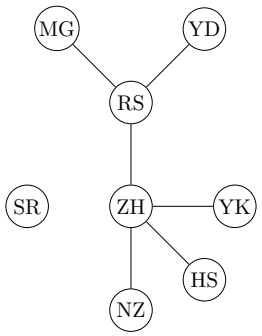
HW #3

Due: 5pm October 20, 2017

Modeling Influence in a Network

This problem considers modeling the flow of influence within a social network using an undirected graphical model. We assume there is a new piece of information currently gaining influence in the network. In particular, for this example we consider a viral GIF that is being shared among the CS281 teaching staff. For each person in the network, we will associate a binary random variable indicating whether they are aware of the information, e.g. RS=1 indicates that Rachit Singh has seen the GIF.

We consider the small fixed network of (relatively unpopular) individuals with a set of connections that form a forest.



With each person in the network we associate a unary log-potential e.g. $\theta_{\text{RS}}(0)$ and $\theta_{\text{RS}}(1)$ indicating a "score" for them seeing the GIF on their own.

s	SR	YD	MG	ZH	HS	RS	NZ	YK
$\theta_s(1)$	2	-2	-2	-8	-2	3	-2	1
$\theta_s(0)$	0	0	0	0	0	0	0	0

Additionally we assume a set of attractive binary log-potentials $\theta_{s-t}(1,1) = 2$ for all connected nodes $s, t \in E$ with all other values $\theta_{s-t}(1,0) = 0$, $\theta_{s-t}(0,1) = 0$, $\theta_{s-t}(0,0) = 0$ across the network.

Problem 1 (30pts)

(Note: for this problem you will be deriving several properties of undirected graphical models and implementing these properties in Python/PyTorch. Before getting started it is worth getting familiar with the `itertools` package in Python and also being sure you understand how PyTorch variables work. In particular try writing a simple scalar function with several vector-valued Variables as inputs and making you understand what the function `.backward` does in practice.)

(a) Implement a function for computing the global score for an assignment of values to the random variables, i.e. a function that takes in $\{0, 1\}$ values for each person and returns the *unnormalized* value without the $A(\theta)$ term. What is the score for an assignment where RS and SR are the *only* people who have seen the GIF?

(b) Implement a function for computing the log-partition function $A(\theta)$ using *brute-force*, i.e. take in a vector θ and return a scalar value calculated by enumeration. What is its value? Using this value compute the *probability* of the assignment in (a)?

For the problems below we will be interested in computing the marginal probabilities for each random variable, i.e. $p(\text{SR} = 1)$, $p(\text{RS} = 1)$, etc.

(c) First, implement a function to compute $p(\text{RS} = 1)$ using brute-force marginalization. Use the functions above to enumerate and sum all assignments with $\text{RS} = 1$.

(d) Using what we know about exponential families, derive an expression for computing all marginals directly from the log-partition function $A(\theta)$. Simplify the expression.

(e) Combine parts (b) and (d) to implement a method for computing the marginal probabilities of this model using PyTorch and autograd. Ensure that this gives the same value as part (c).

(f) Finally compute all the marginals using exact belief propagation with the serial dynamic programming protocol, as described in class and in Murphy section 20.2.1. Use MG as the root of your graph. Verify that your marginals are the same as in the previous two calculations.

(g) How do the relative values of the marginal probabilities at each node differ from the original log-potentials? Explain which nodes values differ most and why this occurs.

(h) (Optional) Implement the parallel protocol for exact BP. How does its practical speed compare to serial?

Solution

(a)

Implementing this, we get an unnormalized score of 148.4132 for the assignment in which only RS and SR have seen the GIF.

(b)

Implementing this function, we get a scalar value of 3423.3716 for the total enumeration. The log partition is then 8.138. We then get a probability of 0.04335 for part (a) when normalized.

(c)

For the assignment with RS=1, we get a probability of 0.880797.

(d)

From section notes, we are given that

$$\frac{dA(\theta)}{d\theta} = \mathbb{E}(\phi(x))$$

$$\phi(x) = \begin{bmatrix} \mathbb{I}(C_1) \\ I(C_2) \\ \vdots \\ \mathbb{I}(C_N) \end{bmatrix},$$

where C_i refers to clique configuration i. Then,

$$\mathbb{E}(\phi(x)) = \mathbb{E}(\begin{bmatrix} \mathbb{I}(C_1) \\ I(C_2) \\ \vdots \\ \mathbb{I}(C_N) \end{bmatrix})$$

$$= \begin{bmatrix} \mathbb{E}(\mathbb{I}(C_1)) \\ \mathbb{E}(I(C_2)) \\ \vdots \\ \mathbb{E}(\mathbb{I}(C_N)) \end{bmatrix}$$

By the fundamental bride of probability, an expectation of an indicator r.v. is equal to the probability of the event described by that indicator taking place.

$$= \begin{bmatrix} P(C_1) \\ P(C_2) \\ \vdots \\ P(C_N) \end{bmatrix}$$

(e)

When implemented, we get basically identical values. We end up with a vector θ containing all combinations within cliques, giving us a vector of length 40. This vector is pasted below:

[0.0134,0.4924,0.0018,0.4924,0.0134,0.4924,0.0018,0.4924,0.0151,0.0001,
0.9436,0.0412,0.2578,0.7009,0.0020,0.0393,0.8445,0.1143,0.0206,
0.0206,0.8445,0.1143,0.0206,0.0206,0.1192,0.8808,0.5058,0.4942,0.5058,
0.4942,0.9588,0.0412,0.8651,0.1349,0.0152,0.9848,0.8651,0.1349,0.2598,0.7402]

(f)

Again, when implemented, we get the same values. We end up with a combination of the cliques again as seen above.

(g)

Consider the example of the node called ZH. This node has a really low score/log unary potential of -8, which seems to intuitively indicate a very low probability of ZH seeing the GIF taking place. Despite this, it actually has the higest probability of taking place among all the unary events in the vector above. This is because it shares the most connections out of all of the nodes in the network, and hence, it benefits from the positive score brought about by binary potentials. These binary potentials intuitively serve as a sort of additive enhancement of probability of seeing a GIF; if a connection sees a GIF, it is more likely that the node in question sees a GIF. Hence, while low log potentials generally correspond to low probabilities, high connectivity with other nodes can often supersede that.

A Note on Sparse Lookups

For the next two problems it will be beneficial to utilize sparse matrices for computational speed. In particular our input data will consist of sparse indicator vectors that act as lookups into a dense weight matrix. For instance, if we say there are 500 users each associated with a vector $\mathbb{R}^{100}$ it implies a matrix $W \in \mathbb{R}^{500 \times 100}$. If we want a fast sparse lookup of the 1st and 10th user we can run the following code:

```
W = nn.Embedding(500, 100)
x = Variable(torch.LongTensor([ [1], [10] ]))
print(W(x))
```

This same trick can be used to greatly speed up the calculation of bag-of-words features from the last homework. Let's use the same example:

- `We like programming. We like food.`

Assume that the vocabulary is

```
["We", "like", "programming", "food", "CS281"]
```

In last homework, we converted the above word id sequence to a vector of length of vocab size:

- `[2, 2, 1, 1, 0]`

Instead we can convert it vector of sentence length (often much shorter):

- `[0, 1, 2, 0, 1, 3]`

In order to calculate the same dot product between $\mathbf{x}$ and a weight vector $\mathbf{w} = [0.5, 0.2, 0.3, 0.1, 0.5]$ we can do sparse lookups and a sum: (verify yourself that it is correct):

```
vocab_size = 4
W = nn.Embedding(vocab_size, 1, padding_idx=text_field.stoi('<pad>'))
W.weight.data = torch.Tensor([ [0.5], [0.2], [0.3], [0.1], [0.5] ])
x = Variable(torch.LongTensor([ [0, 1, 2, 0, 1, 3] ]))
result = torch.sum(W(x), dim=1)).squeeze()
```

This code computes the same dot-product as in HW2 but by doing 6 fast vector lookups into w and summing them together. (The `padding_idx` part ensures the embedding code works when there are sentences of different length by setting a lookup of its argument to return 0. This lets you use a rectangular matrix even with batches of different sentence lengths.) For more details, see the documentation for this module on the PyTorch website.

Problem 2 (Modeling users and jokes with a latent linear model, 25pts)

In this problem, we'll use a latent linear model to jointly model the ratings users assign to jokes. The data set we'll use is a modified and preprocessed variant of the Jester data set (version 2) from `http://eigentaste.berkeley.edu/dataset/`. The data we provide you with are user ratings of 150 jokes. There are over 1.7M ratings with values 1, 2, 3, 4 and 5, from about seventy thousand users. **The data we give you is a modified version of the original Jester data set, see the README, please use the files we provide and not the original ones**. The texts of the jokes are also available. Warning: most of the jokes are bad, tasteless, or both. At best, they were funny to late-night TV hosts in 1999-2000. Note also that some of the jokes do not have any ratings and so can be ignored.

Update: If you are having memory issues with this problem. Please see the note in the readme.

(a) Let $r_{i,j} \in \{1, 2, 3, 4, 5\}$ be the rating of user i on joke j. A latent linear model introduces a vector $u_i \in \mathbb{R}^K$ for each user and a vector $v_j \in \mathbb{R}^K$ for each joke. Then, each rating is modeled as a noisy version of the appropriate inner product. Specifically,

$$r_{i,j} \sim \mathcal{N}(u_i^T v_j, \sigma^2).$$

Draw a directed graphical model with plate diagrams representing this model assuming u_i and v_j are random vectors.

(b) Derive the log-likelihood for this model and the gradients of the log-likelihood with respect to u_i and v_j.

(c) Implement the log-likelihood calculation using PyTorch. Compute the gradients using autograd and confirm that is equal to the manual calculation above. (Hint: Read the documentation for `nn.Embedding` to implement u and v.).

(d) Now set $K = 2$, $\sigma^2 = 1.0$ and run stochastic gradient descent (`optim.SGD`). We have provided a file `utils.py` to load the data and split it into training, validation and test sets. Note that the maximum likelihood estimate of σ^2 is just the mean squared error of your predictions on the training set. Report your MLE of σ^2.

(e) Evaluate different choices of K on the provided validation set. Evaluate $K = 1, \ldots, 10$ and produce a plot that shows the root-mean-squared error on both the training set and the validation set for each trial and for each K. What seems like a good value for K?

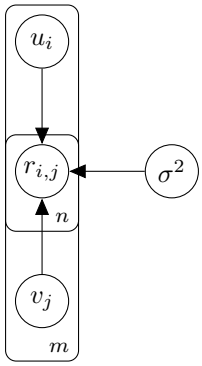
(f) We might imagine that some jokes are just better or worse than others. We might also imagine that some users tend to have higher or lower means in their ratings. In this case, we can introduce biases into the model so that $r_{ij} \approx u_i^{\mathrm{T}} v_j + a_i + b_j + g$, where a_i, b_j and g are user, joke and global biases, respectively. Change the model to incorporate these biases and fit it again with $K = 2$, learning these new biases as well. Write down the likelihood that you are optimizing. One side-effect is that you should be able to rank the jokes from best to worst. What are the best and worst jokes and their respective biases? What is the value of the global bias?

(g) Sometimes we have users or jokes that only have a few ratings. We don't want to overfit with these and so we might want to put priors on them. What are reasonable priors for the latent features and the biases? Modify the above directed graphical model that shows all of these variables and their relationships. Note that you are not required to code this new model up, just discuss resonable priors and write the graphical model.

Solution to 2

(h)

Let us designate n to represent the number of users and m to represent the number of jokes.



(i)

$$\mathcal{L}(u_i, v_j, r_{i,j}) = \log(\prod_{i,j} p(r_{i,j} \mid u_i, v_j))$$

$$= \sum_{i,j} \log(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_{i,j} - u_i^T v_j)^2}{2\sigma^2}})$$

$$= -\frac{nm}{2}\log(2\pi\sigma^2) - \sum_{i,j} \frac{(r_{i,j} - u_i^T v_j)^2}{2\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial u_i} = -\sum_{i,j} \frac{(r_{i,j} - u_i^T v_j) v_j}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial v_j} = -\sum_{i,j} \frac{(r_{i,j} - u_i^T v_j) u_i}{\sigma^2}$$

(j)

Using the log-likelihood implementation, we get essentially the same values for gradients using various test cases/values.

(k)

Running SGD optim with $K = 2$, we get an MLE of 2.6232950234218421.

(l)

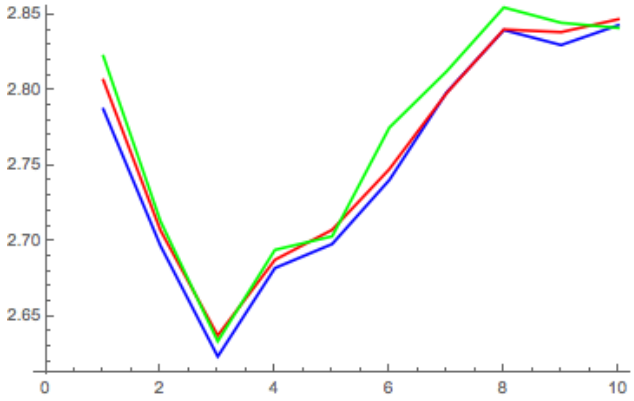
See Table 1 for the results of the evaluation of the model on different values of K. Clearly, the values in the evaluation are all pretty similar across values of K. This may be hard to understand at first, but further testing and exploration/interpretation of the model led me to the conclusion that this is because the model is lacking the bias variables introduced in the next part. The reason this is significant is because the interaction term, while conceptually quite interesting, may not in practice be a very great term to use. The biases center

K	RMSE Train	RMSE Val	RMSE Test
1	2.7875543240297899	2.8068654777383907	2.8227969187089782,
2	2.6967922987008679	2.7070131525441767	2.7131376882238865
3	2.6232950234218421	2.6372265564868956	2.6334142650932547
4	2.6820463524144185	2.6875306115282682	2.6941258059221269
5	2.6980055861118886	2.7074595968400086	2.7031318169310976
6	2.7405598087356617	2.7477790657038747	2.7537423633450419
7	2.7986820913319969	2.7982464346618305	2.8126646650667684
8	2.8401257211258461	2.8404751729547321	2.8550991631223529
9	2.8301227656760234	2.8386980407000953	2.8449235727937495
10	2.843304849357639	2.8472711975944764	2.8414993127556836

Table 1: Results of model across different values of K

scores for different users and jokes and hence make them stronger at fitting the data. The plot is shown below (blue refers to train, red to val, and green to test).

In addition, it's important to note that increasing K may in theory lead to a mix of better specificity in the model and overfitting concerns. With more and more data, we should be able to keep increasing K to create a better fit. The results in the table, for example, show that $K = 3$ is best for this case.



RMSE Train vs. K

(m)

The new likelihood we are optimizing is

$$\mathcal{L}(u_i, v_j, a_i, b_j, r_{i,j}, g) = -\frac{nm}{2}\log(2\pi\sigma^2) - \sum_{i,j}\frac{(r_{i,j} - u_i^T v_j - a_i - b_j - g)^2}{2\sigma^2}.$$

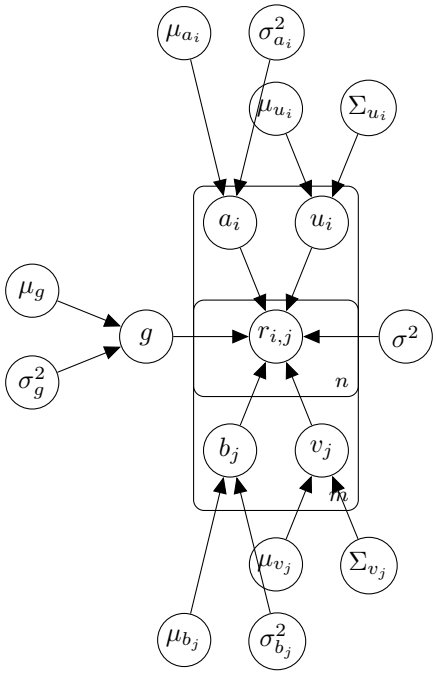
Our global bias comes out to a value of 2.3577, and our best joke is the 131st joke, with $b_{131} = 2.89153934$, while our worst joke is the 51st joke, with $b_{51} = -2.77755117$. Our RMSE values are, for train, val, and test, respectively, $1.835, 1.830, 1.846$.

(n)

Some reasonable priors for the biases and latent features are as follows:

$$g \sim \mathcal{N}(\mu_g, \sigma_g^2); a_i \sim \mathcal{N}(\mu_{a_i}, \sigma_{a_i}^2); b_j \sim \mathcal{N}(\mu_{b_j}, \sigma_{b_j}^2)$$
$$u_i \sim \mathcal{N}(\mu_{u_i}, \Sigma_{u_i}); v_j \sim \mathcal{N}(\mu_{v_j}, \Sigma_{v_j}),$$

In practice, we can often set $\mu_g = 2.5, \mu_{a_i} = 0, \mu_{b_j} = 0, \mu_{u_i} = \overrightarrow{0}_K, \mu_{v_j} = \overrightarrow{0}_K \ \forall \ i, j$, where $\overrightarrow{0}_K$ represents a vector of length K containing all 0s. This is in essence an uninformed prior; allowing for larger values of the variance parameters is in essence admission of larger uncertainty. Just as we did for the mean parameters, unless we have more information, we can use some generic, reasonable, and uninformative prior for the variance parameters (e.g. spherical Gaussian variances, equal across all users and jokes). Learning from the data should soon supersede the priors to create individual distributions for each user and joke. The graphical model then becomes:



Ordinal Regression

Update: If you are running into memory issues with this problem. Please see the note in the readme.

We now address the problem of predicting joke ratings given the text of the joke. The previous models assumed that the ratings where continuous real numbers, while they are actually integers from 1 to 5. To take this into account, we will use an ordinal regression model. Let the rating values be $r = 1, \ldots, R$. In the ordinal regression model the real line is partitioned into R contiguous intervals with boundaries

$$b_1 < b_2 < \ldots < b_{R+1} = +\infty\,, \tag{1}$$

such that the interval $[b_r, b_{r+1})$ corresponds to the r-th rating value. We will assume that $b_1 = -4$, $b_2 = -2$, $b_3 = 0$, $b_4 = 2$ and $b_5 = 4$. Instead of directly modeling the ratings, we will be modeling them in terms of a hidden variable f. We have that the rating y is observed if f falls in the interval for that rating. The conditional probability of y given f is then

$$p(y = r \mid f) = \begin{cases} 1 & \text{if} \quad b_r \leq f < b_{r+1} \\ 0 & \text{if} \quad \text{otherwise} \end{cases} = \Theta(f - b_r) - \Theta(f - b_{r+1})\,, \tag{2}$$

where $\Theta(x)$ is the Heaviside step function, that is, $\Theta(x) = 1$ if $x > 0$ and $\Theta(x) = 0$ otherwise.

Notably there are many possible values that of f that can lead to the same rating. This uncertainty about the exact value of f can be modeled by adding *additive Gaussian noise* to a noise free prediction. Let σ^2 be the variance of this noise. Then $p(f \mid h) = \mathcal{N}(f|h, \sigma^2)$, where h is a new latent variable that is the noise free version of f.

Given some features $\mathbf{x}$ for a particular joke, we can then combine the previous likelihood with a linear model to make predictions for the possible rating values for the joke. In particular, we can assume that the noise-free rating value h is a linear function of $\mathbf{x}$, that is $h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$, where $\mathbf{w}$ is a vector of regression coefficients. We will assume that the prior for $\mathbf{w}$ is $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Computational Note for this Problem: **You will need the following numerical trick, which is implemented as *log_difference* in utils.**

If $a > b$ but $a - b$ is close to 0, you can compute $\log(a - b)$ instead as:

$$\log(1 - \exp(\log b - \log a)) + \log a$$

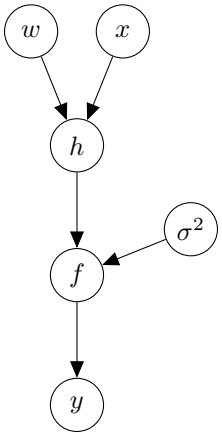
Because a is always larger than b we have that $\exp(\log a - \log b)$ is always smaller than 1 and larger than 0. Therefore, $\log(1 - \exp(b - a))$ is always well defined.

Problem 3 (Ordinal linear regression 25pts)

1. Draw the directed graphical model for applying this model to one data point.
2. Compute the form of $p(y \mid h)$ in the ordinal regression model. Explain how this would differ from a model with no additive noise term.
3. Give the equation for the mean of the predictive distribution in the ordinal regression model. How would is this term affected by σ^2 (include a diagram).
4. Implement a function to compute the log-posterior distribution of the ordinal regression model up to a normalization constant. Use autograd to compute the gradients of the previous function with respect to $\mathbf{w}$ and σ^2. Finds the MAP solution for $\mathbf{w}$ and σ^2 in the ordinal regression model given the available training data using SGD. Report the average RMSE on the provided test set. We have provided some helper functions in `utils.py`.
5. Modify the previous model to have a Gaussian likelihood, that is, $p(y = r \mid h) = \mathcal{N}(r \mid h, \sigma^2)$. Report the resulting average test RMSE of the new model. Does performance increase or decrease? Why?
6. Consider a variant of this model with $\sigma^2(\mathbf{x}) = \mathbf{w}_\sigma{}^\top \mathbf{x}$. How would this model differ? (Optional) Implement this model.
7. How does the performance of the models analyzed in this problem compare to the performance of the model from Problem 2? Which model performs best? Why?

Solution to 3

(o)



(p)

$$p(y = r \,|\, h) = \begin{cases} 1 & b_r \leq h \leq b_{r+1} \\ 0 & else \end{cases}$$

$$= \begin{cases} 1 & b_r \leq w^T x \leq b_{r+1} \\ 0 & else \end{cases}$$

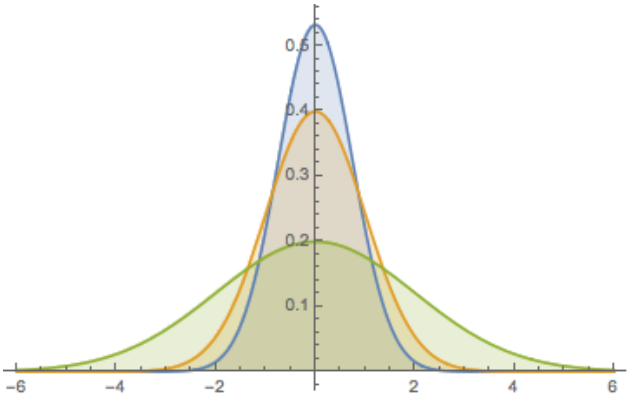
$$= \mathbb{I}(b_r \leq w^T x \leq b_{r+1})$$

This differs from the model with additive noise because the probability distribution of y is deterministically determined by the value of x, whereas in the additive noise model, y is not deterministically determined by x, as the range/bucket $w^T x$ is in is not a surefire sign of the value of y.

(q)

$$\mathbb{E}(y) = \sum_{r=1}^{5} P(b_r \leq f \leq b_{r+1}) r$$

Because f takes into account the added noise of σ^2, with increasing σ^2, if $h \in [b_{r'}, b_{r'+1}), P(f \in [b_{r'}, b_{r'+1}))$ decreases. The actual value of $\mathbb{E}(y)$ could change if h were not centered within an interval, or it might stay the same if h were exactly centered within a interval and all the finite intervals had equal size. In the diagram below, we see a plot of the distribution for low, medium, and high values of σ^2 in blue, yellow, and green, respectively. In this case, the increasing value of the variance does not affect the expectation, presumably because the intervals fulfill the previously stated conditions. However, it is certainly conceivable that if h were not centered within the interval, the probability of f being within the intervals either to the left or to the right would increase substantially with increasing σ^2, leading to a shifted expectation to the left (perhaps if $|h - b_{r'}| < |h - b_{r'+1}|$) or to the right (if $|h - b_{r'}| > |h - b_{r'+1}|$).



Plot of the effect of increasing σ^2

(r)

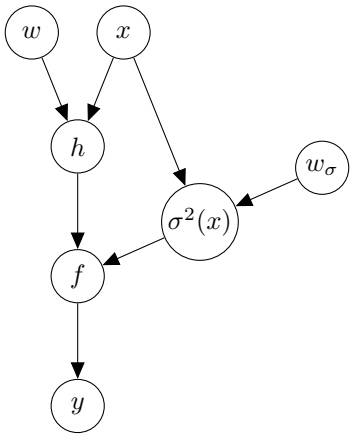
The implementation yields an average RMSE on the validation data of 1.49082139. The average RMSE on the test set is 1.50192074.

(s)

Modifying the previous model, we end up with an average RMSE on the validation data of 2.06329071. The average RMSE on the test set is 2.054819201. Performance decreases, likely because we don't discretize anymore. Under these new continuous predictions, we are always guaranteed to see an error, and thus, the RMSE will always climb with each data point. Meanwhile, in the former ordinal case, we will likely predict many values correctly, and hence, we will not increase error with every data point/prediction. There's also the sense that in the ordinal model, the optimal distributions will in essence be centered between two boundary points, while in the Gaussian likelihood model, we will see optimal distributions be centered at the boundary points. As a result, it is conceivably easier to eliminate error in the ordinal model vs. the Gaussian likelihood model. This also helps to explain why similar Gaussian models in question 2 do not perform too well.

(t)

In this variant of the model, the variance is a function of the data, and hence, variance is not constant. In essence, we begin to train w_σ instead of σ. Training this will likely end up needing a lot more data as well, because we are incurring a different σ for each data point. Thus, even though we are potentially training a small number of weights for σ, we will end up with a lot more sensitivity on this parameter. However, with a lot of data, this is a more descriptive and accurate model because it allows for customization of the variance parameter.



(u)

The performance of this problem's models far exceed that of problem 2's models. It's likely because we are incorporating more information other than the specific joke and specific user. Rather than discretizing/bucketing every aspect of information, we are taking into account the actual construction of the jokes. It's apparent then that we have to take in and interpret more data, seen by longer run times. And hence, we end up with a model that's better suited to analyze and assign joke scores. If we somehow combined both models, we would likely end up with even higher scores as a result of more data and specificity in the model's fitting.