Theory Questions

Yoav Shoshan.

(1) $l(\{w_i\}_{i=1}^{L}, x, y) = \max_{\hat{y}\in[L]} (w_{\hat{y}} \cdot x - w_y \cdot x + \Delta_{zo}(\hat{y}, y))$

(a) Define $l_{\hat{y}} = w_{\hat{y}} x - w_y x + \Delta_{zo}(\hat{y}, y)$

$l_{\hat{y}}$ is linear in $w_{\hat{y}}$, w_y and a linear function in x is convex wrt x *

$$\left(\begin{aligned} * \quad & f(\alpha x_1 + (1-\alpha)x_2) = a(\alpha x_1 + (1-\alpha)x_2) + b \\ & = \alpha(a x_1 + b) + (1-\alpha)(a x_2 + b) \\ & = \alpha f(x_1) + (1-\alpha) f(x_2) \end{aligned} \right)$$

Also, we proved tha if $g = \max\{f_i\}_{i=1}^{L}$ and f_i is convex $\forall i \Rightarrow$ g is convex. So if we define:

$$l = \max_{\hat{y}\in[L]} ((w_{\hat{y}} - w_y) \cdot x + \Delta_{zo}(\hat{y}, y))$$

$$= \max_{\hat{y}\in[L]} \{l_{\hat{y}}\}_{\hat{y}=1}^{L}$$

$\Rightarrow$ l is convex

(b) $f(x, \{w_i\}_{i=1}^{L}) = \arg\max_{y=1}^{L} w_y \cdot x = t$

$$l(\{w\}, x, y) = \max_{\hat{y}\in[L]} (w_{\hat{y}} \cdot x - w_y x + \Delta_{ZO}(\hat{y}, y))$$

$$\geq w_t \cdot x - w_y x + \Delta_{ZO}(t, y) \overset{*}{\geq} \Delta_{ZO}(t, y)$$

$w_t x \geq w_y x \quad \forall x$ because of the definition of t.

(c) $$f_i^{opt} = \arg\max_{y=1}^{L} w_y^{opt} \cdot x = f(x_i, \{w_i^{opt}\})$$

$$f_i^{*} = \arg\max_{y=1}^{L} w_y^{*} \cdot x_i = f(x_i, \{w_y^{*}\})$$

we have

$$0 \underset{\Delta_{ZO} \geq 0}{\leq} \sum_i \Delta_{ZO}(f_i^{opt}, y_i) \underset{\text{section } 2}{\leq} \sum_i l(\{w_i^{opt}\}, x_i, y_i)$$

$$\overset{*}{\leq} \sum_i l(\{w_i\}, x_i, y_i) \quad \left(\begin{array}{l}\text{because } \{w_i^{opt}\} \quad * \\ \text{minimize the sum}\end{array}\right)$$

$$= \sum_i \max_{\hat{y}} \left[(w_{\hat{y}} - w_{y_i}) \cdot x_i + \Delta_{ZO}(\hat{y}, y_i)\right]$$

$\Longrightarrow$ if we show that the exist $\{w_i\}$ s.t.:

$$\max_{\hat{y}} \left[(w_{\hat{y}} - w_{y_i}) \cdot x_i + \Delta_{ZO}(\hat{y}, y_i)\right] = 0$$

than we are done

$\Longrightarrow$

$\Longrightarrow$ * first notice that for $\hat{y} = y_i$

$$(w_{\hat{y}} - w_{y_i}) \cdot x_i + \Delta_{zo}(\hat{y}, y_i) = 0$$

** For $y_i \neq \hat{y}$:

we know from $y_i = \arg\max_{y=1}^{L} w_y \cdot x_i$

that $(w_{\hat{y}} - w_{y_i}) \cdot x_i < 0 \quad \forall i$

and this is specifically true for $\{w_y^*\}$

$$(w^*_{\hat{y}} - w^*_{y_i}) \cdot x_i < 0$$

we can defin $m = \min_{\hat{y} \neq y_i} (w_{\hat{y}} - w_{y_i}) \cdot x_i$

$\Longrightarrow m < 0$

So we can choose $w_{\hat{y}} = -\frac{w^*_{\hat{y}}}{m}$ $\quad w_{y_i} = -\frac{w^*_{y_i}}{m}$

$$\Longrightarrow (w_{\hat{y}} - w_{y_i}) \cdot x_i = \underbrace{-\frac{1}{\underbrace{m}_{<0}} \underbrace{(w^*_{\hat{y}} - w^*_{y_i}) \cdot x_i}_{<0}}_{>0}$$

$$< -\frac{1}{m} \cdot m = -1$$

$\Longrightarrow$ for $\hat{y} \neq y_i$

$$(w_{\hat{y}} - w_{y_i}) \cdot x_i + \Delta(\hat{y}, y_i) < -1 + 1 < 0$$

$\overset{*+**}{\Longrightarrow} \forall_i \quad \ell(\{w_y\}, x_i, y_i) = 0$

$\Longrightarrow 0 \leq \Delta_{zo}(t_i^{opt}, y_i) > 0$

$\Longrightarrow \Delta_{zo}(t_i^{opt}, y_i) = 0$ //

Scanned by CamScanner

(2) we have a sample $S=\{x_1 \dots x_m\}$ and two function families. $F^1 \subseteq Y_1^{\mathcal{X}}$, $F^2 \subseteq Y_2^{Y_1}$

define $F = F_2 \circ F_1$. then:

$F_S = \{\langle f_2(f_1(x_1)) \dots, f_2(f_1(x_m)) \rangle : f_1 \in F^1, f_2 \in F^2\}$

first we notice that operating with some $f_1 \in F_1$ on S, will produce another sample of size m

$S_{f_1} = \{f_1(x_1), f_1(x_2) \dots f_1(x_m)\} = \{y_1, \dots y_m\}$

and we can generate $|F_S^1|$ samples like this

now for some fixed f_1, we operate with some $f_2 \in F_2$ on S_{f_1} and get $F^2_{S_{f_1}} = \{f_2(y_1) \dots, f_2(y_m) : f_2 \in F^2\}$

and we have $|F^2_{S_{f_1}}|$ possibilities for result sets.

of course we have to consider the possibility that $f_i, f_j \in F^1$ $i \neq j$ will produse the same S_{f_1} so

$$|F_S| \leq |F_S^1| \cdot |F^2_{S_{f \in F^1}}| \leq |F_S^1| \cdot \max_{S_{f \in F_1}} |F^2_{S_f}|$$

we reduce by taking S_{f_1} that will produce the largest $|F^2_{S_{f_1}}|$

$\Longrightarrow$

$\Longrightarrow$ Now by definition of the Growth functions

$$|F_S^i| \leq \Pi_{F^i}(m)$$

$$\Rightarrow F_S \leq \Pi_{F^1}(m) \cdot \Pi_{F^2}(m)$$

And this is of course trow for S for which $|F_S| = \Pi_F(m)$ as well

$\Rightarrow$

$$\Pi_F(m) \leq \Pi_{F^1}(m) \cdot \Pi_{F^2}(m)$$

(3) The SGD Algorithm for regularized Hinge loss:

$$w_{T+1} = \begin{cases} (1-\eta) w_T & 1 - y_{T+1} w_T \cdot x_{T+1} < 0 \\ (1-\eta) w_T + c\eta y_T x_T & 1 - y_{T+1} w_T x_{T+1} > 0 \end{cases}$$

So we can set

$$w_{T+1} = \sum_{i=1}^{T+1} \beta_i^{T+1} x_i y_i \qquad w_0 = 0$$

i denote the i-th vector in the SGD process

where $\beta_i^T = \begin{cases} 0 & 1 - y_i w_{i-1} x_i < 0 \\ c\eta (1-\eta)^{T-i} & 1 - y_i w_{i-1} x_i > 0 \end{cases}$

(T is the Total number of steps)

proof with induction:

w_0 is the initial and therefore:

$$w_1 = (1-\eta) w_0 = 0 = \sum_{i=1}^{1} \beta_i x_i y_i \underset{*}{=} c\eta x_1 y_1$$

$$* \quad 1 - y_1 w_0 x_1 \underset{w_0 = 0}{=} 1 > 0 \Rightarrow \beta_1^1 = c\eta (1-\eta)^{1-1} = c\eta$$

first we show that the definition is good for w_2 (This is our base of induction)

if $1 - y_2 w_1 x_2 < 0 \Rightarrow \beta_2^2 = 0 \Rightarrow$

$$w_2 = \beta_1^2 x_1 y_1 = c\eta (1-\eta)^{2-1} x_1 y_1 = (1-\eta) c\eta x_1 y_1 = (1-\eta) w_1$$

if $1 - y_2 w_1 x_2 > 0 \Rightarrow \beta_2^2 = c\eta (1-\eta)^{2-2} = c\eta$

$$w_2 = \beta_1 x_1 y_1 + \beta_2 x_2 y_2 \underset{(\beta_1^2 \text{ is the same})}{=} (1-\eta) c\eta x_1 y_1 + c\eta x_2 y_2 = (1-\eta) w_1 + c\eta x_2 y_2$$

Scanned by CamScanner

$\Rightarrow$ Base of induction is consistent with the SGD for hinge loss Algorithm

Now Assume this definition of α_i^T is correct for a general number of steps t.

for the $T+1$ step all β_i^{T+1} will be updated : $\beta_i^{T+1} = (1-\eta)\beta_i^T$
(This is from the definition of α_i^T)

now if $1 - y_{T+1} w_T x_{T+1} < 0$

$$\Rightarrow \beta_{T+1}^{T+1} = 0$$

$$\Rightarrow w_{T+1} = \sum_{i=1}^{T+1} \beta_i^{T+1} x_i y_i = \underbrace{\sum_{i=1}^{T} \beta_i^{T+1} x_i y_i}_{\alpha_{T+1}^{T+1}=0} =$$

$$= \sum_{i=1}^{T} (1-\eta)\beta_i^T x_i y_i = (1-\eta) w_T$$

and if $1 - y_{T+1} w_T x_{T+1} > 0 \Rightarrow \beta_{T+1}^{T+1} = c\eta(1-\eta)^{T+1-T-1} = c\eta$

$$\Rightarrow w_{T+1} = \sum_{i=1}^{T+1} \beta_i^{T+1} x_i y_i = \sum_{i=1}^{T} (1-\eta)\beta_i^T x_i y_i + \beta_{T+1}^{T+1} x_{T+1} y_{T+1}$$

$$= (1-\eta) w_T + c\eta x_{T+1} y_{T+1}$$

$$\Rightarrow w_0 = 0, \quad w_{T+1} = \sum_{i=1}^{t+1} \beta_i^{T+1} x_i y_i$$

and

$$\beta_i^T = \begin{cases} 0 & 1 - y_i w_{i-1} x_i < 0 \\ c\eta(1-\eta)^{T-i} & 1 - y_i w_{i-1} x_i > 0 \end{cases}$$

are a valid definition of w wrt the SGD Algorithm for hinge loss

$\Rightarrow$ Now we can switch our problem to a non linear one with feature vectors $\phi(x)$:

$$w_{t+1} = \begin{cases} (1-\eta) w_t & 1 - y_{t+1} w_t \phi(x_{t+1}) < 0 \\ (1-\eta) w_t + c\eta y_t \phi(x_i) & 1 - y_{t+1} w_t \phi(x_{t+1}) > 0 \end{cases}$$

and $w_t = \sum_{i=1}^{t} \alpha_i^t y_i \phi(x_i)$

w_{i-1} is calculated similarly.

$$\beta_i^t = \begin{cases} 0 & 1 - y_i w_{i-1} \phi(x_i) < 0 \\ \eta C (1-\eta)^{t-i} & \text{otherwise} \end{cases}$$

So now we only have to calculate

$$\text{sign}(1 - y_i w_{i-1} \phi(x_i)) = \text{sign}\left(1 - y_i \left(\sum_{j=1}^{i-1} \beta_j y_j \phi(x_j)\right) \phi(x_i)\right)$$

$$= \text{sign}\left(1 - y_i y_j \sum_{j=1}^{i-1} \beta_j \phi(x_j) \phi(x_i)\right) =$$

$$\text{sign}\left(1 - y_i y_j \sum_{j=1}^{i-1} \beta_j K(x_j, x_i)\right)$$

and for n steps we will have to make n calculations like that and store n values of $\{\alpha_i^n\}_{i=1}^n$

moreover we can write w_T for each step T. and then for some set of indeces $I_j = \{j_1 \dots j_k : j_p \in [1, \dots T] \ \forall p\}$ $(j \in [1..n])$ for which $x_{j_1} = x_{j_2} = \dots x_{j_k}$ $j_p \in I \ \forall p$

Scanned by CamScanner

and assuming all of the x_{ij} equal x_l (the l-th vector from the whole data)

we can write $\alpha_l = \sum_{j=1} \beta_{jl}$ *

then we can write w as a linear combination of all samples (not all t vectors used for SGD)

$w_t = \sum_{l=1}^{n} \beta_l y_l x_l$ and there are exactly n coeficeints β_l and therefore exactly n Kernel calculations for the next step.

$\Longrightarrow$ complexity $= O(n)$ for each step.

* Note that we dont really have to calculate the sum for each α_l at each step. Say we have at step T $\phi_T = \phi_m$, $m \in \{1 \dots n\}$

Then the updates are:

$$\boxed{\begin{aligned} \alpha_l^T &= (1-\eta)\alpha_l^{T-1} & l \neq m \\ \alpha_m^T &= (1-\eta)\alpha_m^{T-1} & 1 - y_m w_{T-1}\phi_m < 0 \\ & (1-\eta)\alpha_m^{T-1} + C\eta y_m & 1 - y_m w_{T-1}\phi_m > 0 \end{aligned}}$$

(Those are calculated using Kernels)

(4) $x_{t+1} = x_t - \eta \nabla l(x_t) \qquad \eta > 0$

$l(x) \geq 0 \quad \forall o$

$l(y) \leq l(x) + (\nabla l(x))^T (y-x) + \frac{\beta}{2}|x-y|^2$

$\Longrightarrow$ for each step. t:

- $l(x_{t+1}) \leq l(x_t) + (\nabla l(x_t))^T (x_{t+1} - x_t) + \frac{\beta}{2}|x_{t+1} - x_t|^2$

we notice that from the definition of the Algorithm steps:

$x_{t-1} - x_t = -\eta \nabla l(x_t)$

$l(x_{t+1}) \leq l(x_1) - \eta|\nabla l(x_t)|^2 + \frac{\beta}{2}\eta^2 |\nabla l(x_t)|^2$

- $= \underbrace{\eta\left(\frac{\beta}{2}\eta - 1\right)}_{<0} \cdot |\nabla l(x_t)|^2 + l(x_t)$

Becase $\eta < \frac{2}{\beta}$

$|\nabla l(x_t)|^2 \leq \frac{l(x_{t+1}) - l(x_t)}{\eta\left(\frac{\beta}{2}\eta - 1\right)} = \frac{l(x_t) - l(x_{t+1})}{\eta\left(1 - \eta\frac{\beta}{2}\right)}$

$$\sum_{i=1}^{t} |\nabla l(x_i)|^2 = \frac{1}{\eta(1-\eta\frac{\beta}{2})}\left(l(x_0) - l(x_t)\right)$$

telescop series

$$\leq \frac{l(x_0)}{\eta(1-\eta\frac{\beta}{2})} \equiv C$$

$$\left(\begin{matrix} l(x) > 0 \\ \forall x \end{matrix}\right) \& \left(\frac{1}{\eta(1-\eta\frac{\beta}{2})} > 0\right)$$

- So

$$\lim_{t\to\infty} \sum_{i=1}^{t} |\nabla l(x_i)|^2 < \infty$$

$$\Longrightarrow \lim_{t\to\infty} |\nabla l(x_t)|^2 = 0$$

-

Programming assignment

Question 1 - SVM

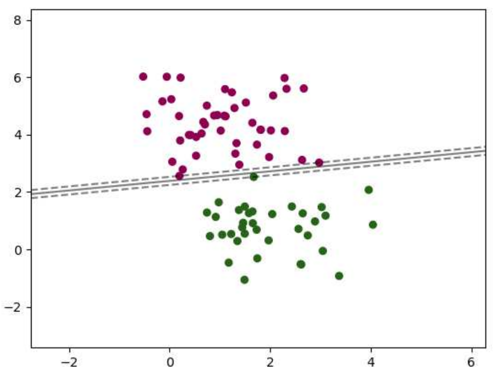
Stdout:

```
C:\Users\shosh\Google_Drive\Studies\Intro_to_ML\ex\ex4\SVM>python svm.py
Section a
number of support fectors for linear kernel = 3
number of support fectors for quadratic kernel = 4
number of support fectors for rbf kernel = 17

Section b
Best C is 100.0, with Accuracy = 1.0
(if many c values yield the maximal accuracy, this is the smallest among them)

Section c
Best Gamma on validation set is 0.3162 with Accuracy = 1.0
```

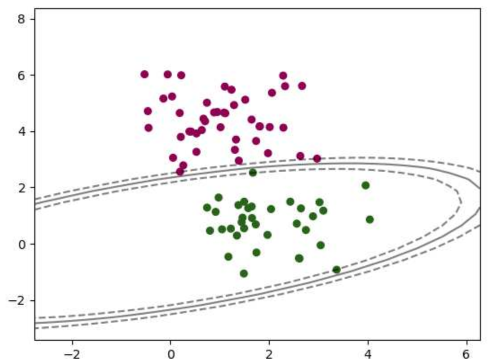
a. In the **Linear Kernel**, we have a separating straight line separating the data as expected. Note that here we have 2 support vectors from one of the classes and 1 from the other. It is of course reasonable as we would expect at least one SV from each class. Otherwise we could have increase the margin from the other classes that have SV by taking a line that is as close as possible to the points from the classes that do not have SV.

   The **Quadratic Kernel** is a 2D ellipsoid. Quadratic both in x and y values. The decision rule should consider $x_1, x_2, x_1x_2, x_1^2, x_2^2$, and we expect to have 2 SV for each class. As we got.

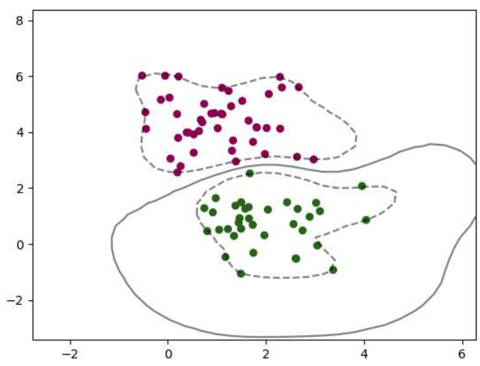
   The **RBF Kernel** is a gaussian around each of the point. Each Gaussian effects the probability of a new dot to be classified as the point that generates this gaussian. The effect of each point on other new points in the space is controlled by the width of each gaussian, and thus, we could have SV as the number of training points (in theory). Here we have 17 SVM. Gaussians that their center is already within a region that is classified be the gaussian of other points do not effect the decision rule in this case.



Linear Kernel

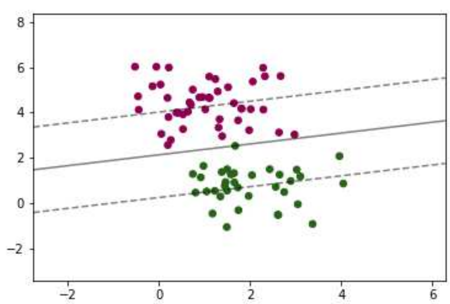


Quadratic Kernel

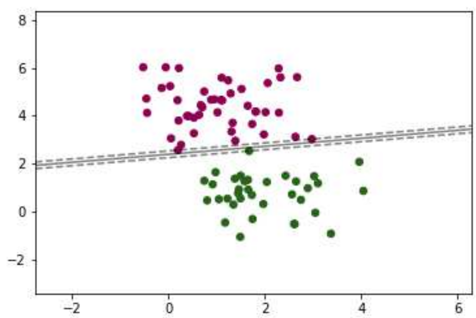


RBF Kernel

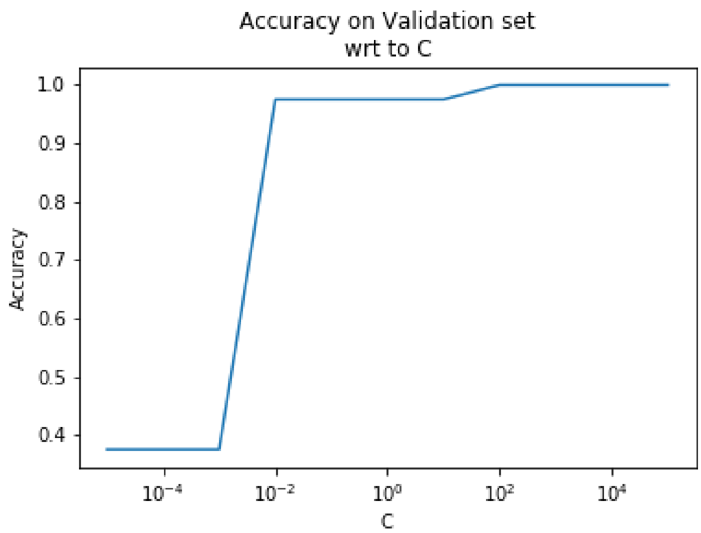
b. Best C we found is 100. This value of penalty is sufficient for not allowing any training error (this is possible in our case because the data is linearly separable). Higher values of C will not generate a better decision rule.
   For small values of C we have increased margin but the trade of is smaller accuracy on test data (see figures bellow)



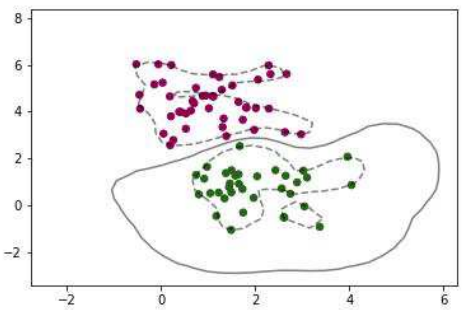
C = 0.01



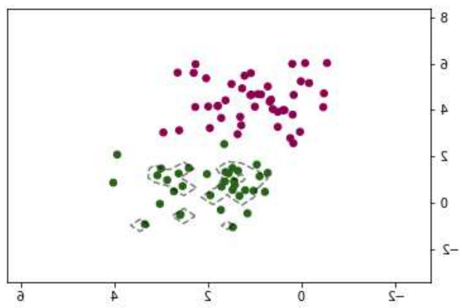
C = 100



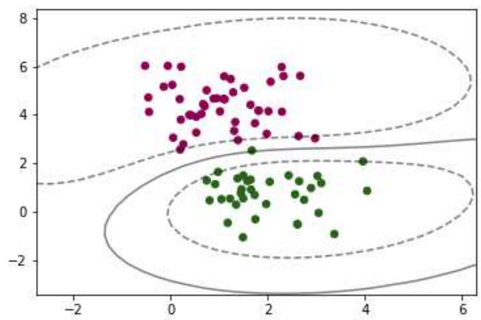
c. Best Gamma we found is 0.316 with 100% accuracy. As you can see from the accuracy plot, we have other values of gamma that achieve such an accuracy. The width of each gaussian centered at each of the training points is decreased with increasing values of gamma, and therefor each point strongly effects new adjacent points, but not other points which are slightly closer to other points. Thus, for increasing gamma, points that are not close to the center of their class cluster could mistakenly be classified with other close clusters of different classes. For small values of gamma, the gaussians are wider and the effect of each point on further points is stronger, this is why for extremely small values of gamma we are mistakenly classify points of other clusters than the points that effect their classification.



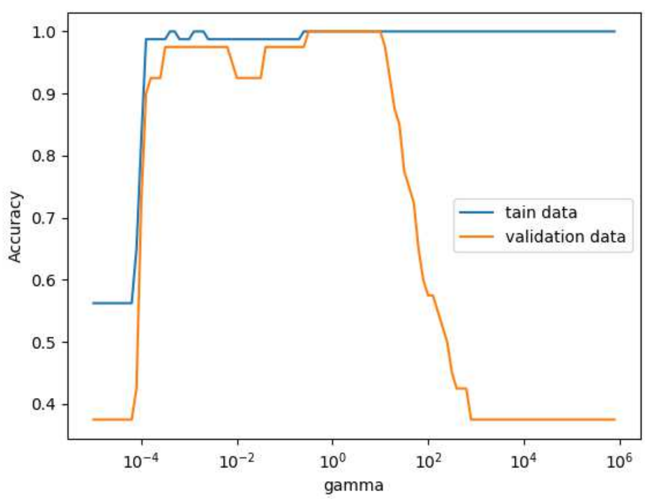
Gamma = 1



Gamma = 100

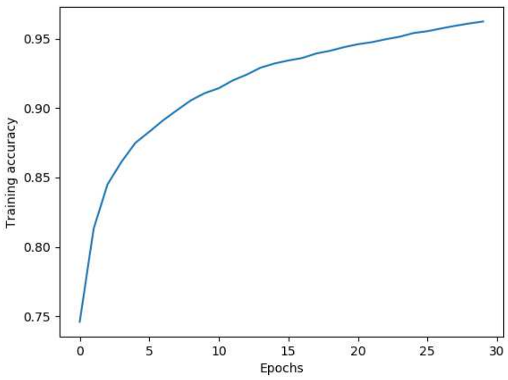


Gamma = 0.1

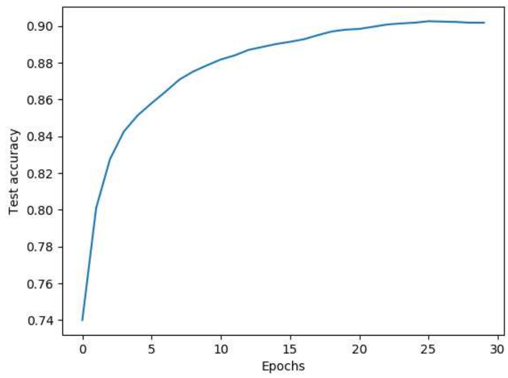


Question 2 – Back Propagation

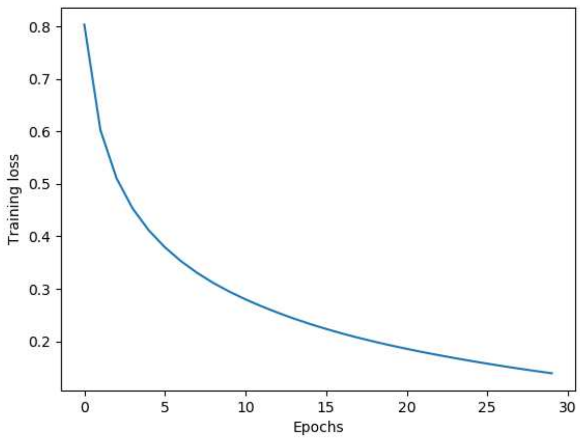
a. See code and results printed to Stdout while running.
b. From all figures we can see that the system learns the training data and is improving on the test data as well. Although Test accuracy seems to reach saturation, it seems that results in the training set could be improved with more epochs. The train accuracy is also slightly higher than test accuracy. This facts suggest overfitting.



Training accuracy



Test accuracy

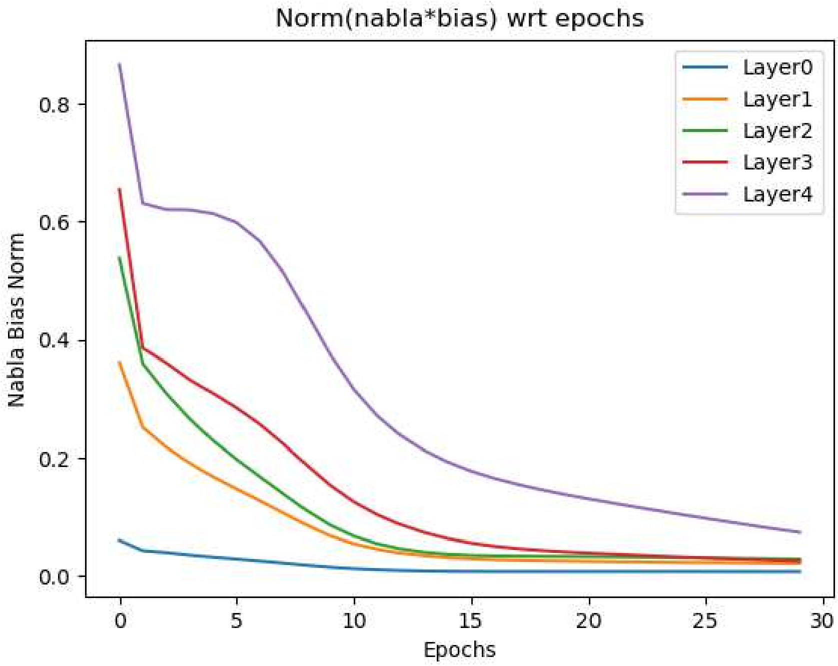


Training Loss

c. Stdout for section c:

```
Initial test accuracy: 0.0963
Epoch 0 test accuracy: 0.8614
Epoch 1 test accuracy: 0.8901
Epoch 2 test accuracy: 0.9047
Epoch 3 test accuracy: 0.9108
Epoch 4 test accuracy: 0.9165
Epoch 5 test accuracy: 0.9233
Epoch 6 test accuracy: 0.9262
Epoch 7 test accuracy: 0.9288
Epoch 8 test accuracy: 0.931
Epoch 9 test accuracy: 0.934
Epoch 10 test accuracy: 0.9364
Epoch 11 test accuracy: 0.9372
Epoch 12 test accuracy: 0.9377
Epoch 13 test accuracy: 0.9396
Epoch 14 test accuracy: 0.9396
Epoch 15 test accuracy: 0.9395
Epoch 16 test accuracy: 0.9403
Epoch 17 test accuracy: 0.9413
Epoch 18 test accuracy: 0.9418
Epoch 19 test accuracy: 0.9422
Epoch 20 test accuracy: 0.9431
Epoch 21 test accuracy: 0.943
Epoch 22 test accuracy: 0.9444
Epoch 23 test accuracy: 0.9457
Epoch 24 test accuracy: 0.9465
Epoch 25 test accuracy: 0.9462
Epoch 26 test accuracy: 0.9461
Epoch 27 test accuracy: 0.9463
Epoch 28 test accuracy: 0.9466
Epoch 29 test accuracy: 0.9472

After 30 epochs, we reached an accuracy of 0.9472 on the test data
```

d. Bias Gradient for each layer, WRT epochs:



We can see that the last layers are learned faster (the nor of the gradient reaches 0 faster) This phenomena could be explained by the properties if the activation function (Sigmoid, in our case) and the algorithm for BP itself.

Due to the fact that each layer's gradient wrt to the input is calculated backwards by the chain rule, and thus multiplied by the gradients of previous layers, and due the fact that we use sigmoid as an activation functions for which the derivative is:

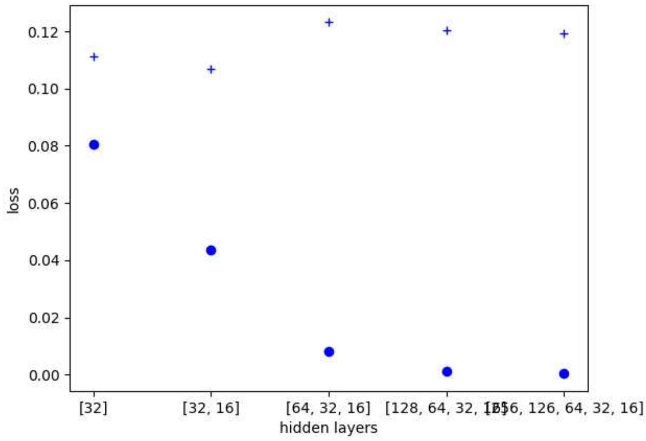
$$\sigma' = \sigma(1-\sigma) \leq \frac{1}{4} \quad (0 \leq \sigma \leq 1)$$

The last layers derivatives are multiplications of more values, all smaller than 0.25. and therefore gradients of layers that are further from the input will be closer to 0 (relative to layers that are closer to the input) with each epoch.

Question 3 – MLP

a. See code and its stdout while running.
b. I run the cmd:
python mlp_main.py -m series -hl 32 -1 32 16 -1 64 32 16 -1 128 64 32 16 -1 256 126 64 32 16

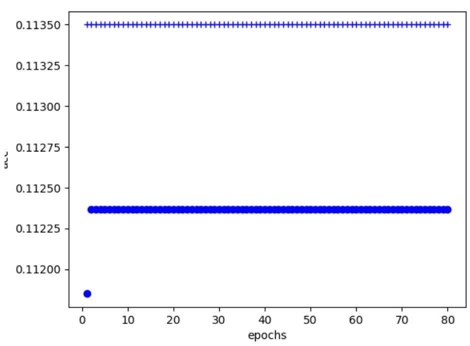
The resulted losses for each model:



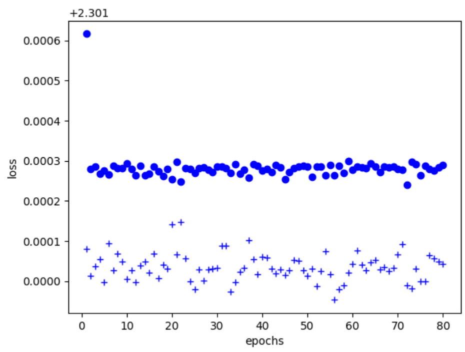
Ass expected, the loos for the training set decreases with the complexity of the model, and for the most complex models, it seems to reach the optimal values (more complex model might not help decrease the loss further)
The validations loss though, is higher for too complex models, which suggest an overfitting for the training data in those models.
The smallest loss is achieved for a not too complex model of just two layers with small dimensions (of course that there might be even better models but I haven't tried them all)

c. I run the command:
python mlp_main.py -m single -e 80 -hl 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16

The resulted losses and accuracies wrt to epochs:



Accuracies



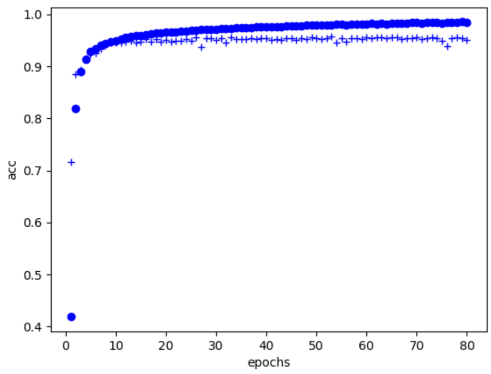
Losses

We had an ultra complex model with 60 layers and clearly it failed with the classification task, on the training data as well as on the test data. This could be explained by the fact that that following a large gradient through the Relu function may cause some neurons to reach such a value that they will never fire again (Relu returns 0 for negative numbers) and then the network is destroyed while the number of dead neurons increases during the learning process. This is more likely to occur when the network is more complex, containing more Relu activations between more layers.

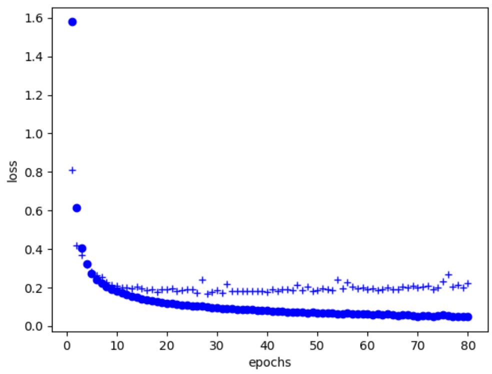
d. See code.

e. I run the command:
python mlp_main.py -m single -e 80 -s 5 -hl 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16

The resulted losses and accuracies wrt to epochs:



Accuracies



Losses

As can be seen in the figures, when allowing skip connections, the model can learn the data and reach desirable results. This is due to the fact that now, there are shorter paths leading from the last layers to the input, thus, the dying neurons phenomena is weaker. If some neurons are dead, we now have a path for bypassing them.

The model "choses" the best path for each activation and allowing complex networks as well as simple ones.