Deciding How to Decide: Dynamic Routing in Artificial Neural Networks

Mason McGill [1] **Pietro Perona** [1]

Abstract

We propose and systematically evaluate three strategies for training dynamically-routed artificial neural networks: graphs of learned transformations through which different input signals may take different paths. Though some approaches have advantages over others, the resulting networks are often qualitatively similar. We find that, in dynamically-routed networks trained to classify images, layers and branches become specialized to process distinct categories of images. Additionally, given a fixed computational budget, dynamically-routed networks tend to perform better than comparable statically-routed networks.

1. Introduction

Some decisions are easier to make than others—for example, large, unoccluded objects are easier to recognize. Additionally, different difficult decisions may require different expertise—an avid birder may know very little about identifying cars. We hypothesize that complex decision-making tasks like visual classification can be meaningfully divided into specialized subtasks, and that a system designed to perform a complex task should first attempt to identify the subtask being presented to it, then use that information to select the most suitable algorithm for its solution.

This approach—dynamically routing signals through an inference system, based on their content—has already been incorporated into machine vision pipelines via methods such as boosting (Viola et al., 2005), coarse-to-fine cascades (Zhou et al., 2013), and random decision forests (Ho, 1995). Dynamic routing is also performed in the primate visual system: spatial information is processed somewhat separately from object identity information (Goodale & Milner, 1992), and faces and other behaviorally-relevant stimuli ellicit responses in anatomically distinct, specialized regions (Moeller et al., 2008; Kornblith et al., 2013). However, state-of-the-art artificial neural networks (ANNs) for visual inference are routed statically (Simonyan & Zisserman, 2014; He et al., 2016; Dosovitskiy et al., 2015; Newell et al., 2016); every input triggers an identical sequence of operations.

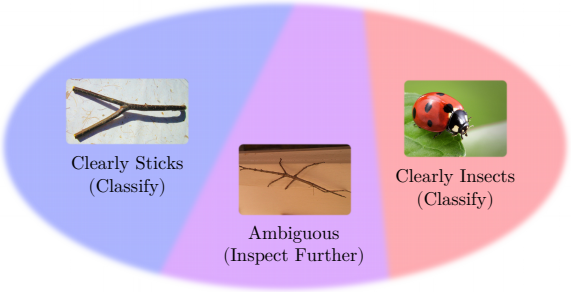

Figure 1. **Motivation for dynamic routing.** For a given data representation, some regions of the input space may be classified confidently, while other regions may be ambiguous.

With this in mind, we propose a mechanism for introducing cascaded evaluation to arbitrary feedforward ANNs, focusing on the task of object recognition as a proof of concept. Instead of classifying images only at the final layer, every layer in the network may attempt to classify images in low-ambiguity regions of its input space, passing ambiguous images forward to subsequent layers for further consideration (see Fig. 1 for an illustration). We propose three approaches to training these networks, test them on small image datasets synthesized from MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky & Hinton, 2009), and quantify the accuracy/efficiency trade-off that occurs when the network parameters are tuned to yield more aggressive early classification policies. Additionally, we propose and evaluate methods for appropriating regularization and optimization techniques developed for statically-routed networks.

2. Related Work

Since the late 1980s, researchers have combined artificial neural networks with decision trees in various

[1]California Institute of Technology, Pasadena, California, USA. Correspondence to: Mason McGill <mmcgill@caltech.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

ways (Utgoff, 1989) (Sirat & Nadal, 1990). More recently, Kontschieder et al. (2015) performed joint optimization of ANN and decision tree parameters, and Bulo & Kontschieder (2014) used randomized multi-layer networks to compute decision tree split functions.

To our knowledge, the family of inference systems we discuss was first described by Denoyer & Gallinari (2014). Additionally, Bengio et al. (2015) explored dynamically skipping layers in neural networks, and Ioannou et al. (2016) explored dynamic routing in networks with equal-length paths. Some recently-developed visual detection systems perform cascaded evaluation of convolutional neural network layers (Li et al., 2015; Cai et al., 2015; Girshick, 2015; Ren et al., 2015); though highly specialized for the task of visual detection, these modifications can radically improve efficiency.

While these approaches lend evidence that dynamic routing can be effective, they either ignore the cost of computation, or do not represent it explicitly, and instead use opaque heuristics to trade accuracy for efficiency. We build on this foundation by deriving training procedures from arbitrary application-provided costs of error and computation, comparing one actor-style and two critic-style strategies, and considering regularization and optimization in the context of dynamically-routed networks.

3. Setup

In a statically-routed, feedforward artificial neural network, every layer transforms a single input feature vector into a single output feature vector. The output feature vector is then used as the input to the following layer (which we'll refer to as the current layer's *sink*), if it exists, or as the ouptut of the network as a whole, if it does not.

We consider networks in which layers may have more than one sink. In such a network, for every n-way junction j a signal reaches, the network must make a decision, $d_j \in \{0..n\}$, such that the signal will propagate through the i^{th} sink if and only if $d_j = i$ (this is illustrated in Fig. 2). We compute d_j as the argmax of the score vector s_j, a learned function of the last feature vector computed before reaching j. We'll refer to this rule for generating d from s as the inference routing policy.

3.1. Multipath Architectures for Convolutional Networks

Convolutional network layers compute collections of *local* descriptions of the input signal. It is unreasonable to expect that this kind of feature vector can explicitly encode the global information relevant to deciding how to route the entire signal (*e.g.*, in the case of object recognition, whether the image was taken indoors, whether the image contains an animal, or the prevalence of occlusion in the scene).

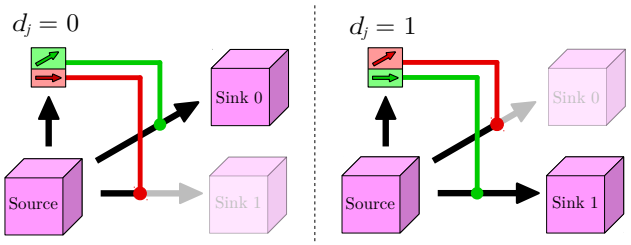

Figure 2. **A 2-way junction**, j. d_j is an integer function of the source features. When $d_j = 0$, the signal is propagated through the top sink, and the bottom sink is inactive. When $d_j = 1$, the signal is propagated through the bottom sink, and the top sink is inactive.

To address this, instead of computing a 2-dimensional array of local features at each layer, we compute a pyramid of features (resembling the pyramids described by Ke et al. (2016)), with local descriptors at the bottom and global descriptors at the top. At every junction j, the score vector s_j is computed by a small routing network operating on the last-computed global descriptor. Our multipath architecture is illustrated in Fig. 3.

3.2. Balancing Accuracy and Efficiency

For a given input, network ν, and set of routing decisions d, we define the cost of performing inference:

$$c_{\text{inf}}(\nu, d) = c_{\text{err}}(\nu, d) + c_{\text{cpt}}(\nu, d), \tag{1}$$

where $c_{\text{err}}(\nu, d)$ is the cost of the inference errors made by the network, and $c_{\text{cpt}}(\nu, d)$ is the cost of computation. In our experiments, unless stated otherwise, c_{err} is the cross-entropy loss and

$$c_{\text{cpt}}(\nu, d) = k_{\text{cpt}} n_{\text{ops}}(\nu, d), \tag{2}$$

where $n_{\text{ops}}(\nu, d)$ is the number of multiply-accumulate operations performed and k_{cpt} is a scalar hyperparameter. This definition assumes a time- or energy-constrained system—every operation consumes roughly the same amount of time and energy, so every operation is equally expensive. c_{cpt} may be defined differently under other constraints (*e.g.* memory bandwidth).

4. Training

We propose three approaches to training dynamically-routed networks, along with complementary approaches to regularization and optimization, and a method for adapting to changes in the cost of computation.

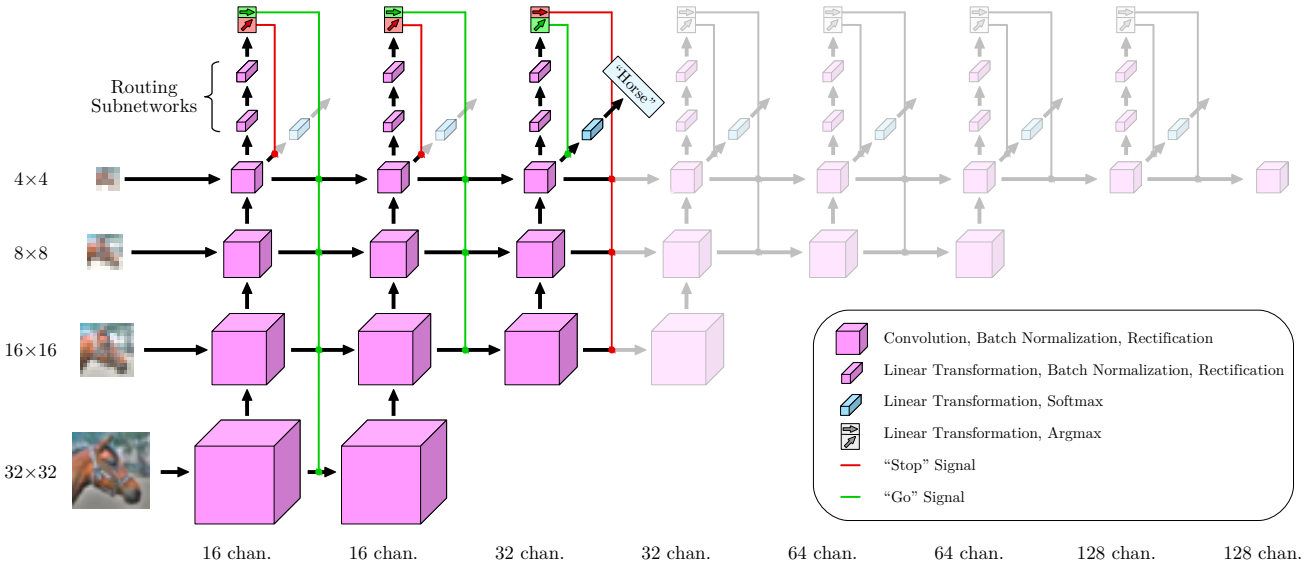

Figure 3. **Our multiscale convolutional architecture.** Once a column is evaluated, the network decides whether to classify the image or evaluate subsequent columns. Deeper columns operate at coarser scales, but compute higher-dimensional representations at each location. All convolutions use 3×3 kernels, downsampling is achieved via 2×2 max pooling, and all routing layers have 16 channels.

4.1. Training Strategy I: Actor Learning

Since d is discrete, $c_{\text{inf}}(\nu, d)$ cannot be minimized via gradient-based methods. However, if d is replaced by a stochastic approximation, $\hat{d}$, during training, we can engineer the gradient of $\mathrm{E}[c_{\text{inf}}(\nu, \hat{d})]$ to be nonzero. We can then learn the routing parameters and classification parameters simultaneously by minimizing the loss

$$L_{\text{ac}} = \mathrm{E}[c_{\text{inf}}(\nu, \hat{d})]. \tag{3}$$

In our experiments, the training routing policy samples $\hat{d}$ such that

$$\Pr(\hat{d}_j = i) = \text{softmax}(s_j/\tau)_i, \tag{4}$$

where τ is the network "temperature": a scalar hyperparameter that decays over the course of training, converging the training routing policy towards the inference routing policy.

4.2. Training Strategy II: Pragmatic Critic Learning

Alternatively, we can attempt to learn to predict the expected utility of making every routing decision. In this case, we minimize the loss

$$L_{\text{cr}} = \mathrm{E}\left[c_{\text{inf}}(\nu, \hat{d}) + \sum_{j \in \mathcal{J}} c^j_{\text{ure}}\right], \tag{5}$$

where $\mathcal{J}$ is the set of junctions encountered when making the routing decisions $\hat{d}$, and c_{ure} is the utility regression error cost, defined:

$$c^j_{\text{ure}} = k_{\text{ure}} \|s_j - u_j\|^2, \tag{6}$$

where

$$u^i_j = -c_{\text{inf}}(\nu^i_j, d), \tag{7}$$

k_{ure} is a scalar hyperparameter, and ν^i_j is the subnetwork consisting of the i^{th} child of ν_j, and all of its descendants. Since we want to learn the policy indirectly (via cost prediction), $\hat{d}$ is treated as constant with respect to optimization.

4.3. Training Strategy III: Optimistic Critic Learning

To improve the stability of the loss and potentially accelerate training, we can adjust the routing utility function u such that, for every junction j, u_j is independent of the routing parameters downstream of j. Instead of predicting the cost of making routing decisions given the *current* downstream routing policy, we can predict the cost of making routing decisions given the *optimal* downstream routing policy. In this optimistic variant of the critic method,

$$u^i_j = -\min_{d'}(c_{\text{inf}}(\nu^i_j, d')). \tag{8}$$

4.4. Regularization

Many regularization techniques involve adding a model-complexity term, c_{mod}, to the loss function to influence learning, effectively imposing soft constraints upon the network parameters (Hoerl & Kennard, 1970; Rudin et al., 1992; Tibshirani, 1996). However, if such a term affects layers in a way that is independent of the amount of signal routed through them, it will either underconstrain frequently-used layers or overconstrain infrequently-used layers. To support both frequently- and infrequently-used layers, we regularize subnetworks as they are activated by $\hat{d}$, instead of regularizing the entire network directly.

For example, to apply L2 regularization to critic networks, we define c_{mod}:

$$c_{\text{mod}} = E\left[k_{\text{L2}} \sum_{w \in \mathcal{W}} w^2\right], \tag{9}$$

where $\mathcal{W}$ is the set of weights associated with the layers activated by $\hat{d}$, and k_{L2} is a scalar hyperparameter.

For actor networks, we apply an extra term to control the magnitude of s, and therefore the extent to which the net explores subpotimal paths:

$$c_{\text{mod}} = E\left[k_{\text{L2}} \sum_{w \in \mathcal{W}} w^2 + k_{\text{dec}} \sum_{j \in \mathcal{J}} \|s_j\|^2\right], \tag{10}$$

where k_{dec} is a scalar hyperparameter indicating the relative cost of decisiveness.

c_{mod} is added to the loss function in all of our experiments. Within c_{mod}, unless stated otherwise, $\hat{d}$ is treated as constant with respect to optimization.

4.5. Adjusting Learning Rates to Compensate for Throughput Variations

Both training techniques attempt to minimize the expected cost of performing inference with the network, over the training routing policy. With this setup, if we use a constant learning rate for every layer in the network, then layers through which the policy routes examples more frequently will receive larger parameter updates, since they contribute more to the expected cost.

To allow every layer to learn as quickly as possible, we scale the learning rate of each layer ℓ dynamically, by a factor α_ℓ, such that the elementwise variance of the loss gradient with respect to ℓ's parameters is independent of the amount of probability density routed through it.

To derive α_ℓ, we consider an alternative routing policy, d^*_ℓ, that routes all signals though ℓ, then routes through subsequent layers based on $\hat{d}$. With this policy, at every training interation, mini-batch stochastic gradient descent shifts the parameters associated with layer ℓ by a vector δ^*_ℓ, defined:

$$\delta^*_\ell = -\lambda \sum_i g^i_\ell, \tag{11}$$

where λ is the global learning rate and g^i_ℓ is the gradient of the loss with respect to the parameters in ℓ, for training example i, under d^*_ℓ. Analogously, the scaled parameter adjustment under $\hat{d}$ can be written

$$\delta_\ell = -\alpha_\ell \lambda \sum_i p^i_\ell g^i_\ell, \tag{12}$$

where p^i_ℓ is the probability with which $\hat{d}$ routes example i through ℓ.

We want to select α_ℓ such that

$$\text{Var}(\delta_\ell) = \text{Var}(\delta^*_\ell). \tag{13}$$

Substituting the definitions of δ_ℓ and δ^*_ℓ,

$$\text{Var}\left(\alpha_\ell \sum_i p^i_\ell g^i_\ell\right) = \text{Var}\left(\sum_i g^i_\ell\right). \tag{14}$$

Since every g^i_ℓ is sampled independently, we can rewrite this equation:

$$n_{\text{ex}} v_\ell \alpha^2_\ell \|p_\ell\|^2 = n_{\text{ex}} v_\ell, \tag{15}$$

where n_{ex} is the number of training examples in the mini-batch and v_ℓ is the elementwise variance of g^i_ℓ, for any i (since every example is sampled via the same mechanism). We can now show that

$$\alpha_\ell = \|p_\ell\|^{-1}. \tag{16}$$

So, for every layer ℓ, we can scale the learning rate by $\|p_\ell\|^{-1}$, and the variance of the weight updates will be similar thoughout the network. We use this technique, unless otherwise specified, in all of our experiments.

4.6. Responding to Changes in the Cost of Computation

We may want a single network to perform well in situations with various degrees of computational resource scarcity (*e.g.* computation may be more expensive when a device battery is low). To make the network's routing behavior responsive to a dynamic c_{cpt}, we can concatenate c_{cpt}'s known

parameters—in our case, $\{k_{\text{cpt}}\}$—to the input of every routing subnetwork, to allow them to modulate the routing policy. To match the scale of the image features and facilitate optimization, we express k_{cpt} in units of cost per ten-million operations.

4.7. Hyperparameters

In all of our experiments, we use a mini-batch size, n_{ex}, of 128, and run 80,000 training iterations. We perform stochastic gradient descent with initial learning rate $0.1/n_{\text{ex}}$ and momentum 0.9. The learning rate decays continuously with a half-life of 10,000 iterations.

The weights of the final layers of routing networks are zero-initialized, and we initialize all other weights using the Xavier initialization method (Glorot & Bengio, 2010). All biases are zero-initialized. We perform batch normalization (Ioffe & Szegedy, 2015) before every rectification operation, with an ϵ of 1×10^{-6}, and an exponential moving average decay constant of 0.9.

τ is initialized to 1.0 for actor networks and 0.1 for critic networks, and decays with a half-life of 10,000 iterations. $k_{\text{dec}} = 0.01$, $k_{\text{ure}} = 0.001$, and $k_{\text{L2}} = 1 \times 10^{-4}$. We selected these values (for τ, k_{dec}, k_{ure}, and k_{L2}) by exploring the hyperparameter space logarithmically, by powers of 10, training and evaluating on the hybrid MNIST/CIFAR-10 dataset (described in section 5.1). At a coarse level, these values are locally optimal—multiplying or dividing any of them by 10 will not improve performance.

4.8. Data Augmentation

We augment our data using an approach that is popular for use with CIFAR-10 (Lin et al., 2013) (Srivastava et al., 2015) (Clevert et al., 2015). We augment each image by applying vertical and horizontal shifts sampled uniformly from the range [-4px,4px], and, if the image is from CIFAR-10, flipping it horizontally with probability 0.5. We fill blank pixels introduced by shifts with the mean color of the image (after gamma-decoding).

5. Experiments

We compare approaches to dynamic routing by training 153 networks to classify small images, varying the policy-learning strategy, regularization strategy, optimization strategy, architecture, cost of computation, and details of the task. The results of these experiments are reported in Fig. 5–10. Our code is available via GitLab.

5.1. Comparing Policy-Learning Strategies

To compare routing strategies in the context of a simple dataset with a high degree of difficulty variation, we train networks to classify images from a small-image dataset synthesized from MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky & Hinton, 2009) (see Fig. 4). Our dataset includes the classes "0", "1", "2", "3", and "4" from MNIST and "airplane", "automobile", "deer", "horse", and "frog" from CIFAR-10 (see Fig. 4). The images from MNIST are resized to match the scale of images from CIFAR-10 (32×32), via linear interpolation, and are color-modulated to make them more difficult to trivially distinguish from CIFAR-10 images (MNIST is a grayscale dataset).

Figure 4. **Sample images from the hybrid MNIST/CIFAR-10 dataset.** We recolor images from MNIST via the following procedure: we select two random colors at least 0.3 units away from each other in RGB space; we then map black pixels to the first color, map white pixels to the second color, and linearly interpolate in between.

For a given computational budget, dynamically-routed networks achieve higher accuracy rates than architecture-matched statically-routed baselines (networks composed of the first n columns of the architecture illustrated in Fig. 3, for $n \in \{1..8\}$). Additionally, dynamically-routed networks tend to avoid routing data along deep paths at the beginning of training (see Fig. 8). This is possibly because the error surfaces of deeper networks are more complicated, or because deeper paths are less stable—changing the parameters in any component layer to better classify images routed along other, overlapping paths may decrease performance. Whatever the mechanism, this tendency to initially find simpler solutions seems to prevent some of the overfitting that occurs with 7- and 8-layer statically-routed networks.

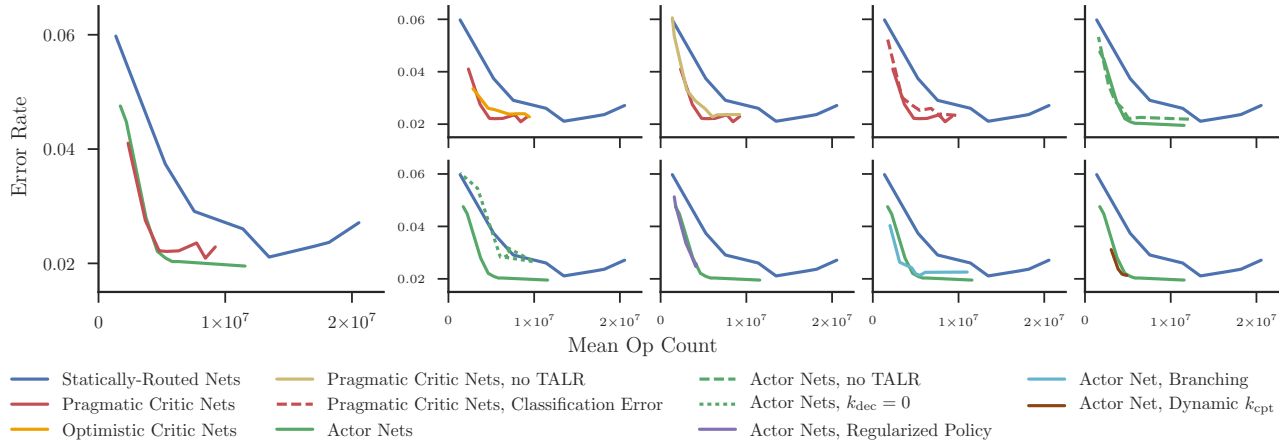

Figure 5. **Hybrid dataset performance.** Every point along the "statically-routed nets" curve corresponds to a network composed of the first n columns of the architecture illustrated in Fig. 3, for $1 \leq n \leq 8$. The points along the "actor net, dynamic k_{cpt}" curve correspond to a single network evaluated with various values of k_{cpt}, as described in section 4.6. The points along all other curves correspond to distinct networks, trained with different values of k_{cpt}. $k_{\text{cpt}} \in \{0, 1 \times 10^{-9}, 2 \times 10^{-9}, 4 \times 10^{-9}, \ldots 6.4 \times 10^{-8}\}$.

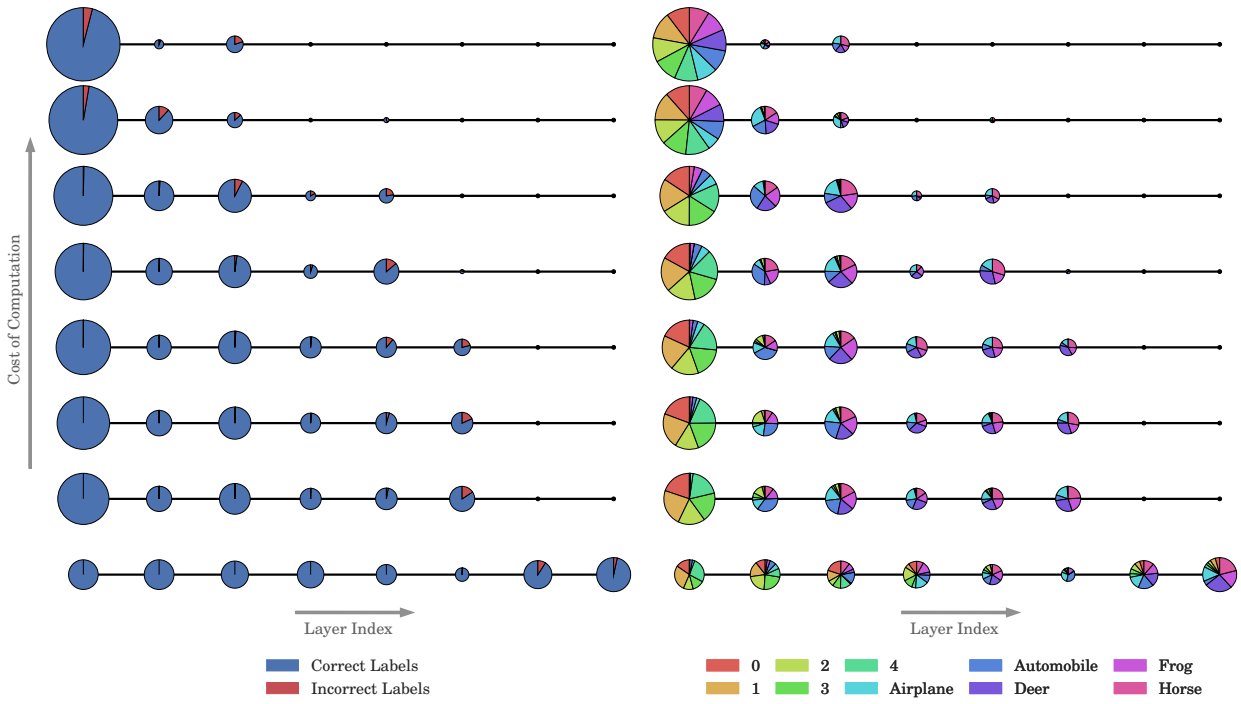

Figure 6. **Dataflow through actor networks** trained to classify images from the hybrid MNIST/CIFAR-10 dataset. Every row is a node-link diagram corresponding to a network, trained with a different k_{cpt}. Each circle indicates, by area, the fraction of examples that are classified at the corresponding layer. The circles are colored to indicate the accuracy of each layer (left) and the kinds of images classified at each layer (right).

Compared to other dynamically-routed networks, optimistic critic networks perform poorly, possibly because optimal routers are a poor approximation for our small, low-capacity router networks. Actor networks perform better than critic networks, possibly because critic networks are forced to learn a potentially-intractable auxilliary task (i.e. it's easier to decide who to call to fix your printer than it is to predict exactly how quickly and effectively everyone you know would fix it). Actor networks also consistently achieve higher peak accuracy rates than comparable statically-routed networks, across experiments.

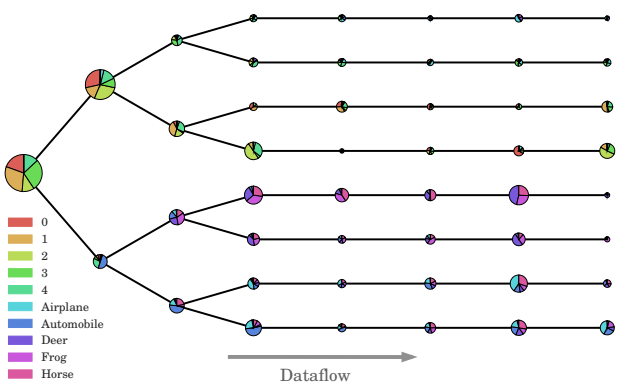

Figure 7. **Dataflow through a branching actor network** trained to classify images in the hybrid dataset, illustrated as in Fig. 6.

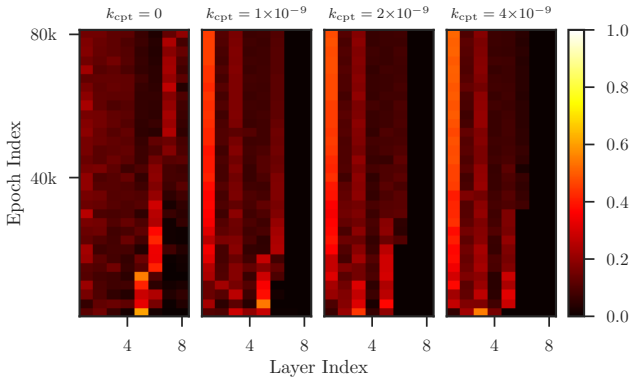

Figure 8. **Dataflow over the course of training.** The heatmaps illustrate the fraction of validation images classified at every terminal node in the bottom four networks in Fig. 6, over the course of training.

Although actor networks may be more performant, critic networks are more flexible. Since critic networks don't require $\mathrm{E}[c_{\text{inf}}(\nu, \hat{d})]$ to be a differentiable function of $\hat{d}$, they can be trained by sampling $\hat{d}$, saving memory, and they support a wider selection of training routing policies (e.g. ϵ-greedy) and c_{inf} definitions. In addition to training the standard critic networks, we train networks using a variant of the pragmatic critic training policy, in which we replace the cross-entropy error in the c_{ure} term with the classification error. Although these networks do not perform as well as the original pragmatic critic networks, they still outperform comparable statically-routed networks.

5.2. Comparing Regularization Strategies

Based on our experiments with the hybrid dataset, regularizing $\hat{d}$, as described in section 4.4, discourages networks from routing data along deep paths, reducing peak accuracy. Additionally, some mechanism for encouraging exploration (in our case, a nonzero k_{dec}) appears to be necessary to train effective actor networks.

5.3. Comparing Optimization Strategies

Throughput-adjusting the learning rates (TALR), as described in section 4.5, improves the hybrid dataset performance of both actor and critic networks in computational-resource-abundant, high-accuracy contexts.

5.4. Comparing Architectures

For a given computational budget, architectures with both 2- and 3-way junctions have a higher capacity than subtrees with only 2-way junctions. On the hybrid dataset, under tight computational constraints, we find that trees with higher degrees of branching achieve higher accuracy rates. Unconstrained, however, they are prone to overfitting.

In dynamically-routed networks, early classification layers tend to have high accuracy rates, pushing difficult decisions downstream. Even without energy contraints, terminal layers specialize in detecting instances of certain classes of images. These classes are usually related (they either all come from MNIST or all come from CIFAR-10.) In networks with both 2- and 3-way junctions, branches specialize to an even greater extent. (See Fig. 6 and 7.)

5.5. Comparing Specialized and Adaptive Networks

We train a single actor network to classify images from the hybrid datset under various levels of computational constraints, using the approach described in section 4.6, sampling k_{cpt} randomly from the set mentioned in Fig. 5 for each training example. This network performs comparably to a collection of 8 actor nets trained with various static values of k_{cpt}, over a significant, central region of the accuracy/efficiency curve, with an 8-fold reduction in memory consumption and training time.

5.6. Exploring the Effects of the Decision Difficulty Distribution

To probe the effect of the inference task's difficulty distribution on the performance of dynamically-routed net-

works, we train networks to classify images from CIFAR-10, adjusting the classification task to vary the frequency of difficult decisions (see Fig. 9). We call these variants CIFAR-2—labelling images as "horse" or "other"—and CIFAR-5—labelling images as "cat", "dog", "deer", "horse", or "other". In this experiment, we compare actor networks (the best-performing networks from the first set of experiments) to architecture-matched statically-routed networks.

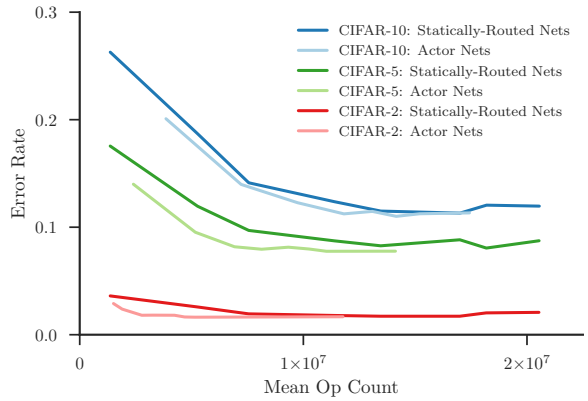

Figure 9. **Performance effects of the task difficulty distribution**, as described in section 5.6. The "statically-routed nets" and "actor nets" curves are drawn analogously to their counterparts in Fig. 5.

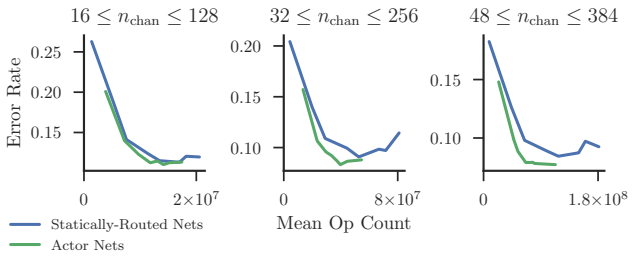

Figure 10. **Performance effects of model capacity**, training and testing on CIFAR-10. (Left) Networks with (subsets of) the architecture illustrated in Fig. 3. (Center) Networks otherwise identical to those presented in the left panel, with the number of output channels of every convolutional layer multiplied by 2, and k_{cpt} divided by 4. (Right) Networks otherwise identical to those presented in the left panel, with the number of output channels of every convolutional layer multiplied by 3, and k_{cpt} divided by 9.

We find that dynamic routing is more beneficial when the task involves many low-difficulty decisions, allowing the network to route more data along shorter paths. While dynamic routing offers only a slight advantage on CIFAR-10, dynamically-routed networks achieve a higher peak accuracy rate on CIFAR-2 than statically-routed networks, at a third of the computational cost.

5.7. Exploring the Effects of Model Capacity

To test whether dynamic routing is advantageous in higher-capacity settings, we train actor networks and architecture-matched statically-routed networks to classify images from CIFAR-10, varying the width of the networks (see Fig. 10). Increasing the model capacity either increases or does not affect the relative advantage of dynamically-routed networks, suggesting that our approach is applicable to more complicated tasks.

6. Discussion

Our experiments suggest that dynamically-routed networks trained under mild computational constraints can operate 2–3 times more efficiently than comparable statically-routed networks, without sacrificing performance. Additionally, despite their higher capacity, dynamically-routed networks may be less prone to overfitting.

When designing a multipath architecture, we suggest supporting early decision-making wherever possible, since cheap, simple routing networks seem to work well. In convolutional architectures, pyramidal layers appear to be reasonable sites for branching.

The actor strategy described in section 4.1 is generally an effective way to learn a routing policy. However, the pragmatic critic strategy described in section 4.2 may be better suited for very large networks (trained via decision sampling to conserve memory) or networks designed for applications with nonsmooth cost-of-inference functions—*e.g.* one in which k_{cpt} has units *errors*/*operation*. Adjusting learning rates to compensate for throughput variations, as described in section 4.5, may improve the performance of deep networks. If the cost of computation is dynamic, a single network, trained with the procedure described in section 5.5, may still be sufficient.

While we test our approach on tasks with some degree of difficulty variation, it is possible that dynamic routing is even more advantageous when performing more complex tasks. For example, video annotation may require specialized modules to recognize locations, objects, faces, human actions, and other scene components or attributes, but having every module constantly operating may be extremely inefficient. A dynamic routing policy could fuse these modules, allowing them to share common components, and activate specialized components as necessary.

Another interesting topic for future research is growing and shrinking dynamically-routed networks during training. With such a network, it is not necessary to specify an architecture. The network will instead take shape over the course of training, as computational contraints, memory contraints, and the data dictate.

Acknowledgements

This work was funded by a generous grant from Google Inc. We would also like to thank Krzysztof Chalupka, Cristina Segalin, and Oisin Mac Aodha for their thoughtful comments.

References

Bengio, Emmanuel, Bacon, Pierre-Luc, Pineau, Joelle, and Precup, Doina. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Bulo, Samuel and Kontschieder, Peter. Neural decision forests for semantic image labelling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 81–88, 2014.

Cai, Zhaowei, Saberian, Mohammad, and Vasconcelos, Nuno. Learning complexity-aware cascades for deep pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3361–3369, 2015.

Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Denoyer, Ludovic and Gallinari, Patrick. Deep sequential neural network. *arXiv preprint arXiv:1410.0510*, 2014.

Dosovitskiy, Alexey, Fischer, Philipp, Ilg, Eddy, Hausser, Philip, Hazirbas, Caner, Golkov, Vladimir, van der Smagt, Patrick, Cremers, Daniel, and Brox, Thomas. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2758–2766, 2015.

Girshick, Ross. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1440–1448, 2015.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pp. 249–256, 2010.

Goodale, Melvyn A and Milner, A David. Separate visual pathways for perception and action. *Trends in neurosciences*, 15(1):20–25, 1992.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Ho, Tin Kam. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pp. 278–282. IEEE, 1995.

Hoerl, Arthur E and Kennard, Robert W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Ioannou, Yani, Robertson, Duncan, Zikic, Darko, Kontschieder, Peter, Shotton, Jamie, Brown, Matthew, and Criminisi, Antonio. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Ke, Tsung-Wei, Maire, Michael, and Yu, Stella X. Neural multigrid. *arXiv preprint arXiv:1611.07661*, 2016.

Kontschieder, Peter, Fiterau, Madalina, Criminisi, Antonio, and Rota Bulo, Samuel. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1467–1475, 2015.

Kornblith, Simon, Cheng, Xueqi, Ohayon, Shay, and Tsao, Doris Y. A network for scene processing in the macaque temporal lobe. *Neuron*, 79(4):766–781, 2013.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009.

LeCun, Yann, Cortes, Corinna, and Burges, Christopher JC. The mnist database of handwritten digits, 1998.

Li, Haoxiang, Lin, Zhe, Shen, Xiaohui, Brandt, Jonathan, and Hua, Gang. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5325–5334, 2015.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Moeller, Sebastian, Freiwald, Winrich A, and Tsao, Doris Y. Patches with links: a unified system for processing faces in the macaque temporal lobe. *Science*, 320(5881):1355–1359, 2008.

Newell, Alejandro, Yang, Kaiyu, and Deng, Jia. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pp. 483–499. Springer, 2016.

Ren, Shaoqing, He, Kaiming, Girshick, Ross, and Sun, Jian. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pp. 91–99, 2015.

Rudin, Leonid I, Osher, Stanley, and Fatemi, Emad. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sirat, JA and Nadal, JP. Neural trees: a new tool for classification. *Network: Computation in Neural Systems*, 1 (4):423–438, 1990.

Srivastava, Rupesh K, Greff, Klaus, and Schmidhuber, Jürgen. Training very deep networks. In *Advances in neural information processing systems*, pp. 2377–2385, 2015.

Tibshirani, Robert. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

Utgoff, Paul E. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1(4):377–391, 1989.

Viola, Paul, Jones, Michael J, and Snow, Daniel. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.

Zhou, Erjin, Fan, Haoqiang, Cao, Zhimin, Jiang, Yuning, and Yin, Qi. Extensive facial landmark localization with coarse-to-fine convolutional network cascade. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 386–391, 2013.