HW4 – machine learning in healthcare

Submitted by: Lidan Fridman 206201816

Tom Levin 307842419

Part 1 – fully connected layers

Models summary

All models are composed of two hidden layers with 300 and 150 neuros and 4 neuros of classification

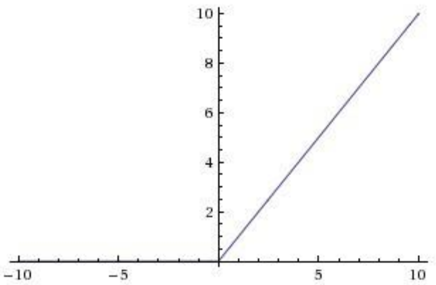
Initialization: He_normal

	Activation Function	batch_size	Batch Normalization	Epochs	Loss	Acc
Task 1	ReLU	64	x	25	0.8	67.43 %
Task 2	tanh	64	x	25	0.82	62.86 %
Task 3 – 25 epochs	Tanh	64	x	25	0.87	63.43 %
Task 3 – 40 epochs	tanh	64	x	40	0.87	64.00 %
Task 4	ReLU	32	x	50	0.83	66.29 %
Task 5	ReLU	32	v	50	0.91	65.14 %

Note: the results and the trends changed in each running iteration probably due to overfitting.

Task2 –

[1]Original activation function: ReLU



This non-linear function gives linearly increasing output for positive inputs, and zero output for negative inputs.

Pros:

- Sparsity – canceling some of the neurons when they're not beneficial to the predictive power of the model thus reducing the complexity of the model (unlike tanh).

[1] https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

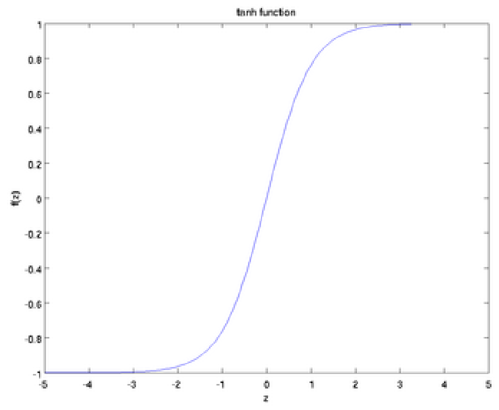
- Less computationally expensive than tanh thanks to simpler mathematical operations.
- Fewer vanishing gradients – gradient is always constant (0 or 1) leading to fast calculation of the loss function and faster learning.

Cons:

- Dying ReLU – when the network is poorly initialized or data poorly normalized, the first rounds of optimizations will produce great changes in the weights and this could eventually lead to dead neural network where many neurons output zero (when a neuron's weight yield zero outputs it's hard to recover from this).
- Not bounded – theoretically, infinite inputs lead to infinite outputs.
- Canceling small/ negative values – they may be of importance and still be ignored (0 weight).

New activation function – tanh

Hyperbolic function, output value differs significantly for inputs close to 0. For very high input values the output is very close to 1 and for very low input values the output is very close to -1.



Pros:

- It produces a zero-centered output (good for backpropagation)
- Bounded [-1, 1]

Cons:

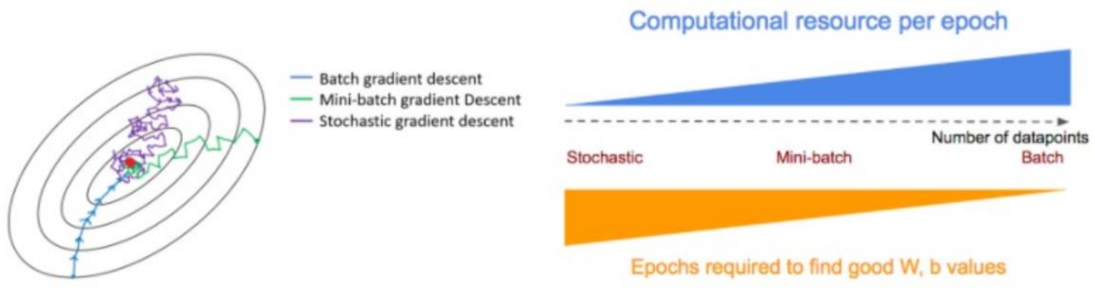
- Not sparse – the neurons would almost always produce an output value, unimportant neurons can't be removed line in ReLU.
- Vanishing gradients – leading to slow convergence and slow learning rate.
- Computationally expensive – complex mathematical operations.

Task3 – The Loss and ACC of the model (with 25 and 40 epochs) were varied between different runs. In general- the performances of the model under both epoch numbers were pretty much the same. When increasing the number of epochs in our model we expect to improve the weights and the model's performances. This did not happen in our case, the difference here between 25 and 40 epochs is redundant. Perhaps because of overfitting or poor-quality data.

Task4 –

In SGD we compute the loss function relying on one training example and therefore the path to the minima is noisier. The optimization reach convergence fast and is less computationally expensive than batch-GD.

Mini-batch GD is a compromise between the two methods and in mini-batch we compute the loss function relying on mini-batch set of training examples. The convergence is smoother (less noisy) than SGD.


Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent
Computational resource per epoch
Number of datapoints
Stochastic
Mini-batch
Batch
Epochs required to find good W, b values

The advantages of mini-batch include computational efficiency, stable convergence toward global minimum (computation over many samples > less noisy) and faster learning because we update the weights more often. In SGD the gradient is more noisy since we use single gradient (and not averaged as in mini-batch) for weights update. Due to the noise it is also difficult to stay in one local minima of the loss function. SGD is also computationally costly.

<u>Task5 –</u> In batch normalization we standardize the inputs to a layer for each mini-batch. Batch normalization shrinks the effect of covariance shift and enables learning more stable input distributions. Covariance shift happens when distribution of the input values changes due to previous layer weighs update. Batch normalization accelerates training and could reduce generalization error.

Part 2 – convolutional neural network (CNN)

Task 1 - 2D CNN

- How many layers does it have?

  5 of convolutional layers and 3 dense (fully connected) layers. Total 8 layers

- How many filter in each layer?

Layer	# filters
1 (2D conv)	64
2 (2D conv)	128
3 (2D conv)	128
4 (2D conv)	256
5 (2D conv)	256

- Would the number of parameters be similar to a fully connected NN?

  The number of parameters here Wouldn't be similar to the number of parameters in a fully connected NN because using CNN enables to reduce the number of parameters drastically, especially in our case- image analysis - when each input is represented by a pixel. For demonstration, the number of learned parameters in a dense layer is calculated by:

$$learnable\ parameters = Weights + biases \quad when \quad Weights = inputs\ X\ outputs\ .$$

We multiply the number of inputs to the layer with the number of outputs from the layer When basically outputs are the number of nodes within the dense layer.
In convolutional layer we use filters (kernels). We will calculate its learnable parameters with the same formula as we used above only that now:

$$(1)\ Inputs = \begin{cases} Number\ of\ \ nodes, & if\ the\ last\ layer\ is\ dense \\ Number\ of\ \ filters, & if\ the\ last\ layer\ is\ convolutional \end{cases}$$

$$(2)\ Outputs = (Number\ of\ \ filters)\ X\ (Size\ of\ \ filters)$$

$$(3)\ biases = Number\ of\ \ filters$$

Furthermore, in CNN we scan those filter on the inputs to perform convolution operation. Because of that, this method consumes less parameters than a fully connected layer (when each neuron is connected to all of the neurons in the next layer).

- Is this specific NN performing regularization?
    1. L2 regularization – reduces overfitting by canceling weights.
    2. Dropout – another method to reduce overfitting by ignoring a fraction (30% in our case) of neurons in specific layers in the training stage.

Task 2 – number of filters

	Filters vector	Loss	Acc
Task 1	`[64, 128, 128, 256, 256]`	7.9743	0.2914
Task 2	`[32, 64, 64, 128, 128]`	4.6175	0.2571

We can think of filters as detectors for features. In general, when we have X feature detectors we will get X features that our NN will learn (body parts, lines, corners etc..). Hence, (In general) we can look at this filter reduction as a reduction of the number of features that the NN need to learn. When reducing the number of filters, we are reducing the complexity of the model. In our case the Loss was consistently lowered by half (in each run) although the accuracy was varied between different runs (sometimes increased and sometimes decreased). We can say that in Task 1 the model was overfitted and therefore the Loss was lowered in Task 2. Although its critical to note that we try here to formalize a black box when each model has a lot of different considerations and the variations of the acc may point that we have insufficient quality in our data that blurs the effect of the filters reduction.