Lab 2C: Classification

Ignacio Boero, Juan Elenter,
Ignacio Hounie and Alejandro Ribeiro*

February 26, 2023

1 Classification Losses

A classification is a partition of reality. Given realizations $\mathbf{x}$ of some object we ascribe to each of them a unique label y_c that marks them as members of class c. In a classification task we want to train an artificial intelligence that when presented with an input $\mathbf{x}$ predicts a class label $\hat{y}$ that matches the label y_c ascribed by the real world (Figure 1).

To model classification as empirical risk minimization (ERM) we just need to define a proper loss. The easiest is to define a hit loss that takes the value $\ell(y, \hat{y}) = 1$ when $\hat{y} \neq y$ and the value $\ell(y, \hat{y}) = 0$ when $\hat{y} = y$. To write this loss formally we define the indicator function $\mathbb{I}(y = y_c)$ which takes the values

$$\mathbb{I}(y = y_c) = 1 \text{ when } y = y_c, \qquad \mathbb{I}(y = y_c) = 0 \text{ when } y \neq y_c. \tag{1}$$

With this definition we can now write the hit loss as

$$\ell(y, \hat{y}) = \sum_{c=1}^{C} \mathbb{I}(y = y_c) \times \Big[1 - \mathbb{I}(\hat{y} = y_c)\Big]. \tag{2}$$

This is a rather cumbersome expression for the hit loss but it does highlight that the goal is to have predictions $\hat{y} = y_c$ whenever we observe that $y = y_c$. It is also a good starting point for generalizations that we discuss in the next two sections.

*In alphabetical order.

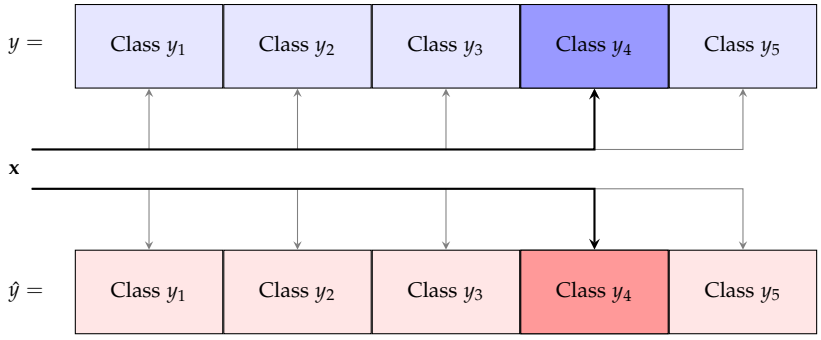

Figure 1. Classification Problem. Associated with input $\mathbf{x}$ nature assigns a unique and definite class label $y = y_c$. We want to train an artificial intelligence that predicts the same label $\hat{y} = y_c$. The hit loss models this requirement [cf. (2)].

1.1 Soft Classification

Class is a particular human obsession. The world rarely assigns unique labels. It is humans that insist on labels. This is relevant because although our data is typically labeled by humans and, therefore, contains unique and definite labels, it is more convenient to make predictions that allow for uncertainty (Figure 2). Thus, instead of predicting a class we predict a vector of C scores $\hat{\mathbf{s}} = [\hat{s}_1; \ldots; \hat{s}_C]$ in which $\hat{s}_c$ represents the predicted likelihood that class y_c is the right class. To train these scores we can modify the hit loss in (2) as follows,

$$\ell(y, \hat{\mathbf{s}}) = \sum_{c=1}^{C} \mathbb{I}[y = y_c] \times \Big[\max_d \hat{s}_d - \hat{s}_c \Big]. \tag{3}$$

The rationale for (3) is that when it comes to make a hard classification decision we opt for the class with the highest score. We therefore want to have $\max_d \hat{s}_d = \hat{s}_c$ when the correct class is $y = y_c$. The loss in (3) is $\ell(y, \hat{y}) = 0$ if this is the case. When $\max_d \hat{s}_d > \hat{s}_c$ the loss in (3) grows away from $\ell(y, \hat{y}) = 0$ in proportion to how far $\hat{s}_c$ is from becoming the maximum score.

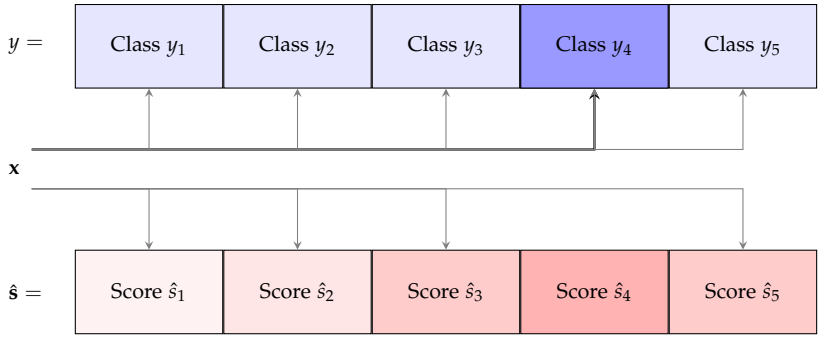

Figure 2. Soft Classification. Instead of predicting a unique and definite class as in Figure 1 we predict scores $\hat{s}_c$ associated with each class. These scores represent the likelihood that the real class is $y = y_c$. The cross entropy loss models this requirement [cf. (5)].

1.2 Cross Entropy Loss

The loss in (3) is a reasonable choice but it is not differentiable with respect to the score variable $\hat{\mathbf{s}}$. This is a drawback for implementing gradient descent. To solve this problem we define the soft maximum function

$$\operatorname{softmax}(\hat{\mathbf{s}}) = \log \sum_{d=1}^{C} e^{\hat{s}_d}. \tag{4}$$

The soft maximum is a differentiable approximation of the maximum in the sense that $\operatorname{softmax}(\hat{\mathbf{s}}) \approx \max_d \hat{s}_d$. We can therefore replace the maximum $\max_d \hat{s}_d$ in (3) by the soft maximum function to define the loss

$$\ell(y, \hat{\mathbf{s}}) = \sum_{c=1}^{C} \mathbb{I}[y = y_c] \times \Big[\log \sum_{d=1}^{C} e^{\hat{s}_d} - \hat{s}_c \Big]. \tag{5}$$

This is the cross entropy loss. Its use in classification tasks is standard.

The motivation of the name cross entropy loss comes from transforming classification scores into classification probabilities. We do that by combining an exponential and a normalization operation,

$$\hat{p}_c = \frac{e^{\hat{s}_c}}{\sum_{d=1}^{C} e^{\hat{s}_d}}. \tag{6}$$

The vector $\hat{\mathbf{p}} = [\hat{p}_1; \ldots; \hat{p}_C]$ is a vector of probabilities because its entries are $\hat{p}_c \in [0, 1]$ and add up to $\sum_{c=1}^{C} \hat{p}_c = 1$. We can then interpret $\hat{p}_c$ as the predicted probability that the correct class is y_c.

We now define the vector of correct probabilities $\mathbf{p}$ in which $p_c = 1$ if $y = y_c$ and $p_c = 0$ otherwise. With these definitions the cross entropy in (5) is equivalent to

$$\ell(\mathbf{p}, \hat{\mathbf{p}}) \;=\; \sum_{c=1}^{C} p_c \times \log \hat{p}_c \;=\; \sum_{c=1}^{C} \Pr[y = y_c] \times \log \Pr[\hat{y} = y_c]. \qquad (7)$$

This quantity is a variation of the entropy of a probability distribution which is defined as $h(\mathbf{p}) = \sum_{c=1}^{C} p_c \times \log p_c$. It is called cross entropy because it mixes the probability p_c of the real probability vector $\mathbf{p}$ with the logarithm $\log \hat{p}_c$ of the probability distribution of the estimated probability vector $\hat{\mathbf{p}}$.

This digression gives as another generalization of (5). Suppose that available labels are not definite. Rather, we are given a probability vector $\mathbf{p}$ whose entries p_c represent the probability that the real class is y_c. We can train with this data if we use the loss

$$\ell(\mathbf{p}, \hat{\mathbf{s}}) = \sum_{c=1}^{C} p_c \times \Big[\log \sum_{d=1}^{C} e^{\hat{s}_d} - \hat{s}_c \Big]. \qquad (8)$$

and produce probability estimates using (6).

The next time you are drawn into a heated discussion on gender, race, or political labels you can point to Equation 8. Classes are not an inherent property of nature. They are just a human construction and we can just as well choose to start assigning classification scores.

2 Readout Layers

In Lab 1B we processed audio with a convolutional neural network (CNN). Our CNN was such that if the input signal contains N entries, the output signal contains N entries as well. This is unsuitable for a classification task in which we want the output to be a vector of scores $\mathbf{s}$ containing as many entries as classes.

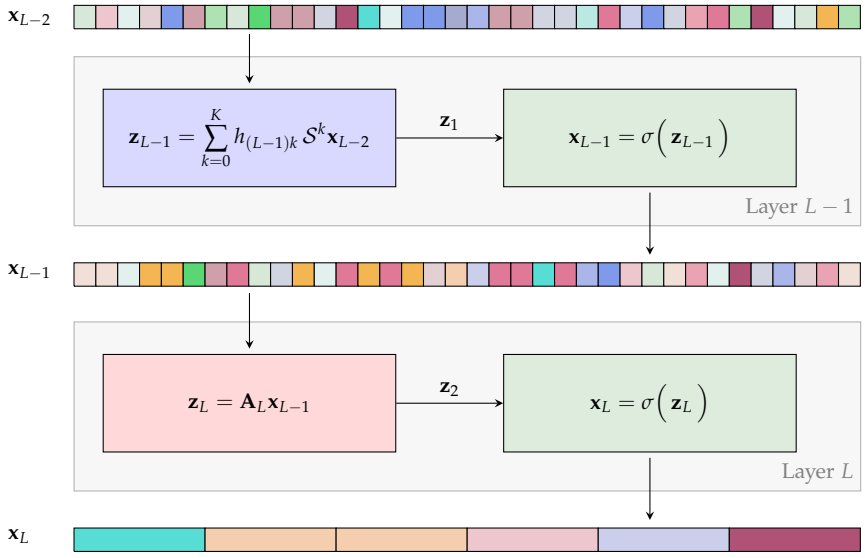

Figure 3. Readout. To use CNNs in classification we add a readout layer. This is a fully connected layer in which the linear map $\mathbf{A}_L$ is of dimension $C \times N$. This matches the dimensionality N of previous layers to the dimensionality C of the number of classes. The output is a vector of likelihood scores for each class.

To sort this out we add a readout layer to match the dimensionalities of the input and output. A readout layer is just a fully connected layer at the output. This is illustrated in Figure 3 for a CNN in which for simplicity we show the last two layers and assume we are processing a single channel. Instead of a convolutional filter, the last layer is the fully connected layer,

$$\mathbf{\Phi}(\mathbf{x}; \mathcal{H}) \;=\; \mathbf{x}_L \;=\; \sigma\Big(\mathbf{A}_L \mathbf{x}_{L-1}\Big). \tag{9}$$

If we have input signals $\mathbf{x} = \mathbf{x}_0$ of dimension N and we have C classes, the matrix $\mathbf{A}_L$ is of dimension $C \times N$. This results in an output $\mathbf{\Phi}(\mathbf{x}; \mathcal{H}) = \mathbf{x}_L$ containing C entries that we can equate to the predicted score $\mathbf{s}$.

In (9) we consider a CNN with layers that have a single channel. In practice, we have already seen that we have to use CNNs with multiple channels. This is not a problem but it requires that we reshape the matrix $\mathbf{X}_{L-1}$ to turn it into a vector. We do that by stacking the columns of $\mathbf{X}_{L-1}$

on top of each other,

$$\text{vec}(\mathbf{X}_{L-1}) = \Big[\mathbf{x}_{L-1}^{1}; \mathbf{x}_{L-1}^{2}; \ldots; \mathbf{x}_{L-1}^{F_{l-1}} \Big]. \tag{10}$$

This operation has no conceptual significance. It is just a reshaping of data. We need to do this to write the readout layer as the matrix multiplication in (9) with $\mathbf{x}_{L-1} = \text{vec}(\mathbf{X}_{L-1})$.

Task 1 Modify the CNN code of Lab2B to endow it with a method that implements a single convolutional layer. Add another method to implement a readout layer. Use these two methods to modify the CNN class to implement a CNN with readout.

This class receives as initialization parameters the number of layers L and the dimensions N and C of the input features $\mathbf{X}$ and the output scores $\mathbf{s}$. To specify the $L-1$ convolutional layers we also accept the vectors $[K_1, \ldots, K_{L-1}]$ and $[F_0, F_1, \ldots, F_{L-1}]$ containing the number of taps K_ℓ of the filters used at each layer and the number of features F_ℓ at the output of each layer.

The forward method of this class takes a matrix $\mathbf{X}$ of dimension $N \times F_0$ as an input and returns the output $\mathbf{\Phi}(\mathbf{x}; \mathcal{H})$ of a CNN with $L-1$ convolutional layers and a readout layer. The dimension of this output vector is C.

Use Relu nonlinearities in all layers.

2.1 Classification of Time Signals

The dimension of the output of a CNN with a readout layer can be chosen to have dimension C equal to the number of classes. We can then train a CNN with readout to classify time signals by solving the empirical risk minimization problem

$$\mathcal{H}^* \;=\; \underset{\mathcal{H}}{\text{argmin}} \; \frac{1}{Q} \sum_{q=1}^{Q} \ell\Big(y_q, \mathbf{\Phi}(\mathbf{x}_q; \mathcal{H})\Big) \,. \tag{11}$$

In (11), the tensor $\mathcal{H}$ contains the convolutional filters in Layers $\ell = 1$ through $\ell = L-1$ as well as the matrix $\mathbf{A}_L$ that defines the readout

layer. The function $\ell(y_q, \mathbf{\Phi}(\mathbf{x}_q; \mathcal{H}))$ is the cross entropy loss between the available class label y_q and the vector of scores $\mathbf{s}_q = \mathbf{\Phi}(\mathbf{x}_q; \mathcal{H})$.

When this CNN is deployed to make operational classifications, the output is a vector of scores $\mathbf{s} = \mathbf{\Phi}(\mathbf{x}; \mathcal{H})$. Entry s_c in this vector is the likelihood we assign to class c. This likelihood can be converted to a hard class decision by selecting the class with the largest score

$$c(\mathbf{x}) \;=\; \underset{d}{\text{argmax}} \;\Big[\mathbf{\Phi}(\mathbf{x}; \mathcal{H})\Big]_d \;. \tag{12}$$

When using CNNs in classification problems we train them to minimize cross entropy losses. However, it is customary to evaluate their performances by counting correct classifications. In particular, if we have a test set with $\tilde{Q}$ entries we evaluate the CNN with the error rate,

$$e(\mathcal{H}) \;=\; \frac{1}{\tilde{Q}} \sum_{q=1}^{\tilde{Q}} \mathbb{I}\Big(y_q \neq \mathbf{\Phi}(\mathbf{x}_q; \mathcal{H})\Big) \;. \tag{13}$$

This is the percentage of erroneous classifications in the test set.

Observe that we can also convert scores $\mathbf{s}$ to probabilities using (6). This is important when we want to gauge the confidence of a classification. In (12) we assign classes to the highest score. Thus, we predict with the same confidence whether the second highest score is close to the highest score or not. Reporting probabilities distinguishes these two situations by saying that the probability of the most likely class is, say, 90% or 60%.

We must say that since we are training with a cross entropy loss this is the interpretation of the output that is most reasonable. The use of (12) is nevertheless more common.

Task 2 Load the data from `dsd.seas.upenn.edu/lab2C/data.zip`. This data contains audio samples in which people are speaking digits from 0 to 9. Use the class of Task 1 to train a CNN with a readout layer to classify the audio signals into the different possible spoken digits. The CNN should have the following parameters: 2 layers, number of channels per layer: 8, kernel size (number of taps) at each layer: 80 and 3 respectively, relu non-linearities, learning rate: 0.01.

Evaluate the training cross entropy loss and the test accuracy in terms of the relative number of incorrect classifications.

The CNN used in Task 2 has failed at class prediction. This is not unexpected. We introduced CNNs to overcome the limitations of fully connected neural networks. In adding the pooling operation we have rendered the architecture closer to a fully connected neural network than to a CNN. We have lost the locality and equivariance that the CNN inherits from the use of convolutions.

This challenge looks insurmountable, because we have an inherent mismatch between the nature of the input – a time signal – and the nature of the output – a class. At some point we need to abandon the time domain. The solution to this challenge is to reduce the time dimension while retaining the structure of time. This is accomplished by pooling, which we explain in the next section.

3 Pooling

Pooling operators reduce the dimension of the input signal while retaining the structure of time. They do so by introducing a reduction factor Δ, dividing the time line in windows of width Δ, and extracting information from every window (Figure 4).

The simplest form of pooling is sampling. Let $\mathbf{w} = [w_0; w_1; \ldots; w_{N-1}]$ be a signal with N components. We define the sampled signal as the signal $\mathbf{x} = [x_0; x_1; \ldots; x_{N_s}]$ in which the component x_m is given by

$$x_m \;=\; w_{m\Delta}\;. \tag{14}$$

Thus, the sampled signal $\mathbf{x}$ copies one out of Δ entries of $\mathbf{w}$ and ignores the rest. The rationale for ignoring entries is that the values of entries in a window, say $w_{m\Delta}$ and $w_{m\Delta+1}$, are not very different and we loose little information by discarding some of them.

In general, computing some aggregate summary of the entries in a pooling window is more effective than plain sampling. Average pooling replaces the entries in each pooling window by their average,

$$x_m \;=\; \frac{1}{\Delta}\Big(w_{m\Delta} + w_{m\Delta+1} + \ldots + w_{m\Delta+(\Delta-1)} \Big)\;. \tag{15}$$

Another common approach to summarizing entries in a pooling window

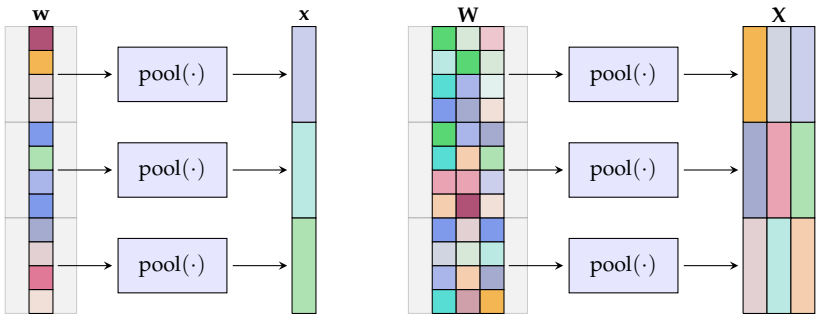

Figure 4. Pooling. We introduce a reduction factor Δ, divide time in windows of width Δ, and summarize information from each window. Pooling yields signals of smaller dimension that retain the locality and shift equivariance of time.

is to compute the maximum value,

$$x_m = \max\left(w_{m\Delta}, w_{m\Delta+1}, \ldots, w_{m\Delta+(\Delta-1)}\right). \tag{16}$$

This is called max pooling. Max pooling is more common than average pooling and both are more common than sampling. In any case, pooling is effective when the signal $\mathbf{w}$ does not change much within the pooling window. When this is the case, sampling, average pooling, and max pooling all produce similar summary signals $\mathbf{x}$.

The pooling operations in (14)-(16) apply to individual vectors. In CNNs we want to pool multiple features. This is done by pooling each feature individually. Given the matrix feature $\mathbf{W} = [\mathbf{w}^1; \mathbf{w}^2; \ldots; \mathbf{w}^F]$ the pooled matrix feature $\mathbf{X} = [\mathbf{x}^1; \mathbf{x}^2; \ldots; \mathbf{x}^F]$ contains the same number of features F and is such that each of the individual vector features $\mathbf{x}^f$ is pooled separately. Thus, Component m of Feature f is given by

$$x_m^f = \text{pool}\left(w_{k\Delta}^f, w_{k\Delta+1}^f, \ldots, w_{k\Delta+(\Delta-1)}^f\right), \tag{17}$$

where the operation $\text{pool}(\cdot)$ stands in for either sampling [cf. (14)], average pooling [cf. (15)], or max pooling [cf. (16)].

3.1 Locality and Equivariance

Pooling operators retain the locality and equivariance of time. The pooled signal $\mathbf{x}$ is also a time signal except that its components are more spaced out. If we recall the definition of the sampling time in Lab 2A, the signal $\mathbf{w}$ has entries spaced by the sampling time T_s and the signal $\mathbf{x}$ has entries spaced by ΔT_s. This is important because it means that the pooled signal $\mathbf{x}$ can be processed with a convolutional filter. In a layered architecture, this fact implies that signals can be pooled at Layer $\ell - 1$ and processed with a convolutional filter at Layer ℓ (Section 4).

Although somewhat obvious it is worth remarking that the summary signal contains a smaller number of entries. If N is a multiple of Δ, the number of entries of $\mathbf{x}$ is $N_s = N/\Delta$. Otherwise, the number of entries of $\mathbf{x}$ is given by the integer division $N_s = N \div K + 1$.

This is important in a classification task because it is almost always the case that the number of classes C is much smaller than the number of entries in the input signal $\mathbf{X}$. Thus, the use of pooling at intermediate layers ends up reducing the complexity of the readout layer. This is a eureka moment. Fully connected neural networks do not work for large dimensional signals but they *do* work for low dimensional signals. We can then use pooling to progressively reduce the dimension of the input signal. All the while we retain the structure of time and the ability of processing with convolutional layers. Once we reach a point at which signals are of sufficiently low complexity we can train an effective readout layer. This is the CNN architecture used in classification tasks that we introduce in the following section.

4 CNNs for Classification Tasks

Combining pooling and readout yields the CNN architecture that is used for classification tasks (Figure 5). In this architecture we have $L - 1$ convolutional layers and one readout layer. Each of the convolutional layers involves a convolutional filter, a pointwise nonlinearity, and a pooling

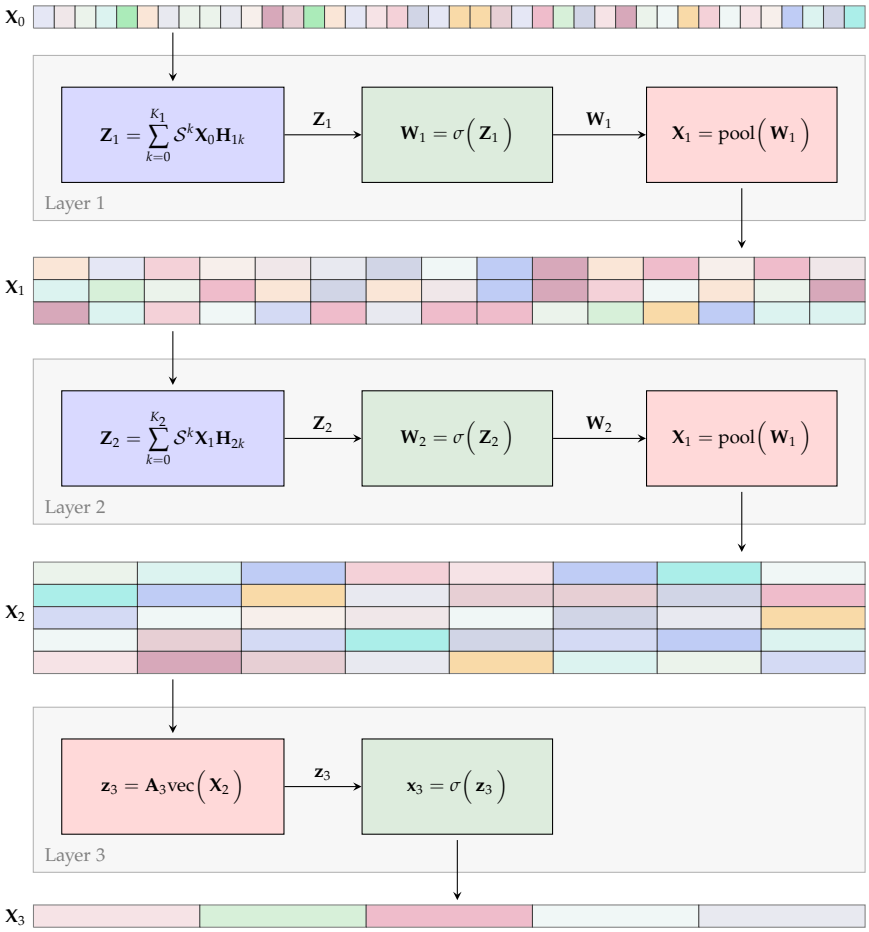

Figure 5. CNN with two convolutional layers and one readout layer. The convolutional layers include pooling operations that progressively reduce the number of components of the outputs of each layer. The readout layer maps the time signal at its input to a vector of classification scores at the output.

operation,

$$\mathbf{Z}_\ell = \sum_{k=0}^{K_1} \mathcal{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}, \qquad \mathbf{W}_\ell = \sigma\Big(\mathbf{Z}_\ell \Big), \qquad \mathbf{X}_\ell = \text{pool}\Big(\mathbf{W}_\ell \Big). \tag{18}$$

The multiple feature layer input $\mathbf{X}_{\ell-1}$ is processed with a MIMO convolutional filter with coefficients $\mathbf{H}_{\ell k}$. The filter output $\mathbf{Z}_\ell$ is processed with the pointwise nonlinearity $\sigma(\cdot)$. The output of this operation is a matrix of features $\mathbf{W}_\ell$ whose dimension is reduced by the pooling operator $\text{pool}(\cdot)$ to yield the Layer ℓ output $\mathbf{X}_\ell$.

The input $\mathbf{X}_{\ell-1}$ is of dimension $N_{\ell-1} \times F_{\ell-1}$. The output $\mathbf{X}_\ell$ is of dimension $N_\ell \times F_\ell$. The change in the length of the features from $N_{\ell-1}$ to N_ℓ is determined by the pooling operator $\text{pool}(\cdot)$. The change in the number of features from $F_{\ell-1}$ to F_ℓ is determined by the filters of the MIMO filter. The coefficients $\mathbf{H}_{\ell k}$ are of dimension $F_{\ell-1} \times F_\ell$.

We follow the $L-1$ convolutional layers with a readout layer,

$$\mathbf{Z}_L = \mathbf{A}_L \text{vec}\Big(\mathbf{X}_{L-1} \Big), \qquad \mathbf{x}_L = \sigma\Big(\mathbf{x}_L \Big). \tag{19}$$

The input to Layer L is $\text{vec}(\mathbf{X}_{L-1})$. This is the output of Layer $L-1$ rearranged in vector form. This input is multiplied by the matrix $\mathbf{A}_L$ to produce the intermediate feature $\mathbf{Z}_L$. This is then passed through the pointwise nonlinearity $\sigma(\cdot)$ to produce the layer's output $\mathbf{X}_L$.

The input vector $\text{vec}(\mathbf{X}_{L-1})$ is of dimension $N_{L-1} \times F_{L-1}$. The output vector is of dimension N_L. This implies that the matrix $\mathbf{A}_L$ has $N_{L-1} \times F_{L-1}$ columns and N_L rows.

The output of Layer L is also the output of the CNN. We write this for reference as

$$\mathbf{\Phi}(\mathbf{x}; \mathcal{H}) = \mathbf{x}_L, \tag{20}$$

where the tensor $\mathcal{H}$ contains the filters of the convolutional filters used in Layers 1 through $L-1$ and the coefficients $\mathbf{A}_L$ of the fully connected layer. The output $\mathbf{\Phi}(\mathbf{x}; \mathcal{H})$ can be interpreted as a vector of classification scores and used in the classification empirical risk minimization problem shown in (11).

A CNN with pooling and readout resembles a CNN with readout only. Both include $L-1$ convolutional layers followed by a fully connected

readout layer. The difference is that in the CNN with pooling the convolutional layers process signals of progressively lower dimension. Thus, the readout matrix $\mathbf{A}_L$ is of lower dimension too. This is expected to avoid the failure we observed in Task 2.

Task 3 Modify the CNN of Task 1 to incorporate pooling. This can be done by modifying the method that implements convolutional layers to incorporate the pooling operation.

As in any CNN the initialization parameters include the number of layers L along with vectors $[K_1, \ldots, K_{L-1}]$ and $[F_0, F_1, \ldots, F_{L-1}]$. These vectors contain the number of taps K_ℓ of the filters used at each layer and the number of features F_ℓ at the output of each layer.

Since we are incorporating pooling the initialization parameters must also include the vecor $[N_0, N_1, \ldots, N_L]$ containing the dimension N_ℓ of the features at the output of each layer. Notice that N_0 matches the dimension of the input signal and $N_L = C$ matches the number of classes.

The forward method of this class takes a matrix $\mathbf{X}$ of dimension $N_0 \times F_0$ as an input and returns the output $\mathbf{\Phi}(\mathbf{x}; \mathcal{H})$ of a CNN with $L-1$ convolutional layers and a readout layer. The dimension of this output is $N_L \times 1$.

Use relu nonlinearities in all layers. Use max pooling in all convolutional layers where pooling is implemented. ■

Task 4 Load the same data used in Task 2. Use the class of Task 3 to train a CNN with a readout layer to classify the audio signals into the different possible spoken digits. The CNN should have the following parameters: 2 layers, number of channels per layer: 8, kernel size (number of taps) at each layer: 80 and 3 respectively, learning rate: 0.05, pooling layers: max pooling.

Evaluate the training cross entropy loss and the test accuracy in terms of the relative number of incorrect classifications.

5 Report

Do not take much time to prepare a lab report. We do not want you to report your code and we don't want you to report your work. Just give us answers to questions we ask. Specifically give us the following:

Question	**Report deliverable**
Task 1	Do not report
Task 2	Report training cross entropy loss
Task 2	Report percentage of incorrect classifications
Task 3	Do not report
Task 4	Report training cross entropy loss
Task 4	Report percentage of incorrect classifications

We will check that your answers are correct. If they are not, we will get back to you and ask you to correct them. As long as you submit responses, you get an A for the assignment. It counts for 10% of your lab grade.

Evaluate the training cross entropy loss and the test accuracy in terms of the relative number of incorrect classifications.