(A) Neural Net & Backprop Review

1 Backprop Warmup

For the function $f(w, x) = \frac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$:

(a) Explain why or why not we should write f as the composition of two simpler functions, that is, find f_1, f_2 such that $f(w, x) = f_2(f_1(w, x))$. If we should, find f_1, f_2.

(b) Draw the network that represents the computation of f.

(c) Write pseudocode for the forward pass and backward pass (backpropagation) in the network.

(d) Update your network drawing with the intermediate values in the forward and backward pass. Use the following weight initialization and inputs $w = [2, -3, -3]$ and $x = [-1, -2]$.gradient derivation from 4d dx = [w[0] * ddot, w[1] * ddot] backprop into

(e) Now consider the function $f(x, y) = \frac{x+\sigma(y)}{\sigma(x)+(x+y)^2}$. What are the forward and backward passes?

2 Model Intuition

(a) What can go wrong if you just initialize all the weights in a neural network to exactly zero? What about to the same value?

(b) Adding nodes in the hidden layer gives the neural network more approximation ability, because you are adding more parameters. How many weight parameters are there in a neural network with architecture specified by $d = \left[d^{(0)}, d^{(1)}, ..., d^{(N)}\right]$, a vector giving the number of nodes in each of the N layers? Evaluate your formula for a 2 hidden layer network with 10 nodes in each hidden layer, an input of size 8, and an output of size 3.

(c) Consider the two networks in the image below, where the added layer in going from Network A to Network B has 10 units with linear activation. Give one advantage of Network A over Network B, and one advantage of Network B over Network A.

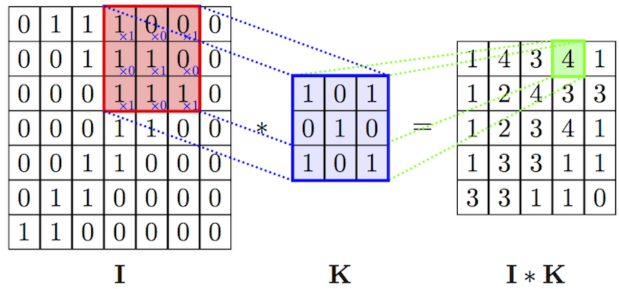
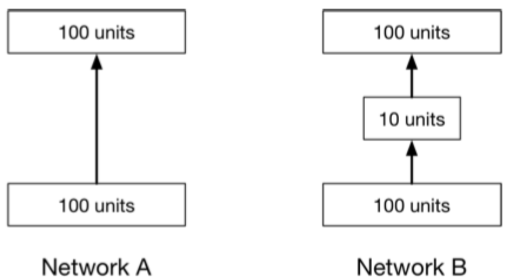


Figure 1: Figure showing an example of one convolution.



3 Convolution and Backprop Revisited

In this problem, we will explore how image masking can help us create useful high-level features that we can use instead of raw pixel values. We will walk through how discrete 2D convolution works and how we can use the backprop algorithm to compute derivatives through this operation. Here we assume that if we index the image I outside its bounds, its corresponding value will be zero.

(a) To start, let's consider convolution in one dimension. Convolution can be viewed as a function that takes a signal $I[]$ and a mask $G[]$, and the discrete convolution at point t of the signal with the mask is

$$(I * G)[t] = \sum_{k=-\infty}^{\infty} I[k]G[t-k]$$

If the mask $G[]$ is nonzero in only a finite range, then the summation can be reduced to just the range in which the mask is nonzero, which makes computing a convolution on a computer possible.

As an example, we can use convolution to compute a derivative approximation with finite differences. The derivative approximation of the signal is $I'[t] \approx (I[t+1] - I[t-1])/2$. Design a mask $G[]$ such that $(I * G)[t] = I'[t]$.

(b) Convolution in two dimensions is similar to the one-dimensional case except that we have an additional dimension to sum over. If we have some image $I[x, y]$ and some mask $G[x, y]$, then the convolution at the point (x, y) is

$$(I * G)[x, y] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I[m, n]G[x - m, y - n]$$

or equivalently,

$$(I * G)[x, y] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} G[m, n]I[x - m, y - n],$$

because convolution is commutative.

In an implementation, we'll have an image I that has three color channels I_r, I_g, I_b each of size $W \times H$ where W is the image width and H is the height. Each color channel represents the intensity of red, green, and blue for each pixel in the image. We also have a mask G with finite support. The mask also has three color channels, G_r, G_g, G_b, and we represent these as a $w \times h$ matrix where w and h are the width and height of the mask. (Note that usually $w \ll W$ and $h \ll H$.) The output $(I * G)[x, y]$ at point (x, y) is

$$(I * G)[x, y] = \sum_{a=0}^{w-1} \sum_{b=0}^{h-1} \sum_{c \in \{r,g,b\}} I_c[x + a, y + b] \cdot G_c[a, b]$$

In this case, the size of the output will be $(1 + W - w) \times (1 + H - h)$, and we evaluate the convolution only within the image I. (For this problem we will not concern ourselves with how to compute the convolution along the boundary of the image.) To reduce the dimension of the output, we can do a strided convolution in which we shift the convolutional mask by s positions instead of a single position, along the image. The resulting output will have size $\lfloor 1 + (W - w)/s \rfloor \times \lfloor 1 + (H - h)/s \rfloor$.

Write pseudocode to compute the convolution of an image I with a set of masks G and a stride of s. Hint: to save yourself from writing low-level loops, you may use the operator $*$ for element-wise multiplication of two matrices (which is not the same as matrix multiplication) and invent other notation when convenient for simple operations like summing all the elements in the matrix.

(c) Masks can be used to identify different types of features in an image such as edges or corners. Design a mask G that outputs a large value for vertically oriented edges in image I. By "edge," we mean a vertical line where a black rectangle borders a white rectangle. (We are not talking about a black line with white on both sides.)

(d) Although handcrafted masks can produce edge detectors and other useful features, we can also learn masks (sometimes better ones) as part of the backpropagation algorithm. These masks are often highly specific to the problem that we are solving. Learning these masks is a lot like learning weights in standard backpropagation, but because the same mask (with the same weights) is used in many different places, the chain rule is applied a little differently and we need to adjust the backpropagation algorithm accordingly. In short, during backpropagation

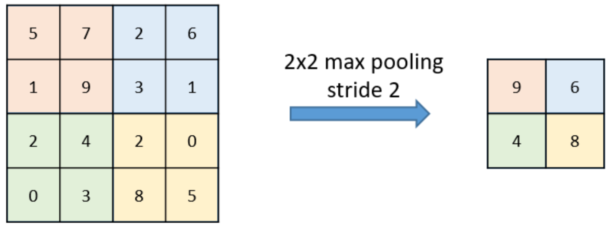


Figure 2: Figure showing an example of one maxpooling.

each weight w in the mask has a partial derivative $\frac{\partial L}{\partial w}$ that receives contributions from every patch of image where w is applied.

Let L be the loss function or cost function our neural network is trying to minimize. Given the input image I, the convolution mask G, the convolution output $R = I * G$, and the partial derivative of the error with respect to each scalar in the output, $\frac{\partial L}{\partial R[i,j]}$, write an expression for the partial derivative of the loss with respect to a mask weight, $\frac{\partial L}{\partial G_c[x,y]}$, where $c \in \{r, g, b\}$. Also write an expression for the derivative of $\frac{\partial L}{\partial I_c[x,y]}$.

(e) Sometimes, the output of a convolution can be large, and we might want to reduce the dimensions of the result. A common method to reduce the dimension of an image is called max pooling. This method works similar to convolution in that we have a mask that moves around the image, but instead of multiplying the mask with a subsection of the image, we take the maximum value in the subimage. Max pooling can also be thought of as downsampling the image but keeping the largest activations for each channel from the original input. To reduce the dimension of the output, we can do a strided max pooling in which we shift the max pooling mask by s positions instead of a single position, along the input. Given a mask size of $w \times h$, and a stride s, the output will be $\lfloor 1 + (W - w)/s \rfloor \times \lfloor 1 + (H - h)/s \rfloor$ for an input image of size $W \times H$.

Let the output of a max pooling operation be an array R. Write a simple expression for element $R[i, j]$ of the output.

(f) Explain how we can use the backprop algorithm to compute derivates through the max pooling operation. (A plain English answer will suffice; equations are optional.)

(B) Generative Models

Recall that generative models attempt to classify data by modelling the prior probability $P(Y)$ and the class conditional probabilities $P(X|Y)$ and selecting the class $k = \text{argmax}_y P(Y = y)P(X = x|Y = y)$ for some data point x. In problems (4)-(6), we explore the effect of the choice of such class conditional distributions on the posterior.

4 Gaussian Classification

Suppose we are dealing with a two-category, one-dimensional classification problem. Let our class conditional probability be Gaussian with shared variance ie. $X|Y = i \sim \mathcal{N}(\mu_i, \sigma^2)$, $P(Y = i) = 1/2$ where $i \in \{0, 1\}$, and $\mu_2 > \mu_1$.

(a) **Find the optimal decision boundary and the corresponding decision rule.**

(b) The probability of misclassification (error rate) is:

$$P_e = P((\text{misclassified as } Y = 1) \mid Y = 2)\, P(Y = 2) + P((\text{misclassified as } Y = 2) \mid Y = 1)\, P(Y = 1).$$

Show that the probability of misclassification (error rate) associated with this decision rule is

$$P_e = \frac{1}{\sqrt{2\pi}} \int_a^\infty e^{-z^2/2} dz,$$

where $a = \dfrac{\mu_2 - \mu_1}{2\sigma}$.

(c) **What is the limit of P_e as σ goes to 0?**

5 Logistic Posterior: different variances

We saw in (4) that Gaussian class conditionals with the same variance lead to a logistic posterior. Now we will consider the case when the class conditionals are Gaussian as in (4), but have different variance, i.e.

$$X|Y = i \sim \mathcal{N}(\mu_i, \sigma_i^2), \quad \text{where } i \in \{0, 1\}$$
$$Y \sim \text{Bernoulli}(\pi)$$

Show that the posterior distribution of the class label given X is also a logistic function, however with a quadratic argument in X. Assuming 0-1 loss, what will the decision boundary look like (i.e., describe what the posterior probability plot looks like)?

6 Logistic Posterior: exponential class conditionals

We have seen in class that Gaussian class conditionals lead to a logistic posterior that is linear or quadratic in X. Now, suppose the class conditionals are exponentially distributed with parameters λ_i, i.e.

$$X|Y = i \sim \lambda_i \exp(-\lambda_i x), \quad \text{where } i \in \{0, 1\}$$
$$Y \sim \text{Bernoulli}(\pi)$$

Show that the posterior distribution of the class label given X is also a logistic function, however with a linear argument in X. Assuming 0-1 loss, what will the decision boundary look like (i.e., describe what the posterior probability plot looks like)?

(C) Assorted Problems

7 Kernelizing Nearest Neighbors

In this question, we will be looking at how we can kernelize k-nearest neighbors (k-NN). k-NN is a simple classifier that relies on nearby sample points to decide what a new point's class should be.

For this problem, you may use without proof that there's a data structure called an *max-heap* that can store tuples of (data point, distance) such that:

- Adding a tuple to a max-heap containing m items costs $O(\log m)$ time.
- Querying for the tuple with the largest distance value costs $O(1)$ time.
- Deleting the tuple with the largest distance value costs $O(\log m)$ time.

(For those of you who are familiar with analysis of algorithms, all times listed are amortized.)

Given a query point q, there are 2 steps to decide what class to predict.

1. Find the k sample points nearest q.
2. Return the class with the most votes from the k sample points.

For the following parts, assuming that our sample points $x \in \mathbb{R}^d$, and we have n sample points.

(a) What is the runtime to classify a newly given query point q, using euclidean distance?

(b) What if instead of looking at the distance between the points in $\mathbb{R}^d$, we wanted to consider the distance in p polynomial space? What dimension would this space be? What would the runtime be to classify a newly given query point q, in terms of n, d, and k, and p?

(c) Instead, we can use the polynomial kernel to compute the distance between 2 points in p polynomial space without having to move all of the points into the higher dimensional space. Using the polynomial kernel, $k(x, y) = (x^T y + \alpha)^p$ instead of Euclidean distance, what is the runtime for k-NN to classify a new point q?

8 Hierarchical Clustering for Phylogenetic Trees

A phylogenetic tree (or "evolutionary tree") is way of representing the branching nature of evolution. Early branches represent major divergences in evolution (for example, modern vertebrae diverging from modern invertebrate), while later branches represent smaller branches in evolution (for example, modern humans diverging from modern monkeys). An example is shown below.

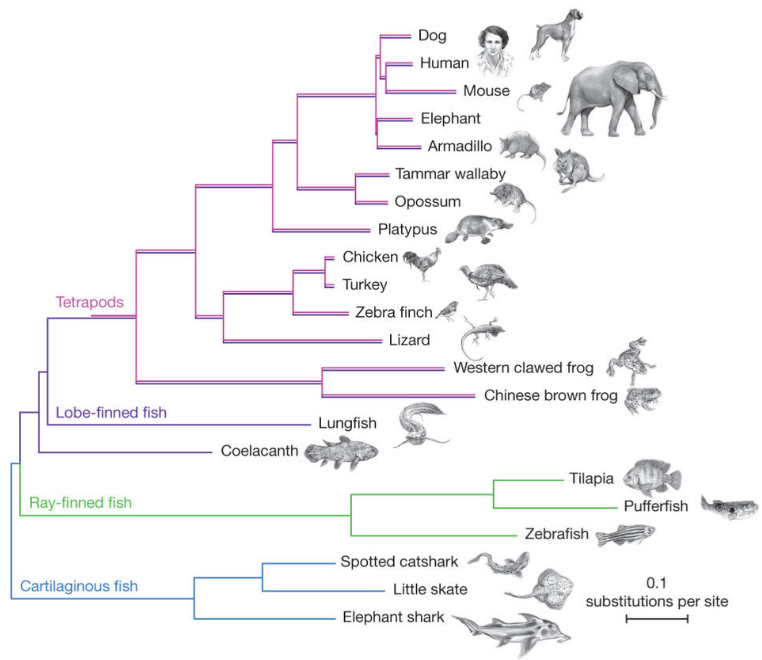


Figure 3: Caption

Creating phylogenetic trees is a popular problem in computational biology. We are going to combine what we know about clustering, decision trees, and unsupervised learning.

We start with all the samples (in this case, animals) in a single cluster and gradually divide it up. This should remind you of decision trees! After k steps, we have at most 2^k clusters. Since we do not have labels, we need to find some way deciding how to split the samples (other than using entropy).

We will use the same objective as in k-means clustering to determine how good our proposed clustering is:

$$\forall i \leq k, \mu_i = \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j$$

$$H(S_1, \ldots, S_k) = \sum_{i=1}^{k} \sum_{x_j \in S_i} |x_j - \mu_i|^2$$

At each iteration, we will split each cluster with more than one element into two clusters. The algorithm terminates when everything is in its own cluster.

(a) Consider the following six animals and their two features. Create the resulting decision tree.

Animal	Lifespan	Wings
Dog	12	0
Human	80	0
Mouse	2	0
Elephant	60	0
Chicken	8	2
Turkey	10	2

(b) Prove that an optimal clustering on $k + 1 < n$ clusters has an objective value that is at least as small as that of the optimal clustering on k clusters.

(c) What is the value of $H(S_1, \ldots, S_k)$ when $k = n$ (the number of samples)?