An EM Algorithm for Capsule Regression

Lawrence K. Saul
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive, Mail Code 0404
La Jolla, CA 92093-0404

Abstract

We investigate a latent variable model for multinomial classification inspired by recent capsule architectures for visual object recognition (Sabour et al., 2017). Capsule architectures use vectors of hidden unit activities to encode the pose of visual objects in an image, and they use the lengths of these vectors to encode the probabilities that objects are present. Probabilities from different capsules can also be propagated through deep multilayer networks to model the part-whole relationships of more complex objects. Notwithstanding the promise of these networks, there still remains much to understand about capsules as primitive computing elements in their own right. In this paper, we study the problem of capsule regression—a higher-dimensional analogue of logistic, probit, and softmax regression in which class probabilities are derived from vectors of competing magnitude. To start, we propose a simple capsule architecture for multinomial classification: the architecture has one capsule per class, and each capsule uses a weight matrix
to compute the vector of hidden unit activities for patterns it seeks to recognize. Next, we show how to model these hidden unit activities as latent variables, and we use a squashing nonlinearity to convert their magnitudes as vectors into normalized probabilities for multinomial classification. When different capsules compete to recognize the same pattern, the squashing nonlinearity induces non-Gaussian terms in the posterior distribution over their latent variables. Nevertheless, we show that exact inference remains tractable and use an Expectation-Maximization procedure to derive least-squares updates for each capsule’s weight matrix. We also present experimental results to demonstrate how these ideas work in practice.

1 Introduction

Recently Sabour et al. (2017) introduced a novel capsule-based architecture for visual object recognition. A capsule is a group of hidden units that responds maximally to the presence of a particular object (or object part) in an image. But most importantly, the capsule responds to this presence in a specific way: its hidden units encode a pose vector for the object—a vector that varies (for instance) with the object’s position and orientation—while the length of this vector encodes the probability that the object is present. Capsules were conceived by Hinton et al. (2011) to address a shortcoming of convolutional neural nets, whose hidden representations of objects in deeper layers are designed to be invariant to changes in pose. With such representations, it is difficult to model the spatial relationships between different objects in the same image. By contrast, the pose vectors in capsules learn equivariant representations of visual objects: these vectors do change with the pose, but only in such a way that the object is still recognized with high probability. These ideas have led to a surge of interest in multilayer capsule networks for increasingly difficult problems in computer vision (Duarte et al.,
Much of this work has focused on the message-passing between capsules in different layers, as is needed to model the part-whole relationships of complex objects or entire scenes (Wang and Liu, 2018; Bahadori, 2018; Hinton et al., 2018; Jeong et al., 2019; Hahn et al., 2019; Ahmed and Torresani, 2019; Tsai et al., 2020; Venkataraman et al., 2020). But capsules also introduced a novel paradigm for subspace learning that deserves to be explored—and elucidated—in its own right (Zhang et al., 2018). Consider, for instance, an individual capsule: its vector of hidden unit activities already represents a powerful generalization of the scalar activity computed by a simple neural element. In this paper, we shall discover an additional source of richness by modeling these hidden unit activities as latent variables in a probabilistic graphical model. The models we study are not exactly a special case of previous work, but they are directly motivated by it. As Sabour et al. (2017) wrote, “there are many possible ways to implement the general idea of capsules ... We want the length of the output vector of a capsule to represent the probability that the entity represented by the capsule is present in the current input.” This general idea is also the starting point for our work.

In this paper, we study the problem of capsule regression, a problem in which multiple capsules must learn in tandem how to map inputs to pose vectors with competing magnitudes. We have written the paper with two readers in mind. The first is a practitioner of deep learning. She will view our models as a kind of evolutionary precursor to existing capsule networks; certainly, she will recognize at once how pose vectors are computed from inputs and converted via squashing nonlinearities into normalized probabilities. The second reader we have in mind is the working data scientist. Where the paper succeeds, she will view our models as a natural generalization of logistic and softmax regression—essentially, a higher-dimensional analogue of these workhorses in
which vectors compete in magnitude to determine how inputs should be classified. We hope that both types of readers see the novel possibilities for learning that these models afford.

The organization of this paper is as follows. In section 2, we formulate our latent variable model for capsule regression. Despite the model’s squashing nonlinearity, we show that exact inference remains tractable and use an Expectation-Maximization procedure (Dempster et al., 1977) to derive least-squares updates for each capsule’s weight matrix. In section 3, we present experimental results on images of handwritten digits and fashion items. Our results highlight how capsules use their internal distributed representations to learn more accurate classifiers. In section 4, we discuss issues that are deserving of further study, such as regularization, scaling, and model-building. Finally, in the appendices, we fill in the technical details and design choices that were omitted from the main development.

2 Model

Our model can be visualized as the belief network with latent variables shown in Figure 1, and our mathematical development is based closely on this representation. Section 2.1 gives an overview of the model and its EM algorithm for parameter estimation; here we cover what is necessary to understand the model at a high level, though not all of what is required to implement it in practice. The later sections fill in these gaps. Section 2.2 focuses on the problem of inference; here we show how to calculate likelihoods and statistics of the posterior distribution of our model. Section 2.3 focuses on the problem of learning; here we show how the EM algorithm uses the model’s posterior statistics to iteratively re-estimate its parameters. Finally, section 2.4 describes a simple
heuristic for initializing the model parameters, based on singular value decomposition, that seems to work well in practice.

2.1 Overview

We use the model in Figure 1 for multinomial classification—that is, we use the model to parameterize the conditional probability $P(y|x)$ where $x \in \mathbb{R}^p$ is a vector-valued input and $y \in \{1, 2, \ldots, m\}$ is a class label. The model predicts the class label $y$ based on the magnitudes of the $m$ vector-valued latent variables $\{h_1, h_2, \ldots, h_m\}$; in particular, for each input $x$, the most likely label $y$ is determined by whichever vector $h_i$ has the largest magnitude. The model’s prediction depends essentially on three constructions: first, how the latent variables $h_i$ depend on the input $x$; second, how the class label $y$ depends on the magnitudes $\|h_i\|$ of these latent variables; and finally, how the prediction $P(y|x)$ depends on the distribution over these magnitudes. We now describe each of these in turn.

The model assumes that the latent variables $\{h_1, h_2, \ldots, h_m\}$ are conditionally in-

$$P(y=j|h_1, h_2, \ldots, h_m) = \frac{\|h_j\|^2}{\sum_{i=1}^m \|h_i\|^2}$$

$$h_i \sim N(W_i x, \sigma^2 I)$$
dependent given the input $x$. Thus we may write

$$P(h_1, h_2, \ldots, h_m | x) = \prod_{i=1}^{m} P(h_i | x),$$  \hspace{1cm} (1)$$

where the conditional independence is represented in the network of Figure 1 by the absence of edges between nodes in its intermediate layer. In addition, the model assumes that each latent variable $h_i \in \mathbb{R}^d$ is normally distributed as

$$P(h_i | x) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp \left( -\frac{1}{2\sigma^2} \| h_i - W_i x \|^2 \right)$$  \hspace{1cm} (2)$$

where the weight matrix $W_i \in \mathbb{R}^{d \times p}$ is a model parameter that must be learned from examples. Each weight matrix $W_i$ specifies a linear transformation from the input space to the latent space, and from eq. (2), we see that it determines the conditional mean $E[h_i | x] = W_i x$ of the latent variable $h_i$. We have not added an explicit offset (or bias term) to this linear transformation, but if desired, the same effect can be accomplished by appending the input vector with an extra element equal to unity.

There is one additional model parameter in eq. (2), namely the variance $\sigma^2$, which determines the sharpness of the Gaussian distribution, and which (unlike the weight matrix) we assume is common to all the distributions $P(h_1 | x), \ldots, P(h_m | x)$ that appear in eq. (1). We note that the model has an especially simple behavior in the limits $\sigma^2 \rightarrow 0$ and $\sigma^2 \rightarrow \infty$: in the former, the latent variables are deterministically specified by $h_i = W_i x$, while in the latter, they are completely delocalized. Though these limits are trivial, we shall see in section 2.2 that many non-trivial aspects of the model can be exactly calculated by a simple interpolation between these two regimes.

Next we describe how the class label $y$ depends on the model’s latent variables. Inspired by previous capsule architectures (Sabour et al., 2017), the model uses a
squashing nonlinearity to convert the magnitudes $\|h_i\|$ into normalized probabilities $P(y|h_1, h_2, \ldots, h_m)$ for multinomial classification. In particular, the model assumes that

$$P(y = j|h_1, h_2, \ldots, h_m) = \frac{\|h_j\|^2}{\sum_{i=1}^m \|h_i\|^2}. \tag{3}$$

We note that eq. (3) is used to compute probabilities for $m \geq 2$ outcomes, while the squashing nonlinearity in Sabour et al. (2017) was used to compute probabilities of binary outcomes. This difference is more or less analogous to the progression from logistic to softmax regression in the simplest linear models of classification.

It is obvious that the model in Figure 1 is far more primitive than the multilayer capsule networks that have been explored for difficult problems in visual object recognition (Sabour et al., 2017; Hinton et al., 2018; Ahmed and Torresani, 2019; Hahn et al., 2019; Jeong et al., 2019; Venkataraman et al., 2020; Tsai et al., 2020). Nevertheless, this model does provide what is arguably the simplest expression of the *raison d’etre* for capsules—namely, the idea that the length of a vector of hidden unit activities can encode the probability that some pattern is present in the input. The model can also be viewed as a higher-dimensional analogue of logistic/probit regression (for $m = 2$ classes) or softmax regression (for $m \geq 3$ classes) in which each class is modeled by a vector of hidden unit activities as opposed to a single scalar dot product.

In developing the model further, it becomes needlessly cumbersome to list the $m$ latent variables $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_m$ wherever they are collectively employed. In what follows, therefore, we denote the collection of these $m$ latent variables by $\mathbf{h} = (\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_m)$. In this way, we may simply write the factorization in eq. (1) as $P(\mathbf{h}|\mathbf{x}) = \prod_i P(\mathbf{h}_i|\mathbf{x})$ and the squashing nonlinearity in eq. (3) as $P(y = j|\mathbf{h}) = \|\mathbf{h}_j\|^2/\|\mathbf{h}\|^2$. As a similar shorthand, we also denote the collection of $m$ weight matrices in the model by
\[ W = (W_1, W_2, \ldots, W_m). \]

Finally, we describe how the model predicts class labels from inputs. We obtain the conditional probability \( P(y = j|x) \) by marginalizing the model’s latent variables

\[
P(y = j|x) = \int_{h \in \mathbb{R}^{dm}} P(y = j|h) P(h|x).
\] (4)

Note that the multidimensional integral in eq. (4) incorporates the squashing nonlinearity in \( P(y = j|h) \) through eq. (3); in particular, eq. (4) is not a purely Gaussian integral. Nevertheless, we show in section 2.2 that this integral can be evaluated analytically. We shall see, in fact, that the result is also simple enough to permit the further calculations required for inference.

We leave these more detailed results for later. For now, though, it behooves us to examine how the integral in eq. (4) simplifies in the opposing limits that the model’s variance \( \sigma^2 \) either vanishes or diverges. In these limits, we have respectively that

\[
P(y = j|x) \to \begin{cases} \frac{||W_j||^2}{\sum_i ||W_i||^2} & \text{as } \sigma^2 \to 0, \\ \frac{1}{m} & \text{as } \sigma^2 \to \infty. \end{cases}
\] (5)

These limits are simple to understand: as \( \sigma^2 \to 0 \), the model’s latent variables are effectively determined by the inputs, which renders the integral in eq. (4) trivial, whereas as \( \sigma^2 \to \infty \), the model becomes maximally uncertain, predicting each class with equal probability regardless of the input. Once again, we note that while these limits are trivial, we shall see that many non-trivial aspects of the model can be exactly calculated by a simple interpolation between these two regimes.

Having demonstrated how the model makes predictions, we turn briefly to the problem of learning—that is, of estimating parameters \( W \) and \( \sigma \) that yield a useful model.
We note that in the course of learning, the predictions of our model typically pass from a regime of high uncertainty (i.e., larger $\sigma^2$) to low uncertainty (i.e., smaller $\sigma^2$), and therefore it is exactly the intermediate regime between the two limits in eq. (5) where we expect the bulk of learning to occur.

We consider the problem of supervised learning, where the goal is to estimate the model parameters $W$ and $\sigma^2$ from a data set of $n$ labeled examples $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. In our model, the simplest form of learning is to maximize the log-conditional likelihood given by

$$L(W, \sigma^2) = \sum_{\ell=1}^{n} \log P(y_{\ell}|x_{\ell}).$$

(6)

As mentioned in the introduction, the objective here is the same as maximum likelihood (ML) estimation in more traditional models of logistic, probit, or softmax regression. However, our capsule architecture replaces the scalar activities of these more traditional models, derived from dot products, with the richer distributed representations derived from matrix-vector multiplications, and it replaces the sigmoidal and softmax nonlinearities of these models with the squashing nonlinearity in eq. (3).

To maximize the likelihood in eq. (6), we can avail ourselves of the Expectation-Maximization procedure (Dempster et al., 1977) for latent variable models. As is usual, the EM algorithm for our model alternates between two steps—an E-step which computes posterior statistics of the model’s latent variables, and an M-step which uses these statistics to re-estimate the model’s parameters and increase the log-conditional likelihood in eq. (6). For the $\ell$th labeled example, the posterior distribution over the latent
variables is given by Bayes rule:

\[
P(h|y_\ell, x_\ell) = \frac{P(y_\ell|h) P(h|x_\ell)}{P(y_\ell|x_\ell)}. \tag{7}
\]

For our purposes, the most important statistics of this distribution are the posterior means \(E[h_i|y_\ell, x_\ell]\) for each latent variable \(h_i\). The E-step of the EM algorithm computes these expected values by averaging over the posterior distribution:

\[
E[h_i|y_\ell, x_\ell] = \int_{h \in \mathbb{R}^d} P(h|y_\ell, x_\ell) h_i. \tag{8}
\]

An important feature of our model is that the multidimensional integral in eq. (8) can also be calculated in closed form. As a result, we can implement the EM algorithm without recourse to approximations such as variational inference (Wainwright and Jordan, 2008) or stochastic simulation (Neal, 1993). Finally, the M-step of the EM algorithm takes a particularly simple form in our model: the weight matrix \(W_i\) is re-estimated by solving the least-squares problem

\[
W_i \leftarrow \arg\min_{\Lambda \in \mathbb{R}^{d \times p}} \left\{ \sum_{\ell=1}^{n} \left\| E[h_i|y_\ell, x_\ell] - \Lambda x_\ell \right\|^2 \right\} \tag{9}
\]

where the posterior means \(E[h_i|y_\ell, x_\ell]\) that appear in eq. (9) are precisely those computed by the E-step (in terms of the current weight matrices).

In sum, our model is defined by the multivariate Gaussian distributions in eq. (1) and the squashing nonlinearity in eq. (3), and its essential parameters (namely, the weight matrices \(W_i\)) are re-estimated by computing the posterior statistics in eq. (8) and solving the least-squares problems in eq. (9). The next sections provide the results that are needed to implement these steps in practice.
2.2 Inference

To perform inference in our model, we must integrate over the model’s latent variables. For example, to predict class labels, we must compute the conditional probability $P(y = j | x)$. Substituting eqs. (1) and (3) into eq. (4), we see that

$$P(y = j | x) = \int_{h \in \mathbb{R}^d} \frac{||h_j||^2}{||h||^2} \prod_i \frac{e^{-\frac{1}{2\sigma^2} ||h_i - W_i x||^2}}{(2\pi\sigma^2)^{d/2}}$$  \hspace{1cm} (10)

Similarly, to carry out the EM algorithm, we must compute the posterior means $E[h_i | x, y = j]$. Substituting eq. (7) into eq. (8), we see that

$$E[h_i | x, y = j] = \frac{1}{P(y = j | x)} \int_{h \in \mathbb{R}^d} h_i \frac{||h_j||^2}{||h||^2} \prod_i \frac{e^{-\frac{1}{2\sigma^2} ||h_i - W_i x||^2}}{(2\pi\sigma^2)^{d/2}}$$  \hspace{1cm} (11)

We show how to evaluate these multidimensional integrals exactly in section A.2. In this section, though, we mainly present the results of these calculations, focusing on what is required to implement the model in practice.

As discussed in the previous section, the integrals in eqs. (10–11) behave simply in the limits $\sigma^2 \to 0$ and $\sigma^2 \to \infty$. It is more convenient, however, to express their overall behavior in terms of the dimensionless ratio

$$\beta = \frac{1}{2\sigma^2} \sum_i ||W_i x||^2,$$  \hspace{1cm} (12)

where $\beta \to 0$ and $\beta \to \infty$ correspond, respectively, to the limits of complete and zero uncertainty. Note that $\beta$ is defined with respect to a particular input $x$, and that it also depends on the model parameters $W$ and $\sigma$. As shorthand, though, we will not denote this dependence explicitly.

To use our model, we must understand how it behaves for intermediate values of $\beta$. 

11
To this end, we define the interpolating functions $\lambda_0(\beta)$ and $\lambda_1(\beta)$ given by

$$
\lambda_0(\beta) = \beta e^{-\beta} \int_0^1 d\rho \frac{\rho^{d\alpha}}{\pi} e^{\rho \beta}, \quad (13)
$$

$$
\lambda_1(\beta) = \beta e^{-\beta} \int_0^1 d\rho \frac{\rho^{d\alpha + 1}}{\pi} e^{\rho \beta}. \quad (14)
$$

The one-dimensional integrals in these definitions can be evaluated analytically, but the most important properties of $\lambda_0(\beta)$ and $\lambda_1(\beta)$ are in fact more easily derived from these integral representations. We show how to evaluate these integrals efficiently in section A.1. From these definitions we also prove that $\lambda_0(\beta)$ and $\lambda_1(\beta)$ are monotonically increasing functions with

$$
\lambda_0(\beta), \lambda_1(\beta) \rightarrow \begin{cases} 
0 & \text{as } \beta \rightarrow 0, \\
1 & \text{as } \beta \rightarrow \infty.
\end{cases} \quad (15)
$$

Thus these interpolating functions can be viewed as providing another continuous measure of the model’s uncertainty, but unlike the dimensionless ratio $\beta$ (or the variance $\sigma^2$), they are bounded between 0 and 1.

The most important inferences in our model take an especially compact form in terms of these interpolating functions. For example, as shown in section A.2, the conditional probability $P(y = j|x)$ in eq. (10) is given exactly by

$$
P(y = j|x) = \lambda_0(\beta) \cdot \frac{||W_jx||^2}{\sum_i ||W_ix||^2} + (1-\lambda_0(\beta)) \cdot \frac{1}{m}. \quad (16)
$$

Note in this expression that $\lambda_0(\beta)$ appears simply as a coefficient that mixes the model’s predictions in the limits of zero and complete uncertainty, as given earlier by eq. (5). It follows from eq. (16) that $\arg \max_j P(y = j|x) = \arg \max_j ||W_jx||^2$, so that
the model can be used as a classifier—that is, to predict the most likely label of an input $x$—without evaluating $P(y = j | x)$ in full. From eq. (16), it is also trivial to verify that this posterior distribution is properly normalized with $\sum_j P(y = j | x) = 1$.

Next we consider the posterior mean $E[h_i | x, y = j]$, given by eq. (11). Recall that each posterior mean $E[h_i | x, y = j]$ is a $d$-dimensional vector (where $d$ is the capsule dimensionality) that lives in the same vector space as its corresponding prior mean $E[h_i | x]$. In addition, it follows from the model’s symmetries that the vector $E[h_i | x, y = j]$ must be parallel to the vector $E[h_i | x]$, differing in magnitude but not direction. Thus we may write

$$E[h_i | x, y = j] \propto E[h_i | x],$$  

(17)

where the constant of proportionality depends in general on the input $x$, the class label $j$, the model parameters $W$ and $\sigma^2$, and the index $i$ of the capsule that hosts the latent variable $h_i$. From the squashing nonlinearity in eq. (3), we might intuitively expect the posterior mean $E[h_i | x, y = j]$ to exceed the prior mean $E[h_i | x]$ in magnitude when $i = j$ and vice versa when $i \neq j$. This intuition is borne out by the exact calculation in section A.2, to which we refer the reader for further details.

We now complete the result implied by eq. (17). Like the conditional probability $P(y = j | x)$ in eq. (16), the posterior mean $E[h_i | x, y = j]$ can also be expressed in a highly intelligible form. To this end, we define a new family of distributions, $Q_i(y = j | x)$, given by

$$Q_i(y = j | x) = \lambda_1(\beta) \cdot \frac{\|W_j x\|^2}{\sum_k \|W_k x\|^2} + (1 - \lambda_1(\beta)) \cdot \frac{2\delta_{ij} + d}{2 + dm},$$  

(18)

where $\lambda_1(\beta)$ is the interpolating function defined in eq. (14) and $\delta_{ij}$ denotes the Kro-
necker delta function. It is easy to verify that the right hand side of eq. (18) defines a distribution, with \( \sum_j Q_i(y = j|x) = 1 \) for each value of \( i \). In terms of these distributions, the posterior means are given by

\[
E[h_i|y = j, x] = \frac{Q_i(y = j|x)}{P(y = j|x)} \cdot E[h_i|x], \tag{19}
\]

so that we can identity the ratio of the distributions \( Q_i(y = j|x) \) and \( P(y = j|x) \) with the implied constant of proportionality in eq. (17). Note that just as the conditional distribution \( P(y = j|x) \) must be properly normalized, satisfying \( \sum_j P(y = j|x) = 1 \), it must also be true that the prior and posterior means are internally consistent, satisfying \( \sum_j P(y = j|x)E[h_i|y = j, x] = E[h_i|x] \). From eq. (19), it is easy to verify that this identity holds.

In sum, we compute the conditional probability \( P(y = j|x) \) from eq. (16), and we compute the posterior means \( E[h_i|x, y = j] \) from eq. (19). We need both these results for the EM algorithm—the first to verify that the likelihood of the data in eq. (6) is increasing, and the second to facilitate the update in eq. (9). In the next section, we examine this update in more detail.

### 2.3 Learning

We can use the EM procedure to derive updates that re-estimate the parameters \((W, \sigma^2)\) of our model. We begin by observing that the log-conditional likelihood \( L(W, \sigma^2) \) in eq. (6) has a symmetry: it is invariant under the rescaling transformation

\[
W \rightarrow \frac{W}{\sqrt{\sigma^2}}, \tag{20}
\]

\[
\sigma^2 \rightarrow 1. \tag{21}
\]
This property follows at once from our expression for the conditional probability $P(y|x)$ in eq. (16) and the fact that the dimensionless ratio $\beta = \frac{1}{2\sigma^2} \sum_i \|W_i x\|^2$ is also invariant under this transformation. Thus for any model with $\sigma^2 \neq 1$, we can obtain an equivalent model with $\sigma^2 = 1$ by rescaling the weights in this way. (The transformed model is equivalent in the sense that it predicts labels from inputs with the same probabilities.) Without loss of generality, we can therefore fix $\sigma^2 = 1$ at the outset of learning and only use the EM algorithm to re-estimate the model’s weight matrices. In the previous sections of this paper, we have used the parameter $\sigma^2$ to formulate our model and develop an intuition for how it behaves. At this point, however, we set $\sigma^2 = 1$ and do not bother\(^1\) to re-estimate it.

We observed earlier that the update for the weight matrix $W_i$ takes the form of the least-squares problem in eq. (9). As noted by Dempster et al. (1977), this result is typical of models with Gaussian latent variables, such as factor analysis (Rubin and Thayer, 1982; Ghahramani and Hinton, 1996) and probit regression (Liu et al., 1998a). In these models the EM procedure also yields iterative least-squares updates for ML estimation, and the update in eq. (9) is derived in exactly the same way.

We gain some further insights by writing out the update in eq. (9) explicitly. Solving the least-squares problem, we see that

$$W_i \leftarrow \left( \sum_{\ell=1}^{n} E[h_i|x_\ell, y_\ell] x_\ell^T \right) \left( \sum_{\ell=1}^{n} x_\ell x_\ell^T \right)^{-1},$$

(22)

where as usual the posterior means $E[h_i|x_\ell, y_\ell]$ on the right hand side are computed from the current model parameters. Note that the matrix inverse in eq. (22) can be pre-

\(^1\)This point is more subtle than we have indicated: it has been shown that the EM algorithm can be accelerated in Gaussian latent variable models by representing the variance explicitly (Liu et al., 1998a). However, we do not pursue that path here.
computed from the data at the outset of learning. This step is extremely useful with inputs of moderate to high dimensionality.

With some further work we can obtain an even more revealing expression for the update in eq. (22). We begin by introducing another shorthand: for each input $x_\ell$ in the training set, we define the conjugate input by

$$v_\ell = \left( \sum_{\ell=1}^{n} x_\ell x_\ell^\top \right)^{-1} x_\ell.$$

(23)

The vectors $x_\ell$ and $v_\ell$ are conjugate in the sense that by construction, the sum $\sum_{\ell=1}^{n} x_\ell v_\ell^\top$ is equal to the $p \times p$ identity matrix, where $p$ is the dimensionality of the input space. Combining this shorthand with the result in eq. (19), we can rewrite the update in eq. (22) as

$$W_i \leftarrow W_i \left( \sum_{\ell=1}^{n} \frac{Q_i(y_\ell|x_\ell)}{P(y_\ell|x_\ell)} x_\ell v_\ell^\top \right).$$

(24)

The term inside the parentheses of this update is a $p \times p$ matrix. Thus from eq. (24), we can also see that the EM algorithm is performing a multiplicative update on each capsule’s weight matrix.

The update in eq. (24) is guaranteed to increase the log-conditional likelihood in eq. (6) except at stationary points. In practice, however, we have found it useful to modify this update in two ways. These modifications do not strictly preserve the EM algorithm’s guarantee of monotonic convergence in the likelihood, but they yield other practical benefits without seeming to compromise the algorithm’s stability. We discuss these next.

First, we modify the update in eq. (24) to learn models that yield more accurate
classifiers. We have noticed that the log-conditional likelihood in eq. (6) does not always track the model’s accuracy as a classifier in the later stages of learning. To counter this problem, we can modify the update in eq. (24) to focus more on the examples that the model predicts least accurately. Such examples can be identified by computing the ratio

$$r_\ell = \frac{\max_{y \neq y_\ell} P(y|x_\ell)}{P(y_\ell|x_\ell)}.$$  

This ratio is greater than one for inputs that are misclassified and less than one for inputs that are classified correctly. In addition, the ratio is close to zero for inputs that are classified correctly with high certainty. To focus on inputs that are misclassified, we choose a threshold $\nu \in [0, 1]$ and identify the inputs with $r_\ell \leq \nu$; next, for this subset of correctly classified inputs, we replace the posterior means $E[h_i|x_\ell, y_\ell]$ in the update of eq. (24) by the prior means $E[h_i|x_\ell]$. This modification leads to the thresholded update:

$$W_i \leftarrow W_i \left( \sum_{r_\ell \leq \nu} x_\ell v_\ell^\top + \sum_{r_\ell > \nu} \frac{Q_i(y_\ell|x_\ell)}{P(y_\ell|x_\ell)} x_\ell v_\ell^\top \right)$$  

By setting $\nu = 0$, we recover the previous update in eq. (24). For positive values of $\nu$, however, the thresholded update focuses the model on those examples whose predictions are least accurate and/or most uncertain.

We emphasize that the correctly classified examples with $r_\ell \leq \nu$ are not dropped from the update altogether; they still appear in the first sum on the right hand side of eq. (22). The goal is not to ignore these examples, but only to reduce their influence on the model after they are correctly classified with high certainty. The thresholded update can be viewed as a heuristic for large-margin classification (Boser et al., 1992), focusing on incorrect examples and/or correct examples near the decision boundary. Though motivated differently, it also resembles certain incremental variants of the EM
algorithm that have been explored for latent variable modeling (Neal and Hinton, 1998).

In addition to eq. (26), we explore one other modification whose goal is to accelerate the algorithm’s rate of convergence. There is a large literature on methods for accelerating EM algorithms while preserving their guarantees of monotonic convergence in the likelihood (Jamshidian and Jennrich, 1993; Liu and Rubin, 1994; Lange, 1995; Jamshidian and Jennrich, 1997; Liu et al., 1998a,b; Salakhutdinov et al., 2003; Varadhan and Roland, 2008; Yu, 2012). For this work, we have adopted the simple approach of incorporating a momentum term (Qian, 1999) with an additional hyper-parameter $\gamma > 0$. Such terms have a long and successful history with gradient-based methods (Rumelhart et al., 1986). Our approach does not preserve the theoretical guarantee of convergence, but in practice we have observed that it behaves remarkably well. With this further modification, the update takes the form

$$W^{(t+1)} = EM_{\nu}(W^{(t)}) + \gamma \left(W^{(t)} - W^{(t-1)}\right),$$

where $EM_{\nu}(W)$ refers to the thresholded update in eq. (26) and $W^{(t)}$ refers to the weight matrices estimated after $t$ iterations of this procedure. We obtained our main results in section 3 with this modified update, but we compare the behavior of eq. (24) versus eqs. (26–27) further in appendix B.

### 2.4 Initialization

EM algorithms do not converge in general to a global maximum of the likelihood, and as a result, the models they discover can depend on how the model parameters are initialized. This is true for the update in eq. (24) as well as the modified updates in eqs. (26–27).
After fixing $\sigma^2 = 1$, we compared two approaches for initializing the model’s weight matrices $W_1, W_2, \ldots, W_m$. In the first approach, we randomly sampled each matrix element from a Gaussian distribution with zero mean and small variance. In the second approach, we initialized the matrices based on a singular value decomposition of the training examples in each class. We refer to the first approach as random initialization and the second approach as subspace initialization.

Roughly speaking, the goal of subspace initialization is to maximize $\|W_i x\|$ for inputs $x$ that belong to the $i^{th}$ class. In particular, let $n_i < n$ denote the number of training examples with label $i$, and let $X_i$ denote the $p \times n_i$ matrix of inputs with this label. We assume that for each class label, we have at least as many examples as the capsule dimensionality: i.e., $n_i \geq d$. The subspace initialization attempts to seed the matrix $W_i$ with an informative projection of these $n_i$ inputs; intuitively, such a projection seems more likely to encode the variability of patterns recognized by the $i^{th}$ capsule. To this end, we compute the leading $d$ eigenvectors $\{v_{i\alpha}\}_{\alpha=1}^d$ and eigenvalues $\{\xi_{i\alpha}\}_{\alpha=1}^d$ of the $p \times p$ matrix $n_i^{-1}X_iX_i^\top$. Then we initialize the $i^{th}$ capsule’s weight matrix as

$$W_i = \frac{1}{\sqrt{d}} \begin{pmatrix} \frac{1}{\sqrt{\xi_{i1}}}v_{i1} \\ \frac{1}{\sqrt{\xi_{i2}}}v_{i2} \\ \vdots \\ \frac{1}{\sqrt{\xi_{id}}}v_{id} \end{pmatrix}. \quad (28)$$

The factors of $\frac{1}{\sqrt{d}}$ and $\frac{1}{\sqrt{\xi_{i\alpha}}}$ in this initialization attempt to control for the dimensionality of the latent space and the dynamic range of the inputs. We obtained our main results in section 3 with the initialization in eq. (28), but we compare the effects of random versus subspace initializations further in appendix B.
3 Experiments

We implemented the model described in the previous section and evaluated its performance in different architectures for multinomial classification. In some especially simple settings, we also sought to understand the latent representations learned by capsule regression. The organization of this section is as follows. In section 3.1, we describe the data sets that we used for benchmarking and the common setup for all of our experiments. In section 3.2, we present a visualization of results from the model in Figure 1 with two-dimensional capsules ($d=2$). This visualization reveals how the latent spaces of different capsules are organized in tandem by the updates of section 2.3 to learn an accurate classifier. In section 3.3, we examine the internal representations learned by the model in Figure 1 with eight-dimensional capsules ($d=8$). Here, we explore how distinct patterns of variability are encoded by different elements of the model’s latent variables. Finally, in section 3.4, we present our main results—a systematic comparison of classifiers obtained from capsules of varying dimensionality as well as different capsule-based architectures (e.g., multiclass, one versus all, all versus all) for multinomial classification.

3.1 Setup

We experimented on two data sets of images, one of handwritten digits (LeCun et al., 1998), and the other of fashion items (Xiao et al., 2017). Both these data sets contain 60K training examples and 10K test examples of 28x28 grayscale images drawn from $m = 10$ classes; they have also been extensively benchmarked. To speed up our experiments, we began by reducing the dimensionality of these images by a factor of four. Specifically, for each data set, we used a singular value decomposition of the
training examples to identify the linear projections with largest variance, and then we used these projections to map each image into an input $x \in \mathbb{R}^p$ for capsule regression, where $p = 196$. This was done for all the experiments in this paper.

We followed a single common protocol for training, validation, and testing (with one exception, mentioned at the end of the section, where we trained on a much larger data set of 1M digit images). We trained our models on the first 50K examples in each training set while holding out the last 10K examples as a validation set. We monitored the classification error rate on the validation set in an attempt to prevent overfitting. We used the subspace initialization in eq. (28) to seed the weight matrices before training, and we used a fixed momentum hyperparameter of $\gamma = 0.9$ in the update of eq. (27). We did not use a fixed value for the thresholding hyperparameter $\nu$. Instead, we divided the learning into five rounds with $\nu$ taking on a fixed value of 0.8 in the first round, 0.6 in the second, 0.4 in the third, 0.2 in the fourth, and 0 in the fifth. We terminated the first round when the error rate on the validation set had not improved for 128 consecutive iterations, and we terminated the next rounds when it had not improved for 64, 32, 16, and 8 consecutive iterations, respectively. Thus each round had a fixed minimum number of iterations, though not a fixed maximum. Finally, we initialized each subsequent round by the best model obtained in the previous one (as measured by the error rate on the validation set). We present some empirical motivation for these choices of hyperparameters in appendix B.

### 3.2 Visualization of results with $d=2$ capsules

We begin by examining one of the simplest models of capsule regression on the MNIST images of handwritten digits. Figures 2 visualizes the latent representations learned by each capsule in Figure 1 where $h_i \in \mathbb{R}^2$. (This model for capsule regression has a
fairly high test error rate of 6.21% on the digit images, but it is also by far the simplest to visualize.) As before, let $E[h_i|x] = W_i x$ denote the expected (vector) value of the latent variable in the $i$th capsule given the input $x$. The $i$th panel in the figure shows a scatterplot of the squashed two-dimensional vectors

$$\Psi_i(x) = \frac{E[h_i|x]}{\sqrt{\sum_{j=1}^{m} ||E[h_j|x]||^2}}$$

(29)

where $x$ is an input from the test set of 10K images. Note that by construction we have that $||\Psi_i(x)||^2 = P(y=i|h = E[h|x])$. It follows that (i) each such vector $\Psi_i(x)$ lies within the unit circle, and (ii) the larger its magnitude, the larger the probability $P(y=i|x)$ that the model assigns to the input $x$. Each vector in these plots is also color-coded by its class label as displayed in the panel titles. Thus, for example, in the upper leftmost panel, we see that test images of zeros (color-coded as black) are represented by vectors that lie near the unit circle and therefore receive large probabilities for being labeled as zeros. In the other panels, however, these same images (again, color-coded as black) are represented by vectors that lie closer to the origin and therefore receive small probabilities for being labeled as anything else. In fact, this pattern repeats itself for all the classes of digits: in each panel, the vectors that lie nearest the unit circle are those that share the same color as the class label in the panel’s title. (At the top right of each panel, we display the most confidently recognized image of each class.)

Figures 3 shows the corresponding result for the same model trained on images of fashion items. The test error rate (15.14%) is higher for this data set, but the same pattern is evident. From the plots in these panels, we can also see which classes of images are most confusable. For example, the black latent variables (representing images of t-shirts) have large radii not only in the upper leftmost panel, but also in the bottom panel.
second from the left. These two panels—corresponding to the capsules for t-shirts and shirts—show that these two classes of images are among the likeliest to be confused.

Naturally it is more difficult to visualize the results from models of capsule regression with higher dimensional \(d > 2\) latent variables. But conceptually it is clear what happens: the circles of unit radius in the panels of Figures 2 and 3 are replaced by hyperspheres of unit radius in \(d\) dimensions. With latent variables of higher dimensionality, we might also expect the capsules to discover richer internal representations of the variability within each class of images. This is what we explore in the next section.

### 3.3 Encoding of variability by \(d = 8\) capsules

The latent variable \(h_i\) in the model of Figure 1 provides an internal representation of the variability of patterns in the \(i^{th}\) class. Though difficult to visualize these patterns in higher-dimensional capsules, we can identify—for instance—the training example in the \(i^{th}\) class for which the \(\alpha^{th}\) element of \(E[h_i|x]\) is largest in magnitude relative to the
other elements of $E[h|x]$. Specifically, for each class $i \in \{1, 2, \ldots, m\}$ and element $\alpha \in \{1, 2, \ldots, d\}$, we can identify the example

$$x_{i\alpha}^* = \arg\max_{y_i = i} \left( \frac{E[h_{i\alpha}|x]}{\|E[h|x]\|} \right)^2.$$ (30)

whose probability $P(y = i|x)$ receives its largest contribution from the $\alpha^{th}$ element of latent variable $h_i \in \mathbb{R}^d$. Such an example can be viewed as a prototype for the pattern of variability encoded by the $\alpha^{th}$ latent variable of the $i^{th}$ capsule.

Figure 4 shows these prototypical examples for the model of Fig. 1 with capsules of dimensionality $d = 8$. For the images of digits, the prototypes exhibit variations in orientation, thickness, and style (e.g., the presence of a loop in the digit TWO, or an extra horizontal bar in the digit SEVEN). For the images of fashion items, the prototypes exhibit variations in brightness, girth, and basic design. The examples in the figure are suggestive of the prototypes discovered by vector quantization (Lloyd, 1957), but in this case they have emerged from internal representations of discriminatively trained
Figure 4: Prototypical training examples from different classes of digits (left) and fashion items (right). For each array, the image in the $i^{th}$ row and $\alpha^{th}$ column shows the training example from the $i^{th}$ class whose probability $P(y = i|\mathbf{x})$ receives its largest contribution from the $\alpha^{th}$ element of the $i^{th}$ capsule’s latent variable $\mathbf{h}_i \in \mathbb{R}^d$. The results are shown for the model in Fig. 1 with capsules of dimensionality $d = 8$.

capsules. It is clear that higher-dimensional capsules can represent a greater diversity of prototypes, and by doing so, they might be expected to yield more accurate classifiers. This is what we explore in the next section.

3.4 Results for classification

Our main experiments investigated the effect of the capsule dimensionality ($d$) on the model’s performance as a classifier. We experimented with three different types of architectures: (i) the multiclass-capsule architecture shown in Figure 1, in which we jointly train $m$ capsules to recognize patterns from $m$ different classes, (ii) a one-versus-all architecture, in which we train $m$ binary-capsule architectures in parallel and label inputs based on the most certain of their individual predictions, and (iii) an all-versus-all
Figure 5: Effect of capsule dimensionality on test error rates for images of handwritten digits (left) and fashion items (right) from multiclass, one-versus-all, and all-versus-all classifiers.

architecture, in which we train $m(m-1)/2$ binary-capsule architectures in parallel and label inputs based on a majority vote. We note that these architectures employ different numbers of capsules, and therefore they have different numbers of weight matrices and learnable parameters even when all their capsules have the same dimensionality. In particular, the one-versus-all architecture has twice as many learnable parameters as the multiclass architecture, while the all-versus-all architecture has $m-1$ times as many learnable parameters.

Figure 5 shows the results of these experiments. The red, blue, and black curves show, respectively, the test error rates of the multiclass, all-versus-one, and all-versus-all classifiers as a function of their capsule dimensionalities. For all these types of classifiers, and for both data sets of images, the plots show that capsules with higher dimensional latent variables are able to learn more accurate classifiers. For capsules
of a fixed dimensionality, the plots also show that the architectures with more capsules (and hence more learnable parameters) are able to learn more accurate classifiers.

The data sets of MNIST handwritten digits and fashion items have been extensively benchmarked, so we can also compare these results to those of other classifiers (LeCun et al., 1998; Xiao et al., 2017). The one-dimensional \(d = 1\) capsule architectures in Figure 5 have test error rates comparable to those of other simple linear models; for example, Xiao et al. (2017) report test error rates of 8.3% and 15.8% for one-versus-all logistic regression on the data sets of digits and fashion items, respectively. Likewise, the models with higher-dimensional capsules offer similar improvements as other nonlinear approaches, such as \(k\)-nearest neighbors, random forests (Breiman, 2001), and fully connected neural nets; standardized implementations of these algorithms in Python’s scikit-learn yield test error rates of roughly 2-3% on digits and 10-12% on fashion items (Xiao et al., 2017). However, none of the models in Figure 5 classify as well as the best-performing models, such as nonlinear support vector machines (Cortes and Vapnik, 1995) or convolutional neural nets LeCun et al. (1998), with test error rates nearer to 1%. We believe that this gap in performance is mainly due to overfitting as opposed to an inability to learn sufficiently complex decision boundaries.

To test this hypothesis, we conducted another set of experiments where we trained the all-versus-all digit classifiers in Figure 5 on a much larger data set of 1M training images (Loosli et al., 2007). The 950K additional images for training were obtained from distortions (e.g., rotation, thickening, etc) of the original MNIST training set. To facilitate a direct comparison with our previous results, we also used the same validation and test sets of 10K images. The results of these experiments are shown in the bottom (green) curve of the left panel of Figure 5. The results show that these all-versus-all capsule architectures have the capacity to learn better classifiers from larger amounts
of training data. But even these classifiers are also plagued by overfitting: for example, the best of these classifiers (with capsule dimensionality $d = 16$) still exhibits a large gap between its test error rate (1.38%) on 10K images and its training error rate (0.0523%) on 1M images.

4 Discussion

In this paper we have introduced capsule regression as a higher-dimensional analogue of simpler log-linear models such as logistic and softmax regression. We experimented with capsule regression in multiclass, one-versus-all, and all-versus-all architectures, and we showed that in all these architectures the model capacity grows in step with the capsule dimensionality. To learn these classifiers, we formulated capsule regression as a latent variable model, and we used the EM procedure to derive iterative least-squares updates for parameter estimation. Despite the squashing nonlinearity in our models, we showed that it remains tractable to perform exact inference over their continuous latent variables. One contribution of our work is to expand the family of tractable latent variable models that can learn meaningful distributed representations of high-dimensional inputs. Our work fits into a larger vision for probabilistic modeling: the “need to develop computationally-tractable representations of uncertainty” has been described as “one of the major open problems in classical AI” (Jordan, 2018).

Another contribution of our work is to highlight certain advantages of capsule regression for supervised learning with distributed representations. Traditional neural nets can learn more flexible decision boundaries than linear models, but this extra capacity comes at a cost: they involve much more complicated optimizations, with learning rates that must be tuned for convergence, and their internal representations (though
effective for classification) can be fairly inscrutable. Models for capsule regression benefit equally from their distributed representations; as shown in Figure 5, with higher-dimensional capsules, these models acquire the capacity to learn increasingly accurate classifiers. But as shown in Figures 2–4, the internal representations of capsules also have a fairly interpretable structure, and as shown in section 2.3, these representations can be learned by simple least-squares updates. There are other plausible benefits to this structure that we have not yet explored. For example, in this paper, we only considered architectures in which all the capsules have the same dimensionality. But these dimensionalities could be varied for classes that exhibit more diversity in their inputs and/or have larger numbers of training examples. It is harder to see how a traditional neural net could be purposefully adapted to reflect these forms of prior knowledge.

There are several issues in our work that deserve further study. The first is regularization: as previously mentioned, the models in section 2 are prone to overfitting even with early stopping on a validation set. It seems worthwhile to explore $\ell_1$ and/or $\ell_2$ regularization of the model parameters, as is common for other types of regression, and also to consider those forms of regularization—such as weight-sharing (Nowlan and Hinton, 2004), drop-out (Srivastava et al., 2014), and reconstruction penalties (Sabour et al., 2017; Qin et al., 2020)—that have been widely used in deep learning. It should also help to incorporate some prior knowledge about images explicitly into the structure of the weight matrices, as is done in convolutional neural nets (LeCun et al., 1998).

Another issue is scaling: for very large data sets, it will be more practical to implement online or mini-batch versions of the updates in section 2.3. The least-squares form of eq. (9) suggests certain possibilities beyond stochastic gradient descent for this purpose. There are, for example, passive-aggressive online algorithms for regression (Crammer et al., 2006) that could be adapted to this setting, with the posterior
mean \( E[h_i|x, y] \) providing a target for the \( i^{th} \) capsule’s regression on the input \( x \) with label \( y \). These posterior means also provide the sufficient statistics for faster incremental variants of the EM algorithm (Neal and Hinton, 1998).

A final issue is model-building: is it possible to extend the ideas for capsule regression in this paper to deeper and more sophisticated architectures? We have already seen that the one-versus-all and all-versus-all architectures in section 3.4 lead to more accurate classifiers than the basic model of capsule regression in Figure 1. But even these architectures are still too primitive for modeling (say) the part-whole relationships of complex visual objects; those relationships may need to be modeled more explicitly, as in the multilayer capsule networks (Sabour et al., 2017) whose squashing nonlinearities were the motivation for our own study. For deeper architectures with such nonlinearities, we believe that the methods in this paper may serve as useful building blocks. We started this paper by noting the promising of existing capsule networks, and it seems fitting, then, that we have come full circle. We conclude on the hopeful note that this work provides yet another bridge between the traditions of latent variable modeling and deep learning.

A Supporting calculations

In this appendix we present the more detailed calculations for inference that were omitted from section 2.2. In particular, in section A.1, we show how to calculate the interpolating coefficients \( \lambda_0(\beta) \) and \( \lambda_1(\beta) \), and in section A.2, we show how to calculate the multidimensional integrals over the model’s latent variables required for inference and learning.
A.1 Computing the interpolating coefficients

It is mostly straightforward to implement the EM algorithm in section 2, but some extra steps are needed to compute the interpolating coefficients, $\lambda_0(\beta)$ and $\lambda_1(\beta)$, defined in eqs. (13) and (14). In this section we show how to evaluate the one-dimensional integrals that appear in these definitions. We also show that $\lambda_0(\beta)$ and $\lambda_1(\beta)$ are monotonically increasing functions with values in the unit interval [0, 1].

We begin by studying a family of integrals that include both eqs. (13) and (14) as a special case. In particular, for integer $s \geq 0$, let

$$I_s(\beta) = \beta e^{-\beta} \int_0^1 d\rho \rho^s e^{\rho \beta}$$  \hspace{1cm} (31)

With this definition, we see that $\lambda_0(\beta)$ in eq. (13) corresponds to $I_s(\beta)$ with the choice $s = \frac{dm}{2}$, whereas $\lambda_1(\beta)$ in eq. (14) corresponds to $I_s(\beta)$ with the choice $s = \frac{dm}{2} + 1$.

Note that by restricting $s$ to be integer-valued, we are assuming that either the capsule dimensionality $d$ or the number of labels $m$ is an even integer. In practice, this is not an especially restrictive assumption because the capsule dimensionality $d$ is a parameter of our own choosing.

Suppose, then, that $s$ in eq. (31) is integer-valued. In that case, we can obtain a closed form (of sorts) for this integral by the method of differentiating under the integral sign:

$$I_s(\beta) = \beta e^{-\beta} \left( \frac{\partial}{\partial \beta} \right)^s \int_0^1 d\rho \ e^{\rho \beta} = \beta e^{-\beta} \left( \frac{\partial}{\partial \beta} \right)^s \left( e^{\beta} - 1 \right) \left( \frac{e^{\beta} - 1}{\beta} \right).$$  \hspace{1cm} (32)

Thus we have $I_0(\beta) = 1 - e^{-\beta}$ for the base case $s = 0$ and $I_1(\beta) = 1 - \frac{1-e^{-\beta}}{\beta}$ for the case $s = 1$. Eq. (32) shows that the integral can be evaluated analytically, but this formula rapidly becomes unwieldy even for moderate values of $s$. Thus another approach is
required.

A better strategy is to integrate the right hand side of eq. (31) by parts. This yields the recursive formula

\[ I_s(\beta) = 1 - \frac{s}{\beta} I_{s-1}(\beta). \]  

(33)

In principle, this recursion can be used to compute \( I_s(\beta) \), starting from the base case at \( I_0(\beta) = 1 - e^{-\beta} \) and iterating eq. (33) for as many steps as needed. In practice, however, a problem arises for large values of \( s \) or small values of \( \beta \) (which typically occur in the early stages of learning). In this case, the recursion becomes numerically unstable; in particular, if early steps of the recursion have already introduced some numerical error, then the right hand side of eq. (33) will tend to amplify this error whenever \( \beta < s \). In fact, eq. (33) is a textbook example of numerical instability (Hill, 2016).

To compute \( I_s(\beta) \) when \( \beta \leq s \), we adopt an equally well-known solution to this problem. Our first step is to derive fairly tight bounds on \( I_s(\beta) \). It is clear from eq. (31) that \( I_s(\beta) \leq I_{s+1}(\beta) \) for all \( \beta \). We can obtain lower and upper bounds on \( I_s(\beta) \) by substituting this inequality into eq. (33). In this way, for \( s \geq 1 \), we find that

\[ \frac{\beta}{\beta + s + 1} \leq I_s(\beta) \leq \frac{\beta}{\beta + s}. \]  

(34)

As an aside, we note that these bounds imply \( 0 \leq I_s(\beta) < 1 \) for all values of \( \beta \). It follows at once, as claimed in section 2.2, that the interpolating coefficients \( \lambda_0(\beta) \) and \( \lambda_1(\beta) \) also lie in the unit interval.

The bounds in eq. (34) provide a starting point to compute \( I_s(\beta) \) when \( \beta \leq s \). To see this, we invert eq. (33) to obtain the backward recursion

\[ I_s(\beta) = \frac{\beta}{s + 1} \left(1 - I_{s+1}(\beta)\right). \]  

(35)
The backward recursion in eq. (35) is numerically stable in exactly the opposite regime as the forward recursion in eq. (33); we can therefore use it to compute \( I_s(\beta) \) when \( \beta \leq s \). We do this in two steps. First, we start by computing lower and upper bounds on \( I_{s+k}(\beta) \) for some sufficiently large value of \( k \). Then we use the backward recursion \( k \) times to transform our upper and lower bounds on \( I_{s+k}(\beta) \) into lower and upper bounds on \( I_s(\beta) \). Since by assumption \( \beta \leq s \), these bounds get tighter at every step of the recursion; it follows that by choosing \( k \) to be sufficiently large, we can compute \( I_s(\beta) \) to whatever numerical accuracy is desired. As a practical matter, in our experiments we set \( k = 64 \) and verified for each evaluation that \( I_s(\beta) \) was computed to within a factor of \( 1 \pm 10^{-4} \).

To summarize, then, we use the forward recursion in eq. (33) to compute \( I_s(\beta) \) when \( \beta > s \), and we use the backward recursion in eq. (35) to compute \( I_s(\beta) \) when \( \beta \leq s \). In practice, we only invoke these recursions to compute \( \lambda_0(\beta) \), setting \( s = \frac{dm}{2} \), because the forward recursion can then be used to compute \( \lambda_1(\beta) \) with \( s = \frac{dm}{2} + 1 \). Note that the value of \( s = \frac{dm}{2} \) is fixed in advance by the capsule architecture. Thus before the outset of learning, it may also be possible to compile lookup tables for \( I_s(\beta) \) and use interpolation schemes for even faster inference. However, we have not pursued that approach here.

We claimed in section 2.2 that the interpolating coefficients \( \lambda_0(\beta) \) and \( \lambda_1(\beta) \) were
monotonically increasing functions of $\beta$. To prove this, we note that

\begin{align*}
I'_s(\beta) &= \frac{d}{d\beta} \left[ \beta e^{-\beta} \left( \frac{\varphi}{\beta} \right)^s \left( \frac{e^\beta - 1}{\beta} \right) \right], \quad (36) \\
&= \left( \frac{1}{\beta} - 1 \right) I_s(\beta) + I_{s+1}(\beta), \quad (37) \\
&= 1 - \left( \frac{\beta + s}{\beta} \right) I_s(\beta) \quad (38) \\
&\geq 0, \quad (39)
\end{align*}

where in the third line we have used eq. (35) to express $I_{s+1}(\beta)$ in terms of $I_s(\beta)$, and in the last line we have appealed to the upper bound in eq. (34). Since $\lambda_0(\beta) = I_{dm_2}(\beta)$ and $\lambda_1(\beta) = I_{dm_{2+1}}(\beta)$, this proves the claim.

A.2 Integrating over the model’s latent variables

In this section we show how to calculate the conditional probability $P(y = j|x)$ in eq. (4) and the posterior mean $E[h | x, y = j]$ in eq. (8). Both these calculations involve multidimensional integrals over all of the model’s latent variables, which we denote collectively by $h \in \mathbb{R}^D$.

We begin by considering the simpler but closely related integral given by

\begin{equation}
J = \int_{h \in \mathbb{R}^D} \frac{1}{(2\pi\sigma^2)^{D/2}} e^{-\frac{||h-m||^2}{2\sigma^2}} \frac{\sigma^2}{||h||^2}, \quad (40)
\end{equation}

This integral is non-trivial due to the last term in the integrand, where the squared magnitude $||h||^2$ appears in the denominator. We include the factor of $\sigma^2$ in the numerator of this term so that $J$ is a dimensionless quantity. For now we also assume that $D > 2$, since for $D \leq 2$ the integral in eq. (40) is not well-defined. After evaluating this inte-
gral, we will see that the results for $P(y = j | x)$ and $E[h_i | x, y = j]$ follow from fairly mechanical calculations.

We can evaluate the integral in eq. (40) by a series of simple transformations. First we use the integral identity

$$
\frac{1}{\|h\|^2} = \int_0^\infty \frac{d\eta}{\eta^3} e^{-\frac{\|h\|^2}{2\eta^2}} \tag{41}
$$

to lift the denominator of $\|h\|^2$ in eq. (40) into the exponent. This identity introduces an auxiliary variable $\eta$ over which we must also integrate to obtain $J$. But now we can reverse the order of integration and perform the resulting (Gaussian) integral over $h$. In this way we find that

$$
J = \sigma^2 e^{-\frac{\|\mu\|^2}{2\sigma^2}} \int_0^\infty \frac{d\eta}{\eta^3} \left( \frac{\eta^2}{\sigma^2 + \eta^2} \right)^{\frac{D-3}{2}} e^{\frac{\|\mu\|^2\eta^2}{2(\sigma^2 + \eta^2)}} \tag{42}
$$

thus replacing the multidimensional integral in eq. (40) by a much simpler one-dimensional integral. Next we make the change of variables $\eta = \sigma \tan \theta$, which yields

$$
J = e^{-\frac{\|\mu\|^2}{2\sigma^2}} \int_0^{\pi/2} d\theta \cos \theta (\sin \theta)^{D-3} e^{\frac{\|\mu\|^2\sin^2 \theta}{2\sigma^2}} \tag{43}
$$

A more recognizable form emerges from one further change of variables $\rho = \sin^2 \theta$. In this way we find

$$
J = \frac{1}{2} \int_0^1 d\rho \rho^{D-2} e^{-\frac{(1-\rho)\|\mu\|^2}{2\sigma^2}} \tag{44}
$$

At this point, the reader may already glimpse the origin of the integrals in eq. (13) and eq. (14), in terms of which we have expressed our results for $P(y = j | x)$ and $E[h_i | x, y = j]$. 35
To evaluate the integrals in eq. (4) and eq. (8), we need one more intermediate result. Let \( h \sim \mathcal{N}(\mu, \sigma^2 I) \) be normally distributed with mean \( \mu \in \mathbb{R}^D \), and let

\[
p_\alpha = E\left[ \frac{h_\alpha^2}{\|h\|^2} \right]
\]

(45)
denote the expected value of the proportion of its squared magnitude from its \( \alpha \)th component. Note that \( \sum_\alpha p_\alpha = 1 \) so that we can view \( p_\alpha \) as a distribution. To compute \( p_\alpha \), we must evaluate the integral

\[
p_\alpha = \int_{h \in \mathbb{R}^D} \frac{1}{(2\pi\sigma^2)^{D/2}} e^{-\frac{|h-\mu|^2}{2\sigma^2}} \frac{h_\alpha^2}{\|h\|^2}.
\]

(46)
The integral in eq. (46) differs from the one in eq. (40) by the numerator of the right-most term in the integrand. But we can relate these two integrals by the method of differentiating under the integral sign:

\[
p_\alpha = \sigma^2 \frac{\partial^2 J}{\partial \mu_\alpha^2} + 2\mu_\alpha \frac{\partial J}{\partial \mu_\alpha} + \left( 1 + \frac{\mu_\alpha^2}{\sigma^2} \right) J.
\]

(47)
Using eq. (47), we can evaluate the integral for \( p_\alpha \) by the more mechanical process of differentiation. We proceed by substituting the result in eq. (44) into the right hand side of eq. (47). After simplifying, we find

\[
p_\alpha = \frac{1}{D} + \left( \frac{\mu_\alpha^2}{\|\mu\|^2} - \frac{1}{D} \right) \cdot \frac{\|\mu\|^2}{2\sigma^2} \cdot \int_0^1 d\rho \rho^{D-1} e^{-\frac{(1-\rho)|\mu|^2}{2\sigma^2}}.
\]

(48)
By summing both sides of eq. (48) over \( \alpha \), it is also easy to verify (as required) that \( \sum_\alpha p_\alpha = 1 \).

With the result in eq. (48), we can now derive the expression for \( P(y = j|x) \) in
eq. (16). Suppose that the dimensionality $D$ of the integral in eq. (46) is equal to the product $dm$, where $d$ is the capsule dimensionality and $m$ is the number of labels. Then we can write $h = (h_1, h_2, \ldots, h_m)$ where each $h_i \in \mathbb{R}^d$, and we can set $\mu = (\mu_1, \mu_2, \ldots, \mu_m)$ where $\mu_i = \mathbf{W}_i \mathbf{x}$. With this notation, we can rewrite the marginalization in eq. (4) as

$$P(y = j | \mathbf{x}) = \int_{h \in \mathbb{R}^d} \left[ \prod_i e^{-\frac{1}{2\sigma^2} \|h_i - \mu_i\|^2} \right] \frac{\|h_j\|^2}{\|h\|^2} \left( \frac{1}{(2\pi\sigma^2)^{d/2}} \right)$$  \hspace{4cm} (49)$$

Comparing eq. (46) and eq. (49), we see that the latter is obtained by summing over $d$ elements of the former—namely, the $d$ elements that select the vector $h_j \in \mathbb{R}^d$ from the larger vector $h \in \mathbb{R}^d$. Performing this sum, we recover the result in eq. (16).

Finally, we show how to derive the result in eq. (19) for the posterior means $E[h_i | \mathbf{x}, y = j]$. Again, using the shorthand notation $\mu_i = \mathbf{W}_i \mathbf{x}$, we can rewrite the calculations in eqs. (7–8) as

$$E[h_i | y = j, \mathbf{x}] = \frac{\int_{h \in \mathbb{R}^d} \left[ \prod_i e^{-\frac{1}{2\sigma^2} \|h_i - \mu_i\|^2} \right] \|h_j\|^2 \ h_i}{\int_{h \in \mathbb{R}^d} \left[ \prod_i e^{-\frac{1}{2\sigma^2} \|h_i - \mu_i\|^2} \right] \|h\|^2} \hspace{4cm} (50)$$

To evaluate this expectation, we observe that it can be related to the marginalization in eq. (49) by the method of differentiating under the integral sign. In particular, we have that

$$E[h_i | y = j, \mathbf{x}] = \mu_i + \sigma^2 \frac{\partial}{\partial \mu_i} \left[ \log P(y = j | \mathbf{x}) \right] \hspace{4cm} (51)$$

We proceed by substituting the result for $P(y = j | \mathbf{x})$ in eq. (16) into the right hand side of eq. (51). After some tedious but straightforward calculation, this yields the result for $E[h_i | \mathbf{x}, y = j]$ in eq. (19).

As noted earlier, the above derivations assume that $D > 2$, since the integral in
eq. (40) is divergent for $D = 2$. But the case $D = 2$ is also of interest: it arises for binary classification ($m = 2$) with scalar capsules ($d = 1$). In this case we can replace eq. (40) with the $\varepsilon$-regularized integral

$$J_\varepsilon = \int_{h \in \mathbb{R}^D} \frac{1}{(2\pi\sigma^2)^{D/2}} e^{-\frac{\|h - \mu\|^2}{2\sigma^2}} \frac{\sigma^2}{\|h\|^2 + \varepsilon^2}.$$  \hspace{1cm} (52)

We can then follow the same steps as before to obtain expressions that reduce to the conditional probability $P(y = j|x)$ and posterior mean $E[h_i|x, y = j]$ in the limit $\varepsilon \to 0$. These limits are well-defined even in the case $D = 2$, and in this case they also yield the results for $P(y = j|x)$ in eq. (16) and $E[h_i|x, y = j]$ in eq. (19).

### B Supporting experiments

We obtained the results in section 3 with the modified update in eq. (27) and the subspace initialization in eq. (28). Most of these results were devoted to comparing models that were identically trained but had different values of the capsule dimensionality, $d$. To make meaningful comparisons, though, it was first necessary to standardize how we initialized the models and which updates we used to train them. In this appendix, we describe some of the preliminary experiments that informed these choices. In particular, section B.1 explores the effect of different updates (with thresholding and/or momentum), and section B.2 explores the effect of random versus subspace initializations.
B.1 Effects of momentum and thresholding

First we consider the effect of the hyperparameters $\nu$ and $\gamma$ on the course of learning. We used these hyperparameters to modify the EM update in eq. (24), and these modifications led to the variants in eqs. (26) and (27). In this section we consider how these modifications affect the log-conditional likelihood $\mathcal{L}(W, \sigma^2)$ in eq. (6) and the number of misclassified examples. To understand these effects, we experimented on the model in Figure 1 with capsule dimensionality $d = 4$. We trained this model on both data sets in four different ways—using the EM update ($\gamma = \nu = 0$) in eq. (24), using the thresholded update ($\gamma = 0, \nu = 0.8$) in eq. (26), and using the momentum update ($\gamma = 0.9$) in eq. (27) both with thresholding ($\nu = 0.8$) and without thresholding ($\nu = 0$). For each experiment, we initialized the model’s weight matrices by eq. (28) and we re-estimated them for 1000 iterations on all 60K training examples.
Figure 7: Effects of momentum ($\gamma = 0.9$) and thresholding ($\nu = 0.8$) on convergence of the EM algorithm for the model in Figure 1 with capsule dimensionality $d = 4$.

The panels in Figure 6 plot the normalized log-loss, computed as $-\frac{1}{n \log m} \cdot \mathcal{L}(W, \sigma^2)$, versus the number of iterations of learning on images of handwritten digits and fashion items. In these plots, the blue curves indicate the updates with momentum, and the dashed curves indicate the updates with thresholding. Two trends are clear: first, the blue curves lie below the black curves, indicating that the log-loss decreases more rapidly when a momentum term is included in the update; second, the dashed curves lie above the solid curves, indicating that the log-loss converges to a higher (worse) value with the thresholded update. The latter result is to be expected, as the thresholded update in eq. (26) only attempts to reduce the log-loss on a subset of training examples. Though not reproduced here, we observed these trends consistently across models of many different cardinalities ($m$) and capsule dimensionalities ($d$).
The panels in Figure 7 plot the classification error rate on the training sets of digit and fashion images versus the number of iterations of learning. Two trends are clear in these plots as well. The first trend is the same as before: the blue curves lie below the black curves, indicating that the error rate decreases more rapidly when a momentum term is included in the update. But the second trend is different: in these plots, the dashed curves lie below the solid curves, with a significant gap emerging after fewer ten iterations and growing thereafter. In particular, on both data sets we see that the error rates converge to a significantly lower value with the thresholded update. In practice, the thresholded update appears to trade the worse likelihoods in Figure 6 for the lower error rates in Figure 7. Again, though not reproduced here, we observed these trends consistently across models of many different cardinalities \((m)\) and capsule dimensionalities \((d)\).

For the models in section 3, we were ultimately more interested in minimizing their error rates as classifiers than maximizing their log-conditional likelihoods. To be sure, the likelihood provides a useful surrogate for the error rate (as the former is differentiable in the model parameters whereas the latter is not). But for our main experiments—based on the above results—we adopted the modified update in eq. (27) with the tunable hyperparameters \(\nu\) and \(\gamma\). Moreover, as shown in section 3, these hyperparameters did not require elaborate tuning to be effective in practice.

**B.2 Effects of initialization**

We also compared the effects of different initializations. For these comparisons, we experimented with the same \(d = 4\) model as in the previous section, but in addition, we trained models whose weight matrices were initialized by different levels of zero-mean Gaussian noise. In particular, for each noise level, we generated twenty different
initializations and trained the resulting models for 1000 iterations. Then for each of these models, we recorded the lowest classification error rate that it achieved on the 60K training examples over the course of learning.

Figure 8 shows the results of these experiments as a box plot over these twenty different initializations. For comparison, we also show the result from the model with the subspace initialization in eq. (28). As expected, there is some spread in the results obtained with the random initializations. However, we found that none of the randomly initialized models (on either data set) learned a classifier as accurate as the model with subspace initialization. For these types of experiments, it is of course necessary to train a large ensemble of randomly initialized models, and therefore we cannot claim to have observed this behavior across a large number of models of different types and sizes. We also note that Figure 8 reports error rates on the training examples, not the test examples, so there may be some benefits of random initialization for preventing over-
fitting. Nevertheless, based on the above results, we adopted the subspace initialization in eq. (28) for all of our experiments in section 3.

Acknowledgements

The author is grateful to the anonymous reviewer whose suggestions improved the paper in many ways.

References


