6.867 Problem Set 3

Anonymous authors

1 Neural Networks

To explore the effect of hyper-parameter choice, initialization, and learning rate on neural network training and accuracy, a flexible neural network function was implemented in MATLAB with the intention of classification.

1.1 ReLU + Softmax

For classification purposes, the output layer of the neural net is chosen to be a Softmax layer $\left(f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}\right)$ with ReLU hidden layers ($f(z) = \max(0, z)$). The corresponding loss is: $Loss = -\sum_{i=1}^{k} y_i \log(f(z)_i)$, with $f(z)_i$ being the Softmax function. In this formulation, y_i is a one-hot vector corresponding to one training point (x, y), ($ie.\, y_i = 1$ if the training point is classified as class i and $y_i = 0$ otherwise). To use a Softmax output layer, the derivative $\frac{\partial Loss}{\partial z^L} = \delta^L$ needs to be derived to allow calculation of the neuron weights by Backpropagation. The basic derivation is shown below:

$$\frac{\partial Loss}{\partial z^L} = \frac{\partial a^L}{\partial z^L} \nabla_a Loss, \qquad where\ a_i^L = f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

$$\frac{\partial a_i}{\partial z_i} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} - \frac{e^{z_i} e^{z_i}}{\left(\sum_{j=1}^{k} e^{z_j}\right)^2}, \qquad \frac{\partial a_i}{\partial z_j} = \frac{-e^{z_i} e^{z_j}}{\left(\sum_{j=1}^{k} e^{z_j}\right)^2}, \qquad \frac{\partial Loss}{\partial a_i} = -\frac{y_i}{a_i}$$

The use of the Softmax output layer only affects the neural network implementation during 1) the feedforward step to calculate the activations (a) in the output layer and 2) the output layer derivative w.r.t the weighted layer inputs, $\frac{\partial Loss}{\partial z^L} = \delta^L$ that begins the Backpropagation scheme for layer-wise gradient calculation.

1.2 Initialization

In order the train the neural network, the initialization scheme for the neuron weights needs to be determined. Due to the saturating non-linearities, the weights need to be kept close to zero to avoid "explosions"; however, the weights cannot all be initialized to zero (as may commonly be done) because then the gradients will all be zero and the neural network (NN) cannot be trained by gradient descent. This condition does not apply to the bias terms because these are then always multiplied by the subsequent small non-zero initial weights and the gradients will then be non-zero even if the bias terms are equal to zero. Therefore, the bias terms in our implementation are initialized to zero. A common approach for weight initialization is to draw the weights randomly from a Gaussian distribution with mean zero and some variance. We want to choose the variance such that the initial variance of the neuron's value is independent of the size of the previous layer $\left(Var(z^l) = Var(z^l | n^{l-1}) \to Var(z^l) = Var(z^{l-1})\right)$. If we zoom in on each layer, the initial neuron values are $z^l = W^l a^{l-1}$. Then assuming that the weight matrix and previous layer activations are independent, each neuron's weights are i.i.d per layer, and the activations are ReLU:

$$Var(z^l) = Var(W^l a^{l-1}) = n^l Var(w^l a^{l-1}) = n^l Var(w^l) E[(a^{l-1})^2], \qquad where\ E[(a^{l-1})^2] = \frac{1}{2} Var(z^{l-1})$$

$$\to Var(z^l) = n^l Var(w^l)\left(\frac{1}{2} Var(z^{l-1})\right) \to \boldsymbol{Var(w^l) = \frac{2}{n^l}} \quad to\ satisfy\ Var(z^l) = Var(z^{l-1})$$

So, the variance of the Gaussian distribution for weights in each layer should be $Var(w^l) = \frac{2}{n^l}$, where n^l is the number of neurons in layer l; biases are initialized to zero for all neurons.

1.3 Regularization

As with other types of regression and classification methods, we may be interested in adding a regularization term on the layer weight matrices to avoid overfitting. We currently consider adding a Frobenius norm regularizer to two of the hidden layers, denoted by superscript 1 and 2:

$$J(w) = Loss(w) + \lambda\left(\left\|W^{(1)}\right\|_F^2 + \left\|W^{(2)}\right\|_F^2\right) \ where\ \|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$$

To train the weights, we need to add a term to the gradient for the layers affected that takes into account the loss due to the regularization term. Specifically, we are interested in $\frac{\partial Loss}{\partial W}$ in the original formulation, which now becomes $\frac{\partial J}{\partial W^l} = \frac{\partial Loss}{\partial W^l} + 2\lambda W^l$ if $l = 1, 2$ and $\frac{\partial J}{\partial W^l} = \frac{\partial Loss}{\partial W^l}$ otherwise in our current case. The first term on the R.H.S of the first equation is the cross-entropy loss due to the change in weights; this term is dependent on the layers preceding it and the term that back propagates. The second term on the R.H.S is the regularization loss term that does not propagate and depends solely on the weight values in one layer (1 and 2 in this case). So, for all layers for which regularization is required/implemented for, the second term ($2\lambda W^l$) needs to be added to the gradient of effected layers after Backpropagation is complete to implement regularization during training.

1.4 Binary classification

To test the implemented neural network (without regularization), the 4 2-D datasets from the previous paper were used. These datasets have an input vector of length 2 and are separated into 2 classes. These include 1 linearly separable dataset (denoted DS1), 1 that is amenable to a linear decision boundary (denoted DS3), 1 that has significant overlap (denoted DS1), and 1 that has significant error when a linear decision boundary is used (denoted DS4). Several different NN architectures are used (single hidden layer with 5 and 500 neurons, two hidden layers with 5 and 500 neurons for each layer), with the classification accuracy compared to linear and RBF kernel SVM in Table 1. To speed training, a validation set is used to determine early stopping; training is stopped when validation classification accuracy has not improved for 10 epochs. The decision boundaries for NN[1,5] on the four data sets are shown in Figure 1; increasing the NN complexity on DS2 is shown in Figure 2. Based on the results, it can be seen that the NN performs perfectly on DS1, which is expected as perceptron would work on this data set and performs better than the linear SVM for all data sets and NNs. NN classification accuracy can be seen to be comparable to RBF SVM, with a possibly slightly better performance on DS2. Furthermore, NN complexity doesn't seem to affect classification accuracy except for DS2, suggesting that complexity is only beneficial when the underlying "true" classification function is complex and further modeling is required as is the case with DS2. This can be seen in Figure 2, as the decision boundary becomes highly nonlinear as NN complexity is increased. This is in contrast with the other data sets, where the decision boundary is relatively constant as NN complexity is increased.

Parameter	NN [1,5]	NN [2,5]	NN [1,500]	NN [2,500]	SVM Linear	SVM RBF
DS1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
DS2	13.5%	7.5%	7.5%	**4.5%**	19.0%	8.5%
DS3	4.5%	**3.5%**	5.0%	5.5%	6.0%	**3.5%**
DS4	5.0%	**4.0%**	5.0%	6.0%	29.75%	5.0%

Table 1. Classification accuracy of neural networks and SVM. Note NN[1,5] denotes 1 hidden layer with 5 neurons per layer.

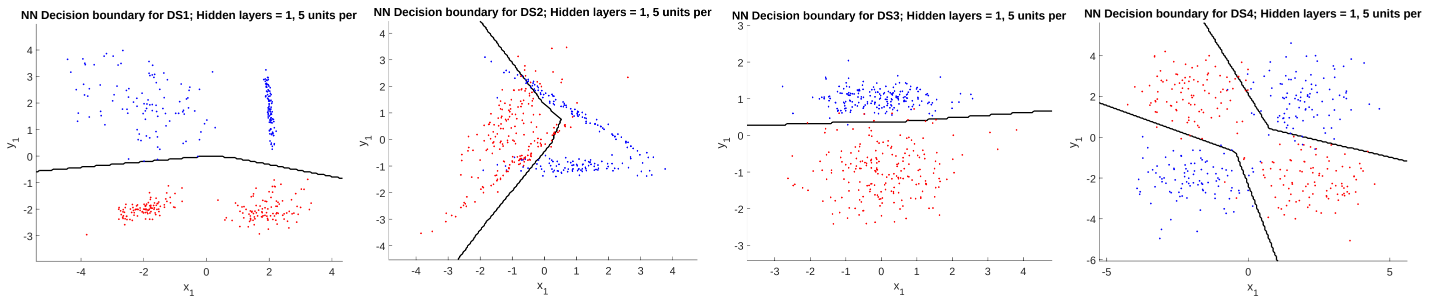

Figure 1. Decision boundary for the NN[1,5] on training data for DS1-DS4

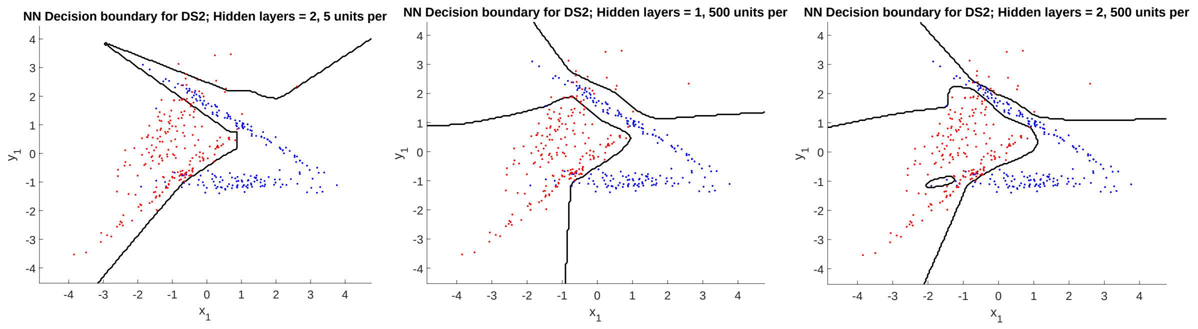

Figure 2. Effect of NN complexity on the decision boundary for DS2 training data (left to right increasing complexity)

1.5 Multi-class classification

The neural network is now expanded for multi-class classification using the MNIST data. Each digit is considered its own class and the neural network is used to classify each digit directly. The data is normalized and split into training, validation and test sets; different training sizes are used and 200 samples per class are used for the validation and test sets. If the data is not normalized, classification accuracy is very low for a low epoch number as the training is slowed down drastically and sufficient accuracy may not be obtained by the specified maximum epoch number. Classification accuracy is compared to the accuracy of RBF SVM and logistic regression for the cases presented in the previous paper. The test accuracies and cases for the binary classifiers (SVM and LR) are shown in Table 2. NN classification accuracy with various hidden layers, layer size and epoch number are shown in Table 3. Several things stand out. First, it can be seen that low complexity neural networks ([1,10] and [2,10]) do not perform as well as more complex networks ([2,100] and [1,100]). Second, low complexity neural networks have a classification accuracy similar to the "even vs. odd" case for SVMs and LR; more complex networks on the other hand achieve an accuracy similar to single digit binary classification for SVM and LR. This suggests that for most cases, a NN is most effective for multi-class classification unless only binary classification is required, at least in this case. Third, increasing the epoch number for the more complex networks only results in a slight improvement in the error rate. Finally, increasing the node number to 500 from 100 in the 2 layer case only has a very small effect on the error rate, suggesting that error rate plateaus with respect to network complexity at some point.

Case	1 vs 7 raw	1 vs 7 normalized	3 vs. 5 raw	3 vs. 5 normalized	4 vs. 9 raw	4 vs. 9 normalized	Even vs. Odd raw	Even vs. Odd normalized
LR Test Error	0.66%	1.33%	6.66%	4.0%	5.66%	5.66%	14.13%	11.6%

SVM Test Error	1.33%	1%	5.33%	4.66%	5.33%	6.33%	16.33%	11.2%

Table 2. Misclassification rates for binary classification of MNIST data for LR and SVM

Parameter	NN[1,10], 50 epochs	NN[2,10], 50 epochs	NN[1,100], 50 epochs	NN[2,100], 50 epochs	NN[1,100], 381 epochs	NN[2,500], 115 epochs
Error Rate	16.8%	23.2%	8.93%	7.73%	8.06%	7.33%

Table 3. Neural network misclassification rates for MNIST classification

2 Convolutional Neural Networks

2.1 Convolutional filter receptive field

First, to understand the effect of layer number convolutional neural networks (CNN), a simple example is considered with two convolutional layers in series. The first layer applies a 5x5 patch to the image, producing feature map Z_1 and the second applies a 3x3 patch to Z_1 to produce a second feature map, Z_2. We are interested in how this affects the *receptive field (RF)* of a node in the second layer, or how much a node in the second layer can "see" of the original image. Since a 5x5 patch is applied to form Z_1, the RF of a node in Z_1 is 25 pixels (a patch forms the input to one node in Z_1). The patch size for the second layer is 3x3, so the RF of a node in Z_2 is 9 nodes in Z_1; the resulting RF of a node in Z_2 is then $9(25) = 225$ pixels.
Based on this simple example, it is apparent that adding if additional convolutional layers are added in series, then each node in the subsequent layers will have an RF that is exponentially larger (ie. If there were three layers, each with 3x3 patches, the RF of a node in the third layer would be $(3^2)^3 = 729$.). It follows that a deeper network will result in nodes that take into account more of the original input image, which may improve network accuracy.

2.2 Convolutional network implementation

A CNN is implemented with Tensorflow with the following parameters:

- 2 convolution layers followed by 1 flattening layer and a dense layer with 64 nodes
- ReLU activation function for the convolution and dense layers
- A softmax cross entropy loss function is used
- The loss function is minimized through gradient descent

The implementation is tested on a set of 451 works of art from 11 different artists created by Zoya Bylinskii. Training took 1500 steps resulting in 96.2% training accuracy on the full training set and 62.1% validation set accuracy. During training, the network is adjusting the weights according to the gradients calculated with the training data and calculating the resulting classification accuracy at each step with the validation set. The validation set classification accuracy can be used to terminate training if the accuracy is non-increasing for a set number of steps.

2.3 Network features

There are several methods to improve CNN performance and several are investigated here to determine their effect. The methods that are investigated are: 1) early stopping and 2) the addition of pooling layers.

2.3.1 Early stopping

First, the effect of early stopping on CNN validation accuracy is investigated. The validation and training accuracies of the CNN are observed during training as a function of training steps; shown in Figure 3. It can be seen that the validation accuracy stops increasing after training step 1000 and only the training accuracy increases. Therefore, it can be concluded that stopping CNN training at step 1000 or soon thereafter would result in similar classification accuracy as the fully trained CNN with a much lower training time.

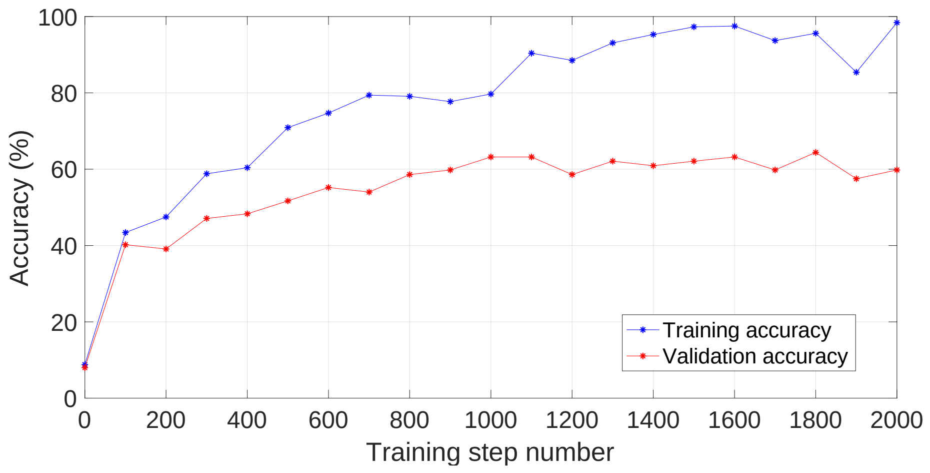

Figure 3. Training and validation accuracy for the CNN as a function of training steps

2.3.2 Addition of pooling layers

To investigate the effect of pooling layers, several different pooling schemes are used. The schemes chosen are a combination of pooling location (after the 1st or 2nd convolution layers or both), pooling size, and stride length. The accuracy for the combinations chosen are shown in Table 4.

	2 layer pool, 2x2, 2 stride	2 layer pool, 3x3, 2 stride	2 layer pool, 4x4, 2 stride	2 layer pool, 3x3, 3 stride	1st layer pool, 2x2, 2 stride	2nd layer pool, 2x2, 2 stride
Training accuracy	83.8%	75.0%	73.4%	76.1%	90.4%	**92.6%**
Validation accuracy	62.1%	59.8%	52.9%	57.5%	**62.3%**	58.6%

Table 4. Effect of pooling on CNN accuracy

Several things stand out in Table 4. First, when pooling is used after both convolution layers, increasing pooling size has a negative effect on validation set accuracy. This is likely because pooling is works to smooth the data, in a way possibly "destroying" some of important features in the data. Second, increasing the stride length seems to decrease validation accuracy. Increasing stride reduces the layer volume and can be seen as increasing regularization, which may not be helpful in this case. If an increased stride increases validation accuracy, it may be that the CNN is overfitting a little on the training data. Third, pooling on the first layer seems to have a positive effect on validation accuracy while pooling on the second layer seems to have a negative effect on validation accuracy. This is likely because how information flows in the CNN. Pooling after the first layer regularizes the data before the second convolution layer, acting as a smoothing filter for the second layer, so the effect flows downstream. On the other hand, pooling after the second (and last) layer smooths the input to the dense layer immediately. The lack of another convolution layer after this pool may be the cause of the reduced validation accuracy.

2.4 CNN accuracy with data variation

Next, we're interested in how our "optimal" CNN performs on distorted versions of the training images. Based on the results in the previous section, a CNN with 2x2, 2 stride pooling only after the 1st convolution layer is used with 1100 training steps. The distortions used along with the classification accuracy are shown in Table 5 for the "optimal" CNN, original CNN (1500 steps, no pooling), pooling CNN (1500 steps, 1st layer pooling), and early stopping CNN (1100 steps, no pooling).

Distortion	Normal data	Translated data	Brightened data	Darkened data	High contrast data	Low contrast data	Flipped data	Inverted data
CNN accuracy (optimal)	**66.7%**	40.2%	37.9%	47.1%	63.2%	**59.8%**	41.4%	2.3%
CNN accuracy (original)	63.2%	26.4%	46.0%	**43.7%**	64.4%	**59.8%**	**47.1%**	**8.0%**
CNN accuracy (pooling)	62.1%	**43.7%**	47.1%	40.2%	**67.8%**	56.3%	40.2%	4.6%
CNN accuracy (early stopping)	64.4%	28.7%	**48.3%**	41.4%	64.4%	56.3%	42.5%	**8.0%**

Table 5. CNN accuracy on distorted data sets

Examining the results in Table 5 it is a bit surprising that most of the distortions result in similar CNN accuracies (~40%), with the exception of the contrast data which have similar accuracy to the normal data. Additionally, the CNN seems to have an extreme amount of trouble with the inverted data, showing a complete inability to classify the images under the color inversion distortion. These two results suggest that the main features the CNN has learned are color patterns (how the colors relate to each other pixel to pixel) and the actual colors that are used. This is because the distortions with the exception of the inversion maintain color data and color patterns throughout the images. The relatively high accuracy of the high and low contrast sets can be explained by the fact that color data and patterns are the mostly the same, they just work to bring the colors more or less in contrast respectively.
Comparing the difference in accuracies between the CNNs with different hyper-parameters, two things stand out. First, pooling seems to have a significant positive effect on the classification accuracy of the translated data and a negative effect on the inverted data. Second, the combination of pooling and early stopping in this case have a synergistic negative effect on CNN accuracy on the brightened and inverted data. The takeaway here is that different network features allow the CNN to learn/neglect different image features and that CNN features do not have an independent effect on CNN classification ability.

3 Topic Models

For this section, document topic modeling with latent Dirichlet allocation (LDA) is considered. The LDA implementation within MATLAB 2017b is used with the 20Newsgroups dataset, which consists of 19,997 articles for 20 categories taken from the Usenet newsgroups collection.

3.1 Training the LDA model

First, an LDA model is trained with 100 topics as the hyper-parameter. The documents are read in and pre-processed to remove words of length 1 (ie. Words such as I, a, etc.), words given by the MATLAB function *stopWords* (which is described as "common words removed before document analysis") and other words added based on training iterations that were found to not be useful (such as "EDU", "com", etc). Pre-processing is done to avoid the LDA model placing emphasis on common words that are topic-agnostic.

3.2 Generated topics

The topics discovered and associated words for 10 randomly chosen topics are shown in Figure 4. Some of the topics shown are easily identified (44: hard drives, 57: looking to buy, 90: philosophy, 66: black holes, 9: earth studies) while some of the others are not so clear. Overall, the topics shown can be inferred to be sub-topics of the 20 topics in the underlying data. To examine this inference, we can look at two of the original article categories and observe which topics (and their corresponding words) are most probabilistically associated with these articles on an average basis.

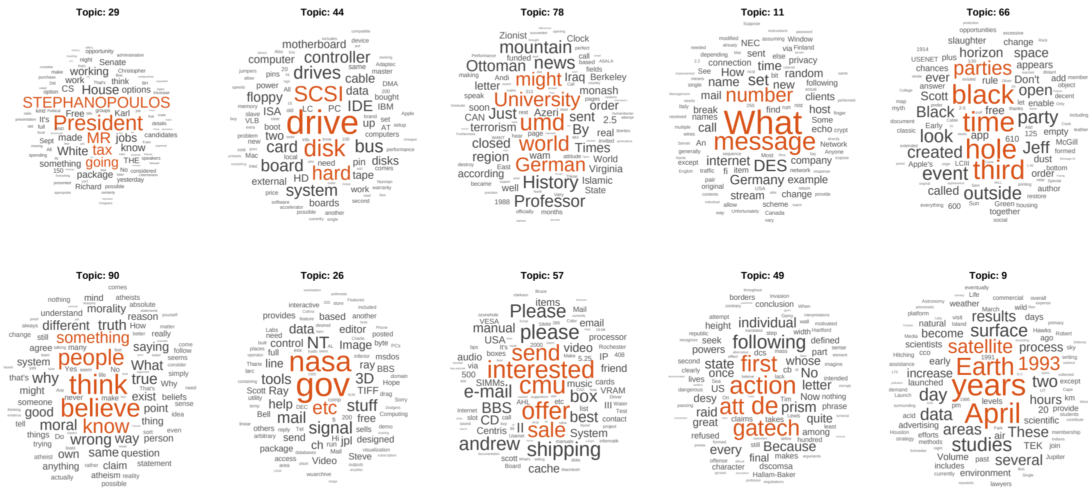

Figure 4. Randomly chosen word clouds for LDA model with 100 topics

The article topics "sci.space" and "rec.autos" are chosen. The three topics most associated with these article groups by the LDA model are topics are 68, 30, and 93 for "rec.autos" (with probability 4.6%, 2.9%, and 2.3%) and 79, 59, and 66 for "sci.space" (with probability 2.57%, 2.52%, and 2.45%). The word clouds associated with these topics are shown in Figure 5. For "rec.autos," the words associated with the top three topics make sense, with topic 68 containing words directly associated with automobiles such as "car" or "cars", topic 30 containing words that might be associated with purchasing/selling cars and topic 93 containing words that might be associated with operating a car like "back up" or "go". For "sci.space", the words in each of the three topics also make sense, with topic 66 containing words relating to the American space program, NASA, topic 59 containing words describing needs and problems associated with the space industry, such as "access" and "net cost," and topic 79 containing words related to thinking about space such as "believe", and "think" (space is a common source of inspiration for thinking about one's place or beliefs).

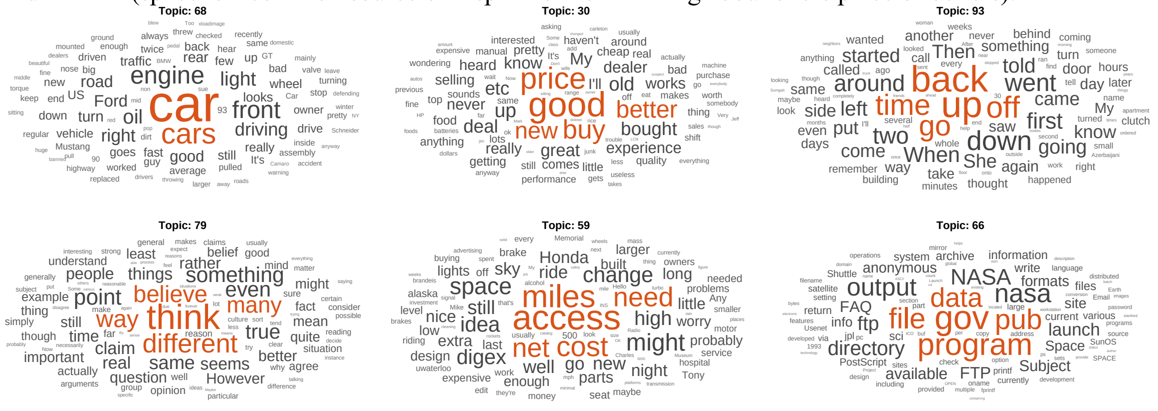

Figure 5. Word cloud for three highest probability topics in "rec.autos" (top row) and "sci.space" (bottom row)

3.3 Effect of topic size

Next, the effect of topic size on LDA topic generation is investigated. An LDA model was trained for 50, 20, and 10 topics; the resulting word clouds for the model topics are shown in Figure 8, Figure 7, and Figure 6 respectively. Examining the word clouds generated, the effects of reducing topic number are apparent.

First, comparing the word clouds for the 10 and 20 topic trained LDA models, it seems that reducing topic number forces the model to compress the words in each topic together. For example, in the 20 topic word cloud, numbers are confined to their own topic (topic #8) while they are combined with other words in the 10 topic word cloud (topic #4). This follows logically with knowledge of the underlying data (20 topics); reducing the topic number from 20 should force some words in the topics to be intermingled. This is why

some of the topics in the 10 topic word cloud are hard to classify by one “real” topic and the higher topic word clouds are more distinct. Conversely, the LDA models with a high topic count result in topics that seem to be more specialized/nuanced than the topics in the underlying data. Topic 44 shown in Figure 4 for example seems to be about hard drives, which is a specific piece of hardware in a computer (two topics in the underlying data).
Second, examining the various input topics from the 20Newsgroups data set, some of the topics are closely related (for example, it can be reasoned that the “rec.sport.hockey” and “rec.sport.baseball” documents are very similar in words used) so even though there are 20 topics, fewer than 20 topics may be used to represent the document set. This may be one reason that 20 topic LDA has a topic full of numbers (#8) even though it is not one of the underlying topics.

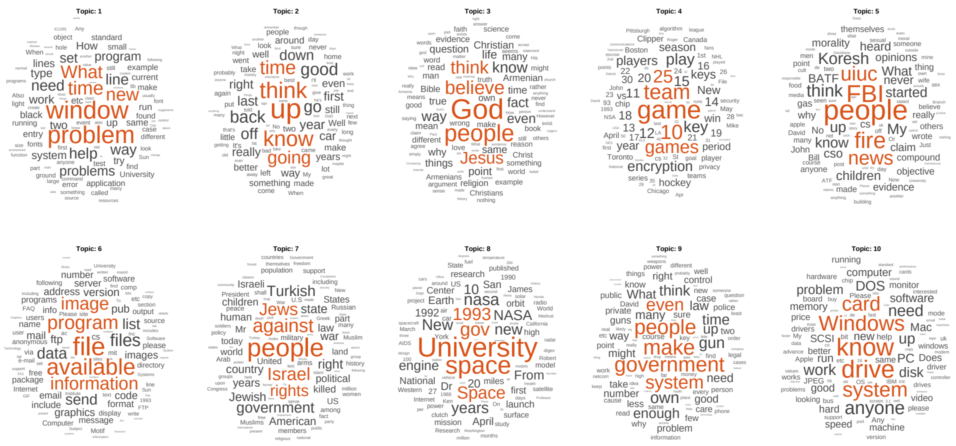

Figure 6. Word clouds for LDA model with 10 topics

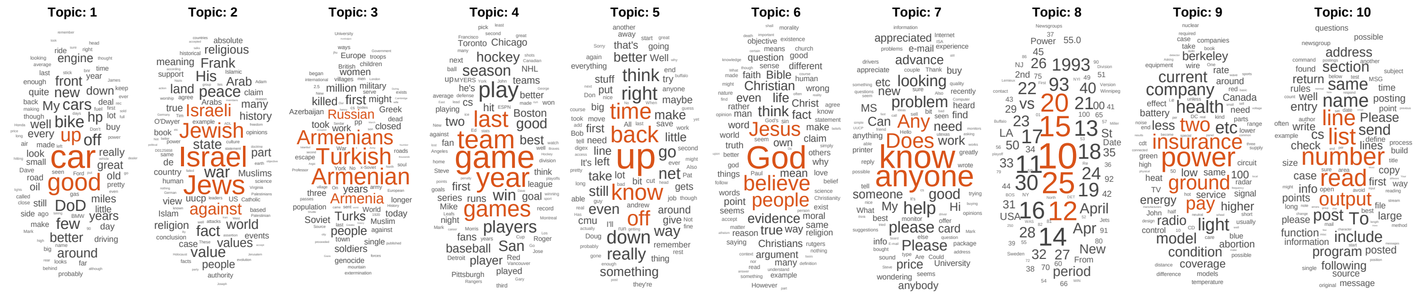
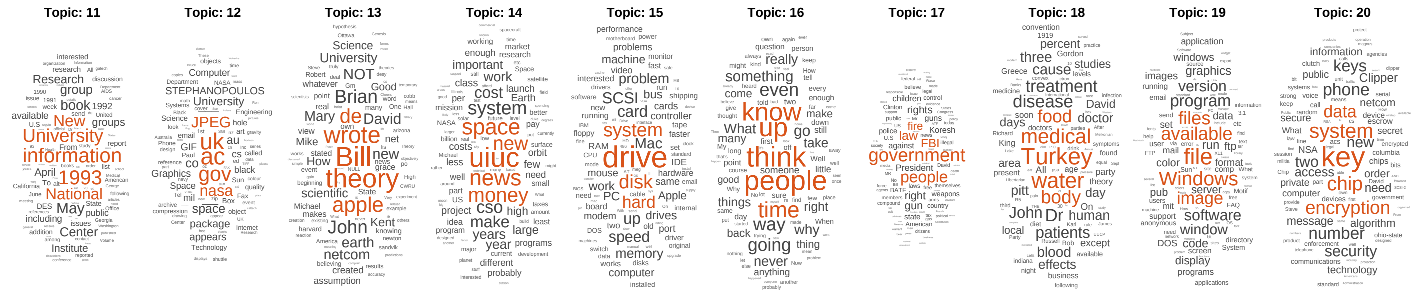

Figure 7. Word clouds for LDA model with 20 topics

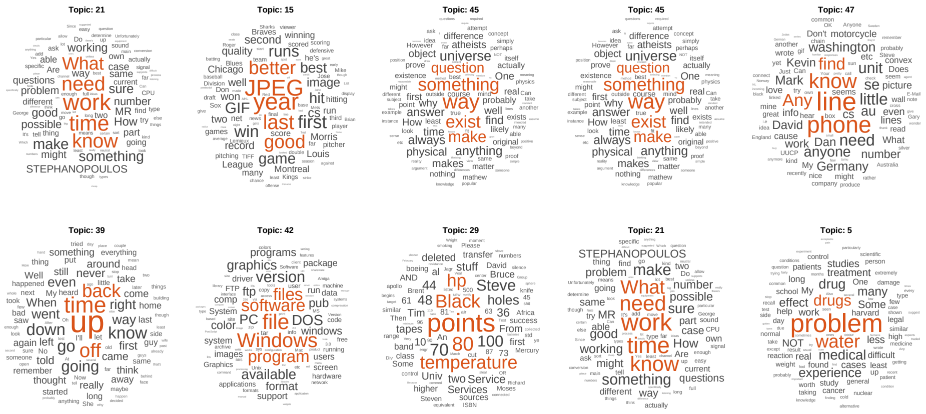

Figure 8. Randomly chosen word clouds for LDA model with 50 topics