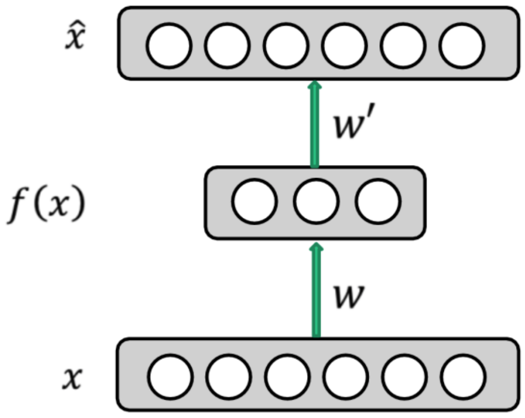
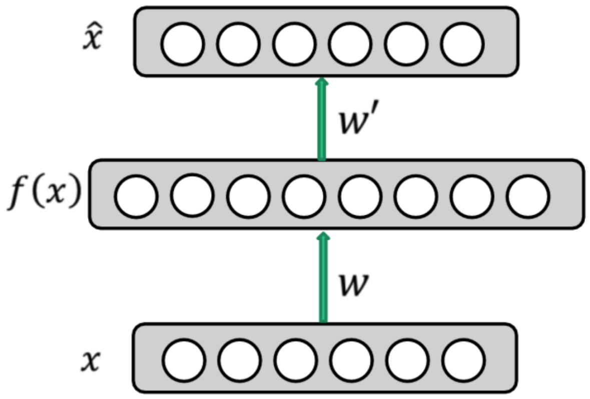
Take-Home Practice on Deep Learning (Non-Graded) -- Solutions

1) Graphically illustrate the difference between "Undercomplete" vs "Overcomplete" autoencoders:

Solution:

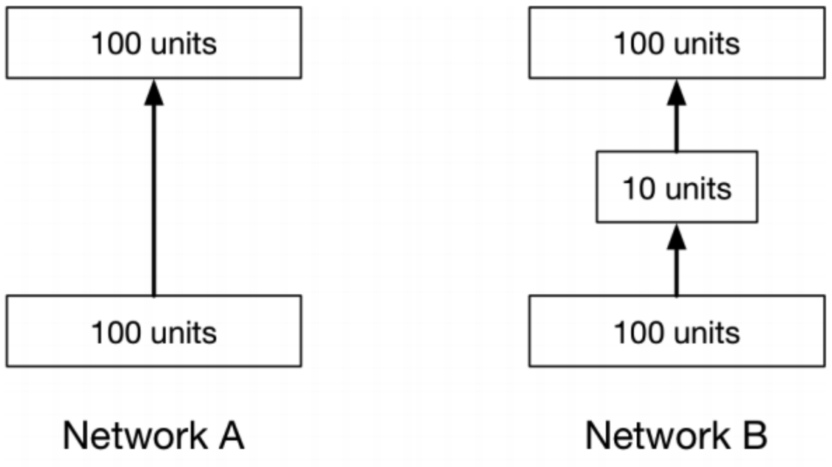


(Undercomplete AE)



(Overcomplete AE)

2) Consider the following two multilayer perceptrons where all of the layers use linear activation functions. Both networks receive input vectors of size 25 and the last layer shown is the output layer. (Assume no biases)



a) Which network do you think has more weights?

Solution:
For network A:
The first hidden layer has 100 neurons, and the output layer has 100 neurons. From the input to the first hidden layer, each of the 25 inputs is connected to each of the 100 neurons. This gives us 25 * 100 weights from input to first hidden layer. Similarly, each neuron of the first hidden layer is connected to all 100 neurons of the output layer, which gives a total of 100 * 100 weights from the hidden layer to the output layer. Thus the total number of weights in network A is:

Network A = 25 * 100 + 100*100 = 12500 weights

For network B:
There are 100 neurons in the first hidden layer, so from input to first hidden layer we get 25 * 100 weights. Then the second hidden layer has 10 neurons, which means from the first hidden layer to second hidden layer we get 100 * 10 weights. Finally the output layer has 100 neurons, thus we get 10 * 100 weights from second hidden layer to output layer.

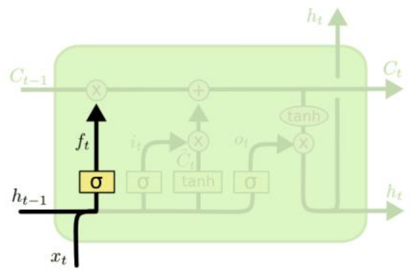
Total number of weights in Network B = 25 *100 + 100 * 10 + 10 *100 = 4500 weights

Thus we see that Network A has more weights than Network B.

b) Give one advantage of Network B over Network A?

Solution:
Possible answers:
- B has fewer connections, so it's less prone to overfitting
- B has fewer connections, so backprop requires fewer operations
- B has a bottleneck layer, so the network is forced to learn a compact representation (like an autoencoder)

3) Consider the weight vector [0.2, 0.1 0 0.1] that represents weights from the input ($X_t = 0.5$) of an LSTM Cell to input gate (i), memory cell (c), forget gate (f), and output gate (o). The weight from the previous hidden state ($h_{t-1} = 0.02$) to these same entities is [0.05, 0.02, 0 0.5], and the biases for each of these gates are [0, 0, 0, 0]. The input σ denotes the sigmoid function.
Useful σ values (rounded): $\sigma(0) = 0.5$, $\sigma(0.1) = 0.52$, $\sigma(0.06) = 0.51$
Useful tanh values (rounded): $\tanh(0.05) = 0.05$, $\tanh(0.076) = 0.076$



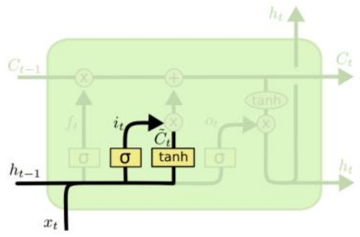
If the previous cell state C_{t-1} is 0.1, compute the following:

a) Forget gate: $f_t =$

Solution: $f_t = \sigma(W_{f1} * X_t + W_{f2} * h_{t-1} + b_f) = \sigma(0*0.5 + 0*0.02 + 0)$

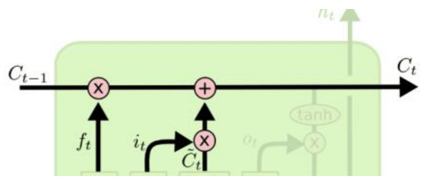
$= \sigma(0) = 0.5$

b) Input gate: i_t =



Solution: $i_t = \sigma(W_{i1} * X_t + W_{i2} * h_{t-1} + b_i)$
$= \sigma(0.2 * 0.5 + 0.05 * 0.02 + 0)$
$= \sigma(0.101) = 0.52$

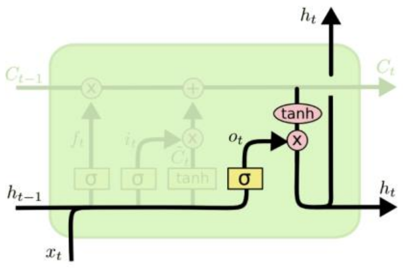
Cell State: $\tilde{C}_t = \tanh(W_{C1} * X_t + W_{C2} * h_{t-1} + b_C)$
$= \tanh(0.1 * 0.5 + 0.02 * 0.02 + 0)$
$= \tanh(0.0504) = 0.05$



c) Memory Update: C_t =

Solution: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t = 0.5 * 0.1 + 0.52 * 0.05 = 0.076$

d) Output Gate & Hidden State:



o_t =

Solution:

$o_t = \sigma(W_{O1} * X_t + W_{O2} * h_{t-1} + b_O)$
$= \sigma(0.1 * 0.5 + 0.02 * 0.5 + 0)$
$= \sigma(0.06) = 0.51$

h_t =

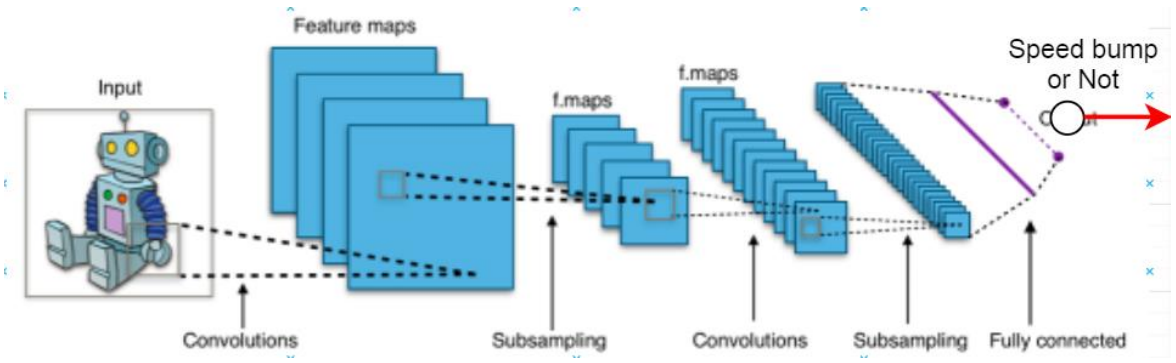
Solution:

$h_t = o_t * \tanh(C_t) = 0.51 * \tanh(0.076) = 0.51 * 0.076 = 0.038$

4) Neural network architecting.

a) You are given a camera stream from an autonomous car and are tasked with identifying speed bumps. You have 5 million training examples. Illustrate an appropriate network architecture for this task.

Solution:
A convolutional network connected to a fully connected output would leverage the spatial feature detection abilities of the CNN and the non-linear binary classification of the fully connected network. Because we have a lot of training data, we can learn the CNN features from scratch.

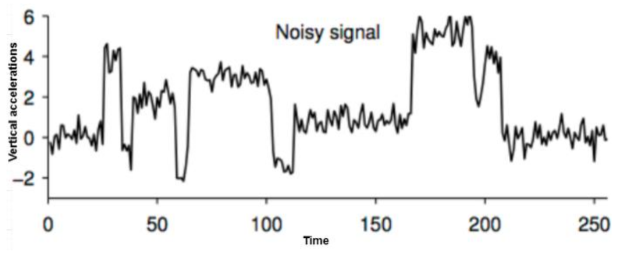


Base image from: https://towardsdatascience.com/introducing-convolutional-neural-networks-in-deep-learning-400f9c3ad5e9

b) How would you modify the solution from above if your new task is to detect stop signs, but you only have 1000 training examples.

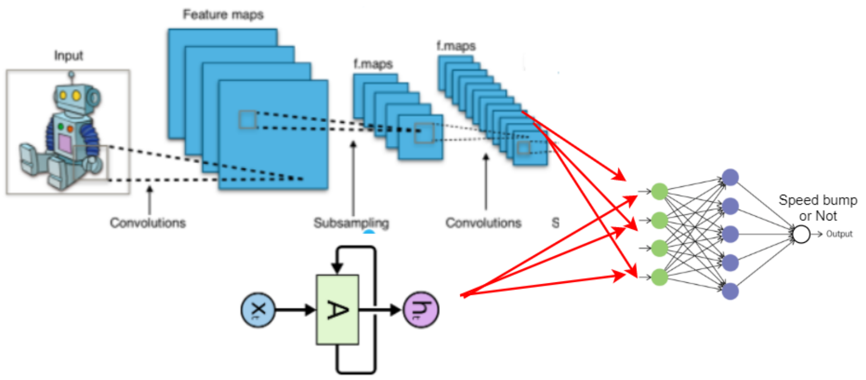
Solution:
With such limited training examples we would need to use a pre-trained CNN on the road bumps task to extract the features and then train the final fully-connected layers on the stop sign class (see slides on transfer learning).

c) Turning back to the speed bump example (a), in addition to the camera stream, you are now given a signal stream from an accelerometer. Graphically illustrate an appropriate network architecture that incorporates both the camera input and the accelerometer stream (an example of the accelerometer data is shown below, along with a helpful annotation) to identify whether the car has gone over a speed bump.



Example continuous data stream from car accelerometer.

Solution:
Use the CNN on the camera images, but on the accelerometer data a RNN like an LSTM would be useful to model the temporal dependencies. Combining the outputs of both networks into a fully connected layer(s) would allow the network to learn associations between the two.



5) CNN -

a) Given the following 4x4 input “image” and 2x2 filter, convolve the filter over the image using a stride of 1.

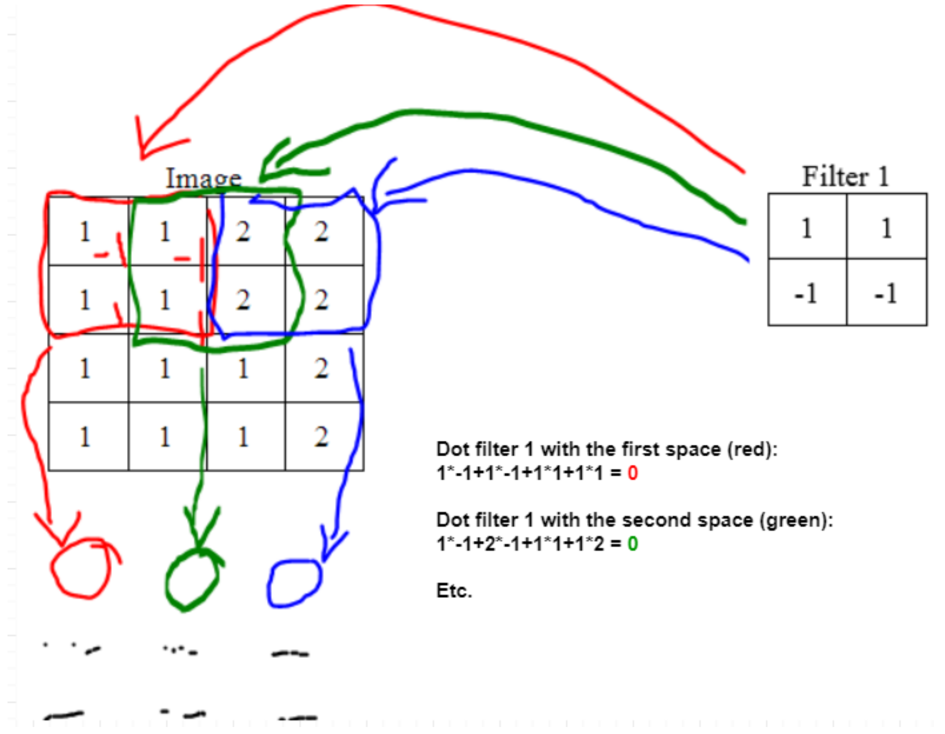
Image

1	1	2	2
1	1	2	2
1	1	1	2
1	1	1	2

Filter 1

1	1
-1	-1

Solution:



Dot filter 1 with the first space (red):
1*-1+1*-1+1*1+1*1 = 0

Dot filter 1 with the second space (green):
1*-1+2*-1+1*1+1*2 = 0

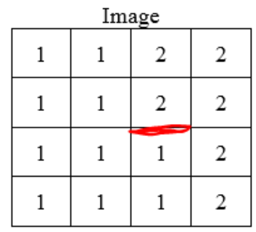
Etc.

Result of Filter 1 convolved over the Image

0	0	0
0	1	1
0	0	0

b) Based on the results of the convolution what type of features would you say Filter 1 detects?

Solution: Filter one detects horizontal edges. The horizontal edge (gradient or change in value intensities) found in the input image is highlighted in red below.



Image

1	1	2	2
1	1	2	2
1	1	1	2
1	1	1	2

c) Design a new 2x2 filter to detect vertical edges in the same image from (a), for convenience the image is repeated below as well as a grid for filling in your filter.

Image

1	1	2	2
1	1	2	2
1	1	1	2
1	1	1	2

Filter 2

Solution:
This probably takes a little bit of trial and error and understanding the pattern matching needed.

Image

1	1	2	2
1	1	2	2
1	1	1	2
1	1	1	2

Filter 2

-1	**1**
-1	**1**

d) Convolve your filter of the image and fill in the results in the grid below. (*Did your filter work as expected?)*

Result of Filter 2 convolved over the Image

Solution:

Result of Filter 2 convolved over the Image

0	2	0
0	1	1
0	0	2

Yes, the result of the filter shows 1's or 2's in the same geometric regions as the vertical edges in the Image. The 2's show up where the edge is stronger (both values are changing) and the 1's show up where only one value has changed.

e) Given the result of the previous image and a new filter below. Fill in the below result grids for, i) a max pooling layer with stride of 1 and size 2x2, ii) applying a ReLU activation.

Output Z

-1	2	0
-1	1	1
0	0	-2

Result of max pooling layer on Output Z.

Result of ReLU activation on Output Z.

Solution:

Result of max pooling layer on Output Z.

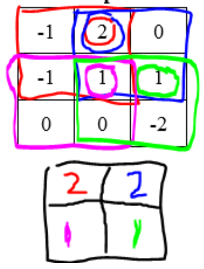
2	2
1	1

Result of ReLU activation on Output Z.

0	2	0
0	1	1
0	0	2

Max pooling is result is the process of sliding the 2x2 filter across the input and returning the maximum value in the filter size.

Output Z



-1	2	0
-1	1	1
0	0	-2

2	2
1	1

ReLU applies the following function to each value in the input: output = max(x, 0)