Lab 1C: Neural Networks

Ignacio Hounie, Javier Porras and Alejandro Ribeiro[*]

September 2, 2024

1 Neural Networks

Neural Networks are information processing architectures made up of a composition of layers, each of which is itself the composition of a linear transformation with a pointwise nonlinearity.

For a network with an arbitrary number of layers L we define the input output relationship through the recursion

$$\mathbf{x}_0 = \mathbf{x}, \qquad \mathbf{x}_\ell = \sigma\Big(\mathbf{z}_\ell\Big) = \sigma\Big(\mathbf{A}_\ell \mathbf{x}_{\ell-1}\Big), \qquad \mathbf{x}_L = \mathbf{\Phi}(\mathbf{x}; \mathcal{A}). \tag{1}$$

In this recursion the output of Layer $\ell - 1$ produces the output of Layer ℓ as $\mathbf{x}_\ell = \sigma(\mathbf{A}_\ell \mathbf{x}_{\ell-1})$. This operation is a composition of the linear map $\mathbf{A}_\ell \mathbf{x}_{\ell-1}$ with a pointwise nonlinear function σ. This is a function that acts separately on each individual components; see Section 1.1. To complete the recursion we redefine the input $\mathbf{x}$ as the output of Layer 0, $\mathbf{x}_0 = \mathbf{x}$. The output of the neural network is the output of layer L, $\mathbf{x}_L = \mathbf{\Phi}(\mathbf{x}; \mathcal{A})$. In this notation $\mathcal{A}$ is the tensor $\mathcal{A} := [\mathbf{A}_1, \ldots, \mathbf{A}_L]$ that groups the L matrices that are used at each of the L layers.

A neural network with three layers is depicted in Figure 1. The input to the neural network is a vector, or signal, $\mathbf{x}$ which we will also denote as $\mathbf{x}_0 = \mathbf{x}$. This signal is passed to the first layer of the neural network where it is processed with a linear transformation defined by the matrix

[*]In alphabetical order.

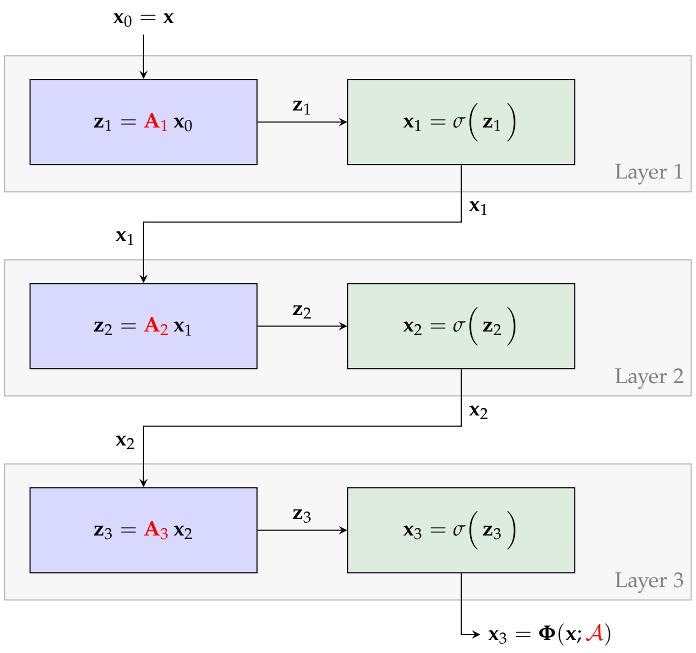
$\mathbf{x}_0 = \mathbf{x}$
$\mathbf{z}_1 = \mathbf{A}_1\, \mathbf{x}_0$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\big(\mathbf{z}_1\big)$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \mathbf{A}_2\, \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\big(\mathbf{z}_2\big)$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \mathbf{A}_3\, \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\big(\mathbf{z}_3\big)$
Layer 3
$\mathbf{x}_3 = \mathbf{\Phi}(\mathbf{x}; \mathcal{A})$

Figure 1. A neural network with three layers. The neural network is a composition of layers, each of which is itself the composition of a linear transformation with a pointwise nonlinearity. The output of the neural network depends on the tensor $\mathcal{A} := [\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3]$. This is the parameter that we determine during training.

$\mathbf{A}_1$. This produces the intermediate signal $\mathbf{z}_1 = \mathbf{A}_1\mathbf{x}$ which is then passed through a pointwise nonlinearity σ to produce the first layer output

$$\mathbf{x}_1 = \sigma\Big(\,\mathbf{z}_1\,\Big) = \sigma\Big(\,\mathbf{A}_1\mathbf{x}_0\,\Big). \tag{2}$$

The output $\mathbf{x}_1$ of Layer 1 becomes an input to Layer 2. In this layer, the signal is processed by a linear transformation defined by the matrix $\mathbf{A}_2$ and a pointwise nonlinearity to produce the Layer 2 output

$$\mathbf{x}_2 = \sigma\Big(\,\mathbf{z}_2\,\Big) = \sigma\Big(\,\mathbf{A}_2\mathbf{x}_1\,\Big). \tag{3}$$

This output becomes now an input to the third layer where an analogous composition of a linear map with a pointwise nonlinearity produces the output of the neural network:

$$\mathbf{\Phi}(\mathbf{x}; \mathcal{A}) = \mathbf{x}_3 = \sigma\Big(\,\mathbf{z}_3\,\Big) = \sigma\Big(\,\mathbf{A}_3\mathbf{x}_2\,\Big). \tag{4}$$

In (4) we use $\mathbf{\Phi}(\mathbf{x}; \mathcal{A})$ to denote the neural network output. This output is a function of the input $\mathbf{x}$ and a function of the matrices $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$. These three matrices are grouped in the tensor $\mathcal{A} := [\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3]$. This is the parameter that we must determine during training.

This training process entails solutions of an empirical risk minimization problem (ERM). For clarity, recall that we are given data pairs $(\mathbf{x}_i, \mathbf{y}_i)$ in which $\mathbf{x}_i$ is the input to a system of interest and $\mathbf{y}_i$ is the respective observed output. In the ERM problem we evaluate the neural network outputs $\hat{\mathbf{y}}_i := \mathbf{\Phi}(\mathbf{x}_i; \mathcal{A})$ and compare them with the corresponding system outputs $\mathbf{y}_i$ using a given loss functions $\ell(\mathbf{y}, \hat{\mathbf{y}})$. The ERM objective is the average of these individual losses. Thus, training a neural network implies finding the tensor $\mathcal{A}^*$ that solves the ERM problem,

$$\mathcal{A}^* \;=\; \operatorname*{argmin}_{\mathcal{A}} \frac{1}{N} \sum_{i=1}^{N} \ell\Big(\mathbf{y}_i, \mathbf{\Phi}(\mathbf{x}_i; \mathcal{A})\Big) \tag{5}$$

Notice that in (5) we must have that the dimensions of the system outputs $\mathbf{y}_i$ and the dimensions of the neural network outputs $\hat{\mathbf{y}}_i := \mathbf{\Phi}(\mathbf{x}_i; \mathcal{A})$ must be the same. See Section 1.2 for details on the specification of the search space of (5)

The historical development of neural networks drew inspiration from early models of biological neurons. Given our current understanding of biological neurons and their interactions it is untenable to say that the architecture in Figure 1 mimics biological brains.

A better motivation is that neural networks are nonlinear maps that retain some of the advantages of linear maps. This is because nonlinearities are restricted to act on linear combinations of individual components. For example, if x_{0p} and x_{0q} are components of the input vector $\mathbf{x}_0 = \mathbf{x}$, the nonlinear operation $\sigma(x_{0p} + x_{0q})$ may happen in Layer 1. However, the nonlinear operation $x_{0p}x_{0q}$ cannot happen.

1.1 Pointwise nonlinearities

That the nonlinear function σ in (1) is pointwise means that it acts on each component of the input separately. This means that if we write the intermediate signal as $\mathbf{z}_\ell = [z_{\ell 1}; z_{\ell 2}; \ldots; z_{\ell p}]$, the nonlinearity σ produces

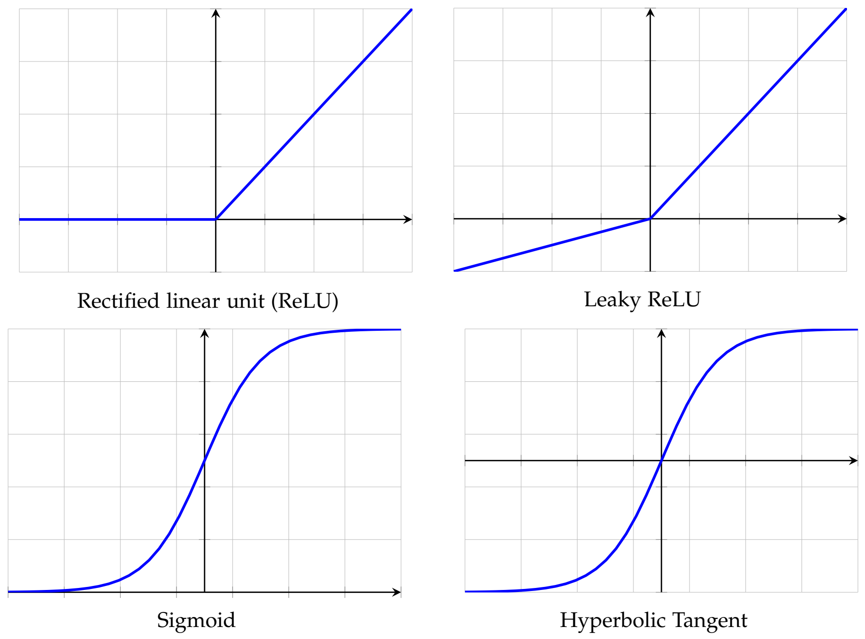

Figure 2. Activation functions.

as an output the vector

$$\sigma\Big(\mathbf{z}_\ell\Big) = \sigma\Big(\,[z_{\ell 1}; z_{\ell 2}; \ldots; z_{\ell p}]\,\Big) = \Big[\,\sigma(z_{\ell 1}); \sigma(z_{\ell 2}); \ldots; \sigma(z_{\ell p})\,\Big]. \tag{6}$$

The most popular choice for a pointwise nonlinearity is a rectified linear unit (ReLU). A ReLU is a simple function that just zeros the entry when it is negative (see Figure 2),

$$\sigma(z) = \max(0, z). \tag{7}$$

In this expression we can think that the output $\sigma(z)$ is activated when $z \geq 0$ and not activated when $z < 0$. For this reason pointwise nonlinearities are also called activation functions, whether we use a ReLU or not.

A variation of a ReLU nonlinearity is a leaky ReLU. Given a constant $0 < \alpha < 1$, a leaky ReLU is defined as

$$\sigma(z) = \max(\alpha z, z). \tag{8}$$

Given that $0 < \alpha < 1$ the output of the leaky ReLU is $\sigma(z) = z$ when $z \geq 0$ and $\sigma(z) = \alpha z$ when $z < 0$. The idea is to use a constant $\alpha \ll 1$ so that negative outputs are attenuated significantly. The advantage of a leaky ReLU is that it does not eliminate negative outputs entirely. This is particularly useful during training.

Another common choice of activation function is the logistic or sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{9}$$

This function approaches 0 when z is a large negative number and approaches 1 when z is a large positive number. When z has a small absolute value the sigmoid is approximately linear. Sigmoids are useful when we want outputs that range between 0 and 1.

The final activation function we define is the hyperbolic function

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{10}$$

This function is similar in shape to the sigmoid except that its range is between plus and minus 1. An interesting interpretation of the hyperbolic function is that it tapers large values. When the argument z has small absolute value the output of a hyperbolic function is close to its input, $\sigma(z) \approx z$. When the argument z is large, the output is saturated to plus or minus 1.

Notice that the ReLU and leaky ReLU functions in (7) and (8) are not differentiable at $z = 0$ whereas the sigmoid and hyperbolic tangent in (9) and (10) are differentiable everywhere. This is an important distinction but it has little practical significance.

Further observe that all these four activation functions are such that the values at the output have less variability than the values at the input. ReLUs and leaky ReLUs maintain positive values but eliminate or drastically reduce negative values. Sigmoids and hyperbolic tangents squash the range of the input to $(0, 1)$ or $(-1, 1)$. Reducing variability improves the stability of information processing with neural networks relative to information processing with linear functions.

1.2 Neural network specification

The input to each layer of a neural network is a vector and the output is another vector. We say that we specify a neural network when we specify the number of entries of each of these vectors. Thus, if $p_0 = p$ is the number of components of the input $\mathbf{x}_0$ and p_ℓ is the number of components at the output $\mathbf{x}_\ell$ of layer ℓ of a neural network with L layers, the neural network is specified by the numbers L and $p_0, p_1, \ldots, p_L$. Of these numbers p_0 and p_L are determined by the data because p_0 is the dimension of the input $\mathbf{x}$ and p_L matches the dimension of the output $\mathbf{y}$.

To complete the specification of a neural network we must also specify the nonlinearities that are used at each layer. In most cases, the same activation function is used in all layers.

Task 1 Define an object to represent a neural network to process scholastic achievement data. The inputs are high school GPA and SAT scores. The output is an estimate of the Penn GPA. This neural network has 2 layers and the output of Layer 1 has dimension $p_1 = 20$. The neural network uses ReLU nonlinearities in both layers.

Endow this neural network object with a method that takes as input the vector $\mathbf{x} = [\text{HS GPA}; \text{SAT}]$ with high school GPA and SAT scores and produces estimates of Penn GPAs. ■

Notice that while we say that we are specifying the neural network we are, in reality, specifying a *family* or a *class* of neural networks. An actual neural network is a conrete choice of the tensor $\mathcal{A} := [\mathbf{A}_1, \ldots, \mathbf{A}_L]$. The intent of a neural network specification like the one we give in Task 1 is to specify the search space for the ERM problem in (5).

1.3 Neurons and Weights

It is common practice to describe neural networks in terms of neurons and weights instead of input vectors and linear maps as we do in Figure 1. In this alternative nomenclature the components of the output of each layer are called neurons and the entries of the matrices that define the

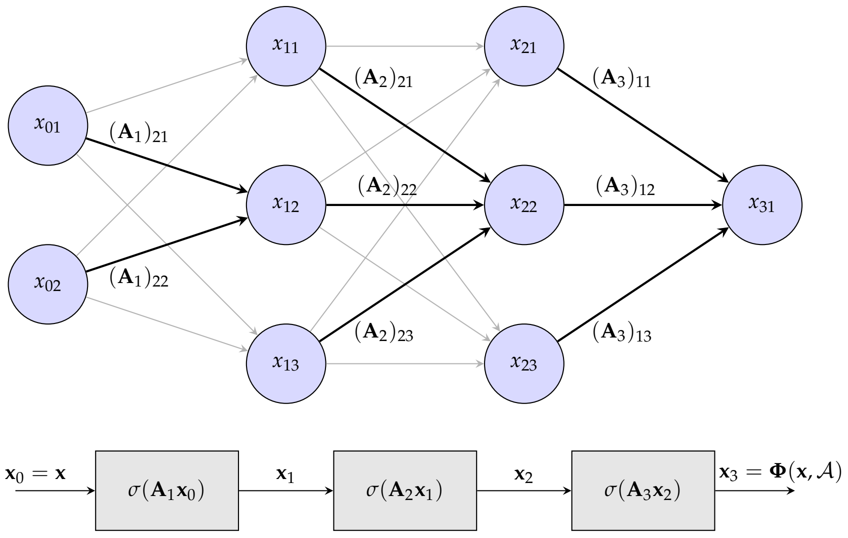

Figure 3. A neural network with three layers. Neural networks are often represented as collections of neurons and weights. Neurons represent the input and output vectors of each layer and weights represent the entries of the matrices that are contained wihtin each layer.

input-output relationship of each layer are called weights. A scheme is given in Figure 3.

At Layer ℓ the components of the Layer output $\mathbf{x}_\ell$ are the layer's neurons. If this vector is of dimension p_ℓ we say that the layer contains p_ℓ neurons. This language is extended to the input vector, whose entries are termed input neurons. In Figure 3 we have 2 input neurons, 3 neurons in Layers 1 and 2, and 1 neuron in Layer 3. Since Layer 3 is the output of the neural network we equivalently say that we have 1 output neuron. The neurons that are neither input nor output neurons are sometimes called intermediate neurons or hidden neurons.

In Task 2, the neural network has two layers. Layer 1 has $p_1 = 20$ neurons and Layer 2 has $p_2 = 1$ neuron. The latter is because the outputs we are trying to predict are scalar numbers. The input dimension is $p_0 = 2$. We therefore say that this neural network has 2 input neurons, 1 output neuron, and 20 hidden neurons in a single layer.

As it follows from (1), the input-output relationship of Layer ℓ is determined by the matrix $\mathbf{A}_\ell$. This matrix must have $\ell-1$ columns and ℓ rows because it processes the vector $\mathbf{x}_{\ell-1}$ – which has $\ell-1$ neurons – to produce the vector $\mathbf{x}_\ell$ – which has ℓ neurons. The entries of matrix $\mathbf{A}_\ell$ are called weights and we say that entry $(\mathbf{A}_\ell)_{pq}$ is the weight that connects neuron q at the input of Layer ℓ with neuron p at the output of Layer ℓ.

2 Train and Test Sets

The minimization of empirical risks carries a subtle challenge. To see which one, consider an ERM problem in which we do not choose a parameterization. In this case we are simply seeking an artificial intelligence $\mathbf{\Phi}(\mathbf{x})$ that solves the following problem

$$\mathbf{\Phi}^* \;=\; \operatorname*{argmin}_{\mathbf{\Phi}} \frac{1}{N}\sum_{i=1}^{N} \ell\Big(\mathbf{y}_i, \mathbf{\Phi}(\mathbf{x}_i)\Big)\,. \tag{11}$$

This problem is the opposite of challenging. We just choose the map $\mathbf{\Phi}(\mathbf{x}_i) = y_i$ and select $\mathbf{\Phi}(\mathbf{x})$ at random for any other $\mathbf{x}$. This solution is as easy as it is nonsensical. It does not solve the artificial intelligence problem of finding a function $\mathbf{\Phi}(\mathbf{x})$ that can make predictions about the future. It memorizes the past perfectly but it says nothing about future inputs that are not exact matches of some past inputs.

Given that it is nonsensical there is no point in working with (11). And, indeed, we are not attempting to solve (11). We are attempting to solve the ERM problem in (5) in which predictions $\hat{\mathbf{y}} = \mathbf{\Phi}(\mathbf{x}_i;\mathcal{A})$ are outputs of a family of neural networks that follow some specification – as discussed in Section 1.2. Nevertheless, there is the mystifying fact that from the perspective of empirical risk a function $\mathbf{\Phi}^*$ that solves (11) is at least as good, and in all likelihood better, than a function $\mathbf{\Phi}(\mathbf{x}_i;\mathcal{A}^*)$ that solves (5). Indeed, since the empirical risk of $\mathbf{\Phi}^*$ is null and the empirical risk is nonnegative, we must have,

$$0 \;=\frac{1}{N}\sum_{i=1}^{N} \ell\Big(\mathbf{y}_i, \mathbf{\Phi}^*(\mathbf{x}_i)\Big) \;\leq\; \frac{1}{N}\sum_{i=1}^{N} \ell\Big(\mathbf{y}_i, \mathbf{\Phi}(\mathbf{x}_i;\mathcal{A}^*)\Big). \tag{12}$$

The answer to this riddle is that the merit of an artificial intelligence is not its empirical risk but the risk it attains during operation.

Suppose then that we observe $\tilde{N}$ points $(\tilde{\mathbf{x}}_j, \tilde{\mathbf{y}}_j)$ during operation of the neural network. The true metric of performance of the neural network is its ability to make predictions $\mathbf{\Phi}(\tilde{\mathbf{x}}_j; \mathcal{A})$ that are close to observed outcomes. This is measured but the operational risk

$$\tilde{r}(\mathcal{A}) \;=\; \frac{1}{\tilde{N}} \sum_{j=1}^{\tilde{N}} \ell\Big(\tilde{\mathbf{y}}_j, \mathbf{\Phi}(\tilde{\mathbf{x}}_j; \mathcal{A})\Big) \,. \tag{13}$$

This solves the riddle of (12) because it is possible that the operational risk of the neural network is better than the operational risk of the function $\mathbf{\Phi}^*$. This is not just possible but does happen in practice. The reasons why this may happen or not are the subject matter of statistical learning theory. We will not study this here.

Observe that the operational risk becomes available a posteriori; after the neural network goes into service and starts making predictions. To gain a handle on operational risk before service we divide available data into two sets. A training set with the N samples $(\mathbf{x}_i, \mathbf{y}_i)$ that we use for solving (5) and a *test* set with $\tilde{N}$ samples $(\tilde{\mathbf{x}}_i, \tilde{\mathbf{y}}_i)$ that we can use to estimate the operational risk. It is important that the test set is *not* used in training and it is important that the test set is representative of the data as a whole. In this context the operational risk is also called to test error.

Task 2 Divide the data of Lab 1A into a test set with $\tilde{N} = 100$ samples and use the remaining samples as a training set. Use it to train the neural network of Task 1. Use a stepsize of 0.01, a batch size of 64 and train for 200 epochs. Evaluate the training error and the test error. They will be different. Comment. ■

Task 3 We will incorporate the gender data that we have so far ignored. To that end encode gender as Female = 1 and Male = −1. Redefine the object in Task 3 so that it can now process this information. In this neural network we use a single intermediate layer with 20 hidden neurons. ■

Task 4 Divide the data of Lab 1A into a test set with $\tilde{N} = 100$ samples and use the remaining samples as a training set. Use it to train the neural network of Task 3. Use a step size of 0.01, a batch size of 64 and train for 200 epochs. Evaluate the training error and the test error.

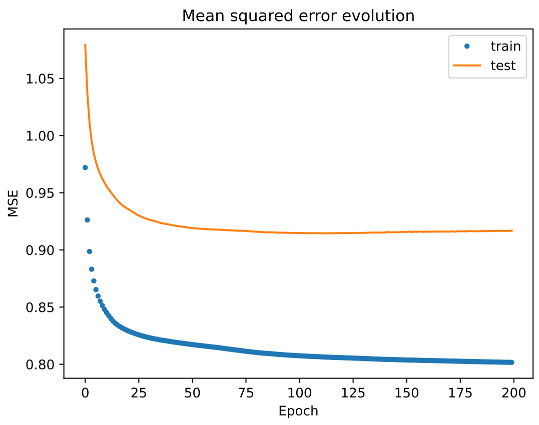

Figure 4. MSE of Neural Network predictions for the training testing sets as a function of the training epoch.

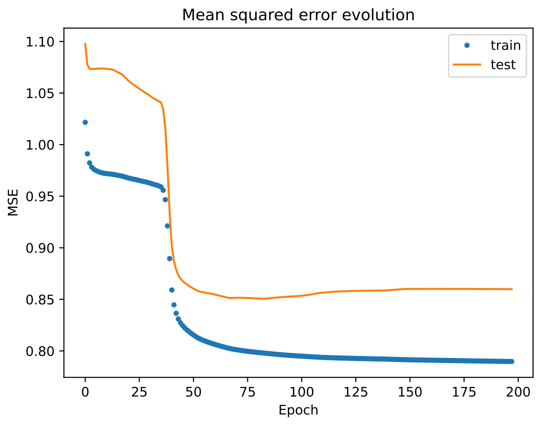

Figure 5. Neural network using Gender Data. MSE of Neural Network predictions for the training testing sets as a function of the training epoch.

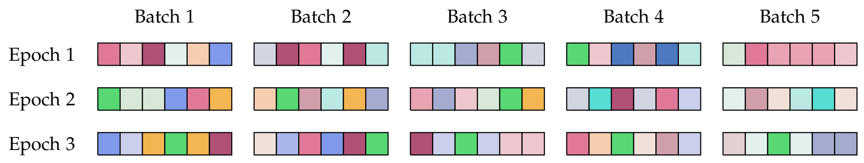

Figure 6. Sampling without Replacement. The training set is divided into N/B batches of size B containing *different* samples. This tends to be less erratic than the random sampling assumed in (14) because all samples are used during training and no sample is used more than once. If we need to run more than N/B iterations we run several epochs in which the training set is sampled in the same way. Samples are reshuffled from epoch to epoch.

Task 5 Given that we have decided to use gender information in our decisions, it is fair that we are asked what effect this choice has. Evaluate the average predicted GPA for Female and Male applicants and observe that data can be perilous. Or not. Comment. ■

3 Sampling Without Replacement

In Tasks 2 and 4 we trained our neural networks using the stochastic gradient descent (SGD) algorithm we introduced in Lab 1B. As a reminder, this algorithm utilizes the stochastic gradients,

$$\hat{g}(w) \;=\; \frac{1}{B}\sum_{i=1}^{B}\frac{\partial}{\partial w}\ell\Big(y_i, \Phi(x_i; w)\Big) \;, \tag{14}$$

which are averages of gradients associated with individual data pairs over a batch of $B \ll N$ samples. These stochastic gradients form the basis of the SGD recursion,

$$w(k+1) = w(k) - \epsilon\hat{g}(w(k)), \tag{15}$$

In (14), the samples in the batch are chosen independently at random at each SGD iteration. While this is justifiable in theory it tends to perform erratically in practice because there is a significant variability in the samples that are drawn in different runs. Indeed, there is a significant likelihood that some samples are never chosen in any batch and a significant likelihood that some samples are drawn in several batches.

We overcome this erratic behavior by dividing the training set into N/B batches of size B containing *different* samples. We call this choice of batches *sampling without replacement.* This nomenclature indicates that samples in different batches are different and that all samples are part of one batch.

This process is sketched in Figure 6. The training set in the illustration contains $N = 30$ samples that we divide into $N/B = 5$ batches of $B = 6$ samples each. If we need to run more than $N/B = 5$ stochastic gradient descent iterations we repeat the process of dividing the training set into batches. To avoid having batches with the exact same set of samples, the training set is reshuffled before it is divided into batches. Each of the new set of batches that are created after reshuffling is called an epoch.

Task 6 Reimplement Task 2 using sampling without replacement. This is a task that is easier to implement using functionalities built in Pytorch. We recommend that you look at the posted solution.

Compare the test loss of this trained neural network with the test loss of the neural network trained in Task 2. They should have similar test losses in this case. This is because the dataset has limited complexity but it is not true in general. In upcoming labs we will use sampling without replacement unless we say otherwise. ■

4 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	Code for forward method
Task 2	Train error
Task 2	Test error
Task 2	Paragraph discussing differences between train and test risks
Task 3	Do not report
Task 4	Train error
Task 4	Test error
Task 5	Test errors for Male and Female applicants
Task 5	Comment on the perils of data
Task 6	Do not report

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.