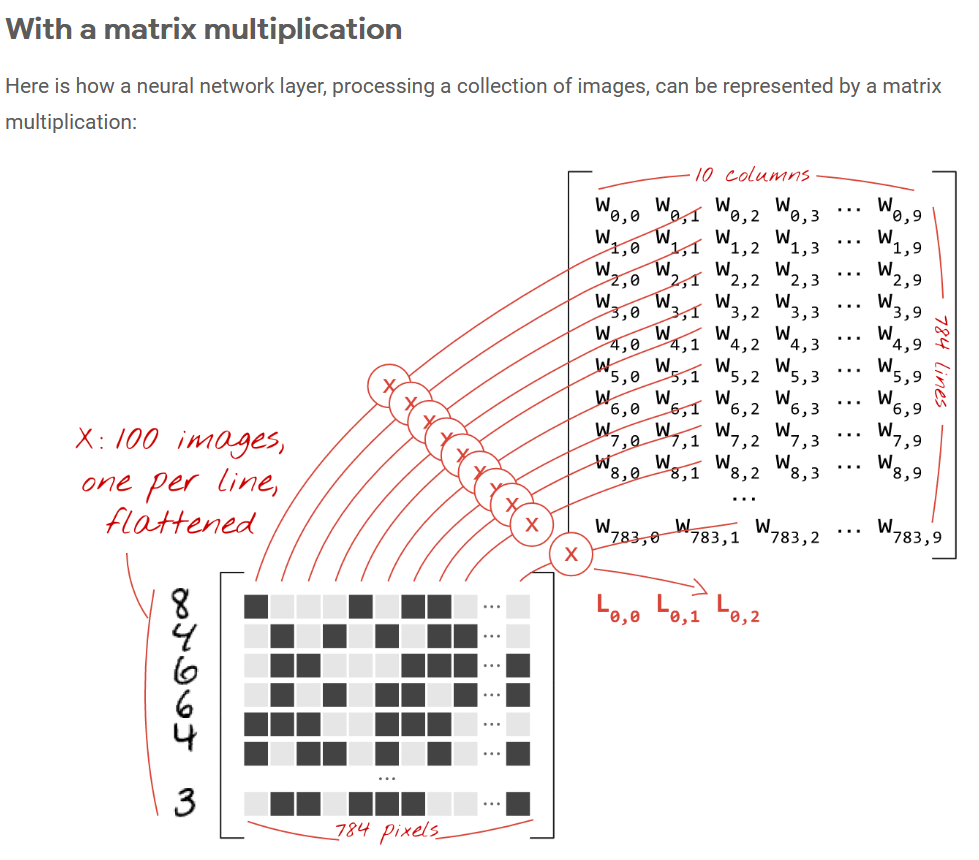
**Machine Learning and AI – Homework 4 Due May 24th**

**Each question is worth 10 points.**

**Question 1.**

**On the slide 4 “Neural Networks 101” shown on the link and image below,** [**https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist#3**](https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist#3)

****

**Please clarify why the weights matrix W has dimensions 10x784; explain what each dimension corresponds to and how it is used to represent the summations / multiplications in the logit.**

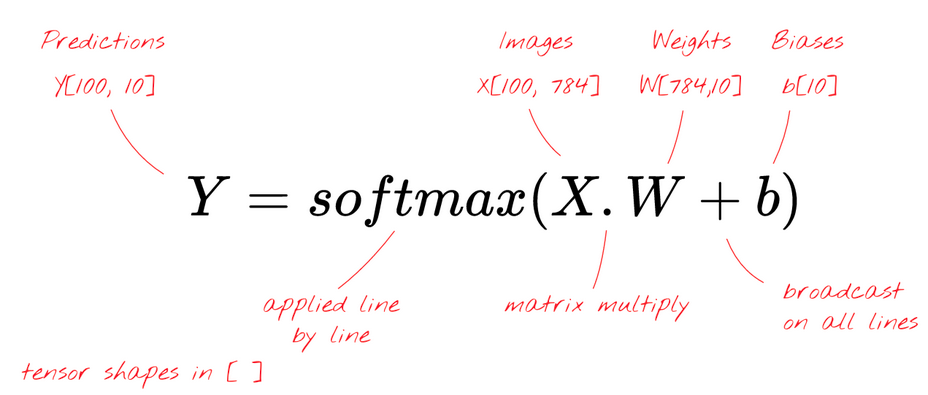
In the MNIST dataset, the numbers written by hand are grayscale images with 28x28 pixels. The easiest way to put them into groups is to feed the 28x28, or 784 pixels, into a 1-layer neural network. The picture above shows a one-layer neural network with 10 output neurons, because we want to divide numbers into 10 groups (0 to 9).

In a neural network, each "neuron" does a weighted sum of all its inputs, adds a constant called "bias," and then sends the result through a "activation function" that is not linear. The "weights" and "biases" are parameters that will be set by training. They start out with values that are chosen at random.

We figure out the weighted sum of all the pixels in the first image by using the first column of weights in the weights matrix W. This amount is the same as the first neuron. We do the same thing for the second neuron using the second column of weights, and so on up to the tenth neuron. Then, we can do the same thing for the other 99 images. If we call the group of 100 images a "matrix" and call it "X," then all of the weighted sums for our 10 neurons based on 100 images are just "X.W," which is a matrix multiplication. Now, each neuron must add its own bias (a constant). Because there are 10 neurons, there are also 10 bias constants. We will call this 10 value vector b. It needs to be added to each line of the matrix that was already made. We will write this with a simple plus sign using a trick called "broadcasting."

**Question 2.**

**On the same slide 4 “Neural Networks 101”, you see the matrices from Question 1 in the formula**

****

**What is the “broadcasting” used for in this matrix multiplication and how does it work ? Why are the predictions matrix 100x10 ?**

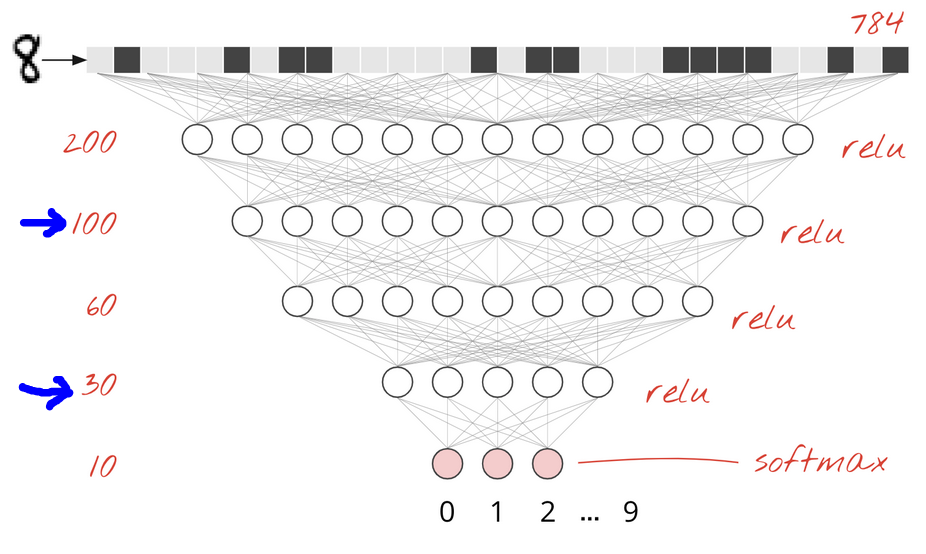
"Broadcasting" is a common trick in Python and numpy, its library for scientific computing. It changes the way that normal operations work on matrices with different sizes. "Broadcasting add" means "if you're trying to add two matrices but can't because their sizes don't match, try to make as many copies of the smaller one as you need to make it work." We finally use an activation function, like "softmax" (which is explained below), and get a formula that describes a 1-layer neural network that was trained on 100 images.

Because we want to divide handwritten digits into 10 groups, the last layer of our neural network has 10 neurons (0,..9). It should give back 10 numbers between 0 and 1 that show how likely it is that this digit will be a 0, 1, 2, and so on. On the last layer, we will use a function called "softmax" to do this. To use softmax on a vector, you take the exponential of each element and then normalize the vector, usually by dividing it by its "L1" norm, which is the sum of absolute values, so that the normalized values add up to 1 and can be thought of as probabilities.

This makes a 100x10 matrix because each iteration has a batch size of 100 and each prediction is spread across the 10 classes, with the largest number showing the most likely number it is predicting.

**Question 3.**

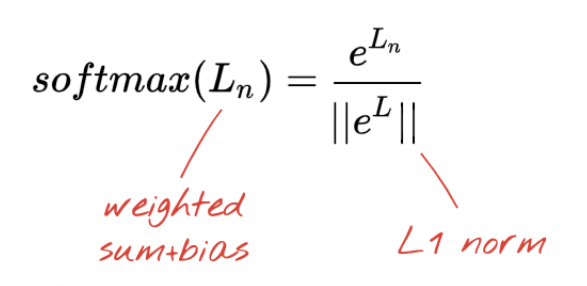
**On the same slide 4 “Neural Networks 101”, see the image below. How many connections we have coming in each neuron in the layers I point with blue arrow below ? Why do we have 10 neurons in the last layer ?**

****

300 and 90 because a dense layer is a layer of neurons where each neuron is connected to all the neurons in the layer below it. We need 10 output neurons because we want to divide numbers into 10 groups (0 to 9).

**Question 4.**

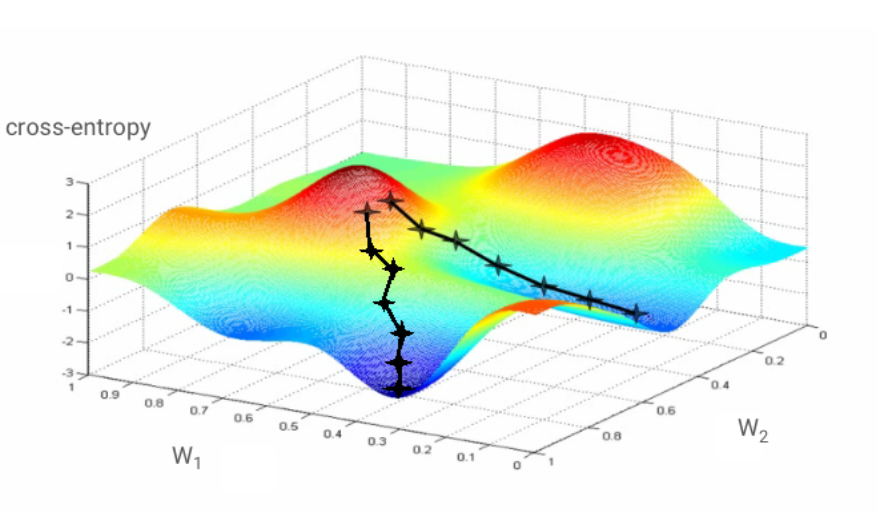
**On the same slide 4 “Neural Networks 101”, what is the output of the softmax function when applied to the logit (the L). How does the output of the softmax compares with previous logit functions like the logistic for example, and what does this output represent ?**

****

Softmax function is an activation function that turns numbers, or logits, into probabilities that add up to 1. The Softmax function gives you a vector that shows how likely each of a list of possible outcomes is to happen. In neural networks, the "sigmoid" was the traditional activation function, but the "relu" was found to have better convergence properties almost everywhere and is now preferred. For a two-class logistic regression, the sigmoid function is used. For a multiclass logistic regression, the softmax function is used. All sigmoid functions have the property that they map the whole number line into a small range, such as between 0 and 1 or -1 and 1. One use of a sigmoid function is to turn a real value into one that can be interpreted as a probability.

**Question 5.**

**On the same slide 4 “Neural Networks 101”, what is cross-entropy ? What do the valleys and mountains in the cross-entropy plot shown below ?**

****

"Cross-entropy distance," as it is called, is the best way to solve classification problems. This will be our "loss" or "error" function. "Training" the neural network means using training images and labels to change the weights and biases so that the cross-entropy loss function is as small as possible. How it works is as follows. Cross-entropy depends on the weights, biases, pixels of the training image, and the known class of the image. Gradient descent is an iterative optimization algorithm of the first order that finds the local minimum of a function that can be changed. It starts with a convex function and changes its parameters over and over to find the local minimum of a given function. Essentially Up is positive, and down is negative. To maximize a function, the algorithm starts at a random point, measures the gradient, moves in the direction of the gradient, and starts over with a new starting point. In the same way, taking small steps in the opposite direction can reduce the function. We figure out the cost function based on its initial values, and over a series of steps, the parameter estimates are changed so that the cost function ends up implying a minimum value.

**Question 6.**

**How do the filters / kernels designed to capture elements of images in CNNs.**

**Could you give an example of a 10x10 filter that is designed to capture vertical lines towards the left edge of the image and, and one filter that is designed to capture semi-circles (the open side of the semi-circle is towards the right side of the image). For helping you to answer the question, please see the additional links posted regarding the CNNs, and the information therein, which we already touched upon in the class.**

In the CNN, a convolutional matrix (also called a filter or kernel) is "slid" across the image and applied at each point. The value that comes out is then used for that pixel in the result. In convolution, a "kernel" is used to pull out "features" from an image. A kernel is a matrix of weights that are multiplied with the input to find relevant features. A filter, on the other hand, is made up of several kernels, each of which is connected to a different channel of the input. Filters always have one more dimension than the kernels they work on.

vertical\_filter = [[1, 0],

[2, 0,

[1, 0]]

plot\_filter(images, vertical\_filter)

When you apply a 5x5x3 filter (a semicircle and a line) to a 32x32x3 image (a circle and a square), you get a 28x28 activation map that shows where the image is most like the filter. When you add a set of filters on top of that (pass it through the 2nd conv layer), you'll get activations that represent higher-level features. These features could be semicircles (a combination of a curve and a straight edge) or squares (combination of several straight edges).

EX: <http://ethen8181.github.io/machine-learning/deep_learning/cnn_image_tensorflow.html>

**Question 7.**

**Define the terms filter / kernel, stride, activation map (or wrongly “layer”) and padding, in Convolutional Neural Networks (CNNs). Let’s say you have a set of images which have mostly vertical, horizontal, diagonal lines and a few circles. How many filters would you design to capture all these elements in the image ?**

**Filter**: Convolution filters are data filters with more than one dimension that are used in the Convolution layer to help pull out specific features from the data being fed in.

**Kernel**: a filter that takes the features out of the images. The kernel is a matrix that moves over the input data, does the dot product with the sub-region of input data, and gets the output as the matrix of dot products.

**Stride**: Stride is a part of convolutional neural networks, which are neural networks that are set up to compress video and image data. Stride is a setting in the neural network's filter that changes how much the image or video moves.

**Activation map**: Activation maps are just a visual representation of these activation numbers at different layers of the network as a given image moves through as a result of different linear algebraic operations.

**Padding**: A term used in convolutional neural networks (CNNs) to describe how many pixels are added to an image when the kernel of a CNN processes it. For instance, if the padding in a CNN is set to zero, every new pixel value will have a value of zero.

Several convolutional layers of stride 2. In CNN models, there are often many more than three convolutional kernels. It is common for a convolutional layer to have 16 kernels or even 64 kernels.

Each of these convolution kernels acts as a different kind of filter, making a channel/feature map that shows something different. For example, kernels could filter top edges, bottom edges, diagonal lines, and so on. In networks with a lot more nodes, these kernels could be used to filter out things like animal eyes or bird wings.

When there are more convolutional kernels, there are more channels/feature maps and more data, which takes up more memory. The stride 2 convolution helps to save memory because the width and height of the output channel are half of what they were before. This is based on the assumption that reflection padding is being used. If not, it could be a little bit smaller.

**Question 8.**

**What are the resulting size number of activation maps (wrongly I called “layer”), when we have a starting image of 100x100 pixels, with 30 filters/kernel of size 20x20, stride 2 and padding 0 ?**

Use the following formula to find out how big each activation map is:

A picture containing text

Description automatically generated

so the height of the activation map that comes out is

A picture containing tiled

Description automatically generated

Both the height and the width are the same.

So the number of activation maps ends up being 24 x 24 pixels.

**Question 9.**

**In the posted paper “Deep Learning: New Computational Modeling Techniques for Genomics” in the class materials, explain how does the DNA is transformed to and image so that is can be processed with CNNs ? List some of the applications mentioned in the paper of Artificial Neural Networks in general to DNA data.**

To fit a tabular representation, other input data must first be changed. For example, a DNA sequence must be turned into a list of k-mer counts. Tabular data are the norm for many supervised machine learning models, from simple linear models like logistic regression to more flexible nonlinear models like neural networks and many others.

Applications include classifying transcription factor binding sites and predicting molecular phenotypes like chromatin features, DNA contact maps, DNA methylation, gene expression, translation efficiency, RBP binding, and microRNA (miRNA) targets.

**Question 10.**

**What are the different types of Artificial Neural Networks described in the posted paper ?**

**Describe differences of each with the feed forward neural networks we have see so far, and what are the ideal applications for each of these new types of Artificial Neural Networks described in the paper.**

**fully connected:** Referring to a layer that applies an activation function to each value after an affine transformation of a vector. Fully connected neural networks have been used for a number of genomics applications, such as predicting the percentage of exons spliced in for a given sequence based on sequence features like the presence of splice factor binding motifs or sequence conservation, ranking potential disease-causing genetic variants, and predicting cis-regulatory elements in a given genomic region based on features like chromatin marks, gene expression, and evolutionary conservation.

**convolutional**: When talking about a layer of a neural network that processes data stored in n-dimensional arrays, like images. Several small parts of the input array are given the same fully connected layer. A convolutional layer can be thought of as a set of position weight matrices that are scanned across a DNA sequence. The first convolutional neural networks (CNNs) to be used on genomics data were DeepBind, DeepSEA, and Basset. In DeepBind, many single-task models were trained to predict the binarized (bound or not bound) in vitro and in vivo binding affinities of transcription factors and RNA-binding proteins (RBP). The DeepSEA model predicted whether or not chromatin features, such as transcription factor binding, were present or absent. Basset predicted DNA features that are either accessible or not accessible based on their sequence.

**recurrent**: Refers to a layer of a neural network that processes data in order. At each step in the process, the same neural network is used to update a memory variable that is passed on to the next step. In genomics, RNNs have been used to combine the results of CNNs to predict single-cell DNA methylation states, RBP binding, transcription factor binding, and DNA accessibility. RNNs have also been used in miRNA biology. For example, deepTarget was better than existing models at predicting miRNA binding targets from pairs of mRNA and miRNA sequences, and deepMiRGene was better than existing methods at predicting the presence of precursor miRNAs from the mRNA sequence and its predicted secondary structure. Base calling from raw DNA sequencing data is another task for which RNNs have been used to make predictions. Based on changes in electric current measured by the Oxford Nanopore MinION sequencer, DeepNano correctly guessed the identity of the base.

**graph convolutional**: Referring to neural networks that process graph-structured data; they extend convolution beyond regular structures like DNA sequences and images to graphs with arbitrary structures. Each node and edge in the graph is given the same neural network. GCNs have been used to solve a number of problems in biology and chemistry. One method, for example, used an unsupervised method to get new features of proteins from networks of protein-protein interactions. These features were then used to predict how proteins work in different tissues. GCNs have also been used to model the side effects of taking more than one drug. In chemistry, graph convolutions have been used to predict the solubility of molecules, the effectiveness of drugs, and the efficiency of photovoltaic cells. GCNs can be used in genomics to do things like predict the expression of a binary gene based on the expression of other genes or classify different types of cancer.

**autoencoders**: Unsupervised neural networks are trained to figure out what the input was.

One or more bottleneck layers have fewer dimensions than the input. This causes the data to be compressed and forces the autoencoder to pick out important features and leave out less important ones when it reconstructs the data. Autoencoders have been used to fill in missing data, get signatures of gene expression, and find expression outliers in gene expression data from microarrays and bulk RNA sequencing. Autoencoders have been used for imputation, dimensionality reduction, and representation learning in the field of single-cell genomics. Also, previous biological knowledge has been added to the autoencoder architecture in order to figure out a new way to represent cells from single-cell RNA sequencing (scRNA-seq) data. This makes it easier to group cells together and see what they look like. In the autoencoder framework, tailored loss functions are also used to deal with noise characteristics of scRNA-seq data, such as sparse count data.

**generative adversarial networks**: (GANs) Unsupervised learning models try to make data points that can't be told apart from the ones that have already been seen. As a relatively new method, GANs are not used very often in genomics right now.

They have been used to make DNA sequences that code for proteins and to design DNA probes for microarrays that bind proteins. People have said that GANs can make sequences that are better than those in the training data set, as measured by how well they bind to proteins. GANs have been used to simulate scRNA-seq data and dimensionality reduction in the field of single-cell genomics.