An Algorithm for Routing Capsules in All Domains with Sample Applications in Vision and Language

Franz A. Heinsen
franz@glassroom.com

Abstract

Building on recent work on capsule networks, we propose a new form of "routing by agreement" that activates output capsules in a layer as a function of their net benefit to use and net cost to ignore input capsules from earlier layers. As sample applications, we present two capsule networks that use our algorithm *without change* in different domains: vision and language.[1] The first network achieves new state-of-the-art accuracy of **99.1%** on the smallNORB visual recognition task with fewer parameters and an order of magnitude less training than previous capsule models, and we find evidence that it learns to perform a form of "reverse graphics." The second network achieves new state-of-the-art performance on the root sentences of the Stanford Sentiment Treebank: **58.5%** accuracy on finegrained and **95.6%** on binary labels with a single-task model that routes frozen embeddings from a pretrained transformer as capsules. Both networks are trained with the same regime.

1 Introduction

Capsule networks with routing by agreement can be more effective than convolutional neural networks for segmenting highly overlapping images (Sabour et al., 2017) and for generalizing to different poses of objects embedded in images and resisting white-box adversarial image attacks (Hinton et al., 2018), typically requiring fewer parameters but more training and computation.

A capsule is a group of neurons whose outputs represent different properties of the same entity in different contexts. Routing by agreement is an iterative form of clustering in which a capsule detects an entity by looking for agreement among votes from input capsules that have already detected parts of the entity in a previous layer.

1.1 Our Routing Algorithm

Here, we propose a new variant of "EM routing" (Hinton et al., 2018), a form of routing by agreement that uses the expectation-maximization (EM) algorithm to cluster similar votes from input capsules to output capsules. Each output capsule iteratively maximizes the probability of input votes assigned to it, given its probabilistic model.

Our EM routing algorithm has multiple differences compared to previous ones. The most significant difference is that we compute each output capsule's activation by applying a logistic function to the difference between a *net benefit to use* and a *net cost to ignore* (*i.e.*, not use) each input capsule. We compute the share of each input capsule used or ignored by each output capsule in a new procedure we call the *D-Step*, executed between the E-Step and M-Step of each EM iteration. We are motivated by the intuitive notion that

> *"output capsules should benefit from the input data they use, and lose benefits from any input data they ignore,"*

as they maximize the probability of votes from those input capsules they use in each EM iteration.

We simultaneously (a) optimize the entire layer for a training objective, (b) iteratively maximize the probability of input capsule votes each output capsule uses, and (c) find mixtures of net input capsule benefits and costs in service of (a) and (b). We like to think of this mechanism as finding "the combination of net benefits and costs that produces greater profit," or, more colorfully, maximizing *"bang per bit."*

[1] In both domains, we use the same routing code, available at https://github.com/glassroom/heinsen_routing along with pretrained models and replication instructions.

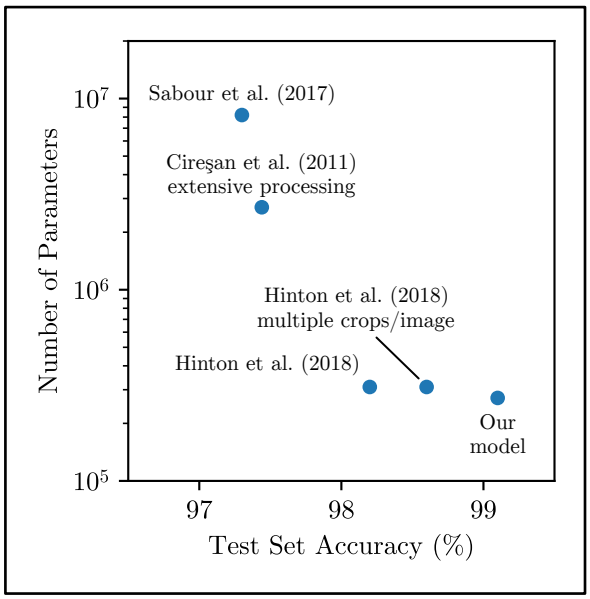

Figure 1: Test set accuracy and number of parameters of models that have achieved state-of-the-art results on smallNORB visual recognition.

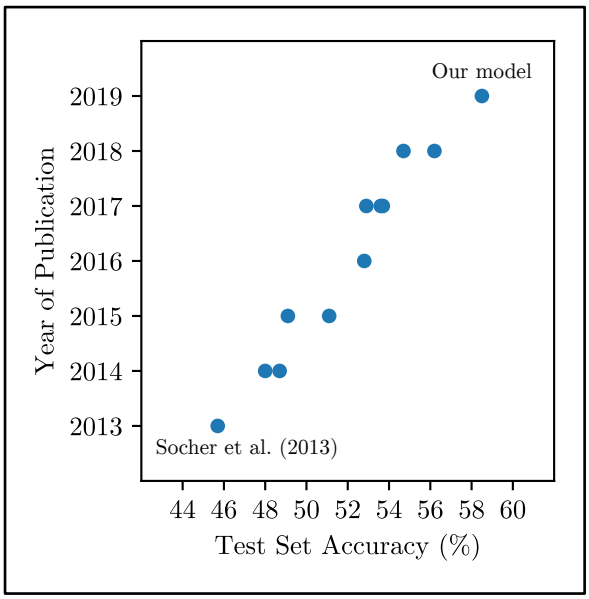

Figure 2: Test set accuracy and publication year of models that have achieved state-of-the-art results on SST-5/R root sentence classification.

Another significant difference of our routing algorithm, compared to previous ones, is that it accepts variable-size inputs, such as sequences of contextualized token embeddings in natural language applications. A contextualized token embedding is a special case of a capsule, one whose neuron outputs represent different properties of the same token id in different contexts.

1.2 Current Challenges to Scaling

At present, scaling EM routing algorithms such as ours to large datasets or tasks with large output spaces is impractical, due mainly to limitations of current software (*e.g.*, PyTorch, TensorFlow) and hardware (*e.g.*, GPUs, TPUs) systems, which are highly optimized for a small set of computational "kernels" and rely on low-level code that is tightly coupled with memory hardware, leading to poor memory reuse and poor performance on non-standard workloads, including basic operations on capsules (Barham and Isard, 2019). We expect these challenges to scaling will gradually be overcome. Until then, we can evaluate our routing algorithm on smaller datasets and output spaces.

1.3 Sample Applications in Two Domains

For comparison with prior work on capsule networks, we evaluate our routing algorithm on the smallNORB visual recognition task (LeCun et al., 2004), in which objects must be recognized from stereo images with varying azimuths, elevations, and lighting conditions. We construct a capsule network that routes pixel convolutions as capsules to achieve new state-of-the-art accuracy of 99.1% on this task. Compared to the previous state of the art (Hinton et al., 2018), our smallNORB model has approximately 273,000 instead of 310,000 parameters, trains in 50 instead of 300 epochs, and accepts as input non-downsampled pairs of 96×96 images, instead of 32×32 downsampled crops that are nine times smaller. Also, we do not average over multiple crops per image to compute test accuracy. Fig. 1 shows how our model compares to prior state-of-the-art models on two criteria.

We find evidence that our smallNORB model learns to encode pose solely from pixel data and classification labels, *i.e.*, it learns its own form of "reverse graphics" without us explicitly having to optimize for it. See the visualization in Fig. 4 (p. 9) and the 24 plots and corresponding captions in Supp. Fig. 6 (p. 14) and Supp. Fig. 7 (p. 15).

We also evaluate our routing algorithm on a natural language task: classifying the root sentences of the Stanford Sentiment Treebank (Socher et al., 2013) into fine-grained (SST-5/R) and binary (SST-2/R) labels. (We add the "/R" designation to distinguish these root-sentence tasks from classification of phrases in the parse trees.)

For evaluation of our algorithm on SST-5/R

and SST-2/R, we construct a capsule network that routes frozen token embeddings from a transformer (Vaswani et al., 2017) as capsules. The network has between 141,000 and 143,000 parameters, depending on the final number of output capsules, one per label. In our implementation, we use a GPT-2_{large} (Radford et al., 2018) as the pretrained transformer, the largest such model publicly available at the time of writing. Our SST model achieves new state-of-the-art test set accuracies of 58.5% on SST-5/R and 95.6% on SST-2/Rfor single-task models. Fig. 2 shows how our SST model compares to prior state-of-the-art models on SST-5/R.

1.4 Our Motivation

In applying our routing algorithm in two domains, vision and natural language, we are motivated by the goal of developing universal, composable learning algorithms that can adapt to any domain. Our work is but a small step in this direction.

2 Our Routing Algorithm

For brevity and clarity, we assume familiarity with both capsule networks and the matrix EM routing algorithm proposed by Hinton et al. (2018), so we focus our discussion here mainly on those aspects of our work that are new. Also, while our algorithm generalizes to any probabilistic model that can be used in an expectation-maximization loop, we restrict our discussion here only to one case: a multidimensional Gaussian model.

As shown in Alg. 1, for a given sample, our algorithm dynamically routes $n^{(\text{inp})}$ input capsules to $n^{(\text{out})}$ output capsules, where $n^{(\text{inp})}$ is specified in advance if samples are of fixed size or left unspecified if samples are of variable size. Input capsules are tensors of size $d^{(\text{cov})} \times d^{(\text{inp})}$, and output capsules are tensors of size $d^{(\text{cov})} \times d^{(\text{out})}$.

Per sample, we accept as input two tensors: *scores* and *capsules* ($a_i^{(\text{inp})}$, $\mu_{icd}^{(\text{inp})}$). We return as output three tensors: *scores*, *capsules*, and *variances* ($a_j^{(\text{out})}$, $\mu_{jch}^{(\text{out})}$, $(\sigma_{jch}^{(\text{out})})^2$). The indices are

$$\begin{aligned} i &= (1, 2, \ldots, n^{(\text{inp})}), \\ j &= (1, 2, \ldots, n^{(\text{out})}), \\ c &= (1, 2, \ldots, d^{(\text{cov})}), \\ d &= (1, 2, \ldots, d^{(\text{inp})}), \\ h &= (1, 2, \ldots, d^{(\text{out})}). \end{aligned}$$

Intuitively, we can think of $n^{(\text{inp})}$ as the number of detectable parts, $n^{(\text{out})}$ as the number of detectable entities, each consisting of or associated with one or more parts, $d^{(\text{cov})}$ as the dimension of the covector space, or dual, of the spaces in which parts and entities have properties, and $d^{(\text{inp})}$ and $d^{(\text{out})}$ as the dimensions of part and entity properties, respectively, in those spaces.

For example, if we wanted to detect, say, dogs and cats embedded in images, from 64 detectable animal parts, the values of $n^{(\text{inp})}$, $n^{(\text{out})}$, $d^{(\text{cov})}$, $d^{(\text{inp})}$, and $d^{(\text{out})}$ would be 64, 2, 4, 4, and 4, respectively. Each of the 64 input capsules and 2 output capsules would be a 4×4 tensor capable of representing the spatial relationship between the viewer of an image and objects embedded in the image.

In other domains, the dimensions $d^{(\text{cov})}$ of the dual space, $d^{(\text{inp})}$ of part properties, and $d^{(\text{out})}$ of entity properties may be very different.

When $n^{(\text{inp})}$ is unspecified, the number of detectable parts is variable. In that case, it might be desirable for input capsules themselves to have properties that represent their type and/or position. This is commonly done, for example, in language models, which add relative or absolute position information to token embeddings.

2.1 Votes

Before starting the routing loop, we compute votes from each input to each output capsule,

$$V_{ijch} = \begin{cases} \sum_d W_{ijdh}\, \mu_{icd}^{(\text{inp})} + B_{ijch}, & \text{if } n^{(\text{inp})} \text{ fixed} \\ \sum_d W_{djh}\, \mu_{icd}^{(\text{inp})} + B_{jch}, & \text{otherwise} \end{cases} \tag{1}$$

where V_{ijch} is a tensor of votes computed from the ith input capsule to the jth output capsule for each component ch of the output capsule, and W_{ijh}^d and B_{ijch} (or W_{jh}^d and B_{jch}, if $n^{(\text{inp})}$ is unspecified) are parameters. We perform tensor contraction on index d and compute elementwise (Hadamard) operations, as indicated, along indexes i, j, c, and h, with conventional broadcasting implicitly assumed for any missing dimensions. For each output capsule j, we obtain a different ich slice of input capsule votes.

Intuitively, the computation of V_{ijch} in (1) can be understood as $n^{(\text{inp})} \times n^{(\text{out})}$ simultaneous matrix-matrix multiplications, each applying a $d^{(\text{out})} \times d^{(\text{inp})}$ linear transformation to a $d^{(\text{inp})} \times d^{(\text{cov})}$ transposed input capsule, followed by addition of biases to each transformed capsule and then another transposition, to obtain $n^{(\text{inp})} \times n^{(\text{out})}$ votes

Algorithm 1: Our Routing Algorithm. Per sample, we route $n^{(\text{inp})}$ input capsules of shape $d^{(\text{cov})} \times d^{(\text{inp})}$ to $n^{(\text{out})}$ output capsules of shape $d^{(\text{cov})} \times d^{(\text{out})}$, where $n^{(\text{inp})}$ is the number of detectable entity parts and $n^{(\text{out})}$ is the number of detectable entities. For clarity, all tensor operations shown are element-wise (*i.e.*, there are no Einstein summations) and implicitly assume conventional broadcasting for missing dimensions. Tensor indexes are $i = (1, 2, \ldots, n^{(\text{inp})})$, $j = (1, 2, \ldots, n^{(\text{out})})$, $c = (1, 2, \ldots, d^{(\text{cov})})$, $d = (1, 2, \ldots, d^{(\text{inp})})$, $h = (1, 2, \ldots, d^{(\text{out})})$.

Input: $(a_i^{(\text{inp})},\ \mu_{icd}^{(\text{inp})})$.
Output: $(a_j^{(\text{out})},\ \mu_{jch}^{(\text{out})},\ (\sigma_{jch}^{(\text{out})})^2)$.

$V_{ijch} \longleftarrow \begin{cases} \sum_d W_{ijdh}\, \mu_{icd}^{(\text{inp})} + B_{ijch}, & \text{if } n^{(\text{inp})} \text{ fixed} \\ \sum_d W_{djh}\, \mu_{icd}^{(\text{inp})} + B_{jch}, & \text{otherwise} \end{cases}$;

for $n^{(\text{iters})}$ *iterations* **do**

  begin E-Step

    if *on first iteration* **then**

      $R_{ij} \longleftarrow \frac{1}{n^{(\text{out})}}$;

    else

      $P_{ij} \longleftarrow \frac{1}{\sqrt{\prod_{ch} 2\pi \left(\sigma_{jch}^{(\text{out})}\right)^2}} \exp\left(-\sum_{ch} \frac{\left(V_{ijch} - \mu_{jch}^{(\text{out})}\right)^2}{2\left(\sigma_{jch}^{(\text{out})}\right)^2}\right)$;

      $R_{ij} \longleftarrow \frac{f\left(a_j^{(\text{out})}\right) P_{ij}}{\sum_j f\left(a_j^{(\text{out})}\right) P_{ij}}$;

    end

  end

  begin D-Step

    $D_{ij}^{(\text{use})} \longleftarrow f(a_i^{(\text{inp})}) R_{ij}$;

    $D_{ij}^{(\text{ign})} \longleftarrow f(a_i^{(\text{inp})}) - D_{ij}^{(\text{use})}$;

  end

  begin M-Step

    $a_j^{(\text{out})} \longleftarrow \begin{cases} \sum_i \beta_{ij}^{(\text{use})} D_{ij}^{(\text{use})} - \sum_i \beta_{ij}^{(\text{ign})} D_{ij}^{(\text{ign})}, & \text{if } n^{(\text{inp})} \text{ fixed} \\ \sum_i \beta_j^{(\text{use})} D_{ij}^{(\text{use})} - \sum_i \beta_j^{(\text{ign})} D_{ij}^{(\text{ign})}, & \text{otherwise} \end{cases}$;

    $\mu_{jch}^{(\text{out})} \longleftarrow \frac{\sum_i D_{ij}^{(\text{use})} V_{ijch}}{\sum_i D_{ij}^{(\text{use})}}$;

    $(\sigma_{jch}^{(\text{out})})^2 \longleftarrow \frac{\sum_i D_{ij}^{(\text{use})} \left(V_{ijch} - \mu_{jch}^{(\text{out})}\right)^2}{\sum_i D_{ij}^{(\text{use})}}$;

  end

end

of size $d^{(\text{cov})} \times d^{(\text{out})}$.

When $n^{(\text{inp})}$ is left unspecified, we remove index i from the parameters used to compute V_{ijch}, so we reduce their size by a factor of $n^{(\text{inp})}$. In this case, the computation of V_{ijch} applies $n^{(\text{out})}$ simultaneous matrix-matrix multiplications to each input capsule, followed by addition of $n^{(\text{out})}$ corresponding ch biases.

2.1.1 Adapting to Variable-Size Outputs

A trivial adaptation of our algorithm, which we do not show in our equations for clarity and brevity, would be to allow both $n^{(\text{inp})}$ and $n^{(\text{out})}$ to be unspecified, allowing us to have a variable number of input capsules voting for an equal number of output capsules. We would remove indexes ij from the parameters used to compute V_{ijch}, reducing their size by a factor of $n^{(\text{inp})} \times n^{(\text{out})}$. In that case, the computation of V_{ijch} would be equivalent to applying the same linear transformation followed by addition of the same biases.

2.2 Routing Loop

Unlike previous versions of EM routing, which on each iteration compute first an M-Step and then an E-Step procedure, our algorithm computes these procedures in the conventional order: first the E-Step and then the M-Step. This ordering removes the final, superfluous E-Step from the loop, and also, we believe, simplifies exposition.

We also introduce a new procedure, which we call the *D-Step*, between the E-Step and M-Step. The D-Step computes the share of each input capsule's data used or ignored by each output capsule, for subsequent use in the computation of output scores $a_j^{(\text{out})}$. These computations, described in 2.4 and 2.5, represent our most significant departure from previous forms of EM routing.

Another difference is that in our algorithm, $a_i^{(\text{inp})}$ and $a_j^{(\text{out})}$ are "pre-activation" scores in the interval $[-\infty, \infty]$ on which we apply logistic functions as needed, at the last minute as it were, to compute activations. This trivial modification facilitates more flexible use of the "raw" values of $a_i^{(\text{inp})}$ and $a_j^{(\text{out})}$ by subsequent layers and/or objective functions, with more numerical stability. For example, a subsequent layer can apply a Softmax function to the output scores $a_j^{(\text{out})}$ to induce a distribution over output capsule activations.

In the following subsections, we describe the computations performed by the E-Step, D-Step, and M-Step on each iteration, in order of execution, emphasizing those computations which are new or different with respect to previous work.

2.3 E-Step

At the start of each iteration, for each sample, our E-Step computes an $n^{(\text{inp})} \times n^{(\text{out})}$ tensor R_{ij} of routing probabilities for assigning each input capsule i to each output capsule j,

$$R_{ij} = \begin{cases} \frac{1}{n^{(\text{out})}}, & \text{on first iteration} \\ \frac{f\left(a_j^{(\text{out})}\right) P_{ij}}{\sum_j f\left(a_j^{(\text{out})}\right) P_{ij}}, & \text{otherwise} \end{cases} \tag{2}$$

where f is the logistic function, $a_j^{(\text{out})}$ is the jth output score computed in the previous iteration's M-Step, and P_{ij} stands for

$$P\left(V_{ij} \;\middle|\; \text{capsule}_j^{(\text{out})} \sim \mathcal{N}\left(\mu_j^{(\text{out})}, (\sigma_j^{(\text{out})})^2\right)\right),$$

the probability density of capsule i's vote for capsule j (over its ch components), given capsule j's Gaussian model (of dimension $d^{(\text{cov})} \times d^{(\text{out})}$), updated in the previous iteration's M-Step, as in other forms of EM routing, except that in our case votes have two indexes (ch) instead of one.

In our implementation, for numerical stability, we compute R_{ij} after the first iteration by applying a Softmax function to simplified log-sums, instead of using the second equation in (2).

2.4 D-Step

At the beginning of each D-Step, we multiply the assigned routing probabilities R_{ij} by logistic function activations of input scores, which act as gates, to obtain $D_{ij}^{(\text{use})}$, the *share of data used* from each input capsule i to update each output capsule j's Gaussian model,

$$D_{ij}^{(\text{use})} = f\left(a_i^{(\text{inp})}\right) R_{ij}, \tag{3}$$

where f is the logistic function and $a_i^{(\text{inp})}$ is the input score associated with input capsule i. Except for the "last-minute" application of the logistic function, (3) is the same as its corresponding equation in previous forms of EM routing. However, our notation explicitly differentiates between R_{ij}, the routing probabilities, and $D_{ij}^{(\text{use})}$, the share of capsule i's data used by capsule j.

Each row of routing probabilities R_{ij} adds up to 1, and f maps $[-\infty, \infty]$ to $[0, 1]$; therefore,

$$0 \leq D_{ij}^{(\text{use})} \leq f\left(a_i^{(\text{inp})}\right) \leq 1,$$

that is, $D_{ij}^{(\text{use})}$ has values that range from 0 ("completely ignore input capsule i in output capsule j's model") to 1 ("use the whole of input capsule i in output capsule j's model"), but never exceeds $f(a_i^{(\text{inp})})$ ("how much of input capsule i can we use among all output capsules?").

Given these relationships, we can compute the *share of data ignored* (*i.e.*, not used) $D_{ij}^{(\text{ign})}$ from each input capsule i by each output capsule j,

$$D_{ij}^{(\text{ign})} = f\left(a_i^{(\text{inp})}\right) - D_{ij}^{(\text{use})}, \tag{4}$$

such that for each input and output capsule pair ij the two shares, $D_{ij}^{(\text{use})}$ and $D_{ij}^{(\text{ign})}$, plus the data that is "gated off" by logistic activation of the corresponding input score, $1 - f(a_i^{(\text{inp})})$, add up to 1, or the whole input capsule,

$$D_{ij}^{(\text{use})} + D_{ij}^{(\text{ign})} + \left(1 - f\left(a_i^{(\text{inp})}\right)\right) = 1,$$

accounting for all input data.

2.5 M-Step

The M-Step computes updated output scores $a_j^{(\text{out})}$ and weighted means and variances $\mu_{jch}^{(\text{out})}$ and $(\sigma_{jch}^{(\text{out})})^2$, respectively, to maximize the probability that each output capsule j's Gaussian model would generate the votes computed from each input capsule i used by j. We discuss these computations in the subsections that follow.

2.5.1 Output Scores

Previous forms of EM routing use the minimum description length principle to derive approximations of the *cost to activate* and the *cost not to activate* each output capsule j, and compute output activations by applying a logistic function to the difference between those approximations. Such costs must be approximated because the only known method for accurately computing them would require inverting $n^{(\text{inp})} \times n^{(\text{out})}$ vote-computation matrices, which is impractical. See Hinton et al. (2018) for details.

We use a different approach, motivated by the intuitive notion that

> *"output capsules should benefit from the input data they use, and lose benefits from any input data they ignore,"*

as they maximize the probability of votes from those input capsules they use in each iteration.

For each output capsule j, we compute output score $a_j^{(\text{out})}$ as the difference of a *net benefit to use* and a *net cost to ignore* input capsule data,

$$a_j^{(\text{out})} = \begin{cases} \sum_i \beta_{ij}^{(\text{use})} D_{ij}^{(\text{use})} - \sum_i \beta_{ij}^{(\text{ign})} D_{ij}^{(\text{ign})}, \\ \quad \text{if } n^{(\text{inp})} \text{ fixed} \\ \sum_i \beta_j^{(\text{use})} D_{ij}^{(\text{use})} - \sum_i \beta_j^{(\text{ign})} D_{ij}^{(\text{ign})}, \\ \quad \text{otherwise} \end{cases} \tag{5}$$

where $D_{ij}^{(\text{use})}$ and $D_{ij}^{(\text{ign})}$ are the shares of input capsule i's data used and ignored by output capsule j, computed in (3) and (4), respectively, and $\beta_{ij}^{(\text{use})}$ and $\beta_{ij}^{(\text{ign})}$ (or $\beta_j^{(\text{use})}$ and $\beta_j^{(\text{ign})}$, if $n^{(\text{inp})}$ is unspecified) are parameters representing each output capsule's net benefit to use and net cost to ignore input capsule data, respectively. The adjective "net" denotes that these parameters can have positive ("credits") or negative ("debits") values.

In the next iteration's E-Step, when we activate $a_j^{(\text{out})}$ by applying a logistic function to it in (2) for weighting P_{ij}, we induce in each output capsule a distribution $(p, 1-p)$ over a quantity equal to the output capsule's net benefits from using input capsules less its net costs from ignoring input capsules, while we maximize the probability that its Gaussian model would generate the votes computed from those capsules it uses, for optimization of a training objective specified elsewhere.

Informally, and more intuitively, we can think of this multi-faceted mechanism as finding, for each output capsule, "the combination of net benefits and costs that produces greater profit," where "greater profit" stands for maximizing input vote probabilities at each output capsule and optimizing the whole layer for another objective. We prefer to think of it as maximizing *"bang per bit."*

Finally, there is an interesting connection between our activation mechanism and that used by previous forms of EM routing: All else being equal, at each output capsule, using more data from an input capsule is associated with greater description length, and using less data from the input capsule is associated with the opposite.

2.5.2 Output Capsule Probabilistic Models

At the end of each M-Step, we compute updated means $\mu_{jch}^{(\text{out})}$ and variances $(\sigma_{jch}^{(\text{out})})^2$ of every output capsule j's Gaussian model, weighted by $D_{ij}^{(\text{use})}$,

the amount of data the output capsule uses from each input capsule i. See Alg. 1 for details.

3 Sample Application: smallNORB

The smallNORB dataset (LeCun et al., 2004) has grayscale stereo 96×96 images of five classes of toys: airplanes, cars, trucks, humans, and animals. There are 10 toys in each class, five selected for the training set and five for the test set. Each toy is photographed stereographically at 18 different azimuths (0-340 degrees), 9 different elevations, and 6 lighting conditions, such that the training and test sets each contain 24,300 pairs of images. Supp. Fig. 9 shows samples from each class.

3.1 Architecture

The architecture of our smallNORB model is described in detail in Fig. 3. At a high level of abstraction, we can think of the model as doing two things: First, it applies a sequence of standard convolutions to detect 64 toy parts and their 4×4 poses (spatial relationships to the viewer of the image) in multiple possible locations in the image (steps (a) through (d)) in Fig. 3). Then, the model applies two layers of our routing algorithm, one to detect 64 larger toy parts and their poses, and another to detect five categories of toys and their poses (Fig. 3(e)). The routing layers are meant to induce the standard convolutions to learn to recognize toy parts and their poses.

3.1.1 Use of Variable-Size Inputs

To keep the number of parameters small, we leave $n^{(\text{inp})}$ unspecified in the first routing layer, so it accepts a variable number of input capsules without regard for their location in the image. To counteract this loss of location information, we stack input images with two tensors of coordinate values evenly spaced from -1.0 to 1.0, one horizontally and one vertically, as shown in Fig. 3(b).

Besides reducing the number of parameters, our decision to accept a variable number of capsules in the first routing layer makes our model capable of accepting images of variable size, limited only by memory, though we do not make use of this capability here.

3.2 Initialization and Training

We initialize all convolutions with Kaiming normal initialization (He et al., 2015) and the two routing layers as follows: Normal initialization scaled by $\frac{1}{d^{(\text{inp})}}$ for the multilinear weights that compute votes, zeros for the bias parameters, and zeros for the net benefit and cost parameters.

We train the model for 50 epochs with a batch size of 20, and a learning rate that starts at 10^{-5}, increases linearly to 5×10^{-4} for the first 10% of training iterations, and then declines back to 10^{-5} with a cosine shape over the remaining iterations. We use the RAdam optimizer (Liu et al., 2019).

During training, we add 16 pixels of padding on each side to each pair of images and randomly crop them to 96×96 size. We do not use any other image processing, nor any metadata, nor any additional data in training.

For regularization, we use mixup (Zhang et al., 2017) with Beta distribution parameters (0.2, 0.2), inducing the iterative EM clustering algorithms in our routing layers to learn to distinguish samples that have been mixed together.

The objective function is a Cross Entropy loss, computed on Softmax activations of the output scores of the final routing layer's five capsules. Supp. Fig. 8 shows validation loss and accuracy after each epoch of training.

3.3 Results

Tab. 1 shows test set accuracy, number of parameters, and number of epochs to train our smallNORB model, and how it compares to prior capsule models that have achieved state-of-the-art results. Compared to the previous state of the art (Hinton et al., 2018), our model has fewer parameters, trains in an order of magnitude fewer epochs, and accepts full-size unprocessed images instead of downsampled, cropped ones. We do not use multiple crops per image to compute test accuracy.

Capsule Network	Test Set Accuracy	No. of Params	Train Epochs
Sabour et al. (2017)	97.3	8,200K	N/A
Ciresan et al. (2011)*	97.4	2700K	N/A
Hinton et al. (2018)	98.2	310K	300
Hinton et al. (2018)**	98.6	310K	300
Our Model	**99.1**	**273K**	**50**

* Extensive image processing in training.

** Reported accuracy is mean of multiple random crops per image.

Table 1: Test set accuracy, parameters, and number of training epochs of models with state-of-the-art performance on smallNORB.

3.4 Analysis

We find evidence that our smallNORB model learns to perform its own form of "reverse graph-

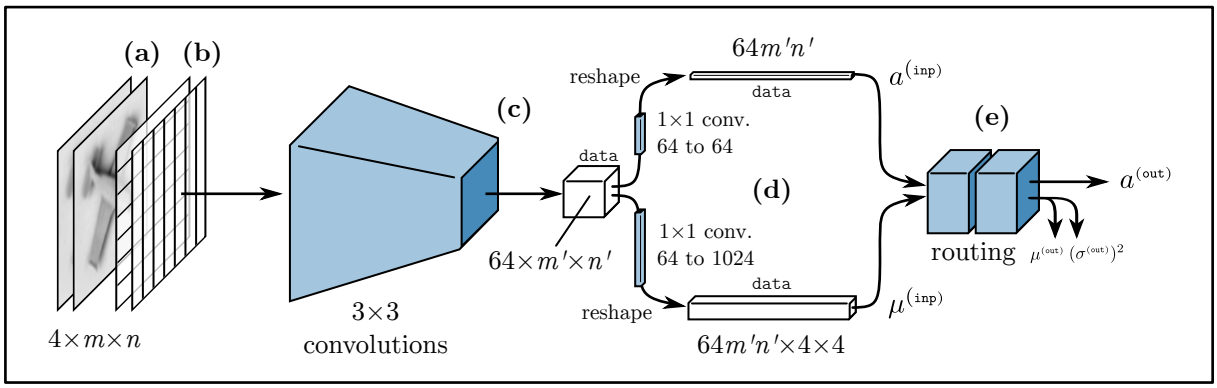

Figure 3: Our smallNORB model. **(a)** We stack each pair of images with **(b)** coordinate values evenly spaced from -1.0 to 1.0, horizontally and vertically, creating an input tensor of shape $4 \times m \times n$, where $m = n = 96$ for unmodified images at test time. **(c)** We apply six 3×3 convolutions, each with 64 output channels and alternating strides of 1 and 2. Each convolution is preceded by batch normalization and followed by a Swish activation (Ramachandran et al., 2017) with constant $\beta = 1$. The last convolution outputs a tensor of shape $64\times m'\times n'$. **(d)** We compute $a^{(\text{inp})}$ and $\mu^{(\text{inp})}$ by applying two 1×1 convolutions, with 64 and 1024 output channels, respectively, and reshape them as shown. Both convolutions are preceded by batch normalization. After reshaping, $a^{(\text{inp})}$ consists of $64m'n'$ input scores, representing possible presence or absence of 64 toy parts in $m'n'$ image locations. $\mu^{(\text{inp})}$ consists of $64m'n'$ slices of shape 4×4, each representing a pose for one of 64 parts in $m'n'$ locations. **(e)** We apply two layers of our routing algorithm; the first one routes a variable number of input capsules to 64 output capsules, each representing a larger toy part with a 4×4 pose; the second one routes those capsules to five capsules, each representing a type of toy with a 4×4 pose. For prediction, we apply a Softmax to $a^{(\text{out})}$.

ics" without explicitly optimizing for it, solely from pixel data and classification labels. It learns to use all four pose vectors jointly to represent poses, in a way that feels quite alien compared to the typical human approach (*e.g.*, a 3×3 matrix inside a 4×4 matrix with translation data).

The visualization in Fig. 4 shows multidimensional scaling (MDS) representations in $\mathbb{R}^2$ of the trajectories of an activated class capsule's pose vectors as we feed test images of an object in the class with varying elevations to our model. We can see that the four pose vectors that constitute the class capsule jointly move and eventually seem to "flip" as we change viewpoint elevation. The same visualization for other objects in the dataset, and for varying azimuth, look qualitatively similar.

We also analyze quantitatively how pose vectors change as we vary azimuth and elevation for every category and instance of toy in the dataset, and find that pose vectors change in ways that are consistent with variation in azimuth and elevation. See the 24 plots and their captions in Supp. Fig. 6 and Supp. Fig. 7 for details.

Much more work remains to be done to understand and quantify our routing algorithm's ability to learn "reverse graphics." However, we think such work falls outside the scope of this paper, given that we also evaluate our routing algorithm in another domain, natural language.

4 Sample Application: SST

The Stanford Sentiment Treebank (SST) (Socher et al., 2013) consists of 11,855 sentences extracted from movie reviews, parsed into trees with 215,154 unique phrases. The root sentences are split into training (8,544), validation (1,101) and test (2,210) sets, each with its own subset of the parse trees. The fine-grained root sentence classification task (SST-5/R) involves selecting one of five labels (`very negative`, `negative`, `neutral`, `positive`, `very positive`) for each root sentence in the test set. The binary root sentence classification task (SST-2/R) involves selecting one of two labels (`negative`, `positive`) after removing all `neutral` samples from the dataset, leaving 9,613 root sentences, split into training (6,920), validation (872), and test (1,821) sets, each with their own subset of the parse trees.

We chose SST, and SST-5/R in particular, for three reasons: First, its size is small enough to

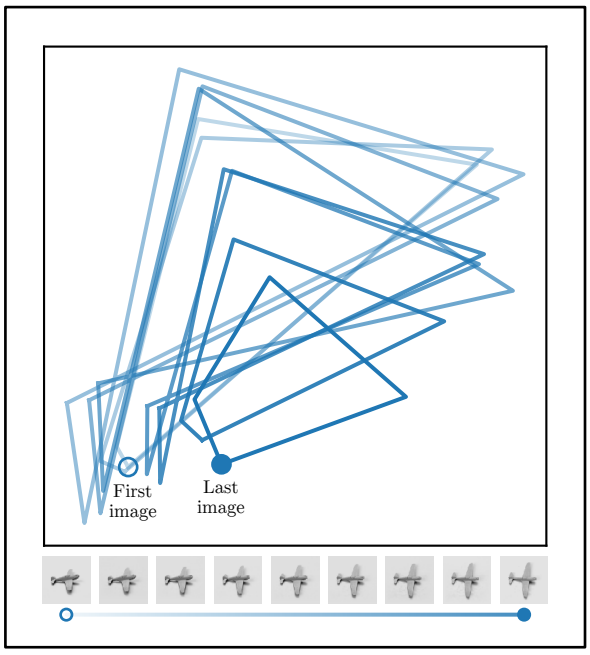

Figure 4: Multidimensional scaling (MDS) representations in $\mathbb{R}^2$ of the trajectories of an activated class capsule's four pose vectors, each of size $d^{(\text{out})} = 4$, as we feed test images of an object in the class with varying elevations to our trained smallNORB model. For each image, the $\mathbb{R}^2$ coordinates are plotted as four connected vertices, each vertex corresponding to a pose vector, preserving as much as possible the pairwise distances between pose vectors from all images. Circles indicate the activated capsule's first pose vector for the first and last image.

avoid the challenges to scaling mentioned earlier. Second, since this dataset's release in 2013, no model has come close to human performance on SST-5/R, as measured by accuracy on its labels, which were assigned by human beings. Finally, we suspect SST-5/R has remained challenging because it is less susceptible than other language tasks to the "Clever Hans" effect,[2] thanks to its small number of samples (only 2,210 test sentences), variety of linguistic constructions (*e.g.*, idiosyncratic movie fan idioms), complex syntactic phenomena (*e.g.*, nested negations), and subtle labels (*e.g.*, `neutral`). We think these qualities make it more difficult for models to find and exploit spurious statistical cues in the data.

[2]The Clever Hans effect occurs when seemingly impressive model performance is explained by exploitation of spurious statistical cues in the data. The effect has been documented in various natural language datasets, for example, by McCoy et al. (2019) and Niven and Kao (2019).

4.1 Architecture

The architecture of our SST model closely resembles that of our smallNORB model. We show and describe it in detail in Fig. 5. At a high level of abstraction, we can think of our SST model as doing two things: First, it applies a nonlinear transformation to every embedding from a pretrained transformer, mapping each one to a vector with 64 elements indicating present or absence of "parts of sentiments" (steps (a) through (d)) in Fig. 5). Then, the model applies two layers of our routing algorithm, one to detect 64 composite parts, and another to detect classification labels, five for SST-5/R and two for SST-2/R (Fig. 5(e)). The routing layers induce the nonlinear transformation to learn to recognize useful "parts of sentiments."

4.1.1 Use of Variable-Size Inputs

The number of tokens in sentences is variable, so we leave $n^{(\text{inp})}$ unspecified in the first routing layer of our SST model. This first routing layer accepts any number of input capsules without regard for their position in the sequence or the depth of the transformer layer from which they originate.

Transformer embeddings incorporate information about their position in a sequence, but not about layer depth. To counteract the loss of depth information, we add a "depth-of-layer" parameter to the input tensor, as shown in Fig. 5(b). In this parameter, each transformer layer has a corresponding vector slice, which we add element-wise to every embedding in the input tensor originating from that transformer layer.

4.1.2 Use of GPT-2

In our implementation, we use a GPT-2_{large} (Radford et al., 2018) as the pretrained transformer. We chose this model mainly because it was the largest one publicly available at the time of writing.

GPT-2_{large} has approximately 774 million parameters and outputs 37 layers (36 hidden plus one visible) of dimension 1280. If a sentence contains, say, 10 tokens, this GPT-2 model transforms it into 37 sequences, each with 10 embeddings of size 1280. We concatenate them into an input tensor of shape $10 \times 37 \times 1280$ for our SST model.

4.2 Initialization and Training

For both tasks, SST-5/R and SST-2/R, we initialize the routing layers exactly as those in the smallNORB model, the depth embeddings with zeros,

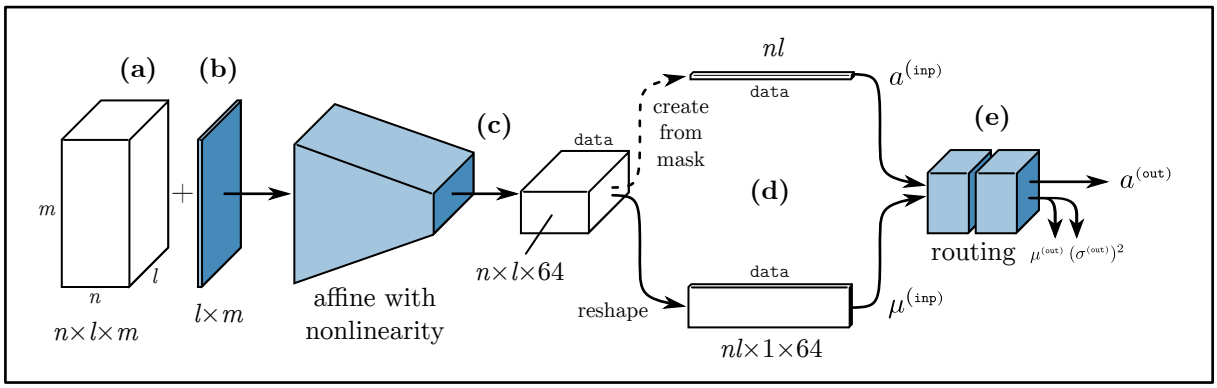

Figure 5: Our SST model. **(a)** For each sample, the input is a tensor of transformer embeddings of shape $n \times l \times m$, where n is the number of tokens, l is the number of transformer layers, and m is the embedding size. **(b)** We add to the input tensor a depth-of-layer parameter of shape $l \times m$. **(c)** We apply an affine transformation that maps m to 64, preceded by layer normalization on m and followed by a Swish activation with constant $\beta = 1$, obtaining a tensor of shape $n \times l \times 64$. **(d)** We reshape the tensor as shown to obtain $\mu^{(\text{inp})}$, consisting of ln input capsules of size 1×64. We create $a^{(\text{inp})} \longleftarrow \log(\frac{x}{1-x})$ from a mask x of length nl with ones and zeros indicating, respectively, which embeddings correspond to token ids and which correspond to any padding necessary to group samples in batches. In other words, $a^{(\text{inp})}$ is a vector of logits whose elements are equal to ∞ for token ids and $-\infty$ for padding (*e.g.*, `float("inf")` and `float("-inf")` in Python). **(e)** We apply two layers of our routing algorithm; the first one routes a variable number of capsules in $\mu^{(\text{inp})}$ to 64 capsules of shape 1×2; the second one routes those capsules to five or two capsules of equal shape, each representing a classification label in SST-5/R or SST-2/R. For prediction, we apply a Softmax to output scores $a^{(\text{out})}$.

and the affine map of the nonlinear transformation with Kaiming normal.

For both tasks, the training regime is exactly the same as for the smallNORB model, except that we train the SST model for only 3 epochs. We train the model on all unique token sequences in the parse trees of the training set. Supp. Fig. 8 shows loss and accuracy on the root-sentence validation set after each epoch of training for both SST-5/R and SST-2/R.

4.3 Results on SST-5/R

As Tab. 2 shows, we achieve new state-of-the-art test set accuracy of 58.5% on SST-5/R, which is a significant improvement (2.3 percentage points) over previous state-of-the-art test set accuracy.

4.4 Results on SST-2/R

The same model (changing only the final number of capsules to two) achieves 95.6% test set accuracy on SST-2/R, a new state-of-the-art performance for single-task models. State-of-the-art test set accuracy for models or ensembles trained on multiple tasks is 96.8% (Yang et al., 2019). See Tab. 3.

Architecture/Pretraining	SST-5/R
RNTN/none (Socher et al., 2013)	45.7
CNN/word2vec (Kim et al., 2014)	48.0
Para-Vec/on dataset (Le and Mikolov, 2014)	48.7
LSTM/on PP2B (Wieting et al., 2015)	49.1
DMN/GloVe (Kumar et al., 2015)	51.1
NSE/GloVe (Mundkhdalai and Yu, 2016)	52.8
ByteLSTM/82M reviews (Radford et al., 2017)	52.9
CT-LSTM/word2vec (Looks et al., 2017)	53.6
BCN+Char/CoVe (McCann et al., 2017)	53.7
BCN/ELMo (Peters et al., 2018)	54.7
SuBiLSTM+Char/CoVe (Brahma, 2018)	56.2
Our Model/GPT-2_{large}	**58.5**

Table 2: State-of-the-art test set accuracy on SST-5/R since its publication is associated with both new architectures and new pretraining methods.

4.5 Analysis

We are surprised to see such a significant improvement over the state of the art on SST-5/R, and a match of state-of-the-art performance on SST-2/R compared to previous single-task models, considering that (a) we do *not* finetune the transformer model, (b) we use a model that closely resembles the one we use for a visual task, and (c) we train both this model and the one for a visual task with exactly the same training regime (the only differ-

Architecture	SST-2/R
Multi-task models or ensembles:	
Snorkel (Ratner et al., 2018) (ensemble)	96.2
MT-DNN (Liu et al., 2019) (single model)	95.6
MT-DNN (Liu et al., 2019) (ensemble)	96.5
XLNet (Yang et al., 2019) (ensemble)	**96.8**
Single-task models:	
BCN+Char+CoVe (McCann et al., 2017)	90.3
Block-sparse LSTM (Radford et al., 2017)	93.2
BERT (Devlin et al., 2018)	94.9
Our Model+GPT-2_{large}	**95.6**

Table 3: Recent state-of-the-art test set accuracy on SST-2/R by multi-task models or ensembles and by single-task models like ours.

ence is in the number of epochs we train).

We also note that progress on SST-5/R has been remarkably steady, year after year, since its publication in 2013, as AI researchers have devised new architectures that exploit new pretraining mechanisms (Tab. 2). Our results represent a continuation of this long-term trend. As of yet, no model has come close to accurately modeling the labeling decisions of human beings on SST-5/R. Performance on the binary task, SST-2/R, whose labels lack the subtlety of the finegrained ones, has been close to human baseline for several years now.

Finally, our successful use of a capsule network to route embeddings from a pretrained transformer links two areas of AI research that have been largely independent from each other: capsule networks with routing by agreement, used mainly for visual tasks, and transformers with self-attention, used mainly for sequence tasks.

5 Related Work

Hinton et al. (2018) proposed the first form of EM routing and showed that capsule networks using it to route matrix capsules can generalize to different poses of objects in images and resist white-box adversarial attacks better than conventional CNNs. Their "related work" section compares capsule networks to other efforts for improving the ability of visual recognition models to deal effectively with viewpoint variations.

Sabour et al. (2017) showed that capsule networks with an earlier form of routing by agreement, operating on vector capsules, can be more effective than conventional CNNs for segmenting highly overlapping images of digits.

Barham and Isard (2019) showed that current challenges to scaling capsule networks are due mainly to current software (*e.g.*, PyTorch, TensorFlow) and hardware (*e.g.*, GPUs, TPUs) systems, which are highly optimized for a fairly small set of computational kernels, in a way that is tightly coupled with memory hardware, leading to poor performance on non-standard workloads, including basic operations on capsules.

Coenen et al. (2019) found evidence that BERT, and possibly other transformer architectures, learn to embed sequences of natural language as *trees*. Their work inspired us to wonder if capsule networks might be able to recognize such embedded "language trees" in different "poses," analogously to the way in which capsule networks can recognize different poses of objects embedded in images.

Vaswani et al. (2017) proposed transformer models using query-key-value dot-product attention, and showed that such models can be more effective than prior methods for modeling sequences. EM routing algorithm can be seen as a new kind of attention mechanism in which output capsules "compete with each other" for the attention of input capsules, with each output capsule seeing a different set of input capsule votes.

6 Future Work

Capsule networks are a recent innovation, and our routing algorithm is even more recent. Its behavior and properties are not yet fully understood. As the challenges to scaling mentioned in 1.2 are overcome, we think it would make sense to conduct comprehensive evaluations of our algorithm in multiple domains.

We are also intrigued about using our routing algorithm for natural language modeling. At present this seems impractical, due not only to the aforementioned challenges to scaling, but also to the computational complexity of the algorithm itself. Consider: If we wanted to use our routing algorithm to predict the next capsule in a natural language sequence of capsules, over a dictionary of typical size, say, 3×10^4 subword ids, every layer of the model would have to execute 3×10^4 simultaneous iterative EM algorithms. A more practical alternative in the near future might be to construct models combining blocks of query-key-value self-attention layers, which are currently more practical, with layers of our routing algorithm.

Another possible avenue for future research involves experimenting with probabilistic models

other than a multidimensional Gaussian in output capsules. While our limited experiments show that a multidimensional Gaussian works remarkably well, we harbor some doubts about its effectiveness with capsules of much greater size.

Finally, we wonder about using non-probabilistic clustering in our routing algorithm, k-means being the most obvious choice, given its relationship to EM and its proven effectiveness handling large-scale data in other settings.

7 Conclusion

Building on recent work on capsule networks, we propose a new form of routing by agreement that computes output capsule activations as a logistic function of net benefits to use and net costs to ignore input capsules. To make this computation possible, we introduce a new step in the EM routing loop, the D-Step, for computing the share of data used and ignored from each input capsule by each output capsule, accounting for all input capsule data. We construct our routing algorithm to accept variable-size inputs, which proves useful not only for domains other than vision, but also for keeping the number of model parameters small. Finally, we use "pre-activation" scores to which we apply logistic functions as needed, facilitating more flexible use of these scores by subsequent layers and/or objective functions, with more numerical stability.

As sample applications, we present two capsule networks using our routing algorithm in two domains, vision and language. Our first network achieves state-of-the-art performance in smallNORB with fewer parameters and an order of magnitude less training than previous capsule networks, and we find evidence that it learns a form of "reverse graphics" from pixel data and classification labels. Our second network achieves new state-of-the-art performance classifying the root sentences of the Stanford Sentiment Treebank into fine-grained and binary labels by routing a pre-trained transformer's embeddings as capsules.

Acknowledgments

We thank Russell T. Klophaus for his feedback.

References

Paul Barham and Michael Isard. 2019. Machine learning systems are stuck in a rut. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. ACM, New York, NY, USA, HotOS '19, pages 177–183.

Dan C. Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. 2011. High-performance neural networks for visual object classification. *CoRR* abs/1102.0183.

Andy Coenen, Emily Reif, Ann Yuan, Been Kim, Adam Pearce, Fernanda B. Viégas, and Martin Wattenberg. 2019. Visualizing and measuring the geometry of BERT. *CoRR* abs/1906.02715.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR* abs/1810.04805.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR* abs/1502.01852.

Geoffrey Hinton, Sara Sabour, and Nicholas Frosst. 2018. Matrix capsules with em routing.

Yann LeCun, Fu Jie Huang, and Léon Bottou. 2004. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Washington, DC, USA, CVPR'04, pages 97–104.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2019. On the variance of the adaptive learning rate and beyond.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, pages 3428–3448.

Timothy Niven and Hung-Yu Kao. 2019. Probing neural network comprehension of natural language arguments. *CoRR* abs/1907.07355.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018. Language models are unsupervised multitask learners .

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for activation functions. *CoRR* abs/1710.05941.

Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. 2017. Dynamic routing between capsules. *CoRR* abs/1710.09829.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Citeseer, volume 1631, page 1642.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR* abs/1706.03762.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR* abs/1906.08237.

Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *CoRR* abs/1710.09412.

List of Supplementary Figures

- Fig. 6: Change in pose vectors by azimuth.
- Fig. 7: Change in pose vectors by elevation.
- Fig. 8: Validation losses and accuracies.
- Fig. 9: Sample smallNORB images.

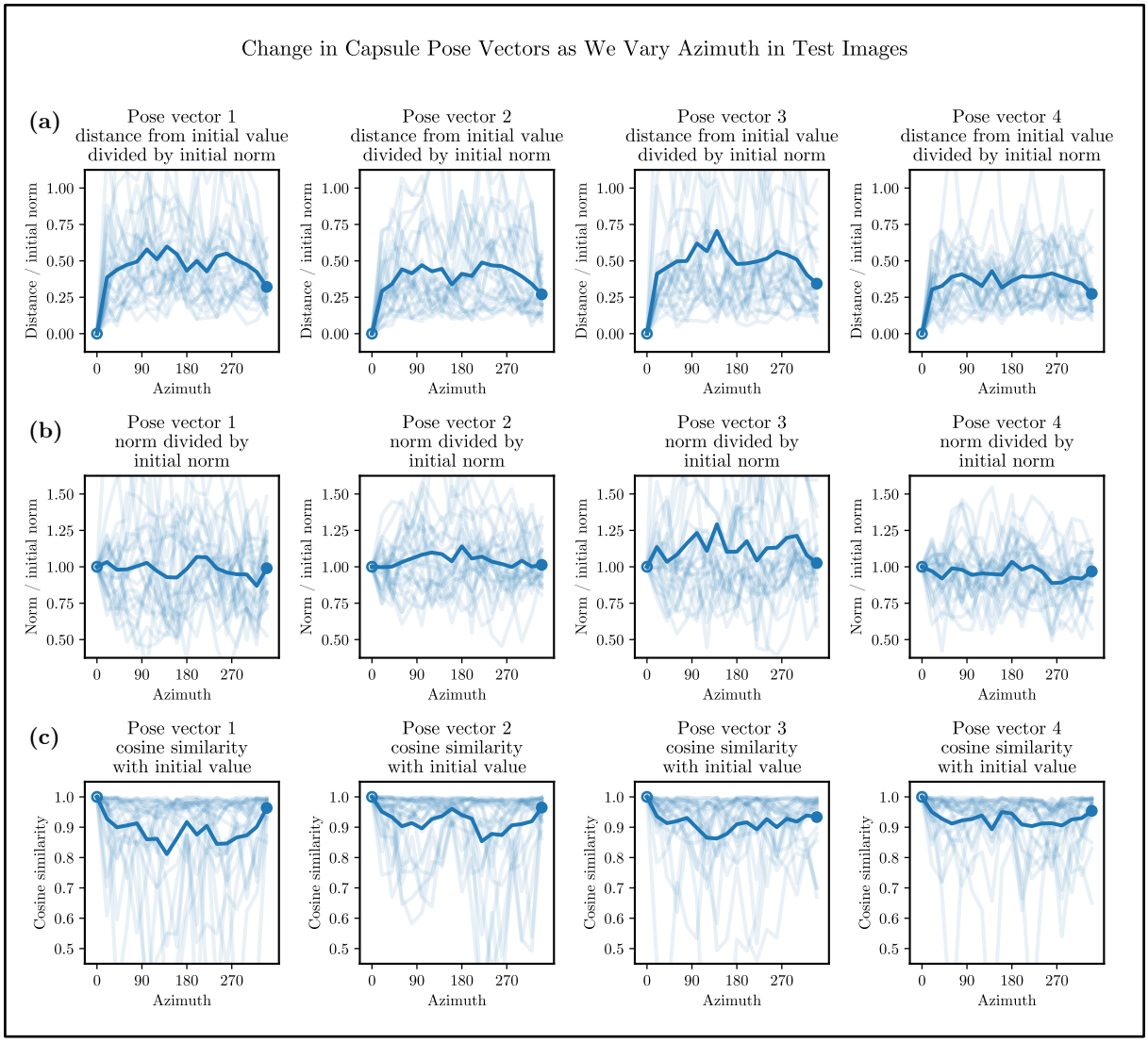

Figure 6: For each toy category and instance in the test set, we feed our smallNORB model a sequence of images with varying azimuth, while keeping everything else constant, and plot the change in each pose vector of the activated capsule. The mean change is shown in dark blue. **(a)** The first row of plots show each pose vector's Euclidean distance to its initial value, divided by the norm of the initial value, as azimuth varies from 0 to 340 degrees. We can see that the pose vector tends to move away from and then back close to its initial value, consistent with rotation. **(b)** The second row shows the norm of all pose vectors, divided by their initial norms. We can see that pose vector norms tend to stay close to the initial norm, consistent with rotation. **(c)** The third row shows cosine similarity between pose vectors and their initial value. We can see that the angle tends to increase and then decrease, consistent with rotation.

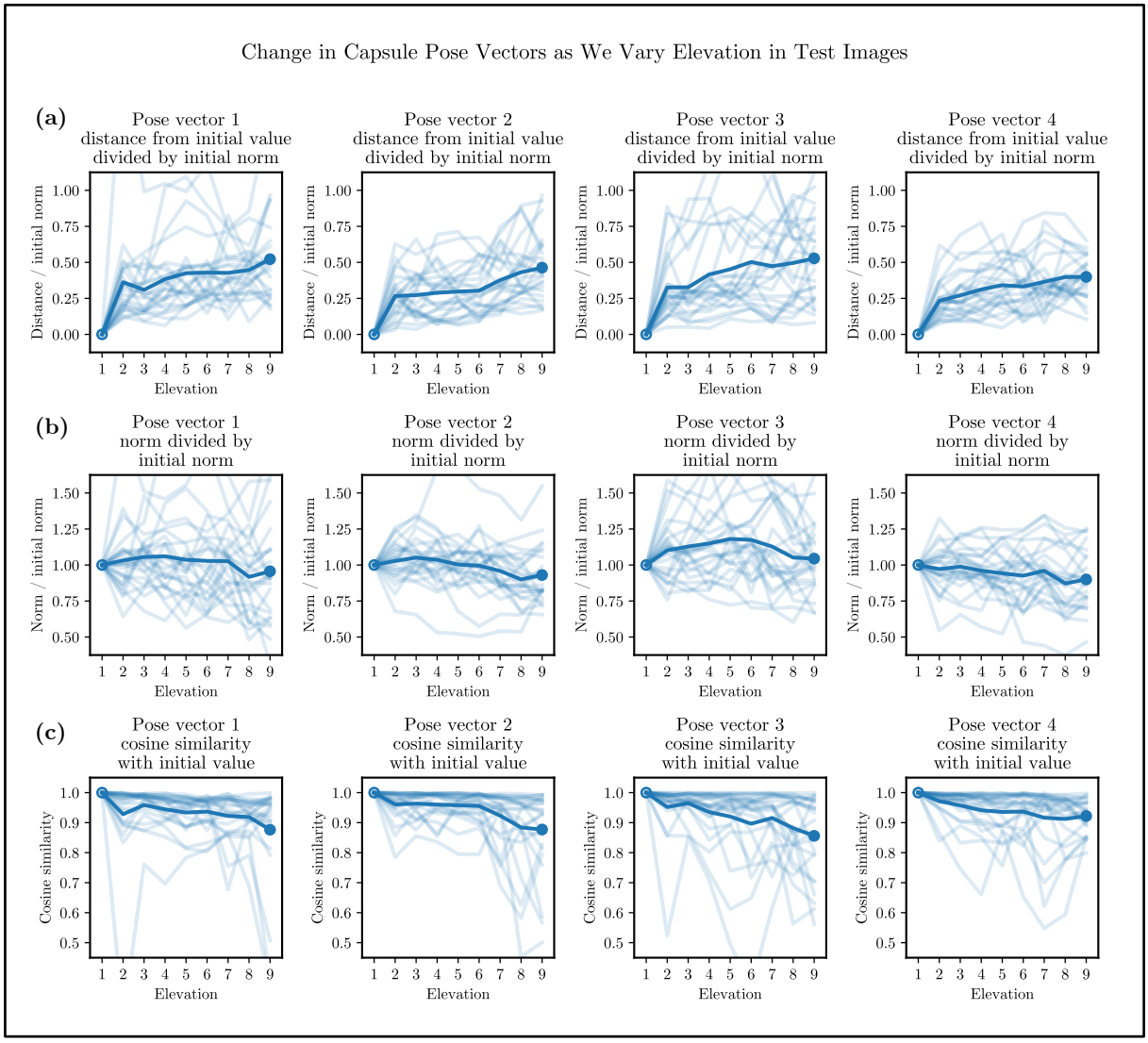

Figure 7: For each toy category and instance in the test set, we feed our smallNORB model a sequence of images with varying elevation, while keeping everything else constant, and plot the change in each pose vector of the activated capsule. The mean change is shown in dark blue. **(a)** The first row of plots show each pose vector's Euclidean distance to its initial value, divided by the norm of the initial value, as elevation varies through nine levels (from near flat to looking from above). We can see that the pose vector moves away from its initial value, consistent with the change in elevation. **(b)** The second row shows the norm of all pose vectors, divided by their initial norms. We can see that pose vector norms tend to stay close to the initial norm, consistent with rotation due to the change in elevation. **(c)** The third row shows cosine similarity between each pose vector and its initial value. We can see that the angle tends to increase but not decrease back to its original value, consistent with the change in elevation.

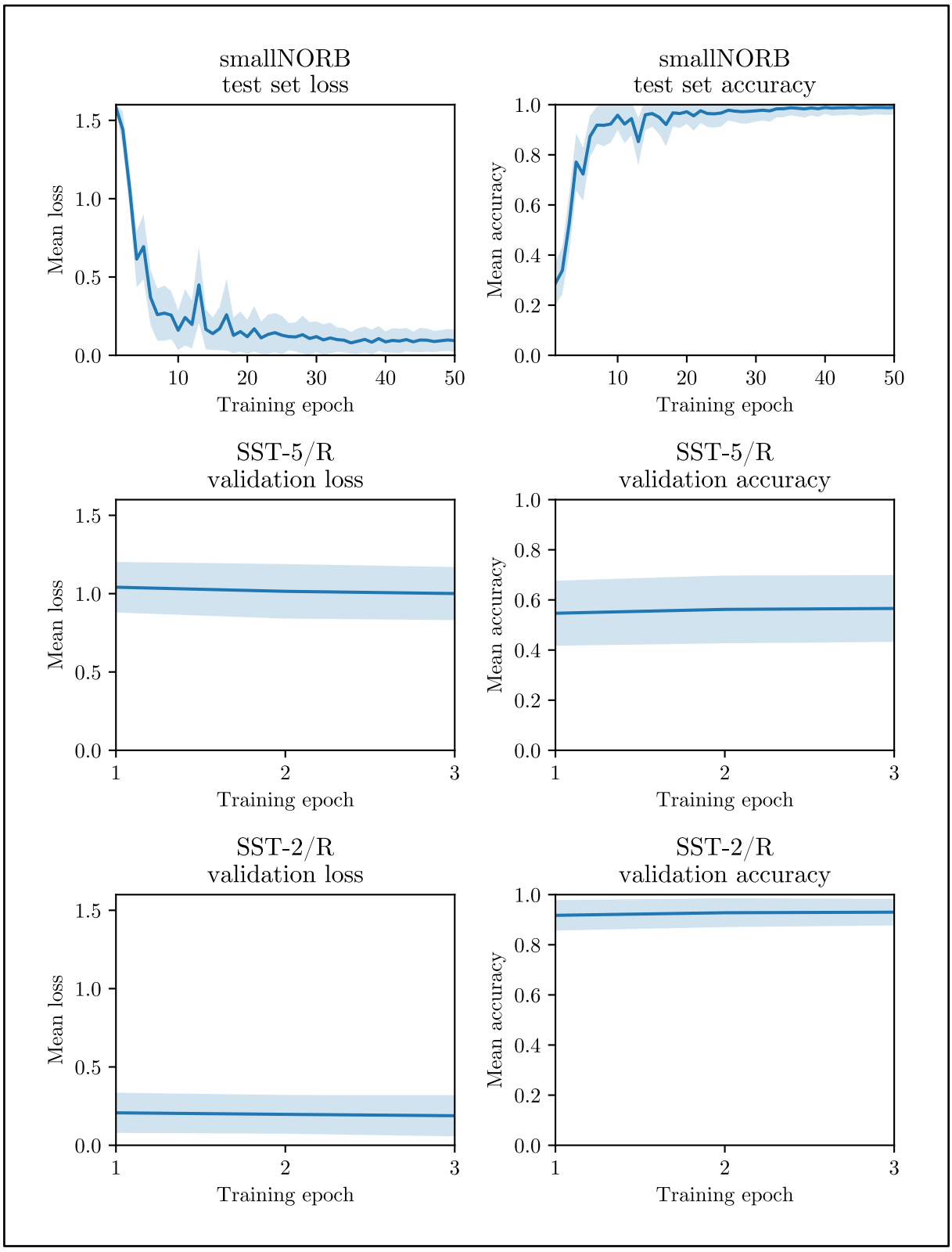

Figure 8: Mean validation loss and accuracy after each epoch of training. Shaded area denotes standard deviation of batches, with 20 samples each. Note: smallNORB dataset does not have a validation split.

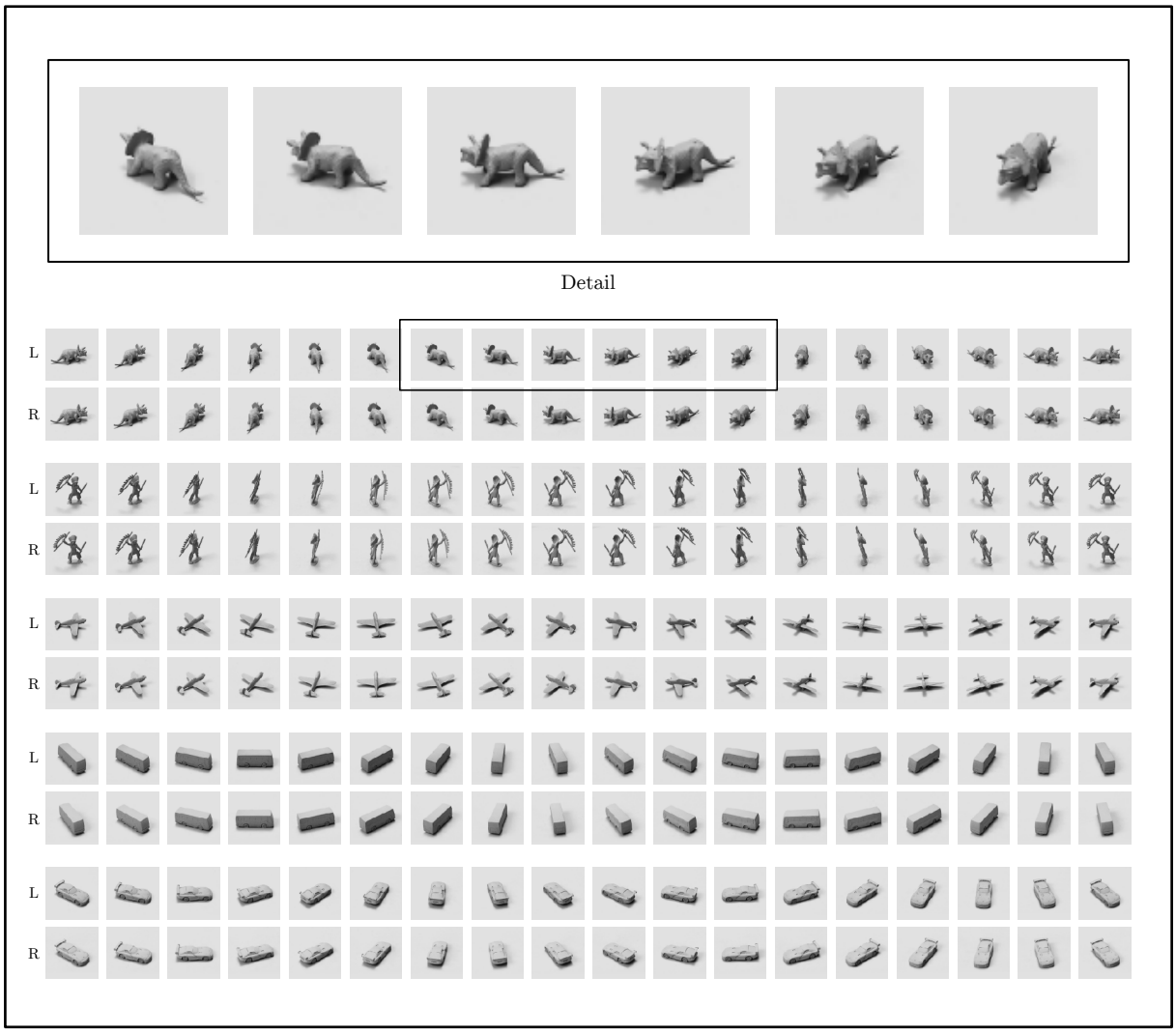

Figure 9: Sample smallNORB stereographic images of one toy in each of five toy categories. For each toy we show 18 image-pair samples of varying azimuth while keeping elevation and lighting constant.