Machine Learning & Data Mining
Practice Questions

Caltech CS/CNS/EE 155
March 11th, 2015

1 Equivalence of Matrix Norm Definitions

(The purpose of this question is to improve your comfort in manipulating matrix algebra.)

The squared Frobenious norm has two equivalent definitions:

$$\|X\|_F^2 = \texttt{trace}\left(X^TX\right), \quad \text{and} \quad \|X\|_F^2 = \sum_d \sigma_d^2, \tag{1}$$

where σ_d denotes the d-th singular value of X, i.e., σ_d forms the diagonal of Σ in the SVD of $X = U\Sigma V^T$.
Similarly, the trace norm has two equivalent definitions:

$$\|X\|_* = \texttt{trace}\left(\sqrt{X^TX}\right), \quad \text{and} \quad \|X\|_* = \sum_d \sigma_d, \tag{2}$$

where for any symmetric square matrix A, if $B = \sqrt{A}$, then $BB = A$.
Question: Prove that the two expressions in (1) are equivalent, and prove that the two expressions in (2) are equivalent. You can assume that X is a square matrix for convenience.

Solution. We will leverage the SVD of $X = U\Sigma V^T$. To show (1), we see that

$$\begin{aligned}
\texttt{trace}\left(X^TX\right) &= \texttt{trace}\left(V\Sigma U^TU\Sigma V^T\right) && (3)\\
&= \texttt{trace}\left(V\Sigma\Sigma V^T\right) && (4)\\
&= \texttt{trace}\left(V\Sigma^2 V^T\right) && (5)\\
&= \texttt{trace}\left(\Sigma^2 V^TV\right) && (6)\\
&= \texttt{trace}\left(\Sigma^2\right) && (7)\\
&= \sum_d \sigma_d^2, && (8)
\end{aligned}$$

where (1) follows from the fact that U is orthogonal and thus $U^TU = I$, (6) follows from the rotational invarance of the trace operator: $\texttt{trace}(ABC) = \texttt{trace}(BCA) = \texttt{trace}(CAB)$, and (7) follows from the fact that V is orthogonal and thus $V^TV = I$.
To show (2), we see that

$$\begin{aligned}
\texttt{trace}\left(\sqrt{X^TX}\right) &= \texttt{trace}\left(\sqrt{V\Sigma U^TU\Sigma V^T}\right) && (9)\\
&= \texttt{trace}\left(\sqrt{\Sigma^2}\right) && (10)\\
&= \texttt{trace}\left(\Sigma\right) && (11)\\
&= \sum_d \sigma_d, && (12)
\end{aligned}$$

where (10) follows from the same logic as (4)-(7).

2 Properties of Bootstrap Sampling

(The purpose of this question is to improve your comfort in manipulating probability concepts.)

Let $S = \{(x_i, y_i)\}_i = 1^N$ denote a training set of size N. In this question, we will analyze the properties of a single bootstrapped dataset of S. Recall that Bootstrapping is the process of creating a new dataset S' of size N where each entry in S' is sampled uniformly from S (with replacement). Procedurally, this means

- Initialize $S' \leftarrow \emptyset$
- For $j = 1, \ldots, N$
  - sample (x'_j, y'_j) by sampling uniformly from S (and independently of other samples)
  - add (x'_j, y'_j) to S'
- Return S'

This means that some entries in S will appear multiple times in S', and some entries won't appear at all. For a specific $(x, y) \in S$, what is the probability that (x, y) will appear at least once S'.

Question: What does this probability converge to in the limit as N increases? You can assume that every $(x, y) \in S$ is unique (there are no duplicates in S).

Solution. Let $P((x, y) \in S')$ denote the probability that (x, y) appears at least once in S'. Then it suffices to compute $P((x, y) \notin S') = 1 - P((x, y) \in S')$. We can write $P((x, y) \notin S')$ as

$$\begin{aligned} P((x,y) \notin S') &= P\left(\forall (x'_j, y'_j) \in S' : \ (x,y) \neq (x'_j, y'_j)\right) && (13) \\ &= \prod_{j=1}^{N} P((x,y) \neq (x'_j, y'_j)) && (14) \\ &= \prod_{j=1}^{N} \left(1 - \frac{1}{N}\right) && (15) \\ &= \left(1 - \frac{1}{N}\right)^N, && (16) \end{aligned}$$

where (14) follows from the independence of sampling each (x'_j, y'_j), and (15) is simply using the definition of uniform distribution. Thus we can see that the probability of (x, y) appearing at least once in S' is

$$P((x,y) \in S') = 1 - P((x,y) \notin S') = 1 - \left(1 - \frac{1}{N}\right)^N.$$

In the limit as N increases, we can solve:

$$c \equiv \lim_{N \to \infty} \log\left(\left(1 - \frac{1}{N}\right)^N\right), \tag{17}$$

after which we know that:

$$\lim_{N \to \infty} 1 - \left(1 - \frac{1}{N}\right)^N = 1 - e^c. \tag{18}$$

We will use L'Hospital's rule[1] to calculate (17):

$$\begin{aligned}
\lim_{N\to\infty} \log\left(\left(1-\frac{1}{N}\right)^N\right) &= \lim_{N\to\infty} N\log\left(1-\frac{1}{N}\right) \\
&= \lim_{N\to\infty} \frac{\log\left(1-\frac{1}{N}\right)}{\frac{1}{N}} \\
&= \lim_{N\to\infty} \frac{\frac{1}{1-\frac{1}{N}}\frac{1}{N^2}}{-\frac{1}{N^2}} && (19) \\
&= \lim_{N\to\infty} -\frac{1}{1-\frac{1}{N}} \\
&= -1, && (20)
\end{aligned}$$

where (19) follows from applying L'Hospital's rule and the chain rule when differentiating $\log(1-1/N)$. Thus we see that

$$\lim_{N\to\infty} P((x,y)\in S') = 1-\frac{1}{e}.$$

[1] http://en.wikipedia.org/wiki/L%27H%C3%B4pital%27s_rule

3 Bias-Variance Decomposition

(The purpose of this question is to improve your comfort with manipulating probability concepts and loss functions.)

Let f_S denote a regression model[2] whose parameters are trained using training set S. Then if we treat the training set S as a random variable (i.e., sampled from the true test distribution), then f_S is also a random variable.

For any test data point x with true label y, we can write the expected squared loss of f_S as

$$E_S\left[(f_S(x) - y)^2\right], \tag{21}$$

where the expectation is taken over the randomness of the training set S.

Question #1: Derive the bias-variance decomposition of (21):

$$E_S\left[(f_S(x) - y)^2\right] = E_S\left[\left(f_S(x) - \bar{f}(x)\right)^2\right] + E_S\left[\left(\bar{f}(x) - y\right)^2\right], \tag{22}$$

where $\bar{f}$ denotes the expectation of f_S over the randomness of S:

$$\bar{f}(x) = E_S\left[f_S(x)\right]. \tag{23}$$

Question #2: Why are the two terms in the RHS of (22) referred to as the variance and bias of $f_S(x)$, respectively?

Solution. The first term in the RHS of (22) is called the variance of $f_S(x)$ because it measures the variance of the distribution of $f_S(x)$, and actually does not depend on the true label y at all. The second term in the RHS of (22) is called the bias of $f_S(x)$ because it measures the bias of the distribution of $f_S(x)$ relative to the true label y. In other words, if the bias term were 0, then the distribution of $f_S(x)$ could be interpreted as being "centered" around the true label y.

The derivation is as follows:

$$\begin{aligned}
E_S\left[(f_S(x) - y)^2\right] &= E_S\left[f_S(x)^2 - 2yf_S(x) + y^2\right] \\
&= E_S\left[f_S(x)^2\right] - 2yE_S\left[f_S(x)\right] + y^2 && (24)\\
&= E_S\left[f_S(x)^2\right] - 2y\bar{f}(x) + y^2 && (25)\\
&= E_S\left[f_S(x)^2\right] - 2y\bar{f}(x) + y^2 + 2\bar{f}(x)^2 - 2\bar{f}(x)^2 && (26)\\
&= E_S\left[f_S(x)^2\right] - 2y\bar{f}(x) + y^2 + 2\bar{f}(x) - 2E_S\left[f_S(x)\right]\bar{f}(x) && (27)\\
&= E_S\left[f_S(x)^2 - 2f_S(x)\bar{f}(x) + \bar{f}(x)^2\right] + E_S\left[\bar{f}(x)^2 - 2y\bar{f}(x) + y^2\right] && (28)\\
&= E_S\left[\left(f_S(x) - \bar{f}(x)\right)^2\right] + E_S\left[\left(\bar{f}(x) - y\right)^2\right].
\end{aligned}$$

(24) follows from linearity of expectation. (25) follows from applying (23). (26) follows from just adding and substracting $2\bar{f}(x)^2$ (note that $\bar{f}(x)$ is a deterministic constant given x). (27) follows from applying (23). (28) follows from linearity of expectation and re-arranging terms.

[2]For any input x, $f_S(x)$ outputs a scalar real value.

4 Multiclass SVM

(The purpose of this question is to improve your comfort in reasoning about constraints in machine learning optimization problems.)

Consider the multiclass SVM model that makes predictions on input $x \in \Re^D$ via:

$$h(x|w) = \text{argmax}_{y \in \{1,\ldots,K\}} w_y^T x,$$

where the model is defined as:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix} \in \Re^{DK},$$

for each model sub-vector $w_k \in \Re^D$.

The multiclass SVM objective over a training set $S = \{(x_i, y_i)\}_{i=1}^N$ can be described as:

$$\text{argmin}_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{i=1}^N \xi_i \tag{29}$$

s.t.

$$\forall i, \forall y' \in \{1,\ldots,K\} : w_{y_i}^T x_i - w_{y'}^T x_i \geq \mathbf{1}_{[y_i \neq y']} - \xi_i \tag{30}$$

Note that $C \geq 0$ is the hyperparameter that trade-off off between regularization and training loss.

Question: For a single training data point (x_i, y_i), compute the stochastic (sub-)gradient of w by differentiating:

$$\text{argmin}_{w,\xi} \frac{1}{2N}\|w\|^2 + \frac{C}{N}\xi_i \tag{31}$$

s.t.

$$\forall y' \in \{1,\ldots,K\} : w_{y_i}^T x_i - w_{y'}^T x_i \geq \mathbf{1}_{[y_i \neq y']} - \xi_i \tag{32}$$

It will be convenient to use the following equivalent definition of ξ_i:

$$\xi_i = \max_{y' \in \{1,\ldots,K\}} \left\{ \mathbf{1}_{[y_i \neq y']} - \left(w_{y_i}^T x_i - w_{y'}^T x_i \right) \right\}. \tag{33}$$

Also, assume for simplicity that exactly one y' is maximal in (33).

Solution. We can write the gradient w.r.t. w of (31) as:

$$\nabla_w = \frac{1}{N} w + \frac{C}{N}\frac{\partial \xi_i}{\partial w}.$$

Let $\hat{y}$ denote the $y' \in \{1,\ldots,K\}$ that maximizes the definition in (33) for the current value of w. Then we can simplify the definition of ξ_i as:

$$\xi_i = \mathbf{1}_{[y_i \neq \hat{y}]} - \left(w_{y_i}^T x_i - w_{\hat{y}}^T x_i \right), \tag{34}$$

which implies that $\partial\xi_i/\partial w$ is only potentially non-zero in the sub-vectors corresponding to w_{y_i} and $w_{\hat{y}}$.[3]

In the case where $\hat{y} = y_i$ (i.e., the current model w predicts the correct $\hat{y} = y_i$ with sufficiently large margin), then (34) implies that $\xi_i = 0$ and that

$$\frac{\partial \xi_i}{\partial w} = 0.$$

In the case where $\hat{y} \neq y_i$, then (34) implies that $\xi_i > 0$ and that for each sub-vector w_k:

$$\frac{\partial \xi_i}{\partial w_k} = \begin{cases} x_i & \text{if } k = \hat{y} \\ -x_i & \text{if } k = y_i \\ 0 & \text{otherwise} \end{cases}.$$

Note that this derivation extends to general structured SVMs – it just requires an algorithm to compute $\hat{y}$ since that could take exponential time via exhaustive search.

[3]For general structured prediction, finding the $\hat{y}$ that maximizes (34) requires us to do something beyond exhaustive enumeration. For instance, for sequence labeling problems $\hat{y}$ can be computed via dynamic programming approaches such as Viterbi.

5 Featurized Latent Factor Models

(The purpose of this question is to improve your comfort in working with more complicated latent-factor models that activate more than one pair of latent factors for each data point.)

Consider a featurized version of collaborative filtering, where we have user features $z \in \Re^F$ and item features $x \in \Re^D$. We model the rating a user with features z would give to an item with features x as

$$y \approx (Uz)^T(Vx),$$

where $U \in \Re^{K\times F}$ and $V \in \Re^{K\times D}$ are projection matrices that map user and item features into a K-dimensional latent feature space.[4]

Given training data, our goal is to learn U and V by minimizing the regularized training loss:

$$\text{argmin}_{U,V} \frac{\lambda}{2}\left(\|U\|_F^2 + \|V\|_F^2\right) + \frac{1}{2}\sum_{(z,x,y)\in S}\left(y - (Uz)^T(Vx)\right)^2. \tag{35}$$

Question: Derive the gradient for U.

Solution. The answer is:

$$\partial_U = \lambda U - \sum_{(z,x,y)\in S}\left[\left(y - (Uz)^T(Vx)\right)Vxz^T\right].$$

The answer can be derived by applying the chain rule to:

$$\frac{\partial}{\partial U}\left(y - (Uz)^T(Vx)\right)^2 = 2\left(y - (Uz)^T(Vx)\right)\frac{\partial}{\partial U}\left[-(Uz)^T(Vx)\right].$$

We next observe that:

$$(Uz)^T(Vx) = \texttt{trace}\left(z^TU^TVx\right) = \texttt{trace}\left(U^TVxz^T\right),$$

and use the following matrix derivative identity:

$$\frac{\partial}{\partial U}\texttt{trace}\left(U^TM\right) = M$$

to achieve the result.

[4]This setting reduces to conventional feature-less collaborative filtering by assuming that each z and x have exactly one entry 1 and the rest 0.

6 Tensor Latent Factor Models

(The purpose of this question is to improve your comfort in working with higher order models.)

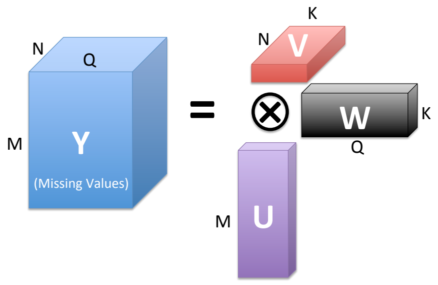

Figure 1: Illustration of 3-way tensor latent factor model.

Consider a three-way D-dimensional latent factor model as depicted in Figure 1:

$$y_{abc} \approx \langle u_a, v_b, w_c \rangle = \sum_{k=1}^{K} u_{ak} v_{bk} w_{ck},$$

where $\langle a, b, c \rangle$ denotes a 3-way dot product, and each latent factor u_a, v_b, w_c is a D-dimensional vector that we will estimate from data. Let $S = \{(a, b, c)\}$ denote a set of training indices where we've observed Y_{abc}. Our goal is to learn the model parameters A, B, and C to minimize the regularized training loss over a training set S:

$$\text{argmin}_{U,V,W} \frac{\lambda}{2} \left(\|U\|_F^2 + \|V\|_F^2 + \|W\|_F^2 \right) + \frac{1}{2} \sum_{(a,b,c) \in S} \left(Y_{abc} - \langle u_a, v_b, w_c \rangle \right)^2 . \tag{36}$$

where u_a, v_b, and w_c denote the corresponding columns of $U \in \Re^{K \times M}$, $V \in \Re^{K \times N}$, and $W \in \Re^{K \times Q}$, respectively.

Question: Compute the gradient w.r.t. u_a of (36).

Hint: use the Hadamard product[5] in representing your solution:

$$v_b \circ w_c = \begin{bmatrix} v_{b1} w_{c1} \\ v_{b2} w_{c2} \\ \vdots \\ v_{bK} w_{cK} \end{bmatrix} \in \Re^K .$$

Solution. Define $S_a = \{(a', b', c') \in S : \; a' = a\}$ as the training indices with first index being a. Focusing just on the u_a component, we can rewrite (36) as

$$\text{argmin}_{u_a} \frac{\lambda}{2} \|u_a\|^2 + \frac{1}{2} \sum_{(a,b,c) \in S_a} \left(Y_{abc} - u_a^T (v_b \circ w_c) \right)^2 . \tag{37}$$

[5]http://en.wikipedia.org/wiki/Hadamard_product_%28matrices%29

We can write the gradient of (37) w.r.t. u_a as

$$\partial_{u_a} = \lambda u_a - \sum_{(a,b,c)\in S_a} \left(Y_{abc} - u_a^T(v_b \circ w_c)\right)(v_b \circ w_c).$$

Note that this is basically identical to the 2-way latent factor model except for the additional Hadamard product. In other words, when optimizing u_a, we treat $(v_b \circ w_c)$ as the "features" and u_a as the linear model parameter.

7 Neural Net Backprop Gradient Derivation

(The purpose of this question is to improve your comfort with models that have multiple layers.)

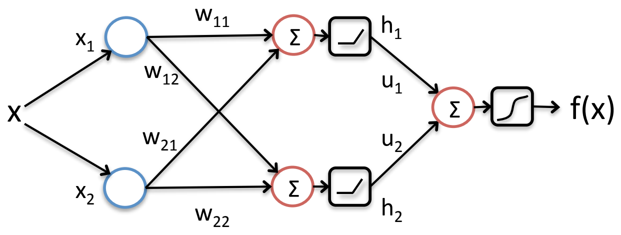

Figure 2: Illustration of Simple Neural Network.

In this question, we will consider the following neural network depicted in Figure 2. There are six parameters in the model, u_1, u_2, w_{11}, w_{21}, w_{12}, and w_{22}.

The network takes in a 2-dimensional input x and outputs a real value $f(x) \in [0,1]$. The final output $f(x)$ is a weighted combination of two hidden node activations with a sigmoid transfer function:

$$f(x) = \sigma\left(u_1 h_1(x) + u_2 h_2(x)\right), \tag{38}$$

where:

$$\sigma(s) = \frac{e^s}{1+e^s}.$$

The two hidden layers outputs, $h_1(x)$ and $h_2(x)$, are weighted combinations of the input x with a rectilinear transfer function:

$$h_i(x) = \tau\left(w_{1i}x_1 + w_{2i}x_2\right), \tag{39}$$

where:

$$\tau(s) = \max\{0, s\}.$$

Question: For a given training data point (x, y), compute the stochastic gradient of the squared-loss of (x, y) w.r.t. w_{11}:

$$\frac{\partial}{\partial w_{11}} L(y, f(x)) \equiv \frac{\partial}{\partial w_{11}} \left(y - f(x)\right)^2.$$

Ignore the case where $f(x)$ might not be differentiable (because the rectilinear function is not differentiable everywhere).

Hint: write out the formula using the chain rule and use the following definition of the derivative of $\sigma(s)$:

$$\frac{\partial}{\partial s}\sigma(s) = \sigma(s)(1-\sigma(s)).$$

Solution. We first note that only the first hidden node h_1 depends on w_{11}, so we can write out the chain rule of derivatives from $L(y, f(x))$ to w_{11} through h_1:

$$\frac{\partial}{\partial w_{11}} L(y, f(x)) = \frac{\partial L}{\partial f}\frac{\partial f}{\partial h_1}\frac{\partial h_1}{\partial w_{11}}. \tag{40}$$

We can expand each term in (40) as:

$$\frac{\partial L}{\partial f} = -2(y - f),$$

$$\frac{\partial f}{\partial h_1} = \frac{\partial \sigma(u_1 h_1 + u_2 h_2)}{\partial h_1} = \sigma(u_1 h_1 + u_2 h_2)(1 - \sigma(u_1 h_1 + u_2 h_2))u_1,$$

$$\frac{\partial h_1}{\partial w_{11}} = \begin{cases} x_1 & \text{if } w_{11}x_1 + w_{21}x_2 > 0 \\ 0 & \text{otherwise} \end{cases}.$$

(In the case where $w_{11}x_1 + w_{21}x_2 = 0$, then $\partial h_1/\partial w_{11}$ is undefined because $\tau(0)$ is not differentiable. However, in practice this basically never happens, and so we generally ignore this case during stochastic gradient descent.)

8 Convolutional Kernels

(The purpose of this question is to improve your comfort in reasoning about spatial models such as convolutional networks.)

Here are three convolution filters:

$$K_1 = \begin{bmatrix} 0 & 0 & -0.375 & 0 & 0 \\ 0 & -0.375 & -0.75 & -0.375 & 0 \\ -0.375 & -0.75 & 7 & -0.75 & -0.375 \\ 0 & -0.375 & -0.75 & -0.375 & 0 \\ 0 & 0 & -0.375 & 0 & 0 \end{bmatrix},$$

$$K_2 = \begin{bmatrix} 0.0054 & 0.0180 & 0.0268 & 0.0180 & 0.0054 \\ 0.0180 & 0.0597 & 0.0890 & 0.0597 & 0.0180 \\ 0.0268 & 0.0890 & 0.1328 & 0.0890 & 0.0268 \\ 0.0180 & 0.0597 & 0.0890 & 0.0597 & 0.0180 \\ 0.0054 & 0.0180 & 0.0268 & 0.0180 & 0.0054 \end{bmatrix},$$

$$K_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

This question involves convolutional filters. Let K be a $(2r+1) \times (2r+1)$-dimensional convolutional filter. Convolving an image X with K corresponds to a new image $\tilde{X}$ with pixel values:

$$\tilde{X}_{i,j} = \sum_{i'=-r}^{r} \sum_{j'=-r}^{r} X_{i+i',j+j'} K_{i'+r+1,j'+r+1}.$$

In other words, each pixel $\tilde{X}_{i,j}$ in the convolved image is computed via the weighted sum of an image patch of X centered at $X_{i,j}$ with weights K (ignore boundary cases).

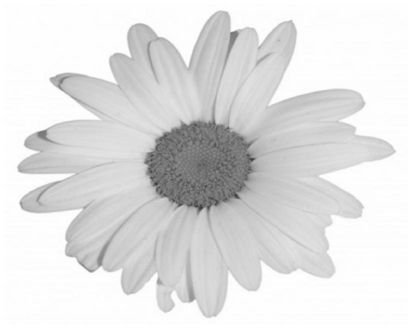

Figure 3: Original Image

Question: Suppose we convolve the three filters K_1, K_2, and K_3 on the image shown in Figure 3. Which resulting image in Figure 4 corresponds to which kernel?

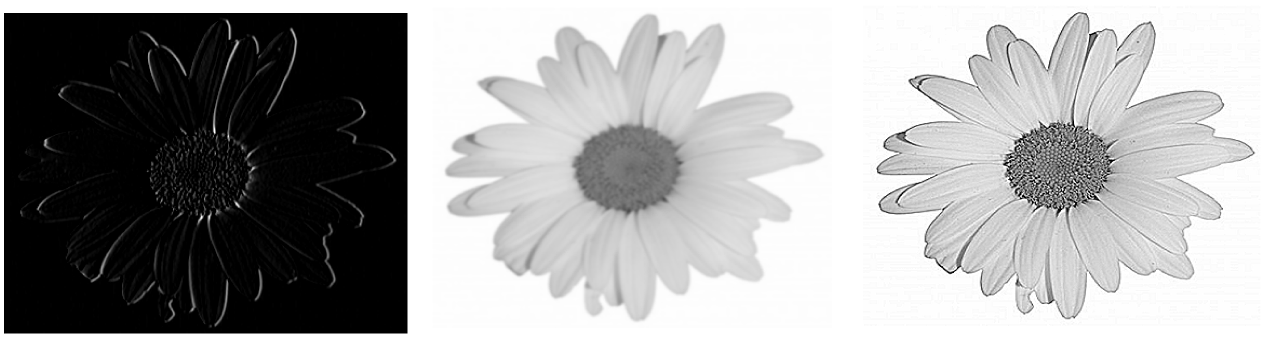

Figure 4: Convolved Images

Solution. K_1 corresponds to the right image in Figure 4. K_1 is a sharpening filter, because it accentuates pixels that have high contrast with neighboring pixels.

K_2 corresponds to the middle image in Figure 4. K_2 is a blurring filter, because it computes a weighted average of a neighborhood of pixels.

K_3 corresponds to the left iamgein Figure 4. K_3 is an edge detection (or Gabor) filter, because it only activates when there is an increase in pixel intensity from left-to-right. Note that the sum of K_3 is 0, meaning that the resulting image $\tilde{X}$ is black whenever there are no pixel intensity gradients in the image neighborhood patch.

9 Hidden Markov Models

(The purpose of this question is to improve your comfort in working with structured or graphical models.)

Given an input sequence $x = (x^1, \ldots, x^M)$ and output sequence $y = (y^1, \ldots, y^M)$, a hidden Markov model decomposes the probability of $P(x, y)$ as:

$$P(x,y) = \prod_{i=1}^{M} P(x^i|y^i)P(y^i|y^{i-1}), \tag{41}$$

where $y^0 = 0$ denotes a special start state.

Suppose each $y^i \in \{1, 2\}$ can take one of two states, and each $x^i \in \{A, B\}$ can take one of two observations. The $P(y^i|y^{i-1})$ probability table is:

$P(y^i\|y^{i-1})$	$y^i = 1$	$y^i = 2$
$y^{i-1} = 0$	2/3	1/3
$y^{i-1} = 1$	3/4	1/4
$y^{i-1} = 2$	1/2	1/2

and the $P(x^i|y^i)$ probability table is:

$P(x^i\|y^i)$	$x^i = A$	$x^i = B$
$y^i = 1$	2/3	1/3
$y^i = 2$	1/5	4/5

Question: Compute $P(x = (A, A, B))$ and $\operatorname{argmax}_y P(x = (A, A, B), y)$.

Solution. We can compute $P(x = (A, A, B))$ by marginalizing out all possible y's:

$$P(x = (A,A,B)) = \sum_y P(x = (A,A,B), y), \tag{42}$$

and then we can just use (41).

OPTION 1: Brute Force Exhaustive Enumeration of All Possible y's. This problem is small enough to exhaustively enumerate all possible y's. There are 8 possible y's that are length-3:

$$y \in \{(1,1,1), (1,1,2), (1,2,1), (1,2,2), (2,1,1), (2,1,2), (2,2,1), (2,2,2)\}.$$

One could simply brute force compute $P(x = (A, A, B))$ via (42) by summing over the 8 possible y's:

$$P(x = (A,A,B), y = (1,1,1)) = \frac{2}{3}\frac{2}{3}\frac{2}{3}\frac{3}{4}\frac{1}{3}\frac{3}{4} = \frac{72}{1296}.$$

$$P(x = (A,A,B), y = (1,1,2)) = \frac{2}{3}\frac{2}{3}\frac{2}{3}\frac{3}{4}\frac{4}{5}\frac{1}{4} = \frac{96}{2160}.$$

$$P(x = (A,A,B), y = (1,2,1)) = \frac{2}{3}\frac{2}{3}\frac{1}{5}\frac{1}{4}\frac{1}{3}\frac{1}{2} = \frac{4}{1080}.$$

$$P(x=(A,A,B),y=(1,2,2)) = \frac{2}{3}\frac{2}{3}\frac{1}{5}\frac{1}{4}\frac{4}{5}\frac{1}{2} = \frac{16}{1800}.$$
$$P(x=(A,A,B),y=(2,1,1)) = \frac{1}{5}\frac{1}{3}\frac{2}{3}\frac{1}{2}\frac{1}{3}\frac{3}{4} = \frac{6}{1080}.$$
$$P(x=(A,A,B),y=(2,1,2)) = \frac{1}{5}\frac{1}{3}\frac{2}{3}\frac{1}{2}\frac{4}{5}\frac{1}{4} = \frac{8}{1800}.$$
$$P(x=(A,A,B),y=(2,2,1)) = \frac{1}{5}\frac{1}{3}\frac{1}{5}\frac{1}{2}\frac{1}{3}\frac{1}{2} = \frac{1}{900}.$$
$$P(x=(A,A,B),y=(2,2,2)) = \frac{1}{5}\frac{1}{3}\frac{1}{5}\frac{1}{2}\frac{4}{5}\frac{1}{2} = \frac{4}{1500}.$$

The sum over the above eight values yields:

$$P(x=(A,A,B) = \frac{72}{1296}+\frac{96}{2160}+\frac{4}{1080}+\frac{16}{1800}+\frac{6}{1080}+\frac{8}{1800}+\frac{1}{900}+\frac{4}{1500} \approx 0.126.$$

and taking the max over the above eight values yields:

$$\operatorname{argmax}_y P(x=(A,A,B),y) = (1,1,1).$$

OPTION 2a: Dynamic Programming via Forward Algorithm to Compute $P(x=(A,A,B))$. The dynamic programming solution leverages the following observation for length-M inputs x:

$$\begin{aligned}
P(x) &= \sum_y P(x,y) && (43)\\
&= \sum_y \prod_{i=1}^{M} P\left(x^i|y^i\right) P\left(y^i|y^{i-1}\right) && (44)\\
&= \sum_{y^{(1:M-1)}} \prod_{i=1}^{M-1} P\left(x^i|y^i\right) P\left(y^i|y^{i-1}\right) \sum_{y^M} P\left(x^M|y^M\right) P\left(y^M|y^{M-1}\right) && (45)\\
&= \sum_{y^{(1:M-1)}} P\left(x^{(1:M-1)}, y^{(1:M-1)}\right) \sum_{y^M} P\left(x^M|y^M\right) P(y^M|y^{M-1}), && (46)
\end{aligned}$$

where $y^{(1:M-1)}$ denotes the length-$(M-1)$ prefix subsequence of y (i.e., the first $M-1$ tokens).

(46) defines a recursive definition of $P(x)$ that we can exploit to more efficiently and compactly compute $P(x)$. For a given x, define α_t^i as:

$$\alpha_t^i = \sum_{y^{(1:i-1)}} P\left(x^{(1:i)}, y^{(1:i-1)} \oplus t\right) \equiv P\left(x^{(1:i)}\left|y^i = t\right.\right),$$

where $y^{(1:i-1)} \oplus t$ denotes the list append operator that appends token t to the end of $y^{(1:i-1)}$. In other words α_t^i is the total joint probability of the first i tokens in x, $x^{(1:i)}$, conditioned on $y^i = t$. Thus, we can write $P(x)$ for lenght-M x as:

$$P(x) = \sum_t P\left(x|y^M = t\right) = \sum_t \alpha_t^M. \tag{47}$$

Exploiting (46), we can recursively define α_t^i as:

$$\alpha_t^i = P\left(x^i \left| y^i = t\right.\right) \sum_{t'} \alpha_{t'}^{i-1} P\left(y^i = t \left| y^{i-1} = t'\right.\right), \tag{48}$$

which gives us an efficient and compact way to recursively compute each α^i vector and finally (47):

$$\alpha^1 = \begin{bmatrix} \frac{2}{3}\frac{2}{3} \\ \frac{1}{5}\frac{1}{3} \end{bmatrix} \approx \begin{bmatrix} 0.4444 \\ 0.0667 \end{bmatrix},$$

$$\alpha^2 = \begin{bmatrix} \frac{2}{3}\left(\frac{3}{4}\alpha_1^1 + \frac{1}{2}\alpha_2^1\right) \\ \frac{1}{5}\left(\frac{1}{4}\alpha_1^1 + \frac{1}{2}\alpha_2^1\right) \end{bmatrix} \approx \begin{bmatrix} 0.2444 \\ 0.0289 \end{bmatrix},$$

$$\alpha^3 = \begin{bmatrix} \frac{1}{3}\left(\frac{3}{4}\alpha_1^2 + \frac{1}{2}\alpha_2^2\right) \\ \frac{4}{5}\left(\frac{1}{4}\alpha_1^2 + \frac{1}{2}\alpha_2^2\right) \end{bmatrix} \approx \begin{bmatrix} 0.0659 \\ 0.0604 \end{bmatrix}.$$

Via (47) summing over α^3 yields $P(x = (A, A, B)) = \alpha_1^3 + \alpha_2^3 \approx 0.126$.

OPTION 2b: Dynamic Programming via Viterbi to Compute $\operatorname{argmax}_y P(x = (A, A, B), y)$. Define $\hat{y}_t^i$ as the length-i solution ending in the token t that maximizes probability of $P(x^{(1:i)}, \hat{y}_t^i)$:

$$\hat{y}_t^i = \left(\operatorname{argmax}_{y^1,\ldots,y^{i-1}} P\left(x^{(1:i)}, y^{(1:i-1)} \oplus t\right)\right) \oplus t. \tag{49}$$

We can also keep track of the probability as α_t^i:

$$\alpha_t^i = P\left(x^{(1:i)}, \hat{y}_t^i\right). \tag{50}$$

Assuming we have computed $\hat{y}_t^M$, the most likely y can be solved via:

$$\operatorname{argmax}_y P(x, y) = \operatorname{argmax}_t P\left(x, \hat{y}_t^M\right).$$

The key observation is that we can compute $\hat{y}$ and α recursively via:

$$\hat{y}_t^i = \left(\operatorname{argmax}_{\hat{y}_{t'}^{i-1}} P\left(x^{(1:i)}, y_{t'}^{i-1} \oplus t\right)\right) \oplus t, \tag{51}$$

$$\alpha_t^i = \operatorname{argmax}_{t'} \alpha_{t'}^{i-1} P\left(x^i \left| t\right.\right) P\left(t \left| t'\right.\right). \tag{52}$$

Typically (51) is computed as a side product of computing (52).

$$\alpha^1 = \begin{bmatrix} \frac{2}{3}\frac{2}{3} \\ \frac{1}{5}\frac{1}{3} \end{bmatrix} \approx \begin{bmatrix} 0.444 \\ 0.067 \end{bmatrix}, \quad \hat{y}^1 = \begin{bmatrix} (1) \\ (2) \end{bmatrix}.$$

$$\alpha^2 = \begin{bmatrix} \frac{2}{3}\frac{3}{4}\alpha_1^1 \\ \frac{1}{5}\frac{1}{4}\alpha_1^1 \end{bmatrix} = \begin{bmatrix} 0.222 \\ 0.022 \end{bmatrix}, \quad \hat{y}^2 = \begin{bmatrix} (1,1) \\ (1,2) \end{bmatrix}.$$

$$\alpha^3 = \begin{bmatrix} \frac{1}{3}\frac{3}{4}\alpha_1^2 \\ \frac{4}{5}\frac{1}{4}\alpha_1^2 \end{bmatrix} = \begin{bmatrix} 0.056 \\ 0.044 \end{bmatrix}, \quad \hat{y}^2 = \begin{bmatrix} (1,1,1) \\ (1,1,2) \end{bmatrix}.$$