CO395 Group 57

Dongxiao Huang, Zheng Xun Phang, Yufeng Zhang

Question 1

Linear layers

If each image in X is reshaped into a row vector, then the forward pass can be written as

$$z = XW + \mathbf{1}b^T$$

For a linear layer, the partial derivatives are

$$\frac{\partial z}{\partial X} = W \qquad \frac{\partial z}{\partial W} = X \qquad \frac{\partial z}{\partial b} = \mathbf{1}$$

To compute derivatives of the loss function L, we apply the chain rule

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial X} \qquad \frac{\partial L}{\partial W} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial W} \qquad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b}$$

Of course, images are not row vectors, so they must be reshaped appropriately.

ReLU activation

The forward pass of ReLU is $\max(0, x)$.

Its derivative is 1 for $x > 0$ and 0 for $x < 0$. At $x = 0$, its derivative does not exist, but its subderivative lies between 0 and 1, so we simply set it to 0. We apply the chain rule just like the linear layers.

Question 2

Training Phase

During training, the forward pass of inverted dropout will multiply the output of some neurons by 0, so it's effectively removing neurons. We set the proportion p of neurons to dropout, and scale the outputs of the remaining neurons by $1/(1-p)$. The backward pass computes

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial X} = \frac{\partial L}{\partial z}\frac{\partial \begin{Bmatrix} \mathrm{X}_{ij} & if\ D_{ij} = 1 \\ 0 & if\ D_{ij} = 0 \end{Bmatrix}}{\partial X} = \frac{\partial L}{\partial z}\frac{\partial \begin{Bmatrix} 1 & if\ D_{ij} = 1 \\ 0 & if\ D_{ij} = 0 \end{Bmatrix}}{\partial X} = \frac{\partial L}{\partial z}\mathcal{D}$$

Testing Phase

During testing, we use all neurons in both forward and backward pass, as if they were never dropped out. The backward pass computes

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial z}\mathbf{1}$$

Question 3

The softmax function $\sigma : \Re^C \to [0,1]^C$ can represent a probability distribution over C classes:

$$\sigma(z_1, \ldots, z_C) = \frac{1}{\exp(z_1) + \cdots + \exp(z_C)} \begin{bmatrix} \exp(z_1) \\ \vdots \\ \exp(z_C) \end{bmatrix} := \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_C \end{bmatrix}$$

We used the "normalization trick" described in the coursework manual for numerical stability.

Derivatives of the softmax function are

$$\frac{\partial \sigma_i}{\partial z_j} = (\delta_{i,j} - \sigma_j)\,\sigma_i \qquad \forall\, i, j \in \{1, \ldots, C\}$$

where $\delta_{i,j} = 0$ if $i \neq j$, otherwise $\delta_{i,j} = 1$.

The cross entropy loss of n images is

$$L = -\frac{1}{n} \sum_{i=1}^{n} [y_{i,1} \log \sigma_{i,1} + \cdots + y_{i,C} \log \sigma_{i,C}]$$

where $y_{i,k} = 1$ if image i belongs to class k, otherwise $y_{i,k} = 0$. Similarly, $\sigma_{i,k}$ is the softmax probability that image i belongs to class k.

Derivatives of the loss function for $k \in \{1, \ldots, C\}$ are

$$\begin{aligned}
\frac{\partial L}{\partial z_k} &= -\frac{1}{n} \sum_{i=1}^{n} \left[\frac{y_{i,1}}{\sigma_{i,1}} \frac{\partial \sigma_{i,1}}{\partial z_k} + \cdots + \frac{y_{i,C}}{\sigma_{i,C}} \frac{\partial \sigma_{i,C}}{\partial z_k} \right] \\
&= -\frac{1}{n} \sum_{i=1}^{n} [y_{i,1}\,(\delta_{1,k} - \sigma_{i,k}) + \cdots + y_{i,C}\,(\delta_{C,k} - \sigma_{i,k})] \\
&= \frac{1}{n} \sum_{i=1}^{n} (\sigma_{i,k} - y_{i,1}\,\delta_{1,k} - \cdots - y_{i,C}\,\delta_{C,k}) \qquad \text{since } y_{i,1} + \cdots + y_{i,C} = 1
\end{aligned}$$

Note that $y_{i,1}\,\delta_{1,k} + \cdots + y_{i,C}\,\delta_{C,k}$ is 1 when k is such that $y_{i,k} = 1$, and it is 0 for other values of k. So computing the derivative involves going through each row of

$$\begin{bmatrix} \sigma_{1,1} & \cdots & \sigma_{1,C} \\ & \vdots & \\ \sigma_{n,1} & \cdots & \sigma_{n,C} \end{bmatrix}$$

and subtracting 1 from the appropriate column.

Question 4

Sanity Check

Once after the design of network, in order to check whether the network works or not, a small dataset was used to check if the network can reach overfitting and it can reach 50% accuracy effortlessly on the validation set.

Architecture

The shapes of our input and output neurons are $32 \times 32 \times 3$ and 10 respectively.

Our activation function is ReLU and optimization algorithm is Stochastic Gradient Descent.

As a sanity check, we overfitted 50 images in the CIFAR-10 dataset using the architecture and hyperparameters in column A of the table below.

To achieve at least 50% accuracy on the CIFAR-10 dataset, we used the architecture and hyperparameters in column B.

	A	B
Hidden Neurons	[50]	[50, 8]
Dropout	None	None
Regularization	None	L2 with $\lambda = 0$
Epochs	20	20
Batch Size	100	100
Learning Rate	0.001	0.005
Learning Rate Decay	0.95	0.95

Plots of loss function and classification accuracy are provided below.

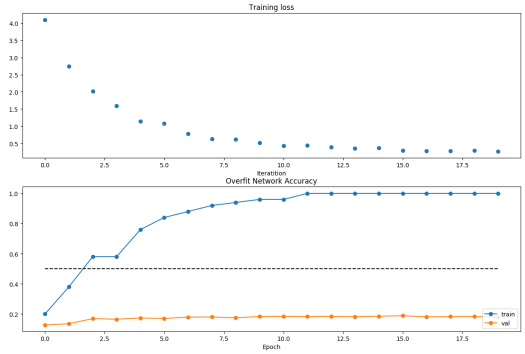


Figure 1: Overfitting

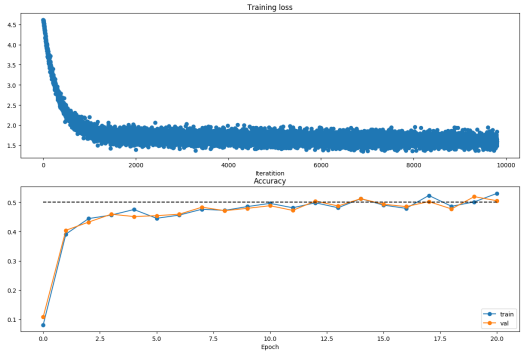


Figure 2: 50% accuracy

Question 5

Step 1: Architecture Design

A reasonable architecture to start with is: 2 hidden layers with about 512 and 128 neurons respectively, since 512 and 128 are between our input size ($48 \times 48 = 2304$ pixels) and output size (7 labels). This is a common rule of thumb; there is no consensus on what a good architecture is anyway.

We initialized the neural net's weights from a standard Normal distribution.

We used the default momentum rate of 0.9.

We used the default learning rate update schedule, which decays the learning rate by 0.95.

For a start, we used the default "stopping criterion" in `solver.py` which is simply the number of iterations. We tried early stopping but it actually led to lower classification accuracy.

Step 2: Learning Rate

We plotted both training and validation classification accuracy (after 9000 iterations) for various learning rates, which range from 10^{-5} to 10^{-2} on a log scale.

Why 9000 iterations? There are about 270 iterations per epoch and we trained our neural net for 35 epochs, for a total of 9450 iterations.

The optimal learning rate is the one with highest validation accuracy i.e. lowest asymptotic error. This turns out to be around 5×10^{-4} and it results in a training accuracy of 72% (error of 28%) and a validation accuracy of 42% (error of 58%).

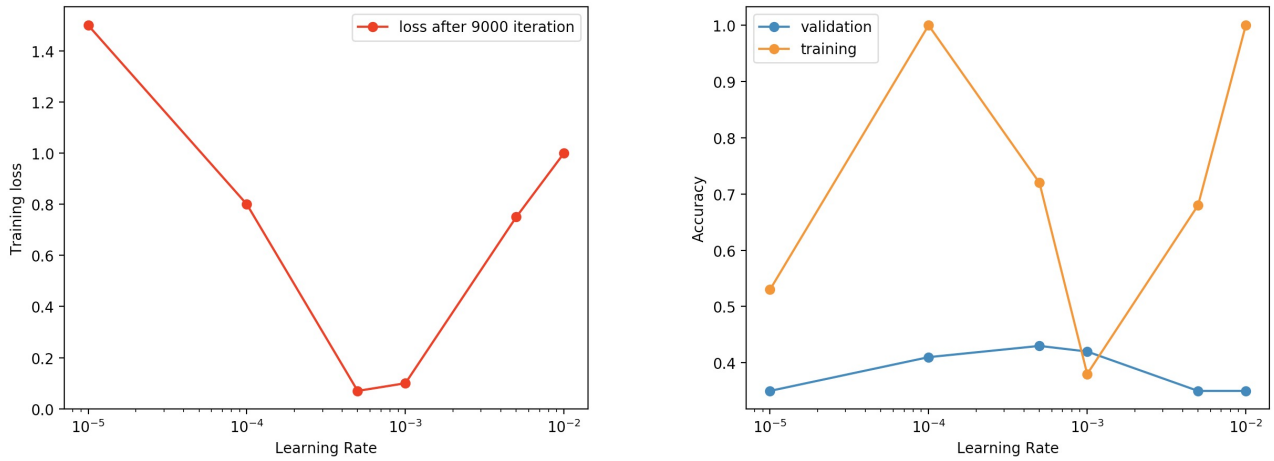
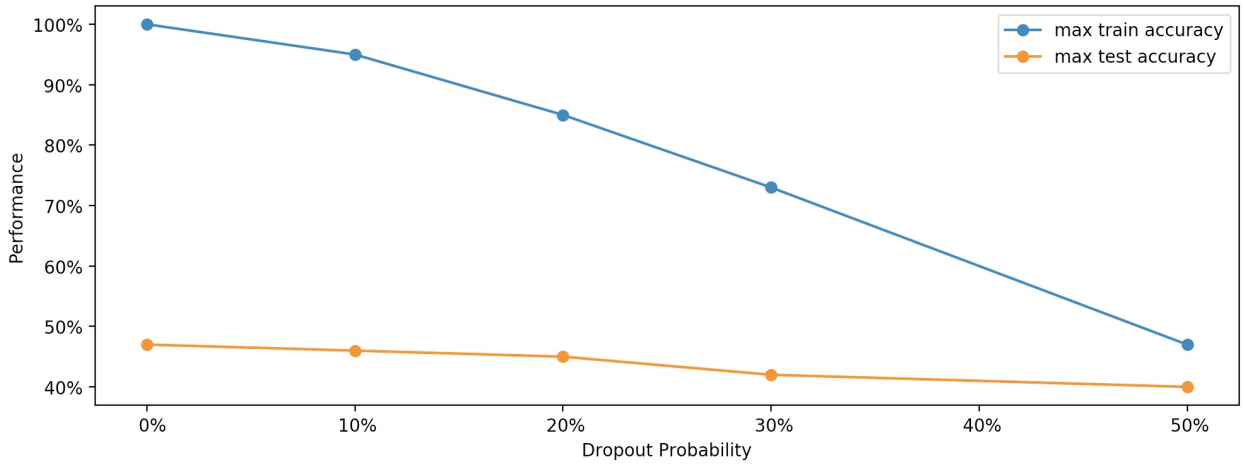


Figure 3: Different Learning Rates' Performance

Step 3: Dropout

We dropped out various proportions of neurons and plotted the training and test accuracy:



Dropout does not improve training and test accuracies. This suggests that our neural net might be underfitting, so we should probably increase the number of neurons.

Step 4: L2 Regularization

We vary the L2 penalty rate λ from 0 to 0.5 and plotted the training loss, training accuracy and validation accuracy in Figure 4.

Just like dropout, L2 regularization does not improve validation accuracy i.e. it is maximized when $\lambda = 0$. This is consistent with our dropout finding that our neural net is underfitting.

Between dropout and regularization, we prefer the latter because it's more easily reproducible without access to our code. Dropout has an element of arbitrariness.

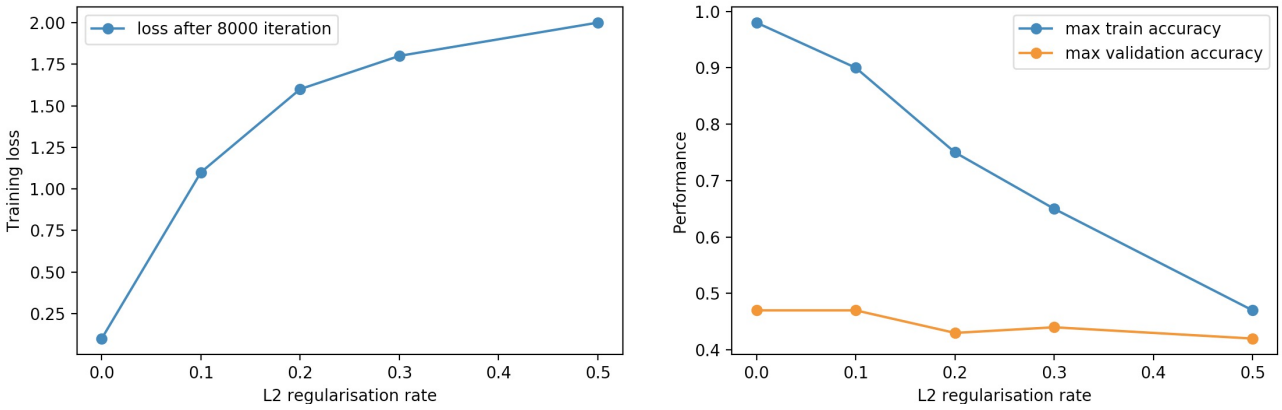


Figure 4: L2 regularization

Step 5: Topology

We have experimented with the following architectures:

- (Wide) 1 layer with 1024 neurons
- (Deep) 5 layers with 512 neurons each
- (Narrowing) 3 layers with [512, 256, 128] neurons
- (Widening) 3 layers with [128, 256, 512] neurons
- (Fan in and out) 3 layers with [512, 128, 512] neurons

The last architecture is inspired by variational auto-encoders. We have also varied the number of neurons for the "narrowing" and "widening" architectures.

Final Architecture and Performance

Our best neural net has [512, 512, 512] neurons and it was trained with these hyperparameters: number of epochs = 35, batch size = 100, learning rate = 0.005 and learning rate decay = 0.95

Confusion matrix based on test data:

Actual Class	Predicted Class						
	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Neutral
Anger	157	3	48	73	88	18	80
Disgust	6	26	4	9	1	2	8
Fear	46	2	166	59	101	42	80
Happiness	62	2	35	568	95	32	101
Sadness	81	1	75	105	241	26	124
Surprise	22	0	73	24	28	275	29
Neutral	39	1	44	107	124	26	266

Note: It makes no sense to compute classification rate for each emotion separately. For example, if the actual emotion is Anger then (Fear, Surprise) counts as a true negative for Anger even though it is a misclassification! In other words, if an instance of Fear is misclassified as Surprise, it should not be taken as evidence that our neural nets are good at detecting Anger.

Summary statistics from the confusion matrix:

	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Neutral
Precision	0.380	0.743	0.373	0.601	0.355	0.653	0.387
Recall	0.336	0.464	0.335	0.635	0.369	0.610	0.438
F_1 score	0.357	0.571	0.353	0.617	0.362	0.631	0.411

$$\text{Classification rate} = \frac{157 + 26 + \cdots + 266}{3625} = 0.469$$

Table 1: Simple Neural Network

Question A1

The dataset of the previous coursework has only 1004 images, which is far too few for training a neural network as it will overfit. With more observations and appropriate regularization, a neural network may outperform a decision tree, so we cannot conclude that one algorithm is better than the other in general.

Even if we had a larger dataset, we need to tune hyperparameters. Without doing so, the decision trees and neural nets have not been trained properly.

In general, no algorithm will perform better than all other algorithms on all datasets (No free lunch theorem).

Finally, there may be other criteria for deciding the best learning algorithm. Even if our neural net has a lower accuracy than a decision tree (which is unlikely), the former does not require hand-coded features like facial action units. On the other hand, decision trees are often considered to be more interpretable i.e. a glass-box model rather than black-box model.

Question A2

For the decision trees coursework, we need to train new decision trees and retrain existing decision trees, because the information gain from splitting on every attribute will change. Of course, we also need to map each emotion to their corresponding integer labels.

For the neural networks coursework, we only need to change the `num_classes` parameter to the new number of classes we have. This will automatically increase the dimensions of our softmax output vector, so we don't need to encode emotions like in decision trees.

The entire neural net must be retrained since new emotions will affect the softmax probabilities of existing emotions. If we have normalized the training data by subtracting the mean image, then we need to recompute the mean image too.

For both learning algorithms, we need to retune all hyper-parameters by cross validation. For decision trees, this means tuning the maximum depth. For neural nets, this means adjusting the learning rate, network topology (number of neurons and layers) and regularization factor.

Question 6

We have trained a convolutional neural network (CNN) with batch size = 32, number of epochs = 150 and the following architecture:

Data Argumentation	1. Randomly rotate images 45 degrees 2. Randomly (25%) shift images horizontally and images vertically 3. Randomly flip images horizontal and vertically
Convolution2D	2 convolutional layers of 32 kernels with size of 3×3 using ReLU activation
Optimization	2×2 Maxpool Layer (and dropout with probability 0.25)
Convolution2D	2 convolutional layers of 64 kernels with size of 3x3 using ReLU activation
Optimization	2×2 Maxpool Layer (and dropout with probability 0.25)
Full Connected Network	512 neurons Activation: ReLU Dropout with probability 0.5 Activation: Softmax

Confusion matrix based on test data:

Actual Class	Predicted Class						
	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Neutral
Anger	242	3	51	21	76	8	66
Disgust	23	11	6	2	9	2	3
Fear	70	0	143	18	140	41	84
Happiness	41	0	14	714	27	19	80
Sadness	62	0	60	37	313	6	175
Surprise	13	0	59	19	18	293	13
Neutral	43	0	31	50	87	5	391

Summary statistics:

	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Neutral
Precision	0.490	0.786	0.393	0.829	0.467	0.783	0.482
Recall	0.518	0.196	0.288	0.798	0.479	0.706	0.644
F_1 score	0.504	0.314	0.333	0.813	0.473	0.743	0.551

$$\text{Classification rate} = \frac{242 + 11 + \cdots + 391}{3589} = 0.587$$

Table 2: Convolutional Neural Network

Comparing Table 2 with Table 1, it is clear that the precision, recall and F_1 scores of the CNN are better than the simple NN except that the recall of *Fear* is worse for CNN. After using the CNN, the classification rate improves nearly 14%.

This isn't surprising because in the CNN, we use data argumentation techniques, which generates transformed images to train the CNN. The learned weights will be more robust to various transformations and hence less likely to overfit.

Moreover, the convolution kernel helps to extract "local" properties of an image. After using a set of filters, the CNN will learn the most important pattern of images trained. In addition, the maxpool layers will downsample the outputs to reduce the size of representation, which will simplify the data and improve the accuracy. While, in NN, the pictures are read as input neurons, which contain many trivial pixels in data points, so the accuracy will be worse than the CNN.

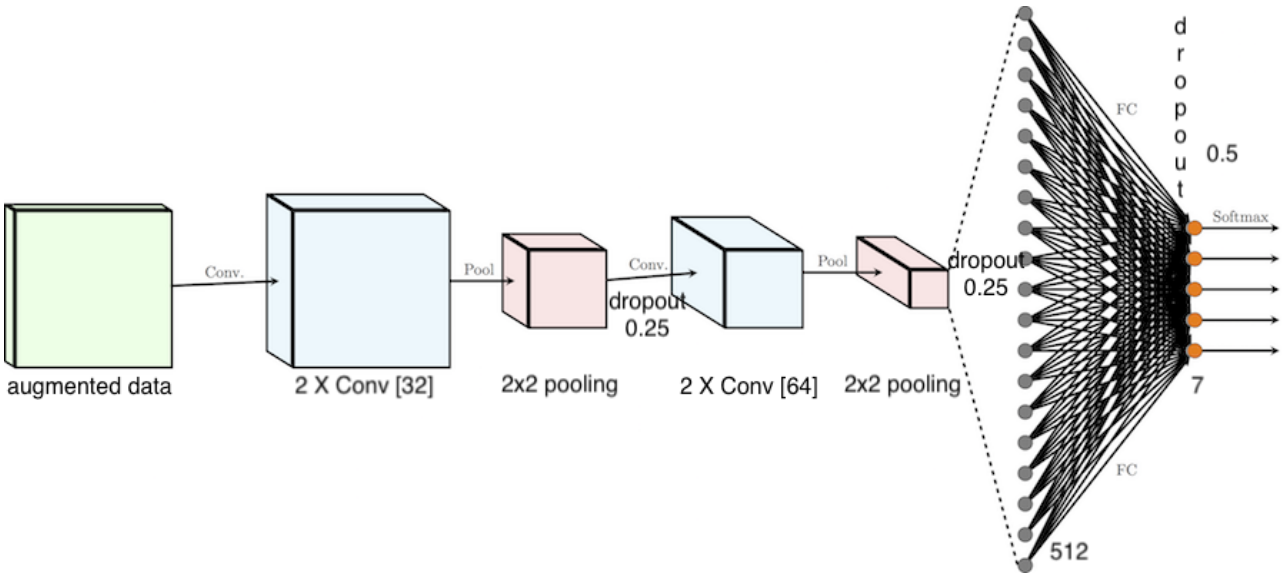


Figure 5: Architecture of Convolutional Neural Network