Deep Learning - Report

Hail Hochman (ID 322766353), Binyamin Cohen (ID 212924401)

Part 3: Training

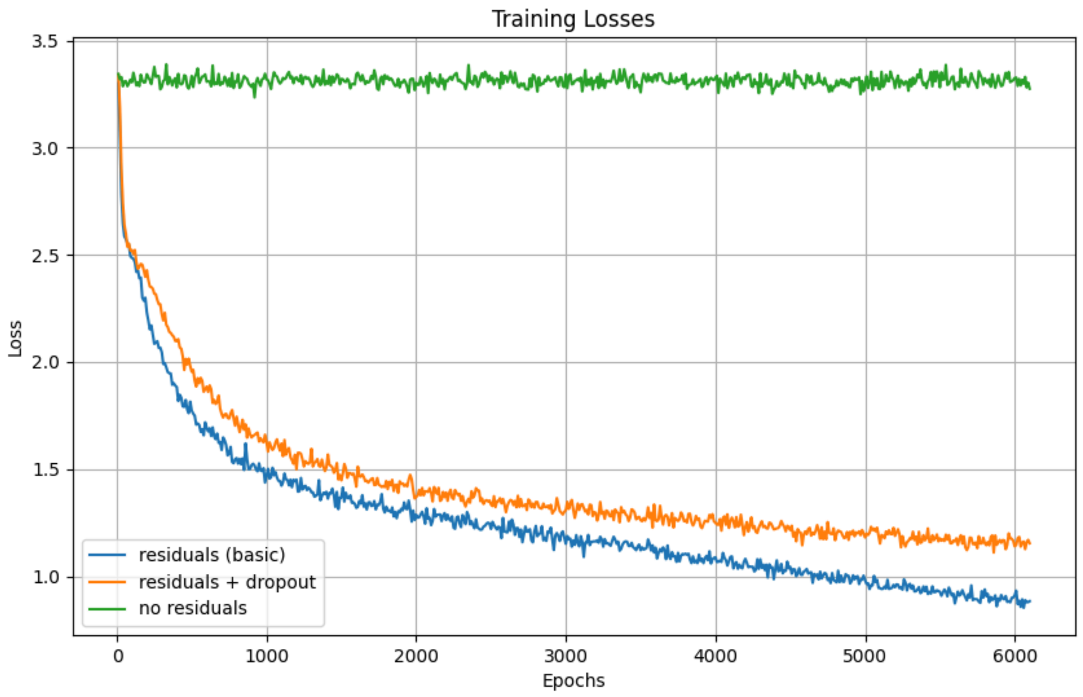
We can play with the parameters to improve training. Parameters that can be changed:

- Batch_size (default: 64)
- n_layers (default: 6)
- n_heads (default: 6)
- embed_size (default: 192)
- learning_rate (default: 5e-4)
- Dropout (default: no)
- Residual Connections (default: yes)

In order to compare results of different settings, we limited the number of epochs (5000-6000) and tested the decreasing of the loss through the train. After we got the best parameters according to the limited training, we trained using the best parameters with more epochs (50000).

Residuals and dropout

We will start by showing the effects of residual connections and dropouts. Let's show the result of the default parameters with/without residuals, and with/without dropout.
For simplicity, we will use only a dropout of 0.1 as suggested, and implement it after the embedding layer and after the self attention (we could check different setups of dropout, but decided to leave it to the reader, and show only the general conclusions).



We can see that when we don't use residual connections the network doesn't learn, at least after 6000 batches.
With regard to the dropout, adding dropout in fact lowers the performance of the model with respect to the loss. But it doesn't mean that the actual text generation will have lower quality, since dropout prevents overfitting and improves robustness. Theoretically we could evaluate the language model quality with perplexity or quality of generated text, but for now we are just going to use dropout, according to the common understanding that dropout actually helps the model.

Layers and Attention heads

We are now going to check different configurations of the number of attention layers, and amount of heads in each of them. We tried a few setups - number of layers from 5 to 15, and number of heads from 5 to 12.
Here is a table of losses after 5000 batches:

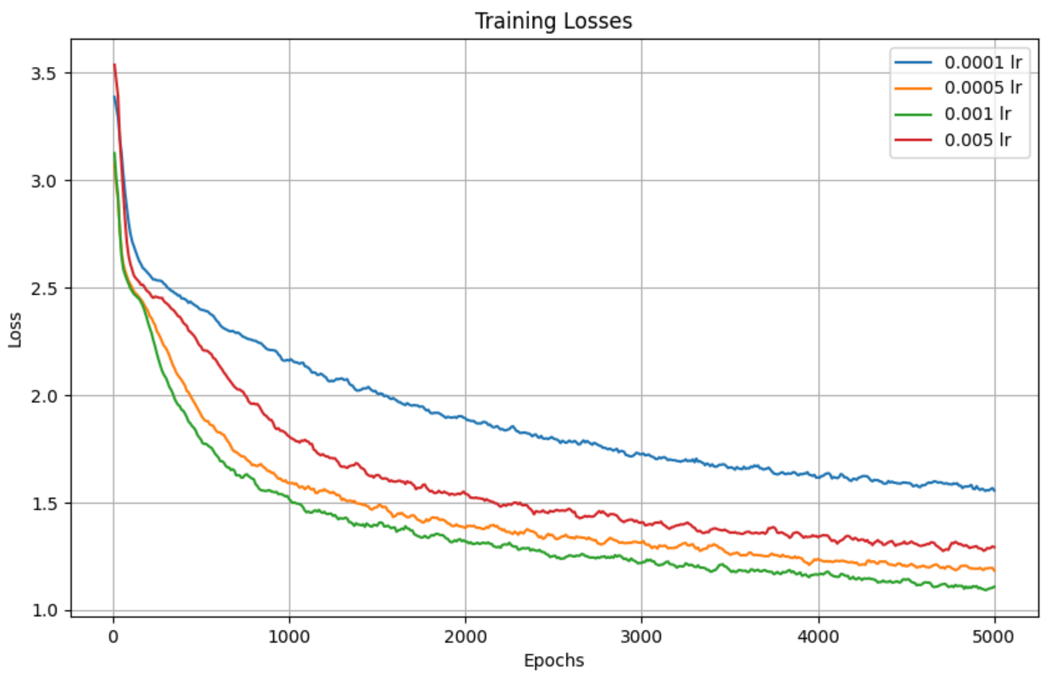
Layers \ Heads	5	6	8	10	12	Avg
5	1.223	1.225	1.221	1.225	1.245	1.227
6	1.206	1.181	1.205	1.227	1.229	1.209
8	1.196	1.165	1.155	1.16	1.22	1.179
10	1.153	1.128	1.182	1.218	1.227	1.181
12	1.133	1.165	1.18	1.161	1.168	1.161
15	1.032	**1.025**	1.08	1.074	1.11	**1.064**
Avg	1.157	**1.148**	1.170	1.177	1.199	-

We can see that the effect of layers is significant on the performance of the network. In most cases we can see a decrease in loss when increasing the number of layers. The downside is that the deeper the network, the longer the training time. When trained with 5 layers for 5000 batches it took 10 minutes, and with 15 layers it took around 30 minutes.
On the other hand, the increase of the number of heads doesn't necessarily improve the performance of the model. It can be explained by the fact that the embedding size stays mostly the same, and therefore when increasing the number of heads, the output of each head has a smaller dimension - lower expressibility.
So to the final model we chose 15 layers and 6 heads.

Learning rate

We checked some different learning rates, when we used only 6 layers and 6 heads because of running time limitations.

We checked learning rate values of 1e-4, 5e-4, 1e-3, 5e-3:



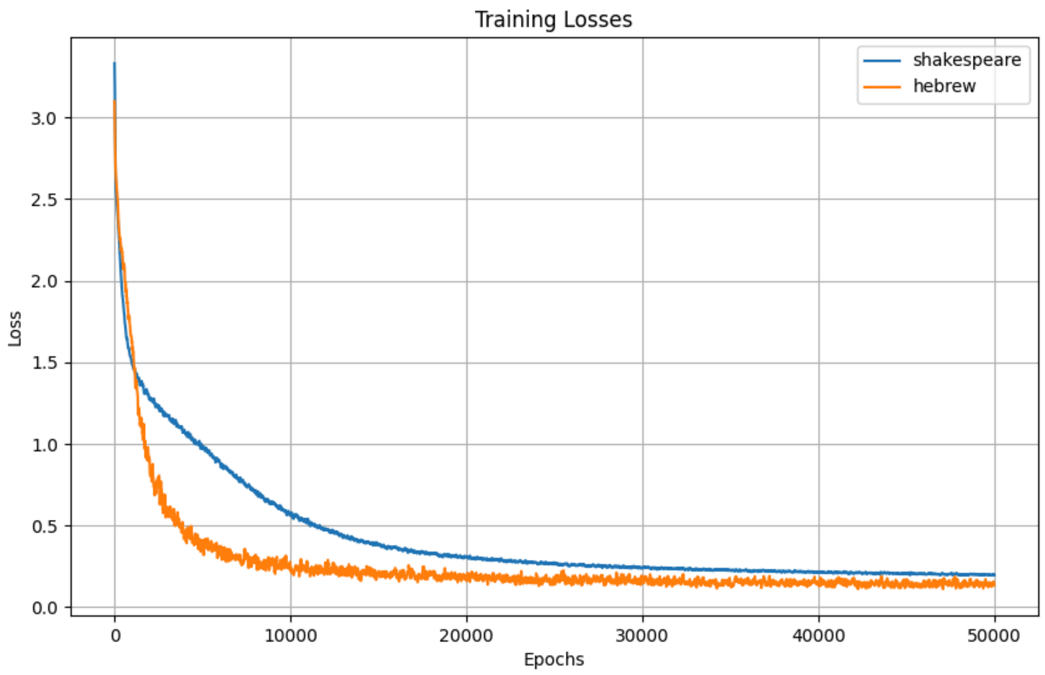
We can see that the best learning rate (from the ones we tried) is lr=0.001 (1e-3), so we will use it.
(Note: we smoothed the curves slightly with average on window of 5 values)
We decided to keep the default of 64 batch size and 192 embedding dimension, as they are standard values and we focused on other parameters. But the reader is encouraged to add other experiments as he sees fit.

Summary:

1. Best loss we found on Shakespeare data was around 0.2.
    a. We had 50000 updates (50k batches, 1 update for each one).
    b. Training sequences = 50k * 64 samples per batch = 3.2M sequences
    c. Network parameters: 6.72M parameters
2. Parameters:
    a. Batch_size = 64
    b. n_layers = 15
    c. n_heads = 6
    d. embed_size = 192
    e. learning_rate = 1e-3
3. Modifications:
    a. Dropout of 0.1 (after embedding layer and self attention layer)
    b. Initialization:
        i. **biases:** initialized with 0

    ii. **normalization layer:** initialized with ones
    iii. **embedding:** initialized with normal values (mean 0, std 0.02)
    iv. **weights of linear layers:** initialized with Xavier method (uniform with boundaries according to the size of the matrix).
  c. Residual Connections (as in the default implementation)

4. Hebrew - I get around 0.14 loss with the same settings as in the english data, better than the english corpus by 0.06. Interestingly enough, the convergence of the Hebrew data was also quicker. We speculate that the vocabulary is smaller in Hebrew (no big and small letters), therefore it's quicker and easier for the model to learn (Note: English data is 1.1M characters, when Hebrew is around 1.5M, so the size isn't an explanation). There is definitely a correlation between the loss and quality, although I didn't create an automatic metric to measure the quality.
   Comparison between the Hebrew and the Shakespeare data:

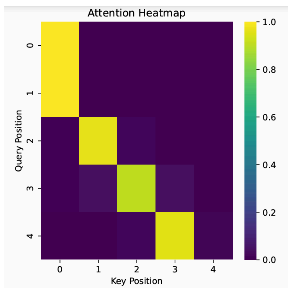
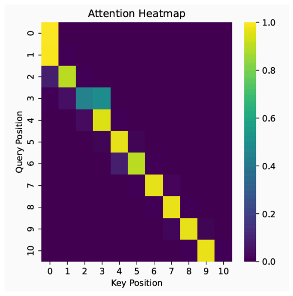
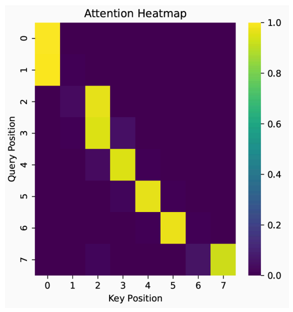
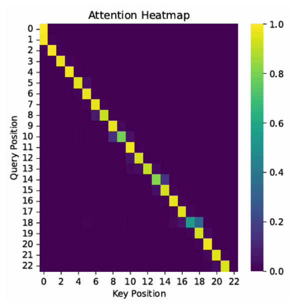


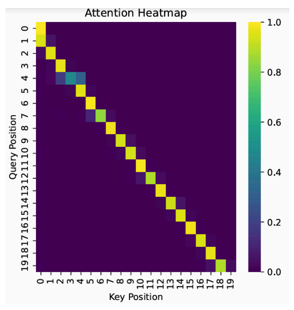
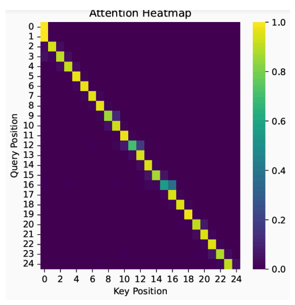
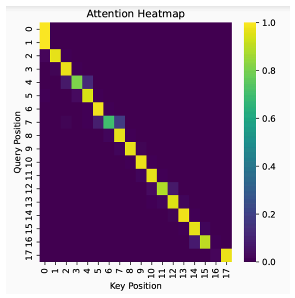
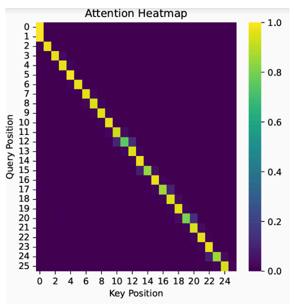
Part 5 - Interpretability:

To make the analysis simpler, we did not take the best model (15 layers with 6 heads in each layer). We chose to analyze a model with 6 layers and 6 heads for each layer.
The model gained a loss of 0.4 and generated good-looking english texts.
To save the trained model and use it, we added functions to *TransformerLM* that allow saving the trained model to a pth file and loading the model from a file.
We used these functions after training and in *interpretability.py* where we tested the trained model with different inputs.
When we saw a repeating pattern among the inputs, we tried to understand what it meant in the text. In addition, we tried to identify what happens at different indices in the inputs, such as in the case of repeated letters in the sequence or spaces.
We tested the model on several text samples and looked for patterns in different combinations of layers and heads.
We created for each sample a heatmap of each layer + head.
The x-axis represents the “key-position”, while the y-axis represents the “query-position”.
In all the patterns we found, the pattern repeated itself in different inputs, but sometimes there was also a lot of noise and sometimes it was clearer.

We found the following patterns (we count the layers and the heads from 1):

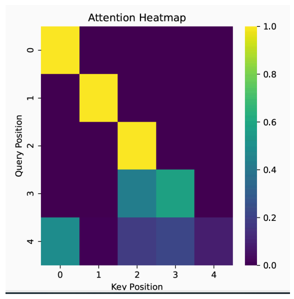
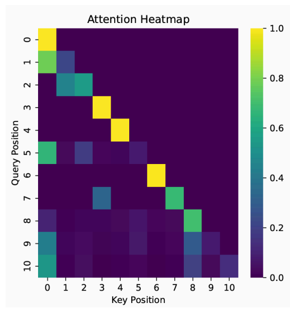
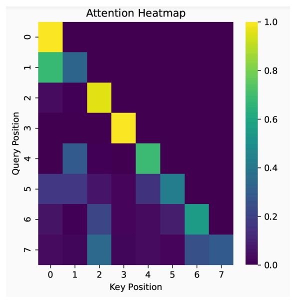
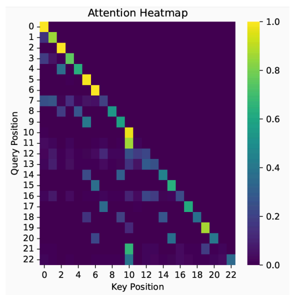
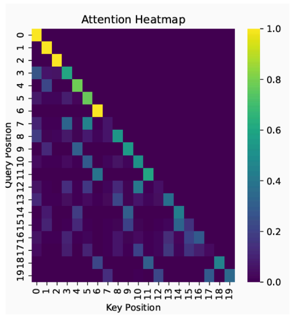
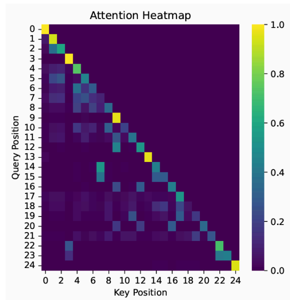
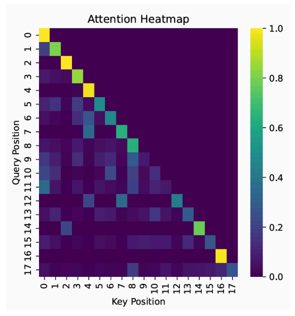
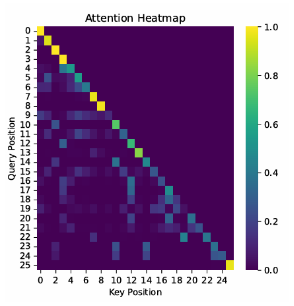
Layer 1 head 1 - look for the last previous character:

“Hello”	“how are you”	“no! Why?”	“these are John and Mary”
			
"I am very very tired"	"attention is all you need"	“what is your name?”	"The children went to sleep"

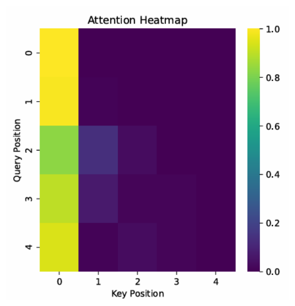
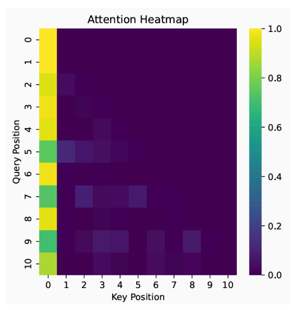
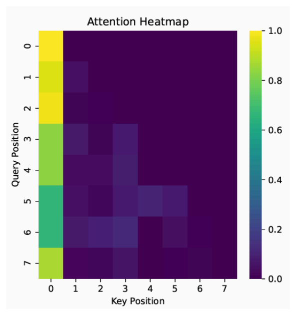
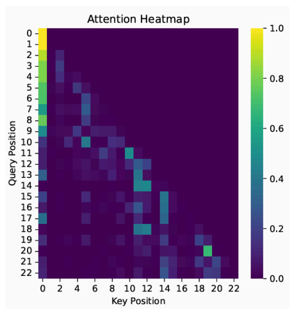
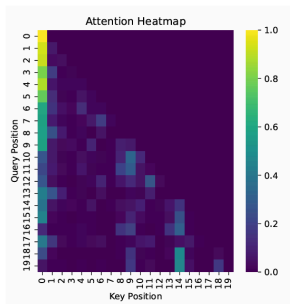
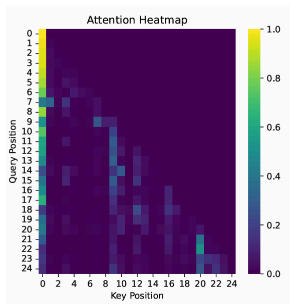
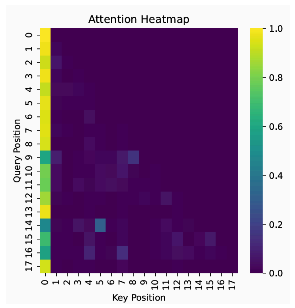
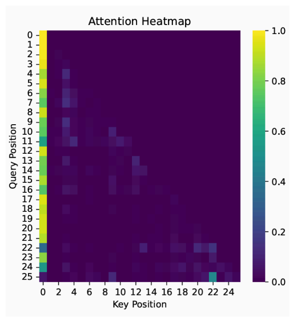
We can see in the examples above that for most of the “query” positions, the “key” position that receives the highest score is the position of the previous letter.
For example - for the input "hello", the highest score obtained for position 2 on the Y-axis is in 1 position on the X-axis. For position 0 the highest score obtained is in the same position, because there is no previous letter for this position.

Layer 1 head 5 - look for the previous characters that are equal to the current character (include the current):

“Hello”	“how are you”	“no! why?”	“these are John and Mary”
			
"I am very very tired"	"attention is all you need"	“what is your name?”	"The children went to sleep"
			

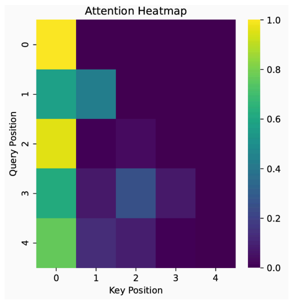
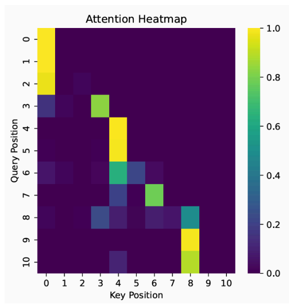
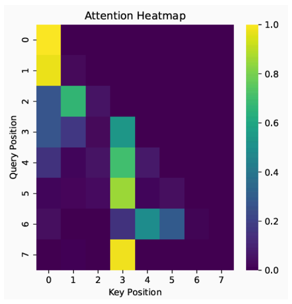
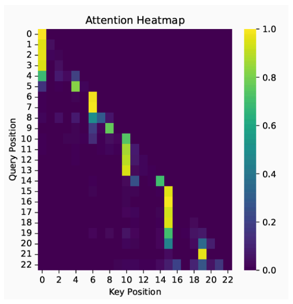
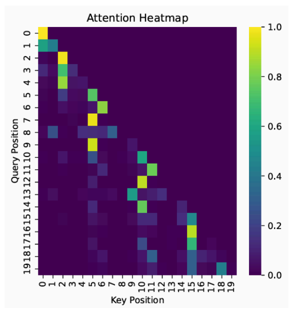
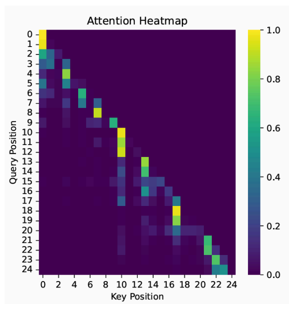
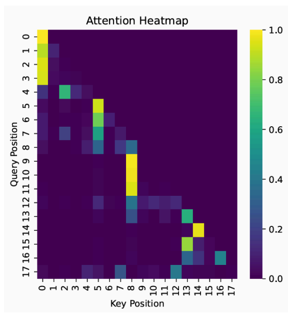
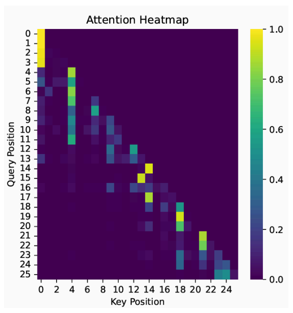
There is a lot of noise in these examples, but we still found a repeating pattern: for many of the letters that appear more than once in the sentence, the highest-scoring positions are all positions of that letter up to and including the current position.
For example - in the sentence "The children went to sleep", we can see that for the position of the last letter e in the sentence - 24 in y-axis, the positions that receive the highest score are 24, 23, 14, 10 and 2 in x-axis. These are the positions of the letter e in the entire sentence.
In the sentence “these are John and Mary”, for the position of the last letter a in the sentence - 20 in y-axis, the positions that receive the highest score are 20, 15, 6 in x-axis. These are the positions of the letter a in the entire sentence.

Layer 4 head 5 - look for the first character in the sequence:

“Hello”	“how are you”	“no! why?”	“these are John and Mary”
			
"I am very very tired"	"attention is all you need"	What is your name?	"The children went to sleep"
			

In this layer+head, in most of the samples we tested we got that for each "query" position, the key position that got the highest score is the position of the first letter in the sequence.

Layer 3 head 5 - look for start of the word (the character after the last space / start of the sequence):

“Hello”	“how are you”	“no! why?”	“these are John and Mary”
			
"I am very very tired"	"attention is all you need"	“what is your name?”	"The children went to sleep"
			

At this point, in most samples it was possible to identify a pattern in which each letter receives the highest score in the letter that begins the word that contained that letter.
For example - for the input “how are you”, the highest score for the letters ‘y’, ‘o’, ‘u’ in indices 8, 9, 10 is accepted in index 8 on x-axis - the index of the first letter ‘y’.

Conclusion

This exercise really helped us understand the material better. The code writing was focused on the challenging and relevant parts, dealing with the core concepts without too much technical stuff. The exercise took some time to complete, but it was definitely worth it. Additionally, it was interesting to examine the attention patterns and actually feel the results in the hand. Overall, we enjoyed doing this exercise and learned from it.