EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Discussion 6

1. **Inductive Biases in Standard Deep Learning Building Blocks** In this question, we will compile and review inductive biases that standard deep learning building blocks have. This question is inspired by Battaglia et al. (2018), so we highly recommend you read through this paper.

   Inductive biases are any assumptions that learners utilize to learn the world and predict the output. These assumptions can either be related to the date-generation process, the functional family space chosen, or the space of the solutions. Inductive biases could be a regularization term added for better generalization or preventing overfitting, and they could also be embedded in the model function family or model architecture (e.g. CNN vs MLP). Inductive biases generally reduce the amount of data needed to fit the model while constraining the model's flexibility. Ideally, inductive biases will improve training efficiency (small data points, small gradient steps, faster optimization), while maintaining performance and generalizing well. However, if inductive biases are mismatched with the problem domain, this will lead our solutions to be sub-optimal. Another way to think about this is through the lens of the bias-variance tradeoff: inductive biases induce a large bias term in approximation error.

   The first example of inductive bias we'll look at is Ridge Regression. What inductive biases lead us to introduce a L2 Penalty on the ordinary least squares objective?

   Solution: When we fit such a model, minimizing least squares with an L2 penalty implies the constraint that the learned function should be a linear function, and generalization errors should be minimized under a quadratic penalty. This also assumes **that real data is generated by a true linear model with Gaussian noise that has been added**. The L2 penalty allows us to achieve a model with smaller parameters.

   Note that we don't usually say any assumptions explicitly when using Ridge Regression. I.e. inductive biases may not be explicitly stated by the programmer; they are simply baked into the design of the model.

   Now let's discuss the inductive biases of deep learning building blocks. Fill in the table below.

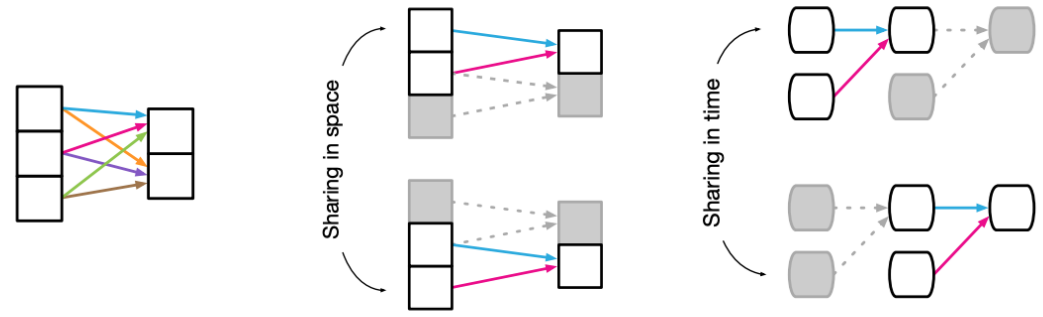
   Solution: Let's talk about inductive biases of standard deep learning building blocks: fully connected layer, convolutional layer and recurrent layer. First, the fully connected layer is the most prevalent building block. This layer takes units (elements of input vector) as input. All units in the previous layer are connected to all units in the following layer (all-to-all connections). Therefore, this network has very weak inductive bias, but note that it is able to fit to any function.

   Next, the convolutional layer takes pixels (patches or grids), and some notable features include locality and translational invariance. Locality is induced by the sparse connections of the CNN, which allow the

Blocks	Inputs	Interactions	Invariance	Inductive Bias
Fully Connected	**Solution:** Units	**Solution:** All-to-all	**Solution:** No	**Solution:** Weak
Convolutional	**Solution:** pixels, grids, patches	**Solution:** local	**Solution:** space translation	**Solution:** locality
Recurrent	**Solution:** time steps	**Solution:** sequential	**Solution:** time translation	**Solution:** sequentiality

model to extract locally correlated features. Translation invariance is induced by the reuse of parameters across input patches (Fig. 1). These inductive biases are usually very effective for natural image data.

Lastly, the recurrent layer takes in input over time, along with the hidden state of the previous layer. The parameters are shared over time, so it has the inductive bias of temporal invariance (Fig. 1).



(a) Fully connected (b) Convolutional (c) Recurrent

Figure 1: How weight sharing occurs in common deep learning building blocks. The arrows with the same color means the same parameter are shared. (a) There is no sharing in fully connected layers. (b) The parameters are shared over space in convolutional layers. (c) The parameters are shared over the time in recurrent layers

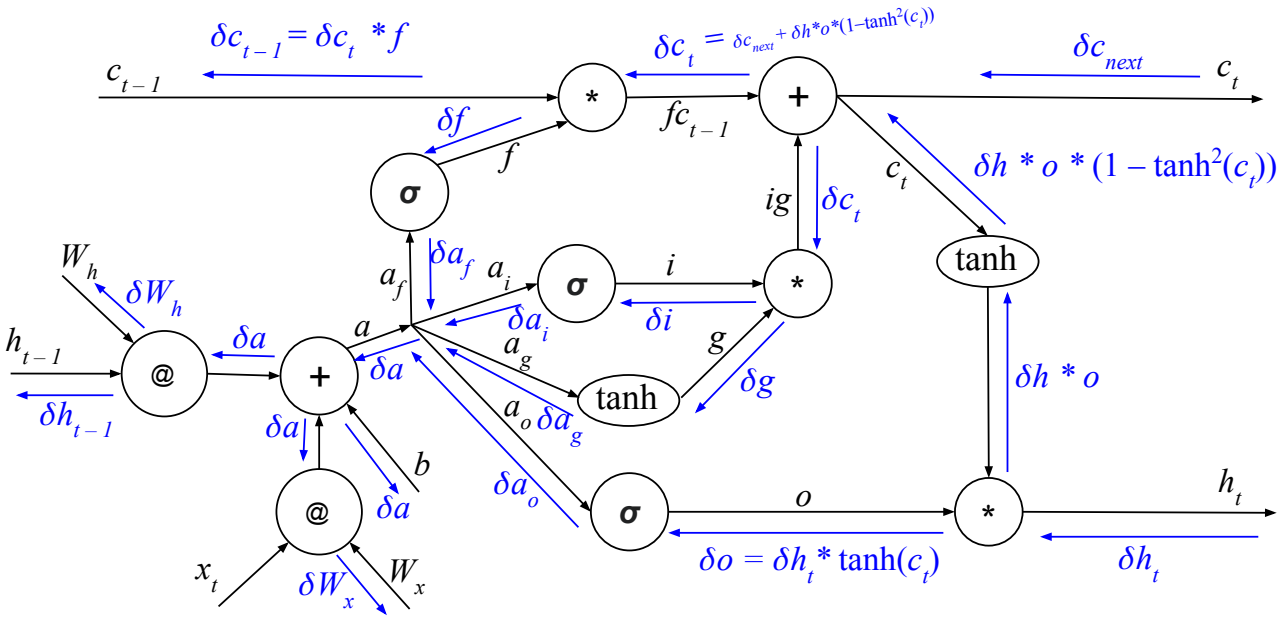
2. **Long Short Term Memory (LSTM)** Vanilla RNNs suffer from vanishing / exploding gradients. The intuition behind LSTMs is similar to that of skip connections that allowed us to pass gradients forward a few layers to mitigate this problem. LSTMs rely on a different kind of recurrent cell, whose schematic is shown below. The @ symbol represents matrix-vector multiplication.

Solution: TAs should discussion in section: We can compare the LSTM to a vanilla RNN. Since a vanilla RNN had to use the hidden state h_t both to produce outputs as well as store memories, h_t gets updated with an affine map and a tanh activation at every timestep, which can easily lead to vanishing or exploding gradients. In contrast, the LSTM can use the cell state C_t as its "long-term memory," and backpropagation through the long term memory is much easier since the cell states change fairly slowly in a simple way (simply being a moving average of the compute $\tilde{C}_t$'s.). On the other hand, the hidden state in the LSTM can serve as a "short-term memory" and change quickly through time.

You may find the following derivatives useful. $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, and $\tanh'(x) = 1 - \tanh^2(x)$

Let's say the upstream gradients are δh_t and δc_{next}. Trace the gradients through the LSTM cell above!

Solution: Note: In the solution below, we use the shorthand $\delta z = \frac{\partial L}{\partial z}$ where L is the loss function and z is an arbitrary variable.



Solution: The picture was getting a bit cluttered so the partial derivatives (labeled on the diagram) are listed here:

$$\delta c_t = \delta c_{next} + \delta h * o * (1 - \tanh^2(c_t)$$

$$\delta f = \delta c_t * c_{t-1}$$

$$\delta g = \delta c_t * i$$

$$\delta o = \delta h * \tanh^2(c_t)$$

$$\delta a_f = \delta f * f * (1 - f)$$

$$\delta a_i = \delta i * i * (1 - i)$$

$$\delta a_g = \delta g * (1 - \tanh^2(a_g))$$

$$\delta a_o = \delta o * o * (1 - o)$$

$$\delta a = \begin{bmatrix} \delta a_f \\ \delta a_i \\ \delta a_g \\ \delta a_o \end{bmatrix}$$

$$\delta b = \delta a$$

$$\delta W_h = \delta a \cdot h_{t-1}^\top$$

$$\delta h_{t-1} = W_h \cdot \delta a$$

$$\delta W_x = \delta a \cdot x_t^\top$$

References

Battaglia, P., Hamrick, J. B. C., Bapst, V., Sanchez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K., Nash, C., Langston, V. J., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018. URL https://arxiv.org/pdf/1806.01261.pdf.

Contributors:

- Anant Sahai.
- Kumar Krishna Agrawal.
- Matthew Lacayo.
- Suhong Moon.
- Shivam Singhal.
- Bryan Wu.
- Saagar Sanghavi.

