inference-time prediction then can be stopped at the latest prediction layer that is within the budget. Note that this work deals with the case where it is **not known apriori** where the interrupt during inference time will occur. We define the optimal at each auxiliary loss as the result from training the ANN only for that loss to convergence. Then our objective is to have near-optimal final predictions and competitive early ones. Near-optimal final accuracy is imperative for anytime predictors, because, as demonstrated in Fig. 1, accuracy gains are often exponentially more expensive as model sizes grow, so that reducing 1% error rate could take 50% extra computation. Unfortunately, existing anytime predictors often optimize the anytime losses in static weighted sums (Lee et al., 2015; Zamir et al., 2017; Huang et al., 2018) that poorly optimize final predictions, as we will show in Sec. 3 and Sec. 5.

Instead, we optimize the losses in an **adaptive** weighted sum, where the weight of a loss is inversely proportional to the empirical mean of the loss on the training set. Intuitively, this normalizes losses to have the same scale, so that the optimization leads each loss to be about the same relative to its optimal. We provide multiple theoretical considerations to motivate such weights. First of all, when the losses are mean square errors, our approach is maximizing the likelihood of a model where the prediction targets have Gaussian noises. Secondly, inspired by the maximum likelihood estimation, we optimize the model parameters and the loss weights jointly, with log-barriers on the weights to avoid the trivial solution of zero weights. Finally, we find the joint optimization equivalent to optimizing the geometric mean of the expected training losses, an objective that treats the relative improvement of each loss equally. Empirically, we show on multiple models and visual recognition data-sets that the proposed adaptive weights outperform natural, non-adaptive weighting schemes as follows. We compare small ANNs using our adaptive weights against ANNs that are $50 \sim 100\%$ larger but use non-adaptive weights. The small ANNs can reach the same final accuracy as the larger ones, and reach each accuracy level faster.

Early and late accuracy in an ANN are often anti-correlated (e.g., Fig. 7 in (Huang et al., 2018) shows ANNs with better final predictions have worse early ones). To mitigate this *fundamental* issue we propose to assemble ANNs of exponentially increasing depths. If ANNs are near-optimal in a late fraction of their layers, the exponential ensemble only pays a constant fraction of additional computation to be near-optimal at every test-time budget. In addition, exponential ensembles outperform linear ensembles of networks, which are commonly used baselines for existing works (Zamir et al., 2017; Huang et al., 2018). In summary our contributions are:

- We derive an adaptive weight scheme for training losses in ANNs from multiple theoretical considerations, and show that experimentally this scheme achieves near-optimal final accuracy *and* competitive anytime ones on multiple data-sets and models.
- We assemble ANNs of exponentially increasing depths to achieve near-optimal anytime predictions at every budget at the cost of a constant fraction of additional consumed budget.

2 Related Works

Meta-algorithms for anytime and budgeted prediction. Anytime and budgeted prediction has a rich history in learning literature. (Weinberger et al., 2009; Xu, Weinberger, and Chapelle, 2012; Xu et al., 2013) sequentially generate features to empower the final predictor. (Reyzin, 2011; Grubb and Bagnell, 2012; Hu et al., 2016) apply boosting and greedy methods to order feature and predictor computation. (Karayev et al., 2012; Odena, Lawson, and Olah, 2017) form Markov Decision Processes for computation of weak predictors and features, and learn policies to order them. However, these meta-algorithms are not easily compatible with complex and accurate predictors like DNNs, because the anytime predictions without DNNs are inaccurate, and there are no intermediate results during the computation of the DNNs. Cascade designs for budgeted prediction (Viola and Jones, 2001; Lefakis and Fleuret, 2010; Chen et al., 2012; Xu et al., 2014; Cai, Saberian, and Vasconcelos, 2015; Nan and Saligrama, 2017; Bolukbasi et al., 2017; Guan et al., 2017) reduce the average test-time computation by early exiting on easy samples and saving computation for difficult ones. As cascades build upon existing anytime predictors, or combine multiple predictors, they are orthogonal to learning ANNs end-to-end.

Neural networks with early auxiliary predictions. Multiple works have addressed training DNNs with early auxiliary predictions for various purposes. (Lee et al., 2015; Szegedy et al., 2017; Zhao et al., 2017; Larsson, Maire, and Shakhnarovich, 2017) use them to regularize the networks for faster and better convergence. (Bengio et al., 2009; Zamir et al., 2017) set the auxiliary predictions from easy to hard for curriculum learning. (Xie and Tu, 2015; Chen and Koltun, 2017) make pixel level predictions in images, and find learning early predictions in coarse scales also improve the fine resolution predictions. (Huang et al., 2018) shows the crucial importance of maintaining multi-scale features for high quality early classifications. The above works use manually-tuned static weights to combine the auxiliary losses, or change the weights only once (Chen and Koltun, 2017). This work proposes adaptive weights to balance the losses to the same scales online, and provides multiple theoretical motivations. We empirically show adaptive losses induce better ANNs on multiple models, including the state-of-the-art anytime predictor for image recognition, MSDNet (Huang et al., 2018).

Model compression. Many works have studied how to compress neural networks. (Li et al., 2017; Liu et al., 2017) prune network weights and connections. (Hubara et al., 2016; Rastegari et al., 2016; Iandola et al., 2016) quantize weights within networks to reduce computation and memory footprint. (Wang et al., 2017; Veit and Belongie, 2017) dynamically skip network computation based on samples. (Ba and Caruana, 2014; Hinton, Vinyals, and Dean, 2014) transfer knowledge of deep networks into shallow ones by changing the training target of shallow networks. These works are orthogonal to ours, because they train a separate model for each trade-off between computation and accuracy, but we train a single model to handle all possible trade-offs.

3 Optimizing Anytime Predictors

As illustrated in Fig. 2, a feed-forward network consists of a sequence of transformations $f_1, ..., f_L$ of feature maps. Starting with the input feature map x_0, each subsequent feature map is generated by $x_i = f_i(x_{i-1})$. Typical DNNs use the final feature map x_L to produce predictions, and hence require the completion of the whole network for results. Anytime neural networks (ANNs) instead introduce auxiliary predictions and losses using the intermediate feature maps $x_1, ..., x_{L-1}$, and thus, have early predictions that are improving with computation.

Weighted sum objective. Let the intermediate predictions be $\hat{y}_i = g_i(x_i)$ for some function g_i, and let the corresponding expected loss be $\ell_i = E_{(x_0,y)\sim\mathcal{D}}[\ell(y, \hat{y}_i)]$, where $\mathcal{D}$ is the distribution of the data, and ℓ is some loss such as cross-entropy. Let θ be the parameter of the ANN, and define the optimal loss at prediction $\hat{y}_i$ to be $\ell_i* = \min_\theta \ell_i(\theta)$. Then the goal of anytime prediction is to seek a universal $\theta^* \in \cap_{i=1}^L \{\theta' : \theta' = \arg\min_\theta \ell_i(\theta)\}$. Such an ideal θ^* does not exist in general as this is a multi-objective optimization, which only has Pareto front, a set containing all solutions such that improving one ℓ_i necessitates degrading others. Finding all solutions in the Pareto front for ANNs is not practical or useful, since this requires training multiple models, but each ANN only runs one. Hence, following previous works on anytime models (Lee et al., 2015; Zamir et al., 2017; Huang et al., 2018), we optimize the losses in a weighted sum $\min_\theta \sum_{i=1}^L B_i \ell_i(\theta)$, where B_i is the weight of the loss ℓ_i. We call the choices of B_i *weight schemes*.

Static weight schemes. Previous works often use static weight schemes as part of their formulation. Lee et al. (2015); Xie and Tu (2015); Huang et al. (2018) use CONST scheme that sets $B_i = 1$ for all i. Zamir et al. (2017) use

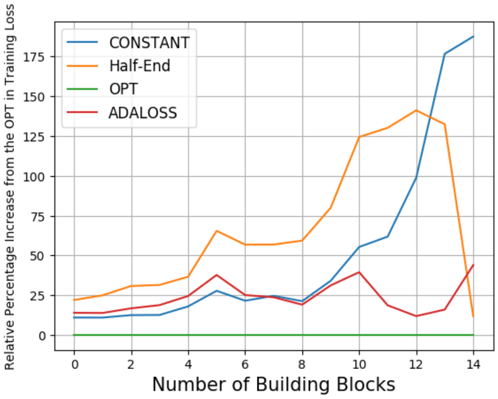


Figure 3: Relative Percentage Increase in Training Loss vs. depths (lower is better). CONST scheme is increasingly worse than the optimal at deep layers. AdaLoss performs about equally well on all layers in comparison to the OPT.

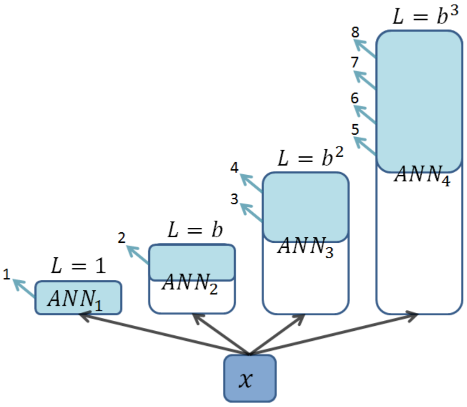


Figure 4: Ensemble of exponentially deepening anytime neural network (EANN) computes its ANNs in order of their depths. An anytime result is used if it is better than all previous ones on a validation set (layers in light blue).

LINEAR scheme that sets B_1 to B_L to linearly increase from 0.25 to 1. However, as we will show in Sec. 5.2, these static schemes not only cannot adjust weights in a data and model-dependent manner, but also may significantly degrade predictions at later layers.

Qualitative weight scheme comparison. Before we formally introduce our proposed adaptive weights, we first shed light on how existing static weights suffer. We experiment with a ResNet of 15 basic residual blocks on CIFAR100 (Krizhevsky, 2009) data-set (See Sec. 5 for dataset details). An anytime predictor is attached to each residual block, and we estimate the optimal performance (OPT) in training cross entropy of predictor i by training a network that has weight only on ℓ_i to convergence. Then for each weight scheme we train an ANN to measure the relative increase in training loss at each depth i from the OPT. In Fig. 3, we observe that the intuitive CONST scheme has high relative losses in late layers. This indicates that there is not enough weights in the late layers, though losses have the same B_i. We also note that balancing the weights is non-trivial. For instance, if we put half of the total weights in the final layer and distribute the other half evenly, we get the "Half-End" scheme. As expected, the final loss is improved, but this is at the cost of significant increases of early training losses. In contrast, the adaptive weight scheme that we propose next (AdaLoss), achieves roughly even relative increases in training losses automatically, and is much better than the CONST scheme in the late layers.

Adaptive Loss Balancing (AdaLoss). Given all losses are of the same form (cross-entropy), it may be surprising that better performance is achieved with differing weights. Because early features typically have less predictive power than later ones, early losses are naturally on a larger scale and possess larger gradients. Hence, if we weigh losses equally, early losses and gradients often dominate later ones, and the optimization becomes focused on the early losses. To automatically balance the weights among the losses of different scales, we propose an adaptive loss balancing scheme (AdaLoss). Specifically, we keep an exponential average of each loss $\hat{\ell}_i$ during training, and set $B_i \propto \frac{1}{\hat{\ell}_i}$. This is inspired by (Chen and Koltun, 2017), which scales the losses to the same scale *only once* during training, and provides a brief intuitive argument: the adaptive weights set the losses to be on the same scale. We next present multiple theoretical justifications for AdaLoss.

Before considering general cases, we first consider a simple example, where the loss function $\ell(y, \hat{y}) = \|y - \hat{y}\|_2^2$ is the square loss. For this example, we model each $y|x$ to be sampled from the multiplication of L independent Gaussian distributions, $\mathcal{N}(\hat{y}_i, \sigma_i^2 I)$ for $i = 1, ..., L$, where $\hat{y}_i(x; \theta)$ is the i^{th} prediction, and $\sigma_i^2 \in \mathbb{R}^+$, i.e., $Pr(y|x; \theta, \sigma_1^2, ..., \sigma_L^2) \propto \prod_{i=1}^{L} \frac{1}{\sqrt{\sigma_i^2}} \exp(-\frac{\|y-\hat{y}_i\|_2^2}{2\sigma_i^2})$. Then we compute the empirical expected log-likelihood for a maximum likelihood estimator (MLE):

$$\hat{E}\big[\ln(Pr(y|x))\big] \propto \hat{E}\big[\sum_{i=1}^{L}(-\frac{\|y-\hat{y}_i\|_2^2}{\sigma_i^2} - \ln \sigma_i^2)\big] \quad (1)$$

$$= \sum_{i=1}^{L}(-\frac{\tilde{\ell}_i}{\sigma_i^2} - \ln \sigma_i^2), \quad (2)$$

where $\hat{E}$ is averaging over samples, and $\tilde{\ell}_i$ is the empirical estimate of ℓ_i. If we fix θ and optimize over σ_i^2, we get $\sigma_i^2 = \tilde{\ell}_i$. As computing the empirical means is expensive over large data-sets, AdaLoss replaces $\tilde{\ell}_i$ with $\hat{\ell}_i$, the exponential moving average of the losses, and sets $B_i \propto \hat{\ell}_i^{-1} \approx \sigma_i^{-2}$ so as to solve the MLE online by jointly updating θ and B_i. We note that the naturally appeared $\ln \sigma_i^2$ terms in Eq. 2 are log-barriers preventing $B_i = 0$.

Inspired by this observation, we form the following joint optimization over θ and B_i for general losses without probability models:

$$\min_{\theta, B_1, ..., B_L} \sum_{i=1}^{L}(B_i \ell_i(\theta) - \lambda \ln B_i), \quad (3)$$

where $\lambda > 0$ is a hyper parameter to balance between the log-barriers and weighted losses. Under the optimal condition, $B_i = \frac{\lambda}{\ell_i}$. AdaLoss estimates this with $B_i \propto \hat{\ell}_i(\theta)^{-1}$. We can also eliminate B_i from Eq. 3 under the optimal condition, and we transform Eq. 3 to the following problem:

$$\min_{\theta} \sum_{i=1}^{L} \ln \ell_i(\theta). \quad (4)$$

This is equivalent to minimizing the geometric mean of the expected training losses, and it differs from minimizing the expected geometric mean of losses, as $\ln$ and expectation are not commutable. Eq. 4 discards any constant scaling of losses automatically discarded as constant offsets, so that the scale difference between the early and late losses are automatically reconciled. Geometric mean is also known as the canonical mean for multiple positive quantities of various scales. AdaLoss optimizes Eq. 4, since the objective gradient is $\sum_{i=1}^{L} \frac{\nabla \ell_i(\theta)}{\ell_i(\theta)}$. AdaLoss wants to weigh each $\ell_i(\theta)$ by exactly $\frac{1}{\ell_i(\theta)}$, and estimates the weight by $\frac{1}{\hat{\ell}_i(\theta)}$. This concludes our theoretical considerations for AdaLoss.

4 Sequence of Exponentially Deepening Anytime Neural Networks (EANN)

In practice, we often observe ANNs using AdaLoss to be much more competitive in their later half than the early half on validation sets, such as in Table. 1 of Sec. 5.2. Fortunately, we can leverage this effect to form competitive anytime predictors at every budget, with a constant fraction of additional computation. Specifically, we assemble ANNs whose depths grow exponentially. Each ANN only starts computing if the smaller ones are finished, and its predictions are used if they are better than the best existing ones in validation. We call this ensemble an **EANN**, as illustrated in Fig. 4. An EANN only delays the computation of any large ANN by at most a constant fraction of computation, because the earlier networks are exponentially smaller. Hence, if each ANN is near-optimal in later predictions, then we can achieve near-optimal accuracy at any test-time interruption, with the extra computation. Formally, the following proposition characterizes the exponential base and the increased computational cost.

Proposition 4.1. *Let $b > 1$. Assume for any L, any ANN of depth L has competitive anytime prediction at depth $i > \frac{L}{b}$ against the optimal of depth i. Then after B layers of computation, EANN produces anytime predictions that are competitive against the optimal of depth $\frac{B}{C}$ for some $C > 1$, such that $\sup_B C = 2 + \frac{1}{b-1}$, and C has expectation $E_{B \sim uniform(1,L)}[C] \leq 1 - \frac{1}{2b} + \frac{1+\ln(b)}{b-1}$.*

This proposition says that an EANN is competitive at any budget B against the optimal of the cost $\frac{B}{C}$. Furthermore, the stronger each anytime model is, i.e., the larger b becomes, the smaller the computation inflation, C, is: as b approaches ∞, $\sup_B C$, shrinks to 2, and $E[C]$, shrinks to 1. Moreover, if we have M number of parallel workers instead of one, we can speed up EANNs by computing ANNs in parallel in a first-in-first-out schedule, so that we effectively increase the constant b to b^M for computing C. It is also worth noting that if we form the sequence using regular networks instead of ANNs, then we will lose the ability to output frequently, since at budget B, we only produce $\Theta(\log(B))$ intermediate predictions instead of the $\Theta(B)$ predictions in an EANN. We will further have a larger cost inflation, C, such that $\sup_B C \geq 4$ and $E[C] \geq 1.5 + \sqrt{2} \approx 2.91$, so that the average cost inflation is at least about 2.91. We defer the proofs to the appendix.

5 Experiments

We list the key questions that our experiments aim to answer.

- How do anytime predictions trained with adaptive weights compare against those trained with static constant weights (over different architectures)? (Sec. 5.2)
- How do underlying DNN architectures affect ANNs? (Sec. 5.2)
- How can sub-par early predictions in ANNs be mitigated by ANN ensembles? (Sec. 5.3)
- How does data-set difficulty affect the adaptive weights scheme? (Sec. 5.4)

5.1 Data-sets and Training Details

Data-sets. We experiment on CIFAR10, CIFAR100 (Krizhevsky, 2009), SVHN (Netzer et al., 2011)[1] and ILSVRC (Russakovsky et al., 2015)[2].

Training details. We optimize the models using stochastic gradient descent, with initial learning rate of 0.1, momentum of 0.9 and a weight decay of 1e-4. On CIFAR and SVHN, we divide the learning rate by 10 at 1/2 and 3/4 of the total epochs. We train for 300 epochs on CIFAR and 60 epochs on SVHN. On ILSVRC, we train for 90 epochs, and divide the learning rate by 10 at epoch 30 and 60. We evaluate test error using single-crop.

Base models. We compare our proposed AdaLoss weights against the intuitive CONST weights. On CIFAR and SVHN, we also compare AdaLoss against LINEAR and OPT, defined in Sec. 3. We evaluate the weights on multiple models including ResNet (He et al., 2016) and DenseNet (Huang et al., 2017), and MSDNet (Huang et al., 2018). For ResNet and DenseNet, we augment them with auxiliary predictors and losses, and call the resulting models ResANN and DenseANN, and defer the details of these models to the appendix Sec. B.

	1/4	1/2	3/4	1
OPT	0.00	0.00	0.00	0.00
CONST	**15.07**	16.40	18.76	18.90
LINEAR	25.67	13.02	12.97	12.65
ADALOSS	32.99	**9.97**	**3.96**	**2.73**

Table 1: Average relative percentage increase in error from the OPT on CIFAR and SVHN at 1/4, 1/2, 3/4 and 1 of the total cost. E.g., the bottom right entry means that if OPT has a 10% final error rate, then AdaLoss has about 10.27%.

	1/4	1/2	3/4	1
ResANN50+C	**54.34**	35.61	27.23	25.14
ResANN50+A	54.98	**34.92**	**26.59**	**24.42**
DenseANN169+C	48.15	45.00	29.09	25.60
DenseANN169+A	**47.17**	**44.64**	**28.22**	**24.07**
MSDNet38	**33.9**	28.0	25.7	24.3
MSDNet38+A	35.75	28.04	25.82	**23.99**

Table 2: Test error rates at different fraction of the total costs on ResANN50, DenseANN169, and MSDNet38 on ILSVRC. The postfix +C and +A stand for CONST and AdaLoss respectively. Published results of MSDNet38 (Huang et al., 2018) uses CONST.

5.2 Weight Scheme Comparisons

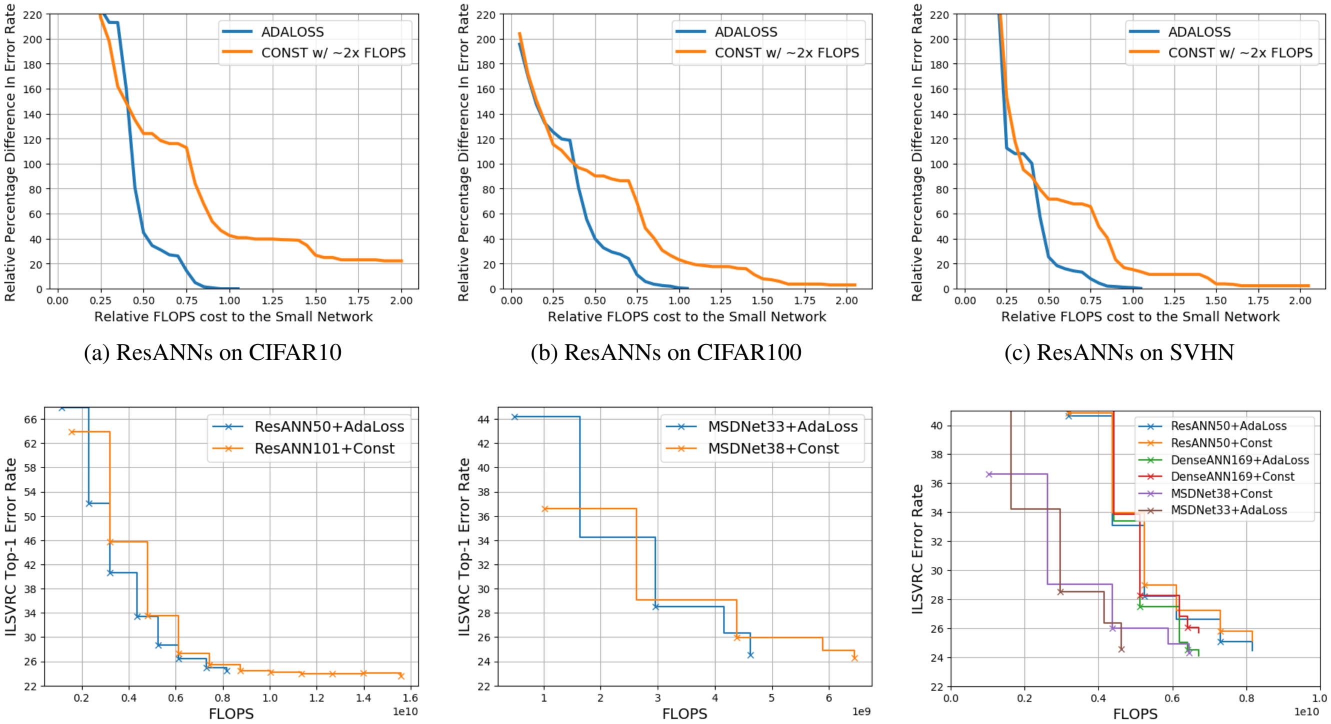
AdaLoss vs. CONST on the same models. Table 1 presents the average relative test error rate increase from OPT on 12 ResANNs on CIFAR10, CIFAR100 and SVHN[3]. As training an OPT for each depth is too expensive, we instead report the average relative comparison at 1/4, 1/2, 3/4, and 1 of the total ANN costs. We observe that the CONST scheme makes $15 \sim 18\%$ more errors than the OPT, and the relative gap widens at later layers. The LINEAR scheme also has about 13% relative gap in later layers. In contrast, AdaLoss enjoys small performance gaps in the later half of layers. On ILSVRC, we compare AdaLoss against CONST on ResANN50, DenseANN169, and MSDNet38, which have similar final errors and total computational costs (See Fig. 5f). In Table 2, we observe the trade-offs between early and late accuracy on ResANN50 and MSDNet38. Furthermore, DenseANN169 performs *uniformly* better with AdaLoss than with CONST. Since comparing the weight schemes requires evaluating ANNs at multiple budget limits, and AdaLoss and CONST outperform each other at a significant fraction of depths on most of our experiments, we consider the two schemes *incomparable on the same model*.

Small networks with AdaLoss vs. large ones with CONST. Our previous comparison between AdaLoss and CONST on the same models is not fully conclusive, since each scheme can outperform the other at a significant por-

[1] Both CIFAR data-sets consist of 32x32 colored images. CIFAR10 and CIFAR100 have 10 and 100 classes, and each have 50000 training and 10000 testing images. We held out the last 5000 training samples in CIFAR10 and CIFAR100 for validation; the same parameters are then used in other models. We adopt the standard augmentation from Lee et al. (2015); He et al. (2016). SVHN contains around 600000 training and around 26032 testing 32x32 images of numeric digits from the Google Street Views. We adopt the same pad-and-crop augmentations of CIFAR for SVHN, and also add Gaussian blur.

[2] ILSVRC2012 (Russakovsky et al., 2015) is a visual recognition data-set containing around 1.2 million natural and 50000 validation images for 1000 classes. We report the top-1 error rates on the validation set using a single-crop of size 224x224, after scaling the smaller side of the image to 256, following (He et al., 2016).

[3] The 12 models are named by (n, c) drawn from $\{7, 9, 13, 17, 25\} \times \{16, 32\}$ and $\{(9, 64), (9, 128)\}$, where n represents the number of residual units in each of the three blocks of the network, and c is the filter size of the first convolution.



(a) ResANNs on CIFAR10

(b) ResANNs on CIFAR100

(c) ResANNs on SVHN

(d) ResANNs on ILSVRC

(e) MSDNet on ILSVRC

(f) ANNs comparison on ILSVRC

Figure 5: **(a-e)** Comparing small networks with AdaLoss versus big ones using CONST. With AdaLoss, the small networks achieve the same accuracy levels faster than large networks with CONST. **(f)** ANNs performance are mostly decided by underlying models, but AdaLoss is beneficial regardless models.

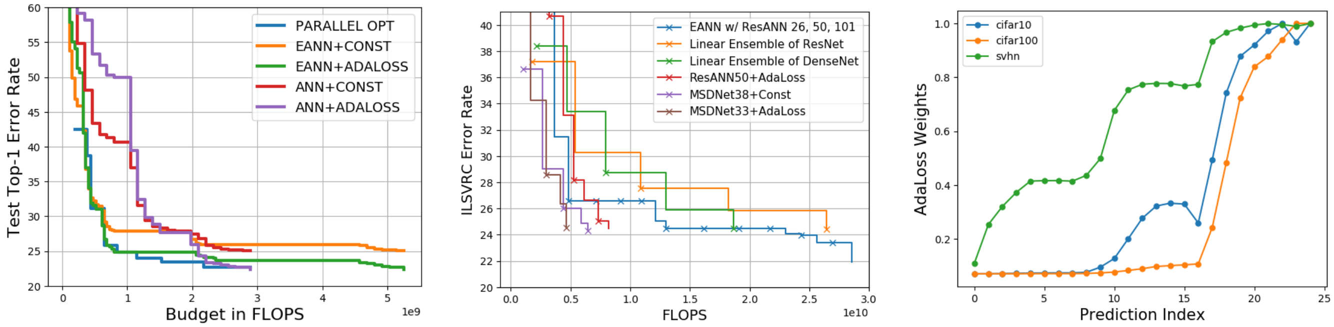
tion of the total cost. To address this, we set the final error rate, model architecture type, and the filter size c as constants, and vary the model depths so that AdaLoss and CONST reach the target final error rate. Then we compare the early predictions and the costs of models. On each of CIFAR10, 100 and SVHN, we compare six pairs of ResANNs, where the CONST uses twice the computation as AdaLoss[4]. Fig. 5a, 5b, and 5c show the averaged relative comparisons[5], and they show that the small ANNs with AdaLoss are better anytime predictors than the large ones with CONST, because both models have the same final accuracy (on CIFAR10, the small ones are even better), and the small models reach the same error rates faster than the large ones. We have similar observations on ILSVRC using ResANNs and MSDNets in Fig. 5d and Fig. 5e. For instance, MSDNet (Huang et al., 2018) is the state-of-the-art anytime predictor. The published MSDNet38 uses CONST, and has 24.3% error rate using 6.6e9 total FLOPS in convolutions. By switching to AdaLoss, we improve a much smaller MSDNet33 (details in the appendix), which costs 4.5e9 FLOPS, to reach 24.5% final error. The two models also have similar early errors.

AdaLoss can reach the same accuracies with similar or smaller costs than CONST, because in practice, a linear decrease in final error rate may often require an exponential increase in total computation, and CONST degrades the final performances significantly (Table 1). Since AdaLoss requires much smaller models than CONST to reach the same final errors, and with a fixed final error rate, AdaLoss reaches each early error rate with less or similar cost, we conclude that AdaLoss is the better scheme for anytime predictions.

Various base networks on ILSVRC. We compare ResANNs, DenseANNs and MSDNets that have final error rate of near 24% in Fig. 5f, and observe that the anytime performance is mostly decided by the specific underlying model. MSDNets are more cost-effective than DenseANNs, which in turn are better than ResANNs. However, AdaLoss is helpful regardless of underlying model. Both ResANN50 and DenseANN169 see improvements switching from CONST to AdaLoss, which is also shown in Table 2. Thanks to AdaLoss, DenseANN169 achieves the same final error using similar FLOPS as the original published results of MSDNet38 (Huang et al., 2018). This suggests that Huang et al. (2018) improve over DenseANNs by having better early predictions without sacrificing the final cost efficiency via impressive architecture insight. AdaLoss brings a complementary improvement to MSDNets, as it enables smaller MSDNets to reach the final error rates of bigger MSDNets, while

[4] AdaLoss takes (n, c) from $\{7, 9, 13\} \times \{16, 32\}$, and CONST takes (n, c) from $\{13, 17, 25\} \times \{16, 32\}$.

[5] The relative plots pivot at the final predictor from AdaLoss, e.g., the location (0.5, 200) means having half the computation and 200% extra relative errors than the final predictor from AdaLoss



(a) EANNs on CIFAR100

(b) EANN on ILSVRC

(c) AdaLoss Weights on three data-sets

Figure 6: **(a)** EANN performs better if the ANNs use AdaLoss instead of CONST. **(b)** EANN outperforms linear ensembles of DNNs on ILSVRC. **(c)** The learned adaptive weights of the same model on three data-sets.

having similar or better early predictions.

5.3 EANN: Closing Early Performance Gaps by Delaying Final Predictions.

EANNs on CIFAR100. In Fig. 6a, we assemble ResANNs to form EANNs[6] on CIFAR100 and make three observations. First, EANNs are better than the ANN in early computation, because the ensembles dedicate early predictions to small networks. Even though CONST has the best early predictions as in Table 1, it is still better to deploy small networks. Second, because the final prediction of each network is kept for a long period, AdaLoss leads to significantly better EANNs than CONST does, thanks to the superior final predictions from AdaLoss. Finally, though EANNs delay computation of large networks, it actually appears closer to the OPT, because of accuracy saturation. Hence, EANNs should be considered when performance saturation is severe.

EANN on ILSVRC. Huang et al. (2018) and Zamir et al. (2017) use ensembles of networks of linearly growing sizes as baseline anytime predictors. However, in Fig. 6b, an EANN using ResANNs of depths 26, 50 and 101 outperforms the linear ensembles of ResNets and DenseNets significantly on ILSVRC. In particular, this drastically reduces the gap between ensembles and the state-of-the-art anytime predictor MSDNet (Huang et al., 2018). Comparing ResANN 50 and the EANN, we note that the EANN achieves better early accuracy but delays final predictions. As the accuracy is not saturated by ResANN 26, the delay appears significant. Hence, EANNs may not be the best when the performance is not saturated or when the constant fraction of extra cost is critical.

5.4 Data-set Difficulty versus Adaptive Weights

In Fig. 6c, we plot the final AdaLoss weights of the same ResANN model (25,32) on CIFAR10, CIFAR100, and SVHN to study the effects of the data-sets on the weights. We observe that from the easiest data-set, SVHN, to the hardest, CIFAR100, the weights are more concentrated on the final layers. This suggests that AdaLoss can automatically decide that harder data-sets need more concentrated final weights to have near-optimal final performance, whereas on easy data-sets, more efforts are directed to early predictions. Hence, AdaLoss weights may provide information for practitioners to design and choose models based on data-sets.

6 Conclusion and Discussion

This work devises simple adaptive weights, AdaLoss, for training anytime predictions in DNNs. We provide multiple theoretical motivations for such weights, and show experimentally that adaptive weights enable small ANNs to outperform large ANNs with the commonly used non-adaptive constant weights. Future works on adaptive weights includes examining AdaLoss for multi-task problems and investigating its "first-order" variants that normalize the losses by individual gradient norms to address unknown offsets of losses as well as the unknown scales. We also note that this work can be combined with orthogonal works in early-exit budgeted predictions (Guan et al., 2017; Bolukbasi et al., 2017) for saving average test computation.

Acknowledgements

This work was conducted in part through collaborative participation in the Robotics Consortium sponsored by the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement W911NF-10-2-0016.

References

Ba, L. J., and Caruana, R. 2014. Do deep nets really need to be deep? In *Proceedings of NIPS*.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *ICML*.

Boddy, M., and Dean, T. 1989. Solving time-dependent planning problems. In *IJCAI*.

[6] The ResANNs have $c = 32$ and $n = 7, 13, 25$, so that they form an EANN with an exponential base $b \approx 2$. By proposition 4.1, the average cost inflation is $E[C] \approx 2.44$ for $b = 2$, so that the EANN should compete against the OPT of $n = 20$, using 2.44 times of original costs.

Bolukbasi, T.; Wang, J.; Dekel, O.; and Saligrama, V. 2017. Adaptive neural networks for fast test-time prediction. In *ICML*.

Cai, Z.; Saberian, M. J.; and Vasconcelos, N. 2015. Learning Complexity-Aware Cascades for Deep Pedestrian Detection. In *ICCV*.

Chen, Q., and Koltun, V. 2017. Photographic image synthesis with cascaded refinement networks. In *ICCV*.

Chen, M.; Weinberger, K. Q.; Chapelle, O.; Kedem, D.; and Xu, Z. 2012. Classifier Cascade for Minimizing Feature Evaluation Cost. In *AISTATS*.

Grubb, A., and Bagnell, J. A. 2012. SpeedBoost: Anytime Prediction with Uniform Near-Optimality. In *AISTATS*.

Guan, J.; Liu, Y.; Liu, Q.; and Peng, J. 2017. Energy-efficient amortized inference with cascaded deep classifiers. In *arxiv preprint, arxiv.org/abs/1710.03368*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Hinton, G.; Vinyals, O.; and Dean, J. 2014. Distilling the knowledge in a neural network. In *Deep Learning and Representation Learning Workshop, NIPS*.

Horvitz, E. J. 1987. Reasoning about beliefs and actions under computational resource constraints. In *UAI*.

Hu, H.; Grubb, A.; Hebert, M.; and Bagnell, J. A. 2016. Efficient feature group sequencing for anytime linear prediction. In *UAI*.

Huang, G.; Liu, Z.; Weinberger, K. Q.; and van der Maaten, L. 2017. Densely connected convolutional networks. In *CVPR*.

Huang, G.; Chen, D.; Li, T.; Wu, F.; van der Maaten, L.; and Weinberger, K. Q. 2018. Multi-scale dense convolutional networks for efficient prediction. In *ICLR*.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks. In *NIPS*.

Iandola, F. N.; Han, S.; Moskewicz, M. W.; Ashraf, K.; Dally, W. J.; and Keutzer, K. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. In *arxiv preprint: 1602.07360*.

Karayev, S.; Baumgartner, T.; Fritz, M.; and Darrell, T. 2012. Timely Object Recognition. In *NIPS*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, 1097–1105.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report.

Larsson, G.; Maire, M.; and Shakhnarovich, G. 2017. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*.

Lee, C.-Y.; Xie, S.; Gallagher, P. W.; Zhang, Z.; and Tu, Z. 2015. Deeply-supervised nets. In *AISTATS*.

Lefakis, L., and Fleuret, F. 2010. Joint Cascade Optimization Using a Product of Boosted Classifiers. In *NIPS*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning filters for efficient convnets. In *ICLR*.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *arxiv preprint:1708.06519*.

Nan, F., and Saligrama, V. 2017. Dynamic model selection for prediction under a budget. In *NIPS*.

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Odena, A.; Lawson, D.; and Olah, C. 2017. Changing model behavior at test-time using reinforcement. In *Arxive preprint: 1702.07780*.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

Reyzin, L. 2011. Boosting on a budget: Sampling for feature-efficient prediction. In *ICML*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV*.

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*.

Veit, A., and Belongie, S. 2017. Convolutional networks with adaptive computation graphs. *arXiv preprint arXiv:1711.11503*.

Viola, P. A., and Jones, M. J. 2001. Rapid Object Detection using a Boosted Cascade of Simple Features. In *CVPR*.

Wang, X.; Yu, F.; Dou, Z.-Y.; and Gonzalez, J. E. 2017. Skipnet: Learning dynamic routing in convolutional networks. *arXiv preprint arXiv:1711.09485*.

Weinberger, K.; Dasgupta, A.; Langford, J.; Smola, A.; and Attenberg, J. 2009. Feature Hashing for Large Scale Multitask Learning. In *ICML*.

Xie, S., and Tu, Z. 2015. Holistically-nested edge detection. In *ICCV*.

Xie, S.; Girshick, R.; Dollr, P.; Tu, Z.; and He, K. 2017. Aggregated residual transformations for deep neural networks. In *CVPR*.

Xu, Z.; Kusner, M.; Huang, G.; and Weinberger, K. Q. 2013. Anytime Representation Learning. In *ICML*.

Xu, Z.; Kusner, M. J.; Weinberger, K. Q.; Chen, M.; and Chapelle, O. 2014. Classifier cascades and trees for minimizing feature evaluation cost. *JMLR*.

Xu, Z.; Weinberger, K.; and Chapelle, O. 2012. The Greedy Miser: Learning under Test-time Budgets. In *ICML*.

Zamir, A. R.; Wu, T.-L.; Sun, L.; Shen, W.; Malik, J.; and Savarese, S. 2017. Feedback networks. In *CVPR*.

Zhao, H.; Shi, J.; Qi, X.; Wang, X.; and Jia, J. 2017. Pyramid scene parsing network. In *CVPR*.

Zilberstein, S. 1996. Using anytime algorithms in intelligent systems. *AI Magazine* 17(3):73–83.

A Sketch of Proof of Proposition 4.1

Proof. For each budget consumed x, we compute the cost x' of the optimal that EANN is competitive against. The goal is then to analyze the ratio $C = \frac{x}{x'}$. The first ANN in EANN has depth 1. The optimal and the result of EANN are the same. Now assume EANN is on depth z of ANN number $n + 1$ for $n \geq 0$, which has depth b^n.

(Case 1) For $z \leq b^{n-1}$, EANN reuse the result from the end of ANN number n. The cost spent is $x = z + \sum_{i=0}^{n-1} b^i = z + \frac{b^n-1}{b-1}$. The optimal we compete has cost of the last ANN, which is b^{n-1} The ratio satisfies:

$$\begin{aligned} C = x/x' &= \frac{z}{b^{n-1}} + 1 + \frac{1}{b-1} - \frac{1}{b^{n-1}(b-1)} \\ &\leq 2 + \frac{1}{b-1} - \frac{1}{b^{n-1}(b-1)} < 2 + \frac{1}{b-1}. \end{aligned}$$

Furthermore, since C increases with z,

$$\begin{aligned} & E_{z\sim Uniform(0,b^{n-1})}[C] \\ &\leq b^{1-n} \int_0^{b^{n-1}} zb^{1-n} + 1 + \frac{1}{b-1}\, dz \\ &= 1.5 + \frac{1}{b-1}. \end{aligned}$$

(Case 2) For $b^{n-1} < z \leq b^n$, EANN outputs anytime results from ANN number $n + 1$ at depth z. The cost is still $x = z + \frac{b^n-1}{b-1}$. The optimal competitor has cost $x' = z$. Hence the ratio is

$$\begin{aligned} C = x/x' &= 1 + \frac{b^n - 1}{z(b-1)} \\ &\leq 2 + \frac{1}{b-1} - \frac{1}{b^{n-1}(b-1)} < 2 + \frac{1}{b-1}. \end{aligned}$$

Furthermore, since C decreases with z,

$$\begin{aligned} & E_{z\sim Uniform(b^{n-1},b^n)}[C] \\ &\leq 1 + \frac{1}{b^n - b^{n-1}} \int_{b^{n-1}}^{b^n} \frac{b^n-1}{z(b-1)}\, dz \\ &= 1 + \frac{(b - b^{1-n}) \ln b}{(b-1)^2} \\ &< 1 + \frac{b \ln b}{(b-1)^2} \end{aligned}$$

Finally, since case 1 and case 2 happen with probability $\frac{1}{b}$ and $(1 - \frac{1}{b})$, we have

$$\sup_B C = 2 + \frac{1}{b-1} \tag{5}$$

and

$$E_{B\sim Uniform(0,L)}[C] \leq 1 - \frac{1}{2b} + \frac{1}{b-1} + \frac{\ln b}{b-1}. \tag{6}$$

We also note that with large b, $\sup_B C \to 2$ and $E[C] \to 1$ from above. □

If we form a sequence of regular networks that grow exponentially in depth instead of ANN, then the worst case happen right before a new prediction is produced. Hence the ratio between the consumed budget and the cost of the optimal that the current anytime prediction can compete, C, right before the number $n + 1$ network is completed, is

$$\frac{\sum_{i=1}^n b^i}{b^{n-1}} \xrightarrow{n\to\infty} \frac{b^2}{b-1} = 2 + (b-1) + \frac{1}{b-1} \geq 4.$$

Note that $(b - 1) + \frac{1}{b-1} \geq 2$ and the inequality is tight at $b = 2$. Hence we know $\sup_B C$ is at least 4. Furthermore, the expected value of C, assume B is uniformly sampled such that the interruption happens on the $(n + 1)^{th}$ network, is:

$$\begin{aligned} E[C] &= \frac{1}{b^n} \int_0^{b^n} \frac{x + \frac{b^n-1}{b-1}}{b^{n-1}}\, dx \\ &\xrightarrow{n\to\infty} 1.5 + \frac{b-1}{2} + \frac{1}{b-1} \geq 1.5 + \sqrt{2} \approx 2.91. \end{aligned}$$

The inequality is tight at $b = 1 + \sqrt{2}$. With large n, since almost all budgets are consumed by the last few networks, we know the overall expectation $E_{B\sim Uniform(0,L)}[C]$ approaches $1.5 + \frac{b-1}{2} + \frac{1}{b-1}$, which is at least $1.5 + \sqrt{2}$.

B Implementation Details of ANNs

CIFAR and SVHN ResANNs. For CIFAR10, CIFAR100 (Krizhevsky, 2009), and SVHN (Netzer et al., 2011), ResANN follow (He et al., 2016) to have three blocks, each of which has n residual units. Each of such basic residual units consists of two 3x3 convolutions, which are interleaved by BN-ReLU. A pre-activation (BN-ReLU) is applied to the input of the residual units. The result of the second 3x3 conv and the initial input are added together as the output of the unit. The auxiliary predictors each applies a BN-ReLU and a global average pooling on its input feature map, and applies a linear prediction. The auxiliary loss is the cross-entropy loss, treating the linear prediction results as logits. For each (n, c) pair such that $n < 25$, we set the anytime prediction period s to be 1, i.e., every residual block leads to an auxiliary prediction. We set the prediction period $s = 3$ for $n = 25$.

ResANNs on ILSVRC. Residual blocks for ILSVRC are bottleneck blocks, which consists of a chain of 1x1 conv, 3x3 conv and 1x1 conv. These convolutions are interleaved by BN-ReLU, and pre-activation BN-ReLU is also applied. Again, the output of the unit is the sum of the input feature map and the result of the final conv. ResANN50 and 101 are augmented from ResNet50 and 101 (He et al., 2016), where we add BN-ReLU, global pooling and linear prediction to every two bottleneck residual units for ResNet50, and every three for ResNet101. We create ResANN26 for creating

EANN on ILSVRC, and ResANN26 has four blocks, each of which has two bottleneck residual units. The prediction period is every two units, using the same linear predictors.

DenseANNs on ILSVRC. We augment DenseNet169 (Huang et al., 2017) to create DenseANN 169. DenseNet169 has 82 dense layers, each of which has a 1x1 conv that project concatenation of previous features to $4k$ channels, where k is the growth rate (Huang et al., 2017), followed by a 3x3 conv to generate k channels of features for the dense layer. The two convs are interleaved by BN-ReLU, and a pre-activation BN-ReLU is used for each layer. The 82 layers are organized into four blocks of size 6, 12, 32 and 32. Between each neighboring blocks, a 1x1 conv followed by BN-ReLU-2x2-average-pooling is applied to shrink the existing feature maps by half in the hight, width, and channel dimensions. We add linear anytime predictions every 14 dense layers, starting from layer 12 (1-based indexing). The original DenseNet paper (Huang et al., 2017) mentioned that they use drop-out with keep rate 0.9 after each conv in CIFAR and SVHN, but we found drop-out to be detrimental to performance on ILSVRC.

MSDNet on ILSVRC. MSDNet38 is described in the appendix of (Huang et al., 2018). We set the four blocks to have 10, 9, 10 and 9 layers, and drop the feature maps of the finest resolution after each block as suggest in the original paper. We successfully reproduced the published results to 24.3% error rate on ILSVRC using our Tensorflow implementation. We used the original published results for MSDNet38+CONST in the main text. We use MSDNet33, which has four blocks of 8, 8, 8 and 9 layers, for the small network that uses AdaLoss. We predict using MSDNet33 every seven layers, starting at the fifth layer (1-based indexing).

C Additional Details of AdaLoss

C.1 Weight Regularization

In practice, some expected loss ℓ_i could be much larger than the other losses, so that AdaLoss may assign such ℓ_i too small a weight for it to receive enough optimization to recover. To prevent this, we mix the uniform constant weight with AdaLoss as a form of regularization as follows in Eq. 7. Such mixture prevents the weight of ℓ_i from being too close to zero.

$$\min_{\theta} \sum_{i=1}^{L} \big(\alpha(1-\gamma)\ln \ell_i(\theta) + \gamma \ell_i(\theta)\big), \tag{7}$$

where $\alpha > 0$ and $\gamma > 0$ are hyper parameters. In practice, since DNNs often have elaborate learning rate schedules that assume $B_L = 1$, we choose $\alpha = \min_i \hat{\ell}_i(\theta)$ at each iteration to scale the max weight to 1. We choose $\gamma = 0.05$ from validation sets on CIFAR10 and CIFAR100 from the set $\{0, 0.05, 0.15\}$.

C.2 Ablation Study of AdaLoss parameters on CIFAR

We conduct ablation studies for the parameters of AdaLoss: (1) γ in Eq. 7, which is the mixture weight of the uniform static weighting, (2) the exponential moving average (EMA) momentum, m, for updating the expected loss $\hat{\ell}_i$ at each stochastic gradient descent (SGD) step, and (3) the number of SGD steps e to wait between updating AdaLoss weights B_i using the learned $\hat{\ell}_i$. We choose $\gamma \in \{0, 0.05, 0.15\}$, $m \in \{0.9, 0.99\}$, and $e \in \{1, 100\}$, and evaluate them on CIFAR10 and CIFAR100 ResANNs whose $n \in \{9, 17, 25\}$ and $c = 32$. Over the 72 experiments, we found the effects of m, and e are almost negligible, as they generate $< 0.5\%$ of relative difference in error rates on average, which translates to 0.1% absolute error difference on CIFAR100. These comparisons are in Table 4 and Table 5. In the experiment sections, we choose $m = 0.9$ and $e = 1$.

γ	1/4	1/2	3/4	1	sum
0.0	0.00	0.00	0.00	0.00	0.00
0.05	-20.08	-2.15	2.22	2.43	-17.59
0.15	-23.88	-0.20	5.18	5.17	-13.72

Table 3: Relative percentage increase in error rate by switching from $\gamma = 0$. (lower is better.) A small amount of $\gamma = 0.5$ drastically improves early predictions without increasing late error rate much.

EMA m	1/4	1/2	3/4	1
0.9	0.00	0.00	0.00	0.00
0.99	-0.29	0.25	0.05	0.15

Table 4: Relative percentage increase in error rate by switching from $m = 0.9$. (lower is better.) The two options essentially result in the same error rates.

Update period e	1/4	1/2	3/4	1
1	0.00	0.00	0.00	0.00
100	0.71	0.23	0.24	0.45

Table 5: Relative percentage increase in error rate by switching from $e = 0$. (lower is better.) The options are essentially the same on CIFAR10 and CIFAR100.

However, γ does affect the performance significantly, as show in Table 3. $\gamma = 0$ means pure AdaLoss and $\gamma = 1$ means CONST. We observe that with $\gamma = 0.05$, the small amount of uniform static weight reduces the error rate at 1/4 of the total cost by 20% relatively, but at the cost of minor 2.5% relative increase in late predictions. Increasing γ further to 0.15 has only marginal benefits to early predictions, but has the same negative impact to late accuracy. This suggests that while a small γ helps, we should only use a small amount. Throughout the experiment sections in the main text, we choose $\gamma = 0.05$.